XL CILAMCE
IBERO-LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING
NOVEMBER 11-14, 2019
Praiamar Natal Hotel & Convention
Natal, RN-BRAZIL

A CUDA ACCELERATED NUMERICAL INTEGRATION OF ELASTOPLASTIC FINITE ELEMENTS RESIDUALS

Natália R. Vilas Boas
Philippe R. B. Devloo
Omar Durán
nataliarvboas@gmail.com
phil@fec.unicamp.com
omar.duran@cepetro.unicamp.br
Laboratory of Computational Mechanics, University of Campinas (LabMeC/UNICAMP)
Rua Josiah Willard Gibbs, 85, 13083-839, Campinas/São Paulo, Brazil

Abstract. Finite Element Method (FEM) is a numerical technique to approximate partial differential equations. It has been widely used to approximate solutions of physical problems in different fields of research. The numerical simulation challenging engineering problems with small error require fine meshes and leads to high computational cost. To overcome this difficulty parallel computing is becoming a mainstream tool. Among the techniques available to improve the performance of this type of computational application is the execution of the algorithm using Graphics Processing Unit (GPU) programming. Although GPU was originally developed for graphics processing, it has been used in the last years as a general purpose machine with high parallelism power through the availability of libraries such as CUDA or OpenGL. The purpose of this research is to develop an efficient algorithm for the evaluation of the finite element residual and Jacobian matrix. We target the particular variational formulation of an elastoplastic problem with associative plasticity but will try to show that the approach can be extended to other fields and problems. The presented strategy for the calculation of the residual and tangent matrix rely on several computational ingredients such as gathering and scattering operations, sparse matrix multiplications, and a parallel coloring procedure for assembly process. The verification of the nonlinear approximated solution includes comparison with regular CPU implementation in terms of numerical results and execution efficiency. For residual computations of elasto-plasticity, the GPU outperforms the CPU by a factor of up to 10 (details of the architecture are given in the paper).

Keywords: Finite Element Assembly, Residual Numerical Integration, Elastoplasticity, CUDA

1 Introduction

The Finite Element Method (FEM) is one of the most important numerical techniques to find approximate solutions of partial differential equations (PDEs). According to Becker [1], this method allows the construction of base functions to approximate the solution of PDEs with a systematic approach. The main idea is that these functions can be defined piecewise over subregions of the domain called finite elements and can be chosen to be simple functions such as low-degree polynomials. Bhavikatti [2] states that although FEM has been previously used in mechanical structures, it is now widely used as a technique for solving complex problems in different fields of engineering: civil, mechanical, nuclear, biomedical, geomechanics, and others. Problems in these fields lead to high computational demand and cost a lot of CPU time and other computer resources.

Most computer codes are written to be executed sequentially: a problem is split into instructions and these instructions are executed one after the other. In these cases, the performance improvement depends on the advance in CPU efficiency: the software can achieve a significant speedup as each new generation of processors is introduced. However, Kirk and Wen-Mei [3] highlight that since 2003 a stagnation of performance improvement of general applications has been noticed due to high energy consumption and heat dissipation that limits the increase of the clock frequency. Therefor the industry offers a new approach: to increase the number of cores inside each processor.

According to Kirk and Wen-Mei [3], this new approach has a huge impact on the software developer community. Zhang and Shen [4] state that parallel computing in high-performance computers has gradually become a mainstream tool for dealing with large and detailed numerical problems in FEM analysis. Many parallel algorithm to finds approximation using FEM were developed in parallel computers. However, they may require a large number of CPUs to achieve high speed.

Graphics Processing Units (GPUs) were originally developed for images and video processing. Due to the market demand for high-quality real-time graphics in computer applications, these processors have undergone a high technological advancement. For example, Kirk and Wen-Mei [3] state that in an electronic gaming application one needs to render scenes at a resolution of 60 frames per second. According to Micikevicius [5], a GPU consists of a set of multiprocessors and each multiprocessor has its own stream processors and shared memory. All multiprocessors of a GPU have access to global device memory and memory latency is hidden if thousands of threads are executed concurrently. The main difference between GPU and CPU is that CPUs may be efficient with a small number of threads per core while GPUs achieve high performance when thousands of threads are executed concurrently.

Because of the technological advancement of GPUs, researchers who wanted to improve the performance of their applications started to explore its use for non-graphical applications. This trend became known as General-Purpose computing on the GPU (GPGPU). Since then, GPU has been used for numeric simulation of problems in fields of science and engineering. Zhang and Shen [4] say that methods that use GPU's powerful computing resource to accelerate finite element analysis have naturally emerged in the last few years. Among the steps of the finite element calculation for the approximation of solutions of boundary value problems, the evaluation of the elementary stiffness matrix and residual vector, as well as the assembly process of the linear system are the most time-consuming processes in terms of both memory and runtime.

Previous studies on implementing finite element computations for elasticity problems obtained the following results. Zhang and Shen [4] implemented a code to approximate elasticity problems in two and three dimensions using FEM with GPU. The authors used a coloring method to perform the assembly of the global operators. The tests were conducted on a platform composed of an Intel Core2 Duo E7400 processor and an NVIDIA Geforce GT 430. A speedup of 7x is achieved for approximation with two-dimensional elements and 10x for tree-dimensional elements. The linear system solution received a speedup of 3.5x and 6x for two and three-dimensional elements, respectively. Mafi [6] used a GPU-based parallel computing approach to perform real-time analysis of soft objects deformation through a non-linear approximation using FEM. The author used a coalesced data structure to compute FEM matrices in GPU. The computation time of the matrices evaluation achieved a speedup of 28x in an

NVIDIA Geforce GTX 470 when compared to a sequential CPU implementation with an Intel Core i7-3770 processor.

In this work a strategy is proposed to compute the global finite element residual and Jacobian matrix of a non-linear elasticity problem. The strategy is applied to approximate an elastoplastic problem through a FEM simulation of a wellbore under internal and external stresses, as well as to compute an initial stress represented by a hydrostatic pre-stress. The proposed scheme consists in pre-computing and storing constant data and perform GPU parallelized operations to evaluate the global operators. The correctness of the nonlinear approximation is verified by comparing the results with an approximation obtained using a Runge-Kutta approximation on a very fine mesh. The GPU's performance is compared with an equivalent CPU code and with a classical assembly through a C++ open-source library for the development of finite element simulations. A Modified Initial Stiffness method is implemented to provide better convergence to the approximation.

The paper is organized as follows. Section 2 presents the problem statement. In section 3 the classical discretization and notation for FEM are presented. Section 4 shows the adopted structure to solve the problem. Section 5 presents a CUDA C++ implementation and GPU resources used in this research. In section 6 the numerical results are described. Finally, in section 7 the conclusions of this research effort are presented.

2 Statement of the problem

Denoting Ω as the domain for the PDE problem in $\mathbb{R}^2$ with boundary $\partial\Omega = \partial\Omega_{\mathrm{D}} \cup \partial\Omega_{\mathrm{N}}$. Where D and N stand for the boundary with Dirichlet and Neumann data, respectively. The governing equations for the elastoplastic deformation consist of three parts: a conservation law; a constitutive equation; and boundary conditions.

$$\begin{aligned} \operatorname{div}(\boldsymbol{\sigma}(x)) &= 0 \;\; x \in \Omega && (1)\\ \mathbf{u}(\mathbf{s}) &= \mathbf{u}_{\mathbf{D}}(\mathbf{s}) \;\; s \in \partial\Omega_D && (2)\\ \boldsymbol{\sigma}(\boldsymbol{s}) \cdot \mathbf{n} &= \mathbf{t}(\mathrm{s}) \; s \in \partial\Omega_N && (3) \end{aligned}$$

where $\boldsymbol{\sigma}$ [MPa] is the Cauchy stress, $\mathbf{u}$ [m] represents the displacement vector, $\mathbf{t}$ [MPa] is the normal traction over $\partial\Omega_N$ and $\mathbf{n}$ is the outward normal.

The stress σ is a function of the elastic part of the deformation tensor.

$$\boldsymbol{\sigma}(\mathbf{u}) = 2\mu\boldsymbol{\varepsilon}^e + \lambda \operatorname{tr}(\boldsymbol{\varepsilon}^e)\mathrm{I} \;\text{ in }\; \Omega. \tag{4}$$

where λ [MPa] and μ [MPa] are the first and second Lamé constants.

Under the assumption of small deformation, the total strain tensor is expressed as:

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla^t\mathbf{u}\right) \tag{5}$$

The decomposition of the strain tensor in elastic and plastic deformation will be defined subsequently.

Elastoplastic constitutive modeling A body that undergoes elastic deformation is characterized by the complete recovery of its undeformed configuration upon removal of the applied loads. In this case the deformation depends only on the load applied to the body. Irreversible deformations occur if a body is subjected to a loading cycle beyond its elastic limit. In this case the deformation will be formally split in elastic and plastic deformations. The elastic limit for the stress is also known as the yield stress [7].

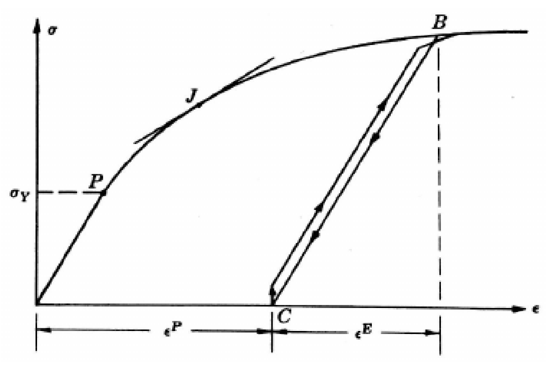

Figure 1. Stress-strain relationship. Extracted from Mase [7].

In Figure 1 the point P corresponds to the yield stress and divides the stress-strain curve into elastic and plastic ranges. Once this point is not always well-defined, it is possible to define it as point J (Johnson's apparent elastic limit), in which the slope of the curve corresponds to 50% of its initial value. Incrementally in the plastic range, unloading from point B results in the state point following the line BC which is parallel to the linear portion of the curve and divides the total deformation rate of the body into plastic ε^p and elastic ε^e.

A large number of engineering materials, such as metals, concrete, rocks, clays, and soils in general, may be modeled as plastic under a wide range of circumstances of practical interest. The study of these materials is described by an incremental stress strain relation. The basic items are:

- Strain tensor increment decomposed into elastic and plastic strain increment;
- Elastic constitutive law;
- Yield criterion;
- Plastic flow rule.

$$
\begin{aligned}
\delta\epsilon &= \delta\epsilon^e + \delta\epsilon^p && (6)\\
\delta\epsilon^p &= \delta\gamma N(\sigma, A) && (7)\\
\delta A &= \delta\gamma H(\sigma, A) && (8)\\
\delta\sigma &= 2\mu\delta\epsilon^e + \lambda \mathrm{tr}(\delta\epsilon^e) I && (9)\\
\delta\gamma \cdot \delta\Phi(\sigma, A) &\equiv 0 && (10)
\end{aligned}
$$

where $\Phi(\sigma, A)$ is the yield surface and $N(\sigma, A) = \frac{\partial\Phi}{\partial\sigma}$ defines the direction of plastic deformation and $H(\sigma, A)$ defines the evolution of the damage variable A.

It should be noted that either $\delta\gamma \equiv 0$ and $\Phi(\sigma, A) < 0$ or $\delta\gamma > 0$ and $\Phi(\sigma, A) \equiv 0$

Mohr-Coulomb plasticity In this research, we use the Mohr-Coulomb criterion as the constitutive relationship for plasticity. This criterion applies to the modeling of materials such as concrete or soil and represents that yielding happens when the relation of the shear stress and normal stress exceeds a given value:

$$\tau = c - \sigma \tan\phi \tag{11}$$

from which c [MPa] is the cohesion and ϕ [°] is the frictional angle. The Mohr-Coulomb has the following definition in principal stresses space:

$$\Phi(\sigma, A) = (\sigma_{max} - \sigma_{min}) + (\sigma_{max} + \sigma_{min}) \sin\phi - 2c\cos\phi. \tag{12}$$

The Mohr Coulomb law is perfectly plastic and does not depend on A. Therefore $\delta A \equiv 0$ and $H(\sigma, A) \equiv 0$.

3 Finite element approximation and notation

Consider a geometrical partition $\mathcal{T}_h = \{\Omega_e\}$ of the region Ω by convex elements Ω_e with boundaries $\partial\Omega_e$. The index h stands for the maximum diameter of the elements Ω_e. The following functional space is required:

$$\mathrm{H}^1(\Omega) = \left\{ v \in L^2(\Omega) : \nabla v \in \mathrm{L}^2(\Omega) \right\}. \tag{13}$$

The classical one-field weak statement for the mechanical problem is defined in section 2 and formulated as:

Find $\mathbf{u} \in \mathbf{V} = \left\{ v \in \mathrm{H}^1(\Omega), v|_{\partial\Omega_D} = 0 \right\}$ such that:

$$\sum_{\Omega_e \in \mathcal{T}_h} \int_{\Omega_e} \boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\mathbf{u})) \cdot \boldsymbol{\varepsilon}(\boldsymbol{v}) \, \mathrm{dV} = \sum_{\Omega_e \in \partial\Omega_N} \int_{\partial\Omega_N} \mathbf{t} \cdot \boldsymbol{v} \mathrm{dS} \;\; \forall \;\; \mathbf{v} \in \mathbf{V}. \tag{14}$$

The discrete version of the weak statement is obtained by considering finite-dimensional space $\mathbf{V}_h \subset \mathbf{V}$ with H^1-conforming finite elements that requires the continuity of functions over the element interfaces. Consequently, the discrete version for the classical one-field weak statement is:

Find $\mathbf{u}_h \in \mathbf{V}_h$ such that:

$$\sum_{\Omega_e \in \mathcal{T}_h} \int_{\Omega_e} \boldsymbol{\sigma}_h(\boldsymbol{\varepsilon}_h(\mathbf{u}_h)) \cdot \boldsymbol{\varepsilon}_h(\boldsymbol{v}_h) \, \mathrm{dV} = \sum_{\Omega_e \in \partial\Omega_N} \int_{\partial\Omega_N} \mathbf{t} \cdot \boldsymbol{v}_h \mathrm{dS} \;\; \forall \;\; \mathbf{v}_h \in \mathbf{V}_h. \tag{15}$$

Residual expression The incremental form of the weak statement above leads to the definition of the following residual expression:

$$\mathbf{R} = \mathbf{R}_\sigma + \mathbf{R}_l \tag{16}$$

where:

$$\mathbf{R}_\sigma = \sum_{\Omega_e \in \mathcal{T}_h} \int_{\Omega_e} \boldsymbol{\sigma}_h(\boldsymbol{\varepsilon}_h(\mathbf{u}_h)) \cdot \boldsymbol{\varepsilon}_h(\boldsymbol{v}_h) \, \mathrm{dV} \;\; \forall \;\; \mathbf{v}_h \in \mathbf{V}_h \tag{17}$$

$$\mathbf{R}_l = -\sum_{\Omega_e \in \partial\Omega_N} \int_{\partial\Omega_N} \mathbf{t} \cdot \boldsymbol{v}_h \mathrm{dS} \;\; \forall \;\; \mathbf{v}_h \in \mathbf{V}_h. \tag{18}$$

Matrix form of the finite element problem

The discrete problem described in Eq. (15) leads to an algebraic problem:

Find $\delta\vec{u} \in \mathbb{R}^{\mathcal{N}}$ such that:

$$\mathbf{K}\delta\vec{u} = -\mathbf{R}, \;\; \mathbf{K} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}, \; \mathbf{R} \in \mathbb{R}^{\mathcal{N}}. \tag{19}$$

The variable $\delta\vec{u}$ represents the finite element increment that will be added to the approximation $\dot{\mathbf{u}}_h$. $\mathcal{N}$ stands for the global number of degrees of freedom. The matrix $\mathbf{K}$ can be either the global jacobian matrix or a secant matrix $\tilde{\mathbf{K}} \approx \mathbf{K}$ and the vector $\mathbf{R}$ represents the global residual, usually numerically integrated or evaluated during a Newton or Quasi-Newton process.

In FEM terminology, the assembly process represents the construction of $\mathbf{K}$ and $\mathbf{R}$ as:

$$\mathbf{K} = \sum_{e=1}^{N_e} \mathbf{K}_e \;\; \text{and} \;\; \mathbf{R} = \sum_{e=1}^{N_e} \mathbf{R}_e. \tag{20}$$

The evaluation of $\mathbf{K}_e$ is given by the integral expression:

$$\mathbf{K}_e = \int_{\Omega_e} \mathbf{B}_e^t \mathbf{D} \mathbf{B}_e = \sum_{k=1}^{np_e} \omega_k |J_k| \mathbf{B}_e^t(\mathbf{x}_k) \mathbf{D}(\mathbf{x}_k) \mathbf{B}_e(\mathbf{x}_k). \tag{21}$$

The evaluation of $\mathbf{R}_e$ is given by the integral expression:

$$\mathbf{R}_e = \int_{\Omega_e} \mathbf{B}_e^t \vec{\sigma}_e = \sum_{k=1}^{np_e} \omega_k |J_k| \mathbf{B}_e^t(\mathbf{x}_k) \vec{\sigma}_e(\mathbf{x}_k). \tag{22}$$

The local stress-strain and strain-displacement relationship are:

$$\vec{\sigma}_e = \mathbf{D}(\mathbf{x}_k) \vec{\varepsilon}_e \tag{23}$$

$$\vec{\varepsilon}_e = \mathbf{B}_e(\mathbf{x}_k) \delta \overrightarrow{u}_e. \tag{24}$$

For the numerical integration the following variables represent:

- $\mathbf{D}(\mathbf{x}_k) \in \mathbb{R}^{n_\sigma \times n_\sigma}$: The local constitutive matrix;
- $\mathbf{B}_e(\mathbf{x}_k) \in \mathbb{R}^{n_\sigma \times \mathcal{N}_e}$: The local strain-displacement matrix;
- $\delta \overrightarrow{u}_e \in \mathbb{R}^{\mathcal{N}_e}$: The degrees of freedom of element e;
- N_e: The number of elements;
- n_σ: The number of fluxes components;
- np: The number of integration points;
- nc: The number of colors;
- $\mathcal{N}_e$: The number of degrees of freedom of element e;
- np_e: The number of integration points of element e;
- ω_k: The integration rule weight at integration point k;
- j_k: The jacobian of the transformation at integration point k;
- $\vec{\sigma}_e$: The strain vector in Voigt notation at integration point k;
- $\vec{\varepsilon}_e$: The strain vector in Voigt notation at the integration point k.

Classical Assembly The most simple and low memory cost algorithm is to serially assemble each element matrix and then sum the contribution of $(\mathbf{K}_e, \mathbf{R}_e)$ to the global matrix using a scatter and add operation into the global matrix and vector $(\mathbf{K}, \mathbf{R})$. Eventually only the load vector $\mathbf{R_e}$ needs to be assembled into $\mathbf{R}$. Algorithm 1 illustrates this assembly process.

Algorithm 1 Classical assembly process.

```
1:  $\mathbf{K} \leftarrow 0^{\mathcal{N}\times\mathcal{N}}$ and $\mathbf{R} \leftarrow 0^{\mathcal{N}}$
2:  for $k \leftarrow 1$ to $N_e$ do
3:      Compute $\mathbf{R}_e = \int_{\Omega_e} \mathbf{B}_e^t \vec{\sigma}_e$
4:      Compute $\mathbf{K}_e = \int_{\Omega_e} \mathbf{B}_e^t \mathbf{D} \mathbf{B}_e$
5:      for $i \leftarrow 1$ to $\mathcal{N}_e$ do
6:          $i_{dest} =$ connectivity$(i, k)$
7:          // Elementary vector gather-scatter
8:          $\mathbf{R}(i_{dest}) \mathrel{+}= \mathbf{R}_e(i)$
9:          for $j \leftarrow 1$ to $\mathcal{N}_e$ do
10:             $j_{dest} =$ connectivity$(j, k)$
11:             // Elementary matrix gather-scatter
12:             $\mathbf{K}(i_{dest}, j_{dest}) \mathrel{+}= \mathbf{K}_e(i, j)$
13:         end for
14:     end for
15: end for
```

This algorithm has the following characteristics:

- Initialize the global matrices $(\mathbf{K}, \mathbf{R})$ with zero;
- Perform the assembly element by element;
- It is general and apply for all matrix structures;
- It performs repetitive access of the global array entries into the matrix structure, for instance a sparse structure. It occurs during the elementary operations of gather and scatter;
- It does not take advantage of the constant data present during the evaluation of Eq. (21) and Eq. (22).
- The algorithm can be executed in parallel by coloring the elements and assemble all the elements belonging to a single color simultaneously. Each color is processed in sequence. This strategy is adopted when the CPU and GPU performance are analyzed in the forthcoming sections.

4 Assembly using integration point contributions

Laouafa and Royis [8] refer to the term *UDA* as an unstructured displacement approach. The basic idea of *UDA* is to perform operations integration point by integration point once an integration rule is defined. For instance, using a Gaussian quadrature method the algebraic problem presented in Eq. (19) can be rewritten as:

Compute the product of the global matrix with $\delta\vec{u} \in \mathbb{R}^{\mathcal{N}}$ as:

$$\mathbf{K}\delta\vec{u} + \mathbf{R}_l = \left(\hat{\mathbf{B}}^t\hat{\mathbf{W}}\hat{\mathbf{D}}\hat{\mathbf{B}}\right)\delta\vec{u} = -\mathbf{R}_\sigma + \mathbf{R}_l,\ \mathbf{R}_\sigma \in \mathbb{R}^{\mathcal{N}},\ \ \mathbf{R}_l \in \mathbb{R}^{\mathcal{N}} \tag{25}$$

where $\hat{\mathbf{B}}$ is the global strain-displacement $N_\sigma \times \mathcal{N}$, it operates over $\delta\vec{u}$:

$$\vec{\varepsilon} = \hat{\mathbf{B}}\delta\vec{u},\ \ \hat{\mathbf{B}} \in \mathbb{R}^{N_\sigma \times \mathcal{N}},\ \ \vec{\varepsilon} \in \mathbb{R}^{N_\sigma}. \tag{26}$$

The matrix $\hat{\mathbf{D}}$ is the constitutive global operator:

$$\vec{\sigma} = \hat{\mathbf{D}}\vec{\varepsilon};\ \ \hat{\mathbf{D}} \in \mathbb{R}^{N_\sigma \times N_\sigma},\ \ \vec{\sigma} \in \mathbb{R}^{N_\sigma} \tag{27}$$

where the term $\mathbf{R}_\sigma$ can be expressed as:

$$\mathbf{R}_\sigma = \hat{\mathbf{B}}^t\hat{\mathbf{W}}\vec{\sigma}. \tag{28}$$

The size N_σ corresponds to the sum $\sum_{e=1}^{N_e} np_e\ n_\sigma$ where np_e is the number of integration points for each element and n_σ is the number of components of the stress tensor in Voigt notation (in two dimensions is equal 3). Some important observations can be made about the expressions in Eq. (25) and Eq. (28):

- Once the geometrical partition and the polynomial order are assigned to every element, the global arrays $\hat{\mathbf{B}}$ and $\hat{\mathbf{W}}$ are constant during the finite element approximation, i.e. only one computation can be done to evaluate them;
- The construction of $\hat{\mathbf{B}}$ involves overlapping information corresponding to common degree of freedom leading to a careful implementation to reach a parallel construction of it.
- The array $\hat{\mathbf{D}}$ can be updated by as a function of ε if required if performing an iterative method to solve a nonlinear elasticity problem Eq. (15);
- The array $\hat{\mathbf{W}}$ contains the weight of the integration point multiplied by the determinant jacobian matrix at each integration point;
- The construction of the operator $\hat{\mathbf{B}}^t\hat{\mathbf{W}}\hat{\mathbf{D}}\hat{\mathbf{B}}$ is implemented in two stages: first at element level $\mathbf{B}_e^t\mathbf{W}_e\mathbf{D}\mathbf{B}_e$ for computing the element residual $\mathbf{R}_e$ and then assembling the element residuals

The adopted path In this research it was adopted an approach different to Laouafa and Royis [8]. Let $\bar{\mathbf{B}} \in \mathbb{R}^{N_\sigma \times \bar{\mathcal{N}}}$ be block-diagonal matrix version of $\hat{\mathbf{B}}$. It is stored $(\sum_{e=1}^{N_e} np_e\ n_\sigma\ \mathcal{N}_e)$ as a global vector indexed per element and subindexed per integration point. The size $\bar{\mathcal{N}}$ is the sum $\sum_{e=1}^{N_e} \mathcal{N}_e$. It contains the values of the partial derivatives for the displacement interpolation functions. With this modification

$\bar{\mathbf{B}}$ can be easily constructed in parallel. Clearly, $\bar{\mathbf{B}}$ is linear with respect to the global $\bar{\delta\vec{u}} \in \mathbb{R}^{\bar{\mathcal{N}}}$ scattered vector version of $\delta\vec{u}$. The array $\hat{\mathbf{I}}$ is a block-diagonal matrix stored ($\sum_{e=1}^{N_e} np_e$) as a global vector indexed per element and subindexed per integration point. The array $\hat{\mathbf{D}}$ is not constructed globally due to reduce the memory being used during the the assembly of $\mathbf{K}$.

$\bar{\mathbf{B}}$ is the global scattered strain-displacement $N_\sigma \times \bar{\mathcal{N}}$, it operates over $\bar{\delta\vec{u}}$:

$$\vec{\varepsilon} = \bar{\mathbf{B}}\bar{\delta\vec{u}}, \quad \bar{\mathbf{B}} \in \mathbb{R}^{N_\sigma \times \bar{\mathcal{N}}}, \quad \vec{\varepsilon} \in \mathbb{R}^{N_\sigma}. \tag{29}$$

The term $\bar{\mathbf{R}}_\sigma \in \mathbb{R}^{\bar{\mathcal{N}}}$ can be expressed as:

$$\bar{\mathbf{R}}_\sigma = \bar{\mathbf{B}}^t \hat{\mathbf{I}} \vec{\sigma}. \tag{30}$$

Computational issues for the adopted strategy The strategy adopted here is separated into two main folds; the residual integration of $\mathbf{R}_\sigma$; and the assembly of $\mathbf{K}$. A concise explanation of them is provided below. The key ingredients for the strategy are listed as:

1. Preprocessing of constant data:
   (a) Parallel construction for $\bar{\mathbf{B}}$ and $\hat{\mathbf{I}}$ and their proper indexation for compute the required matrix operations;
   (b) The global connectivity;
2. Due to the associative characteristic for the elastoplastic equations, the matrix assembly is restricted to a symmetric sparse structure in CSR format or in other terms $\mathbf{K}(K_g, IK_g, JK_g)$;
3. A Greedy-like coloring algorithm to obtain a solution for the minimum sum coloring problem. For an exact solution the reader can be referred to Lecat et al. [9];
4. A set of colors for computing the pairs $(\mathbf{K}_e, \mathbf{R}_e)$ or $\mathbf{R}_e$ in parallel way;
5. Sparse matrix operations;
6. Vector gather, saxpy, and scatter operations;

Residual integration The constitutive Mohr-Coulomb return mapping algorithm is implemented following the directives provided by de Souza Neto et al. [10]. The main difference is that the spectral decomposition is computed by means of an iterative procedure with properly selected initial eigenvalue guesses (See appendix). The multiplication presented in Eq. (29) is performed. Then, the computation of Eq. (27) is performed integration point by integration point in parallel. Formerly, the expression Eq. (30) is performed as global matrix multiplication leading to the residual component $\bar{\mathbf{R}}_\sigma$. Finally, to obtain $\mathbf{R}_\sigma$ a series of gather, saxpy and scatter operations are perform per colored set to reduce $\bar{\mathbf{R}}_\sigma$ into $\mathbf{R}_\sigma$. Because its low cost the linear residual $\mathbf{R}_l$ is assembled once by means of the algorithm 1. Algorithm 2 shows the main steps performed during the numerical integration of $\mathbf{R}_\sigma$.

Matrix assembly The constitutive matrix $\mathbf{D}$ is evaluated during the parallel execution of a set of elements belonging to a color or set of colors to perform $\mathbf{K}_e = \int_{\Omega_e} \mathbf{B}_e^t \mathbf{D} \mathbf{B}_e$ and store them into a global vector $\bar{K}_g \in \mathbb{R}^{\bar{\text{NNZ}}}$ with no overlapping entries. Then, using a loop over the colors, the entries of $\bar{K}_g$ associated to the current color are inserted into the global vector $K_g \in \mathbb{R}^{\text{NNZ}}$ by gathering all the corresponding color entries, apply a vector saxpy over the color entries adding all the contributions to the color local array and finally apply and vector scatter. NNZ stands for the number of nonzeros entries. The size N$\bar{\text{N}}$Z is the sum $\sum_{e=1}^{N_e} \mathcal{N}_e^2$ and it represents the number of non overlapping nonzeros. When the loop over the colors is complete the global $\mathbf{K}$ is assembled.

Algorithm 2 Residual assembly process.

```
1: $\mathbf{R} \leftarrow 0^{\mathcal{N}}$
2: Scatter $\bar{\delta\vec{u}} \leftarrow \delta\vec{u}$
3: Compute $\bar{\vec{\varepsilon}} = \bar{\mathbf{B}}\bar{\delta\vec{u}}$
4: for $k \leftarrow 1$ to $np$ do // Parallel execution
5:     Compute $\vec{\sigma}_e\left(\mathbf{x}_k\right) = \mathbf{D}\left(\mathbf{x}_k\right)\vec{\varepsilon}_e\left(\mathbf{x}_k\right)$
6:     Concatenate $\vec{\sigma} \leftarrow \vec{\sigma}_e$
7: end for
8: Compute $\bar{\mathbf{R}}_\sigma = \bar{\mathbf{B}}^t\hat{\mathbf{I}}\vec{\sigma}$
9: for $c \leftarrow 1$ to $nc$ do // Serial execution
10:     Gather $\mathbf{R}_c \leftarrow$ Color subset $\bar{\mathbf{R}}_\sigma$
11:     Add color contribution $\mathbf{R}_\sigma += \mathbf{R}_c$
12: end for
13: Compute $\mathbf{R} = \mathbf{R}_\sigma + \mathbf{R}_l$
```

Algorithm 3 Matrix assembly process.

```
1: $K_g \leftarrow 0^{\text{NNZ}}$ and $\bar{K}_g \leftarrow 0^{\bar{\text{NNZ}}}$
2: $l \leftarrow 0$
3: for $k \leftarrow 1$ to $N_e$ do // Parallel execution with no overlapping entries
4:     Compute $\mathbf{K}_e = \int_{\Omega_e}\mathbf{B}_e^t\mathbf{D}\mathbf{B}_e$
5:     for $i \leftarrow 1$ to $\mathcal{N}_e$ do
6:         for $j \leftarrow 1$ to $\mathcal{N}_e$ do
7:             // Elementary matrix scatter
8:             $\bar{K}_g\left(l\right) += \mathbf{K}_e(i,j)$
9:             l++;
10:         end for
11:     end for
12: end for
13: for $c \leftarrow 1$ to $nc$ do // Serial execution
14:     Gather $K_c \leftarrow$ Color subset $K_g$
15:     Add the color contribution $K_c +=$ Color subset $\bar{K}_g$
16:     Scatter $K_g \leftarrow K_c$
17: end for
```

5 A CUDA C++ implementation

The constant data presented in section 4 was precomputed through NeoPZ environment, which is a C++ open-source library for the development of finite element simulations. Moreover, the coloring algorithm was also implemented in C++ language. Algorithms 2 and 3 were implemented in C++ and CUDA languages to evaluate performance in both CPU and GPU. Since the result code is written in C++ and CUDA languages, an integration between CUDA and C++ had to be implemented. Such integration was developed as two C++ classes, in which the first is responsible to wrap CUDA API calls to manage memory transfers from CPU to GPU and vice-versa, and the latter is responsible to encapsulate the kernels and CUDA libraries functions calls.

The corresponding operations to the algorithms 2 and 3 are scatter, gather and saxpy operations, sparse matrix-vector multiplication and some kernels implementations. To perform sparse matrix-vector multiplications, gather and scatter operations in the GPU is used the *cuSPARSE* library. According to NVIDIA [11], this library contains a set of basic linear algebra subroutines used for handling sparse matrices. On the other hand, saxpy operations use the *cuBLAS* library. NVIDIA [12] states this library is an implementation of BLAS (Basic Linear Algebra Subprograms) for NVIDIA GPUs. Both *cuSPARSE* and *cuBLAS* are part of NVIDIA GPU-accelerated libraries. According to NVIDIA, these libraries provide highly-optimized functions that perform 2x-10x faster than CPU-only alternatives. Every GPU architecture has its own configuration of number thread blocks. This guarantees an optimum performance the kernels calls through the libraries.

CUDA Kernels The CUDA programming model introduced by NVIDIA supports CPU and GPU execution of an application. Kernels are instructions executed in the GPU by a number of threads. Cecka et al. [13] affirms a warp is a set of 32 threads within a thread block, while a thread block consists of a set of threads running concurrently that communicate through barrier synchronization and shared memory. To achieve the maximum performance of a kernel execution in the GPU, it is desired to maximize the parallel execution, i.e., to maximize the number of active threads in the GPU during the execution of a kernel. However, Mafi [6] highlights three main factors that may limit better performances: registers, shared memory in a thread block and number of threads per thread block.

As seen in algorithm 3, the matrix assembly process consists in computing $\mathbf{K}_e$ in parallel. Thus, the kernel responsible to perform these computations assign each elementary matrix computation to one thread. This kernel is memory-bounded since row and column values of $\mathbf{B}_e$ have to be loaded repeatedly from global memory to compute the result entries. Mafi [6] says these transactions may impact the kernel performance, once the access to global memory is costly. Hence, the use of shared memory leads to greater performance in GPU. However, there is a relationship between the number of thread blocks and the amount of shared memory used by a kernel. More shared memory used by a thread block implies fewer thread blocks available by one kernel calls. For the case of the present kernel, higher polynomial orders imply fewer thread blocks since the dimension of $\mathbf{B}_e$ increases demanding more shared memory.

6 Numerical Results

The numerical results are presented in three subsections organized as follow: Two verifications are presented pointing to check the validity of the spatial approximation properties and elastoplastic process; A series of executions were performed to evaluate the performance in time for CPU and GPU versions of the presented algorithms, and also for the in-house residual integration named as neoPZ; A Quasi-Newton iterative solver of the elastoplastic problem in Eq. (15) is presented and it takes advantage for the fast numerical integration of the residual expression in Eq. (16) and nonlinear acceleration method presented in Sloan et al. [14].

6.1 Verifications

To verify the approximation properties of the implementation, it is considered a linear constitutive behavior with the physical conditions presented in Fig. 2. The elastic and elastoplastic problems associated with a circular domain representing a wellbore region with prescribed Neumann data are approximated. The internal pressure $p_{int} = \boldsymbol{\sigma}\mathbf{n} \cdot \mathbf{n}$ is applied in the wellbore walls, while an external normal stress $\sigma = \boldsymbol{\sigma}\mathbf{n} \cdot \mathbf{n}$ is applied over the external boundary. A hydrostatic pre-stress σ_0 is the initial stress.

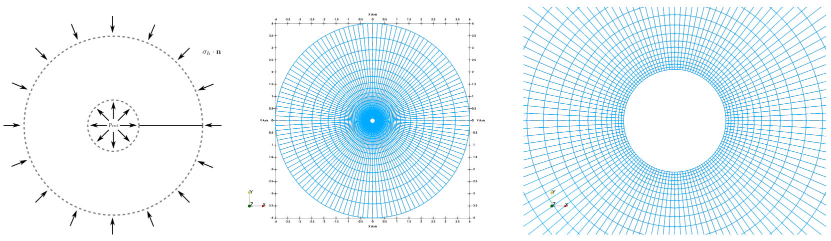

Figure 2. Wellbore region geometry with physical conditions, coarse mesh partition $\mathcal{T}_h|_{l=1}$, and zoom in internal boundary.

The set of geometrical partitions $\mathcal{T}_h$ for different refinement levels are presented in Table 1.

Table 1. Set of geometrical partitions $\mathcal{T}_h$ for different refinement levels.

$\mathcal{N}$	$\mathcal{T}_h\|_{l=1}$	$\mathcal{T}_h\|_{l=2}$	$\mathcal{T}_h\|_{l=3}$	$\mathcal{T}_h\|_{l=4}$	$\mathcal{T}_h\|_{l=5}$
$p = 1$	7936	32256	130048	522240	2093056
$p = 2$	31248	128016	518160	2084880	N/A
$p = 3$	69936	287280	1164336	4687920	N/A
N_e	3844	15876	64516	260100	1044484
h	0.00712436	0.00405445	0.00165619	0.00103663	0.000477711

The material properties used for the numerical results are presented in Table 2.

Table 2. Material properties used for numerical simulations.

Parameter	Symbol [unit]	Value
Internal pressure	p_{int} [MPa]	-40
External stress	σ [MPa]	-49.99375
Hydrostatic stress	σ_0 [MPa]	-50
Internal radius	r_{int} [m]	0.1
External radius	r_{ext} [m]	4.0
Young's modulus	E [MPa]	2000.0
Poisson's ratio	ν	0.2
Cohesion	c [MPa]	5.0
Friction	ϕ [°]	20

Linear setting verification The analytical solution is presented in Coussy [15]. The displacement in Eq. (31), strain in Eq. (32) and stress in Eq. (33) fields are respectively:

$$\mathbf{u} = \frac{(1+\nu)\,(p_{int}-\sigma)}{E}\frac{r_{int}^2}{r}\mathbf{e}_r \tag{31}$$

$$\boldsymbol{\varepsilon} = \frac{(1+\nu)\,(\sigma-p_{int})}{E}\frac{r_{int}^2}{r^2}\left(\mathbf{e}_r\otimes\mathbf{e}_r-\mathbf{e}_\theta\otimes\mathbf{e}_\theta\right) \tag{32}$$

$$\boldsymbol{\sigma} = (\sigma-p_{int})\left(\mathbf{e}_r\otimes\mathbf{e}_r-\mathbf{e}_\theta\otimes\mathbf{e}_\theta\right) \tag{33}$$

The polynomial order and partition selected were $p = \{1, 2\}$ and $\mathcal{T}_h|_{l=1}$ (see Table 1) with several uniform refinements. Ignoring the corresponding elastoplastic data, the parameters being used are presented in Table 2. It was obtained the expected approximation rate in the sense of energy norm is p. Figure 3 documents the verification of the optimal approximation properties for selected finite element discretization.

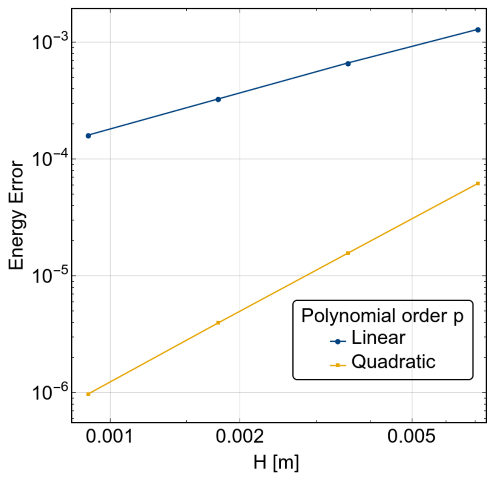

Figure 3. Energy error plots for $p = \{1, 2\}$ with coarse mesh partition $\mathcal{T}_h|_{l=1}$.

Nonlinear setting verification To verify the nonlinear approximation a Runge-Kutta solver was implemented. It is required to simplify the elastoplastic equations and recast the problem described in Eq. (14) as initial value problem:

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \mathbf{f}\left(\mathbf{y}\right) \tag{34}$$

There are several considerations for this case:

- Describe the equations in terms of the Cylindrical coordinate system and its corresponding mixed form.
- The approximation is axisymmetric, leading to a displacement field $\mathbf{u}$ that depends only of the radius and just have a radial component, i.e. $\mathbf{u} = \Phi(r)$.
- The initial value problem is described in terms of one independent variable r and the ODE data is prescribed at the external radius r_{ext}.
- The number of discrete points is large enough to have a reasonable approximation.

Details about the RK method was left apart from this manuscript. The finite element approximation was performed using a quadratic approximation with the geometrical partition $\mathcal{T}_h|_{l=1}$ and the material parameters presented in table 2. Figure 4 shows a remarkable match between the approximations. The number of points for the RK approximation is 2000. The radius where the plasticity ends is approximately $r \approx 0.15$ $[\mathrm{m}]$ for both methods. Thus, Fig. 4 documents the verification for the implementation considering a nonlinear setting.

6.2 Performance analysis

The numerical experiments were conducted in the High-Performance Computing Laboratory and Immersive and Interactive 3D Environment for Scientific Visualization for Petroleum Production cluster (Galileu) at University of Campinas (UNICAMP). The cluster provides two processors Intel® Xeon® E5-2630 v3 @ 2.40 GHz, 64 $[\mathrm{Gb}]$ memory and 8 cores each processor. The graphic processor is an NVIDIA Tesla K40m with 12 $[\mathrm{Gb}]$ global memory and 2880 CUDA cores. It has 3.5 compute capability, 15 multiprocessors and 48 $[\mathrm{Kb}]$ shared memory per thread block.

For each configuration presented in Table 1 were made 5 executions both in CPU and GPU. The code in CPU is parallelized using the Thread Building Blocks, a widely used C++ template library for task parallelism. The comparison between GPU and CPU parallel code is essential to verify if a GPU implementation is necessary. The performance analysis takes into account the execution time of the algorithms presented. The results presented corresponds to the average of the executions made.

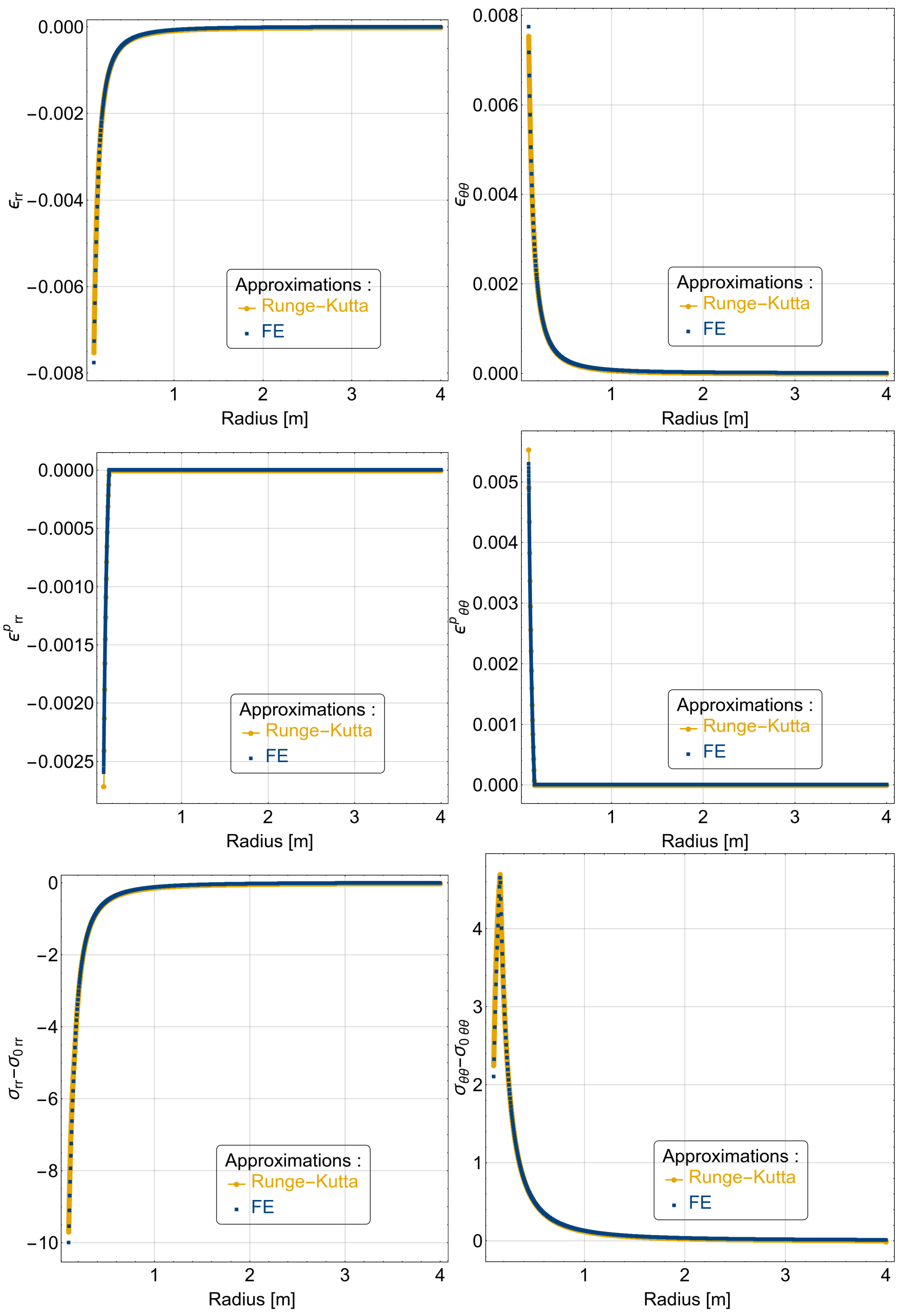

Figure 4. A Runge-Kutta comparison against a finite element approximation with $p = 2$ a mesh partition $\mathcal{T}_h|_{l=1}$.

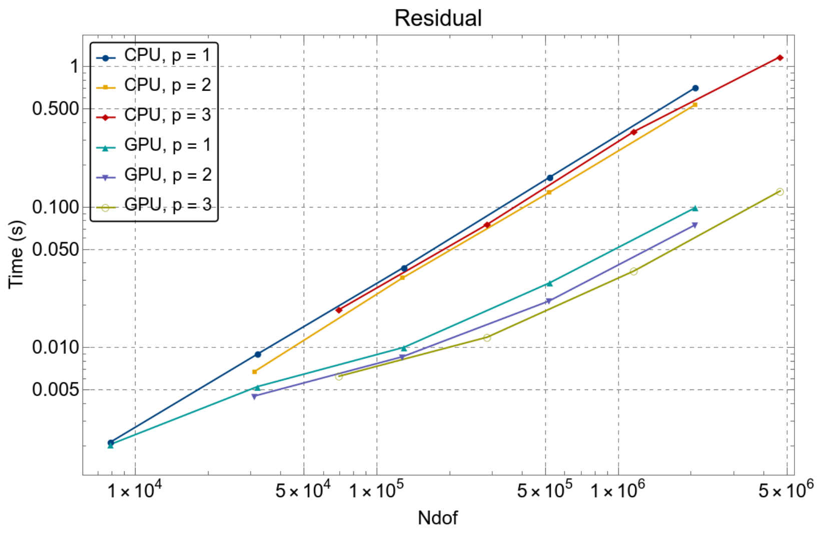

Figure 5. Residual construction performance for CPU and GPU

Figure 5 shows the execution time of the residual construction for both CPU and GPU for the set of configuration presented in Table 1. It is observed that GPU's performance is slightly the same of CPU's for partition $\mathcal{T}_h|_{l=1}$ with linear polynomial order. A speedup of 1.50x and 3x is noticed for quadratic and cubic polynomial orders, respectively. The GPU parallelism becomes evident with the refinement of the mesh. For geometric partition $\mathcal{T}_h|_{l=5}$ and linear polynomial order, GPU's performance reaches a speedup of 7.14x. For geometric partition $\mathcal{T}_h|_{l=4}$ with quadratic and cubic polynomial orders, a speedup of 7.16x and 9.00x is noticed.

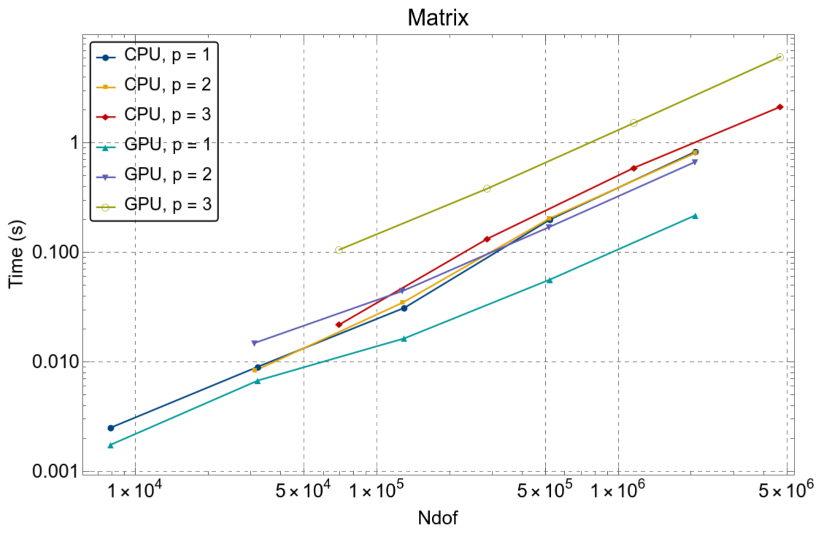

Figure 6. Matrix construction performance for CPU and GPU

Figure 6 shows the execution time of CPU and GPU construction of the matrix for the set of configuration presented in Table 1. It is observed that for linear polynomial order, GPU overcomes CPU by 3.81x for geometric partition $\mathcal{T}_h|_{l=4}$. However, for quadratic polynomial order CPU's and GPU's performances are almost the same. For tests with cubic polynomial order CPU achieved better performance, with a speedup of 2.86x compared with GPU. This is because the amount of shared memory necessary to allocate strain-displacement matrix increases with the rise of polynomial order, leading to less active threads in the calculation of the matrix.

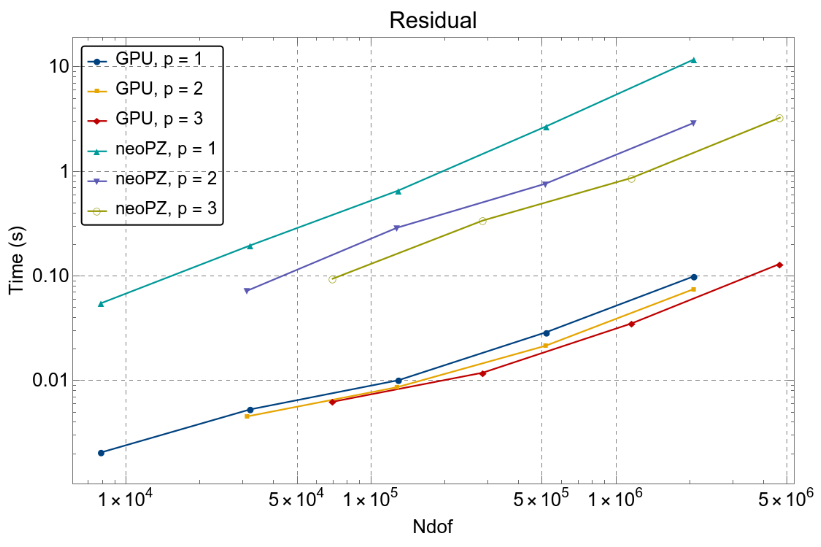

Figure 7. Residual construction performance for GPU and neoPZ

Figure 7 presents the execution time of the residual construction for GPU and the in-house classical residual integration neoPZ. The performance of GPU overcomes the in-house residual integration neoPZ by 26.7x, 15.8x and 15.0x for geometric partition $\mathcal{T}_h|_{l=1}$ with linear, quadratic and cubic polynomial orders, respectively. This speedup increases with the refinement of the finite element mesh, reaching a speedup of 118.44x for linear polynomial order and geometric partition $\mathcal{T}_h|_{l=5}$ and 38.80x and 24.97x speedup for quadratic and cubic polynomial orders and geometric partition $\mathcal{T}_h|_{l=4}$.

6.3 Solving an elastoplastic problem

Seeking for a faster and simple solution of the elastoplastic problem the initial stiffness method IS is adopted with the nonlinear acceleration presented by Sloan et al. [14]. These two methods are summarized as follows.

Initial Stiffness (IS) method the main idea for IS or elastic stiffness method is to construct and approximate the tangent matrix $\tilde{\mathbf{K}} \approx \mathbf{K}$ as the elastic stiffness matrix to compute the iterative solution. The process is performed by a computation of a single $\tilde{\mathbf{K}}$ assembly using algorithm 3 and factorized only once.

The method can be cast into the following steps:

1. Perform a single $\tilde{\mathbf{K}}$ assembly;
2. Decompose $\tilde{\mathbf{K}}$;
3. Compute Newton correction $\delta\vec{u}^{k-1} = -\tilde{\mathbf{K}}^{-1}\mathbf{R}\left(\vec{u}^{k-1}\right)$;
4. Perform a Newton update of $\vec{u}^{k} = \vec{u}^{k-1} + \delta\vec{u}^{k-1}$;
5. Perform 3 to 4, till the residue norm reaches the desired tolerance. i.e. $\left\|\mathbf{R}\left(\vec{u}^{k}\right)\right\| \leq \epsilon$.

On one hand, the main advantage of this strategy is that a single iteration is fast and stable, it is aligned with a fast algorithm for numerical integration of the residual in Eq. (16). On the other hand, the rate of convergence for the algorithm is deficient, especially when the plastic area is considerable extended or arrow. Thus, to accelerate the convergence it is applied the Modified IS method based on the Thomas nonlinear acceleration method presented by Thomas [16] and Sloan et al. [14].

Modified IS method This method use two subsequent states or $\vec{u}^{k}$ and $\mathbf{y}^{k}$. Denoting the Newton update $\mathbf{y}^{k}$:

$$\mathbf{y}^{k} = \vec{u}^{k-1} + \delta\vec{u}^{k-1}. \tag{35}$$

Let be $\delta\vec{u}^{*} = -\tilde{\mathbf{K}}^{-1}\mathbf{R}\left(\mathbf{y}^{k}\right)$ and the new update state defined as follows:

$$\vec{u}^k = \mathbf{y}^k + \omega^{k-1}\, \delta\vec{u}^* \tag{36}$$

where the factor ω is the so-called acceleration factor Sloan et al. [14], defined as:

$$\omega^k = \omega^{k-1} + \frac{\delta\vec{u}^{k-1} \cdot \delta\vec{u}^*}{\delta\vec{u}^{k-1} \cdot \delta\vec{u}^{k-1}}. \tag{37}$$

This method provides better convergence. It requires a single assembly, and two linear solve and two function evaluations per update. Because of the fast numerical integration of Eq. (16), the reasonable augment of the number of residual evaluations does not lead to any large impact on overall computational times. Figure 8 shows the accelerated effect on the convergence against the conventional IS method for partition $\mathcal{T}_h|_{l=3}$, fewer iterations were required to reach the same stop criterion. No kind of instability was observed during the iterative solution process. It is important to remark that by increase the polynomial degree the IS method suffers from a slow of convergence due to the arrow plasticity area around the wellbore region (blue line).

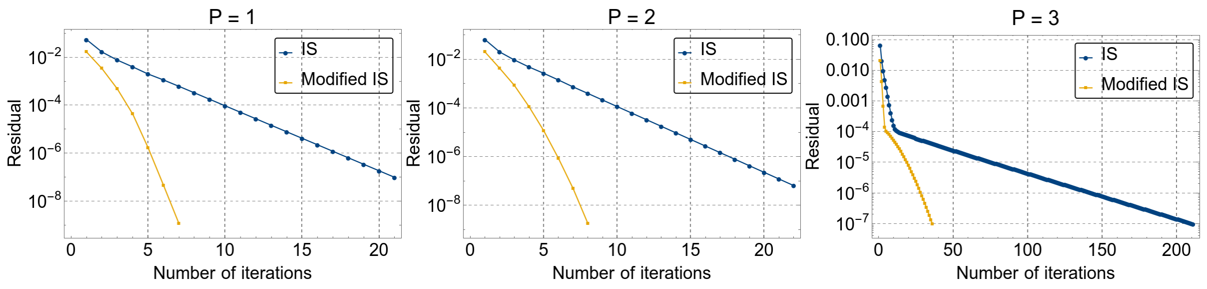

Figure 8. Convergence history for the vertical wellbore problem with $p = \{1, 2, 3\}$.

7 Conclusions

This paper describes an approach for finite element operators assembly in CUDA for elastoplastic problems. This approach differs from the classical assembly process once constant data is computed only once and is stored globally. To verify the implementation of the algorithms, a Runge-Kutta solver was implemented. It was noticed a remarkable match between RK and the FE approximations. The presented structure allows the use of parallelism since the elementary contributions are independent. After the elementary contributions of the tangent matrix and residual are computed in parallel under a coloring scheme, the assembly of the global operators is made using gather and scatter operations. For this research, the residual computation reaches a speedup of 24.9x for a cubic polynomial order refined mesh when compared to the classical assembly approach and 9.00x when compared to an analogous parallel code executed in the CPU. On the other hand, the global tangent construction has good performance when compared with GPU for linear polynomial order, reaching a 3.8x speedup. However, the increase of polynomial order limits the parallelism since more shared memory is necessary to evaluate the elementary matrix. Because of this, a Modified Initial Stiffness method is applied. This method allows the construction of the global matrix only once and takes advantage of the inexpensive residual numerical integration process to achieve the desired convergence in a very efficient manner.

Appendix

Eigenvalues

The first step to evaluate the eigenvalues of the stress tensor in return mapping is the normalization of stress tensor components. After that, the theorem presented by Gerschgorin [17] is used to calculate the interval where the eigenvalues of the stress tensor are. Gershgorin's Theorem states that given a matrix $\mathbf{A}_{nxn}$, all eigenvalues of $\mathbf{A}$ lies in the union of the closed interval:

$$\left[a_{ii} - \sum_{j \neq i} |a_{ij}|\,, a_{ii} + \sum_{j \neq i} |a_{ij}|\right], \quad i, j = 1, ..., n. \tag{38}$$

Thus, it is possible to use Newton's method to compute the maximum and minimum roots of the characteristic polynomial of the stress tensor using the interval evaluated by Gershgorin's theorem as initial guesses. This computation corresponds to the eigenvalues λ_1 and λ_3 of the stress tensor. The intermediate eigenvalue λ_2 is evaluated using the following algebraic relationship:

$$tr(\mathbf{S}) = \lambda_1 + \lambda_2 + \lambda_3 \tag{39}$$

where $\mathbf{S}$ is the stress tensor and λ_i corresponds to its eigenvalues.

Eigenvectors

Given a symmetric matrix with eigenvalues and multiplicity known, the eigenvectors problem is stated by:

$$\mathbf{S} \cdot \vec{v} = \lambda \vec{v} \tag{40}$$

A scheme for eigenvectors calculation is defined using the matrix multiplicity.

Multiplicity 1

If the multiplicity is unitary, one can extract from the matrix $\mathbf{S} - \lambda\mathbf{I}$ a non-singular 2x2 matrix. It is possible to consider three configurations:

$$\begin{bmatrix} x_{11} & x_{12} & y_1 \\ x_{21} & x_{22} & y_2 \\ a & b & c \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_{11} & y_1 & x_{12} \\ a & b & c \\ x_{21} & y_2 & x_{22} \end{bmatrix} \begin{bmatrix} v_1 \\ 1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ y_1 & x_{11} & x_{12} \\ y_2 & x_{21} & x_{22} \end{bmatrix} \begin{bmatrix} 1 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Then it is chosen the configuration for which $|det\ \mathbf{X}|$ is maximum where:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \tag{41}$$

The values v_1 and v_2 are computed as follows:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = -\mathbf{X}^{-1} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{42}$$

Multiplicity 2

In the case multiplicity is 2 the rank of $\mathbf{S} - \lambda\mathbf{I}$ is unitary and there must be at least one diagonal other than zero. It is also possible to consider three configurations:

$$\begin{bmatrix} x_{11} & y_2 & y_2 \\ a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} v_1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ y_1 & x_{11} & y_2 \\ d & e & f \end{bmatrix} \begin{bmatrix} 1 \\ v_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ y_1 & y_2 & x_{11} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ v_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and equivalently:

$$\begin{bmatrix} x_{11} & y_2 & y_2 \\ a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} v_2 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ y_1 & x_{11} & y_2 \\ d & e & f \end{bmatrix} \begin{bmatrix} 0 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ y_1 & y_2 & x_{11} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

It is chosen the configuration for which $|x_{11}|$ is maximum. Then it is defined, for instance:

$$\vec{v_1} = \begin{bmatrix} v_1 \\ 1 \\ 0 \end{bmatrix} \qquad \vec{v_2} = \begin{bmatrix} v_2 \\ 0 \\ 1 \end{bmatrix} \tag{43}$$

Finally, $\vec{v_1}$ and $\vec{v_2}$ are normalized:

$$\tilde{v}_1 = \frac{\vec{v_1}}{||\vec{v_1}||} \qquad \tilde{v}_2 = \frac{\vec{v_2}}{||\vec{v_2}||} \tag{44}$$

Multiplicity 3

In this case the matrix is diagonal and the eigenvectors are the identity matrix.

Acknowledgements

The authors thankfully acknowledge financial support from ANP (Brazilian National Agency of Petroleum, Natural Gas and Biofuels), grant 2014/00090-2.

References

[1] Becker, E., 1981. *Finite elements*. Prentice-Hall, Englewood Cliffs, N.J.

[2] Bhavikatti, S., 2005. *Finite element analysis*. New Age International.

[3] Kirk, D. B. & Wen-Mei, W. H., 2012. *Programming massively parallel processors: a hands-on approach*. Morgan Kaufmann.

[4] Zhang, J. & Shen, D., 2013. GPU-based implementation of finite element method for elasticity using CUDA. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pp. 1003–1008. IEEE.

[5] Micikevicius, P., 2009. 3d finite difference computation on gpus using cuda. In *Proceedings of 2nd workshop on general purpose processing on graphics processing units*, pp. 79–84. ACM.

[6] Mafi, R., 2014. *GPU-based Parallel Computing for Nonlinear Finite Element Deformation Analysis*. PhD thesis.

[7] Mase, G., 1969. *Schaum's Outline of Continuum Mechanics*. McGraw-Hill Education.

[8] Laouafa, F. & Royis, P., 2004. An iterative algorithm for finite element analysis. *Journal of Computational and Applied Mathematics*, vol. 164-165, pp. 469 – 491. Proceedings of the 10th International Congress on Computational and Applied Mathematics.

[9] Lecat, C., Li, C.-M., Lucet, C., & Li, Y., 2015. Exact methods for the minimum sum coloring problem. Doctoral Programming CP.

[10] de Souza Neto, E. A., Peric, D., & Owen, D. R., 2011. *Computational methods for plasticity: theory and applications*. John Wiley & Sons.

[11] NVIDIA, 2018a. cusparse library.

[12] NVIDIA, 2018b. cublas library.

[13] Cecka, C., Lew, A. J., & Darve, E., 2011. Assembly of finite element methods on graphics processors. *International journal for numerical methods in engineering*, vol. 85, n. 5, pp. 640–669.

[14] Sloan, S. W., Sheng, D., & Abbo, A. J., 2000. Accelerated initial stiffness schemes for elastoplasticity. *International Journal for Numerical and Analytical Methods in Geomechanics*, vol. 24, n. 6, pp. 579–599.

[15] Coussy, O., 2004. *Poromechanics*. Wiley.

[16] Thomas, J. N., 1984. An improved accelerated initial stress procedure for elasto-plastic finite element analysis. *International Journal for Numerical and Analytical Methods in Geomechanics*, vol. 8, n. 4, pp. 359–379.

[17] Gerschgorin, S., 1931. Uber die abgrenzung der eigenwerte einer matrix. *Izvestija Akademii Nauk SSSR, Serija Matematika*, vol. 7, n. 3, pp. 749–754.