An Acad Bras Cienc (2022) 94(Suppl. 4): e20211024 DOI 10.1590/0001-3765202220211024
Anais da Academia Brasileira de Ciências | *Annals of the Brazilian Academy of Sciences*
Printed ISSN 0001-3765 I Online ISSN 1678-2690
www.scielo.br/aabc | www.fb.com/aabcjournal

ENGINEERING SCIENCES

Acceleration strategies for Tridimensional Coupled hydromechanical problems based on CPU and GPU programming in MATLAB

JEAN B. JOSEPH, PAULO MARCELO V. RIBEIRO, LEONARDO J.N. GUIMARÃES, CICERO VITOR CHAVES JUNIOR & JONATHAN DA C. TEIXEIRA

Abstract: Large-scale fluid flow in porous media demands intense computations and occurs in the most diverse applications, including groundwater flow and oil recovery. This article presents novel computational strategies applied to reservoir geomechanics. Advances are proposed for the efficient assembly of finite element matrices and the solution of linear systems using highly vectorized code in MATLAB. In the CPU version, element matrix assembly is performed using conventional vectorization procedures, based on two strategies: the explicit matrices, and the multidimensional products. Further assembly of the global sparse matrix is achieved using the native sparse function. For the GPU version, computation of the complete set of element matrices is performed with the same strategies as the CPU approach, using gpuArray structures and the native CUDA support provided by MATLAB Parallel Computing Toolbox. Solution of the resulting linear system in CPU and GPU versions is obtained with two strategies using a one-way approach: the native conjugate gradient solver (pcg), and the one provided by the Eigen library. A broad discussion is presented in a dedicated benchmark, where the different strategies using CPU and GPU are compared in processing time and memory requirements. These analyses present significant speedups over serial codes.

Key words: Computational efficiency, explicit matrix, finite elements, geomechanical problem, GPU programming, sparse stiffness matrices.

1 - INTRODUCTION

The multiple physics that occurs from the interaction between the porous medium and the fluids is interesting to several engineering areas, such as reservoir engineering, biomechanics, chemical and environmental engineering (Keyes et al. 2013). An example is the variations in effective pressures and stress due to the extraction of fluids in oil reservoirs located at great depths.

The variations in pore pressure and in the state of effective stresses that interact during the oil extraction demand intensive studies in the area of multiphase flow in the porous media (Binning & Celia 1999, Li et al. 2013), geomechanics (Inoue & Fontoura 2009) as well as in the scope of modern computational techniques (Mena et al. 2020). In reservoir engineering, the coupling between the physics that govern the fluid flow and the mechanical aspects that cause the deformations of the rocky material plays an important role in production through the compaction and subsidence of the

reservoir (Duran et al. 2020), the integrity of capping rocks (Zheng et al. 2017), structural stability of wells (Gomes et al. 2019), and problems of faults reactivation (Soltanzadeh & Hawkes 2009, Pereira et al. 2014, Zhang et al. 2020).

The importance of knowledge on mechanical behaviour grows continuously, as the unconventional (Zoback & Kohli 2019, Mandal et al. 2020) and conventional reservoirs have been increasingly detected and exploted (Bell & Eruteya 2015). Such phenomena need to be accounted in a more effective way in order to optimize the production of hydrocarbons contained in the field. The reservoir engineers, geologists and hydrogeologists use workflows as a decision-making tool, building more realistic reservoir (or aquifer) models every day.

However, the combination of complex models and large scales are often not used. Their study leads to a computational cost that can cause delays in the definition of production projects, restrictions on the number of development scenarios to be analyzed, or even become unfeasible for computational analysis. Such challenges are constantly found in production problems and advanced hydrocarbons in oil reservoirs (Ferrari et al. 2021), geological storage of CO_2 (Almeida et al. 2010) and hydrological modelling. All those application fields present great challenges for numerical computer simulations due to their large dimensions.

Several studies have been carried out in hydrogeological modelling (Bear 1988, Andersen et al. 2001) and mainly in the recovery and production of hydrocarbons in an oil reservoir that uses modern computational techniques combined with parallel procedures (Bear 1988, Lyupa et al. 2016). In problems involving multiple coupled physics, quite often, aiming at a computational gain, the various phenomena that govern the problem are solved in an implicit sequential manner (Kim et al. 2011), this produces a system of equations of different primary unknowns, governed by separate physical phenomena (i.e. subproblems, where the unknown are the pressure of the fluids in the porous of the rock; the displacements related to the mechanical behaviour of the material; and Saturation / Concentration linked to the transport of different fluids/components), which are coupled to each other over the domain.

For the development of these types of systems, one of the techniques that have been used is the vectorization procedure of assembling the matrices of the elements that compose the model, which, when compared to serial code (conventional repetition structures), proves to be extremely efficient (Bird et al. 2017). On the other hand, there are two disadvantages: (i) the huge amount of memory required to store arrays of auxiliary elements and vectors; and (ii) the compromise in the interpretation and flexibility of the code, since many operations are reduced to a few lines of programming. An alternative to the second limitation emerges with the conversion of the original functions to low-level languages; for example, MATLAB allows support of these operations with the construction of MEX (MATLAB EXecutable) functions, which establish an interface between MATLAB (or functions of the MATLAB) and functions developed in C, C ++ or Fortran.

Several authors have explored techniques for calculating products and assembling matrices B and D (Griffiths et al. 2009, Shiakolas et al. 1994). The authors concluded that explicit matrices are more efficient than the use of traditional numerical integration procedures. The construction algorithm in explicit form consists of a search for common positions (or index collisions produced by the mesh connectivity) to sum the individual values of the element's stiffness, and this demands the storage of the complete set of the element matrices. Besides, two other vectors (I, J) with the corresponding

positions in the global stiffness matrix are necessary, requiring a large RAM use and, consequently, compromising the sparse matrix construction process computationally.

To minimize the use of RAM, (Cuvelier et al. 2016) employ a sequential call to the MATLAB sparse function. Another alternative appears with the use of the global stiffness matrix symmetry, thus reducing the memory consumption of the triplet.

Many authors have explored alternatives for better performance of the sparse matrix construction procedure. In this sense, several functions in the MEX format were developed by third parties as alternatives. Engblom & Lukarski (2016) developed the fsparse function, which applies efficient ordering and uses parallel processing to build the sparse matrix. Another proposal comes with the sparse2 function. In both options, no information about the mesh or connectivity of the elements is provided. In general, the two routines perform better than the MATLAB native command (Zayer et al. 2017).

To effectively explore the geometric information of the domain, Dabrowski et al. (2008) proposed a function defined as *sparse_create*, where vectors I and J are replaced by the element's connectivity matrix, which is smaller than necessary in approaches that use global positions. This aspect allows a more compact function and with low memory usage. Besides, the symmetry of the stiffness matrix can be included as an additional feature in the function argument. The efficiency of the *sparse_create* function is directly associated with the number of collisions of the global stiffness matrix indices. In meshes with few occurrences, the performance gain is small, and the most significant advantage will be the reduction of memory consumption. A big advantage in MATLAB in its 2020a version is support for vectors I and J in single precision format, which further reduces memory consumption compared to the double type (MathWorks 2021).

This study aims to use efficient programming practices in MATLAB, which follow the state of the art in vectorization and construction of problem matrices in finite elements. Also, strategies for solving the resulting equation system will be evaluated, including codes in CPU and GPU versions, concepts of explicit matrices and multidimensional products between matrices, and highly optimized functions in MEX language for the construction of element matrices and acceleration of critical parts of the codes. In such a way that the construction of the sparse matrices appears in an original form with the use of gpuArray structures and the great advantages of MATLAB in its 2020a version, through support for vectors I and J in single precision format, which reduces, even more, the memory consumption. The development process of the element matrices is evaluated in terms of its efficiency given the various acceleration options presented by other authors (sparse2, fsparse) and more advanced options that explore the mesh connectivity (sparse_create). All analyzes are performed in combination with linear solvers using the conjugate gradient method, both in CPU and GPU. There was no data transfer between the CPU and the GPU, only in mesh generation was the gpuArray function used to create the mesh on the GPU. From mesh generation to result plotting all data remains on the GPU. Some recent studies have used the GPU as a way of accelerating the assembly of stiffness matrices and solutions of the System of Equations, (Gasparini et al. 2021) developed a structure with multiple GPUs in up to 2000 processes to solve linear sparse problems focusing on reservoir flow oil and geomechanical simulation. (Ferrari et al. 2021) coupled two software, ABAQUS and Echelon, which solve geomechanical and oil reservoir flow problems, respectively. In this study, the authors used GPU to speed up the process. The use of the GPU is also found in software in Health, it was used, for example, in the genomic

analysis of the Parabricks platform. In this architecture, supercomputers and storage connected to the NetApp cloud are used.

In the present work, the flow problem in porous media is coupled with the geomechanical problem through a one-way approach. In this context, the flow problem is solved independently, without returning information from the mechanical problem (such as changes in porosity and permeability). The entire pressure solution history is saved, subsequently transferred unidirectionally to the mechanical problem, where displacements, stresses and strains are computed. The constitutive model used for the mechanical problem is elastic linear. In large scale models, this type of simulation requires the construction and resolution of linear systems of equation with millions of degrees of freedom. Therefore, efficient computational strategies are discussed and applied to a reference benchmark.

The structure of the article is arranged as follows. First, a brief mathematical description is presented. Section 2 presents a finite element representation in space and time in the sense of solving two subproblems: the mechanical and the Darcy flow problem is presented. The computational aspects used for constructing the matrices and the resolution of the flow problem in a deformable porous area applied to oil reservoirs is detailed on section 3. In section 4, a new procedure was presented to accelerate the construction of the local stiffness matrices of the mesh elements, the assembly of the global stiffness matrix and techniques to solve the problem in a more computationally efficient way. In section 5, the numerical results of the validation exercises are presented demonstrating the potential for the proposed schemes.

2 - MATHEMATICAL DESCRIPTION AND FINITE ELEMENT REPRESENTATION

This section presents an outline of the governing equations for coupled deformation, and fluid flow applied to oil reservoirs (i.e., hydro-mechanical problem) (Cuisiat et al. 1998). We will consider that the fluid flow is isothermal and slightly compressible. For simplicity and without loss of generality, we neglect the capillary effects and consider that the porous medium presents linear elastic behaviour and is subject to the hypothesis of small strains. Such governing equations are composed of conservation and constitutive laws.

Conservation equations are related to the amount of movement and mass. On the macroscopic scale, two main approaches can be found depending on Eulerian and Lagrangian descriptions: the pioneering research by Terzaghi and Biot (Biot 1941) provides a concise explanation for the Eulerian case; and the work of Coussy (Coussy 2004) develop the Lagrangian case. Under the assumption of quasi-static condition, the mechanical deformation can be expressed as follows:

$$\nabla \cdot \sigma + \rho_s g = 0 \tag{1}$$

where σ is the total stress tensor, ρ_s is the density of the solid material, g is the gravity vector, and 0 the null vector. In terms of Cauchy's effective stress tensor, the momentum conservation equation can be written as:

$$\nabla \cdot \left(\sigma' - \alpha p I\right) + \rho_s g = 0 \tag{2}$$

where σ' is the Cauchy effective stress tensor, p the fluid pressure in the pores (in MPa), α is the Biot coeficiente, and $I \in \mathbb{R}^{3x3}$ is the identity matrix. The set of equations in (1) are subjected to the following boundary conditions:

$$\sigma \cdot n = t \ \ in \ \Gamma_N^u \tag{3a}$$

$$u = u_0 \ \ in \ \ \Gamma_D^u \tag{3b}$$

In addition, to the momentum conservation equation and boundary conditions, it is necessary to use equations that relate strains-displacements, known as kinematic relations (Zienkiewicz 2000). Considering the hypothesis of small strains, according to which the strains and rotations are considered infinitesimal strain, the relationship is described as:

$$\varepsilon(u) = \frac{1}{2}\left(\nabla u + \nabla^T u\right) \tag{3c}$$

where u is the displacement vector.

To characterize the material behavior, the effective stress tensor is defined by the following stress-strain relationship:

$$\sigma' = \mathcal{D}\varepsilon \tag{4}$$

where $\mathcal{D} = 2\mu\mathbb{I} + \lambda I \bigotimes I$ is the constitutive matrix of the material, and ε is the elastic strain tensor. With $\mathbb{I} = I - \frac{1}{3} I \bigotimes I$, and $I \bigotimes I$ denoting the tensorial product of the matrices I, the parameters μ and λ and are the Lamé constants (in MPa).

Similarly to Bear (Bear 1988), the mass conservation of the fluid phase in saturated porous medium and the solid phase, are expressed as:

$$\frac{\partial \phi \rho_f}{\partial t} + \nabla \cdot \left(\rho_f v_f + \rho_f \phi \dot{u}\right) = \omega \tag{5a}$$

$$\frac{\partial (1-\phi)\rho_s}{\partial t} + \nabla \cdot \left([1-\phi]\, \rho_s \dot{u}\right) = 0 \tag{5b}$$

where ϕ is the porosity, ρ_f the density of the fluid, $v_f = -\frac{K}{\mu}(\nabla p - \rho g)$ is the fluid velocity described by Darcy's law, ω is the term of mass source, ρ_s the density of the solid and $\dot{u} = \frac{\partial u}{\partial t}$ is the solid phase velocity.

Considering that the fluid phase is slightly compressible, the relationship between the density of the fluid ρ_f and the pore pressure p can be written based on the compressibility of the fluid c_f as:

$$c_f = \frac{1}{\rho_f} \cdot \frac{\partial \rho_f}{\partial p} \tag{6}$$

Combining Eqs. (5a) and (6), after some algebraic manipulations and disregarding the fluid and solid density gradients, the following mass conservation equation for a deformable medium is detained:

$$c_f \phi \frac{\partial p}{\partial t} + \nabla \cdot v_f + \nabla \cdot \dot{u} = f \tag{7}$$

where f is the volumetric source term. The boundary condition is:

$$v_f \cdot n = -\overline{s} + \gamma\left(p - \overline{p}\right) \quad \text{in} \quad \Gamma^p \tag{8}$$

with $\overline{s}$, γ and $\overline{p}$ are the imposed volumetric flow (in m³ / s), the penalty term (m³ / s / MPa), and the prescribed pore pressure, respectively.

The strong form of the coupled deformation problem and fluid flow applied to oil reservoirs can be summarized in two sets of equations, one for the mathematical description for poroelasticity and the other for the fluid velocity described by Darcy's law and is completed by the application of Dirichlet and Neumann boundary conditions, through of Eqs. (7) and (8), respectively.

3 - NUMERICAL FORMULATION

In this section, the numerical resolution scheme of the formulation described in the previous section will be presented. The time derivatives are approximated by the implicit Euler method. The index $(\circ)^{n+1}$ represents terms evaluated in the subsequent state n + 1 and the $(\circ)^{n}$ means terms evaluated in the last state n. For spatial variables, a classic finite element discretization (Galerkin) is adopted (Zienkiewicz 2000).

Second-order derivatives of conservation laws are reduced to first-order derivatives by using the Gauss divergence theorem after integration by parts. Numerical approximations H^1 will be used for the two sets of equations that model the poroelasticity and mass.

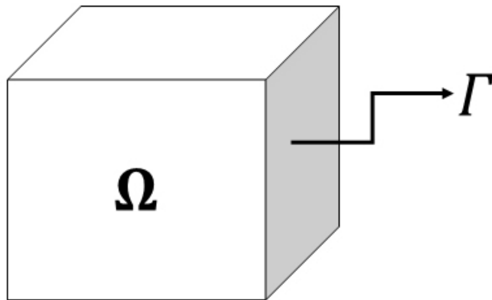

Figure 1. The problem domain.

In the Figure 1 below it is described the domain $\Omega \in \mathbb{R}^3$ bounded by the surface $\Gamma \in \mathbb{R}^2$, with Ω splitted into subdomains or convex elements Ω_e. The boundary of each element is defined as Γ_e and its normal unit vector domain given by n. For the weak form of the poroelastic problem, u_h is used as an approximation variable for the displacement field, while the p_h to the pore pressure field approximation used for the Darcy flow problem. Therefore, we have to $u_h \in V_h$ and $p_h \in S_h$, where we define the required functional approximation spaces:

$$V_h = \left\{ u_h \in \left[H_h^1(\Omega)\right]^3 : u_h = u_0 \;\; on \;\; \Gamma_D^u \right\} \tag{9}$$

$$S_h = \left\{ p_h \in H_h^1(\Omega) \right\} \tag{10}$$

with the respective finite-dimensional subspaces of the allowable variations:

$$V_0 = \left\{ v \in \left[H_h^1(\Omega)\right]^3 : v = 0 \; on \;\; \Gamma_D^u \right\} \tag{11}$$

$$S_0 = \left\{ q \in H_h^1(\Omega) \right\} \tag{12}$$

where $H^1(\Omega)$ is the Sobolev space of the functions with square-integrable derivatives and q the flow in the element.

Choosing admissible members of the function spaces, previously defined, as trial and tests functions following the standard Galerkin method, we have to:

$$u_h = \sum_{i=1}^{m} N_i u_i = N_u^e u^e \tag{13}$$

$$w = \sum_{i=1}^{m} \overline{N}_i w_i = \overline{N}_w^e w^e \tag{14}$$

$$p_h = \sum_{i=1}^{m} N_i p_i = N_p^e p^e \tag{15}$$

$$q = \sum_{i=1}^{m} \overline{N}_i q_i = \overline{N}_p^e q^e \tag{16}$$

where m the number of nodes of each element (tetrahedrons = 4), u_i, w_i, q_i and p_i are the nodal values of the displacements, weighting functions and pore pressures of the element e, respectively. N_i, $\overline{N}_i$, N_i and $\overline{N}_i$ are functions chosen appropriately in order to belong to the spaces V_h, V_0, S_h and S_0 respectively.

Following the standard procedures of the finite element method in the sense of Galerkin's method (Zienkiewicz 2000), we multiply Eqs. (2) and (7) with the respective weighting (tests) functions, integrating by parts the higher order terms, followed by the application of the divergence theorem and, finally, considering the boundary conditions, we eventually obtain the following weak formulation of the problem:

$$\int_{\Omega_e} \left[\lambda \left(\nabla \cdot u_h\right) I + \mu \left(\nabla u_h + \nabla^T u_h\right)\right] : \nabla w \, d\Omega_e = \int_{\Omega_e} \alpha p_h I : \nabla w \, d\Omega_e - \int_{\Omega_e} \rho_s g \cdot w \, d\Omega_e + \int_{\Gamma_N^u} t \cdot w \, d\Gamma_e \tag{17a}$$

$$\begin{aligned} \int_{\Omega_e} \left(\phi c_f\right) p_h^{n+1} q \, d\Omega_e - \Delta t \int_{\Omega_e} \nabla q \frac{K}{\mu} \nabla p_h^{n+1} \, d\Omega_e & \\ + \Delta t \int_{\Gamma^p} \gamma p_h^{n+1} q \, d\Gamma^p = & \\ \int_{\Omega_e} \left(\phi c_f\right) p_h^n q \, d\Omega_e & \\ - \Delta t \int_{\Omega_e} \nabla \cdot \left(\rho_f \frac{K}{\mu} g\right) q \, d\Omega_e - \Delta t \int_{\Omega_e} \nabla \cdot \dot{u} \, q \, d\Omega_e + \Delta t \int_{\Omega_e} f q \, d\Omega_e & \\ + \Delta t \int_{\Gamma^p} \left(\overline{s} + \gamma \overline{p}\right) q \, d\Gamma^p & \end{aligned} \tag{17b}$$

replacing (13) - (16) in (17) and suppressing the indexes we can rewrite more compactly:

$$\int_{\Omega_e} w^{eT} B_u^{eT} D B_u^e u^e d\Omega_e = \int_{\Omega_e} w^{eT} B_u^{eT} \alpha I N_p^e p^e d\Omega_e - \int_{\Omega_e} w^{eT} N_u^{eT} \rho_s g d\Omega_e + \int_{\Gamma_N^u} w^{eT} N_u^{eT} t \, d\Gamma_e \tag{18a}$$

where $B_u^e = \nabla N_u^e$ and $m = (1,1,1,0,0,0)$ is an auxiliary vector.

$$
\begin{aligned}
& l\left\{\int_{\Omega_e} q^{eT} N_p^{eT} \left(\phi c_f\right)^e N_p^{eT}\, d\Omega_e - \Delta t \int_{\Omega_e} q^{eT} \left(\nabla N_p^e\right)^T \left[\frac{K}{\mu}\right]^{\mathrm{e}} \nabla N_p^e\, d\Omega_e \right. \\
& \left. +\Delta t \int_{\Gamma^{\mathrm{p}}} q^{eT} \left(\nabla N_p^e\right)^T \gamma N_p^e\, d\Gamma^p \right\} \left(p^e\right)^{n+1} \\
& = \int_{\Omega_e} q^{eT} N_p^{eT} \left(\phi c_f\right)^e N_p^e\, \left(p^e\right)^n\, d\Omega_e \\
& - \Delta t \int_{\Omega_e} q^{eT} \left(\nabla N_p^e\right)^T \left[\rho_f \frac{K}{\mu}\right]^{\mathrm{e}} \nabla \cdot g\, d\Omega_e \\
& - \Delta t \int_{\Omega_e} q^{eT} N_p^{eT} \mathrm{m}^{\mathrm{T}}\, B_u^e \dot{u}^e\, d\Omega_e \\
& + \Delta t \int_{\Omega_e} q^{eT} N_p^{eT} f^e\, d\Omega_e \\
& + \Delta t \int_{\Gamma^{\mathrm{p}}} q^{eT} N_p^{eT}\, \left(\overline{s} + \gamma \overline{p}\right)\, d\Gamma^p
\end{aligned}
\tag{18b}
$$

As w^{eT} and q^{eT} are arbitrary functions, then equations (18) can be written as:

$$
\int_{\Omega_e} B_u^{eT} D B_u^e u^e d\Omega_e = \int_{\Omega_e} B_u^{eT} \alpha I N_p^e p^e d\Omega_e - \int_{\Omega_e} N_u^{eT}\, \rho_s g d\Omega_e + \int_{\Gamma_{\mathrm{N}}^{\mathrm{u}}} N_u^{eT} t\, d\Gamma_e \tag{19a}
$$

$$
\begin{aligned}
& \left\{\int_{\Omega_e} N_p^{eT} \left(\phi c_f\right)^e N_p^{eT}\, d\Omega_e - \Delta t \int_{\Omega_e} \left(\nabla N_p^e\right)^T \left[\frac{K}{\mu}\right]^{\mathrm{e}} \nabla N_p^e\, d\Omega_e \right. \\
& \left. +\Delta t \int_{\Gamma^{\mathrm{p}}} \left(\nabla N_p^e\right)^T \gamma N_p^e\, d\Gamma^p \right\} \left(p^e\right)^{n+1} \\
& = \int_{\Omega_e} N_p^{eT} \left(\phi c_f\right)^e N_p^e\, \left(p^e\right)^n\, d\Omega_e - \Delta t \int_{\Omega_e} \left(\nabla N_p^e\right)^T \left[\rho_f \frac{K}{\mu}\right]^{\mathrm{e}} \nabla \cdot g\, d\Omega_e \\
& - \Delta t \int_{\Omega_e} N_p^{eT} \mathrm{m}^{\mathrm{T}}\, B_u^e \dot{u}^e\, d\Omega_e \\
& + \Delta t \int_{\Omega_e} N_p^{eT} f^e\, d\Omega_e + \Delta t \int_{\Gamma^{\mathrm{p}}} N_p^{eT}\, \left(\overline{s} + \gamma \overline{p}\right)\, d\Gamma^p
\end{aligned}
\tag{19b}
$$

For the mechanical problem (linear momentum equation - equilibrium equation), the first term on the left side of Eq. (19a) forms the stiffness matrix of the element, while the last two terms on the right side of the equation, represent the vector of external forces and finally, the first term of r.h.s is the coupling matrix of the hydro-mechanical problem. The matrix representation of Eq. (19a) is described below by equation (20a)

$$
\overbrace{K_u^e}^{\textit{stiffness matrix}} \quad \overbrace{u^e}^{\textit{nodal displacement}} = \overbrace{F_{p^e}}^{\textit{first term}} - \overbrace{F_g + F_t}^{\text{last two terms on the r.h.s}} \tag{20a}
$$

As for the Darcy flow problem (mass conservation equation), we have the l.h.s. in the Eq. (19b) is formed by the terms of storage (first term), penalty and the Darcy flow term, in which it presents the same form as the stiffness matrix for the mechanical problem. The first two terms r.h.s. of Eq. (19b) are

the storage and the gravitational term, respectively. The third term represents the coupling term of the tensor-deformation behaviour dependent on the acting stresses that cause the variation of the porous volume (volumetric strain) and the last term of Eq. (19b) represents the sources / sinks of the Darcy flow problem. The equation (20b) represents the matrix representation of Eq. (19b).

$$K_p^e\{p^e\}^{n+1} = \left\{ \overbrace{\widehat{F_{p^e}}}^{\text{the storage term}} - \overbrace{\widehat{F_g}}^{\text{gravitational term}} - \overbrace{\widehat{F_u}}^{\text{the coupling term}} + \overbrace{\widehat{F_{\overline{S}}}}^{\text{sources/sinks}} \right\}^n \tag{20b}$$

For the hydromechanical coupling problem treated in the present study, the one-way coupling will be used (Settari & Walters 2001, Kim et al. 2011). This type of coupling consists of solving the flow and mechanical problem separately. At the same time, the answers to the first problem feed the second, but the solution to the second does not feed the first. This exchange of information is done using the nodal pore pressures p^n of the flow problem obtained in the specific time steps, called n, and later converted into static loads for the mechanical problem, as shown in Figure 2.

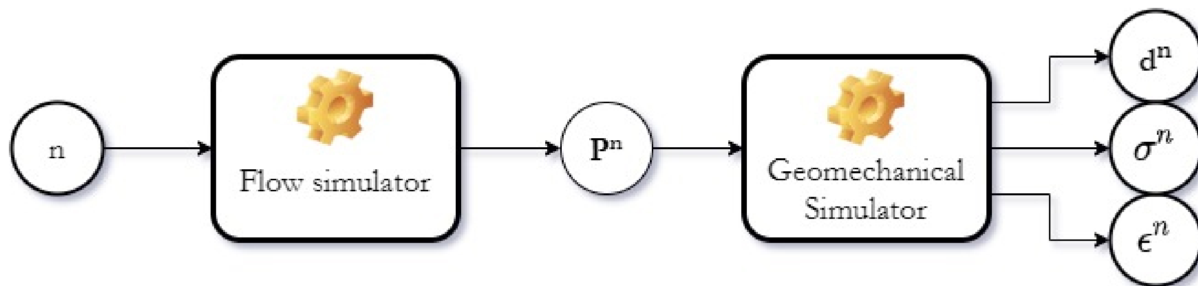

Figure 2. One-way coupling flowchart.

In this article, all the analyzes were concentrated on the stiffness matrix assembly and some aspects of computational efficiency in solving the geomechanical problem.

4 - NOVEL COMPUTACIONAL PROCEDURES

The codes were developed in MATLAB language and are divided into two modules: (i) mechanical and (ii) single-phase flow in porous media, so in this article, the objective was to study strategies to improve the computational efficiency of the mechanical problem. The mechanical problem is defined by the FEM formulations for the tetrahedral element discussed in section 3 by Eq. (19a). The flow module (Darcy flow problem) also follows the formulation discussed in section 3 by Eq. (19b) and is approximated by linear tetrahedra. The two problems are partially coupled in a unidirectional way (one-way). The code is divided into steps, which allow optimization of the mechanical problem according to the following structure i) stiffness matrix assembly of the element (Ke); ii) construction of the global rigidity matrix (KG); iii) coupling and solution of the system of equations.

In the specific case of GPU processing, the Parallel Computing Toolbox is used, with an interface for NVIDIA GPUs. Structures of the gpuArray type are defined, which allow the application of a series of commands for matrix operations. These operations are performed both in the development phase of the problem matrices and in the solution of the system of equations. The graphics processing unit (GPU), has a single chip as the CPU but with the main difference in the number of cores, since it can

have hundreds of cores aligned to a single hardware as shown in Figure 3b. For a computationally massive problem, the GPU surpasses the CPU in GFLOPS (Giga FLoating point OPerations per Second). CUDA (Compute Unified Device Architecture) is the architecture used by NVIDIA for implementation on the GPU (NVIDIA. CUDA C programming guide, 2020) with integration with the C ++ language.

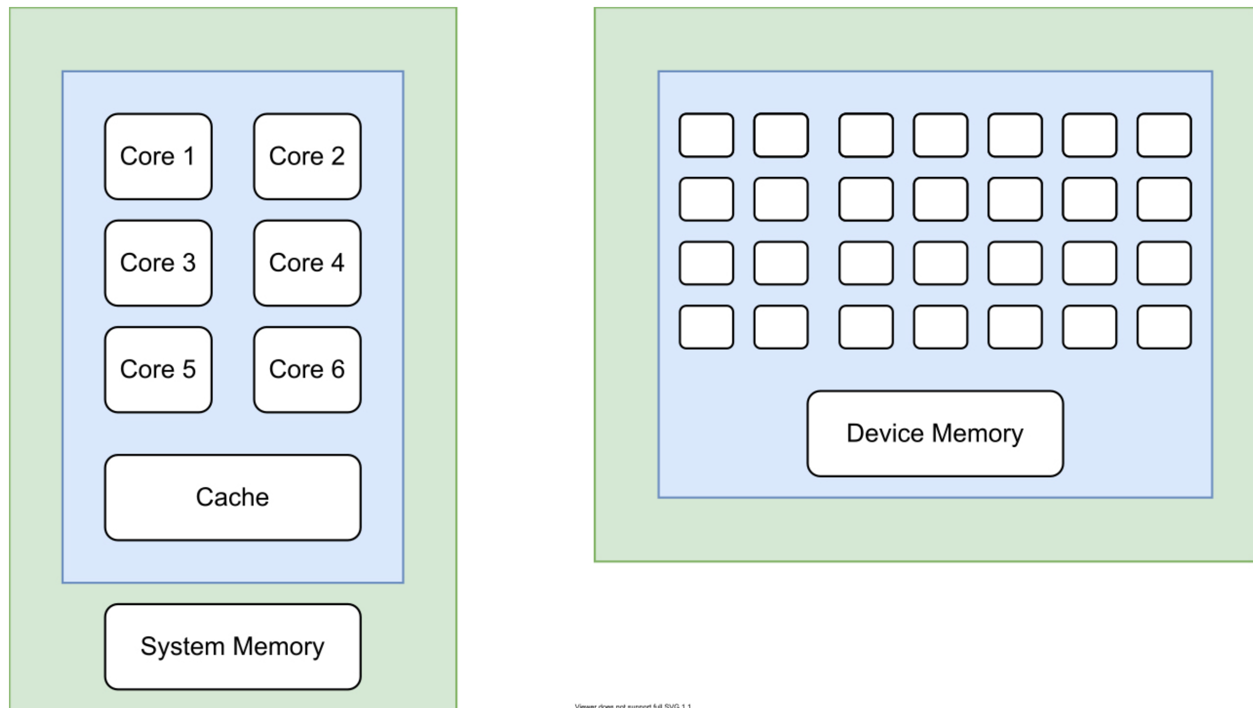

Figure 3. Difference between CPU and GPU in terms of number of cores. (a) CPU with multicolors and (b) GPU with hundreds of colors.

The GPU use in MATLAB requires little effort due to native function support and does not require great CUDA knowledge.

A general outline of the code structure and analysis options is shown in Figure 4.

Three sectors of the codes were the object of study: the explicit and implicit assembly of the local stiffness matrix, the global assembly, and the analysis of techniques to accelerate the solver.

4.1 - Stiffness matrix assembly of the elements (Ke)

According to the scheme shown in Figure 4, the several options for the assembly of the stiffness matrix of the elements are discussed below.

4.2 - Exp (Explicit matrix assembly)

The explicit assembly technique presents a closed form and excellent computational efficiency in algebraic operations. It also results in two significant advantages: the absence of product between matrices and numerical integration procedures. The result is a symbolic expression for each term in the stiffness matrix of the elements. In computational terms, it results in a simple computation of the terms involved in each element of the stiffness matrix, with possible elimination of symmetric terms

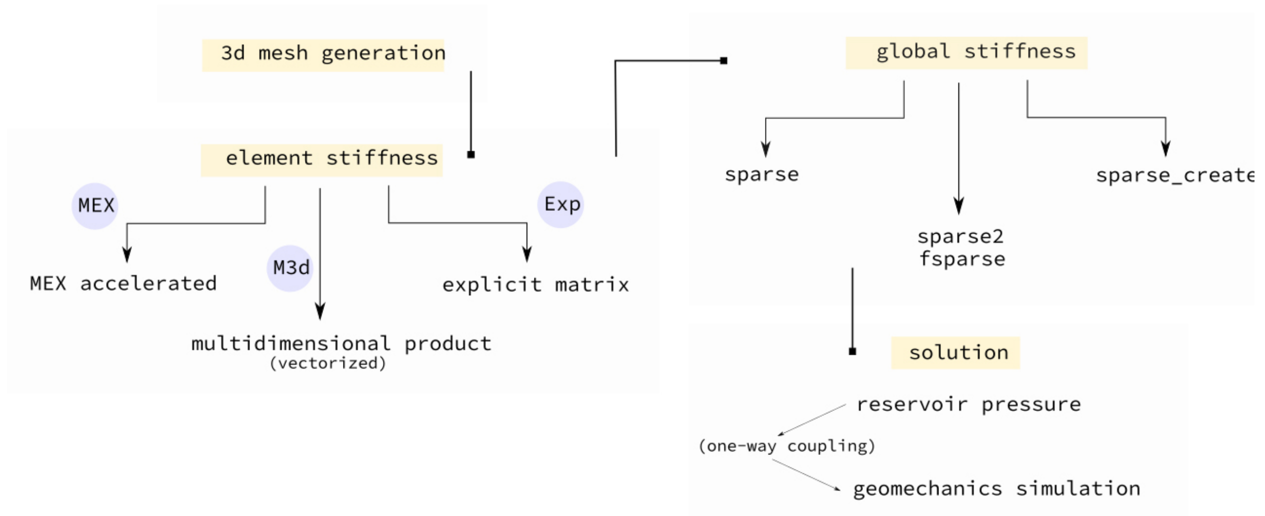

Figure 4. General scheme of the code structure.

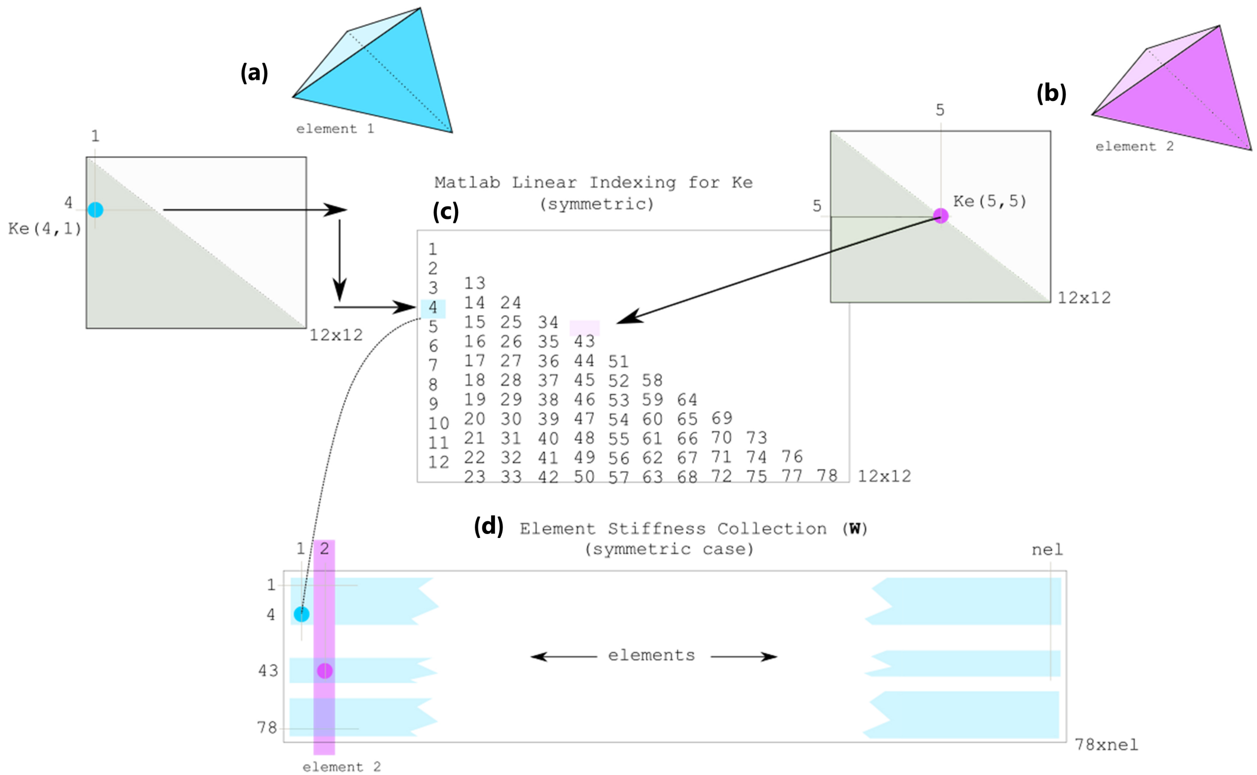

Figure 5. Linear indexing and idea of the method for allocation of terms from the explicit matrix (symmetry case). (a) element 1; (b) element 2; (c) lower triangular indices; (d) final collection with arrays of elements.

and compression of common variables. This process can be performed both on the CPU and GPU. In this last case, it is only necessary to replace the conventional matrices for gpuArray type structures. The computational procedures for the assembly of the W matrix in Figure 5 consist of elaborating explicit expressions for the terms of the lower triangular part. For linear tetrahedron (with 3 degrees of freedom per node), this implies 78 unique terms. The terms are organized in a logic that follows

MATLAB's linear indexing pattern, with columns being read frequently. In the symmetrical option, only those terms are stored, while in the option without (complete) symmetry, the other terms are obtained by mirroring the lower part. Figure 5 provides an example for the symmetrical option. A linear index is assigned to each term of the stiffness matrices element (Figure 5a-b). In sequence, these terms are allocated to a global collection of terms, where each column corresponds to the contribution of a given element (Figure 5d). In the end, a matrix is obtained with 78 rows and number of columns equal to the total number of mesh elements (nelem).

The option with symmetry presents expressive advantages in terms of physical memory. For the W collection, this implies 46% savings in a linear tetrahedron (only 78 terms out of 144). The same behaviour is expected for vectors L and C, which contain the global position indices. The reference code for the suggested procedure is indicated in Algorithm 1.

Algorithm 1: sample code for W computation

```
1   % computes nodes in a vectorized form
2   node1=conec(:,1);node2=conec(:,2);node3=conec(:,3);node4=conec(:,4);
3   % computes coordinates in a vectorized form
3   X1=coord(node1,1); Y1=coord(node1,2); Z1=coord(node1,3); ...
4   % evaluates auxiliar geometric vectors for the explicit assembly
5   b1=…; c1=…; d1=…; b1b1=b1*b1; c1c1=c1*c1; d1d1=d1*d1;
7   % ---------------------------------------------------------------
8   % computes each individual line of the stiffness collection W
9   % ---------------------------------------------------------------
10  K1= (E.*(2*b1b1*pois + 2*c1c1*pois + 2*d1.∧2*pois - 2*b1b1 -- …
    c1c1 - d1d1)). /(72*VV*(2*pois∧2 + pois - 1));
11  % continues for the remaining lines and defines W
12  W=[K1;K2;K3; K4; …
13  V=W(:); % V is the Column-shaped local stiffness matrix
```

Note: some lines omitted for sake of simplicity.

The calculus of each line in W is vectorized, which means that it is processed simultaneously in all the mesh elements. This is the main difference in Algorithm 1. An example is given for the first term K1 of the linear tetrahedron, indicated below:

$$K1 = \left(E. * \left(2 * b1b1 * pois + 2 * c1c1 * pois + 2 * d1.^{2} * pois - 2 * b1b1 - c1c1 - d1d1\right)\right) ./ \left(72 * VV * \left(2 * pois^{2} + pois - 1\right)\right)$$

where *K1* represents a vector with a dimension equal to the number of elements in the mesh, and each column corresponds to a term of the stiffness matrix of a specific element. *E* the Young's Module, *pois* the poison's ratio. The terms b1, c1, d1 and their products b1b1, c1c1 and d1d1 indicate constants associated with the nodal coordinates and vectors. The same occurs for *VV* in line 10 of algorithm

1, which specifies a vector with the volume of all elements in the mesh. That way, *K1* is an algebraic operation between vectors, resulting in a new vector, and it is also used to define the columns in the terms Ke (1,1) of the stiffness matrix of the elements in the mesh. The same logic could be applied to the remaining terms of the stiffness matrix of the elements.

There is still the evaluation of vectors L and C responsible for indexing the global stiffness matrix values. The construction of these vectors can be easily vectorized, and the logic is the recovery of global positions associated with each degree of local freedom. For the linear tetrahedron, with 3 degrees of freedom per node, the following relations are effective:

$$[3n_i - 2 \quad 3n_i - 1 \qquad 3n_i] \rightarrow \textit{for each node } n_i$$

Algorithm 2: sample code for L and C computation

```
1   % computes global indexes in a vectorized fashion
2   % GL is a collection of 12 global indexes in each line
3   GL=[3*node1-2 3*node1-1 3*node1 3*node2-2 3*node2-1 3*node2 …
    3*node3-2 3*node3-1 3*node3 3*node4-2 3*node4-1 3*node4];
4   % ----------------------------------------------------------------
5   % Option 1 - full computation (without symmetry)
6   % ----------------------------------------------------------------
7   GL=permute(GL,[3 2 1]); % organize in 3d slices for each element
8   aux=repmat(GL,1,12,1); % repeat each slice in 12 columns
9   L=aux(:); % organize in vector format
10  aux=repmat(GL,12,1,1); % repeat each slice in 12 lines
11  C=aux(:); % organize in vector format
12  % ----------------------------------------------------------------
13  % Option 2 - symmetric computation
14  % ----------------------------------------------------------------
15  % the indexing in the next line follows Figure 5 explanation
16  aux=GL(:,[1:12 2:12 3:12 4:12 5:12 6:12 7:12 8:12 9:12 10:12 …
    11:12 12]);
17  L=aux(:);% organize in vector format
    % the indexing in the next line follows (See Figure 5)
18  aux=GL(:,[1*ones(1,12) 2*ones(1,11) 3*ones(1,10) 4*ones(1,9) …
    5*ones(1,8) 6*ones(1,7) 7*ones(1,6) 8*ones(1,5) 9*ones(1,4) …
    10*ones(1,3) 11*ones(1,2) 12]);
19  C=aux(:); % organize in vector format
```

Note: some lines omitted for sake of simplicity.

Therefore, each node results in 3 global indexes. There are 12 positions in total for the linear tetrahedron, which can be obtained by repeating the above rule for each of the 4 nodes of the element (defined later as lines of GL matrix). The complete definition comes with the permutations of these terms for each element in the mesh. In the lack of symmetry, each element contributes 144 positions to the vectors L and C. In the symmetric case, this dimension is reduced to 78. Algorithm 2 indicates the required procedures for constructing these vectors, which are also valid for the problem of a linear tetrahedron. After the triplets evaluation, the global stiffness matrix is obtained with a single call of the sparse command, as discussed in previous sections.

4.3 - M3d – Multidimensional product

This strategy employs the concept of a multidimensional array product and appears as an alternative to the procedure with explicit matrices. It explores the characteristic that matrices B and D can be quickly computed in a vectorized function (following the same logic of previous section) and the product involved in the construction of the elements matrices can be performed synchronously. As long as individual planes are defined for each element, and a simultaneous operation according to the Eq. (21) are possible:

$$(K_e)_\alpha = B_u^{\alpha T} D_\alpha B_u^\alpha V_\alpha \tag{21}$$

where the several planes (or pages) α are associated with mesh elements as shown in Figure 6.

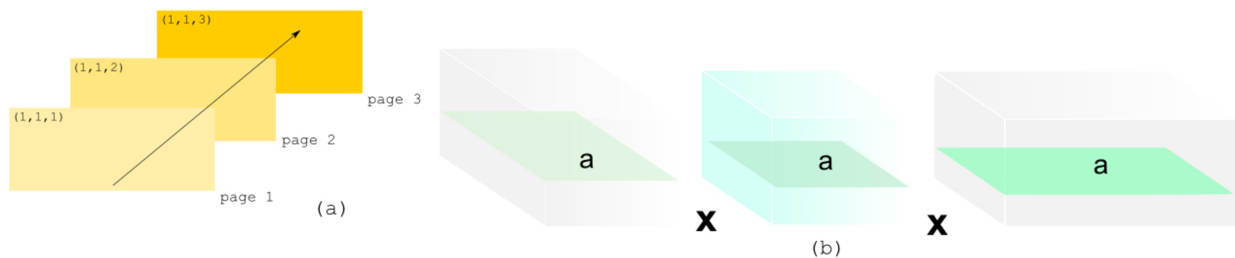

Figure 6. Interpretation of the concept of multidimensional arrays: (a) several pages of the same index, (b) the multidimensional product applied to the problem, with evidence for the α plane.

This procedure avoids two specific problems: (i) the construction of explicit terms presented in the previous section, which involves tedious algebraic computation; (ii) the serial (and slow) realization of numerous products among the matrices that integrate the elements in the stiffness matrix (identified by planes). As a disadvantage, it performs the complete product between the matrices and does not take any advantage of the problem symmetry.

In this paper, two options are given for the multidimensional product:

1) The compilation of a MEX function called MTIMESX (Tursa 2022), suitable for CPU operations;

2) The application of native *pagefun* command, which applies operations to each page of a multidimensional array stored in the GPU.

The code structure for any of these options is very similar, and the GPU details are presented in Algorithm 3 for the case of a linear tetrahedron. It is very easy to create a vector on the GPU and to transfer data to the GPU, just use the MATLAB function *gpuArray*. This is easily reached by using gpuArray type structures.

Algorithm 3: sample code for multidimensional product on GPU

```
1  % coordinates are converted from CPU to GPU arrays
2  X1=gpuArray(X1); Y1=gpuArray(Y1); Z1=gpuArray(Z1); …
3  % defines 3d arrays for B and D on the GPU
4  B=zeros(6,12,nel,'gpuArray'); D=repmat(D,1,1,nel); D=gpuArray(D);
5  % defines element volumes on the GPU
6  VV=gpuArray(VV);
7  % populates B indexes (sample indexes presented)
8  B(1,1,:)=(1./(6*VV)).*b1;
9  B(1,4,:)=(1./(6*VV)).*b2; …
10 % defines volume vector as pages (slices)
11 Ve=permute(VV', [3 2 1]);
12 % ---------------------------------------------------------------
13 % computes the multidimensional product B'DBV with pagefun on GPU
14 % ---------------------------------------------------------------
15 % first computation B'D
16 BtD=pagefun(@mtimes,pagefun(@ctranspose,B),D);
17 % second computation B'DB
18 BtDB=pagefun(@mtimes,BtD,B);
19 % final computation
20 Ke=pagefun(@mtimes,BtDB,Ve);
```

Note: some lines omitted for sake of simplicity.

4.4 - Global stiffness matrix assembly (KG)

Once the L, C and V triplets are obtained, the procedure follows with the global stiffness matrix construction, defined as sparse, for better storage efficiency. For large scale problems, two important characteristics must be considered: (i) the size occupied in RAM by the triplets; (ii) the time required to construct the sparse matrix. Item (i) and the possible alternatives for RAM reduction have already been discussed in the previous section. Operations with native solvers in MATLAB require the explicit storage of matrices. This way, emphasis will be given to the options for the solution in item (ii). The numerical experiments of this work employ the native sparse function of MATLAB v. 2020a. Furthermore, analyses are performed with alternatives proposed by other authors in MEX functions.

MATLAB's native sparse function produces a sparse matrix from the triplets through the following logic: a pair of indexes (k) and C(k) defines a V(k) value. Common terms (collisions) are added together, and the final size of the matrix corresponds only to the space allocated to the non-zero terms of these ++goooperations. This step is indicated in Eq. (22)

$$KG = sparse(L, C, V, ndof, ndof); \tag{22}$$

where ndof is the total degree of freedom of the problem.

In large scale problems, computation of the vectors L, C and V indicated in Eq. (22) may be unfeasible, with the occurrence of insufficient VRAM memory in the GPU (unlike in RAM). Even the storage of the triplets in the GPU does not guarantee success in assembly, which may result in insufficient memory allocation and error in building the sparse array in GPU, even though the final size of the global array is smaller than the VRAM memory. In this case, two solutions may apply:

- The construction of the sparse array in CPU, and later sending it to the GPU with the gpuArray command. Here is an important consideration: in the CPU the performance of the sparse command is now compromised with the use of swap memory. This occurs as triplets increase relative to RAM. The performance is also lower than with the accelerator card;
- The assembly is divided in the GPU, with the progressive assembly of smaller arrays, obtained by the partial construction of triplets in GPU, which use only a fraction of these vectors, with adequate size for allocation in VRAM, and which can be allocated in the device. This can be obtained through repetition structures, which clean the variables already used in the previous stages and allow new allocation in the GPU. The same fractioning considerations described for the GPU also apply to the case of CPU construction, in limited RAM scenarios.

According to the considerations above, any efficient strategy use of the sparse command must observe the following criteria: RAM, VRAM memory, integer numerical format for the L and C vectors and possible symmetry of the element stifffnessglo matrix.

4.5 - Iterative solver using the conjugate gradient

An alternative to the direct solver appears with iterative methods, such as the conjugate gradient method. In this case, the user experience is crucial for defining the convergence and accuracy of the results. Memory consumption is often less than necessary for using direct solvers. However, the convergence process can be slow, depending on the starting vector for finding solutions. In the transient phenomenon, this can be minimized using the previous time solution as the starting point for the next solution. For a few iterations, the method presents excellent performance. The possibility of GPU acceleration is also an advantage of iterative solvers since they require small memory requirements and make it possible to allocate in accelerator cards. Among all MATLAB functions, the pcg function was used in both CPU and GPU. No type of pre-conditioner was used.

The Eigen library solvers are also explored in the CPU as an alternative to the MATLAB solvers. The interface of this library with MATLAB is made through the MEX language. The library also allows shared memory parallel computing with the Open Multi-Processing (OpenMP) structure. The pcg function was used with and without parallelization. The global stiffness matrix assembly occurs internally and it is enough to supply the triplets: line, column, and values.

5 - RESULTS AND DISCUSSIONS

To evaluate the performance of the developed code, a comparison of the solutions obtained by the developed code with a reference solution was performed. In all simulations MATLAB version 9.8 (R2020a) was used on an Intel(R) Core (TM) i7-8700 CPU 3.19 GHz, with 32 GB of RAM, 6 processors and an NVIDIA TITAN Xp GPU with 12 GB of VRAM memory.

5.1 - Validation Exercises

The problem addressed, for the validation of the developed code, was presented by (Dean et al. 2006), who studied coupling techniques between the flow and mechanical problem and its effects on reservoir fluid production, presented in Figure 7. The reservoir region (R2) is located in the central region of the model, where depletion occurs through a vertical well. The adjacent rocks represent the region R1, as illustrated in Figure 7. Table I presents the geometric properties of the problem.

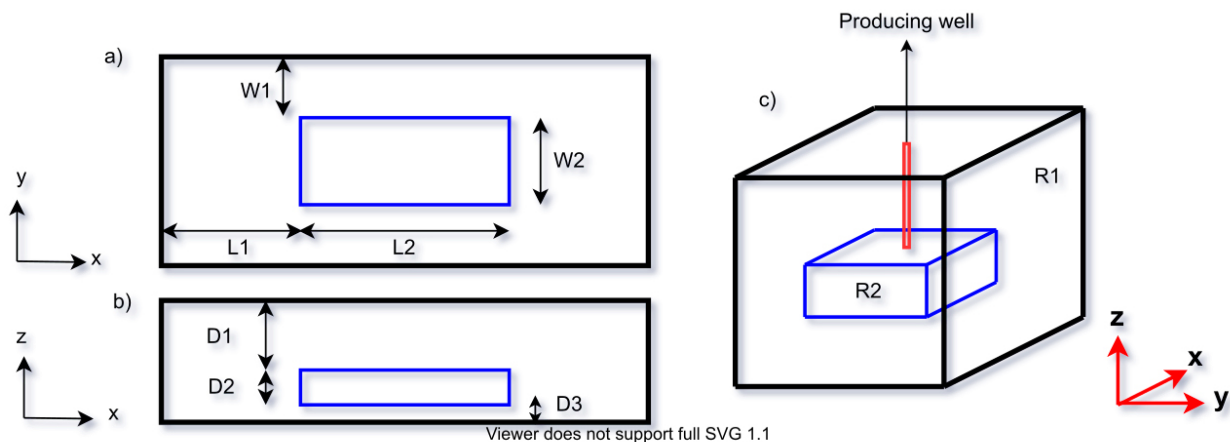

Figure 7. Geometry of problem 3 (Dean et al. 2006). (a) topo view (b) front view and (c) 3d view.

The boundary conditions of the problem are arranged as follows: on the lateral and base faces of the model, the nodal displacement rates are zero. On the top face, vertical stress of zero (0 MPa) is imposed so that the vertical stress gradient is approximately 0.0224 MPa/m. This means that the model initially a uniaxial stress behaviour. The Young modulus for adjacent rocks and reservoir are 6,895 and 68.95 MPa, respectively. For all rocks, the Poisson ratio of 0.25 was admitted. Zero flow is imposed on the entire contour of the problem. A flow rate of 0.092 m^3/s is imposed on the producing well located in the central region of the reservoir and kept constant throughout the simulation time (4,000 days). The finite element mesh employed has 31,752 elements and 6,292 nodes and will be defined ”0 mesh” from now on.

In Figure 8, the mean pressure distribution in the reservoir and the displacement of the central point at the top of the reservoir over the simulation time are presented. The results show a good agreement with the results obtained by Dean et al. (2006).

Table I. Geometric parameters of the model, dimensions are in meters.

<table>
<tr><th colspan="4">Lengths of regions in meters</th></tr>
<tr><td>Lenght</td><td>L1=6098</td><td>L2=6705,8</td><td>-</td></tr>
<tr><td>Width</td><td>W1=3048</td><td>W2=3352</td><td>-</td></tr>
<tr><td>Depth</td><td>D1=3048</td><td>D2=3048</td><td>D3=61</td></tr>
<tr><th colspan="4">Total dimensions</th></tr>
<tr><td>Directions</td><td colspan="2">R1</td><td>R2</td></tr>
<tr><td>x</td><td colspan="2">2*L1+L2=18897,8</td><td>L2=6705,8</td></tr>
<tr><td>y</td><td colspan="2">2*W1+W2=9448,8</td><td>W2=3352</td></tr>
<tr><td>z</td><td colspan="2">D1+D2+D3=3185,2</td><td>D2=76,2</td></tr>
</table>

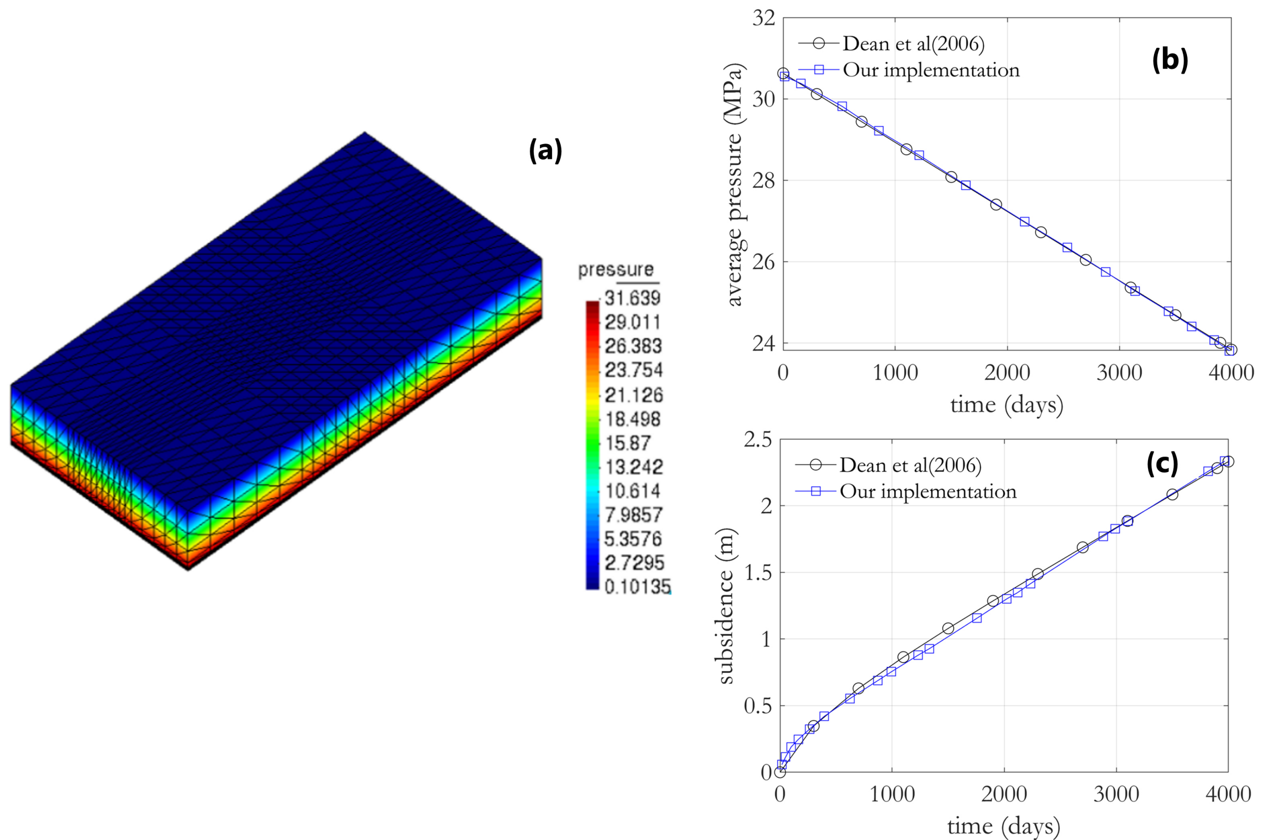

Figure 8. **(a) Pressure profile in the reservoir, (b) mean pressure (c) compaction of the reservoir.**

5.2 - Study of efficiency of the methodologies

In order to analyze the computational efficiency of the code developed for the geomechanical problem, different levels of refinement were used, as shown in Table II, and mesh 0 presents the same degree of refinement used by (Dean et al. 2006) and was used in the validation stage. The computer

setup was able to assemble a KG global stiffness matrix with 14,709,888 elements and solve a linear equation system with 8,003,151 unknowns due to the exhaustion of the RAM memory that can be seen in Figures 10b and 12b added together. For the resolution of the system using the conjugated gradient a tolerance of tol=1e-04 and maximum iteration of maxit=1e05 was used in both CPU and GPU.

Table II. Different meshes analyzed.

meshes	Problem configuration	
	nelem	**ndof**
0	31752	188976
1	1843200	1010919
2	5171328	2822079
3	8323200	4534959
4	14709888	8003151

Focus is given on evaluating the computational efficiency of the geomechanical problem for meshes 1 to 4. Table III shows a summary of the acronyms with their definition for the techniques used to build the local stiffness matrices Ke both in the CPU and GPU, and Table IV shows the times over which all the evaluations were made.

Table III. Some important definitions.

Acronyms	Definitions
Imp	Mounting Ke Implicitly on CPU
Exp	Mounting Ke Explicitly on CPU
ExpS	Mounting Ke Explicitly on CPU using symmetry
ImpGPU	Mounting Ke Implicitly on GPU
ExpGPUS	Mounting Ke Explicitly on GPU Using Symmetry

Table IV shows the times where each nodal pore pressures of the flow problem were obtained and which are assessed in the mechanical problem.

Table IV. Chosen time steps for calculating the geotechnical problem.

Steps	1	2	3	4	5	6	7	8
Times(days)	13	157	527	853	1214	1630	2155	2534

Steps	9	10	11	12	13	14
Times(days)	2878	3140	3438	3646	3845	3981

5.3 - Calculation of local stiffness matrices Ke

Looking at the construction time graphics of the stiffness Ke matrices and comparing the techniques used, it is clear that the explicit construction on the GPU, taking advantage of the symmetry, is the most efficient technique, as shown in Figure 9a. Besides analyzing the graphics globally, it is also interesting to do local analysis with attention to each technique in the CPU and GPU. The best way to build Ke on the CPU is the ExpS technique, and ExpGPUS is the most computationally efficient technique on the GPU. Imp, ExpS and ExpGPUS are techniques for constructing local stiffness matrices implicit in the CPU, explicit with symmetry in the CPU and explicit with symmetry in the GPU. For this problem, the least efficient technique to build the Ke matrix is Imp Figure 9b.

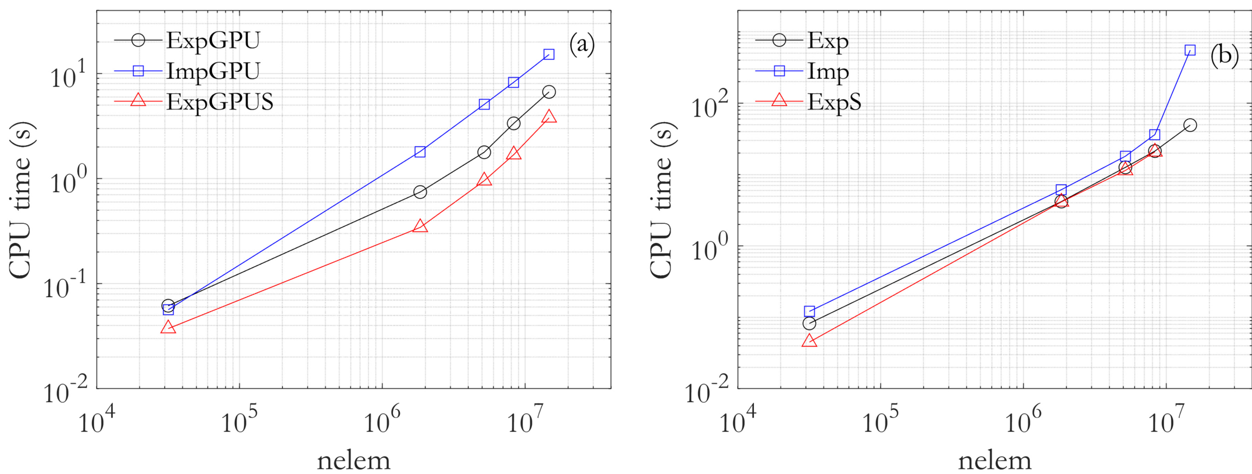

Figure 9. Build time of local stiffness matrices that (a) on the GPU (b) on the CPU.

All the considerations made previously can be seen in the graph of Figure 10a, where the ExpGPUS technique was the most efficient, having a SpeedUp 140 times faster than Imp. Another interesting observation that can be made in Figure 10b is that the memory consumption for the storage of the vectors are very similar both for the calculation of the matrices mounted explicitly and implicitly.

5.3.1 - Assembly of KG global stiffness matrix

Another crucial segment in the study is the assembly of the global stiffness matrices, which several techniques were used, which where previously described in section 3. It can be seen in Figure 11a as follows that the sparseGPUS technique was the most efficient in the GPU. It is worth noting that there was a loop for the assembly of KG on the GPU. In the CPU, the sparse_create had a better performance Figure 11b because it consumes less RAM due to the absence of the triplets (I, J, V) Figure 12b, but which are necessary for the other techniques. On the other hand, it can be seen in Figure 12b on the CPU that the sparse2 strategy had a very expressive consumption of RAM, managed to assemble meshes in the order of 5 million elements, up to mesh 2. The empty cells of Figure 12a are due to the lack of memory.

The performance in Figure 11b can be justified by the memory consumption of the KG assembly shown in Figure 12b together with the memory of the Ke shown in Figure 10b. They clearly show that there was an excessive consumption in the assembly of KG and Ke.

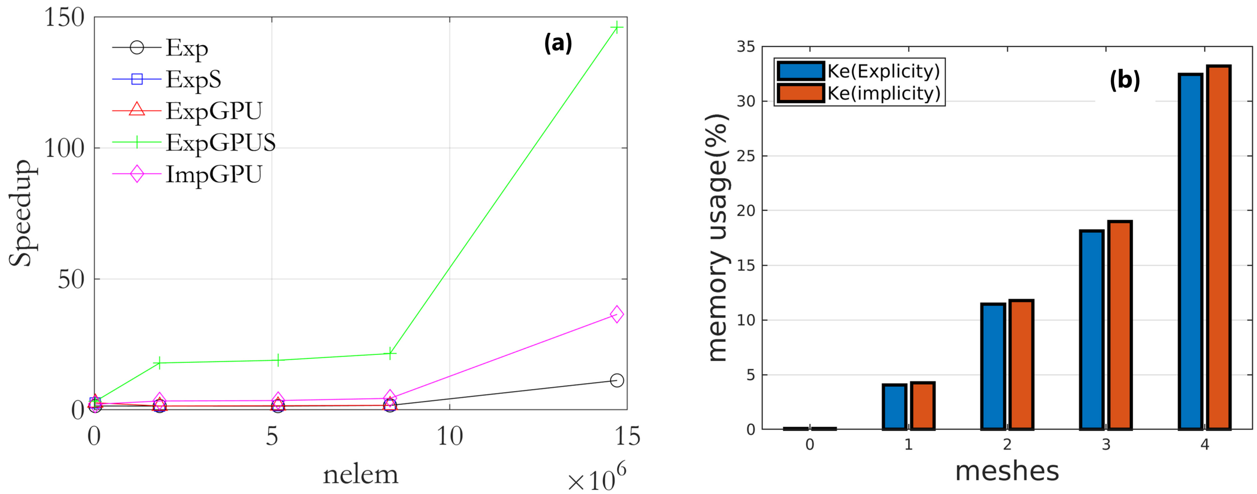

Figure 10. (a) Speedup of the construction of arrays of elements in relation to the Imp technique (b) Comparison of percentage of memory used by Ke assembled explicitly and implicitly.

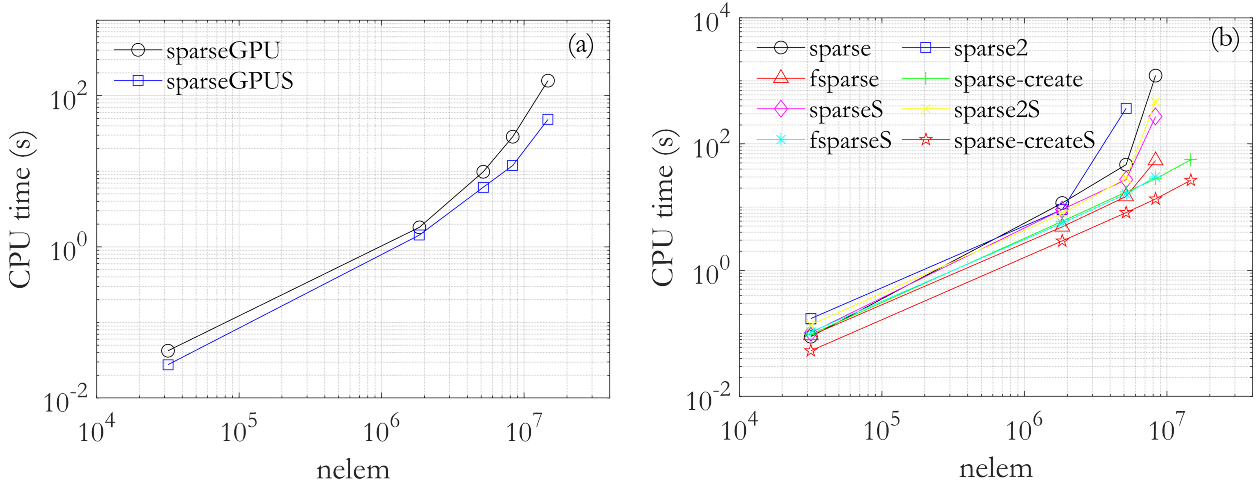

Figure 11. KG global stiffness matrix build time: (a) on GPU and (b) on CPU.

Figure 13a shows the most efficient technique through SpeedUp compared to the CPU sparse technique. It can also be observed that the assembly of the KG on the GPU using the native sparse function of MATLAB is the most efficient. Plot for the sparseGPU and sparse-createS are quite close, which means, for a machine that does not have a GPU equipped architecture, that the sparse_create is an excellent alternative.

Figure 13b shows how excessive memory consumption compromises CPU time.

5.3.2 - Calculation of the solution time of linear of the system in linear equations

A last segment that was the object of analysis is linear equation system solution strategies, a crucial point that needed special attention both in CPU and GPU versions. First, it can be seen in Figure 14a that in the GPU, the best way to solve the problem is to use the symmetry in the assembly of the KG and later the use of the native function of MATLAB pcg with initial displacement vector as a part of

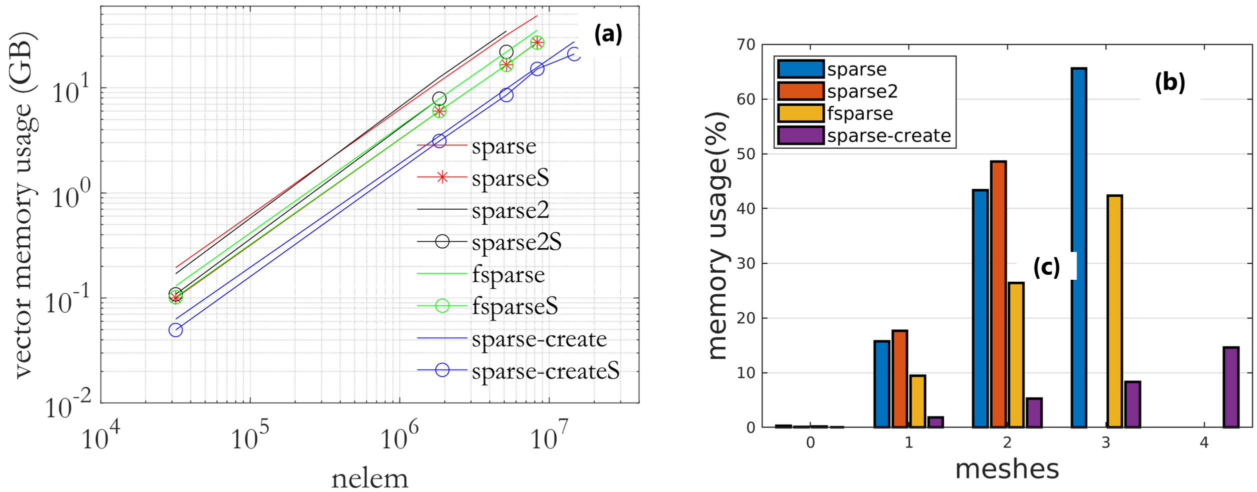

Figure 12. (a) Vector memory vs nelem (b) Percentage of memory consumed by KG versus available memory.

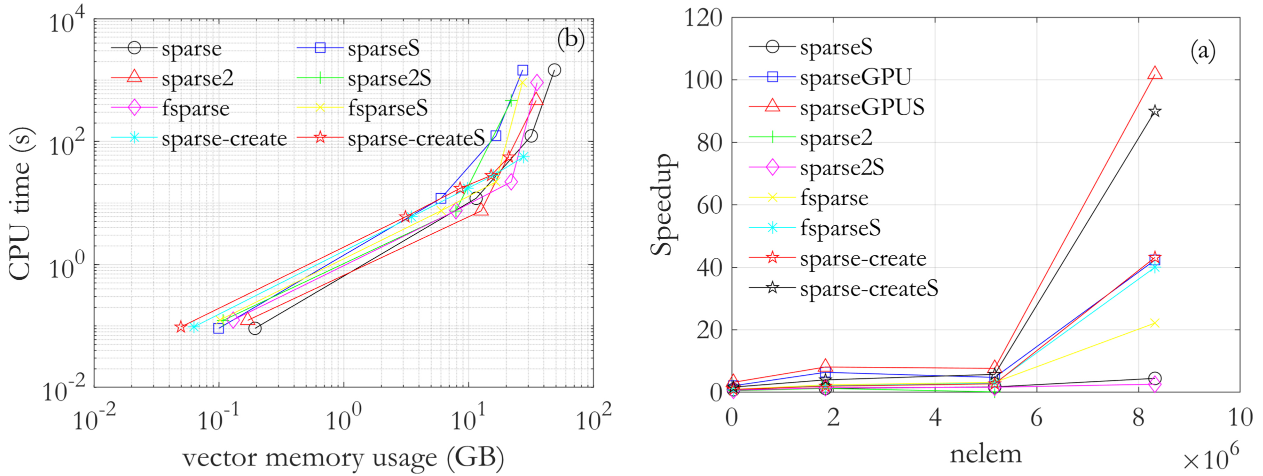

Figure 13. (a) Speedup of global KG array assembly in relation to CPU sparse (b) memory consumption versus CPU time.

solution. It can be seen in Figure 14a that with larger meshes the symmetry in the KG assembly has an influence on the system solution. This observation applies to the system solution in the CPU as well as in the GPU.

In the CPU version, the most efficient technique was pcgEigenOMP. In the CPU version, the most efficient technique was pcgEigenOMP. The solution time presented in the graphics is average, taking into account the 14 times steps presented in Table IV. Figure 15 shows the speedup in the problem for grid 4, presented in Table IV. It can be observed that the PCGGPUS technique had the best performance compared to other techniques. It obtained a speedup of more than 14 times compared to the PCG technique on the CPU.

In the Figure 16 illustrates interesting information about the comparative construction times of Ke and Kg added to the execution time in solution of the problem. To understand Figure 16, four observations can be made:

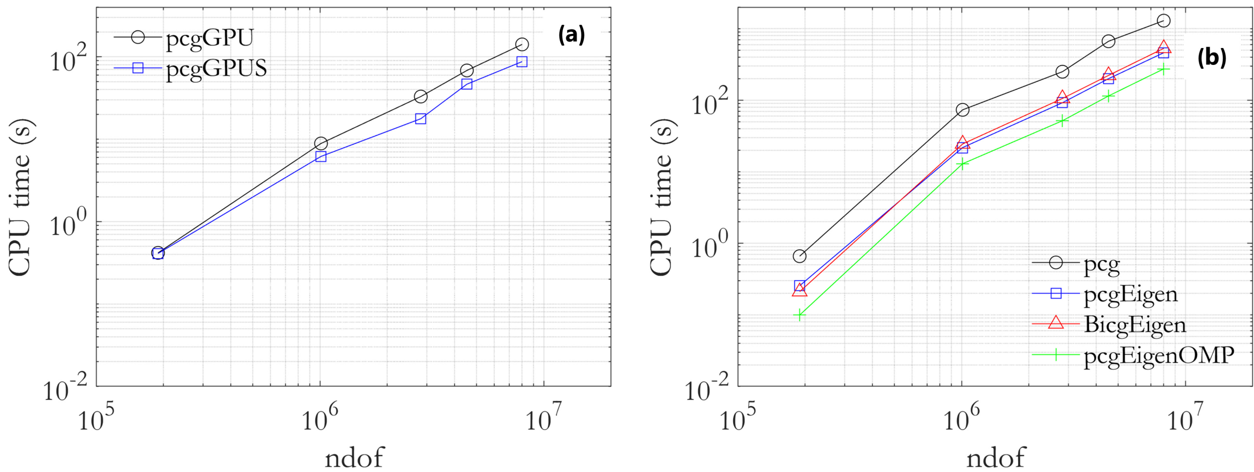

Figure 14. Comparison of the processing times of solvers (a) on the GPU and (b) on the CPU.

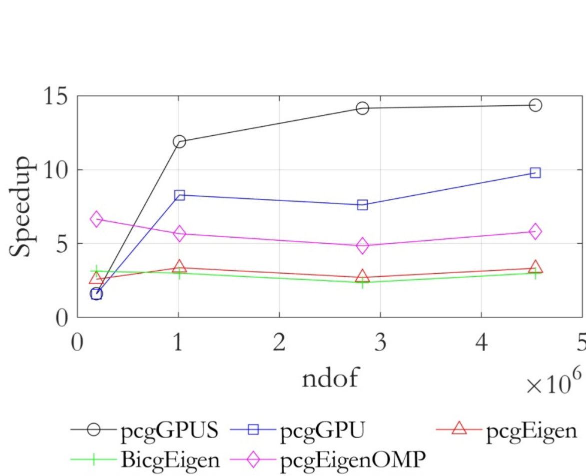

Figure 15. Speedup in relation to solution times using pcg.

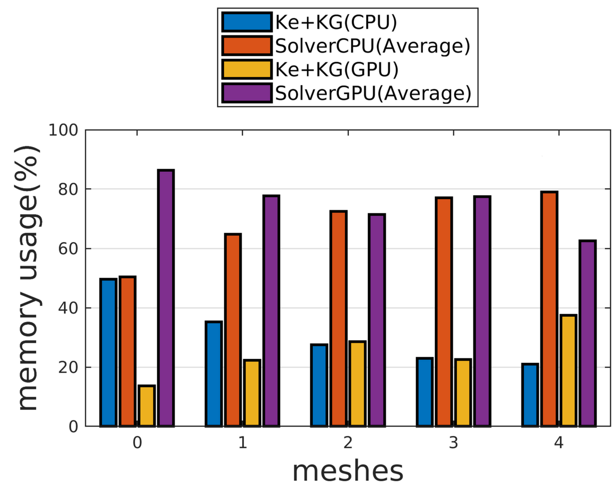

Figure 16. Comparisons in Percentage in the total time of the sum of the Ke and KG times against the solver in the CPU and in the GPU for each mesh.

1. The first one is the first 5 meshes named 0 for mesh 0, 1 for mesh 1 and so on;

2. In the second observation, each mesh is evaluated the participation in the total time of Ke and KG together in front of the solver both in CPU and GPU;

3. The third interesting observation that can be observed is the percentage of the added time of Ke and KG that in the total decreases as the size of the mesh increases. In GPU, this observation is different: with larger meshes, the percentage of time of the solution tends to decrease for a certain size, and the presence of a loop in the construction of KG compromises the efficiency of the assembly, which can justify this behaviour differs from the construction in CPU. For linear elastic problems in the CPU, where there is no need for global loop reconstruction, efficient assembly in record time does not seem to bring great advantages for large-scale meshes. However, for non-linear problems or problems where reconstruction of the Ke and KG arrays

becomes mandatory, the construction time of a Ke and KG in cases of sparse-createS in the CPU and sparseS in the GPU can be interesting in total time;

4. Finally, the fourth observation is in the time steps, this type of problem in which there may be larger or smaller time steps, the percentage of the average solution time will suffer effects in the total time.

6 - CONCLUSIONS

In this article, alternatives for simulation of the geomechanical problem partially coupled to fluid flow, both in the CPU and in the GPU, present in subsurface problems, with Finite Element Method in the elastic and linear analysis, using MATLAB, were presented. Three segments of the code were evaluated in terms of computational efficiency: (i) the calculation of local stiffness matrices Ke, (ii) the assembly of the global stiffness matrix KG and (iii) alternatives for solution of the linear system. In the first part of the analysis, the proposed code was able to calculate in the GPU the stiffness matrices Ke for a mesh of 14,709,888 elements in a time of 3.79 seconds, a technique named *ExpGPUS*, the great exploitation of this technique was the utilization of the symmetry of the matrix Ke for being elastic and linear. The best technique in the CPU is the explicit assembly using the symmetry, the *ExpS* that assembled the same mesh in 49.25 seconds, comparing these two techniques obtained a local speedup in about 13 times and globally *ExpGPUS* had a speedup of more than 140 compared to the assembly in the implicit CPU. Another critical point was the assembly of the global matrix KG. The best technique observed was *sparseGPUS*, which uses the native function of MATLAB sparse on the GPU. The developed geomechanical code was able to assemble this technique in the previous mesh in 48.47 seconds, compared with this same technique. However, on the CPU, our machine was unable to affect the assembly for this amount of elements, due to the comparison effect on the CPU set up a smaller mesh of 8,323,200 with one has much higher that was 1,459.20 seconds. In the CPU, the best technique was using the function sparse_create that obtained 56.24 seconds for the mesh of 14,709,888 elements. The last analyzed segment was the solution time for the different system solution strategies presented previously. The best strategy observed in best computational performance was the gradient conjugated in the GPU with KG mounted using symmetry, known as *pcgGPUS*, that solved this same in 20.40 minutes 14 time steps with an average of 1,457 minutes for each time step. On the CPU, the best technique was *pcgEigenOMP* which achieved a performance of 64.05 minutes with an average of 4,575 minutes for each time step. In this study, the solution of the equation system consumes a good part of the analysis time. An analysis in the GPU for this geomechanical problem using the combination of *KeExpGPUS* to calculate the Ke, s*parseGPUS* to mount KG and pcgGPUS for the solution of the problem. The triplets (*ExpGPUS, sparseGPUS, pcgGPUS*) was the best combination of the analyses made in this article. For a CPU analysis, the best combination was (*ExpS, sparse_create, pcgEigenOMP*). Finally, the total time to solve the problem proposed in this article was critical and a function of some parameters, mainly the mesh size and time steps. In this article, for the mesh with 14,709,888 elements and 8,323,200 degrees of freedom, it was necessary, using the best techniques of each segment in the global, of a total time of 21.27 minutes for the 14 time steps. Graph 16 shows an interesting observation about the participation of the times of the calculation of the local stiffness matrices and the construction of KG added in front of the solution time, it was observed that in the

CPU, for Ke and KG, their participation in the total time decreases the measure the mesh increases and for mesh 4 they consumed approximately together only 21% of the total time, considering an average solution time. In the GPU, the participation of the time of the matrices Ke and KG together is more significant in the total time due to the loop in the construction of KG, representing approximately 37% of the total time necessary to solve the problem. The use of the GPU did not require any knowledge of programming codes in CUDA to obtain all the advantages mentioned above. The article brings the possibility of accelerating geomechanical simulations with modern GPU type devices without programming in low-level languages besides the excellent possibility in CPU with EIGEN integration. This confers to the article innovative characteristics in terms of an overview of several techniques applied to the reservoir simulation problem, going beyond the global matrix construction process and with an effective discussion of strategies that minimize the solution time of this problem. In this context, the current proposal arises, which seeks the efficient solution of one-way simulations in geomechanics of oil reservoirs using programming structures in MATLAB.

Acknowledgments

The author of this article would like to thank the CENPES-PETROBRAS (SIGER - the Petrobras Network on Simulation and Management of Petroleum Reservoirs), the Pernambuco Foundation for the Support of Science and Technology and the Energi Simulation Foundation grants 53/2020 for funding this research.

REFERENCES

ALMEIDA AS, LIMA STC, ROCHA PS, DE ANDRADE AM, BRANCO CC & PINTO AC. 2010. CCGS opportunities in the Santos basin pre-salt development. In: SPE International Conference on Health, Safety and Environment in Oil and Gas Exploration and Production, volume 2, p. 840-849.

ANDERSEN J, REFSGAARD JC & JENSEN KH. 2001. Distributed hydrological modelling of the Senegal River Basin - Model construction and validation. J Hydrol 247(3-4): 200-214.

BEAR J 1988. Dynamics of Fluids in Porous Media. New York: Dover Publications.

BELL JD & ERUTREYA OE. 2015. Estimating Properties of Unconsolidated Petroleum Reservoirs using Image Analysis. J Geol Geophys 5: 1-4.

BINNING P & CELIA MA. 1999. Practical implementation of the fractional flow approach to multi-phase flow simulation. Adv Water Resour 22(5): 461-478.

BIOT MA. 1941. General theory of three-dimensional consolidation. J Appl Phys 12(2): 155-164.

BIRD RE, COOMBS WM & GIANI S. 2017. Fast native-MATLAB stiffness assembly for SIPG linear elasticity. Comput Math Appl 74(12): 3209-3230. Disponível em: http://dx.doi.org/10.1016/j.camwa.2017.08.022.

COUSSY O. 2004. Poromechanics. Chichester: J Wiley & Sons.

CUISIAT F, GUTIERREZ M, LEWIS RW & MASTERS I. 1998. Petroleum reservoir simulation coupling flow and deformation. In: European Petroleum Conference. OnePetro 2: 63-72.

CUVELIER F, JAPHET C & SCARELLA G. 2016. An efficient way to assemble finite element matrices in vector languages. BIT Numer Math 56(3): 833-864.

DABROWSKI M, KROTKIEWSKI M, SCHMID DW. 2008. MILAMIN: MATLAB-based finite element method solver for large problems. Geochem Geophys Geosyst 9(4): 1-24.

DEAN RH, GAI X, STONE CM & MINKOFF SE. 2006. A comparison of techniques for coupling porous flow and geomechanics. SPE J 11(1): 132-140.

DURAN O, SANEI M, DEVLOO PRV & SANTOS EST. 2020. An enhanced sequential fully implicit scheme for reservoir geomechanics. Comput Geosci 24(4): 1557-1587.

ENGBLOM S & LUKARSKI D. 2016. Fast Matlab compatible sparse assembly on multicore computers. Parallel Comput 56: 1-17.

FERRARI A, PIZZOLATO A, PETROSELLI S, MONACO S & OTTAVIANI D. 2021. A consistent framework for coupling modern reservoir flow simulator with mechanics. In: IOP Conference Series: Earth and Environmental Science. Bristol: IOP Publishing, p. 012113.

GASPARINI L ET AL. 2021. Hybrid parallel iterative sparse linear solver framework for reservoir geomechanical and

flow simulation. J Comput Sci. https://doi.org/10.1016/j.jocs.2021.101330.

GOMES AFC, MARINHO JLG & SANTOS JPL. 2019. Numerical Simulation of Drilling Fluid Behavior in Different Depths of an Oil Well. Braz J Pet Gas 13(4): 309-322.

GRIFFITHS DV, HUANG J & SCHIERMEYER RP. 2009. Elastic stiffness of straight-sided triangular finite elements by analytical and numerical integration. Commun Numer Methods Eng 25(3): 247-162. DOI: https://doi.org/10.1002/cnm.1124.

INOUE N & FONTOURAS ABDA. 2009. Explicit Coupling Between Flow and Geomechanical Simulators. In: International Conference on Computational Methods for Coupled Problems in Science and Engineering, p. 2-5.

KEYES DE ET AL. 2013. Multiphysics simulations: Challenges and opportunities. Int J High Perform Comput Appl 27(1): 4-83.

KIM J, TCHELEPI HA & JUANES R. 2011. Stability, accuracy, and efficiency of sequential methods for coupled flow and geomechanics. SPE J 16(2): 249-262.

LI X, ZHANG Y, WANG X & GE W ET AL. 2013. GPU-based numerical simulation of multi-phase flow in porous media using multiple-relaxation-time lattice Boltzmann method. Chem Eng Sci 102: 209-219. Disponível em: https://dx.doi.org/10.1016/j.ces.2013.06.037.

LYUPA AA, MOROZOV DN, TRAPEZNIKOVA MA, CHETVERUSHKIN, CHURBANOVA NG & LEMESHEVSKY SV. 2016. Simulation of Oil Recovery Processes with the Employment of High-Performance Computing Systems. Math Models Comput Simul 8(2): 129-134.

MANDAL P, SAROUT J & REZAEE R. 2020. Geomechanical appraisal and prospectivity analysis of the Goldwyer shale accounting for stress variation and formation anisotropy. Int J Rock Mech Min Sci. 135: 104513. 10.1016/j.ijrmms.2020.104513.

MATHWORKS. 2021. Disponível em: https://www.mathworks.com/products/matlab.html. Acesso em: 7 maio 2021.

MENA H, PFURTSCHELLER LM & STILLFJORD T. 2020. GPU acceleration of splitting schemes applied to differential matrix equations. Numer Algorithms 83(1): 395-419.

NAGEL NB. 2001. Compaction and subsidence issues within the petroleum industry: From Wilmington to Ekofisk and beyond. Physics and Chemistry of the Earth, Part A: Solid Earth and Geodesy 26(1-2): 3-14.

PEREIRA LC, GUIMARÃES LJN, HOROWITZ B & SÁNCHEZ M. 2014. Coupled hydro-mechanical fault reactivation analysis incorporating evidence theory for uncertainty quantification. Comput Geotechnics 56: 202-215.

SETTARI A & WALTERS DA. 2001. Advances in coupled geomechanical and reservoir modeling with applications to reservoir compaction. SPE J 6(3): 334-342.

SHIAKOLAS PS, LAWRENCE KL & NAMBIAR RV. 1994. Closed-form expressions for the linear and quadratic strain tetrahedral finite elements. Comput Struct 50(6): 743-747.

SOLTANZADEH H & HAWKES CD. 2009. Assessing fault reactivation tendency within and surrounding porous reservoirs during fluid production or injection. Int J Rock Mech Mining Sci 46(1): 1-7.

TURSA J. 2022. MTIMESX - Fast Matrix Multiply with Multi-Dimensional Support (https://www.mathworks.com/matlabcentral/fileexchange/25977-mtimesx-fast-matrix-multiply-with-multi-dimensional-support), MATLAB Central File Exchange. Retrieved July 31, 2022.

ZAYER R, STEINBERGER M & SEIDEL HP. 2017. Sparse matrix assembly on the GPU through multiplication patterns. In: IEEE High Performance Extreme Computing Conference (HPEC), p. 1-8.

ZHANG F, YIN Z, CHEN Z, MAXWELL, ZHANG L & WU Y. 2020. Fault reactivation and induced seismicity during multistage hydraulic fracturing: Microseismic analysis and geomechanical modeling. SPE J 25(2): 692-711.

ZHENG D, XU H, WANG J, SUN J, ZHAO K, LI C, SHI L & TANG L. 2017. Key evaluation techniques in the process of gas reservoir being converted into underground gas storage. Pet Explor Dev 44(5): 840-849.

ZIENKIEWICZ OC, TAYLOR RL & TAYLOR RL. 2000. The finite element method: solid mechanics. UK: Butterworth-Heinemann.

ZOBACK M & KOHLI A. 2019. Unconventional Reservoir Geomechanics. In: Unconventional Reservoir Geomechanics: Shale Gas, Tight Oil, and Induced Seismicity. Cambridge: Cambridge University Press.

How to cite

JOSEPH JB, VIEIRA RIBEIRO PMV, GUIMARÃES LJN, CHAVES JUNIOR CV & TEIXEIRA JC. 2022. Acceleration strategies for Tridimensional Coupled hydromechanical problems based on CPU and GPU programming in MATLAB. An Acad Bras Cienc 94: e20211024. DOI 10.1590/0001-3765202220211024.

Manuscript received on July 18, 2021;
accepted for publication on November 17, 2021

JEAN B. JOSEPH
https://orcid.org/0000-0002-5625-8487

PAULO MARCELO V. RIBEIRO
https://orcid.org/0000-0002-9261-8953

LEONARDO J.N. GUIMARÃES
https://orcid.org/0000-0001-6803-6024

CICERO VITOR CHAVES JUNIOR
https://orcid.org/0000-0002-1690-4544

JONATHAN DA C. TEIXEIRA
https://orcid.org/0000-0003-4271-1479

Universidade Federal de Pernambuco, Av. Prof. Moraes Rego, 1235, 50670-901 Recife, PE, Brazil

Correspondence to: **Jean Baptiste Joseph**

E-mail: jean.baptiste@ufpe.br

Author contributions

Jean Baptiste Joseph: Software development, Validation, Investigation, Data Curation, Writing - Original Draft, Writing – Review and Editing, Visualization. Paulo Marcelo Vieira Ribeiro: Conceptualization, Methodology, Supervision, Writing - Original Draft, Review and Editing. Leonardo J. N. Guimarães: Conceptualization, Methodology, Supervision, and Review. Cicero Vitor Chaves Junior: Software development, Investigation and Data Curation. Jonathan da Cunha Teixeira: Formal Analysis, Supervision, Writing – Review and Editing.