COB-2023 0664
ELLIPTIC GRID GENERATION USING PARALLEL COMPUTING

Juan Carlos Assis Silva
Programa de Pós-Graduação em Engenharia Mecânica, Universidade Federal Fluminense, Rua Passo da Pátria 156, Bloco E, Sala 211, Niterói, RJ 24210-240, Brasil
jcasilva@id.uff.br

Rômulo Bessi Freitas
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, Estrada de Adrianópolis, 1.317, Santa Rita, Nova Iguaçu, RJ 26041-271, Brasil
romulo.freitas@cefet-rj.br

Leonardo Santos de Brito Alves
Programa de Pós-Graduação em Engenharia Mecânica, Universidade Federal Fluminense, Rua Passo da Pátria 156, Bloco E, Sala 211, Niterói, RJ 24210-240, Brasil
leonardo.alves@mec.uff.br

Abstract. *Through the last few years, numerical computations of fluid flow governing equations have been used to understand a wide class of engineering problems. In many such cases, the continuous governing equations are solved at discrete points in space and time, where partial differential equations are transformed into algebraic equations. Independently of which technique is employed to discretize the equations (e.g. finite volume, finite difference, finite element, and so on), it is necessary to know a priori the discrete points that represent the geometry of interest, called grid or mesh. Since the numerical solution quality is strongly related to the grid quality, this work focuses on generating two-dimensional grids based on nonlinear Poisson-type equations and algebraic transformations. Furthermore, boundary layer problems often require grid refinement, as well as the orthogonality of the grid elements near the wall to facilitate boundary condition implementation. Both requirements were also considered. As direct numerical simulation (DNS) usually requires a large number of grid points, serial computations of grids may, unfortunately, be too time-consuming. In order to reduce this cost, this work applies parallel computation techniques using a message-passing interface (MPI) paradigm through the PETSc library structure. In this way, the domain is split into different CPUs that compute each part of the solution, reducing the computational time required. The nonlinear Poisson equation was solved using a finite difference technique coupled with parallel Jacobi iterations. The results show the code's scalability, speed-up and efficiency.*

Keywords: *Grid Generation, Parallel Computing, Computational Fluid Dynamics*

1. INTRODUCTION

The numerical computation of fluid flow is nowadays the main approach to understanding a wide class of scientific problems, it occurs because there is not a closed form solution of the Navier-Stokes equations, that models the behavior of Newtonian fluids (Pletcher *et al.*, 2012). In many such cases, the continuous governing equations are transformed into algebraic equations and solved just at discrete points in space and time. There are some ways to discretize the equations, such as finite volume, finite difference, finite element, and so on. However, regardless of which method will be employed, the points in space where the solution will be computed must be known a *priori*. This set of points is called grid or mesh (Anderson and Wendt, 1995).

Since the numerical solutions of fluid flows are strongly related to grid quality, it is important to ensure that the grid represents the correct geometry of interest and allows an accurate solution. There are two general types of grids: structured and unstructured grids. According to Liseikin (2009), when both the grid point organization and the form of the grid cells are defined by a general rule, i.e. does not depend on cell location, the grid is considered as structured. When the connection of the neighboring grid nodes varies from point to point, the grid is called unstructured.

While finite volumes and also finite element methods, are able to use both types of grids (LeVeque, 2002), classical finite difference schemes (FDS) require a structured grid. Nevertheless, recent works are developing FDS on unstructured grids (Feng *et al.*, 2018; Liu *et al.*, 2020; Zhen *et al.*, 2021).

For classical FDS, the curvilinear grid of physical space must be mapped into a cartesian and uniform grid in the computational space (Anderson and Wendt, 1995). This mapping must have a non-vanishing jacobian and, hence, the point distribution in physical space must be smooth. This work will focus on the generation of structured grids for generalized coordinates system since the our in-house CFD code uses classical FDS (Santos, 2020).

In order to generate these grids, the most commom approach is to take advantage of smoothness of Laplace's operator and set the grid as its solution (Thompson, 1982). However, the use of the classical Laplace's operator is impractical because it does not allow grid refinement as necessary in boundary layer problems. Spekreijse (1995) shows an approach to obtaining a smooth grid, with all the advantages of Laplace's operator that also includes grid refinement at walls as well as boundary orthogonality, based on boundary points distribution. This is due to the control functions present in Poison-type equations (Thompson *et al.*, 1998).

Direct Numerical Simulation (DNS) often requires a considerable number of points in the grid to capture all physical scales. Hence, the numerical solutions of the Poisson-type equation for the grid generation may be too time consuming for serial computing. Unlike previous works from our group where serial mesh generation was employd (Nunes, 2021), a parallel computational technique is applied in order to reduce the solver CPU-Time. This is done using the Message Passing Interface (MPI) paradigm, through the PETSc e-infrastructure (Balay *et al.*, 2023).

The nonlinear Poisson-type equations were discretized with a second-order accurate FDS, and the resulting linear system is solved using parallel Jacobi iterations. The results show the code's performance with classical metrics for parallel implementation, such as scalability, speed-up and efficiency. They also show some test case grids, where the deviation from orthogonality is used as grid quality metric.

2. METHODOLOGY

As introduced in the prior section, FDS are often developed using rectangular and uniform grids. In this way, to describe curvilinear space it is introduced a mapping between the physical (curvilinear) and computational (uniform) spaces and this process is illustrated at Figure 1. Clearly, this mapping must have a non-vanishing Jacobian and also must have continuous metrics. Therefore, this require the smoothness of point distribution in physical space.

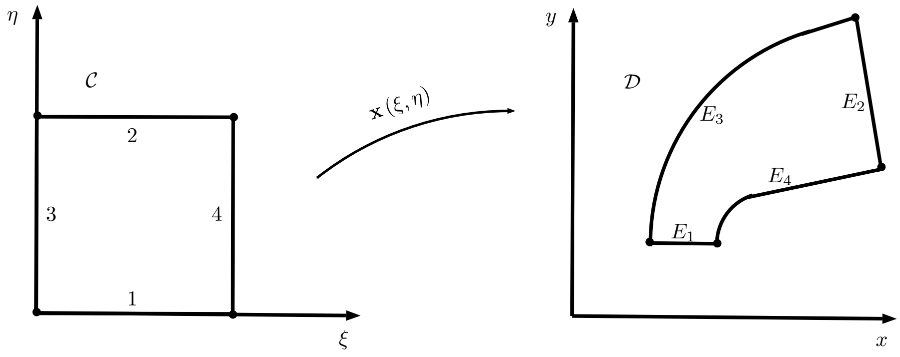

Figure 1. Illustrations of mapping between computational ($\mathcal{C}$) and physical ($\mathcal{D}$) spaces.

Originally Thompson *et al.* (1985) introduced the use of Poison-type equations with prescribed control functions and then Spekreijse (1995) improve this method and proves the existence of a one-to-one mapping. In this approach, an intermediary space here called *parameter space* is introduced between computational and physical spaces, and this procedure allows the interior points at the physical space to reflect the boundary points distribution. This procedure is illustrated in Figure 2.

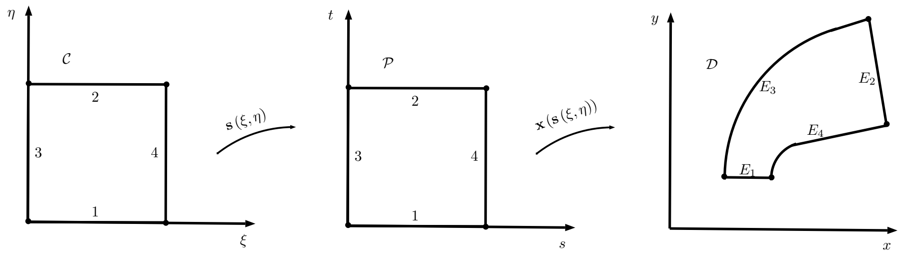

Figure 2. Illustrations of mapping between computational ($\mathcal{C}$) and physical ($\mathcal{D}$) spaces through a intermediary parameter ($\mathcal{P}$) space.

2.1 Mathematical formulation

This work will focus on generations of 2D grids. However, the approach may be used to generate 3D grids as well. All equations and solution procedure are described bellow. The readers are encouraged to review the work by Spekreijse (1995) for a complete and formal discussion about the derivations of these equations.

Defining

$$\mathbf{x} = [x, y]^T, \quad \mathbf{s} = [s, t]^T, \tag{1}$$

allow us to write the elliptic equation used for grid generation as

$$P\,\mathbf{x}_{\xi\xi} + 2\,Q\,\mathbf{x}_{\xi\eta} + R\,\mathbf{x}_{\eta\eta} + S\,\mathbf{x}_{\xi} + T\,\mathbf{x}_{\eta} = 0, \tag{2}$$

where subscripts mean the partial derivatives with respect to computational coordinate system (ξ, η). The coefficients at the above equation are given as

$$P = \langle \mathbf{x}_\eta, \mathbf{x}_\eta \rangle, \tag{3}$$

$$Q = -\langle \mathbf{x}_\xi, \mathbf{x}_\eta \rangle, \tag{4}$$

$$R = \langle \mathbf{x}_\xi, \mathbf{x}_\xi \rangle, \tag{5}$$

$$S = P\,P^1_{11} + 2\,Q\,P^1_{12} + R\,P^1_{22}, \tag{6}$$

$$T = P\,P^2_{11} + 2\,Q\,P^2_{12} + R\,P^2_{22}, \tag{7}$$

where $\langle \cdot, \cdot \rangle$ is the inner product. P^k_{ij} are called control functions and are computed by

$$\begin{pmatrix} s_\xi & s_\eta \\ t_\xi & t_\eta \end{pmatrix} \begin{pmatrix} P^1_{11} \\ P^2_{11} \end{pmatrix} = \begin{pmatrix} s_{\xi\xi} \\ t_{\xi\xi} \end{pmatrix}, \quad \begin{pmatrix} s_\xi & s_\eta \\ t_\xi & t_\eta \end{pmatrix} \begin{pmatrix} P^1_{12} \\ P^2_{12} \end{pmatrix} = \begin{pmatrix} s_{\xi\eta} \\ t_{\xi\eta} \end{pmatrix}, \quad \begin{pmatrix} s_\xi & s_\eta \\ t_\xi & t_\eta \end{pmatrix} \begin{pmatrix} P^1_{22} \\ P^2_{22} \end{pmatrix} = \begin{pmatrix} s_{\eta\eta} \\ t_{\eta\eta} \end{pmatrix}. \tag{8}$$

One can note that the control functions depend only on the parameter space, and do not change during the solution of Eq (2). The inner domain of parameter space is always a function of its boundary values and might be computed by solving the system of equations

$$\begin{aligned} s_{ij} &= s_{i0}\,(1 - t_{ij}) + s_{iNy}\,t_{ij} \\ t_{ij} &= t_{i0}\,(1 - s_{ij}) + t_{Nxj}\,s_{ij}. \end{aligned} \tag{9}$$

to get an elliptic grid. However, it is not orthogonal at boundaries.

The values of parameter variables at boundaries are given as follows

- $s = 0$ at Edge E_1 and $s = 1$ at Edge E_2.
- s is the normalized arclenght along Edges E_3 and E_4.
- $t = 0$ at Edge E_3 and $t = 1$ at Edge E_4.
- t is the normalized arclenght along Edges E_1 and E_2.

This means that the parameter space takes into account clustered regions at prescribed boundary points, and this information is used by control functions to allow cluster regions at the inner domain. As in any elliptical system, the convergence towards an accurate solution is always faster with good initial conditions. Hence, before starting the elliptic grid generation, a simple linear transfinite grid is used to generate a good initial solution.

2.2 Orthogonal boundaries

Simulations of the Navier-Stokes equations for problems that contain a solid wall might often require boundary conditions that uses wall normal gradients. One exemple is the well know wall adiabatic condition. Implementing such conditions without ghost points may be difficult, as the normal wall derivative might be composed of both computational directions. this is extremely facilitated if the layers in physical space are orthogonal with respect to the wall. In his works, Spekreijse (1995) discussed the generation of grids that are orthogonal at all boundaries, however, it is not often required and in this way, this will focus in generate grids with at most three orthogonal boundaries.

The only thing that must be changed to generate an orthogonal grid is the parameter space, the boundary values of s are now computed solving the Harmonic equation

$$P\,\mathbf{s}_{\xi\xi} + 2\,Q\,\mathbf{s}_{\xi\eta} + R\,\mathbf{s}_{\eta\eta} + \Delta\xi\,\mathbf{s}_{\xi} + \Delta\eta\,\mathbf{s}_{\eta} = 0, \tag{10}$$

where P, Q, and R were defined by Eqs (3), (4), and (5) respectively. And also,

$$\Delta\xi = \frac{1}{J}\left[(PJ^{-1})_\xi - (QJ^{-1})_\eta\right], \quad \Delta\eta = \frac{1}{J}\left[(RJ^{-1})_\eta - (QJ^{-1})_\xi\right], \quad \text{and} \quad J = PR + Q. \tag{11}$$

subject to the boundaries conditions

- $s = 0$ at Edge E_1 and $s = 1$ at Edge E_2.
- $\frac{\partial s}{\partial \mathbf{n}} = 0$ along Edges E_3 and E_4 if orthogonality is desired along those.
- $t = 0$ at Edge E_3 and $t = 1$ at Edge E_4.
- $\frac{\partial t}{\partial \mathbf{n}} = 0$ along Edges E_1 and E_2 if orthogonality is desired along those.

One can note that the system of two equations is linear and decoupled, therefore, it will need to solve both equations when orthogonality is desired in different directions. After solving the Eq (10) the inner values of parameter variables are computed through a quadratic Hermite interpolation.

$$\begin{aligned} s_{ij} &= s_{i0}\, H_0(t_{ij}) + s_{iNy}\, H_1(t_{ij}) \\ t_{ij} &= t_{i0}\, H_0(s_{ij}) + t_{Nxj}\, H_1(s_{ij}) \end{aligned} \tag{12}$$

where

$$H_0(l) = 1 - l^2, \quad H_1(l) = l^2. \tag{13}$$

2.3 Numerical Solution

All differential equations were solved using a second-order accurate FDS. The inner domain was discretized using a central scheme whereas boundaries were discretized using a forward-backward stencil. After discretization of the non-linear Eq. (2) and the linear Eq. (10), the resulting algebraic linear system may be writen as

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{14}$$

which can be solved using a Jacobi iteration

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^k \right), \quad \text{with} \quad i = 0, 1, 2, \ldots N-1, \tag{15}$$

where, to guarantee stability, under-relaxation is used, i.e.

$$x_i^{n+1} = \omega x^{k+1} + (1-\omega) x^k, \quad \text{with} \quad 0 < \omega < 1. \tag{16}$$

As shown in Eq. (15), the solution at step $k+1$ depends only the data on k, and due the nature of finite-difference discretization, the solution on each grid node on $\mathbf{x}_{ij}^{k+1}$ depends only the values of $\mathbf{x}_{mn}^{k+1}$ where $m \in (i-1, i, i+1)$ and $n \in (j-1, j, j+1)$. That means the Jacobi iteration is naturally suitable for parallelization.

To proceed the parallel implementation, the computational space is divided by N processors, each processor computes the next iteration of a set of points. Since the code is built using the MPI paradigm, a data communication between processors is performed at the end of each iteration. The code was implemented using the PETSc e-infrastructure (Balay *et al.*, 2023). An algorithm for generating elliptic grids with orthogonal boundaries is given below.

Algorithm 1 Generate elliptic grids with orthogonality at boundaries

```
1: Given a set of boundaries points
2: Generate a simple linear transfinite
3: Compute the parameter boundaries points and then the parameter inner points (Eq. (9))
4: Compute control functions (Eq. (8))
5: Compute elliptic grid (Eq. (2))
If: Orthogonality is desired then:
6: Compute the new parameter space (Eq. (10))
7: Interpolate the boundary values of parameter space using Hermite interpolation (Eq. (12))
8: Compute control functions (Eq. (8))
9: Compute elliptic grid with orthogonality (Eq. (2)).
```

3. RESULTS

All results discussed in this section were obtained using a workstation with the following characteristics: CPU 2 $\times$ intel Xeon © E5-2640 v4 @ 2.40 GHz, 160 Gb RAM. The compiler used was gcc (Ubuntu 9.4.0) and PETSC 3.18.4.

Before discussing parallelism and efficiency, the first results showed two test cases. With this was possible to evaluate the capability of the code to generate grids that respect the desired conditions of clustered regions and orthogonality. The first one is a grid over a blunt body, and the result are shown in Fig. 3. The second test case is a backward-facing step, and the result is shown in Fig. 4.

In both results, each column in the top row shows each step of grid generation. The first column shows the simple linear transfinite grid, the second column shows the elliptic grid with no required orthogonal boundaries, and the third column shows the final elliptic grid with orthogonality required at some edges. The second row show the deviation for orthogonality of each grid in first row. The deviation from orthogonality can be calculated as $|\pi/2 - \theta|$, where θ is the angle between to the lines $\xi = const$ and $\eta = const$ at each point.

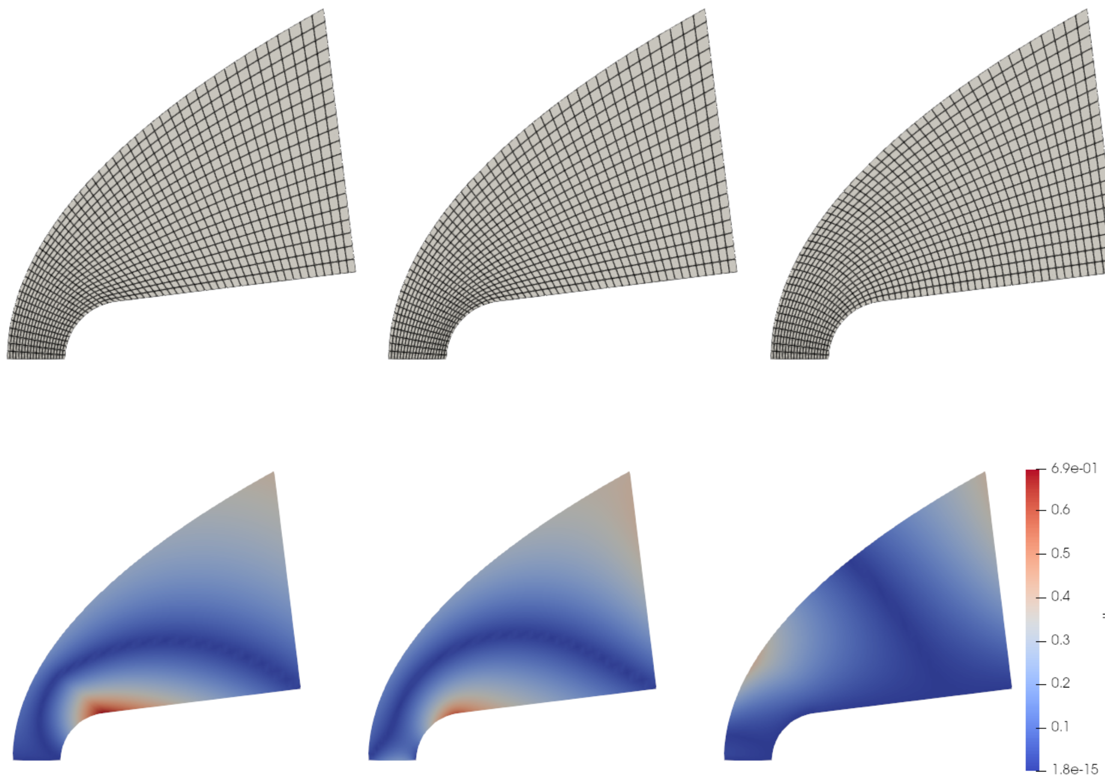

Figure 3. Grid over a two-dimensional bunt body. The top row shows the solution grid for linear transfinite interpolation, elliptic grid, and elliptic grid with orthogonal boundaries at symmetry axis and body wall from left to right, respectively. The bottom row shows the deviation from orthogonality for each grid above it.

‘

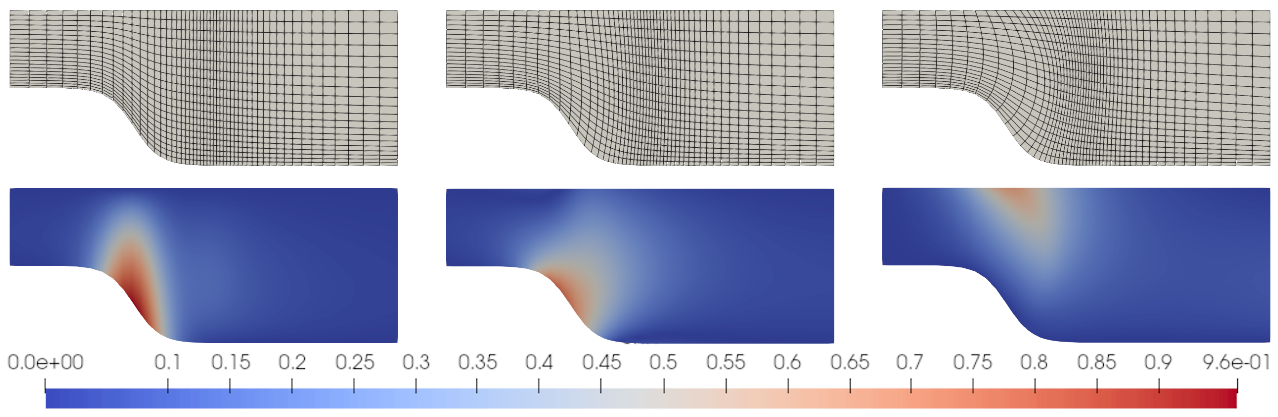

Figure 4. Grid over a two-dimensional backward-facing step. The first row shows the solution grid for linear transfinite interpolation, elliptic grid, and elliptic grid with orthogonal boundaries at inlet and body wall, from left to right, respectively. The second row shows the deviation from orthogonality for each grid above it.

One can note in Fig. 3 that the grid has smooth-spaced points, and no clustered regions. This test case was used to check if orthogonality conditions were imposed correctly and if the grid agrees with these conditions. The results show that the deviation from orthogonality at the symmetry axis and body wall was drastically reduced in the final grid, which means that the final grid is always closer to orthogonal at these edges.

Similar results are shown in Fig.4 for a backward-facing step. However, this grid contains a clustered region behind the step. One might note that clustering regions at boundary points, induces clustered regions at inner domain. This does not occur when using the Laplace equation for grid generation.

Fig. 5 shows the results for the time-asymptotic complexity of the algorithm, when using a single processor. The number of points in each direction was established as $N_x = N_y = N$. The results exponentially approach the expected time-asymptotic behavior of $\mathcal{O}(N^2)$. This is slightly due the non-linearity of the governing equations, which can generate stiffness that delays the convergence of Jacobi procedure, leading to a high number of iteration to for the same tolerance.

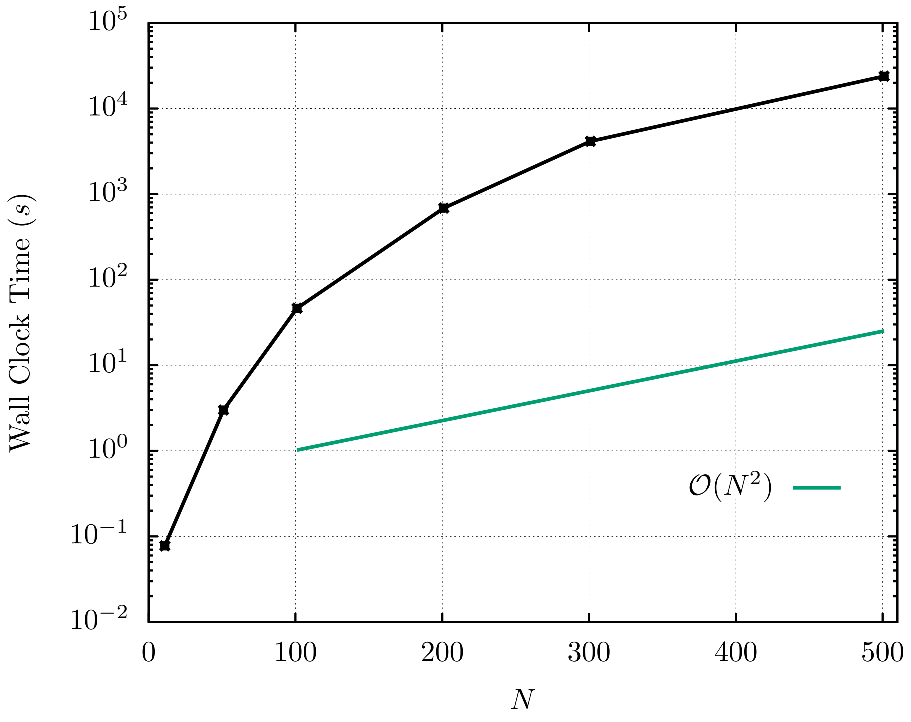

Figure 5. Time-asymptotic complexity of Jacobi solver.

Other important results are showed in Fig. 6, which presents the code's scalability and speed-up. For those tests, a fixed grid size was chosen ($N = 500$), and the number of processors (MPI ranks) is increased from one to sixteen. The speed-up is defined as the ratio between the serial wall-clock time and the parallel wall-clock time. The results also show that the code's behavior is nearly linear. This means every time the number of processors is multiplied by n the wall-clock time required the reach the same convergence is divided by n. The same thing can be verified in the speed-up plot.

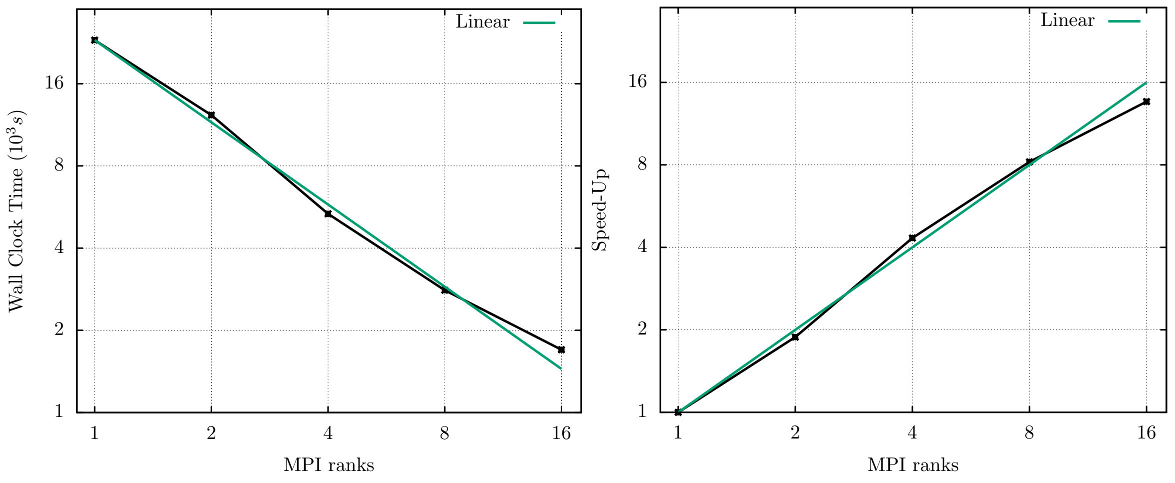

Figure 6. Code's scalability (left) and speed-up (right).

Finally, Fig. 7 shows the code's parallel efficiency. Efficiency is defined here as the ratio between the speed-up and the number of processors used. On this definition, $\eta = 1$ means the code is 100% efficient. Although there is small oscillation around one, the efficiency plots shows the code is nearly linear, which agree with the last results.

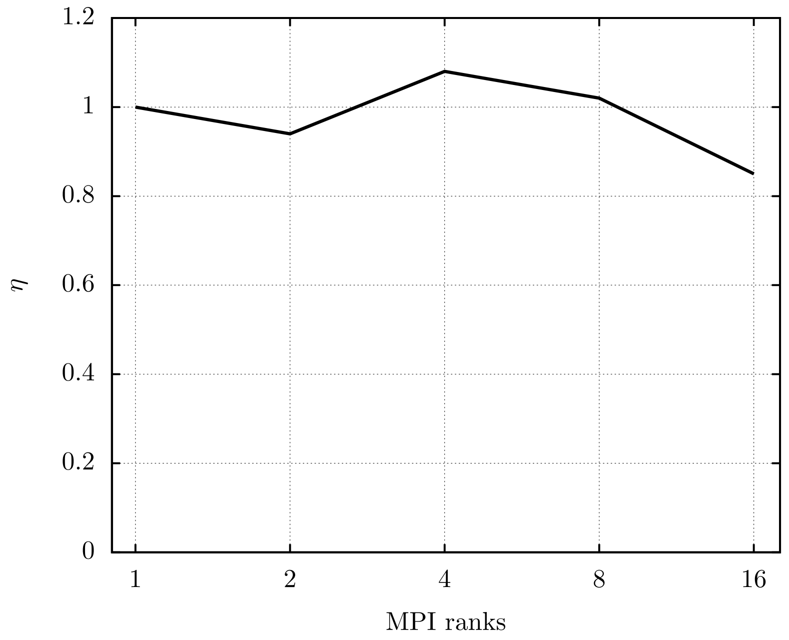

Figure 7. Code's efficiency.

4. CONCLUSION

A methodology for structured elliptic grid generation was discussed and applied to generate smooth grids over different types of bodies. The results show that the grid is always smooth at the inner domain, and orthogonality at the boundaries was successfully imposed.

Clearly, the present approach is limited to regions which have four boundaries. Circular, or O-type grids, may not be directly generated. However, as showed by Spekreijse (1995), it is possible generate O-Type grids, by connecting multiple domains. Nonetheless, this feature was not implement for this study.

The results also show that the time-asymptotic complexity of the algorithm tends to $\mathcal{O}(N^2)$ and the time required may be to high for more complex geometries, where a high number of points and different clustering regions are required. Due to the parallel implementation, the time required is significantly reduced, as shown by scalability results. Although using the same models from other authors, this work successfully shows the impact of parallel implementation on reducing the time required for grid generation.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge the financial support received from SOARD/AFOSR Grant FA9550-18-1-0419, FA9550-22-1-0033, and FA8655-23-1-7031. JCAS gratefully acknowledges the support of CAPES for graduate scholarship and also FAPERJ for Mestrado Nota 10 scholarship.

6. REFERENCES

Anderson, J.D. and Wendt, J., 1995. *Computational fluid dynamics*, Vol. 206. Springer.

Balay, S., Abhyankar, S., Adams, M.F., Benson, S., Brown, J., Brune, P., Buschelman, K., Constantinescu, E.M., Dalcin, L., Dener, A., Eijkhout, V., Faibussowitsch, J., Gropp, W.D., Hapla, V., Isaac, T., Jolivet, P., Karpeev, D., Kaushik, D., Knepley, M.G., Kong, F., Kruger, S., May, D.A., McInnes, L.C., Mills, R.T., Mitchell, L., Munson, T., Roman, J.E., Rupp, K., Sanan, P., Sarich, J., Smith, B.F., Zampini, S., Zhang, H., Zhang, H. and Zhang, J., 2023. "PETSc Web page". `https://petsc.org/`. URL `https://petsc.org/`.

Feng, L., Liu, F., Turner, I., Yang, Q. and Zhuang, P., 2018. "Unstructured mesh finite difference/finite element method for the 2d time-space riesz fractional diffusion equation on irregular convex domains". *Applied Mathematical Modelling*, Vol. 59, pp. 441–463.

LeVeque, R.J., 2002. *Finite volume methods for hyperbolic problems*, Vol. 31. Cambridge university press.

Liseikin, V.D., 2009. *Grid generation methods*, Vol. 1. Springer.

Liu, Y., Yang, L., Shu, C. and Zhang, H., 2020. "Three-dimensional high-order least square-based finite difference-finite volume method on unstructured grids". *Physics of Fluids*, Vol. 32, No. 12.

Nunes, M.S.S., 2021. *Generation of steady-States for Stationary and Convectively Unstable Flows Using the Frequency Displacement Procedure*. Ph.D. thesis, Universidade Federal Fluminense.

Pletcher, R.H., Tannehill, J.C. and Anderson, D., 2012. *Computational fluid mechanics and heat transfer*. CRC press.

Santos, R.D.d., 2020. *Um estudo sobre os métodos de Runge-Kutta com forte estabilidade linear e não linear*. Ph.D. thesis, Universidade Federal Fluminense.

Spekreijse, S.P., 1995. "Elliptic grid generation based on laplace equations and algebraic transformations". *Journal of Computational Physics*, Vol. 118, No. 1, pp. 38–61.

Thompson, J.F., 1982. "Elliptic grid generation". *Applied Mathematics and Computation*, Vol. 10, pp. 79–105.

Thompson, J.F., Soni, B.K. and Weatherill, N.P., 1998. *Handbook of grid generation*. CRC press.

Thompson, J.F., Warsi, Z.U. and Mastin, C.W., 1985. *Numerical grid generation: foundations and applications*. Elsevier North-Holland, Inc.

Zhen, M., Qu, K. and Cai, J., 2021. "A novel finite difference method for euler equations in 2d unstructured meshes". *arXiv preprint arXiv:2102.12933*.

7. RESPONSIBILITY NOTICE

The authors are solely responsible for the printed material included in this paper.