UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA CIVIL E AMBIENTAL
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA CIVIL

JEAN BAPTISTE JOSEPH

ESTRATEGIAS DE ACELERAÇÃO PARA PROBLEMAS HIDROMECÂNICOS TRIDIMENSIONAIS ACOPLADOS, BASEADOS EM PROGRAMAÇÃO NA GPU E CPU: APLICAÇÃO À GEOMECÂNICA DE RESERVATÓRIOS

Recife
2023

JEAN BAPTISTE JOSEPH JEAN

ESTRATEGIAS DE ACELERAÇÃO PARA PROBLEMAS HIDROMECÂNICOS TRIDIMENSIONAIS ACOPLADOS, BASEADOS EM PROGRAMAÇÃO NA GPU E CPU: APLICAÇÃO À GEOMECÂNICA DE RESERVATÓRIOS

Tese submetida ao corpo docente do Programa de Pós-Graduação em Engenharia Civil da Universidade Federal de Pernambuco como parte integrante dos requisitos necessários ao título de Doutor em Engenharia Civil.

Área de concentração: Geotecnia

Orientador: Prof. Dr. Leonardo José do Nascimento Guimarães.

Coorientador: Prof. Dr. Paulo Marcelo Vieira Ribeiro.

Recife

2023

Catalogação na fonte
Bibliotecário Gabriel Luz, CRB-4 / 2222

J83e Joseph, Jean Baptiste.
Estratégias de aceleração para problemas hidromecânicos tridimensionais acoplados, baseados em programação na GPU e CPU: aplicação a geomecânica de reservatórios / Jean Baptiste Joseph, 2023.
95 f.: il.

Orientador: Prof. Dr. Leonardo José do Nascimento Guimarães.
Coorientador: Prof. Dr. Paulo Marcelo Vieira Ribeiro.
Tese (Doutorado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Civil. Recife, 2023.
Inclui referências.

1. Engenharia civil. 2. Programação de GPU. 3. Eficiência computacional. 4. Elementos finitos. 5. Matrizes de rigidez esparsas. 6. Matriz explícita. 7. Problemas geomecânicos. I. Guimarães, Leonardo José do Nascimento (Orientador). II. Ribeiro, Paulo Marcelo Vieira (Coorientador). III. Título.

624 CDD (22. ed.)

UFPE
BCTG / 2023 - 272

JEAN BAPTISTE JOSEPH

ESTRATÉGIAS DE ACELERAÇÃO PARA PROBLEMAS HIDROMECÂNICOS TRIDIMENSIONAIS ACOPLADOS, BASEADOS EM PROGRAMAÇÃO NA GPU E CPU: APLICAÇÃO A GEOMECÂNICA DE RESERVATÓRIOS

Tese em Engenharia Civil da Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, como requisito para obtenção do título de Doutor em Engenharia Civil, Área de Geotecnia.

Aprovada em 25/09/2023

Orientador: Prof. Dr. Leonardo José do Nascimento Guimarães – UFPE

Coorientador: Prof. Dr. Paulo Marcelo Vieira Ribeiro – UFPE

BANCA EXAMINADORA

participação por videoconferência
Prof. Dr. Igor Fernandes Gomes (examinador interno)
Universidade Federal de Pernambuco

participação por videoconferência
Prof. Dr. Osvaldo Luís Manzoli (examinador externo)
Universidade Estadual Paulista Júlio de Mesquita Filho

participação por videoconferência
Prof. Dr. Michael Andrade Macedo (examinador externo)
Universidade Federal de Uberlândia

participação por videoconferência
Prof. Dr. Manoel Porfírio Cordão Neto (examinador externo)
Universidade de Brasília

participação por videoconferência

Prof. Dr. Lícia Mouta da Costa (examinadora externa)

Universidade Federal de Pernambuco

Aos meus pais e a toda minha família por todo o apoio recebido, meu muito obrigado. Este trabalho é dedicado a vocês.

AGRADECIMENTOS

Através desta tese, desejo expressar minha profunda gratidão a todos que desempenharam um papel fundamental nesta jornada acadêmica de pesquisa. São inúmeras as pessoas e instituições que me acompanharam ao longo desses anos e contribuíram para o sucesso deste projeto. É com imensa satisfação que faço meus agradecimentos:

Gostaria de começar agradecendo aos meus colegas do grupo de pesquisa, LMCG, que me acolheram calorosamente e compartilharam conhecimentos valiosos. Aos meus orientadores, que generosamente partilharam suas experiências e orientações que foram cruciais para o desenvolvimento desta tese.

Gostaria de expressar minha sincera gratidão aos colaboradores da SIM ENGENHARIA GROUP LTDA, a startup da qual sou fundador e diretor executivo. Em particular, desejo destacar a Dra. Nayara Belfort, minha sócia, cujo constante apoio tem sido fundamental nessa nova fase da empresa. Sua dedicação permitiu que eu me concentrasse na produção deste trabalho.

À INURED, Interuniversity Institute for Research and Development, que desempenhou um papel significativo nos meus cinco primeiros anos no Brasil, fornecendo suporte financeiro. Destaco com gratidão o professor Louis Herns Marcelin, seu presidente.

À Universidade Federal de Pernambuco, onde pude realizar meu curso de Engenharia Civil.

Um agradecimento especial aos meus estimados orientadores, professor Leonardo Guimarães e professor Paulo Marcelo, que guiaram minha tese com paciência e conhecimento. O professor Leonardo, em particular, aceitou com dedicação compartilhar seus conhecimentos, pelo qual sou imensamente grato.

À FACEPE, cujo apoio financeiro durante uma boa parte do doutorado demonstrou confiança em minha pesquisa e contribuiu para o êxito deste projeto.

À PETROBRAS e à Fundação *Energi Simulation*, pelo suporte concedido às pesquisas realizadas e ao desenvolvimento desta tese.

Agradeço também às colaboradoras da secretaria do LMCG, Rose e Priscila, por sua paciência em responder às minhas dúvidas administrativas.

Ao LITPEG, o Instituto de Pesquisa em Energia e Gás, no qual faço parte do conselho de gestão e represento os alunos de doutorado.

Aos colegas do sexto andar do LITPEG, com quem compartilhei o espaço de pesquisa e que sempre me apoiaram. Especial agradecimento.

À minha irmã Juliette Joseph e aos meus irmãos Benedic Joseph e Davidson Joseph, pelo carinho e apoio. Aos meus pais, Gislaine e Renold pelas lições de vida e pela compreensão em relação à minha jornada longe de casa para perseguir meu sonho.

Por fim, um agradecimento especial à minha esposa Iracema e minha filha Esther, cujo amor incondicional e apoio incansável foram essenciais para que eu alcançasse este marco em minha carreira acadêmica. Sem o suporte da família, essa jornada não seria possível.

A todos que mencionei e àqueles que, por eventual esquecimento, meu profundo obrigado. Sua contribuição é o alicerce sobre o qual esta tese foi construída.

RESUMO

Esta tese de doutorado aborda a eficiência computacional na simulação de problemas geomecânicos em reservatórios por meio do Método dos Elementos Finitos com milhões graus de liberdade, destacando a implementação de técnicas avançadas de programação em GPU. O estudo concentrou-se na análise de eficiência computacional de três componentes cruciais: o cálculo das matrizes de rigidez locais (Ke), a montagem da matriz de rigidez global (KG) e alternativas de solução do problema mecânico. A eficácia da programação em GPU foi evidenciada pela significativa redução nos tempos de execução, especialmente na GPU, utilizando a técnica ExpGPUS para o cálculo das matrizes Ke. A montagem eficiente da matriz global KG na GPU, empregando a técnica sparseGPUS, também se destacou, proporcionando uma nova abordagem para lidar com problemas com milhões graus de liberdade. A estratégia de solução adotada, particularmente o uso do gradiente conjugado na GPU (pcgGPUS), demonstrou ser altamente eficaz, resultando em tempos totais de resolução notavelmente inferiores em comparação com as técnicas na CPU. A contribuição das matrizes Ke e KG para o tempo total de solução foi analisada, revelando insights valiosos sobre a distribuição do esforço computacional. A aplicação bem-sucedida dessas técnicas em MATLAB, sem a necessidade de conhecimento avançado em programação de GPU, destaca a acessibilidade e a aplicabilidade prática desses métodos em simulações geomecânicas. Os termos como programação de GPU, eficiência computacional, elementos finitos, matrizes de rigidez esparsas, matriz explícita e problemas geomecânicos foram largamente debatidos neste trabalho. Esta tese representa uma contribuição significativa para a otimização de simulações em reservatórios, oferecendo uma visão abrangente e inovadora para enfrentar desafios complexos, integração softwares comerciais, para diminuir o tempo de processamento e melhorar a produtividade.

Palavras-chave: programação de GPU; eficiência computacional; elementos finitos; matrizes de rigidez esparsas; matriz explícita; problemas geomecânicos.

ABSTRACT

This doctoral thesis addresses computational efficiency in the simulation of geomechanical problems in reservoirs using the Finite Element Method with millions of degrees of freedom, highlighting the implementation of advanced GPU programming techniques. The study focused on analyzing the computational efficiency of three crucial components: the calculation of local stiffness matrices (Ke), the assembly of the global stiffness matrix (KG) and alternative solutions to the mechanical problem. The effectiveness of GPU programming was demonstrated by the significant reduction in execution times, especially on the GPU, using the ExpGPUS technique for calculating the Ke matrices. The efficient assembly of the global KG matrix on the GPU, using the sparseGPUS technique, also stood out, providing a new approach to dealing with problems with millions of degrees of freedom. The solution strategy adopted, particularly the use of the conjugate gradient on the GPU (pcgGPUS), proved to be highly effective, resulting in notably lower total solving times compared to CPU techniques. The contribution of the Ke and KG matrices to the total solution time was analyzed, revealing valuable insights into the distribution of computational effort. The successful application of these techniques in MATLAB, without the need for advanced knowledge of GPU programming, highlights the accessibility and practical applicability of these methods in geomechanical simulations. Terms such as GPU programming, computational efficiency, finite elements, sparse stiffness matrices, explicit matrix and geomechanical problems were extensively discussed in this work. This thesis represents a significant contribution to the optimization of reservoir simulations, offering a comprehensive and innovative vision for tackling complex challenges, integrating commercial software, reducing processing time and improving productivity.

Keywords: GPU programming; computational efficiency; finite elements; sparse stiffness matrices; explicit matrix; geomechanical problem.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

símbolos	Definições
$\boldsymbol{\sigma}$	Tensor de tensões totais
ρ	Peso específico do material
$\boldsymbol{g}$	Vetor de gravidade
$\boldsymbol{\sigma'}$	Tensor de tensões efetivas
α	Coeficiente de Biot
$\boldsymbol{p}$	Pressão do fluido nos poros
$\boldsymbol{D}$	Matriz constitutiva do material
ε	Tensor de deformações
$\boldsymbol{u}$	Vetor de deslocamentos
Ω	Domínio do problema
$\boldsymbol{W}$	Funções de ponderação
$\boldsymbol{K}_e$	Matriz de rigidez do elemento
$\boldsymbol{f}_e$	Vetor de forças do elemento
V_e	Volume do elemento
$\boldsymbol{d}$	Vetor de deslocamentos do elemento
$\boldsymbol{K}_g$	Matriz de rigidez global
$\boldsymbol{f}_g$	Vetor de forças global
$\boldsymbol{d}_g$	Vetor de deslocamentos global
$\boldsymbol{v}_f$	Velocidade do fluido
$\boldsymbol{\kappa}$	Matriz de permeabilidade intrínseca do meio
μ	Viscosidade do fluido
C_f	Compressibilidade do fluido
q	Fluxo real
$\overline{q}$	Fluxo prescrito
$\boldsymbol{n}$	Vetor normal a superfície de contorno
t	Tempo
V	Domínio do problema de fluxo

A	Contorno do problema de fluxo
C_e^f	Matriz de armazenamento do elemento
$\boldsymbol{K}_e^f$	Matriz de condutibilidade do elemento
f_e^f	Vetor de forças do elemento
r	Termo de recarga
CPU	Central Processing Unit
GPU	Graphics Processing Unit
MEF	Método dos elementos finitos
RAM	Random Access Memory
VRAM	Video RAM
Nelem	Número de elementos
Ndof	Número de graus de liberdade
$\boldsymbol{B}$	Gradiente das matrizes das funções de Forma
ϕ	Porosidade do meio

SUMÁRIO

1 INTRODUÇÃO

A exploração de hidrocarbonetos em reservatórios em grande profundidade é uma tarefa complexa que envolve alterações substanciais das condições iniciais tanto dos sistemas de rocha quanto dos fluidos envolvidos. Um exemplo é o estudo de Lonardelli et al. (2017), que propuseram um estudo que integra dados de origens e escalas diferentes para obter um modelo geomecânico mais consistente para avaliação da integridade das formações geológicas no entorno de um reservatório. No seu estudo, foram considerados dados como registos de poços, testes de injeção, testes de fuga e histórico de produção, bem como testes experimentais de mecânica das rochas.

Outro fator muito importante para o campo da modelagem e simulação de reservatórios de hidrocarbonetos é a capacidade de processamento de dados necessária para as análises, em geral, multifísicas. Tanto o armazenamento das informações provenientes dos sistemas de monitoramento e das simulações numéricas, quanto a execução, pré e pós-processamento de tais simulações podem chegar a enormes volumes de dados, na casa dos milhares de Gigabytes (10^9 bytes).

Atualmente, com os notáveis avanços no poder de processamento dos computadores, é possível analisar grandes quantidades de dados de maneira bem mais eficiente. Na 60ª edição do *TOP500* do ano 2023, a lista dos supercomputadores mais rápidos do mundo é liderada pelo *Frontier*, localizado no Laboratório Nacional *Oak Ridge* do Departamento de Energia dos EUA. Este impressionante cluster apresenta uma configuração de quase 9 milhões de núcleos de processador, composto por CPUs (*Central Processing Unit*) AMD EPYC 64C de 2 GHz e aceleradores AMD *Instinct* MI250X, resultando em um extraordinário desempenho de 1,102 *exaflops* (10^{18} flops).

Em segundo lugar, encontra-se o *Fugaku*, sediado no Instituto RIKEN no Japão, com sua arquitetura baseada em processadores ARM e oferecendo 442 petaflops de potência de processamento.

A presença do "LUMI" no *European Supercomputing* Centre na Finlândia, utilizando a mesma plataforma HPE Cray EX235a do líder da lista, porém equipado com 1,1 milhão de núcleos de processador, garante a ele o terceiro lugar com uma capacidade de 151 petaflops. Esses avanços refletem o notável progresso na computação de alto desempenho e sua relevância para uma ampla gama de aplicações científicas e tecnológicas.

No brasil, o supercomputador Pégaso, apresentado na Figura 1, desenvolvido pela estatal brasileira voltada para a exploração e produção de petróleo e gás natural, a Petrobras, emerge como um marco extraordinário na paisagem da computação de alto desempenho na América Latina. De acordo com o *TOP500*, Pégaso que ocupou, no ranking da Top500 do ano 2023, a posição 35 do ranking mundial, sendo o mais rápido da América Latina. Vale ressaltar que ele ocupou no ano 2022 o 33º maior supercomputador global.

Com uma capacidade de processamento de 19 *Petaflops* (10^{15} *flops*), equivalente à soma de supercomputadores Dragão e Atlas da própria Petrobras, o Pégaso se destaca como um poderoso aliado na transformação digital da empresa. Com sua rede de 400 gbps, 678 *terabytes* de memória RAM e 2016 GPUs (*Graphics Processing Unit*).

O computador utilizado neste trabalho é equipado com um processador Intel Core i7-12700H de 2,30 GHz e uma capacidade de RAM de 16 GB (sendo utilizável 15,8 GB), ele está operando com um sistema operacional de 64 bits e uma arquitetura x64. Embora sua configuração de hardware seja modesta, quando comparada a sistemas de alta performance, ele oferece um desempenho satisfatório para atividades do dia a dia e tarefas computacionais de menor intensidade.

Figura 1:Supercomputador Pégaso da Petrobras

Fonte: (Petrobras, 2023)

1.1 MOTIVAÇÃO

Neste campo de atuação, a geomecânica desempenha um papel crucial, especialmente no cálculo das tensões nos reservatórios e adjacências e na avaliação da resistência das formações rochosas envolvidas no problema de injeção e extração de fluidos da subsuperfície, permitindo assim a análise de integridade do sistema poço-reservatório-rochas capeadoras. Inicialmente num estado de equilíbrio mecânico, as atividades de produção alteram esse equilíbrio, deformando a rocha e exigindo uma compreensão profunda da geomecânica daquela região afetada pelo empreendimento.

Ao longo de todo o ciclo de exploração e produção, a geomecânica é aplicada para modelar tensões na escala de bacia, garantir a estabilidade de poços, prever a produção de areia, monitorar efeitos do fraturamento hidráulico e assegurar operações seguras (FJAER et al., 2008).

A extração de sal na subsuperfície é outro campo importante em que a geomecânica desempenha um papel muito importante. Conforme (FIRME, 2022) o entendimento dos aspectos geomecânicos em projetos estratégicos relacionados ao sal é fundamental.

A simulação numérica do escoamento em reservatórios porosos também depende da geomecânica para incorporar parâmetros cruciais na estratégia de recuperação de óleo e gás (RODRIGUES, CUNHA, CHALATURNYK, 2007). Mudanças nas tensões podem ocorrer durante processos de depleção ou injeção de água, impactando nas propriedades do reservatório. O comportamento elásto-plástico das rochas é parte fundamental da mecânica de rochas, enfatizando a necessidade de determinar as propriedades do maciço rochoso para garantir a estabilidade do reservatório em desenvolvimento.

Um dos fenômenos-chave que ocorre durante a produção é a redução da pressão nos poros do reservatório em regiões próximas aos poços produtores, o que resulta em um aumento das tensões efetivas atuantes e na subsequente compactação das rochas. Embora a compactação do reservatório possa contribuir para a manutenção da pressão e melhorar a recuperação de hidrocarbonetos (SULAK; DANIELSEN, 1988), ela não está isenta de possíveis repercussões negativas.

Recentemente, muitos autores propuseram o acoplamento de modelos geomecânicos com simulações dinâmicas de fluxo de fluidos (LONGUEMARE et al., 2002, TRANS et al, 2022, LUO et al, 2016). O campo de Belridge, Califórnia, é um

exemplo da relevância que os fenômenos geomecânicos podem ter no comportamento de um reservatório. Nele, aproximadamente 1.000 poços sofreram danos severos nos seus revestimentos ao longo das duas últimas décadas de produção (FREDRICH et al., 2000).

Esses desafios não se limitam apenas à compactação, pois para reservatórios fraturados, as mudanças nas tensões podem afetar a condutividade das fraturas e criar caminhos preferenciais de fluxo (MAINGUY; LONGUEMARE, 2002).

Historicamente, alguns campos importantes enfrentaram problemas substanciais de subsidência. O campo de Wilmington, localizado próximo à bacia sedimentar de Los Angeles, experimentou uma subsidência significativa devido à produção, causando danos financeiros consideráveis, os quais foram mitigados por meio da injeção de água para manter a pressão do reservatório (ALLEN, 1968). No campo de Ekofisk, no Mar do Norte, a depleção do reservatório resultou em um aumento substancial das tensões efetivas, levando a uma subsidência da superfície, causando impactos significativos na infraestrutura (SULAK; DANIELSEN, 1988).

Nos últimos anos, a geomecânica emergiu como uma disciplina crucial no desenvolvimento de reservatórios, sendo aplicada em diversos campos petrolíferos. Isso destaca a importância da modelagem geomecânica para gerenciar e prever o comportamento do reservatório (LONARDELLI et al., 2017; SAMIER; ONAISI; DE GENNARO, 2008).

A subsidência da superfície devido à exploração de águas subterrâneas e hidrocarbonetos tem desencadeado um extenso estudo de simulações acopladas de fluxo e geomecânica, permitindo a modelagem de fenômenos como compactação, subsidência, fraturamento induzido e reativação de falhas, dentre outros aspectos mecânicos (SAMIER; ONAISI; DE GENNARO, 2008).

Além disso, essas simulações considerando os efeitos da geomecânica possibilitam a avaliação das mudanças no estado de tensões durante o desenvolvimento do reservatório, um tópico crucial para determinar a janela operacional para perfuração de novos poços e garantir a integridade dos poços existentes, à medida que as condições *in situ* evoluem ao longo da produção.

No trabalho de Sanei, Duran e P.R. Devloo (2017), foi desenvolvido um método de acoplamento entre o fluxo de fluido e a relação tensão-deformação não linear em meios porosos para investigar a variação da porosidade em comparação com a resposta linear. Por outro lado, (JIANG, e YANG, 2018) propuseram uma abordagem de acoplamento utilizando diferenças finitas miméticas para o escoamento e um modelo geomecânico

baseado em elementos finitos de Galerkin, com o intuito de caracterizar de forma precisa os padrões de produção de reservatórios fraturados de gás de folhelho.

1.2 OBJETIVOS

Nesta seção os objetivos gerais e específicos da tese serão apresentados para uma melhor compreensão do trabalho.

1.2.1 Objetivo Geral

O principal objetivo da tese é desenvolver e propor estratégias eficientes para acelerar simulações computacionais em problemas de larga escala na área da geomecânica de reservatórios, utilizando o programa MATLAB como base, com um computador de mesa acessível.

1.2.2 Objetivos Específicos

- Desenvolver uma caixa de ferramentas utilizando o MATLAB com o propósito de resolver problemas de larga escala na área de Geomecânica de Reservatório;
- Reduzir o tempo necessário para processar as simulações e aumentar a capacidade de escalabilidade dos modelos geomecânicos;
- Comparar várias técnicas de aceleração em problemas geomecânica de reservatório existentes na literatura;
- Comparar os tempos de processamentos de simulação na CPU e GPU;
- Implementar a técnica de acoplamento *oneway* em MATLAB;

1.2.3 Organização dos Capítulos

A tese é dividida em cinco capítulos, cada um detalhando diferentes aspectos da pesquisa. Começa com a "**Introdução**", que apresenta a motivação, os objetivos da pesquisa e a organização da tese.

O Capítulo seguinte, "**Revisão Bibliográfica**", apresenta a base teórica dos principais trabalhos sobre simulação numérica em reservatório de petróleo e métodos de acoplamento, assim como as principais estratégias de aceleração baseadas em CPU e GPU.

No terceiro capítulo, "**Metodologia**", detalha de uma forma sequencial técnicas e estratégias significativas para a programação na GPU, incluindo a utilização do MATLAB e a caixa de ferramentas de computação paralela. Esta seção também se concentra na geomecânica de reservatório e em várias estratégias de vetorização e aceleração. Nesta seção, os principais algoritmos e fluxogramas são apresentados. Por fim os aspectos matemáticos e computacionais são apresentados.

No Capítulo 4, são apresentados os "**Resultados**" da pesquisa, que inclui dois exemplos sobre o uso da GPU e um estudo de caso de geomecânica, seguido de cálculos de matrizes de rigidez e cálculos do tempo de solução do sistema de equações lineares.

O último capítulo, "**Conclusões e sugestões de trabalhos futuros**", apresenta as conclusões do trabalho como também as sugestões para futuros trabalhos a serem desenvolvidos nos próximos anos.

2 REVISÃO BIBLIOGRAFICA

Este capítulo apresentará os tipos de estratégias de acoplamento e os métodos para minimizar o custo computacional em simulações complexas. Além disso, serão examinados em detalhes os conceitos básicos e essenciais que moldam as simulações numéricas de reservatórios com acoplamento geomecânico.

2.1 SIMULAÇÃO NUMÉRICA DE RESERVATÓRIOS COM ACOPLAMENTO GEOMECÂNICO

As operações relacionadas à prospecção e produção de hidrocarbonetos são amplamente influenciadas pelas novas tecnologias devido às suas significativas relevâncias econômicas e os riscos envolvidos. O alto progresso na área da ciência da computação viabilizou a criação de softwares de alta complexidade destinados à simulação de reservatórios. Essas aplicações computacionais encontram uma extensiva aplicabilidade na indústria petrolífera, de modo que praticamente nenhum projeto de exploração e produção é concretizado sem a realização de uma avaliação por meio do emprego de simuladores numéricos.

As múltiplas interações físicas que ocorrem entre o meio poroso e os fluidos que preenchem os espaços vazios são de importante interesse de áreas como a engenharia de reservatórios, biomecânica, engenharia química e ambiental (KEYES et al., 2013; LIU, 2018). Um exemplo são as variações nas pressões e tensões efetivas que ocorrem devido a retirada dos fluidos dos reservatórios de petróleo localizados em grandes profundidades.

As variações da pressão de poros e no estado de tensões efetivas que interagem durante o processo de extração de óleo, demandam intensos estudos nas áreas de fluxo multifásico em meios porosos, (PHILIP, 1999; XIPENG LI, 2013), geomecânica (INOUE, 2009; LAUTENSCHLAGER et al., 2013) como também no âmbito de técnicas computacionais modernas (HERMANN MENA,2019).

Na engenharia de reservatórios de petróleo o acoplamento entre os aspectos físicos que governam o problema hidromecânico, como as deformações do material rochoso induzidas pela variação da pressão dos fluidos no reservatório, apresentam um importante papel na produção e podem resultar na subsidência e compactação do reservatório (NAGEL,2001; DURAN et al., 2020), afetar a integridade das rochas capeadoras (LEI et. al, 2017) e a estabilidade estruturais dos poços (GOMES et al., 2019), e levar à reativação de falhas geológicas (PEREIRA et. al, 2014; ZHANG et. al, 2020).

A importância do conhecimento do comportamento mecânico das rochas cresce continuamente, à medida que reservatórios não-convencionais (ZOBACK, KOHLI, 2019; MANDAL, REZAEE, SAROUT, 2020) e convencionais cada vez mais profundos têm sido detectados e explorados (BELL, ERUTEYA, 2016). Tais fenômenos precisam ser contabilizados para estabelecer de forma mais precisa quais estratégias e métodos deverão ser empregados para otimizar a produção de hidrocarbonetos. Em vista disso, atualmente, profissionais das engenharias, geólogia e hidrogeologia utilizam complexos fluxos de trabalho como ferramenta de tomada de decisão, construindo modelos de reservatórios (ou aquíferos) cada dia mais realísticos.

Entretanto, a combinação de modelos complexos e de grande escala frequentemente dificultam o estudo do comportamento do reservatório pelo alto custo computacional, podendo causar atrasos na definição de projetos de produção, restrição na quantidade de cenários de desenvolvimento ou até mesmo inviabilidade computacional. Tais desafios são constantemente encontrados em problemas de produção e recuperação avançada de hidrocarbonetos em reservatório de petróleo (BELTRAO, 2009), armazenamento geológico de CO2 (MALIK, 2000) e na modelagem hidrogeológica. Todos esses campos de atuação apresentam grandes desafios de simulação computacional numérica devido às suas grandes dimensões.

Vários estudos foram realizados nas áreas de modelagem hidrogeológica (BEAR J,1987; ANDERSON et al., 1992) e na recuperação e produção de hidrocarbonetos em reservatórios de petróleo que fazem o uso de técnicas computacionais modernas aliadas a procedimentos paralelos (LYUPA et al., 2016). Em problemas que envolvem simulações multifísicas acopladas, que frequentemente resultam no aumento do custo computacional, os diversos fenômenos que governam o problema tratado são comumente resolvidos de forma sequencial implícita (KIM, 2010, KIM et al, 2011; KIM et al, 2012). Assim, são desenvolvidos sistemas de equações de diferentes incógnitas primárias, regidas por fenômenos físicos separados. Essas equações têm por incógnitas: pressão de fluido, em meios porosos e saturação/concentração ligados aos problemas de fluxo e transporte de diferentes fluidos/componentes e deslocamentos ligados ao problema mecânico. Tais problemas estão acoplados uns aos outros sobre o mesmo domínio.

Pesquisadores, no contexto global, têm se dedicado a investigar abordagens destinadas a superar as limitações previamente mencionadas. Esses procedimentos, em

sua maioria, podem ser classificados em duas categorias principais. A primeira, conhecida como “Totalmente Acoplada” ou “Solução Implícita”, implica na solução simultânea do conjunto de equações diferenciais que descrevem os problemas de fluxo e mecânico. A segunda abordagem é denominada “Parcialmente Acoplada” ou “Solução Explícita”, e envolve a resolução das equações de fluxo e mecânicas utilizando metodologias distintas para a solução de equações diferenciais. Em outras palavras, são empregados dois métodos numéricos independentes que interagem periodicamente por meio da troca de informações via um programa gerenciador. Este trabalho foi desenvolvido utilizando a segunda abordagem, apenas considerando influência do fluxo no problema mecânico, sem que haja a influência do problema mecânico no problema de fluxo. Este procedimento é conhecido como *oneway*.

Pesquisadores como SETTARI et al, 2001; DEAN et al., 2006; SAMIER et al, 2007; FONTOURA e INOUE, 2009 fizeram a escolha de adotar a abordagem “Parcialmente Acoplada”, onde através da utilização de softwares de simulação de reservatórios preexistentes para resolver o problema simulação de fluxo, geralmente via diferenças finitas, ao mesmo tempo em que se resolve o problema de simulação geomecânica através do Método dos Elementos Finitos. Neste caso, quando os problemas mecânico e hidráulico se influenciam mutuamente, tem-se a abordagem *twoway*. Um exemplo de aplicação desta técnica é o estudo apresentado por Lima, Guimarães e Pereira (2021), que mostram os efeitos geomecânicos relacionados à produção de óleo e gás em um reservatório brasileiro.

Na próxima seção serão abordadas as principais equações que regem este problema acoplado. Foi considerado que o fluxo de fluido é isotérmico e levemente compressível (por uma questão de simplicidade). Sem perda de generalidade, são desprezados os efeitos capilares e considerados que o meio poroso apresenta comportamento elástico-linear e está sujeito a hipótese de pequenas deformações. Tais equações governantes são compostas por um conjunto de leis de conservação e leis constitutivas.

Para resolver essas equações do problema acoplado hidromecânico em reservatórios de petróleo, diversos esquemas de acoplamento têm sido desenvolvidos e vários estudos contribuíram para essa área, como as pesquisas de (PREVOST, 1997;

ANTONIN, SETTARI e MOURITS, 1998; THOMAS, L. CHIN, PIERSON e SYLTE, 2002; TRAN, NGHIEM e BUCHANAN, 2005; JHA e RUBEN JUANES, 2007).

A seguir são apresentadas quatro abordagens muito estudadas para o acoplamento do problema hidromecânico: “Totalmente Acoplado”, “Acoplamento parcial e iterativo”, “Acoplamento Sequencial” e “Acoplamento explícito” (A. SETTARI e WALTERS, 2001; R. H. DEAN, GAI, C. M. STONE0 e MINKOFF, 2006).

As características desses esquemas de acoplamento são as seguintes:

1. Totalmente Acoplado: Neste esquema, as equações governantes de fluxo e deformação são resolvidas simultaneamente, em um único sistema global de equações, em cada passo de tempo. Essa abordagem requer uma implementação cuidadosa, solucionadores lineares sofisticados, pré-condicionadores e é computacionalmente dispendiosa (GUTIERREZ e ROLAND W. LEWIS, 2002; XIKUI LI, Z. LIU e R. W. LEWIS, 2005; JHA e RUBEN JUANES, 2007; PAN, SEPEHRNOORI e L. Y. CHIN, 2009);
2. Acoplamento Sequencial: Nesse esquema sequencial, várias abordagens são desenvolvidas onde os problemas hidráulico e mecânico são resolvidos separadamente e sequencialmente, com as variáveis primárias de um problema mantidas constantes no outro problema, gerando com isso a necessidade de iterar até que a solução atinja a tolerância desejada. Para problemas de reservatórios de petróleo, essa abordagem é geralmente mais eficiente quando comparada com a abordagem totalmente acoplada, tanto para problemas de poromecânica lineares quanto não lineares (F. ARMERO e SIMO, 1992; TRAN, NGHIEM e BUCHANAN, 2005; R. H. DEAN, GAI, C. M. STONE e MINKOFF, 2006; M. F. WHEELER e GAI, 2007; DANA, GANIS e M. F. WHEELER, 2018);
3. Acoplamento Explícito: O esquema de acoplamento explícito é um caso especial do método sequencial, no qual é realizada apenas uma iteração (PARK, 1983; F. ARMERO e SIMO, 1992; F. ARMERO, 1999

2.1.1 Formulação Matemática

As equações de conservação estão relacionadas à quantidade de movimento para o problema mecânico e à conservação de massa para o problema hidráulico. Na escala macroscópica, duas abordagens principais podem ser encontradas dependendo das

descrições euleriana e lagrangiana: a pesquisa pioneira de Terzaghi e Biot (TERZAGHI, 1923; TERZAGHI, 1924; BIOT, 1941), que fornece uma explicação concisa para o caso euleriano; e o trabalho de Coussy (2004) e Lewis e Sheffler (1998), que desenvolvem o caso lagrangiano. Sob a suposição de condição quase-estática, o equilíbrio mecânico pode ser descrito da seguinte forma:

$$\nabla \cdot \sigma + \rho_s \mathbf{g} = \mathbf{0} \tag{2.1}$$

Sendo:

σ = tensor de tensões totais

ρ_s = densidade do material

$\mathbf{g}$ = vetor gravidade.

Em termos do tensor de tensões efetivas, a equação da conservação da quantidade de movimento torna-se:

$$\nabla \cdot \left(\sigma' - \alpha \mathbf{p}\mathbf{I}\right) + \rho_s \mathbf{g} = \mathbf{0} \tag{2.2}$$

sendo:

σ' = tensor de tensões efetivas

$\mathbf{p}$ = pressões do fluido nos poros (em MPa)

α = coeficiente de Biot

$\mathbf{I} \in \mathbb{R}^{3x3}$ = matriz identidade.

O conjunto de equações em (2.1) está sujeita às seguintes condições de contorno:

$$\sigma \cdot \mathbf{n} = \mathbf{t} \qquad in \qquad \Gamma_N^{\mathbf{u}} \tag{2.3}$$

$$\mathbf{u} = \mathbf{u}_0 \qquad in \qquad \Gamma_D^{\mathbf{u}} \tag{2.4}$$

Onde u_0 o vetor de deslocamento inicial e $t\ as\ \ tensções\ nas\ fronteias.$

Além das equações de conservação da quantidade de movimento e das condições de contorno é necessário utilizar equações que relacionam deformações e deslocamentos, conhecidas como relações cinemáticas (LEWIS, SHEFFER, 1998; ZIENKIEWICZ,

TAYLOR, 2000; VERRUIJT, 2013). Considerando a hipótese de pequenas deformações, tal relação é descrita como:

$$\varepsilon(\mathbf{u}) = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla^T\mathbf{u}\right) \tag{2.5}$$

onde $\mathbf{u}$ é o vetor de deslocamentos e ε o tensor de deformações.

Para caracterizar o comportamento do material, utilizamos a equação constitutiva da relação tensão-deformação que relaciona o tensor de tensões efetivas com o tensor de deformações:

$$\sigma' = \mathbf{D}\varepsilon \tag{2.6}$$

sendo

$\mathbf{D} = 2\mu\mathbb{I} + \lambda\mathbf{I}\otimes\mathbf{I}$ é a matriz constitutiva do material e ε o tensor de deformação. Com $\mathbf{I}\otimes\mathbf{I}$ denotando o produto tensorial das matrizes $\mathbf{I}$, , $\mathbb{I} = \mathbf{I} - \frac{1}{3}\mathbf{I}\otimes\mathbf{I}$ e λ e μ as constantes de Lamé (em MPa).

Similarmente a Bear (1972), a conservação de massa para a fase fluida que satura o meio poroso e da fase sólida, são expressas como:

$$\frac{\partial\phi\rho_f}{\partial t} + \nabla\cdot(\rho_f\mathbf{v_f} + \rho_f\phi\dot{\mathbf{u}}) = \omega \tag{2.7}$$

$$\frac{\partial(1-\phi)\rho_s}{\partial t} + \nabla\cdot((1-\phi)\rho_s\dot{\mathbf{u}}) = 0 \tag{2.8}$$

sendo:

ϕ = porosidade.

ρ_f = densidade do fluido.

$\mathbf{v_f} = -\frac{\mathbf{K}}{\mu}(\nabla p - \rho\mathbf{g})$= velocidade do fluido no sólido (cuja relação constitutiva está relacionada com a lei de Darcy).

ω = termo de fonte/sumidouro mássico.

ρ_s = densidade do sólido.

$\dot{\mathbf{u}} = \frac{\partial\mathbf{u}}{\partial t}$ é a velocidade do sólido.

Considerando que a fase fluida é levemente compressível, temos que a relação entre a densidade do ρ_f fluido e a pressão do fluido nos poros p pode ser

escrita com base na compressibilidade do fluido (c_f) como:

$$c_f = \frac{1}{\rho_f}\frac{\partial \rho_f}{\partial p} \tag{2.9}$$

Combinando as equações (2.7) e (2.9), após algumas manipulações algébricas considerando da derivada material com relação à velocidade da fase sólida, chega-se à seguinte equação da conservação de massa para um meio deformável:

$$c_f \phi \frac{\partial p}{\partial t} + \nabla \cdot \mathbf{v_f} + \nabla \cdot \mathbf{u} = f \tag{2.10}$$

sendo f o termo de fonte/sumidouro volumétrico. Desta forma, adotou-se a condição de contorno mista para o fechamento da equação (2.10):

$$\mathbf{v_f} \cdot \mathbf{n} = -\bar{s} + \gamma(p - \bar{p}) \qquad in \qquad \Gamma^p \tag{2.11}$$

sendo $\bar{s}$, γ e $\bar{p}$ a vazão volumétrica imposta (em m³/s), o termo de penalização (m³/s/MPa) e a pressão prescrita, respectivamente.

2.1.2 Formulação Numérica

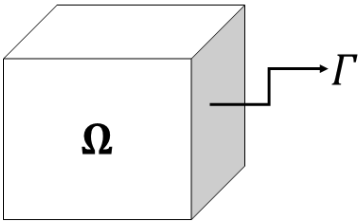
Nesta seção, será apresentado o esquema de resolução numérica da formulação descrita na seção anterior. As derivadas temporais são aproximadas pelo método de Euler implícito. O índice $(\cdot)^{n+1}$ representa os termos avaliados no estado subsequente n + 1 e o $(\cdot)^{n}$ significa termos avaliados no último estado n. Para as variáveis espaciais, uma discretização via elementos finitos clássicos de Galerkin é adotada.

As derivadas de segunda ordem das leis de conservação são reduzidas às derivadas de primeira ordem pelo uso do teorema da divergência após a integração por partes (Formulação Fraca). As aproximações numéricas H^1 serão usadas para os dois conjuntos de equações que modelam a poroelasticidade e a conservação de massa da fase fluida.

Considere o domínio $\Omega \in \mathbb{R}^3$ delimitado pela superfície $\Gamma \in \mathbb{R}^2$ apresentado na Figura 2, sendo Ω particionado em subdomínios ou elementos convexos Ω_e. Os contornos de cada elementos é definido como Γ_e e seu vetor normal unitário exterior ao domínio dado por $\mathbf{n}$. Para a forma fraca do problema poroelástico, $\mathbf{u}_h$ é utilizado como variável de aproximação do campo dos deslocamentos, enquanto p_h é utilizado como

aproximação do campo da pressão para o problema do fluxo de Darcy. Portanto, temos que $\mathbf{u}_h \in \mathbf{V}_h$ e $p_h \in S_h$ onde definimos os espaços funcionais:

Figura 2: O domínio do problema



Fonte: o Autor (2023)

$$\mathbf{V}_h = \{\mathbf{u}_h \in \left[H_h^1(\Omega)\right]^3 : \mathbf{u}_h = \mathbf{u}_0 \quad on\ \Gamma_D^{\mathbf{u}}\} \tag{2.12}$$

$$\mathbf{S}_h = \{p_h \in H_h^1(\Omega)\} \tag{2.13}$$

Com os respectivos espaços funcionais das variações admissíveis:

$$\mathbf{V_0} = \{\mathbf{v} \in \left[H_h^1(\Omega)\right]^3 : \mathbf{v} = \mathbf{0} \quad on\ \Gamma_D^{\mathbf{u}}\} \tag{2.14}$$

$$S_0 = \{q \in L^2(\Omega)\} \tag{2.15}$$

sendo $H^1(\Omega)$, $L^2(\Omega)$ o espaço de Sobolev das funções com primeiras derivadas generalizadas e quadrado-integráveis e o espaço de funções quadrado-integráveis finitas, respectivamente.

Após escolha dos membros dos espaços funcionais, anteriormente definidos, admissíveis como funções de forma e seguindo o método de Galerkin, temos que:

$$\mathbf{u}_h = \sum_{i=1}^{m} \mathbf{N}_i \mathbf{u}_i = \mathbf{N_u}^e \mathbf{u}^e \tag{2.16}$$

$$\mathbf{v} = \sum_{i=1}^{m} \bar{\mathbf{N}}_i \mathbf{v}_i = \bar{\mathbf{N}}_{\mathbf{w}}{}^e \mathbf{v}^e \tag{2.17}$$

$$p_h = \sum_{i=1}^{m} N_i p_i = \mathbf{N}_p^e \mathbf{p}^e \tag{2.18}$$

$$q = \sum_{i=1}^{m} \bar{N}_i q_i = \bar{\mathbf{N}}_p^e \mathbf{q}^e \tag{2.19}$$

sendo m o número de nós de cada elemento (tetraedros = 4); $\mathbf{u}_i$, $\mathbf{w}_i$, q_i e p_i os valores nodais dos deslocamentos, das funções de ponderação, vazões, e pressões do elemento e , respectivamente. $\mathbf{N}_i$, $\bar{\mathbf{N}}_i$, N_i e $\bar{N}_i$ são funções escolhidas apropriadamente de forma a pertencerem ao espaços $\mathbf{V}_h$, $\mathbf{V_0}$, S_h e S_o, respectivamente.

Seguindo o procedimento padrão do método dos elementos finitos, aplicando o método de Galerkin (LEWIS, SCHREFLER, 1998; ZIENKIEWICZ, TAYLOR, 2000), multiplica-se as equações ((2.2) e (2.11) com as respectivas funções de ponderação. Posteriormente ao integra-se por partes os termos de ordem superior, seguido da aplicação do teorema da divergência e, por fim, considerando as condições de contorno, eventualmente obtemos a seguinte formulação fraca do problema:

$$\begin{aligned}&\int_{\Omega_e}\left[\lambda(\nabla\cdot\mathbf{u}_h)\mathbf{I}+\mu\left(\nabla\mathbf{u}_h+\nabla^T\mathbf{u}_h\right)\right]:\nabla\mathbf{v}d\Omega_e=\\&\int_{\Omega_e}\alpha p_h\mathbf{I}:\nabla\mathbf{v}\,d\Omega_e-\int_{\Omega_e}\rho_s\mathbf{g}\cdot\mathbf{v}\,d\Omega_e+\int_{\Gamma_N^{\mathbf{u}}}\mathbf{t}\cdot\mathbf{v}d\Gamma_e\end{aligned}\tag{2.20}$$

$$\begin{aligned}&\int_{\Omega_e}(\phi c_f)\,p_h^{n+1}q\,d\Omega_e-\Delta t\int_{\Omega_e}\nabla q\frac{\mathbf{K}}{\mu}\nabla p_h^{n+1}\,d\Omega_e+\Delta t\int_{\Gamma^p}\gamma\,p_h^{n+1}\,q\,d\Gamma^p=\\&\int_{\Omega_e}(\phi c_f)\,p_h^{n}q\,d\Omega_e-\Delta t\int_{\Omega_e}\rho_f\nabla\cdot\mathbf{g}\,q\,d\Omega_e-\Delta t\int_{\Omega_e}\nabla\cdot\dot{\mathbf{u}}\,q\,d\Omega_e+\\&\Delta t\int_{\Omega_e}fq\,d\Omega_e+\Delta t\int_{\Gamma^p}(\bar{s}+\gamma\bar{p})q\,d\Gamma^p\end{aligned}\tag{2.21}$$

ao substituir as equações (2.16) e (2.19) nas equações (2.20) e (2.21) e suprimir os índices podemos reescrever de forma mais compacta:

$$\begin{aligned}&\int_{\Omega_e}\mathbf{w}^{eT}{\mathbf{B_u}}^{eT}\mathbf{D}{\mathbf{B_u}}^{e}\mathbf{u}^e d\Omega_e=\\&\int_{\Omega_e}\mathbf{w}^{eT}{\mathbf{B_u}}^{eT}\alpha\mathbf{I}\mathbf{N}_p^e\mathbf{p}^e d\Omega_e-\int_{\Omega_e}\mathbf{w}^{eT}{\mathbf{N_u}}^{eT}\rho_s\mathbf{g}\,d\Omega_e+\int_{\Gamma_N^{\mathbf{u}}}\mathbf{w}^{eT}{\mathbf{N_u}}^{eT}\mathbf{t}d\Gamma_e\end{aligned}\tag{2.22}$$

$$\begin{aligned}&\left\{\int_{\Omega_e}\boldsymbol{q}^{eT}\boldsymbol{N_p^{eT}}\left(\phi c_f\right)^e\boldsymbol{N_p^{eT}}\,d\Omega_e-\Delta t\int_{\Omega_e}\boldsymbol{q}^{eT}\left(\nabla\boldsymbol{N_p^e}\right)^T\left[\frac{\boldsymbol{K}}{\mu}\right]^{\mathrm{e}}\nabla\boldsymbol{N_p^e}\,d\Omega_e\right.\\&\qquad\left.+\Delta t\int_{\Gamma^{\mathrm{p}}}\boldsymbol{q}^{eT}\left(\nabla\boldsymbol{N_p^e}\right)^T\gamma\boldsymbol{N_p^e}\,d\Gamma^p\right\}(\boldsymbol{p}^e)^{n+1}\\&\qquad=\int_{\Omega_e}\boldsymbol{q}^{eT}\boldsymbol{N_p^{eT}}\left(\phi c_f\right)^e\boldsymbol{N_p^e}\,(\boldsymbol{p}^e)^n\,d\Omega_e-\Delta t\int_{\Omega_e}\boldsymbol{q}^{eT}\left(\nabla\boldsymbol{N_p^e}\right)^T\left[\rho_f\frac{\boldsymbol{K}}{\mu}\right]^{\mathrm{e}}\nabla\cdot\boldsymbol{g}\,d\Omega_e\\&\qquad-\Delta t\int_{\Omega_e}\boldsymbol{q}^{eT}\boldsymbol{N_p^{eT}}\mathbf{m}^{\mathrm{T}}\boldsymbol{B_u^e}\dot{\boldsymbol{u}}^{\boldsymbol{e}}\,d\Omega_e+\Delta t\int_{\Omega_e}\boldsymbol{q}^{eT}\boldsymbol{N_p^{eT}}f^e\,d\Omega_e\\&\qquad+\Delta t\int_{\Gamma^{\mathrm{p}}}\boldsymbol{q}^{eT}\boldsymbol{N_p^{eT}}(\bar{s}+\gamma\bar{p})\,d\Gamma^p\end{aligned}\tag{2.23}$$

sendo ${\mathbf{B_u}}^{e}=\nabla{\mathbf{N_u}}^{e}$ e $\mathbf{m}=(1,1,1,0,0,0)$um vetor auxiliar.

Como $\mathbf{w}^{eT}$ e $\mathbf{q}^{eT}$ são funções arbitrárias, logo as equações (2.22) e (2.23) podem ser escritas como:

$$\begin{aligned}&\int_{\Omega_e} \mathbf{B_u}^{eT}\mathbf{D}\mathbf{B_u}^{e}\mathbf{u}^{e}d\Omega_e = \\ &\int_{\Omega_e} \mathbf{B_u}^{eT}\alpha\mathbf{I}\mathbf{N}_p^e\mathbf{p}^e d\Omega_e - \int_{\Omega_e} \mathbf{N_u}^{eT}\rho_s\mathbf{g}\, d\Omega_e + \int_{\Gamma_N^{\mathbf{u}}} \mathbf{N_u}^{eT}\mathbf{t}d\Gamma_e\end{aligned} \tag{2.24}$$

$$\begin{aligned}&\left[\int_{\Omega_e} \mathbf{N}_p^{eT}[(\phi c_f)^e + \gamma]\mathbf{N}_p^e d\Omega_e - \Delta t \int_{\Omega_e} \left(\nabla \mathbf{N}_p^e\right)^T \left[\frac{\mathbf{K}}{\mu}\right]^e \nabla \mathbf{N}_p^e d\Omega_e\right](\mathbf{p}^e)^{n+1} = \\ &\int_{\Omega_e} \mathbf{N}_p^{eT}(\phi c_f)^e \mathbf{N}_p^e (\mathbf{p}^e)^n d\Omega_e - \Delta t \int_{\Omega_e} \left(\nabla \mathbf{N}_p^e\right)^T \left[\frac{\rho_f \mathbf{K}}{\mu}\right]^e \nabla \cdot \mathbf{g} d\Omega_e - \\ &\Delta t \int_{\Omega_e} \mathbf{N}_p^{eT}\mathbf{m}^T\mathbf{B_u}^e\dot{\mathbf{u}}^e d\Omega_e + \Delta t \int_{\Gamma^p} \mathbf{N}_p^{eT}(f^e + \bar{s} + \gamma\bar{p}) d\Omega_e\end{aligned} \tag{2.25}$$

A equação (2.24) representa a equação de equilíbrio na forma matricial de um elemento finito tetraédrico e com quatro nós. Nesta expressão, o primeiro termo do lado esquerdo da equação representa a matriz de rigidez do elemento e, enquanto os dois últimos termos do lado direito da equação representam o vetor de forças externas que não tem efeito na deformação elástica do problema. Por fim, o primeiro termo do lado direito é a matriz de acoplamento do problema hidromecânico.

A equação (2.25) representa o problema de fluxo de Darcy através da equação de conservação de massa. A primeira expressão do lado esquerdo da equação (2.25) representa o termo de armazenamento no tempo n+1, enquanto os dois primeiros termos do lado direito da equação (2.25) são o termo de armazenamento no tempo n e o termo gravitacional, respectivamente. O terceiro termo, que não foi implementado neste trabalho, representa o acoplamento do comportamento tensão-deformação dependente das tensões atuantes que causam a variação do meio poroso (deformação volumétrica). Por fim o último termo da equação (2.25) representa as fontes/sumidouros do problema de fluxo.

2.2 ESTRATÉGIAS DE ACELERAÇÃO BASEADAS EM PROGRAMAÇÃO NA GPU E CPU

Na proposta atual o problema de escoamento em meios porosos é acoplado ao problema geomecânica por meio de uma abordagem unidirecional (*oneway*). Neste contexto, o problema de fluxo é resolvido de forma independente, sem retorno de informações do problema mecânico (tais como mudanças na porosidade e

permeabilidade). Neste tipo de acoplamento, todo o histórico de soluções de pressões é armazenado, sendo posteriormente transferido de forma unidirecional ao problema mecânico, onde são computados deslocamentos, tensões e deformações. Neste trabalho o modelo constitutivo utilizado para o problema mecânico é elástico-linear. Em modelos de larga escala são simulações que necessitam da construção e resolução de sistemas de equações lineares com milhões de graus de liberdade.

Para a construção destes tipos de sistemas, uma das técnicas que vêm sendo utilizadas é o procedimento de vetorização da montagem das matrizes dos elementos que compõem o modelo, que quando comparado a um código com estruturas de repetição convencionais se apresenta extremamente eficiente (CUVELIER et al., 2015). Por outro lado, surgem duas desvantagens:

1. A enorme quantidade de memória necessária ao armazenamento de matrizes dos elementos e vetores auxiliares;
2. O comprometimento da interpretação e flexibilidade do código, uma vez que inúmeras operações são reduzidas a poucas linhas de programação.

Uma alternativa para a segunda limitação surge com a conversão das funções originais para linguagens de baixo nível, por exemplo, o MATLAB permite suporte dessas operações com a construção de funções MEX (MATLAB EXecutable), que estabelecem uma interface entre o MATLAB (ou funções do MATLAB) e funções elaboradas em C, C++ ou Fortran.

A Biblioteca MTIMESX para produto multidimensional foi desenvolvida por meio das técnicas de vetorização e da característica matemática que compõe o sistema linear discreto resultante (TURSA, 2020).

As estratégias de aceleração para a montagem das matrizes deste trabalho basearam-se em duas estratégias distintas:

1. O cálculo completo das matrizes dos elementos;
2. O cálculo da parte triangular inferior.

Essas duas abordagens permitem uma avaliação de possíveis vantagens obtidas com a simetria do problema no tempo de processamento.

Diversos autores vêm explorando técnicas para acelerar a montagem da matriz resultante dos produtos e montagem das matrizes B e D (SHIAKOLAS et al. (1994); GRIFFITHS, 2009). Os autores concluíram que a utilização de matrizes em forma explícita é mais eficiente que a utilização de procedimentos tradicionais de integração numérica. O algoritmo da construção em forma explícita consiste em uma busca de

posições comuns (ou colisões de índices produzidas pela conectividade da malha) para soma dos valores individuais da rigidez elementos, e isto demanda o armazenamento do conjunto completo das matrizes dos elementos. Além disso, outros dois vetores (I, J) com as posições correspondentes na matriz de rigidez global são necessários, demandando uma grande utilização de memória RAM e, por consequência, comprometendo computacionalmente o processo de construção da matriz esparsa.

Visando a minimização da utilização de memória RAM, Cuvelier et al. (2016) emprega uma chamada sequencial da função *sparse* do MATLAB. No entanto, tal aplicação evita a utilização de baixa velocidade de memória no swap. Essa alternativa surge com a utilização da simetria da matriz de rigidez global, reduzindo desta forma o consumo de memória do triplet.

Diversos autores exploraram alternativas para melhor desempenho do procedimento de construção da matriz esparsa. Neste sentido, diversas funções em formato MEX foram desenvolvidas por terceiros como alternativas. Engblom e Lukarski (2016a) desenvolveram a função fsparse, que utiliza processamento paralelo para a construção da matriz esparsa. Outra proposta surge com a função sparse2, componente do pacote SuiteSpase (DAVIS, 2019). Nas duas opções nenhuma informação sobre a conectividade dos elementos é fornecida. De maneira geral, as duas rotinas apresentam desempenho superior ao comando nativo do MATLAB (ZAYER et al., 2017).

De forma a explorar, efetivamente, as informações geométricas do domínio, os trabalhos de Krotkiewski e Dabrowski (2008) propuseram uma função definida como *sparse_create*, onde os vetores I e J são substituídos pela matriz de conectividade dos elementos, que apresentam tamanho inferior ao necessário em abordagens que empregam posições globais. Esta função *sparse_create* permite um arranjo mais compacto e com baixa utilização de memória. Além disso, a simetria da matriz de rigidez pode ser incluída como característica adicional no argumento da função. A eficiência da função *sparse_create* está diretamente associada ao número de colisões dos índices da matriz de rigidez global.

Em malhas com poucas ocorrências o ganho de performance é pequeno e a vantagem mais expressiva será a redução do consumo de memória. Uma grande vantagem no MATLAB em sua versão 2020a apresenta suporte para vetores I e J em formato inteiro, o que reduz ainda mais o consumo de memória comparativamente ao tipo double (MathWorks,2020).

A proposta atual deste trabalho visa utilizar práticas de programação eficiente em MATLAB, que seguem o estado da arte em vetorização e construção de matrizes do problema em elementos finitos. Além disso, serão avaliadas estratégias para solução do sistema de equações resultantes contemplando: códigos em versões CPU e GPU, conceitos de matrizes explícitas e produtos multidimensionais entre matrizes, além de funções altamente otimizadas em linguagem MEX para construção das matrizes dos elementos e aceleração de partes críticas dos códigos. A construção das matrizes esparsas surge de forma original com a utilização de estruturas do tipo gpuArray e das grandes vantagens do MATLAB em sua versão 2020a, através do suporte para vetores I e J em formato inteiro, o que reduz ainda mais o consumo de memória. O processo de construção das matrizes dos elementos é avaliado em termos de sua eficiência frente às diversas opções de aceleração apresentadas por outros autores (*sparse2, fsparse*), além de opções mais avançadas que exploram a conectividade da malha (*sparse_create*). Todas as análises são realizadas de forma combinada ao processo de solução do sistema de equações com a utilização do método dos gradientes conjugados, tanto em CPU como em GPU.

Os códigos foram desenvolvidos em linguagem MATLAB para o problema mecânico são definidos pelas formulações em Métodos dos Elementos Finitos (MEF) para elemento tetraédrico de quatro nós utilizando o método de Galerkin, discutidas na seção anterior pela equação (2.24). Todo o processo de aceleração deste trabalho, tanto na CPU quanto na GPU, está no problema mecânico e é dividido nas seguintes etapas:

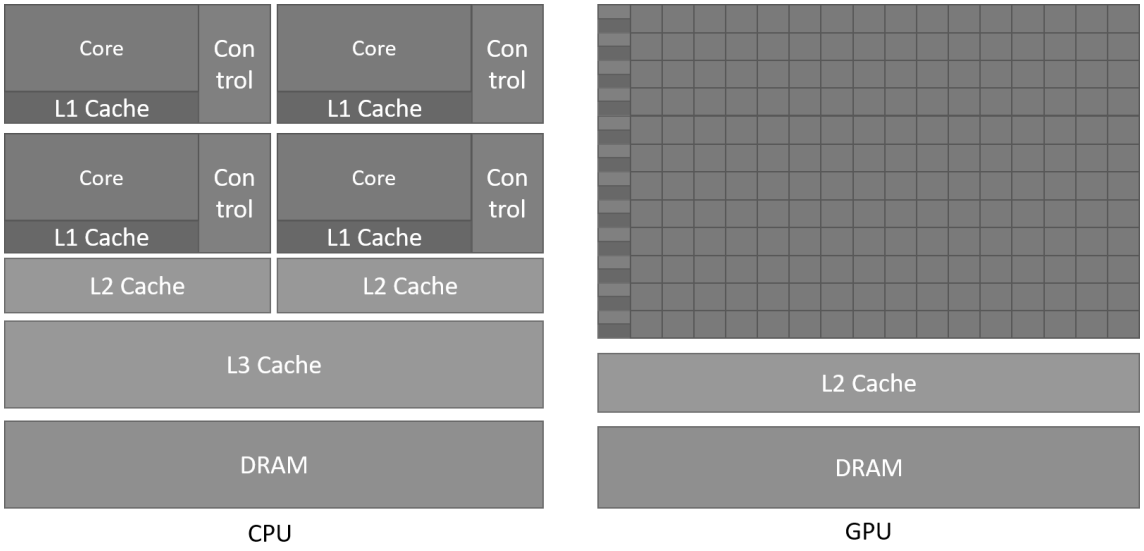
1. Montagem da matriz de rigidez dos elementos (Ke);
2. Construção da matriz de rigidez global (KG);
3. Acoplamento e solução da equação (2.24).

No caso específico da GPU é utilizada a *Parallel Computing Toolbox*, com interface para GPUs NVIDIA. São definidas estruturas do tipo `gpuArray`, que permitem a aplicação de uma série de comandos para operações matriciais. Essas operações são realizadas tanto na etapa de construção das matrizes do problema, como na solução do sistema de equações.

A unidade de processamento gráfica (GPU), possui um único chip como a CPU mas com a principal diferença na quantidade de núcleos, ela pode ter centenas de núcleos alinhados em uma única hardware como mostra na Figura 3. Para problema que é massiva

computacionalmente a GPU supera a CPU em termo de GFLOPS (Giga FLoating point OPerations per Second) (NIKOLAOS et al, 2015). A CUDA (Compute Unified Device Architecture) é a arquitectura utilizada pela NVIDIA para a implementação na GPU (NVIDIA. CUDA C programming guide; 2020) com a integração com a linguagem C++.

Figura 3 : Diferença entre CPU e GPU em termo de número de núcleos. (a) CPU com multicores e (b) GPU com centenas de cores



Fonte: (NVIDIA, 2023)

Por último os níveis L1, L2 e L3 descritos na Figura 3 acima representam características de *cache* de acordo com sua velocidade e tamanho. O cache L1 é menor e mais rápido, o cache L2 é maior e um pouco mais lento, e o cache L3 é ainda maior e mais lento. Cada nível de cache desempenha um papel crucial na otimização do desempenho da CPU, minimizando a quantidade de tempo que a CPU tem que esperar para acessar os dados e instruções de que precisa.

A programação em GPU foi originalmente utilizada para acelerar a renderização de imagens em jogos, porém hoje é utilizada em renderização de gráficos. Neste trabalho, com a aplicação da geomecânica de reservatórios, a GPU é utilizada e tem grande desempenho para acelerar a resolução do sistema de equação linear do problema mecânico a cada passo de tempo. De acordo com o guia de programação da NVIDIA *'CUDA C Best Practices'*, disponível do seu site, nem sempre em outras áreas pode-se aproveitar o poder de processamento, sua utilização em problemas que há transferências frequentes entre a CPU (host) e a GPU pode comprometer seu desempenho.

A utilização da GPU no MATLAB requer poucos esforços devido a simplicidade e não necessita de grande conhecimento em CUDA. Alguns termos técnicos apresentados na Figura 3 acima são descritos na Tabela 1.

Tabela 1: Alguns termos técnicos na programação em GPU

Termos	**Definição**
Core	É uma unidade de processamento única.
DRAM	Dynamic Random Access Memory, é um tipo de memória volátil usada em computadores para armazenar dados temporariamente enquanto o processador trabalha.
Device	uma placa contendo a GPU e a sua memória associada
cache	É uma memória de acesso rápida, ele permite diminuir o tempo do acesso aos dados localizados na memória.
Control	A Unidade de Controle (UC), do inglês Control Unit (CU), é um componente da CPU que direciona a operação do processador

Fonte: O autor (2023)

A Tabela 2 fornece uma visão geral das especificações de algumas GPUs, permitindo que os usuários comparem e escolham a GPU que melhor atenda às suas necessidades de desempenho, memória e recursos de computação paralela.

Tabela 2: Caraterísticas de algumas GPU para Laptops

Caraterísticas da GPU	**GeForce RTX 3080 Ti Laptop GPU**	**GeForce RTX 3080 Laptop GPU**	**GeForce RTX 3070 Laptop GPU**	**GeForce RTX 3060 Laptop GPU**	**GeForce RTX 3050 Laptop GPU**
NVIDIA|CUDA					
Cores	7424	6144	5120	3840	2048 - 2560
Boost Clock (MHz)	1125 - 1590	1245-1710	1290 - 1620	1283 - 1703	990 - 1740
Memory Size (GB)	16	8-16	8	6	4-6
Memory Type	GDDR6	GDDR6	GDDR6	GDDR6	GDDR6

Fonte: O autor (2023)

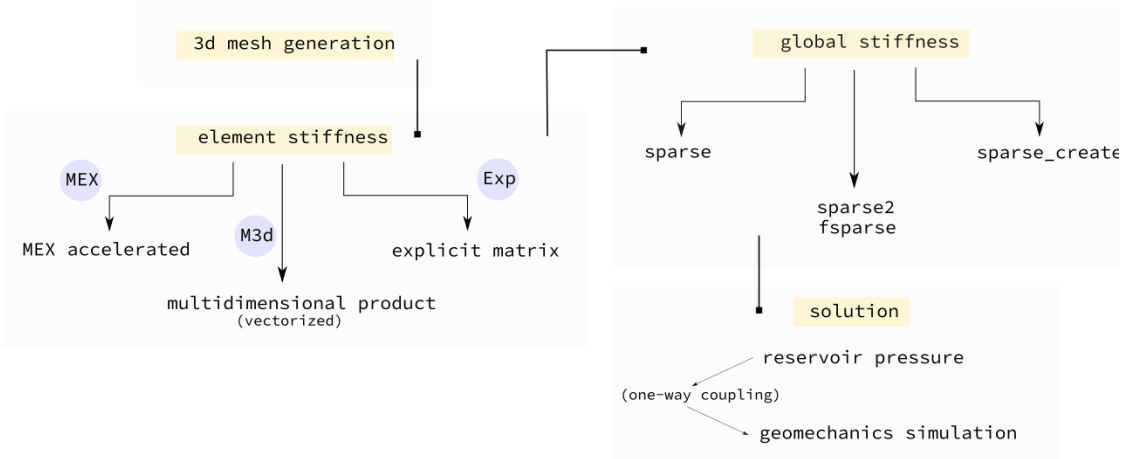
A Tabela 2 descreve diferentes tipos de GPUs da série GeForce RTX encontradas em laptops, especificamente os modelos RTX 3080 Ti, RTX 3080, RTX 3070, RTX 3060 e RTX 3050. Cada linha da tabela apresenta informações sobre as características dessas GPUs. Nesta lista, a RTX 350 foi utilizada neste trabalho para os exemplos da seção resultado.

1. Linha 1: *NVIDIA*|*CUDA*|*Cores* indica o número de núcleos CUDA presentes em cada GPU, indicando a capacidade de processamento paralelo de cada GPU.
2. Linha 2: *Boost Clock* (MHz) indica a faixa de frequência de clock (é uma medida da velocidade de processamento) em que a GPU opera em modo de aumento de desempenho. Os valores variam de acordo com o modelo da GPU, indicando as velocidades mínima e máxima do clock.
3. Linha 3: *Memory Size* (GB) indica a capacidade de memória da GPU em gigabytes, representando a quantidade de memória disponível para armazenar e manipular dados.
4. Linha 4: *Memory Type* indica o tipo de memória usado nas GPUs, que é o GDDR6 (Graphics Double Data Rate 6), uma tecnologia de memória de alto desempenho com alta largura de banda.

Um esquema geral da estrutura do código e opções de análise é apresentado

na Figura 4

Figura 4 Esquema geral da estrutura do código



Fonte: (Joseph et al, 2021)

De forma ainda mais detalhada, são indicadas na Tabela 3 algumas siglas associadas a cada etapa do código.

Tabela 3 Siglas para construções das matrizes dos elementos Ke

Funtion	**Description**
ElemStiff	Construção da matriz de rigidez dos elementos
GlobalStiff	Construção da matriz de rigidez global
Solver	Solução do sistema de equações

Fonte: O autor (2023)

Conforme apresentado anteriormente na Figura 4 três partes dos códigos foram objeto de estudo que são a montagem explicita e implícita (M3d) da matriz de rigidez locais, o uso de funções do tipo MEX que é uma interface entre códigos escritos em C++ e o MATLAB, a montagem global e por fim análise de técnicas para acelerar a resolução do sistema de equação do problema mecânico.

Algumas definições relacionadas à programação na GPU, conforme abordado em detalhes no livro Distributed and Cloud Computing (FOX et al, 2012) e Accelerating MATLAB with GPU Computing (SUH, KIM, 2014).

1. ***GPU (Graphics Processing Unit)****:* a unidade de processamento gráfico, projetada originalmente para renderização de gráficos em jogos e aplicações multimídia. No entanto, as GPUs também podem ser utilizadas para cálculos gerais, fornecendo uma grande capacidade de processamento paralelo (MATHWORK, 2023);
2. ***Programação na GPU***: refere-se ao desenvolvimento de algoritmos e códigos para aproveitar a capacidade de processamento paralelo das GPUs. Essa programação geralmente envolve o uso de APIs (Application Programming Interfaces) específicas, como CUDA (Compute Unified Device Architecture) para NVIDIA, GPUs ou OpenCL (Open Computing Language) para várias marcas de GPUs;
3. ***FLOPS (Floating Point Operations per Second)***: é uma medida de desempenho que indica o número de operações de ponto flutuante (soma, subtração, multiplicação e divisão) que uma GPU (ou qualquer processador) pode executar

por segundo. FLOPS é usado para quantificar a capacidade de processamento de uma GPU;

4. ***Threads***: são unidades básicas de execução em uma GPU. Um *thread* corresponde a uma única tarefa que pode ser executada em paralelo com outros *thread*s. Os *thread*s são organizados em blocos e grids;
5. ***Blocos***: são agrupamentos de *thread*s em uma GPU. Os blocos podem ser organizados em uma ou mais dimensões, dependendo das necessidades do algoritmo. Os blocos podem ser sincronizados para coordenar a execução dos *thread*s;
6. ***Núcleos***: são unidades de processamento dentro de uma GPU que executam instruções em *threads*. Também conhecidos como "CUDA cores" em GPUs, eles são responsáveis por executar as operações matemáticas e lógicas nos *threads*;
7. ***Speedup***: é uma métrica usada para medir o ganho de desempenho obtido ao executar um algoritmo em paralelo na GPU em comparação com a execução sequencial em uma CPU. O *speedup* é calculado pela divisão do tempo de execução sequencial pelo tempo de execução paralela;
8. ***Warp***: é um grupo de *thread*s executadas simultaneamente em uma GPU. Geralmente, um *warp* contém 32 *thread*s e é a menor unidade de escalonamento da GPU. As instruções são executadas em um modelo SIMD (*Single Instruction, Multiple Data*) dentro de um *warp*;
9. ***Memória***: as GPUs possuem diferentes tipos de memória para armazenar dados usados durante a execução dos *kernels* (funções executadas na GPU). Isso inclui memória global (acessível por todos os blocos), memória compartilhada (acessível apenas por *thread*s em um bloco) e memória local (usada para armazenar dados temporários por cada *thread*);
10. ***Streaming Processor/ Streaming Multiprocessor***: é uma unidade de processamento dentro da GPU que executa as tarefas de processamento paralelo. Em geral, as GPUs possuem vários streamings multiprocessors que são responsáveis por executar os *thread*s e controlar a comunicação entre as diferentes memórias da GPU;
11. ***Backslash:*** o operador de barra invertida (\) é comumente empregado para resolver sistemas de equações lineares ou para encontrar a solução de mínimos quadrados para um sistema sobredeterminado. Ele também é conhecido como operador de divisão à esquerda;

12. ***FFT***: refere-se ao algoritmo de transformação rápida de Fourier implementado usando o tipo de dados de precisão dupla. A Transformada Rápida de Fourier (DFT) é um algoritmo matemático usado para calcular com eficiência DFT ou seu inverso. Ela transforma um sinal no domínio do tempo em sua representação no domínio da frequência;
13. ***MTimes***: abreviação de multiplicação de matriz vezes, refere-se à operação de multiplicação de duas matrizes em matemática. É uma operação comum usada em álgebra linear;
14. ***Host***: refere-se à parte do código que é executada na CPU (*Central Processing Unit*) principal do sistema. O host é responsável por coordenar a execução dos *kernels* na GPU (Graphics Processing Unit) e gerenciar a transferência de dados entre a CPU e a GPU;
15. ***Device***: refere-se à GPU, que é o processador paralelo dedicado usado para executar tarefas em paralelo. A GPU é o dispositivo de processamento que realiza cálculos massivamente paralelos e é responsável por executar os *kernels* do CUDA;
16. ***Host Memory***: é a memória principal acessível pelo host;
17. ***Device Memory*** (***Memória do Dispositivo***): é a memória da GPU acessível pelo dispositivo (GPU). É usada para armazenar dados que são acessados e processados pelos *kernels* do CUDA durante a execução na GPU;
18. ***Kernel***: é uma função especial na programação CUDA que é executada paralelamente por vários *threads* na GPU. Os *kernels* são escritos em linguagem CUDA e são responsáveis por realizar cálculos intensivos em paralelo;
19. ***CUDA (Compute Unified Device Architecture):*** é uma plataforma de computação paralela desenvolvida pela NVIDIA. Permite que seja utilizada a capacidade de processamento de GPUs (Unidades de Processamento Gráfico) da NVIDIA para acelerar a execução de tarefas computacionais intensivas.

Neste trabalho, a ferramenta MATLAB foi utilizada para desenvolver as rotinas do problema mecânico. Ela é uma linguagem de programação utilizada para realizar cálculos numéricos e simulações computacionais. Esta ferramenta é amplamente utilizada em diversas áreas, como engenharia, ciência, finanças e processamento de sinais, devido à sua facilidade de uso e recursos avançados.

Uma das principais vantagens do software da empresa MathWorks é a sua capacidade de lidar com computação paralela, ou seja, a execução de cálculos em paralelo utilizando múltiplos processadores ou núcleos de processamento. Para facilitar essa tarefa, o MATLAB oferece a *Parallel Computing Toolbox* (*PCT*).

Neste trabalho, o principal foco se deu principalmente na parte de GPU da *PCT*. Um dos pontos positivos desta ferramenta é o fato de podermos aproveitar as vantagens das GPUs sem programação CUDA ou c-mex explícita. Entretanto, a instalação da *Parallel Computing Toolbox* se torna importante e crucial.

A *PCT* do MATLAB oferece recursos para resolver problemas computacionais e de processamento intensivo de dados, tirando proveito de processadores multicore, GPUs e *clusters* de computadores. Com a PCT, é possível paralelizar aplicativos MATLAB sem a necessidade de programação específica em CUDA ou MPI. Essa caixa de ferramentas permite o uso de funções paralelizadas tanto no MATLAB quanto em outras caixas de ferramentas disponibilizadas pela MathWorks (MathWorks, 2023).

Através da *PCT* é possível realizar a execução de aplicações paralelas de forma interativa ou em modo de lote. A ferramenta possibilita o aproveitamento total da capacidade dos computadores desktops com processadores multicores, por meio da execução de aplicativos em Workers, que são motores computacionais do MATLAB, executados localmente. É importante ressaltar que não é necessário fazer alterações no código para executar os mesmos aplicativos em clusters ou ambientes em nuvem, bastando utilizar o MATLAB *Parallel Server*. Dessa forma, é possível expandir a capacidade de processamento para cenários distribuídos sem a necessidade de modificações significativas no código-fonte original.

A programação na *GPU* é mais desafiadora quando comparada à programação paralela para ambientes multicores. Isso ocorre devido à hierarquia dos threads. Na GPU a estrutura segue esta linha de raciocino, alguns threads compõem um warp, alguns warps compõem um bloco de threads, e alguns blocos de threads compõem um grid (P. Xiang, Y. Yang,H. Zhou,2014).

Para iniciar qualquer implementação utilizando GPU da *PTC do* MATLAB, precisa instalar através a opção *Add-one explorer* que se localizada na interface do MATLAB.

Após a instalação do pacote PTC, ao digitar a função nativa do MATLAB *gpuDevice*, são exibidas várias informações sobre a GPU instalada no seu computador. Abaixo estão algumas das propriedades que são comumente exibidas e a Figura 5 apresenta as características da GPU

Figura 5 Configuração e propriedades da GPU disponível

```
>> gpuDevice

ans =

  CUDADevice with properties:

                      Name: 'NVIDIA GeForce RTX 3050 Laptop GPU'
                     Index: 1
         ComputeCapability: '8.6'
            SupportsDouble: 1
     GraphicsDriverVersion: '512.89'
               DriverModel: 'WDDM'
            ToolkitVersion: 11.8000
        MaxThreadsPerBlock: 1024
          MaxShmemPerBlock: 49152 (49.15 KB)
        MaxThreadBlockSize: [1024 1024 64]
               MaxGridSize: [2.1475e+09 65535 65535]
                 SIMDWidth: 32
               TotalMemory: 4294443008 (4.29 GB)
           AvailableMemory: 3310582171 (3.31 GB)
               CachePolicy: 'balanced'
      MultiprocessorCount: 16
             ClockRateKHz: 1500000
               ComputeMode: 'Default'
     GPUOverlapsTransfers: 1
   KernelExecutionTimeout: 1
         CanMapHostMemory: 1
          DeviceSupported: 1
          DeviceAvailable: 1
           DeviceSelected: 1
```

Fonte: o Autor (2023)

sendo

1. ***Name***: Nome da GPU;
2. ***Index***: Índice da GPU no sistema, que pode ser útil se houver várias GPUs disponíveis;
3. ***ComputeCapability***: Capacidade de computação da GPU, indicando a versão e recursos suportados pela arquitetura da GPU;
4. ***SupportsDouble***: indica se a GPU suporta cálculos em precisão dupla (*double precision*);
5. ***DriverVersion***: versão do driver da GPU instalado no sistema;

6. ***ToolkitVersion***: versão da CUDA *Toolkit* instalada no sistema, que é usada para desenvolver e executar aplicativos em GPU;
7. ***MaxThreadsPerBlock:*** define o número máximo de *threads* que podem ser executados em um único bloco em uma GPU específica*;*
8. **MaxGridSize:** representa o tamanho máximo da grade (*grid*) que pode ser usado em uma execução de kernel em uma GPU específica;
9. ***SIMDwidth*:** é a largura do SIMD (*Single Instruction, Multiple Data*) em uma arquitetura de GPU;
10. ***MaxShmemPerBlock:*** representa a quantidade máxima de memória compartilhada por bloco em uma GPU específica;
11. ***Memory***: capacidade de memória total da GPU disponível para uso.
12. ***FreeMemory***: quantidade de memória livre na GPU atualmente;
13. ***MultiprocessorCount*:** número de multiprocessadores na GPU;
14. ***ClockRateKHz***: taxa de *clock* da GPU em *kilohertz*;
15. ***ComputeMode***: modo de computação da GPU, que indica como a GPU está sendo usada (por exemplo, em computação exclusiva ou compartilhada com a exibição);
16. ***DeviceSupported***: indica se a GPU é suportada pelo MATLAB para computação *em GPU.*

2.2.1 Montagem da matriz de rigidez Global dos elementos com Ke implicita (Ke:M3d)

Seguindo o padrão indicado no esquema mostrado anteriormente na Figura 4, são discutidas diversas opções para a construção da matriz de rigidez dos elementos do problema mecânico. A ineficiência computacionalmente, devido ao triplo *loop*, apresenta limitações em problemas onde a matriz de rigidez global é alocada de forma esparsa e com grande quantidade de graus de liberdade. O trabalho de (KOKO, 2007) apresenta uma forma eficiente com códigos vetorizados para problema de elasticidade linear 2D e 3D. Conforme demonstrado por (RAHMAN, VALDMAN, 2013) e (CUVELIER et al., 2016), um procedimento mais atrativo surge com o cálculo prévio de todas as matrizes dos elementos (e de seus índices de posições globais), e posteriormente são alocados os termos em uma única atribuição da matriz esparsa, indicada no Algoritmo 1 (linha 14).

A redução a uma única estrutura de repetição torna essa última versão mais eficiente. Por outro lado, exige as seguintes etapas:

1. o cálculo simultâneo de todas as matrizes de rigidez dos elementos;
2. a construção dos vetores **L** e **C** com índices de posições globais de linhas e colunas de cada valores das matrizes locais.

Algoritmo 1: código para computação de Ke e montagem de Kg esparso

```
1   for i=1:nelem
2       % stiffness matrix for the i-th element
3               Ke(:,:,i)=B'(i)D(i)B(i)V(i);
4   End
5           L=[...];        % L = global row indexes
6           C=[...];        % C = global column indexes
7   % -------------------------------------------------------------------
8   % defines global sparse matrix in a single call using COO format
9   % -------------------------------------------------------------------
10  % ndof = number of degrees of freedom
11  % triplets given by L, C and Ke(:)
13          V=Ke(:);
14          Kg=sparse(L,C,Ke(:),ndof,ndof);
```

Nota: Algumas linhas foram omitidas para fins de simplicidade

Fonte: o Autor (2023)

Vale observar na linha 14 do Algoritmo 1 anteriormente, cinco parâmetros precisam ser previamente armazenados para a construção da matriz global Kg. Primeiro os índices Linha (L), Coluna (C) de cada valor das matrizes locais de cada elementos armazenados no Ke e os seus respectivos graus de liberdade nomeados de (*ndof*, *ndof*). Esta configuração, apesar é bastante eficiente, demanda uma capacidade de armazenamento de cada parâmetros. As próximas seções indicam estratégias e técnicas que permitem desempenho ainda superior ao apresentado no Algoritmo 1, com foco na construção das matrizes Kg.

2.2.2 Exp – Montagem da Matriz de rigidez Global KG com montagem Ke explicita

Outra abordagem estudada e desenvolvida nesta tese é a técnica de montagem matriz global com Matriz de rigidez dos elementos Ke explícita que apresenta solução

fechada e tem excelente eficiência computacional nas operações algébricas. Também implica em duas vantagens notáveis, que são a ausência de produto entre matrizes e de procedimentos de integração numérica. O resultado é uma expressão simbólica para cada termo da matriz de rigidez dos elementos. Em termos computacionais isto resulta em uma simples substituição dos termos envolvidos em cada elemento da matriz de rigidez, com possível eliminação prévia dos termos simétricos e compactação das variáveis comuns.

Essa construção pode ser realizada tanto em CPU, como em GPU. Na GPU, basta apenas a substituição das matrizes convencionais por estruturas do tipo `gpuArray`. O procedimento computacional para a montagem da matriz **W** mostrado na Figura 6 consiste na elaboração de expressões explícitas para os termos da parte triangular inferior. Para um elemento tetraedro linear (com 3 graus de liberdade por nó) isto implica em 78 termos únicos. Esses são organizados em uma lógica que segue o padrão de indexação linear do MATLAB, com leitura das colunas em forma consecutiva.

Na opção de montagem da matriz utilizando a sua simetria, apenas os 78 termos são armazenados, o que possibilita um ganho enorme em espaço na memória, e os demais termos foram obtidos com um espelhamento dos termos da parte inferior da matriz. A Figura *6* esclarece, com um exemplo para a opção com simetria. Um índice linear é atribuído a cada termo das matrizes de rigidez dos elementos (Figura 6a e Figura 6b). Na sequência, os termos são alocados a uma coleção global de termos, onde cada coluna corresponde a contribuição de um elemento (Figura 6d). Ao final é obtida uma matriz com 78 linhas e colunas em número igual ao total de elementos da malha (nelem).

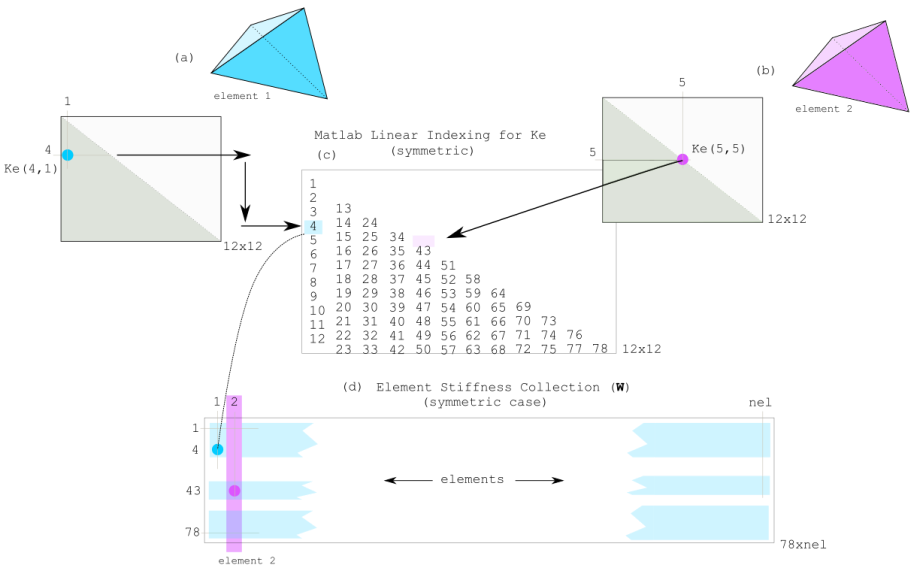
É evidente que a opção com simetria apresenta vantagens expressivas em termos de utilização de memória física. Para a coleção **W** isto implica em 46% de economia em espaço de armazenamento para o elemento tetraedro linear (apenas 78 termos de um total de 144). O mesmo comportamento é esperado para os vetores **L** e **C**, que contêm os índices de posições globais. O código de referência para o procedimento proposto é indicado no Algoritmo 2.

O cálculo de cada linha de **W** da coleção da matriz de rigidez mostrada na Figura 6 é realizado de maneira vetorizada, o que é a principal diferença em relação ao Algoritmo 1. Um exemplo é dado para o primeiro termo **K1** ilustrado na equação (2.26) do tetraedro linear, indicado abaixo:

$$\mathbf{K1} = (E.*(2*b1b1*v + 2*c1c1*v + 2*d1.\hat{}2*v - 2 * b1b1\ \ 20 - c1c1 - d1d1)).\/(72*VV*(2*v\hat{}2 + v - 1)) \quad (2.26)$$

onde **K1** representa um vetor com dimensão igual ao número de elementos da malha e cada coluna corresponde a um termo da matriz de rigidez de um dado elemento. Os termos b1, c1, d1 e seus produtos indicam constantes associadas às coordenadas nodais dos elementos e são vetores. O mesmo ocorre para **VV** na linha 10 do algoritmo 1, que define um vetor com o volume de todos os elementos da malha. Desta forma, K1 é uma operação algébrica entre vetores, resultando em um novo vetor, e definindo em suas colunas os termos **Ke** (1,1) da matriz de rigidez dos elementos da malha. A mesma lógica pode ser aplicada aos termos remanescentes da matriz de rigidez dos elementos.

Figura 6 Indexação linear e ideia do método para alocação dos termos da matriz explícita (caso com simetria). (a) elemento 1; (b) elemento 2; (c) índices da parte triangular inferior; (d) coleção final com matrizes dos elementos.



Fonte: (Joseph et al, 2021)

O algoritmo 2 mostra o procedimento do cálculo da matriz W mostrada na Figura *6* acima.

Algoritmo 2: código de para calcular W
`% computes nodes in a vectorized form`

```
node1=conec(:,1); node2=conec(:,2); node3=conec(:,3); ...
% computes coordinates in a vectorized form
X1=coord(node1,1); Y1=coord(node1,2); Z1=coord(node1,3); ...
% evaluates auxiliar geometric vectors for the explicit assembly
        b1=…; c1=…; d1=…;
% ------------------------------------------------------------
% computes each individual line of the stiffness collection W
% ------------------------------------------------------------
K1=(E.*(2*b1b1*v + 2*c1c1*v + 2*d1.^2*v - 2*b1b1 - …
    c1c1 - d1d1))./(72*VV*(2*v^2 + v - 1));
% continues for the remaining lines and defines W
W=[K1;K2;K3…
V=W(:);
```

Observação: algumas linhas foram omitidas para simplificar

Fonte: O autor (2023)

Resta ainda a avaliação dos vetores **L** e **C,** responsáveis pela indexação dos valores na matriz de rigidez global. A construção desses vetores pode ser facilmente vetorizada, a lógica é a recuperação das posições globais associadas a cada grau de liberdade local. Para o elemento tetraedro linear, com 3 graus de liberdade por nó, são válidas as seguintes relações:

$$[3n_i - 2 \;\; 3n_i - 1 \;\; 3n_i] \rightarrow para\ cada\ nó\ n_i$$

Assim, cada nó resulta em 3 índices globais. Para o tetraedro linear surgem 12 posições no total, que podem ser obtidas pela simples repetição da regra acima a cada um elemento (definidos posteriormente como linhas da matriz **GL**). A definição completa surge com as permutações desses termos para cada elemento da malha. Se a simetria da matriz não for aproveitada, cada elemento contribui com 144 posições para os vetores **L** e **C**. No caso simétrico, essa dimensão é reduzida a 78 posições. O Algoritmo 3 indica os procedimentos necessários para a construção desses vetores, que são válidos para o problema de um elemento tetraedro linear. Após a avaliação dos triplets(I,J,V), a matriz de rigidez global é obtida com uma chamada única do comando *sparse*, conforme discutido nas seções anteriores.

Algoritmo 3: código para cálculo de L e C

```
1  % computes global indexes in a vectorized fashion
2  % GL is a collection of 12 global indexes in each line
```

```
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];
% ------------------------------------------------------------------
% Option 1 - full computation (without symmetry)
% ------------------------------------------------------------------
GL=permute(GL,[3 2 1]);    % organize in 3d slices for each element
aux=repmat(GL,1,12,1);     % repeat each slice in 12 columns
L=aux(:);                  % organize in vector format
aux=repmat(GL,12,1,1);     % repeat each slice in 12 lines
C=aux(:);                  % organize in vector format
% ------------------------------------------------------------------
% Option 2 - symmetric computation
% ------------------------------------------------------------------
% the indexing in the next line follows Fig. 5 explanation
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]);
L=aux(:);                  % organize in vector format
% the indexing in the next line follows Fig. 5 explanation
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]);
C=aux(:);                  % organize in vector format
```

Observação: algumas linhas foram omitidas para simplificar

Fonte: O autor (2023)

2.2.3 M3d – Multidimensional product

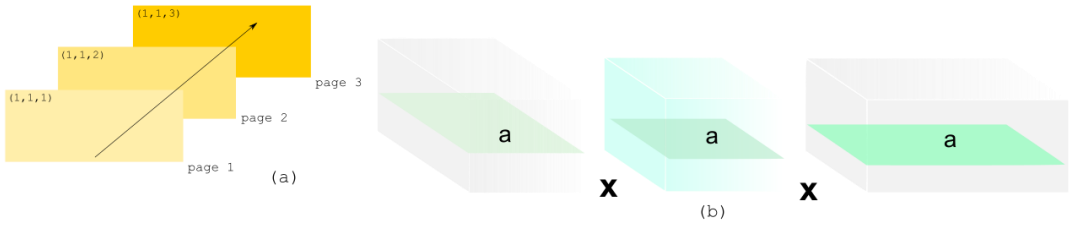
Esta estratégia emprega o conceito de produto multidimensional de matrizes e surge como alternativa ao procedimento com matrizes explícitas. Explora a característica de que as matrizes **B** e **D** da equação (2.22) podem ser computadas rapidamente com operações vetoriais (seguindo a mesma lógica da seção anterior). E que o produto envolvido na construção das matrizes dos elementos pode ser realizado sincronicamente, desde que planos independentes sejam definidos para cada elemento, e que uma operação simultânea mostrada na equação (2.27):

$$(K_e)_\alpha = B_\alpha^T D_\alpha B_\alpha V_\alpha \tag{2.27}$$

onde os diversos planos (ou páginas) α estão associados a elementos da malha. A Figura *7* ilustra como é realizada a construção da equação anterior (2.27). Este procedimento evita dois problemas específicos:

1. a construção de termos explícitos apresentada na seção anterior, que envolve manipulações simbólicas e algebrismo tedioso;
2. a realização serial (e lenta) de numerosos produtos entre as matrizes que compõem a matriz de rigidez do elemento (identificados por planos α). Como desvantagem, realiza o produto completo entre as matrizes e não aproveita uma possível simetria do problema.

Figura 7 Interpretação do conceito de matrizes multidimensionais: (a) diversas páginas de um mesmo índice, (b) o produto multidimensional aplicado ao problema, com evidência para o plano \alpha



Fonte: (Joseph et al, 2021)

Neste trabalho são propostas duas opções para o produto multidimensional:

1) a compilação de uma função MEX denominada MTIMESX (TURSA, 2020), apta a operações em CPU, a construção da matriz Ke é apresentada neste trabalho;

2) a utilização do comando nativo do MATLAB *pagefun*, que aplica uma operação a cada página de uma matriz multidimensional armazenada na GPU;

A estrutura do código para qualquer uma dessas opções é muito similar, com detalhes apresentados nos Algoritmos 4 (CPU) e 5 (GPU) para o caso de um tetraedro linear. Na versão GPU é importante a definição de todas as variáveis na placa aceleradora. Isto é obtido facilmente com a utilização de estruturas do tipo *gpuArray*.

Algoritmo 4: código para produto multidimensional na CPU

```
% defines 3d arrays for B and D (assumed equal for all elements)
B=zeros(6,12,nel); D=repmat(D,1,1,nel);
% populates B indexes (sample indexes presented)
B(1,1,:)=(1./(6*VV)).*b1;
B(1,4,:)=(1./(6*VV)).*b2; …
```

```
% defines volume vector as pages (slices)
Ve=permute(VV', [3 2 1]);
% ------------------------------------------------------------------
% computes the multidimensional product B'DBV with MTIMESX on CPU
% ------------------------------------------------------------------
% first computation B'D
BtD = mtimesx(B,'T',D);     % 'T'flag for transpose
% second computation B'DB
BtDB=mtimesx(BtD,B);
% final computation
Ke=mtimesx(BtDB,Ve);
```

Observação: algumas linhas foram omitidas para simplificar

Fonte: O autor (2023)

Algoritmo 5: código para produto multidimensional na GPU

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

Observação: algumas linhas foram omitidas para simplificar

Fonte: O autor (2023)

Uma vez obtidos os triplets L, C e V, o procedimento segue com a construção da matriz de rigidez global KG, definida como esparsa, para melhor eficiência em termos de armazenamento. Para problemas em larga escala duas características importantes devem ser observadas:

1. O tamanho ocupado em memória RAM pelos triplets;
2. O tempo necessário para a construção da matriz esparsa.

O item (1) e possíveis alternativas para redução de memória RAM já foram discutidos na seção anterior. Em métodos do tipo matrix-free o item (2) pode ser ignorado (isto será discutido na seção específica sobre o solver).

As operações com solucionadores nativos no MATLAB exigem o armazenamento das matrizes de forma explícita. Desta forma, ênfase será dada às opções para solução do item (2).

Conforme discutido por Engblom e Lukaeski (2016) a construção da matriz esparsa antecede qualquer operação do solver, e pode ser facilmente identificada como gargalo computacional em cenários onde a reconstrução da matriz é necessária, tal como em problemas dinâmicos não lineares. Mesmo em operações únicas, o processamento computacional pode ser intenso, e depende do número de colisões (termos comuns) nos vetores **L** e **C**, que devem ser somados termo a termo. Por este motivo, diversos trabalhos são dedicados ao tema, tais como (KROTKIEWSKI & Dabrowski, 2010), (ENGBLOM & LUKARSKI, 2016) e Davis (2019).

Os experimentos numéricos deste trabalho empregam a função *sparse* nativa do MATLAB. Adicionalmente, análises são realizadas com alternativas propostas por outros autores em funções MEX, serão discutidas nas próximas subseções.

2.2.4 Montagem da matriz de rigidez global (KG) utilizando a função *sparse* nativa do MATLAB

A função nativa *sparse* do MATLAB produz uma matriz esparsa a partir dos triplets (**L**, **C**, **V**), por meio da seguinte lógica: um par de índices **L**(k) e **C**(k) que caracteriza um valor **V**(k). Termos comuns (colisões) são somados e o tamanho final da matriz corresponde apenas ao espaço alocado aos termos não-nulos dessas operações. Esta etapa é indicada na Equação (2.28) e pelo Algoritmo 6.

$$KG=sparse(\mathbf{L,C,V},ndof,ndof); \tag{2.28}$$

Sendo *ndof* o número grau de liberdade total do problema, neste caso para o elemento tetraédrico de 4 nós , o *ndof* é igual a 12.

Algoritmo 6: código para definição de matriz esparsa na GPU

```
% defines L and C with integer format
L=int32(L); C=int32(C);
% transfers L,C and V vectors to the GPU
L=gpuArray(L); C=gpuArray(C); V=gpuArray(V);
% generates sparse matrix on the GPU
KG=sparse(L,C,V,ndof,ndof);
```

Observação: algumas linhas foram omitidas para simplificar

Fonte: O autor (2023)

Em versões anteriores o MATLAB exigia formato em dupla precisão para os vetores **L** e **C**. Isto foi modificado a partir da versão 2020a, que permite a utilização de formato inteiro para a indexação. Também é importante notar que as operações podem ser realizadas em GPU, explorando o paralelismo da placa aceleradora. Mais detalhes podem ser consulados no trabalho de (GILBERT et. al, 1992).

Em problemas de larga escala os vetores **L**, **C** e **V** indicados na Equação (2.28) podem ser inviáveis, com a ocorrência de memória VRAM insuficiente na GPU (de forma distinta do que ocorre na memória RAM da CPU). Mesmo o armazenamento dos triplets na GPU não garante sucesso na montagem, que pode resultar em alocação de memória insuficiente e erro no processo de construção da matriz esparsa em GPU, ainda que o tamanho final da matriz global seja inferior a memória VRAM. Neste caso, duas soluções são possíveis:

1. Primeiro, você constrói a matriz esparsa em sua unidade de processamento central (CPU). Depois, para aproveitar do poder de processamento da GPU, utilizou-se o comando *gpuArray* para transferir essa matriz para a GPU, onde operações paralelas são executadas de maneira mais eficiente. Essa abordagem é especialmente útil em tarefas que envolvem grandes conjuntos de dados ou cálculos intensivos.
2. O fracionamento do comando *sparse* em GPU, com a montagem progressiva de matrizes menores, obtidas pela construção parcial dos triplets (**L, C,V**) em GPU, que utilizam apenas uma fração desses vetores, com tamanho adequado a alocação na VRAM, e que podem ser alocadas no dispositivo. Isto pode ser obtido por meio de estruturas de repetição, que limpam as variáveis já utilizadas na rodada anterior e permitem nova alocação na GPU. As mesmas considerações de fracionamento

descritas para a GPU também se aplicam ao caso de construção em CPU, em cenários de memória RAM limitada.

Para entender melhor a técnica descrita no item (2) acima, segue algumas sequências do pseudo-código descrito na Tabela 4

Tabela 4 Algoritmo do fracionamento do comando sparse em CPU e GPU

1. Inicialize as variáveis necessárias (L, C, V, ndof) com os valores apropriados. 2. Converta L e C para o formato de dados inteiros (integer format). 3. Transfira as matrizes L, C e o vetor V para a memória da GPU. 4. Inicialize a matriz esparsa KG como uma matriz vazia na GPU. 5. Defina um tamanho adequado para a alocação de memória na GPU. 6. Para cada iteração: • Limpe as variáveis L, C e V utilizadas na iteração anterior; • Realize a montagem progressiva dos *triplets* (L, C, V) em GPU, utilizando apenas uma fração dos vetores L, C e V; • Adicione os *triplets* montados à matriz KG na GPU. 10. Fim do loop. 11. A matriz esparsa KG estará completa na GPU.

Fonte: O autor (2023)

Diante das considerações expostas acima, qualquer estratégia de utilização eficiente do comando *sparse* deve observar os seguintes critérios:

- a capacidade da memória RAM da CPU e VRAM da GPU;
- o formato numérico inteiro para os vetores **L** e **C** seja considerado**;**
- a possibilidade do uso da simetria da matriz de rigidez dos elementos.

2.2.5 Montagem da matriz de rigidez global (KG) utilizando a função *fsparse* (Engblom e Lukarski, 2016)

Essa é uma alternativa proposta para aceleração do comando *sparse* nativo do MATLAB, por meio de função MEX, que explora o paralelismo de computadores de múltiplos núcleos em CPU, além da consideração da indexação com números inteiros (aspecto presente apenas na versão 2020a do MATLAB). Os autores discutem uma

abordagem progressiva a partir de uma implementação serial. Posteriormente apresentam uma opção com interface *OpenMP*. Os códigos fazem parte da distribuição PARALUTION, que consiste em uma biblioteca para solucionadores iterativos em matrizes esparsas em CPUs multicore e placas aceleradoras gráficas. A utilização da função *fsparse* segue lógica similar ao comando nativo *sparse* do MATLAB.

2.2.6 Montagem da matriz de rigidez global (KG) utilizando a função *sparse2* (Davis, 2019)

Também é uma alternativa ao comando nativo do MATLAB, que mais uma vez não exige a definição dos índices da matriz esparsa em formato de dupla precisão (DABROWSKi et al., 2008). O armazenamento em formato int32 exige apenas 4 bytes, o formato double requer 8 bytes. Isto resulta em diminuição considerável de memória nos triplets. *Sparse*2 é parte do pacote *SuiteSparse* (DAVIS, 2019) que inclui diversos algoritmos para operações em matrizes esparsas com utilização de CPU e GPU.

Alguns fazem parte de distribuições regulares do MATLAB. Outros podem ser incluídos a critério do usuário, tal como o factorize, que representa um *backslash* (\) reutilizável para fatoração da matriz dos coeficientes A no problema de sistemas lineares A\b. No caso específico do *sparse2*, que é parte do CHOLMOD que é uma biblioteca de alto desempenho para fatorização de Cholesky em matrizes esparsas simétricas e definidas positivas. Foi desenvolvida como parte do pacote SuiteSparse pelo Prof. Tim Davis da Texas A&M University. A fatorização de Cholesky é uma técnica importante em álgebra linear e é frequentemente usada em problemas computacionais que envolvem matrizes simétricas e definidas positivas. A forma de utilização da função *sparse2* é idêntica ao comando nativo do MATLAB, basta substituir a função *sparse* pela função *sparse2* da linha 6 do Algoritmo 6.

2.2.7 Montagem da matriz de rigidez global (KG) utilizando Função *sparse_create* (Krotkiewski e Dabrowski, 2008)

As três técnicas apresentadas anteriormente *sparse*, *sparse*2 e *fsparse* se alimentam dos tiplets linha (**L**), coluna (**C**) e valor (**V**). A montagem que utiliza a função *sparse_create*, que é parte da biblioteca MILAMIN (DABROWSKI ,2008) dispensa a

utilização dos vetores **L** e **C**, o que é uma grande vantagem quando são utilizadas malhas com milhões graus de liberdade.

O trabalho de Krotkiewski e Dabrowski (2008) explora efetivamente as informações geométricas do domínio e propõe uma função definida como *sparse_create*, onde os vetores **L** e **C** são desnecessários e substituídos pela matriz de conectividade dos elementos (que já faz parte do problema. Este aspecto permite uma função mais compacta, com menor utilização de memória RAM e mais eficiente na identificação de colisões. Além disso, a simetria da matriz de rigidez e a opção de paralelismo com OpenMP podem ser incluídas como características adicionais no argumento da função.

A eficiência da função *sparse_create* está diretamente associada ao número de colisões dos índices da matriz de rigidez global. Em malhas com poucas ocorrências o ganho de performance é pequeno e a vantagem mais expressiva será a redução do consumo de memória. Em aplicações tradicionais sem simetria um elemento tetraedro linear exige 144 posições em cada vetor L e C. Para a opção com *sparse_create* apenas 12 posições são necessárias, o que é uma vantagem expressiva.

Nesta função, dois parâmetros são necessários: a matriz das conectividades modificada (conecR), com dimensões 12xnelem para elementos tetraédricos, e a matriz dos elementos Ke, armazenada em uma coleção, de forma semelhante ao exposto no caso de matrizes explícitas, com dimensão 144xnelem, onde cada coluna corresponde a matriz de rigidez de um determinado elemento. O Algoritmo 7 indica um exemplo de utilização na versão sem simetria.

Algoritmo 7: sparse_create em elementos de tetraedro linear (geral)

```
% reshapes Ke for column wise collection
Ke=reshape(Ke,[144,nel]);
% connectivity matrix is reshaped to 4 x nel dimensions
conec=uint32(conec);
conec=conec';
-------------------------------------------------
% redefines connectivity matrix 12 x nel for sparse_create format
conecR=uint32(zeros(12,nel));
conecR(1,:)=3*node1-2; conecR(2,:)=3*node1-1; conecR(3,:)=3*node1;
```

```
10  conecR(4,:)=3*node2-2; conecR(5,:)=3*node2-1; conecR(6,:)=3*node2;
11  conecR(7,:)=3*node3-2; conecR(8,:)=3*node3-1; conecR(9,:)=3*node3;
12  conecR(10,:)=3*node4-2; conecR(11,:)=3*node4-1;
13  conecR(12,:)=3*node4;
14  ------------------------------------------------
15  KG=sparse_create(conecR,Ke);
```

Nota: Algumas linhas foram omitidas para fins de simplicidade.

Fonte: O autor (2023)

Algoritmo 8: sparse_create em elementos de tetraedro linear (simétrico)

```
1   % reshapes Ke for column wise collection
2   Ke=reshape(Ke,[78,nel]);
3   % connectivity matrix is reshaped to 4 x nel dimensions
4   conec=uint32(conec);
5   conec=conec';
6   ------------------------------------------------------------------
7   % redefines connectivity matrix 12 x nel for sparse_create format
8   conecR=uint32(zeros(12,nel));
9   conecR(1,:)=3*node1-2; conecR(2,:)=3*node1-1; conecR(3,:)=3*node1;
10  ...
11  ------------------------------------------------
12  % defines options for sparse_create command
13  opts.symmetric=1;     % general (0) or symmetric (1) matrix
14  KG=sparse_create(conecR,Ke,opts);
```

Nota: Algumas linhas foram omitidas para fins de simplicidade.

Fonte: O autor (2023)

Para a opção com simetria os ajustes são relativamente simples e exploram a característica de apenas 78 termos na matriz de rigidez de cada elemento. O Algoritmo 8 anteriormente indica os procedimentos necessários.

A Tabela *5* descreve diferentes algoritmos e funções que envolvem o uso de matrizes esparsas para cálculos e criação de matrizes. Algumas funções operam tanto na CPU quanto na GPU, ou em ambas. Além disso, a Tabela *5* menciona quando é necessário a presença de triplets (L, C, V) como parâmetros de entrada nas funções mencionadas neste trabalho. Como pode-se observar apenas função *sparse_create*, que opera com sem a necessidade de declarar os índices L, C para montar a matriz de rigidez global KG.

Tabela 5 Características das funções sparse, *fsparse*, *sparse2* e *sparse_create*

	CPU	**GPU**	**triplets L, C e V**	**algoritmo**
Sparse	✓	✓	✓	`KG=sparse(L,C,V,ndof,ndof);`
Fsparse	✓		✓	`KG=fsparse(L,C,V,ndof,ndof);`
sparse2	✓		✓	`KG=sparse2(L,C,V,ndof,ndof);`
sparse_create	✓			`KG=sparse_create(conecR,Ke,opts);`

2.2.8 Solver iterativo utilizando o gradiente conjugado na CPU e na GPU

Uma alternativa ao solucionador de sistema de equação direto surge com a utilização de métodos iterativos, tal como o método dos gradientes conjugados. Neste caso, a experiência do usuário é crucial para a definição de convergência e acurácia dos resultados. O consumo de memória é muitas vezes menor que o necessário para a utilização do operador barra invertida. No entanto, o processo de convergência pode ser lento, a depender do vetor de partida para busca de soluções. Em fenômenos transientes isto pode ser amenizado com a utilização da solução do instante anterior como partida para a próxima solução.

Para poucas iterações o método apresenta excelente desempenho. O Algoritmo 9 apresenta a forma de uso da função *pcg* do MATLAB. A matriz de rigidez global KG do tipo double, o vetor de carregamento F devido aos vetores de pressões do fluido, também do tipo double, a tolerância e a máxima iteração são fornecidas através das variáveis tol e maxit respetivamente e por fim o vetor de chute inicial para aumentar a velocidade de convergência , são importantes para resolver o sistema de equação ao utilizar a função pcg mencionada no Algoritmo 9.

Algoritmo 9: Método de gradientes conjugados pré-condicionados na CPU e na GPU

```
1   % solver with pcg
2   sol= pcg(KG,F,tol,maxit,[],[],chute);
```

Nota: Algumas linhas foram omitidas para fins de simplicidade.

Fonte: O autor (2023)

A possibilidade de aceleração por GPU também é uma vantagem dos solucionadores iterativos, visto que exigem pequenos requisitos de memória e viabilizam a alocação em placas aceleradoras. Na GPU na linha 2 do algoritmo 9 KG deve previamente está alocada na GPU através da função nativa do MATLAB *gpuArray* (veja a linha 6 do Algoritmo 5).

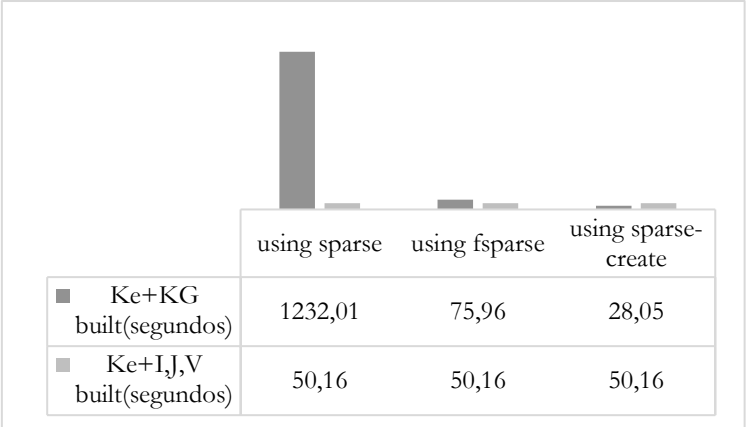
2.2.9 Solver iterativo pcgEigen e pcgEigenOMP na CPU

No MATLAB, a função nativa *pcg* descrita na seção anterior não é multi-*threaded*, não oferece a possibilidade de realizar o processamento em vários *threads* de execução *simultaneamente ,* que é uma grande desvantagem de eficiência computacional. As soluções iterativas *pcg*Eigen e pcgEigenOMP, ambas usam bibliotecas Eigen que faz essa interface de comunicação entre as rotinas escritas em c/c++(c-mex) com o programa MATLAB, porém a diferença entre as duas funções é no uso do Multi-*Thread* que utilizou a estrutura Open Multi-Processing(openMP).

A função em linguagem C é compilada no próprio MATLAB com compiladores do tipo MinGW, GCC, Clang ou Visual Studio e é utilizada com a extensão mex, que recebe parâmetros do próprio *workspace do* MATLAB. Todo esse processo é extremamente simples com a utilização da toolbox MATLAB Coder. O algoritmo 10 apresento o uso das funções *pcg_eigen* e *pcg_eigen_OMP*. Existem duas grandes vantagens comparativamente à função nativa do MATLAB:

1. A montagem da matriz de rigidez global se torna desnecessário o que é uma excelente vantagem;
2. os índices I e J podem ser do tipo inteiro que reduz a quantidade do uso da memória RAM. Justificando a observação (1) na
3. Figura *8*, pode-se observar o ganho em tempo considerável evitando para a função *sparse* devido a sua ineficiência ao construir a matriz KG que obteve um *speedup* de mais que 24 vezes comparativamente à forma tradicional que é (Ke + KG).

Figura 8 Comparação de tempo para as funções *sparse*, *fsparse* e *sparse_create* com construção do KG e sem a construção do KG na CPU



	using sparse	using fsparse	using sparse-create
Ke+KG built(segundos)	1232,01	75,96	28,05
Ke+I,J,V built(segundos)	50,16	50,16	50,16

Algoritmo 10: pcgEigen e pcgEigenOMP na CPU e GPU

```
% solver with pcg
if Flag==3
        sol=pcg_eigen(i,j,v,F,chute,maxit,tol);
elseif Flag==4
        sol=pcg_eigen_OMP(i,j,v,F,chute,maxit,tol)
End
```

Nota: Algumas linhas foram omitidas para fins de simplicidade.

Fonte: O autor (2023)

Vale lembra que as condições de contorno devem previamente prescritas diretamente nas posições i,j do valor v(i,j). Pode-se observar a forma de entradas tanto no uso do *pcg_eigen* como no *pcg_eigen_OMP* são idênticas.

3 METODOLOGIA DE ACOPLAMENTO E ACELERAÇÃO DO PROBLEMA

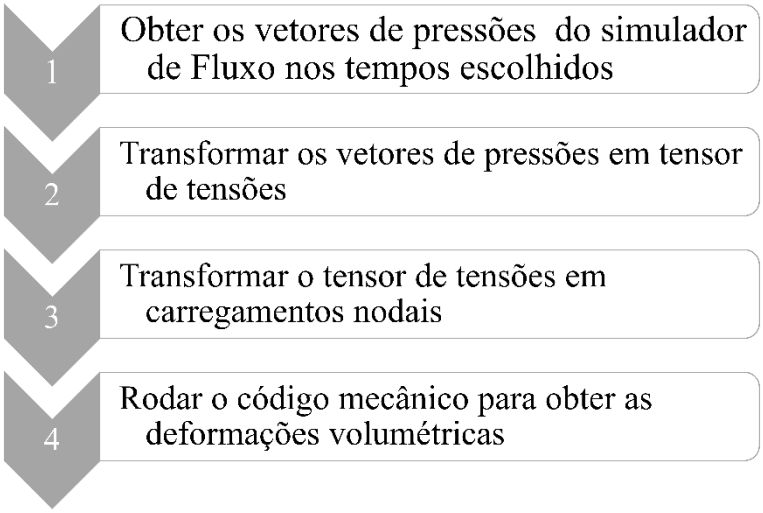
A metodologia adotada nesta pesquisa visa alcançar os objetivos específicos estabelecidos anteriormente para o desenvolvimento e proposição de estratégias eficientes destinadas a acelerar simulações computacionais em problemas de larga escala na área da geomecânica de reservatórios.

O processo de otimização do código mecânico através do método unidirecional e iterativo implementado neste estudo (*oneway*) baseia-se nas abordagens desenvolvidas por Inoue & Fontoura (2009) e Samier, Onaisi & Gennaro (2008), respetivamente. A principal diferença deste trabalho reside no seu foco exclusivo na influência do problema de fluxo no contexto da mecânica e na implementação de técnicas eficazes para simular e modelar de forma eficiente esses efeitos.

A Figura 9 ilustra as quatro etapas cruciais no processo de acoplamento, juntamente com as técnicas de otimização. Inicialmente, os vetores de pressão são obtidos por meio do software de fluxo monofásico desenvolvido internamente pelo grupo de pesquisa do Laboratório de Métodos Computacionais em Geomecânica, e não houve intervenção de otimização para o problema de fluxo neste trabalho. As três etapas subsequentes da Figura 9, que envolvem o problema mecânico, foram resolvidas por meio do programa de análise de tensões desenvolvido integralmente neste trabalho, utilizando o ambiente MATLAB.

As análises realizadas foram lineares e elásticas utilizando o Método dos Elementos Finitos, voltadas para problemas tridimensionais. No código desenvolvido, o usuário tem a flexibilidade de escolher entre a obtenção de resultados para um grau de liberdade específico ou para todos os graus de liberdade, dependendo da capacidade de processamento de sua máquina. Ao término da simulação, os resultados podem ser visualizados no módulo de pós-processamento do software *GID Simulation*.

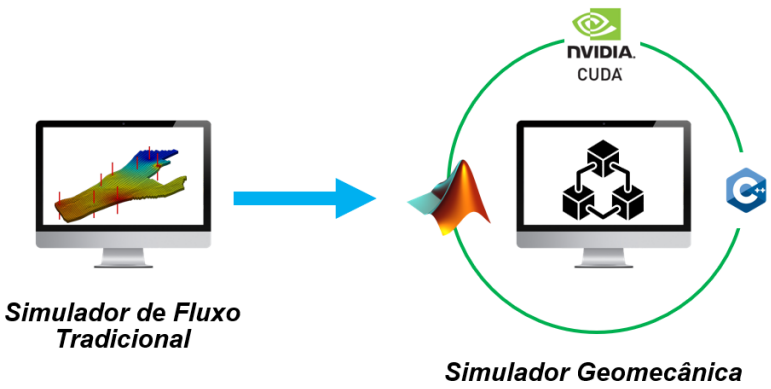
Figura 9 Fluxograma de acoplamento unidirecional (One-way)



Fonte: O Autor (2023)

A Figura 10 ilustra a comunicação entre o software de simulação de fluxo e o simulador geomecânico, a depender do objetivo da análise, esta comunicação pode ser feita em duplo sentido ou não, Outros autores desenvolveram o acoplamento parcial iterativa entre softwares comerciais para simulação de fluxo como Samier, Onaisi & De Gennaro (2008) que optaram para usar os simuladores comerciais EclipseTM para simulação de fluxo e AbaqusTM para as análises mecânicas. Neste caso os autores utilizaram como parâmetro de acoplamento multiplicadores de volume poroso.

Figura 10 Comunicação entre o software de Fluxo e o software mecânico



Fonte: O Autor (2023)

Neste trabalho, a decisão de adotar o acoplamento unidirecional (Fluxo > Mecânico) é adequada para o desenvolvimento de técnicas de otimização no problema mecânico, um futuro procedimento para a técnica *twoway*, as mesmas rotinas otimizadas servirão para atualizar o problema de fluxo através dos parâmetros de acoplamento. Isso contrasta com o trabalho de Inoue & Fontoura (2009), que desenvolveu uma expressão chamada pseudo-compressibilidade para levar em consideração os efeitos mecânicos na equação de fluxo, que é resolvida no método totalmente acoplado, com a resolução pelos simuladores convencionais de reservatório.

Os códigos foram desenvolvidos em linguagem MATLAB e são divididos em dois módulos: mecânico e escoamento monofásico em meios porosos. Assim, o objetivo principal nesta pesquisa foi estudar estratégias que melhoraram a eficiência computacional do problema mecânico. O problema mecânico é definido pelas formulações em Métodos dos Elementos Finitos (MEF) para elemento tetraédrico utilizando *Galerkin* discutidas na seção 2 pela equação(2.24). O módulo de escoamento também segue a formulação e é descrito pela equação (2.25) e é aproximado por tetraedros lineares com graus de liberdade de pressão. Os dois problemas são acoplados parcialmente de forma unidirecional (*Oneway*) mostrado seu fluxograma na Figura 11. Os detalhes sobre o acoplamento são apresentados na discussão sobre a solução do sistema de equações.

O código é dividido em etapas, que permitem otimização do problema mecânico de acordo com a seguinte estrutura:

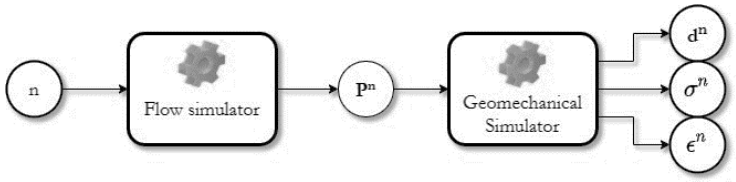
1. Etapa 1: montagem da matriz de rigidez dos elementos (Ke);
2. Etapa 2: construção da matriz de rigidez global (KG);
3. Etapa 3: acoplamento e solução do sistema de equações.

As rotinas exploram as características nativas do MATLAB v.2020a, além de bibliotecas existentes na comunidade MathWorks para operações em CPU. Neste trabalho foi utilizado a biblioteca LAPACK, que é um projeto financiado por muito tempo pelas empresas MathWorks e Intel. Ela é extensa e moderna utilizada para a resolução de sistemas de equações lineares simultâneas, implementada em Fortran desenvolvidos por vários autores.

Neste trabalho, Versões eficientes das funções em CPU são obtidas com a seguinte sequência:

1. Vetorização nativa;
2. Biblioteca MTIMESX para produto multidimensional (TURSA, 2020);
3. Funções *fsparse* (Engblom & Lukarski, 2016a), *sparse2* (Davis, 2019) e *sparse_create* (KROTKIEWSKI, DABROWSKI, 2010) para construção da matriz de rigidez global.

Figura 11 Fluxograma do acoplamento do tipo One-way



Fonte: o Autor (2023)

Todas as estratégias de aceleração desenvolvidas e apresentadas neste trabalho tanto na CPU quando na GPU tem por objetivo de acelerar a resolução do sistema de equação definido pela equação (3.1)

$$\int_{\Omega_e} \mathbf{B_u}^{eT} \mathbf{D} \mathbf{B_u}^{e} \mathbf{u}^e d\Omega_e = \int_{\Omega_e} \mathbf{B_u}^{eT} \alpha \mathbf{I} \mathbf{N}_p^e \mathbf{p}^e d\Omega_e - \int_{\Omega_e} \mathbf{N_u}^{eT} \rho_s \mathbf{g}\, d\Omega_e + \int_{\Gamma_N^{\mathbf{u}}} \mathbf{N_u}^{eT} \mathbf{t} d\Gamma_e \tag{3.1}$$

Ela pode ser escrita de maneira mais compacta da seguinte forma:

$$\overbrace{\boldsymbol{K_u^e}}^{Matriz\ de\ rigidez} \quad \overbrace{\boldsymbol{u^e}}^{deslocamento\ nodal} = \overbrace{F_{p^e}}^{primeiro\ termo} - \overbrace{F_g + F_t}^{\text{último dois termos}} \tag{3.2}$$

No contexto do acoplamento *oneway*, a equação (3.1) é executada em unidade de processamento escolhida pelo usuário, que pode ser na CPU ou GPU, em intervalo de tempo previamente escolhido do simulador de fluxo. Essa resolução envolve a utilização dos vetores de pressão do tempo atual e do tempo anterior. A matriz de rigidez $\boldsymbol{K_u^e}$ para esse problema foi calculada e armazenada após o primeiro processo de montagem, permanecendo inalterada ao longo do tempo. Na seção 2.2, foram apresentadas as técnicas empregadas neste estudo para otimizar a montagem da matriz de rigidez, tanto em nível local quanto global.

Como mencionado anteriormente, a abordagem de acoplamento *oneway* é realizada convertendo as pressões originadas no simulador de fluxo em cargas nodais, que são posteriormente introduzidas no simulador geomecânico. A pressão dos poros $\widetilde{F_{p^e}}$, obtida a partir do simulador de fluxo para cada elemento em cada tempo, foi transformada em carga nodal utilizando as funções de forma dos elementos finitos, conforme apresentado na Equação (3.5)

$$F_{p^e} = \mathbf{B}^T \boldsymbol{D_p} \mathbf{B}\ \mathbf{Ve} \tag{3.3}$$

$$G_{p^e} = \boldsymbol{D_p} \mathbf{B} \tag{3.4}$$

$$F_{p^e} = \mathbf{B}^T G_{p^e} \mathbf{Ve} \tag{3.5}$$

sendo

Ve: o volume do elemento,

G_{p^e} :o tensor de variação de pressão.

Para este problema, o tensor de pressão é formado apenas pelas tensões normais como descrito na equação (3.6)

$$G_{p^e} = [\boldsymbol{dp^e}\ \ \boldsymbol{dp^e}\ \ \boldsymbol{dp^e}\ \mathbf{0}\ \mathbf{0}\ \mathbf{0}] \tag{3.6}$$

sendo

$\boldsymbol{dp^e}$ **:** o vetor de variação de pressão, calculada através da equação (3.7)

$$\boldsymbol{dp^e} = P^{n+1} - P^n \tag{3.7}$$

P^{n+1} e P^n são os vetores de pressões nos tempos atuais e anteriores respectivamente.

4 RESULTADOS

Neste capítulo são apresentados os dois primeiros casos de Dean et al. (2006) para validar o esquema de acoplamento *oneway* desenvolvido nos capítulos anteriores deste trabalho, considerando o problema de fluxo monofásico em reservatório elástico-linear e avaliando a subsidência da superfície e compactação do reservatório durante a extração do fluido (produção primária). O caso 3 do Dean et al. (2006) foi utilizado para avaliar a eficiência computacional a implementação do problema mecânico desenvolvida neste trabalho com malhas de milhões graus de liberdade.

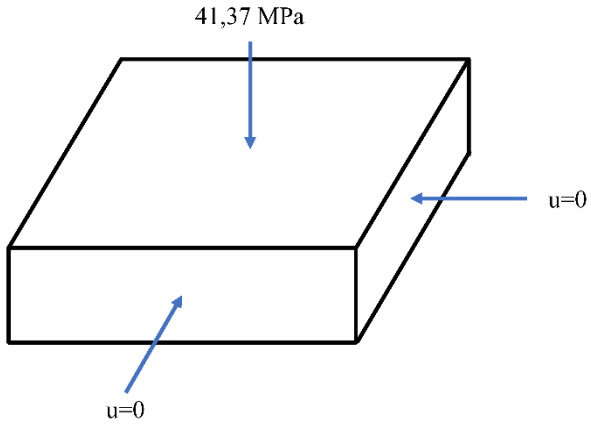
A versão do MATLAB utilizada neste trabalho foi a 9.8 (R2020a), em um computador com sistema operacional Windows 10, Intel(R) Core (TM) i7-8700 CPU 3.19 GHz, com 32 GB de memória RAM, 6 processadores; e uma GPU da NVIDIA TITAN Xp com 12 GB de memória VRAM.

Durante este trabalho foram utilizados os softwares MATLAB e GID Simulation para visualizar os resultados. Todos os gráficos de comparação foram produzidos por códigos próprios implementados no MATLAB, as malhas de elementos finitos foram visualizadas via GID.

4.1 CASO 1 DO DEAN ET AL. (2006)

O primeiro caso do trabalho do Dean et al. (2006) possui uma largura e comprimento igual de 670,56 metros, bem como uma espessura de 60,96 metros, conforme ilustrado na Figura 12 . É importante notar que este reservatório é caracterizado por um confinamento total em todas as suas faces, com exceção da face superior, onde está sujeito a uma tensão confinante de 41,37 MPa.

Figura 12 Caso 1 do Dean et al (2006)



Fonte: O Autor (adaptado do Dean e al, 2006)

Este reservatório representa um componente crítico no campo da geomecânica de reservatórios, e sua análise desempenha um papel fundamental na compreensão das interferências das condições de contorno e comportamentos geomecânicos associados a esse cenário específico

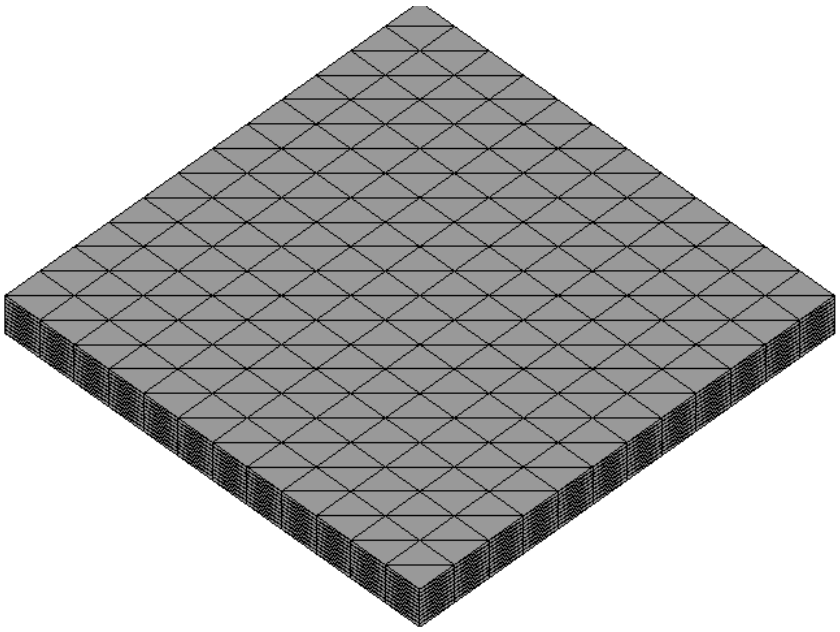
A grade de discretização utilizada consiste em elementos tetraédricos distribuídos em uma malha de tamanho 12 x 12 x 10. Essa malha nos oferece uma representação minuciosa e detalhada da geometria do reservatório mencionado na Tabela 6.

Tabela 6 Características geométricas do problema

Comprimento na direção x (m)	670,56
Comprimento na direção y (m)	670,56
Comprimento na direção x (m)	60,96
Divisões em x	12
Divisões em y	12
Divisões em z	10
Número de elementos	8640
Número de nós	1859

A malha da Figura 13 possui 1859 nós e 8640 elementos como foi descrito nas duas últimas linhas da Tabela *6* acima. Foram utilizados elementos tetraédricos linear de 4 nos por elementos.

Figura 13 Malha de elementos finitos do caso 1 do Dean et al. (2006)



Fonte: O Autor (2023)

As caraterísticas da rocha e do fluido contido no reservatório são mostradas na Tabela *7*. A permeabilidade horizontal (direções x e y) do reservatório é de 50 mD, enquanto a permeabilidade vertical (direção z) é de 5 mD. A porosidade do reservatório é de 20%, o coeficiente de Poisson é de 0,3 e o módulo de elasticidade é de 68,95 MPa.

A pressão de referência no reservatório é mantida a 20,68 MPa, medida a uma profundidade de 1828,8 metros. Um poço produtor, estrategicamente posicionado no centro do reservatório como mostrado na Figura 14, extrairá uma vazão constante de 15.000 barris por dia equivalente a uma taxa de 0,0276 m^3/s. Essa vazão constante introduz mudanças significativas na pressão do reservatório, o que é crucial para o entendimento da interação entre as condições de contorno e o comportamento do fluido.

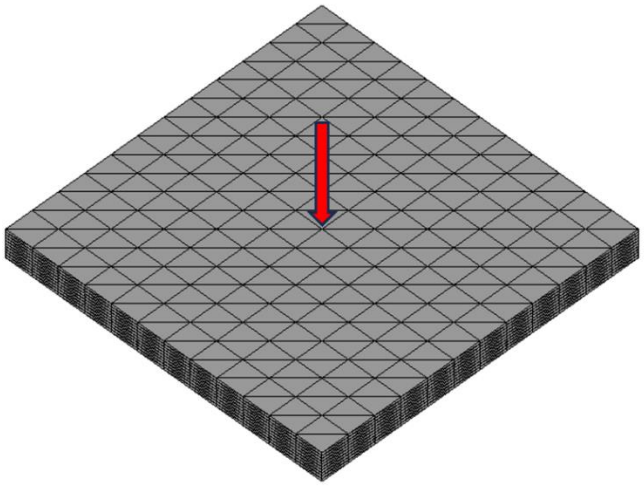
Tabela 7 Propriedades físicas do problema de fluxo e mecânico

Permeabilidade x (mD)	50,0

Permeabilidade y (mD)	50,0
Permeabilidade z (mD)	5,0
Porosidade (%)	20,0
Poisson	0,3
Modulo de Young (MPa)	68,95

Fonte: O Autor (2023)

Figura 14 localização do poço produtor



Fonte: O Autor (2023)

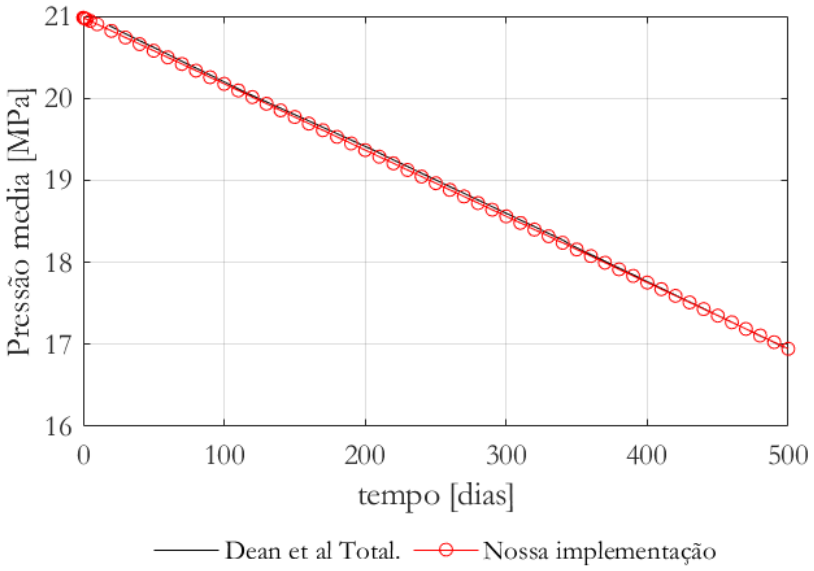
Os resultados de pressões medias e deformações do reservatório obtidos a partir dessa simulação foram comparados com a solução totalmente acoplada alcançada através do artigo do Dean et al. (2006). Conforme os resultados das Figura 15 e Figura 16 da técnica *oneway* implementada nesta pesquisa, produz resultados que se aproximaram substancialmente da solução totalmente acoplada do Dean et al (2006).

É importante destacar que os resultados do problema de fluxo de reservatório apresentados foram gerados por outro software. No âmbito deste estudo, o principal foco de atenção foi direcionado exclusivamente para o problema mecânico.

Os resultados das pressões médias do reservatório, conforme ilustrado na Figura *15*, indicam que o código de simulação de fluxo produz resultados satisfatórios e que serão utilizados como entrada para a análise do problema mecânico. Essa observação

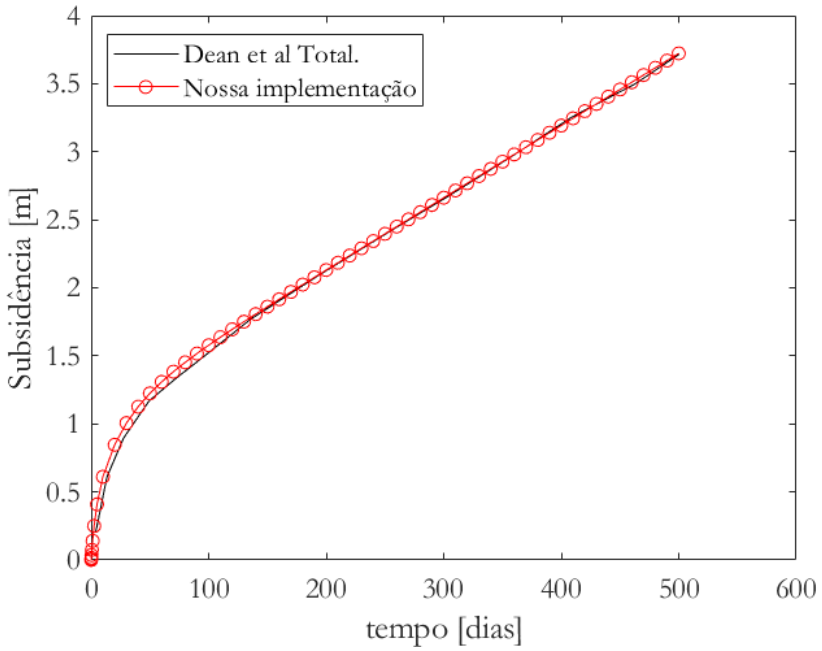
enfatiza a confiabilidade dos resultados gerados pelo software de simulação de fluxo, que servirá como uma base sólida para a investigação mecânica em questão.

Figura 15 Pressão média no reservatório



Fonte: O Autor (2023)

Figura 16 Deslocamento no nó central do topo do reservatório



Fonte: O Autor (2023)

Com base na Tabela *8* fornecida a seguir, mostra-se que os resultados de deslocamento vertical apresentado na Figura 16 acima, chamado de subsidência por estar no topo da malha, mostra a coincidência dos resultados do código desenvolvido com a referência Dean et al (2006). Os resultados são muito semelhantes em todos os tempos avaliados (190 dias, 250 dias e 500 dias). A diferença percentual entre os valores é de 0,00% em todos os casos, o que indica uma excelente concordância entre os resultados das duas implementações.

Tabela 8 Comparação dos resultados de deslocamento do nó central do reservatório

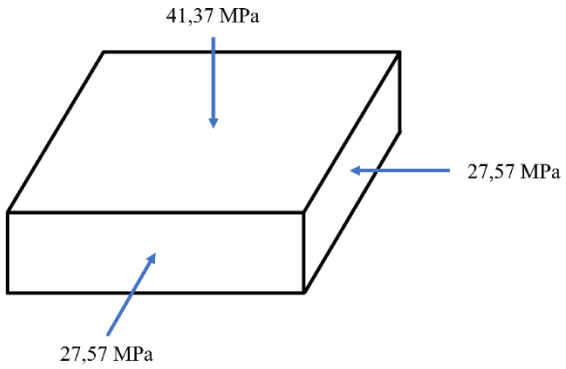
Tempo(dias)	**Desl. Vertical Nossa implementação**	**Desl. Vertical Dean et al, 2006**	**Diferença (%)**
190	2,07	2,07	0,00
250	2,40	2,40	0,00
500	3,72	3,72	0,00

4.2 CASO 2 DO DEAN ET AL. (2006)

Para validar a ferramenta de análise geomecânica aqui desenvolvida foi modelado o caso 2 do Dean et al (2006) que traz outro cenário diferente do anteriormente analisado.

Neste novo cenário, a simulação envolve a aplicação de uma tensão confinante horizontal de 27,57 MPa, o que representa uma mudança fundamental nas condições de contorno em relação ao caso anterior. Nas simulações anteriores, as condições de contorno eram de deslocamentos normais às faces nulos. Essa mudança nas condições de contorno é de grande relevância, pois pode afetar significativamente o comportamento do reservatório e, consequentemente, a forma como ele responde à tensão aplicada.

Figura 17 Caso 2 do Dean et al (2006)



Fonte: O Autor (adaptado do Dean e al, 2006)

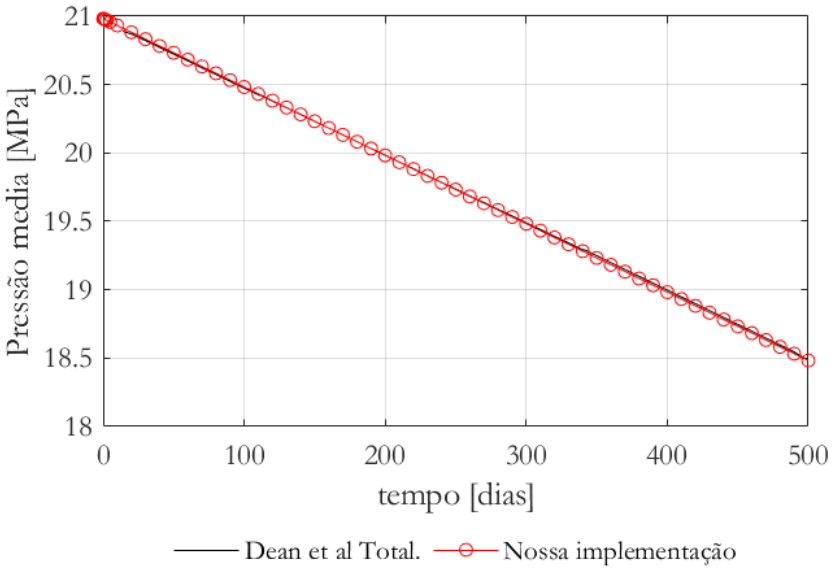
Neste problema, as características geométricas e físicas do reservatório permanecem inalteradas em comparação com o "Caso 1". A única modificação significativa está na aplicação das tensões confinantes horizontais e na eliminação das condições de deslocamento nulo nas faces do reservatório.

Isso significa que as dimensões, geometria, propriedades físicas do material e quaisquer outros fatores relacionados à estrutura do reservatório permanecem consistentes entre os

dois casos. Este novo cenário ajuda a isolar o efeito das tensões horizontais e a entender melhor seu papel no comportamento do reservatório.

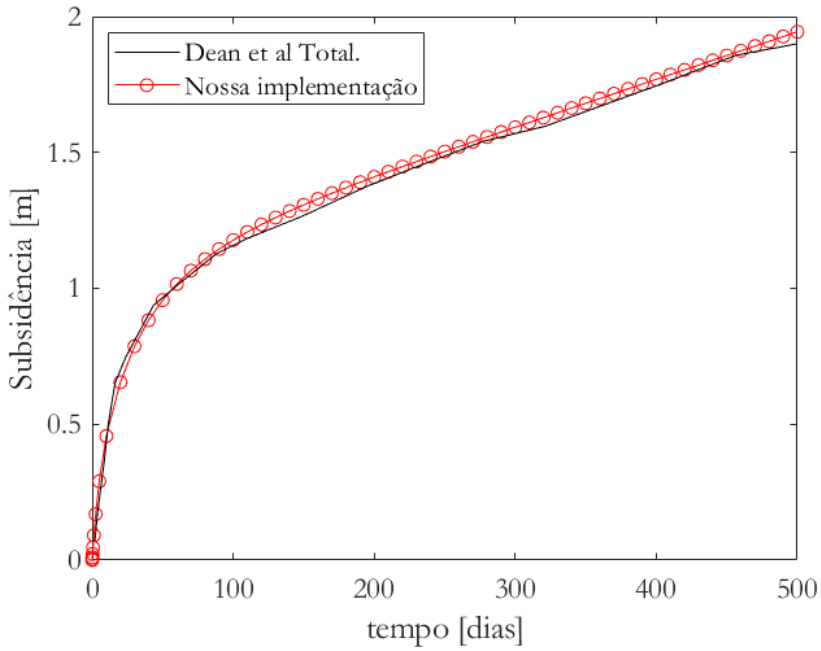
As Figura 18 e Figura 19 mostram consistência entre os resultados do código desenvolvido e os resultados do Dean et al (2006).

Figura 18 Pressão média no reservatório



Fonte: O Autor (2023)

Figura 19 Deslocamento no nó central do topo do reservatório

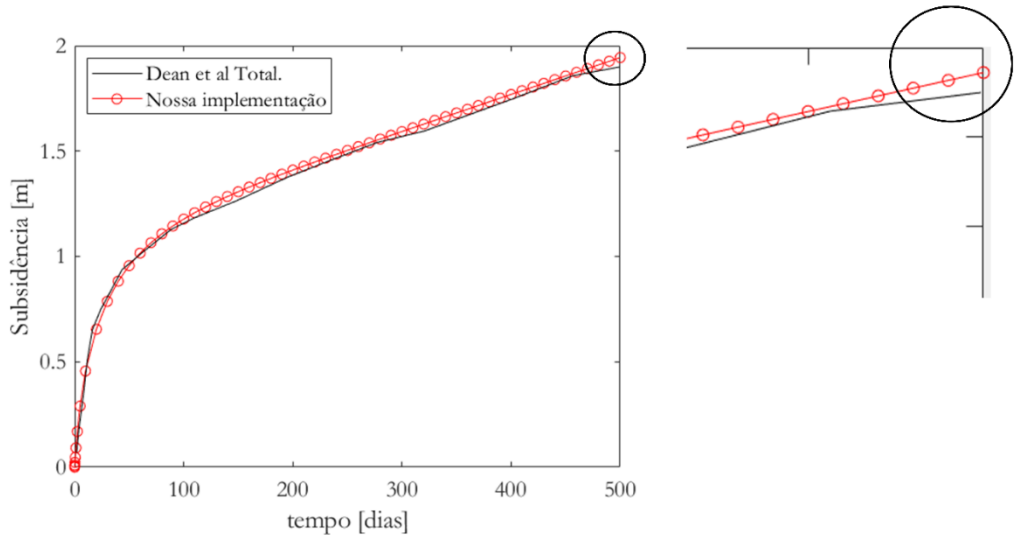


Fonte: O Autor (2023)

Com relação ao primeiro problema, apresentado anteriormente, este caso apresentou pequenas diferenças entre os dois resultados. Como referência de modelo mais preciso, são comparados os resultados do modele totalmente acoplado do Dean para este

mesmo problema. Pode-se observar na Figura 20, uma pequena diferença entre os dois modelos, mais expressiva nos tempos de 190 e 500 dias.

Figura 20 Diferença observada nos dois resultados.



Fonte: O Autor (2023)

Os resultados da Tabela 9 mostra a concordância entre os deslocamentos verticais obtidos pela aqui desenvolvida (*oneway*) e pela referência "Dean et al, 2006" em diferentes instantes de tempo durante a simulação. Aqui estão algumas observações e considerações importantes sobre esses resultados:

- Em geral, os resultados do código desenvolvido estão em concordância com os da referência "Dean et al, 2006". Isso é evidenciado pelas pequenas diferenças percentuais observadas em todos os pontos de tempo.
- A diferença percentual máxima de 5,30% neste ponto de tempo pode ser vista como um desvio ligeiramente maior entre os resultados, mas ainda é uma diferença relativamente pequena, o que mostra que o cenário é de fraco acoplamento entre os problemas de fluxo e mecânicoEsses resultados reforçam a validação e a confiabilidade dos resultados neste novo cenário. Isso é fundamental para garantir que o a ferrramenta numérica aqui desenvolvida seja capaz de reproduzir resultados precisos e confiáveis em cenários semelhantes aos da referência.

Tabela 9 Comparação dos resultados de deslocamento do no central do reservatório

Tempo(dias)	**Desl. Vertical Nossa implementação**	**Desl. Vertical Dean et al, 2006**	**Diferença (%)**
190	1,39	1,32	5,30
250	1,50	1,47	2,04
500	1,94	1,90	2,10

A seguir apresenta-se o problema 3 do trabalho do Dean et al. (2006), que servirá para avaliar a eficiência computacional do código mecânico, objetivo principal deste trabalho.

4.3 APLICAÇÃO AO CENÁRIO 3 DE DEAN ET AL (2006)

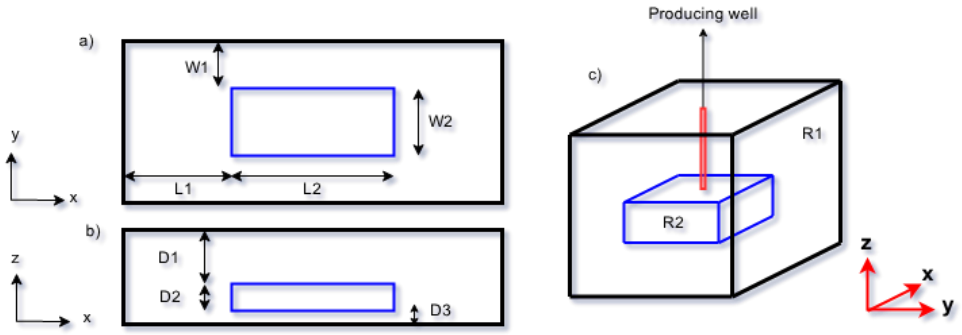
O problema do caso 3 apresentado por Dean et al (2006) é clássico na indústria de óleo e gás em que a produção primária traz efeitos geomecânicos consideráveis nas rochas reservatório e adjacentes, que resultam na subsidência da superfície e a compactação do reservatório. Nesta pesquisa, o foco principal foi exclusivamente na solução do problema mecânico, analisando a eficiência computacional dos três segmentos do código:

1. Cálculo das matrizes de Rigidez locais;
2. Montagem das Matrizes Globais na CPU e GPU;
3. Cálculo do sistema de equação (2.22);

Para efeito de comparação posterior foi mantida as mesmas caraterísticas físicas e geométricas do problema mencionado acima. A grande diferença é que a geometria foi refinada para milhões de graus de liberdade. Pode-se observar que a região R2 é o reservatório, que se localiza na região central do plano(x,y) da Figura 21(a) e tem um

poço produtor passando em todas as suas camadas, na sua região central Figura 21(c). A região R1 se compõe pelas rochas adjacentes ao reservatório.

Figura 21 Geometria da malha. (a) vista do topo (b) vista frontal (c) vista 3d



Fonte: O Autor (2023)

No contexto da geomecânica, a Figura 21(c) representa o problema em que a base e as faces não apresentam deslocamentos, ou seja, estão fixas. Para o problema de fluxo (também na Figura 21(c)), as condições iniciais são definidas da seguinte maneira: no topo, não há pressão exercida pelo fluido, e o gradiente aproximado de pressão é de $9{,}92 * 10^{-03}\ MPa/m$. Todo o carregamento aplicado ao problema mecânico é devido à diferença de pressão causada pela produção no reservatório. O poço está localizado no centro do reservatório e atravessa todas as suas camadas, com uma vazão de 50000 bbl/dia. A Tabela 10 apresenta as propriedades geométricas do problema em estudo, com as dimensões expressas em metros.

Para avaliar a eficiência computacional do código desenvolvido para o problema geomecânico, foram utilizadas as malhas numeradas de 1 a 4, conforme listado na Tabela *11*. A malha 0 representa a malha original descrita no artigo de referência do Dean et al (2006). As propriedades mecânicas do problema envolvem módulos de 6895 MPa para as rochas não reservatórias e 68,95 MPa para as rochas do reservatório. Ambas as rochas possuem um coeficiente de Poisson de v=0,25.

Tabela 10 Propriedades geométricas do problema 3 do Dean et al (2006)

Comprimentos das regiões em metros			
Lenght	L1=6098	L2=6705,8	-
Width	W1=3048	W2=3352	-

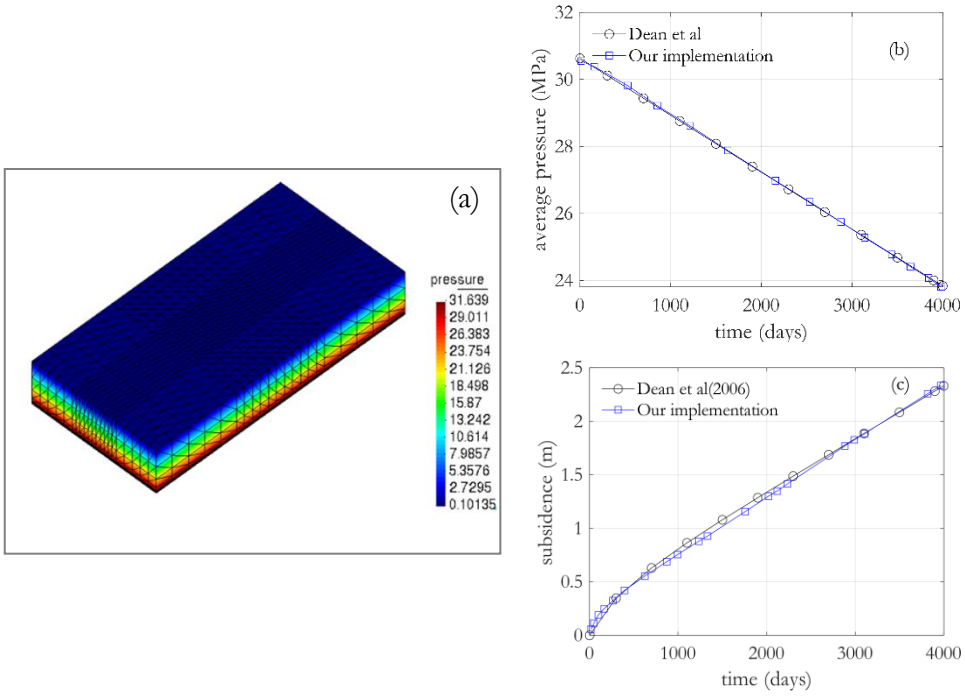
Depth	D1=3048	D2=3048	D3=61
Dimensões totais			
Direções	R1		R2
x	2*L1+L2=18897,8		L2=6705,8
y	2*W1+W2=9448,8		W2=3352
z	D1+D2+D3=3185,2		D2=76,2

Tabela 11 Malha de elementos finitos para elementos tetraédricos lineares

	Configuração do problema	
Malhas	nelem	ndof
0	31752	188976
1	1843200	1010919
2	5171328	2822079
3	8323200	4534959
4	14709888	8003151

Os gráficos da Figura 22(b) e Figura 22(c) ilustram a distribuição de pressão média no reservatório e o deslocamento do ponto central no topo do reservatório ao longo do tempo de produção até 4000 dias respectivamente. Os gráficos mostram para a malha 0 da Tabela 11 que o código desenvolvido apresenta resultados similares aos resultados que foram apresentados no artigo do Dean et al(2006).

Figura 22 (a) Campo de pressão no reservatório, (b) gráficos de pressão média (c) compactação do reservatório



Fonte: O Autor (2023)

Daqui por diante o foco será exclusivamente na avaliação da eficiência computacional do problema geomecânico para as malhas de 1 a 4. Na Tabela 12 apresenta-se um resumo das siglas com sua definição das técnicas utilizadas para construir as matrizes de rigidez locais Ke tanto na CPU como na GPU e na Tabela *13* os tempos ao longo dos quais foram feitas todas as avaliações sobre a eficiência computacional.

Tabela 12 Siglas para construções das matrizes dos elementos Ke

Sigla	**Definição**
Imp	Montagem de Ke Implicitamente na CPU
Exp	Montagem de Ke Explicitamente na CPU
ExpS	Montagem de Ke Explicitamente na CPU utilizando a simetria
ImpGPU	Montagem de Ke Implicitamente na GPU
ExpGPUS	Montagem de Ke Explicitamente na GPU utilizando a simetria

Na Tabela *13* apresenta-se os tempos pelas quais foram obtidas as pressões nodais do problema de fluxo resolvido no domínio do reservatório que foram impostas ao problema geomecânico.

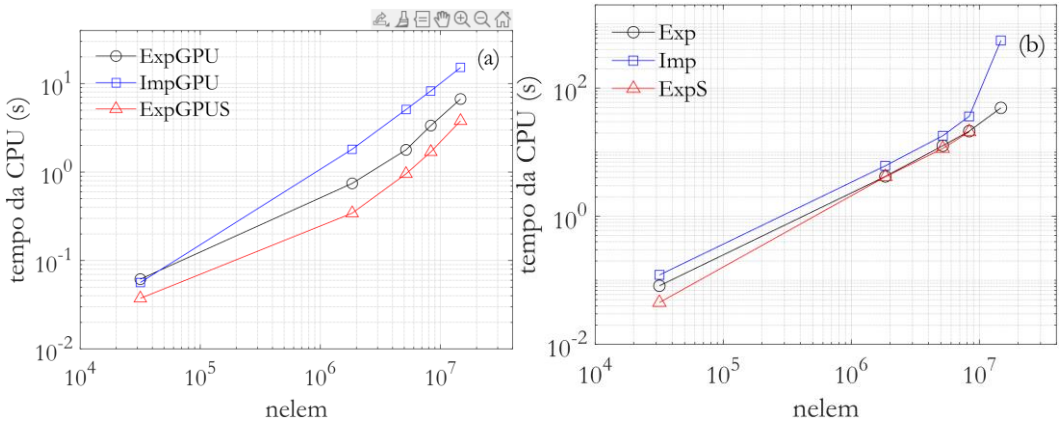
Tabela 13 Passos de tempo do cálculo do problema geomecânica

Passos	1	2	3	4	5	6	7
Tempos(dias)	13	157	527	853	1214	1630	2155
Passos	8	9	10	11	12	13	14
Tempos(dias)	2534	2878	3140	3438	3646	1630	2155

4.3.1 Cálculo das matrizes de rigidez locais Ke

Analisando os gráficos do tempo de construção das matrizes de rigidez Ke e comparando as técnicas utilizadas percebe-se que a construção explicita na GPU, aproveitando a simetria, é a mais eficiente como mostra a Figura 23(a). Além de analisar os gráficos globalmente, também é interessante fazer análises locais atentando para cada técnica na CPU e na GPU. A melhor maneira de construir Ke na CPU é pela técnica ExpS. A técnica ExpGPUS é a mais eficiente computacionalmente na GPU. Para esse problema, a técnica menos eficiente de construir a matriz Ke é a técnica Imp Figura 23(b).

Figura 23 Tempo de construção das matrizes de rigidez locais Ke (a) na GPU

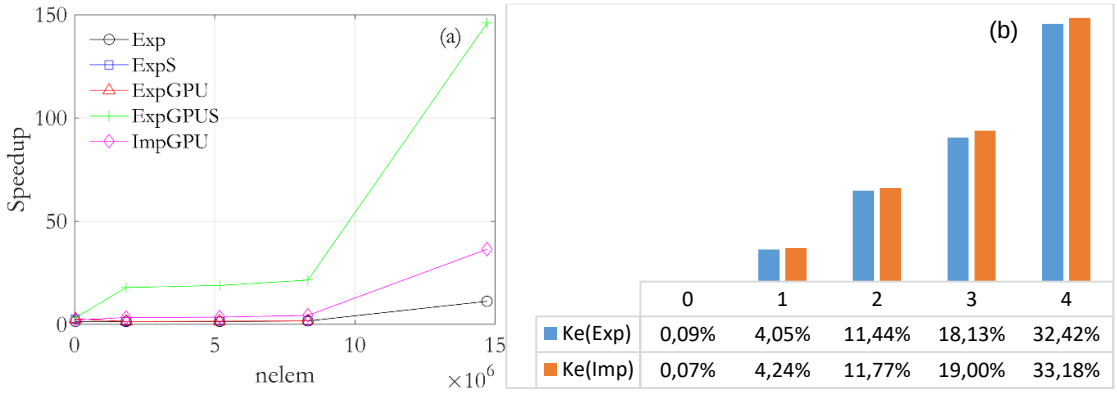


Fonte: O Autor (2023)

Todas as reflexões feitas anteriormente podem ser vistas no gráfico da Figura 24(a), em que a técnica ExpGPUS foi a mais eficiente, tendo um *SpeedUp* de 140 vezes mais rápido em relação ao Imp. Outra observação interessante que pode ser feita na

Figura 24 (b) é que o consumo de memória para o armazenamento dos vetores é bem parecidos tanto para o cálculo das matrizes montadas explicitamente e implicitamente.

Figura 24 *Speedup* da construção das matrizes dos elementos Ke em relação ao Imp (b) Comparação de Porcentagem da memória utilizada por Ke montada explicitamente e implicitamente.



	0	1	2	3	4
Ke(Exp)	0,09%	4,05%	11,44%	18,13%	32,42%
Ke(Imp)	0,07%	4,24%	11,77%	19,00%	33,18%

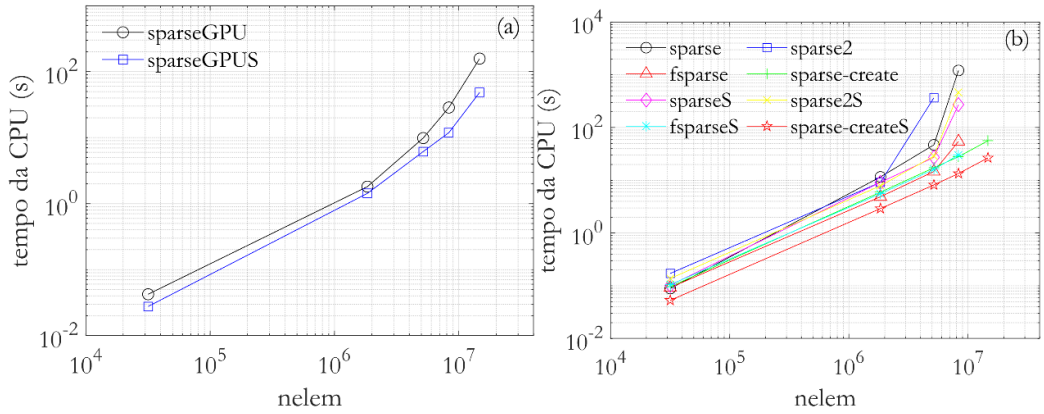
Fonte: O Autor (2023)

4.3.2 Montagem da matriz de rigidez global KG

Outro segmento crucial no estudo é a montagem das matrizes de rigidez global, ao qual foram utilizadas várias técnicas descritas previamente. Abaixo, pode-se observar na Figura 25(a) que a técnica *sparse*GPUS foi a mais eficiente na GPU. Vale registrar que houve um trecho no código com um *loop* para montagem de KG na GPU, O algoritmo deste procedimento foi apresentado na Tabela *4*.

Na CPU, o *sparse*_create teve um melhor desempenho (Figura 25 (b)) pois ela consome menos memória RAM devido à ausência dos vetores de índices de linha, coluna e valores (**I, J, V**) (Figura 26(b)), vetores contendo os índices das linhas, colunas e valores das matrizes locais Ke, mas que são necessários para as outras técnicas. Por outro lado, pode-se observar na Figura 25 (b) na CPU que a estratégia *sparse*2 teve um consumo de memória RAM bastante expressivo. O limite da capacidade da memória utilizando esta função foi até a casa de 5 milhões de elementos, correspondente à malha 2 da Tabela 11. As células vazias da Figura 26(a) significam que não se obteve resposta devido à falta de memória.

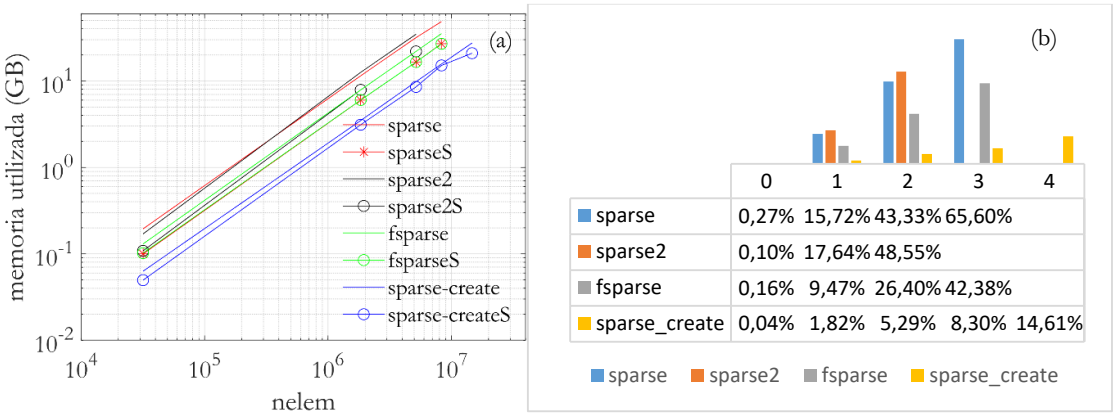
Figura 25 Tempo de construção da matriz de rigidez global KG:



Fonte: O Autor (2023)

Os comportamentos dos gráficos da Figura 25(b) pode ser justificado pelo consumo de memória da montagem do KG mostrado na Figura 26(b) juntamente com a memória do Ke mostrado na Figura 23(b). Eles mostram claramente que houve um consumo excessivo na montagem de KG e Ke somada.

Figura 26 (a) memoria ocupada vs ndof (b) Porcentagem de memória consumido por KG frente à memória disponível

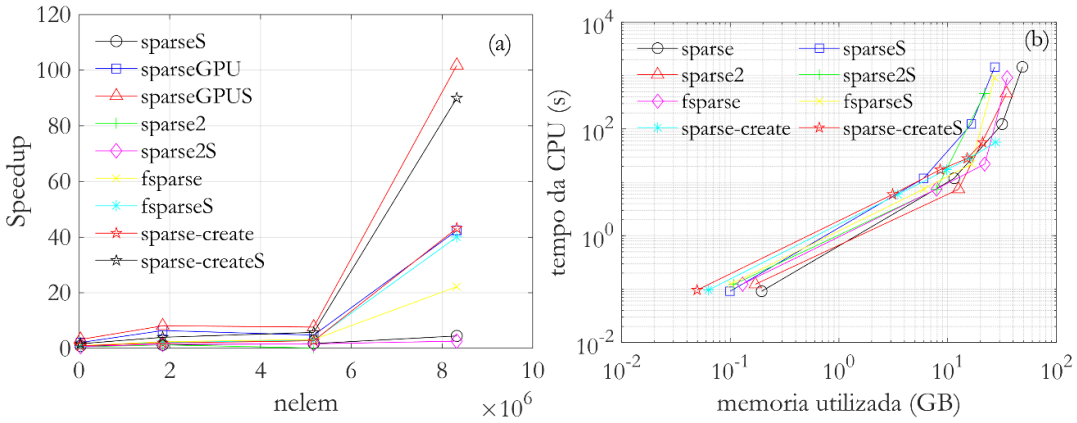


Fonte: O Autor (2023)

A Figura 27(a) mostra a técnica mais eficiente através do *SpeedUp* em relação à técnica *sparse* na CPU. Pode-se observar também que a montagem do KG na GPU utilizando a função nativa *sparse* do MATLAB é a mais eficiente. Percebe-se as

inclinações das curvas *sparse*GPU e *sparse-createS* são bastante próximas, e mostra a eficiência da função *sparse-createS para CPU.*

Figura 27 (a)- Speedup da montagem da matriz global KG em relação ao sparse na CPU (b) consumo de memória versus o tempo da CPU

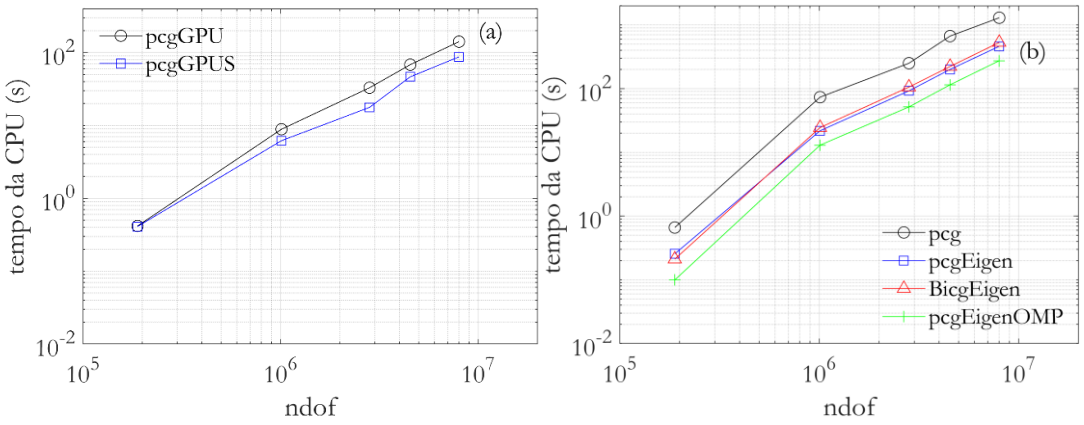


Fonte: O Autor (2023)

4.3.3 Cálculo do tempo de solução do sistema de equações lineares

Por fim, um último segmento que foi objeto de análise e estratégias de solução do sistema de equação lineares, ponto crucial que necessitou uma atenção especial e que as técnicas citadas foram empregadas em versões CPU e GPU. Primeiro pode-se observar na Figura *28*(a) que na GPU, a melhor maneira de resolver o problema é utilizar a simetria na montagem do KG e posteriormente a utilização da função nativa do MATLAB *pcg* com vector de deslocamento inicial na parte da resolução de sistema de equação. Pode-se observar na Figura 28 (a) com malhas maiores a simetria na montagem do KG tem influência na solução do sistema. Esta observação vale tanto para solução de sistema na CPU quando na GPU.

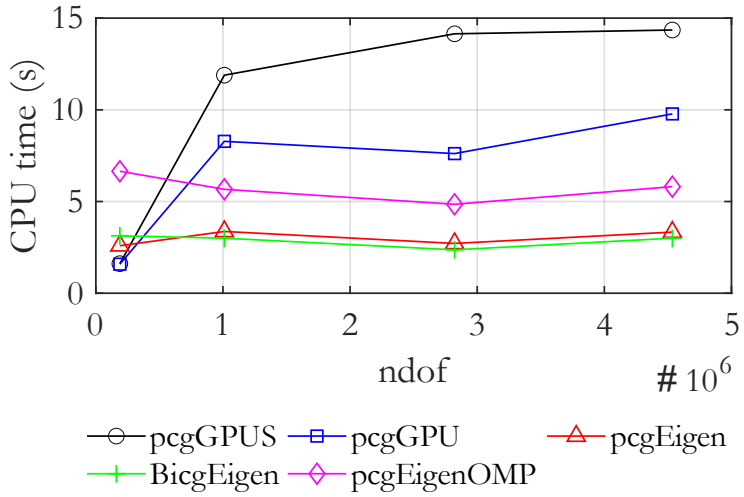
Figura 28 comparação dos tempos de processamento dos solvers



Fonte: O Autor (2023)

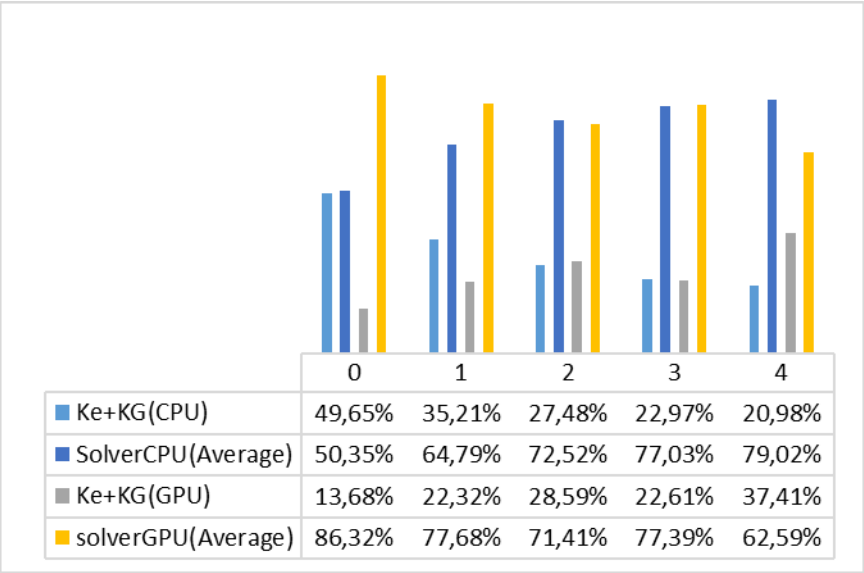
Na versão da CPU, a técnica mais eficiente foi pcgEigenOMP. O tempo de solução apresentada nos gráficos é médio, levando em consideração os 14 passos de tempo apresentados anteriormente na Tabela 13. A Figura *29* apresenta o *speedup* na solução do problema para a malha 4 apresentada na Tabela 11. Pode-se observar que a técnica PCGGPUS teve o melhor desempenho comparativamente a outras técnicas, ela obteve um *speedup* de mais que 14 vezes comparando com a técnica PCG na CPU.

Figura 29 speedup em relação aos tempos de solução utilizando pcg



Fonte: O Autor (2023)

Figura 30 Comparativas em Porcentagem no tempo total da soma



	0	1	2	3	4
Ke+KG(CPU)	49,65%	35,21%	27,48%	22,97%	20,98%
SolverCPU(Average)	50,35%	64,79%	72,52%	77,03%	79,02%
Ke+KG(GPU)	13,68%	22,32%	28,59%	22,61%	37,41%
solverGPU(Average)	86,32%	77,68%	71,41%	77,39%	62,59%

Fonte: O Autor (2023)

A Figura 30 traz informações interessantes sobre as comparativas dos tempos de construção de Ke e Kg somados frente ao tempo de execução na solução de sistema de equação do problema.

A seguir algumas observações a serem feitas para a compreensão da Figura 30. A primeira é que o gráfico traz informação para as 5 primeiras malhas nomeadas de 0 a 4. A segunda observação é que para cada malha é avaliada a participação do tempo de montagem do Ke e KG juntas frente ao solver tanto na CPU quanto na GPU. A terceira observação é a porcentagem do tempo somado de Ke e KG que no total decresce a médica que aumenta o tamanho da malha. Na GPU essa observação é diferente: com malhas maiores a porcentagem do tempo da solução tem tendência a decrescer para um determinado tamanho e a presença de loop na construção de KG compromete a eficiência da montagem, o que pode justificar esse comportamento diferentemente da construção na CPU.

Para problemas lineares e elásticos na CPU, onde não há necessidade da reconstrução da malha global, a montagem eficiente em tempo recorde parece não trazer grandes vantagens para malhas de grande escala, no entanto, para problemas não-lineares ou problemas onde a reconstrução das matrizes Ke e KG se torna obrigatória, o tempo de

construção de uma Ke e KG nos casos de *sparse-createS* na CPU e *sparseS* na GPU podem ser interessantes com relação ao tempo total de análise.

Finalmente, uma última consideração recai sobre os intervalos de tempo. Em problemas desse tipo, nos quais os passos temporais podem variar em tamanho, a porcentagem do tempo médio dedicado à resolução influenciará diretamente no tempo total necessário para solução.

5 CONCLUSÕES E SUGESTÕES DE TRABALHOS FUTUROS

Este trabalho apresentou alternativas de simulação do problema geomecânico aplicado a reservatórios com milhões graus de liberdade. Foram avaliadas estratégias tanto na CPU como na GPU, onde problemas de subsidência e compactação de reservatório foram resolvidos via Método dos Elementos Finitos, como modelos elástico-linear, utilizando o ambiente de programação do MATLAB. Três segmentos de um programa de elementos finitos foram avaliados em termo da eficiência computacional:

1. O cálculo das matrizes de rigidez locais Ke;
2. A montagem da matriz de rigidez global KG;
3. Alternativas de solução do problema.

5.1 CONCLUSÕES

Na primeira parte de análise, o código foi capaz de calcular na GPU as matrizes de rigidez Ke para uma malha de 14,709,888 de elementos em um tempo de 3.79 segundos, técnica nomeada como ExpGPUS. A grande exploração desta técnica foi o aproveitamento da simetria da matriz Ke por conta da adoção do comportamento elástico-linear do material.

Vale observar na CPU a melhor técnica é a montagem explícita utilizando a simetria. O ExpS montou a mesma malha em 49.25 segundos. Comparando estas duas técnicas o ExpGPUS obteve um speedup local (montagem das matrizes Ke) de cerca de 13 vezes. Globalmente, o ExpGPUS teve um speedup de mais que 140 comparando com a montagem na CPU implícita.

Outro ponto muito importante foi a montagem da matriz global KG. A melhor técnica observada foi a sparseGPUS, que utiliza a função nativa do MATLAB *sparse* na GPU. O código geomecânico desenvolvido foi capaz de montar, utilizando esta técnica

sparseGPUS em 48.47 segundos, na CPU. O tempo foi bem superior que foi de 1,459.20 segundos.

Na CPU a melhor técnica para montar a matriz global KG foi a utilização da função *sparse_create* que obteve um tempo de 56.24 segundo para a malha de 14,709,888 de elementos.

O último segmento analisado foi o tempo de solução do problema 3 do Dean com 14,709,888 de elementos para as diferentes estratégias de solução do sistema apresentadas anteriormente. A melhor estratégia observada em termo de melhor desempenho computacional foi o gradiente conjugado na GPU com KG montada utilizando simetria, conhecida com pcgGPUS que resolveu esta mesma num tempo de 20.40 minutos para os 14 passos de tempos com uma média de 1.457 minutos para cada passo de tempo. Na CPU a melhor técnica foi a pcgEigenOMP que obteve um desempenho de 64.05 minutos com uma média de 4,575 minutos para cada passo de tempo.

Neste estudo a solução do sistema de equação consome uma boa parte do tempo da análise. Uma análise na GPU para esse problema geomecânico usando a combinação do KeExpGPUS para calcular as matrizes dos elementos (Ke), sparseGPUS para montar a matriz global (KG) e pcgGPUS para a solução do sistema linear Com o triplets (ExpGPUS, sparseGPUS, pcgGPUS) foi a melhor combinação das análises feitas neste estudo.

Para uma análise na CPU a melhor combinação foi ExpS para o cálculo das matrizes locais Ke, sparse_create a montagem da matriz global KG e pcgEigenOMP para a solução do sistema linear. Por fim, o tempo total para resolver o problema proposto neste trabalho foi muito importante e é função de alguns parâmetros, principalmente o tamanho da malha e quantidade de passos de tempo da análise.

Neste estudo, para a malha com 14,709,888 elementos e 8,323,200 graus de liberdade e utilizando as melhores técnicas de cada segmento, obteve-se um tempo total de execução de 21.27 minutos para os 14 passos de tempo da análise.

O gráfico da Figura 30 traz uma observação interessante sobre a contribuição dos tempos de cálculo das matrizes de rigidez locais e globais Ke e KG somados frente

ao tempo de solução do sistema de equações (?). Foi observado que na CPU, para Ke e KG, as suas participações no tempo total decresce à medida a malha aumenta e para a malha 4 elas consumiram aproximadamente apenas em média 21% do tempo total para resolver o problema. Na GPU, a participação do tempo das matrizes Ke e KG juntos é mais significativo no tempo total devido à utilização do loop na construção da KG, tempo que representa aproximadamente 37% do tempo total necessário para resolver o problema.

A utilização da GPU não necessitou de nenhum conhecimento de programação de códigos em CUDA para obter as vantagens citadas anteriormente. Este trabalho traz a possibilidade de acelerar simulações geomecânicas com dispositivos modernos do tipo GPU sem necessidade de programação em linguagens de baixo nível além de excelente possibilidade em CPU com a integração à biblioteca de código aberto em C++ para operações de álgebra linear, como matrizes, vetores, operações de valores próprios (eigenvalues) e decomposições EIGEN. Isto confere a este estudo características inovadoras em termos de uma visão geral de diversas técnicas aplicadas ao problema de simulação geomecânica de reservatórios, indo além do processo de construção da matriz global e com uma discussão efetiva de estratégias que minimizem o tempo de solução do problema geomecânico partir de estratégias *oneway* com a utilização de estruturas de programação no MATLAB.

5.2 TRABALHOS FUTUROS

Com base em discussões que surgiram durante a produção deste trabalho foram formuladas algumas sugestões para trabalhos futuros:

- Aplicar a metodologia desenvolvida em casos reais de reservatório;
- Acoplar o modelo mecânico a softwares comerciais amplamente utilizados na simulação de escoamento em reservatórios de petróleo, buscando uma maior integração com as práticas convencionais da engenharia de reservatórios;
- A expansão do escopo do modelo para incluir análises que contemplem fenômenos de plasticidade no modelo constitutivo mecânico, o que representará uma evolução significativa para abordar situações mais complexas.
- Aplicar as rotinas desenvolvidas em técnica *twoway* em que o problema mecânico também influencia o problema de fluxo.

Por fim, como desenvolvimento futuro é importante mencionar a redução da dependência do MATLAB pela exploração de outros ambientes de programação que permitam estratégias mais flexíveis de paralelização. O desenvolvimento de softwares open source, promovendo assim a interoperabilidade e a colaboração entre pesquisadores, é outro ponto que também merece ser mencionado.

REFERÊNCIAS BIBLIOGRÁFICAS

Ahmed, B. I., & Al-Jawad, M. S. (2020). Geomechanical modelling and two-way coupling simulation for carbonate gas reservoir. *J Petrol Explor Prod Technol, 10 (4), 3619–3648. https://doi.org/10.1007/s13202-020-00965-7

AHMED, T. H. Reservoir Engineering Handbook . 4. ed. [S.l.]: Elsevier, 2010.

ALLEN, D. R. Physical Changes of Reservoir Properties Caused By Subsidence and Repressuring Operations. Journal of Petroleum Technology , v. 20, n. 01, p. 23–29, 1968.

Al-Ameri, N. (2020). Implication of geomechanical evaluation on tight reservoir development / Sadi reservoir Halfaya oil field. (Doctoral dissertation). Retrieved from https://doi.org/10.13140/RG.2.2.13608.80640

Almeida, A. S., et al. (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 (Vol. 2, pp. 840–849).

AN, C. et al. Adaptive Time Stepping with the Modified Local Error Method for Coupled. In: SPE Reservoir Characterisation and Simulation Conference and Exhibition , 2017.

Andersen, J., Refsgaard, J. C., & Jensen, K. H. (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, J. D., & Erutreya, O. E. (2015). Estimating Properties of Unconsolidated Petroleum Reservoirs using Image Analysis. J Geol Geophys, 5 , 1–4.

Biot, M. A. (1941). General theory of three-dimensional consolidation. J Appl Phys, 12 (2), 155–164.

Bird, R. E., Coombs, W. M., & Giani, S. (2017). Fast native-MATLAB stiffness assembly for SIPG linear elasticity. Comput Math Appl, 74 (12), 3209–3230. https://doi.org/10.1016/j.camwa.2017.08.022

Chen, Yanqing, et al. (2008). Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. ACM Transactions on Mathematical Software, 35 (3), 1–14. https://doi.org/10.1145/1391989.1391995

Coussy, O. (2004). Poromechanics . Chichester: J Wiley & Sons.

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, D. W. (2008). MILAMIN: MATLAB-based finite element method solver for large problems. Geochem Geophys Geosyst, 9 (4), 1–24.

Davis, T. A. (2019). Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra. ACM Trans. Math. Softw., 45 (4). https://doi.org/10.1145/3322125

DEAN, R. H.; GAI, X.; STONE, C. M.; MINKOFF, S. E. A comparison of techniques for coupling porous flow and geomechanics. SPE J, v. 11, n. 1, p. 132-140, 2006.

DURAN, O.; SANEI, M.; DEVLOO, P. R. V.; SANTOS, E. S. T. An enhanced sequential fully implicit scheme for reservoir geomechanics. Comput Geosci, v. 24, n. 4, p. 1557-1587, 2020.

Ferrari, A., et al. (2021). A consistent framework for coupling modern reservoir flow simulator with mechanics. In IOP Conference Series: Earth and Environmental Science (Vol. 012113). IOP Publishing.

Fox, G. C., et al. (2012). Distributed and Cloud Computing: From Parallel Processing to the Internet of Things . Morgan Kaufmann.

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

Gilbert, J. R., Moler, C., & Schreiber, R. (1992). Sparse Matrices in MATLAB: Design and Implementation. SIAM Journal on Matrix Analysis and Applications, 13 (1), 333–356. https://doi.org/10.1137/0613024

Gomes, A. F. C., Marinho, J. L. G., & Santos, J. P. L. (2019). Numerical Simulation of Drilling Fluid Behavior in Different Depths of an Oil Well. Braz J Pet Gas, 13 (4), 309–322.

Griffiths, D. V., Huang, J., & Schiermeyer, C. (1996). A Fast Solver for the Symmetric Generalized Eigenproblem Arising from Linear Stability Analysis of Viscous Flows. SIAM J Sci Comput, 17 (1), 29–41.

Jansen, J. D. (2011). Geomechanics of Reservoir and Civil Engineering . Delft: Delft University of Technology.

Jiang, X., Mavko, G., & Dvorkin, J. (2015). Effective rock properties and seismic velocities in carbonates: Theoretical modeling and application. GEOPHYSICS,

80 (6), D383–D393. https://doi.org/10.1190/geo2014-0603.1

Jiang, X., Mavko, G., & Mukerji, T. (2007). Effective elastic moduli of partially saturated rocks. Geophysics, 72 (3), D29–D36.

Jiang, X., Mavko, G., & Mukerji, T. (2009). Seismic and rock-physics diagnostics of reservoir and seal rocks. GEOPHYSICS, 74 (5), D11–D19. https://doi.org/10.1190/1.3182601

Kamiński, T., & Teisseyre, R. (2014). Geomechanics of salt deposits . Berlin Heidelberg: Springer.

Kjartansson, E., J. (1979). Constant Q-wave propagation and attenuation. J Geophys Res, 84 (1), 473–484.

Kruskopf, M., & Thiele, M. (2020). A high-performance open-source implementation of the full waveform inversion algorithm for seismic data. Comput Geosci, 136 (104403). https://doi.org/10.1016/j.cageo.2020.104403

Kwon, J., & Lee, T. S. (2018). Hybrid parallelization of iterative sparse linear solvers for large-scale 3D reservoir simulation. J Comput Phys, 353 (1), 23–38. https://doi.org/10.1016/j.jcp.2017.09.008

Li, H., & Demanet, L. (2018). 3D modeling and inversion of electromagnetic data using a nested sampling algorithm. Geophysics, 83 (2), E47–E58. https://doi.org/10.1190/GEO2017-0036.1

Li, H., & Mavko, G. (2016). Pore aspect ratio and saturation dependence of elastic moduli for carbonates. Geophysics, 81 (6), MR155–MR167.

MATHWORKS. Disponível em: <www.mathworks.com>. Acesso em: 7 maio 2021.

Mavko, G., Mukerji, T., & Dvorkin, J. (2009). The Rock Physics Handbook: Tools for Seismic Analysis in Porous Media . Cambridge University Press.

Mavko, G., Mukerji, T., & Jizba, D. (2009). Quantitative seismic interpretation: Applying rock physics tools to reduce interpretation risk. Cambridge University Press .

Mohd Noh, H. et al. (2019). Development of a fast coupled-reservoir simulation model for hydraulically fractured wells. J Nat Gas Sci Eng, 64 (2019), 129–142. https://doi.org/10.1016/j.jngse.2018.11.008

Nag, S., & Vlad, S. (2012). An improved parallel solver for 3D stress analysis in reservoirs with complex geological structures. Computers & Fluids, 55 (2012), 123–129. https://doi.org/10.1016/j.compfluid.2012.03.013

Nash, J. C. (1990). Compact Numerical Methods for Computers: Linear Algebra

and Function Minimisation. CRC Press .

Neymark, L. A., & Vergasova, G. V. (2011). Evaluation of boundary conditions for numerical simulation of hydrocarbon migration. Comput Geosci, 15 (4), 601–613.

Oliver, D. S. (2011). Drilling Fluids: Chemistry and Applications. Gulf Professional Publishing .

Oristaglio, M. L., & Batycky, R. P. (2015). SPE-174202-MS Efficient, Robust Finite Element Analysis for Applications in Geomechanics. In SPE Hydraulic Fracturing Technology Conference . https://doi.org/10.2118/174202-MS

Petrobras. Supercomputador Pégaso. 2022. Disponível em:< https://agencia.petrobras.com.br/pt/inovacao/petrobras-e-tetracampea-pegaso-e-o-maior-e-mais-ecoeficiente-supercomputador-da-america-latina-16-11-2022/ >. Acesso em: 16 de agosto de 2023

Pipitone, G. et al. (2015). Efficient parallel iterative linear solvers for large-scale reservoir simulation. Comput Geosci, 19 (6), 1065–1085. https://doi.org/10.1007/s10596-015-9479-0

Pipitone, G., et al. (2015). Efficient parallel iterative linear solvers for large-scale reservoir simulation. Comput Geosci, 19 (6), 1065–1085. https://doi.org/10.1007/s10596-015-9479-0

Ralph, J., & Santos, R. (2016). Advances in reservoir simulation: An overview of current techniques and applications. Petroleum Science, 13 (4), 788–801. https://doi.org/10.1007/s12182-016-0142-z

GID Simulation. Disponível em: <www.gidsimulation.com>. Acesso em: <18.08.2023>.

Smith, L. K., et al. (2017). Parallel algorithms for large-scale reservoir modeling: A comparative study. Computational Geophysics, 21 (2), 283–297. https://doi.org/10.1007/s10596-017-9632-1

Turner, D. R., & Wang, Q. (2018). Application of iterative solvers in reservoir simulation on distributed memory systems. Journal of Computational and Applied Mathematics, 339 , 165–175. https://doi.org/10.1016/j.cam.2017.11.020

Vasquez, M. A., & Rodriguez, L. M. (2019). High-performance computing for large-scale reservoir simulation: A review of recent developments and challenges. Journal of Petroleum Science and Engineering, 177 , 454–468. https://doi.org/10.1016/j.petrol.2019.02.053

WAN, J. Stabilized Finite Element Methods for Coupled Geomechanics and

Multiphase Flow . 2002. 162 f. Tese (Doutorado em [Nome do Curso]) - Stanford University, Stanford, 2002.

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

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

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

ZIENKIEWICZ, O. C.; TAYLOR, R. L.; TAYLOR, R. L. The finite element method: solid mechanics. UK: Butterworth-Heinemann, 2000.

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