Revista Brasileira de Computação Aplicada, Julho, 2020

DOI: 10.5335/rbca.v12i2.10158
Vol. 12, Nº 2, pp. 103–121
Homepage: seer.upf.br/index.php/rbca/index

ISSN 2176-6649

ARTIGO ORIGINAL

Uma implementação paralelizada via a API OpenMP para a simulação numérica de reservatórios de gás natural

A parallel implementation via the OpenMP API for numerical simulation of natural gas reservoirs

Leonardo Tarazona Muzí de Carvalho[1], Leonardo Figueira Werneck[1], Grazione de Souza [1], Helio Pedro Amaral Souto [1]

[1]Departamento Modelagem Computacional, Instituto Politécnico, UERJ, Brasil.

*leomuzicarvalho@gmail.com; lwerneck@iprj.uerj.br; gsouza@iprj.uerj.br, helio@iprj.uerj.br

Recebido: 04/11/2019. **Revisado:** 08/01/2020. **Aceito:** 30/06/2020.

Resumo

Nas últimas décadas, a indústria de óleo e gás tem empregado cada vez mais recursos para reduzir os custos computacionais em simulações numéricas de escoamentos em reservatórios. O estudo de casos realísticos leva, em geral, à solução de sistemas de equações algébricas não-lineares que demandam esforços computacionais significativos de processamento e memória. Tais equações são obtidas a partir da discretização das equações diferenciais parciais utilizadas na modelagem dos escoamentos. Um exemplo de técnica aplicada em implementações de alto desempenho, a qual leva à redução no tempo de processamento, é a *Application Programing Interface* (API) *Open Multi-Processing* (OpenMP), baseada na utilização de memória compartilhada e de linhas de execução (*threads*). Neste trabalho, utiliza-se a paralelização via o OpenMP para melhorar o desempenho de um simulador numérico de escoamentos bidimensionais em reservatórios de gás natural. Os métodos estacionários de Jacobi, Gauss-Seidel e SOR, para a solução de sistemas de equações algébricas, foram paralelizados e comparados. O método SOR foi o escolhido para ser aplicado nos estudos envolvendo a variação do número de volumes da malha computacional e das características do escoamento. Em todas as simulações realizadas obteve-se ganhos com a paralelização em relação ao desempenho das versões seriais, atingindo-se valores máximos de *speedup* superiores a 7 em alguns casos.

Palavras-Chave: API OpenMP; gás natural; paralelização; simulação de reservatórios; métodos estacionários.

Abstract

Over the past decades, the oil and gas industry has increasingly used resources to reduce computational costs in reservoir numerical simulations. Realistic case studies generally lead to the solution of systems of nonlinear algebraic equations that require significant computational processing and memory efforts. We obtain such equations from a process of discretization of partial differential equations used in reservoir flow modeling. An example of a technique applied to high-performance implementations that leads to reduced processing time is the Open Multi-Processing (OpenMP) Application Programming Interface (API), based on shared memory usage and threads. In this work, OpenMP is used to improve the performance of a numerical flow simulator in natural gas reservoirs. Jacobi, Gauss-Seidel, and SOR stationary methods for the solution of algebraic equation systems were parallelized and compared. We choose the SOR method to be applied in studies involving the variation of the number of computational mesh volumes and the flow characteristics. In all simulations performed, gains were obtained with parallelization when compared to the performance of serial versions, reaching maximum speedup values higher than 7 in some cases.

Keywords: API OpenMP; natural gas; parallelization; reservoir simulation; stationary methods.

1 Introdução

Desde a antiguidade, o ser humano tem contato com o gás natural. Babilônios, gregos e persas possuíam templos onde o gás natural era expelido naturalmente. Economicamente, o gás tem sido utilizado desde os séculos XVIII e XIX, quando os chineses utilizavam os locais onde o gás era ejetado para construir fornos voltados para a fabricação de cerâmica e a metalurgia. A partir do século XIX, o gás natural passou a ser obtido como subproduto da produção de petróleo (Werneck, 2016). Graças à criação de gasodutos e do queimador de Bunsen, em 1885, o gás natural obtido junto com o óleo passou a ser amplamente utilizado apesar de, até então, não existir tecnologia que viabilizasse o transporte de grandes quantidades de gás a longas distâncias. O período de construções, após o final da segunda guerra mundial, foi de grande avanço na área, devido à instalação de uma extensa linha de gasodutos, viabilizando a utilização do gás natural em larga escala, principalmente devido à diversidade de aplicações e ao seu menor potencial poluidor em relação ao óleo (Anderson, 1984). O desenvolvimento de tecnologias voltadas para o aproveitamento do gás natural foi ampliado, desta forma, em diferentes frentes, incluindo a área de desenvolvimento de *software* científico. Tais avanços incluem a aplicação de técnicas de computação de alto desempenho (*High Performance Computing* (HPC)), como é o caso deste trabalho.

1.1 Computação de alto desempenho

Desde que os computadores começaram a ser amplamente comercializados, as empresas de tecnologia de todo o mundo têm voltado seus esforços para melhor atender aos consumidores, tendo sido impulsionadas a introduzir melhorias em seus produtos. A alta competitividade levou as grandes fabricantes a investirem em tecnologias de *software* e *hardware* de alto desempenho. Portanto, não poderia ser diferente com relação ao desenvolvimento de processadores e placas gráficas. Cada vez mais, as desenvolvedoras têm buscado entregar novos componentes de processamento extremamente velozes (NVIDIA, 2018). No entanto, para se obter um desempenho computacional significativo, não basta apenas possuir um *hardware* potente. Na verdade, o uso de recursos físicos necessita do emprego de *softwares* otimizados para que seja possível o desenvolvimento da computação de alto desempenho, ou seja, que tem por finalidade reduzir o tempo empregado na execução dos códigos.

Na área de simulação numérica de fenômenos físicos, em problemas de engenharia, à medida que se estuda problemas cada vez mais complexos, a resolução numérica requer o uso intenso de recursos de memória e/ou processamento. Neste contexto, encontra-se a motivação para a implementação de técnicas de computação de alto desempenho. O objetivo desta aplicação é possibilitar a simulação de modelos matemáticos associados a sistemas de equações com um grande número de incógnitas ou, até mesmo, tornar viável as simulações numéricas de problemas físicos que, muitas vezes, não são possíveis de serem realizadas utilizando outras metodologias (Werneck, 2016). Com esta finalidade se utiliza, por exemplo, o processamento em paralelo (uma das técnicas da computação científica de alto desempenho), cuja principal meta é dividir uma determinada tarefa em várias menores e executar cada uma delas, por exemplo, em diferentes núcleos de processamento ou computadores.

Um exemplo de técnica de computação de alto desempenho é o MPI (*Message Passage Interface*). Neste caso, diferentes processadores/computadores são utilizados na realização de uma dada tarefa empregando o acesso à memória distribuída (Carpen-Amarie et al., 2017). Uma alternativa inclui o processamento usando GPUs (*Graphics Processing Unit*), por meio da aplicação da API CUDA (NVIDIA, 2014) ou da OpenACC (NVIDIA, 2018). Também é possível a utilização da API OpenMP, a qual pode ser utilizada em computadores com arquiteturas de memória compartilhada, sendo as *threads* (linhas de execução) empregadas na realização de tarefas em paralelo (Dietrich et al., 2017, ARB, 2011). Sistemas híbridos também podem ser pensados, os quais utilizam em conjunto o acesso às memórias distribuída e compartilhada como, por exemplo, códigos escritos com comandos do MPI e do OpenMP (Yakubov et al., 2013, Redondo, 2017). Salienta-se, no entanto, que os ganhos atingidos com essas tecnologias não são ilimitados. Até o presente momento, não se conhece nenhum tipo de técnica de computação de alto desempenho que obtenha um retorno linear de economia de tempo, de acordo com o aumento de *threads* e/ou processadores, para todos os problemas (Halsey, 2016). Dentro do contexto apresentado e em função do fato da API OpenMP ser aplicada mais facilmente aos códigos já existentes, optou-se pelo seu uso neste trabalho.

1.2 OpenMP

Conforme já mencionado, o OpenMP é uma API que permite a paralelização de processos através do uso de memória compartilhada (Chapman et al., 2008), de maneira mais simples quando comparada, por exemplo, a uma implementação utilizando a biblioteca CUDA ou mesmo a API MPI, já que é uma aplicação que visa a organizar tarefas de baixo nível através de instruções de alto nível. Por exemplo, é possível definir as regiões a serem paralelizadas, o número de *threads*, as variáveis e as regiões de memória a serem compartilhadas e, como o processador e suas *threads* lidarão com um trecho do programa especificado, a paralelização com apenas uma linha de código contendo algumas diretrizes.

Uma *thread* é um conjunto de instruções de um código de programa executado em um determinado tempo de maneira independente. Essencialmente, o sistema operacional aloca os recursos necessários para a execução do processo em questão e, se múltiplas *threads* participam desta execução, elas compartilham, portanto, das mesmas regiões de memória e recursos. A biblioteca do OpenMP (omp.h) já se incube desta organização, para evitar as complexidades inerentes às implementações e aos erros (uma implementação de mais baixo nível poderia acarretar em falhas e em uma

complexidade crescente, de acordo com a evolução do nível de dificuldade do problema a ser tratado computacionalmente) (Chapman et al., 2008).

Apesar da possibilidade de se executar um código inteiro em paralelo, podem existir algumas limitações que a tornam inviável. O OpenMP permite paralelizar laços que são finitos e com término previamente conhecido, não podendo sofrer alterações de tamanho durante a sua execução. Um laço só poderá ser paralelizado se as suas iterações forem independentes, conforme discutido em Chapman et al. (2008). Vale a pena destacar que a biblioteca do OpenMP está disponível para as linguagens de programação C, C++ e Fortran e, mais recentemente, para o Python. Diversas áreas de pesquisa e desenvolvimento, além de aplicações industriais, têm tirado proveito da aplicação de técnicas tais como o OpenMP. Por exemplo, a simulação numérica de reservatórios de óleo e/ou gás (Werneck, 2016, Redondo, 2017, Werneck et al., 2019), sendo que os reservatórios de gás natural são o objeto de interesse deste artigo.

1.3 Simulação numérica de reservatórios de gás

A simulação numérica de reservatórios é uma área da engenharia de reservatórios, um dos ramos da engenharia de petróleo, dedicada à aplicação de métodos numéricos para a determinação de soluções aproximadas, para a dinâmica do escoamento nas formações portadoras de hidrocarbonetos. A simulação pode ser utilizada com a finalidade, por exemplo, de se determinar o campo de pressões no reservatório e estimar as pressões nos poços produtores, contribuindo para a realização do teste de pressão e a avaliação de cenários de produção. Desta forma, é possível otimizar a produção de forma a maximizar o fator de recuperação de hidrocarbonetos.

Segundo de Souza (2013), a aplicação da simulação numérica no gerenciamento de reservatórios tornou-se um padrão na indústria de óleo e gás, com a sua aceitação sendo atribuída a fatores tais como: avanços no desenvolvimento dos recursos computacionais (maior velocidade de processamento e aumento da capacidade de memória); avanços das técnicas numéricas aplicadas na resolução das equações diferenciais parciais (EDPs) que governam o escoamento; capacidade dos simuladores para tratar de diversos casos na escala de campo de interesse prático; desenvolvimento de modelos para a caracterização do sistema poço-reservatório; e capacidade de se estudar as complexas técnicas de recuperação de hidrocarbonetos via a simulação numérica.

O escoamento em reservatórios é, em geral, governado por um conjunto de equações diferenciais parciais não-lineares parabólicas e, portanto, condições inicial e de contorno devam ser fornecidas para que elas possam ser resolvidas. No caso do escoamento de gás natural, a não linearidade das equações decorre da dependência das propriedades físicas do gás com relação à pressão e devido aos efeitos não-lineares oriundos da incorporação de fenômenos típicos do escoamento de gás em meios porosos, ou seja, um meio formado por um arcabouço sólido com a presença de espaços denominados poros, nos quais os fluidos encontram-se armazenados e podem escoar (Ertekin et al., 2001).

Em função da presença das não linearidades, um maior esforço computacional em termos de processamento e memória é exigido quando da solução numérica das equações governantes, para a determinação do campo de pressões no reservatório. Assim, surge a necessidade do uso de técnicas de computação de alto desempenho quando do estudo de casos realísticos, nos quais os reservatórios possuem quilômetros de extensão nas direções x e y e dezenas de metros na direção z. Nesses casos, as malhas computacionais podem possuir um número de nós que pode ultrapassar um milhão. Além disso, em certas aplicações, mais de uma variável dependente pode ser determinada (por exemplo, a temperatura em um escoamento não-isotérmico) e o tempo de produção do reservatório pode se estender por décadas. Nesse cenário, simulações numéricas cujo tempo de execução computacional pode durar mais do que 1 dia não são incomuns. Portanto, os métodos numéricos a serem empregados devem contemplar o tratamento das não linearidades e das escalas envolvidas nos problemas de engenharia. Diante do exposto, conclui-se que as estratégias de computação paralela tornam-se uma alternativa interessante, no sentido de viabilizar a simulação numérica voltada para a análise de cenários de produção realísticos.

2 Escoamento de Gás em Meios Porosos

Um meio poroso possui uma estrutura sólida e poros que podem estar preenchidos com fluidos. Quanto mais poroso for um reservatório, maior será a sua capacidade de armazenamento de hidrocarbonetos (Dandekar, 2013). A porosidade efetiva, ϕ, é a relação entre o volume ocupado pelos poros interconectados e o volume total do meio poroso. Na engenharia de petróleo, a porosidade efetiva é a mais relevante porque somente os poros interconectados permitem o escoamento de fluidos através do reservatório.

Uma outra propriedade da formação rochosa, a permeabilidade, denotada em geral pelo tensor $\mathbf{k}$ (Dandekar, 2013), fornece uma medida da resistência que o meio poroso opõe ao escoamento de um fluido. Ela é denominada de permeabilidade absoluta, ou permeabilidade, se o meio poroso encontra-se 100% saturado (preenchido) por uma única fase líquida. Já a permeabilidade relativa é uma medida da resistência que o meio poroso oferece ao movimento de uma fase, se duas ou mais fases ocupam a formação rochosa.

Para escoamentos nos quais ocorrem efeitos não contemplados pela lei de Darcy clássica, geralmente se utiliza a denominação de escoamento não-Darcy. Como exemplos de efeitos não-Darcy, que podem ser incorporados na permeabilidade aparente, tem-se: o escoamento de fluidos não-newtonianos, os efeitos inerciais e/ou turbulentos, o escorregamento do fluido na superfície sólida, além de outros que causam não linearidades (Aziz and Settari, 1990). Em geral, a lei de Darcy modificada é dada pela Eq. (1):

$$\mathbf{v} = -\frac{\mathbf{k}_a}{\mu}\left(\nabla p - \rho g \nabla D\right), \tag{1}$$

onde $\mathbf{v}$ é a velocidade superficial do fluido, $\mathbf{k}_a$ é denominado de tensor de permeabilidade aparente, μ é a viscosidade do fluido, p é a pressão, ρ é a massa específica do fluido, g é a magnitude da aceleração da gravidade e D é a profundidade . No caso do escoamento a altas velocidades, desvios da lei de Darcy são observados, sendo inicialmente devidos à inércia, seguidos da ocorrência de turbulência. No caso de fluidos não-newtonianos, modificações são adotadas para o cálculo da viscosidade (Aziz and Settari, 1990). Para gases, pode ocorrer o escorregamento do gás na superfície sólida dos poros, que resulta em maiores valores da permeabilidade efetiva quando comparados ao caso de líquidos. Este fenômeno é considerado, em parte, nas simulações deste trabalho.

A partir do contexto exposto, as seguintes hipóteses são consideradas no modelo físico-matemático:

- o reservatório é homogêneo e anisotrópico em relação à permeabilidade absoluta;
- a compressibilidade da rocha é pequena e constante;
- não ocorrem reações químicas;
- o escoamento é bidimensional e ocorre a baixas velocidades;
- o escoamento é monofásico e isotérmico;
- o gás no reservatório é considerado real;
- podem ocorrer efeitos de escorregamento;
- fluido com composição química constante;
- o poço de produção é vertical e penetra totalmente a formação rochosa;
- não ocorrem efeitos eletrocinéticos;
- a ausência de estocagem no poço e dano à formação.

O escorregamento do gás na superfície sólida, em um meio poroso, acontece quando o livre caminho médio das moléculas do fluido é comparável ao raio hidráulico característico dos poros do reservatório (Florence et al., 2007, Li et al., 2016, Chung et al., 2015). Desta forma, o gás escoa no meio poroso diferentemente dos líquidos por duas razões fundamentais: a alta compressibilidade e o efeito Klinkenberg (Klinkenberg, 1941, Aziz and Settari, 1990). A correção de Klinkenberg foi elaborada com o objetivo de incorporar os efeitos do escorregamento nas medidas de permeabilidade (Eq. (2)),

$$\mathbf{k_a} = \left(1 + \frac{b_K}{p}\right)\mathbf{k} \tag{2}$$

onde b_K é chamado de parâmetro de Klinkenberg.

Em função das condições de escoamento, em reservatórios dos tipos *shale gas* e *tight gas sands*, não convencionais, pode ocorrer o fenômeno do escorregamento. A permeabilidade efetiva do gás também pode ser calculada, empregando um modelo mais geral, por Eq. (3):

$$\mathbf{k}_a = f(Kn)\mathbf{k} = (1 + \alpha_k Kn)\left(1 + \frac{4Kn}{1+Kn}\right)\mathbf{k}, \tag{3}$$

onde α_k é o parâmetro de rarefação definido na Eq. (4) (Jiang and Younis, 2015),

$$\alpha_k = \frac{128}{15\pi^2}\arctan(4K_n^{0,4}) \tag{4}$$

e a Eq. (5) representa o número de Knudsen, onde λ (Eq. (6)) é o livre caminho médio das moléculas, e R_h (Eq. (7)) é o raio hidráulico característico dos poros,

$$Kn = \frac{\lambda}{R_h} \tag{5}$$

$$\lambda = \frac{\mu}{p}\sqrt{\frac{\pi ZRT}{2M}} \tag{6}$$

$$R_h = 2\sqrt{2\tau}\sqrt{\frac{k}{\phi}}, \tag{7}$$

onde Z é o fator de compressibilidade do gás, R a constante universal dos gases, T é a temperatura, M a massa molecular do gás, τ é a tortuosidade do meio poroso e k é a média geométrica dos valores das permeabilidades do tensor diagonal $\mathbf{k}$, no caso de um meio poroso considerado anisotrópico (Chen et al., 2006).

Dependendo do valor de Kn, o meio pode ser dito contínuo se $Kn \leq 10^{-3}$, o regime de escorregamento ocorre para $10^{-3} < Kn < 0,1$, com uma transição para $0,1 \leq Kn \leq 10$ e quando $Kn \geq 10$ há o escoamento molecular livre (Bestok, 1999). Para o regime de escorregamento tem-se que α_k é igual a zero.

3 Equação governante

Para se obter uma EDP em termos da pressão utiliza-se, em conjunto a conservação de massa, a lei de Darcy modificada e relações para a determinação das propriedades de rocha e de fluido em função da pressão. A conservação de massa, para o escoamento de gás em meios porosos, pode ser escrita como na Eq. (8) (Ertekin et al., 2001),

$$\frac{\partial}{\partial t}\left(\frac{\rho_{sc}\phi}{B}\right) + \nabla\cdot\left(\frac{\rho_{sc}\mathbf{v}}{B}\right) - \frac{q_{sc}\rho_{sc}}{V_b} = 0 \tag{8}$$

onde ρ_{sc} é a massa específica determinada nas condições padrão de pressão (p_{sc}) e temperatura (T_{sc}), $B = \rho_{sc}/\rho$ é o fator volume de formação do gás, q_{sc} é um termo fonte/sorvedouro em condições padrão e V_b é o volume total (rocha mais poros).

Substituindo-se a Eq. (1) na Eq. (8), considerando que ρ_{sc} é constante e desprezando-se os efeitos gravitacionais tem-se a Eq. (9):

$$\nabla \cdot \left(\frac{\mathbf{k}_a}{\mu B} \nabla p \right) + \frac{q_{sc}}{V_b} = \frac{\partial}{\partial t} \left(\frac{\phi}{B} \right). \tag{9}$$

Utilizando-se a regra da cadeia e lembrando que supõe-se que as propriedades do meio e do fluido dependem somente da pressão (Dyrdahl, 2014, Ertekin et al., 2001), tem-se a Eq. (10)

$$\nabla \cdot \left(\frac{\mathbf{k}_a}{\mu B} \nabla p \right) + \frac{q_{sc}}{V_b} = \Gamma \frac{\partial p}{\partial t} \tag{10}$$

com Γ sendo dado pela Eq. (11)

$$\Gamma = \frac{c_\phi \phi^0}{B} + \phi \frac{d}{dp} \left(\frac{1}{B} \right), \tag{11}$$

onde c_ϕ é o coeficiente de compressibilidade da rocha e ϕ^0 a porosidade de referência. Aqui, considera-se que $\phi = [1 + c_\phi (p - p^0)]\phi^0$, onde p^0 é a pressão de referência (Ertekin et al., 2001).

A viscosidade é determinada como sendo uma função de p, T e γ (Lee et al., 1966), onde a densidade do gás é dada pela razão entre a sua massa molecular e a do ar, $\gamma = M/M_{ar}$. O fator de compressibilidade Z é calculado utilizando a correlacão fornecida por Dranchuk and Abou-Kassem (1975) e a massa específica do gás via a equação de estado para um gás real, $\rho = pM/(ZRT)$.

Como considera-se que o escoamento é bidimensional e que o tensor de permeabilidade é diagonal, tem-se a Eq. (12):

$$\frac{\partial}{\partial x} \left(\frac{k_{ax}}{\mu B} \frac{\partial p}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{k_{ay}}{\mu B} \frac{\partial p}{\partial y} \right) + \frac{q_{sc}}{V_b} = \Gamma \frac{\partial p}{\partial t}. \tag{12}$$

O termo fonte q_{sc} é utilizado para representar a vazão do poço produtor, introduzida via uma condição de contorno interna. Na verdade, o reservatório é tridimensional e possui uma profundidade. Entretanto, devido à sua geometria, condições de contorno e acoplamento poço-reservatório (poço posicionado no centro do reservatório), o escoamento é efetivamente bidimensional e os campos de pressão e velocidade do reservatório não variam com a profundidade.

Para o acoplamento poço-reservatório (Peaceman, 1978, 1983) utiliza-se a Eq. (13) (Ertekin et al., 2001)

$$q_{sc} = -J_w \left(p - p_{wf} \right), \tag{13}$$

onde J_w é o índice de produtividade e p_{wf} a pressão no poço. Substituindo-se a Eq. (13) na Eq. (12) tem-se a Eq. (14)

$$\frac{\partial}{\partial x} \left(\frac{k_{ax}}{\mu B} \frac{\partial p}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{k_{ay}}{\mu B} \frac{\partial p}{\partial y} \right) - \frac{J_w}{V_b} \left(p - p_{wf} \right) = \Gamma \frac{\partial p}{\partial t}. \tag{14}$$

A Eq. (14) é uma EDP não-linear utilizada para computar a pressão do gás, desde que sejam fornecidas as condições inicial e de contorno apropriadas. Como condição inicial utiliza-se a Eq. (15)

$$p(x, y, t = 0) = p_{ini}(x, y) = p_{ini}, \tag{15}$$

onde a pressão inicial antes de começar a produção/injeção é dada por p_{ini}. A porosidade inicial é dada por ϕ_{ini}.

A condição de contorno interna é utilizada no acoplamento poço-reservatório. Já as condições de contorno externas são as de fluxo nulo nas fronteiras (Eq. (16))

$$\left(\frac{\partial p}{\partial x} \right)_{x=0,L_x} = \left(\frac{\partial p}{\partial y} \right)_{y=0,L_y} = 0, \tag{16}$$

onde L_x e L_y são, nesta ordem, os comprimentos do reservatório nas direções x e y.

4 Metodologia de Resolução Numérica

O método das diferenças finitas foi empregado tendo-se em vista a conversão da Eq. (14) em um sistema de equações algébricas, que posteriormente será resolvido numericamente para a determinação dos valores da pressão nos nós discretos da malha computacional (Aziz and Settari, 1990, Ertekin et al., 2001, Abou-Kassem et al., 2006, Chen et al., 2006). Aqui, utilizou-se uma malha de blocos centrados (Ertekin et al., 2001), na qual um conjunto de células é sobreposto ao domínio físico. Uma representação ilustrativa do domínio bidimensional discretizado encontra-se na Fig. 1, considerando um sistema de coordenadas Cartesianas x e y. A solução numérica determinará os valores médios da pressão nos centros das células, sendo que n_x e n_y representam os números totais de células nas direções x e y, respectivamente. Os índices inteiros i e j indicam as numerações das células nas direções de x e y, respectivamente.

Reescreve-se, agora, a equação governante como sendo avaliada na célula indexada (i, j) e no nível de tempo $n + 1$, no qual as pressões devem ser calculadas (Eq. (17)),

$$\left[\frac{\partial}{\partial x} \left(\mathbb{T}'_x \frac{\partial p}{\partial x} \right) dx + \frac{\partial}{\partial y} \left(\mathbb{T}'_y \frac{\partial p}{\partial y} \right) dy \right]_{i,j}^{n+1} = \left(\Gamma p \frac{\partial p}{\partial t} + q_{sc} \right)_{i,j}^{n+1}, \tag{17}$$

sendo $(V_b)_{i,j} = (dxdy)_{i,j} L_z$ e introduziu-se os coeficientes (Eqs. (18) a (20))

$$\mathbb{T}'_x \equiv \frac{A_x k_{ax}}{\mu B}, \tag{18}$$

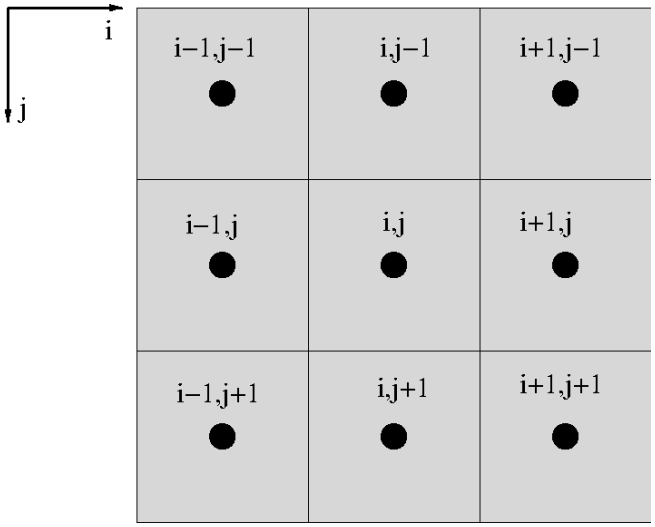

Figura 1: Malha bidimensional

$$\mathbb{T}'_y \equiv \frac{A_y k_{ay}}{\mu B} \tag{19}$$

e

$$(\Gamma_p)_{i,j} = (\Gamma)_{i,j}\,(V_b)_{i,j}\,, \tag{20}$$

onde $(A_x)_{i,j} = dy_{i,j}L_z$ e $(A_y)_{i,j} = dx_{i,j}L_z$, sendo L_z o comprimento do reservatório na direção z (profundidade). Considera-se a hipótese de que dx não varia em y e que dy não varia em x (Ertekin et al., 2001).

Utilizando-se como exemplo a malha computacional da Fig. 1 e aplicando-se o método das diferenças finitas na aproximação das derivadas espaciais (Ertekin et al., 2001), obtém-se a Eq. (21):

$$\frac{\partial}{\partial x}\left(\mathbb{T}'_x\frac{\partial p}{\partial x}\right)^{n+1}_{i,j} \cong \frac{1}{\Delta x_{i,j}}\left[\left(\mathbb{T}'_x\frac{\partial p}{\partial x}\right)_{i+\frac{1}{2},j} - \left(\mathbb{T}'_x\frac{\partial p}{\partial x}\right)_{i-\frac{1}{2},j}\right]^{n+1}, \tag{21}$$

onde Δx e Δy são, respectivamente, os incrementos espaciais da malha computacional nas direções x e y.

Na aproximação numérica das derivadas espaciais, para a discretização na direção x, Fig. 2, sabe-se que $i-1/2$ e $i+1/2$ indicam as interfaces da célula i compartilhada com as suas vizinhas $i-1$ e $i+1$. Assim, utilizando-se uma aproximação centrada no espaço (Eqs. (22) e (23)) (Aziz and Settari, 1990),

$$\left(\frac{\partial p}{\partial x}\right)^{n+1}_{i+\frac{1}{2},j} \cong \frac{p^{n+1}_{i+1,j} - p^{n+1}_{i,j}}{\Delta x_{i+\frac{1}{2},j}} \tag{22}$$

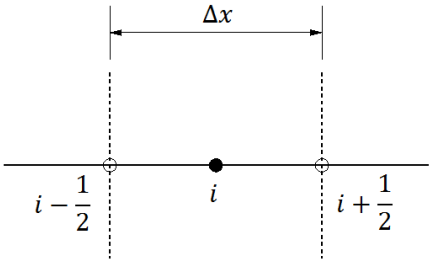

Figura 2: Malha unidimensional

$$\left(\frac{\partial p}{\partial x}\right)^{n+1}_{i-\frac{1}{2},j} \cong \frac{p^{n+1}_{i,j} - p^{n+1}_{i-1,j}}{\Delta x_{i-\frac{1}{2},j}} \tag{23}$$

sendo $\Delta x_{i\pm1/2,j}$ a distância entre os nós i e $i\pm1$. Aproximações para as derivadas em relação a y podem ser obtidas de forma análoga.

Introduz-se, agora, a transmissibilidade para a direção x (Eq. (24)),

$$\mathbb{T}^{n+1}_{x,i\pm\frac{1}{2},j} = \left(\frac{A_x k_{ax}}{\mu B \Delta x}\right)^{n+1}_{i\pm\frac{1}{2},j}, \tag{24}$$

onde a área (A) e a permeabilidade aparente (k_a) nas interfaces são calculadas por uma média harmônica a partir dos valores conhecidos em (i,j) e $(i\pm1,j)$. Para as propriedades de fluido (μ e B), uma média ponderada (com relação aos espaçamentos da malha espacial) é utilizada (Ertekin et al., 2001). Estratégias análogas são utilizadas para a transmissibilidade na direção y.

4.1 Expansão conservativa para o termo temporal

Seguindo o sugerido por Ertekin et al. (2001), utiliza-se uma expansão conservativa para a determinação do coeficiente Γ_p (Eq. (25)),

$$(\Gamma_p)^{n+1}_{i,j} = V_{i,j}\left[\frac{1}{B^n}\frac{\partial\phi}{\partial p} + \phi^{n+1} + \frac{\partial}{\partial p}\left(\frac{1}{B}\right)\right]_{i,j}. \tag{25}$$

Deste modo, usando-se uma formulação totalmente implícita no tempo, obtém-se na Eq. (26) a forma final discretizada da Eq. (17),

$$\begin{aligned} & \mathbb{T}_x\Big|_{i+1/2,j}^{n+1}\left(p_{i+1,j}^{n+1}-p_{i,j}^{n+1}\right)-\mathbb{T}_x\Big|_{i-1/2,j}^{n+1}\left(p_{i,j}^{n+1}-p_{i-1,j}^{n+1}\right) \\ + \quad & \mathbb{T}_y\Big|_{i,j+1/2}^{n+1}\left(p_{i,j+1}^{n+1}-p_{i,j}^{n+1}\right)-\mathbb{T}_y\Big|_{i,j-1/2}^{n+1}\left(p_{i,j}^{n+1}-p_{i,j-1}^{n+1}\right) \\ = \quad & \frac{(\Gamma_p)_{i,j}^{n+1}}{\Delta t}\left(p_{i,j}^{n+1}-p_{i,j}^{n}\right)+(q_{sc})_{i,j}^{n+1} \end{aligned} \tag{26}$$

onde empregou-se uma aproximação atrasada no tempo (Eq. (27)) (Ertekin et al., 2001),

$$\left(\frac{\partial p}{\partial t}\right)_{i,j}^{n+1} \cong \frac{p_{i,j}^{n+1}-p_{i,j}^{n}}{\Delta t}, \tag{27}$$

onde n é o nível de tempo no qual a pressão é conhecida.

4.2 Acoplamento poço-reservatório

Apesar da equação governante ter sido apresentada em termos da pressão no reservatório, através do termo $(q_{sc})_{i,j}^{n+1}$ pode-se estimar a pressão no poço produtor (Peaceman, 1978, 1983, de Souza, 2013). Neste caso, a versão discreta da Eq. (13) é dada pela Eq. (28)

$$(q_{sc})_{i,j}^{n+1} = -(J_w)_{i,j}^{n+1}\left[p_{i,j}^{n+1}-\left(p_{wf}\right)_{i,j}^{n+1}\right], \tag{28}$$

onde o índice de produtividade, J_w, é calculado pela Eq. (29)

$$(J_w)_{i,j}^{n+1} = \left[\frac{2\pi L_z\sqrt{k_{a,x}k_{a,y}}}{B\mu\,\ln\left(\frac{r_{eq}}{r_w}\right)}\right]_{i,j}^{n+1} \tag{29}$$

onde r_w é o raio do poço e o raio equivalente, r_{eq}, é determinado seguindo a metodologia proposta por Peaceman (1983), adaptando-a para a sua aplicação ao caso do uso da permeabilidade aparente (Eq. (30)):

$$r_{eq} = 0,28\left[\frac{\sqrt{\sqrt{\frac{k_{a,y}}{k_{a,x}}}(\Delta x)^2+\sqrt{\frac{k_{a,x}}{k_{a,y}}}(\Delta y)^2}}{\sqrt[4]{\frac{k_{a,y}}{k_{a,x}}}+\sqrt[4]{\frac{k_{a,x}}{k_{a,y}}}}\right]_{i,j}. \tag{30}$$

4.3 Linearização

A Eq. (26) forma um sistema de equações algébricas não-lineares. A partir da resolução desse sistema obtém-se os valores da pressão no reservatório (Islam et al., 2010). No caso particular do sistema de coordenas Cartesianas, a pressão no poço vertical pode ser obtida explicitando-se a pressão do poço na Eq. (28), uma vez conhecida a pressão do reservatório ao final de cada passo de tempo.

No intuito de se utilizar um método iterativo apropriado para a resolução de sistemas lineares, introduz-se uma técnica de linearização empregando o método de Picard (Nick et al., 2013). Escreve-se, então, a partir da Eq. (26) tem-se a Eq. (31),

$$\begin{aligned} & \mathbb{T}_y\Big|_{i,j-1/2}^{\nu,n+1}p_{i,j-1}^{\nu+1,n+1}+\mathbb{T}_x\Big|_{i-1/2,j}^{\nu,n+1}p_{i-1,j}^{\nu+1,n+1} \\ - \quad & \left[\mathbb{T}_y\Big|_{i,j-1/2}^{\nu,n+1}+\mathbb{T}_x\Big|_{i-1/2,j}^{\nu,n+1}+\mathbb{T}_x\Big|_{i+1/2,j}^{\nu,n+1}+\mathbb{T}_y\Big|_{i,j+1/2}^{\nu,n+1}\right]p_{i,j}^{\nu+1,n+1} \\ + \quad & \mathbb{T}_x\Big|_{i+1/2,j}^{\nu,n+1}p_{i+1,j}^{\nu+1,n+1}+\mathbb{T}_y\Big|_{i,j+1/2}^{\nu,n+1}p_{i,j+1}^{\nu+1,n+1} \\ + \quad & \left[\frac{(\Gamma_p)_{i,j}^{\nu,n+1}}{\Delta t}\right]p_{i,j+1}^{\nu+1,n+1}-(q_{sc})_{i,j}^{n+1} \\ = \quad & -\left[(\Gamma_p)_{i,j}^{\nu,n+1}\right]p_{i,j}^{n} \end{aligned} \tag{31}$$

onde ν refere-se ao nível iterativo no qual são calculados os coeficientes e os termos fonte da equação discretizada, empregados na determinação das pressões em $(\nu+1, n+1)$.

Além disso, o procedimento numérico compreende uma iteração externa (método de Picard), onde ocorre um teste de convergência, e iterações internas para o cálculo da pressão p (método iterativo para a resolução do sistema de equações linearizadas: Jacobi, Gauss-Seidel ou SOR). Em outras palavras, a estratégia de solução segue um procedimento de iterações aninhadas (Ertekin et al., 2001):

- cálculo dos coeficientes e dos termos fontes no nível $(\nu, n+1)$;
- obtenção de p^{n+1} e p_{wf}^{n+1} no nível $(\nu+1, n+1)$;
- teste de convergência para a o processo de linearização (iteração externa) e para a resolução do sistema de equações linearizado (iteração interna).

5 Paralelizacão via a API OpenMP

Esta seção é dedicada à paralelização baseada no uso da API OpenMP. Em simulações típicas, para a determinação da pressão quando do escoamento monofásico isotérmico em um reservatório, a solução do sistema de equações algébricas consome a maior parte do tempo total de execução do código numérico. Por esse motivo, paraleliza-se o método iterativo (estacionário) utilizado na resolução do sistema linear.

5.1 Métodos iterativos estacionários

Os métodos conhecidos destinados à solução numérica de sistemas de equações podem ser classificados em diretos ou indiretos, estes últimos chamados também de iterativos (Saad, 2003). Nos métodos diretos, a partir de uma sequência de operações algébricas, obtém-se uma solução que não contém erros numéricos de aproximação. Essas metodologias costumam tirar proveito da estrutura da matriz associada à forma vetorial representativa do sistema algébrico (Eq. (32))

$$\mathbf{Ax} = \mathbf{b}, \tag{32}$$

onde **A** é conhecida como sendo a matriz dos coeficientes, **x** o vetor contendo as incógnitas a serem determinadas e **b** o vetor que contém os termos cujos valores são conhecidos. Contudo, conforme a dimensão do sistema aumenta, os métodos diretos costumam apresentar perdas de desempenho.

No caso dos métodos iterativos, uma estimativa inicial da solução é fornecida e ela é melhorada através de um procedimento iterativo, de forma que se interrompe a sequência de cálculos quando uma dada tolerância numérica é verificada e o processo é dito convergir (Ertekin et al., 2001). Os métodos iterativos são os mais utilizados na simulação de reservatórios. Neste trabalho, optou-se por utilizar uma classe de métodos iterativos mais simples, os chamados estacionários (Perni, 2002). Ressalta-se que existem métodos mais robustos, que são empregados correntemente na simulação de reservatórios em escala de campo, como os não-estacionários (Ertekin et al., 2001). Os leitores podem se reportar ao artigo de Werneck et al. (2019) para mais informações sobre a paralelização, usando o OpenMP e um coprocessador, dos métodos não-estacionários dos Gradientes Conjugados, Gradiente Biconjugado e Gradiente Biconjugado Estabilizado.

5.2 O método de Jacobi

Criado por Carl Gustav Jakob, o método de Jacobi é um método iterativo estacionário simples e útil para a determinação de soluções de sistemas de equações lineares. Sua implementação é descomplicada, conforme pode-se verificar no Algoritmo 1 (Werneck, 2016).

Uma grande desvantagem, apesar da sua simplicidade, é que este método não garante a convergência para a solução de todos os casos e, para sistemas lineares de grande dimensão, ele tende a ficar muito lento devido à sua complexidade computacional (número de operações a serem executadas) ser de $O(n^2)$, onde n representa a ordem da matriz dos coeficientes (Porto da Silveira, 2000).

5.3 O método de Gauss-Seidel

Criado pelos matemáticos Carl Friedrich Gauss e Philipp Ludwig Seidel, esse método pode ser considerado uma adaptação do método de Jacobi, otimizado de modo a utilizar os resultados já atualizados na próxima iteração, economizando no esforço computacional e levando a uma convergência mais rápida. Por utilizar o mesmo princípio do método de Jacobi (portanto, de mesma complexidade computacional), o método de Gauss-Seidel (Algoritmo 2) também não garante a convergência numérica para qualquer caso (Werneck, 2016).

Algoritmo 1: Método de Jacobi

Entrada: forneça **A**, **b**, a dimensão do vetor **b** (n), $\mathbf{x}^{(0)}$, o número máximo de iterações ($nmax$) e a tolerância tol;

para $k = 0 : nmax$ **faça**
  para $i = 0 : (n-1)$ **faça**
    $x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^{(k)} \right)$
  fim
  se
    $max \left| x_i^{(k+1)} - x_i^{(k)} \right| < tol \quad para \quad 0 \leq i \leq (n-1)$
    então
      $\mathbf{x} = \mathbf{x}^{(k+1)}$
  fim
  senão
    Se $k = nmax$, não houve convergência.
  fim
fim

Algoritmo 2: Método de Gauss-Seidel

Entrada: forneça **A**, **b**, a dimensão do vetor **b** (n), $\mathbf{x}^{(0)}$, o número máximo de iterações ($nmax$) e a tolerância tol;

para $k = 0 : nmax$ **faça**
  para $i = 0 : (n-1)$ **faça**
    $x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$
  fim
  se
    $max \left| x_i^{(k+1)} - x_i^{(k)} \right| < tol \quad para \quad 0 \leq i \leq (n-1)$
    então
      $\mathbf{x} = \mathbf{x}^{(k+1)}$
  fim
  senão
    Se $k = nmax$, não houve convergência.
  fim
fim

5.4 O método de Sobre-relaxação Sucessiva

O método de Sobre-relaxação Sucessiva (SOR), veja o Algoritmo 3, consiste em utilizar uma média ponderada, entre as iterações atual e a próxima do método de Gauss-Seidel, com o intuito de acelerar a convergência

(Werneck, 2016). O método se baseia na introdução do fator de relaxação ω, cujos valores estão contidos no intervalo $0 < \omega < 2$. Se $\omega < 1$, o método é classificado como de sub-relaxação. Se $\omega = 1$, então recupera-se o método de Gauss Seidel (Reis da Silva et al., 2013). Neste artigo, usa-se o método SOR com um valor padrão de ω igual a 1,5.

Algoritmo 3: Método de Sobre-relaxação Sucessiva

Entrada: forneça **A**, **b**, a dimensão do vetor **b** (n), $\mathbf{x}^{(0)}$, o parâmetro de sobre-relaxação ω ($0 < \omega < 2$), o número máximo de iterações $nmax$ e a tolerância tol;

para $k = 0 : nmax$ **faça**
  para $i = 0 : (n-1)$ **faça**
    $x_i^{k+1} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$
  fim
  se
    $max\left|x_i^{(k+1)} - x_i^{(k)}\right| < tol \quad para \quad 0 \leq i \leq (n-1)$
    então
      $\mathbf{x} = \mathbf{x}^{(k+1)}$
  fim
  senão
    Se $k = nmax$, não houve convergência.
  fim
fim

5.5 Fluxograma para um passo de tempo

O simulador numérico foi escrito usando a linguagem de programação padrão C. Além disso, o sistema GIT e a IDE Eclipse foram utilizados no gerenciamento de versões e na compilação e execução do código numérico. Na Fig. 3 encontra-se o fluxograma da resolução numérica para o processo iterativo considerando um único passo de tempo (Δt).

5.6 Aplicação da API OpenMP

A grande vantagem do OpenMP é a sua capacidade de otimizar a execução de tarefas que podem ser distribuídas pelas *threads*, para acelerar a execução de algoritmos complexos, geralmente com laços aninhados. No caso em questão, o método SOR (Jacobi ou Gauss-Seidel) é chamado repetidas vezes para fornecer a solução do sistema linear de equações algébricas a cada passo de tempo, até que seja atingida a convergência para uma dada tolerância (*tol*). O esforço computacional despendido cresce com o aumento do tempo físico de produção do reservatório e do número de células da malha computacional, bem como com a redução do valor da tolerância.

A paralelização no OpenMP funciona através do modelo *fork-join* (Fig. 4). A execução do programa inicia com uma *thread* mestre e outras *threads* são acionadas, para a realização de uma dada sequência de operações, quando encontrada uma diretiva *parallel*. Essa diretiva é responsável por demarcar as regiões do código que são executadas por múltiplas *threads* (*fork*). Após a execução das tarefas designadas, a *thread* mestre continua sua execução sequencial (*join*) até que uma nova região paralela seja encontrada, ou até que o programa chegue ao final da sua execução (Werneck, 2016).

Para paralelizar um processo iterativo em um laço, basta introduzir a linha de código *#pragma omp parallel for* (OpenMP Architecture Review Board, 2018). Dessa forma, apenas o laço *for* estritamente abaixo dessa linha de comando é executado em paralelo pelas *threads*. O número de *threads* empregadas pode ser especificado no início do código, ao ser adicionada a diretiva *omp_set_num_threads(int threads)*, ou através da chamada do compilador com a opção *OMP_NUM_THREADS=n* (Intel, 2019). Essas alterações também podem ser introduzidas ao longo do código, antes de cada diretiva *#pragma omp parallel for*, caso seja necessário alterar o número de *threads* utilizadas em diferentes partes do código paralelizado.

Vale enfatizar que a divisão da execução do código (usando várias *threads*) ocorre cada vez que a diretiva *#pragma omp parallel* é encontrada (*fork*), e que a junção (*join*) ocorre ao final da execução da região do código designada para ser paralelizada. Uma determinada região do código também pode ser executada em paralelo através da diretiva *#pragma omp parallel* (OpenMP Architecture Review Board, 2018). Assim sendo, toda sequência de comandos do código dessa região é executada por um grupo de *threads*. A definição de regiões paralelas poupa esforço computacional ao reduzir a quantidade de chamadas do tipo *fork-join* (Chapman et al., 2008).

Algumas vezes, a criação de uma região do código para a sua execução em paralelo pode ser vantajosa. Porém, toda distribuição de trabalho entre as *threads* deve ser gerenciada para que o código seja executado corretamente e que seja obtida uma maior eficiência computacional. A seguir, são apresentadas de forma sucinta as diretivas e cláusulas utilizadas na paralelização dos métodos iterativos estacionários.

5.6.1 *Variáveis privadas*

Para o funcionamento correto dos laços paralelizados é necessário, também, definir as variáveis que são privadas a uma determinada tarefa. Deixar de definir as variáveis privadas pode causar problemas de consistência de memória, possibilitando que as variáveis internas de uma determinada tarefa sejam acessadas e alteradas por outras *threads*, que estão executando outras tarefas (Chapman et al., 2008). Esse problema pode ser resolvido através da adição da cláusula *private(var1,var2,...)* (OpenMP Architecture Review Board, 2018), onde *var1,var2,...* são as variáveis privadas daquela tarefa.

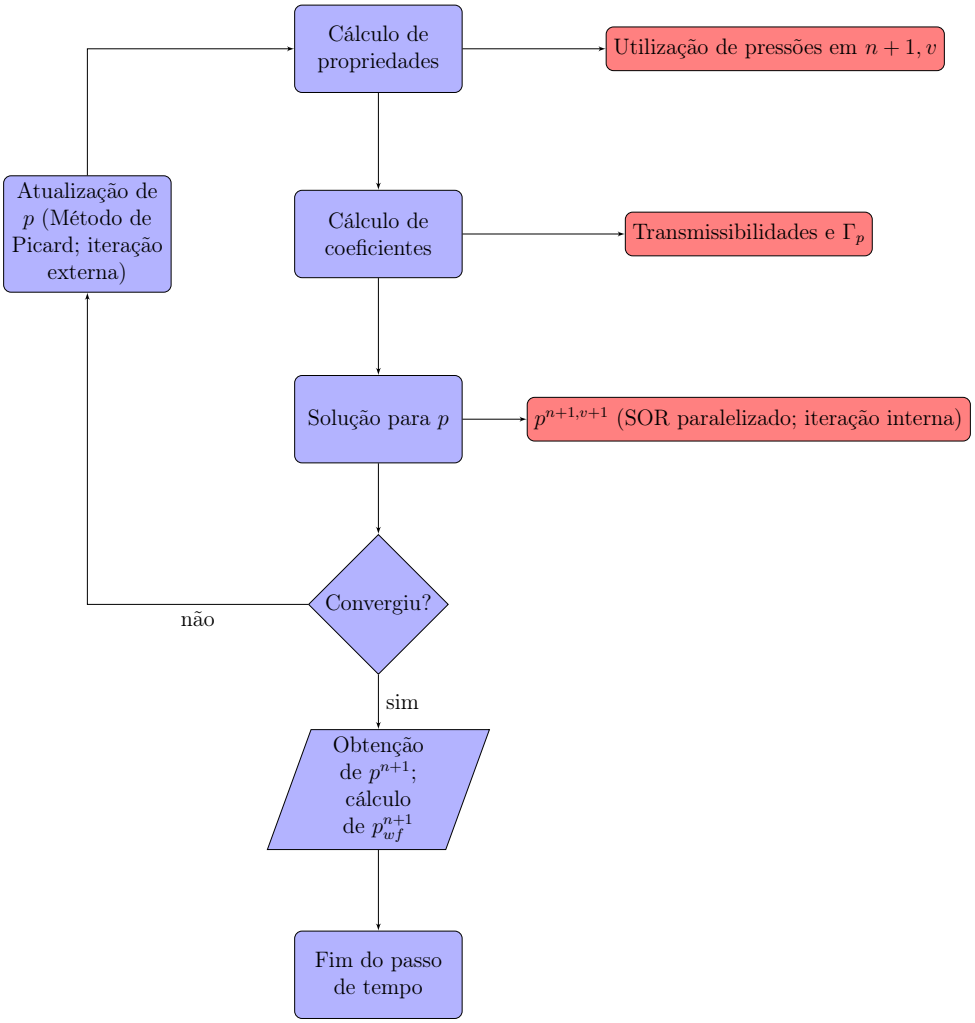

Figura 3: Fluxograma de solução para um passo de tempo

5.6.2 *Variáveis compartilhadas*

Além das variáveis privadas, em certas situações, algumas variáveis podem ser acessadas ao mesmo tempo por mais de uma *thread*. Nesse caso, uma determinada *thread* pode sobrescrever o valor de uma variável que está sendo utilizada, ao mesmo tempo, em uma outra tarefa por uma outra *thread*. Isso é feito por intermédio da cláusula *default(shared)* (OpenMP Architecture Review Board, 2018) na abertura da região paralelizada para definir que, por padrão, todas as variáveis contidas naquela região serão compartilhadas pelas *threads* (OpenMP Architecture Review Board, 2018).

5.6.3 *Partes sequenciais de execução única*

Dentro de uma região do código, paralelizada, ainda podem existir trechos que devem ser executados sequencialmente. Por padrão, as *threads* são livres para executar as tarefas de uma dada região. Portanto, uma maneira de se evitar que uma dada sequência de comandos seja executada por mais de uma *thread*, numa região paralelizada, é através da diretiva *#pragma omp single* (OpenMP Architecture Review Board, 2018). É importante destacar que uma outra opção seria a utilização da diretiva *#pragma omp critical* (OpenMP Architecture Review Board, 2018), que faz com que a região seja executada somente pela *thread* mestre. Entretanto, essa opção acarreta um maior esforço computacional (Chapman et al., 2008) e não é imprescindível.

5.6.4 *Laços aninhados*

Laços aninhados podem ser executados de modo independente pelas *threads* se as operações aninhadas forem independentes entre si. A introdução das diretivas *#pragma omp parallel for* acima dos laços fará com que apenas as iterações do laço mais externo sejam tratadas em paralelo. Para garantir que a paralelização atuará também nos laços internos é necessário incluir a cláusula *collapse(n)*, onde *n* especifica o número de laços *for* que são colapsados em uma sequência iterativa maior que é paralelizada (Intel, 2019). Tal cláusula não pode ser aplicada na paralelização dos métodos iterativos estacionários aqui considerados, uma vez que os laços internos não são independentes. Portanto, optou-se por paralelizar apenas os laços externos.

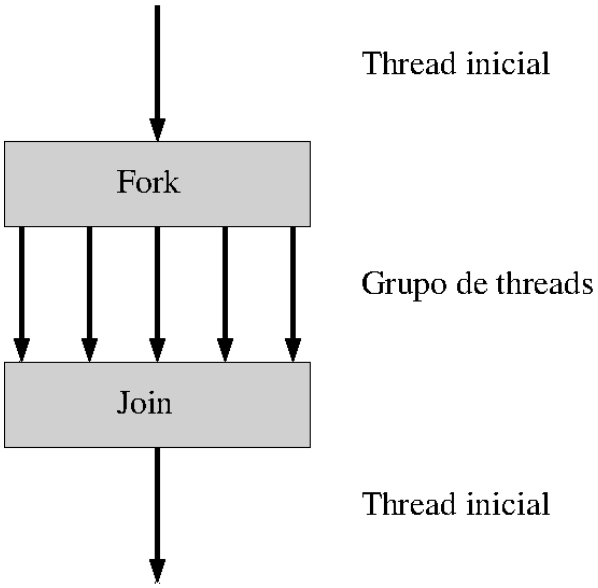

Figura 4: Modelo *fork-join*

5.6.5 *Somatórios*

Uma atenção especial também deve ser dada ao caso das variáveis privadas, de uma região paralelizada, que são utilizadas em operações acumulativas (somatórios) ao longo das sequências iterativas do código. Ao se adicionar a diretiva *reduction(operador:lista)* (OpenMP Architecture Review Board, 2018) pode-se garantir que o valor dos somatórios não seja sobrescrito por conta do trabalho independente das *threads*.

5.7 Versões paralelizadas dos métodos estacionários

O objetivo principal deste trabalho é a paralelização dos métodos iterativos estacionários de Jacobi, Gauss-Seidel e SOR introduzidos na Seção 5.1, onde os seus respectivos algoritmos foram fornecidos. Agora, nos Algoritmos 4, 5 e 6 são apresentadas as novas versões paralelizadas dos algoritmos empregando as diretivas e cláusulas da API OpenMP.

6 Resultados Numéricos

Nesta seção são apresentados os resultados numéricos, obtidos a partir da implementação paralelizada dos métodos iterativos estacionários, correspondentes às simulações do escoamento monofásico isotérmico em um reservatório portador de gás natural. Eles foram determinados através da variação de parâmetros numéricos e físicos. Também são fornecidos, para alguns casos, os tempos de execução computacional das versões seriais e paralelizadas.

Algoritmo 4: Método de Jacobi paralelizado

Entrada: forneça **A**, **b**, a dimensão do vetor **b** (n), $\mathbf{x}^{(0)}$, o número máximo de iterações ($nmax$) e a tolerância tol;

```
1  k = 0
2  #pragma omp parallel
3  enquanto k < nmax faça
4      #pragma omp for private(i,j,n,aux)
5      para i = 0 : (n − 1) faça
6          para j = 1 : n faça
7              se j ≠ i então
8                  aux = a_ij x_j^(k)
9              fim
10             x_i^(k+1) = (1/a_ii) (b_i − aux)
11         fim
12     fim
13     #pragma omp for private(i,n)
        reduction(+:erro)
14     para i = 0 : (n − 1) faça
15         erro = erro + |x_i^(k+1) − x_i^(k)|
16     fim
17     se max(erro) < tol  para  0 ≤ i ≤ (n − 1) então
18         x = x^(k+1)
19         k = k + 1
20     fim
21     senão
22         Se k = nmax, não houve convergência.
23     fim
24 fim
```

6.1 Conjunto padrão de dados

Baseado em dados coletados na literatura, tais como nos trabalhos de Jiang and Younis (2015) e Li et al. (2016), escolheu-se um conjunto de referência para os dados de entrada, os quais estão disponíveis na Tabela 1. A referida tabela também traz as informações de referência para as simulações incorporando o efeito de escorregamento, usando o modelo de Klinkenberg e a correção dependente do número de Knudsen. O método padrão para as simulações é o SOR, à exceção dos resultados destinados ao estudo comparativo do desempenho dos três métodos estacionários considerados neste trabalho.

Também, para o conjunto referência, a produção de gás é viabilizada por meio de um poço vertical de comprimento L_{wf}, centralizado no plano xy, e mantendo-se uma vazão de produção constante, Q_{sc}. A pressão no poço é calculada através do modelo de acoplamento poço-reservatório já apresentado na Seção 4.2.

Utiliza-se um passo de tempo variável no início da simulação (Δt_{ini}) e uma razão de crescimento do passo de tempo ($\delta_{\Delta t}$) é estipulada. Quando o seu valor máximo (Δt_{max}) é alcançado, ele é mantido constante até o término da simulação (t_{max}). Esse modelo de simulação é utilizado na indústria do petróleo quando se deseja de-

Algoritmo 5: Método de Gauss-Seidel paralelizado

Entrada: forneça **A**, **b**, a dimensão do vetor **b** (n), $\mathbf{x}^{(0)}$, o número máximo de iterações ($nmax$) e a tolerância tol;

$k = 0$
#pragma omp parallel
enquanto $k < nmax$ **faça**
  #pragma omp for private(i,j,n,aux,aux1)
  para $i = 0 : (n-1)$ **faça**
    para $j = 1 : (i-1)$ **faça**
      $aux = a_{ij} x_j^{(k+1)}$
    fim
    para $j = i+1 : n$ **faça**
      $aux1 = a_{ij} x_j^{(k)}$
    fim
    $x_i^{k+1} = \frac{1}{a_{ii}} (b_i - aux - aux1)$
  fim
  #pragma omp for private(i,n) reduction(+:erro)
  para $i = 0 : (n-1)$ **faça**
    $erro = erro + \left| x_i^{(k+1)} - x_i^{(k)} \right|$
  fim
  se $max(erro) < tol \quad para \quad 0 \leq i \leq (n-1)$ **então**
    $\mathbf{x} = \mathbf{x}^{(k+1)}$
    $k = k + 1$
  fim
  senão
    Se $k = nmax$, não houve convergência.
  fim
fim

Algoritmo 6: Método de Sobre-relaxação Sucessiva paralelizado

Entrada: forneça **A**, **b**, a dimensão do vetor **b** (n), $\mathbf{x}^{(0)}$, o parâmetro de sobre-relaxação ω ($0 < \omega < 2$), o número máximo de iterações $nmax$ e a tolerância tol;

$k = 0$
#pragma omp parallel
enquanto $k < nmax$ **faça**
  #pragma omp for private(i,j,n,aux,aux1)
  para $i = 0 : (n-1)$ **faça**
    para $j = 1 : (i-1)$ **faça**
      $aux = a_{ij} x_j^{(k+1)}$
    fim
    para $j = i+1 : n$ **faça**
      $aux1 = a_{ij} x_j^{(k)}$
    fim
    $x_i^{k+1} = (1-\omega) x_i^{(k)} + \frac{\omega}{a_{ii}} (b_i - aux - aux1)$
  fim
  #pragma omp for private(i,n) reduction(+:erro)
  para $i = 0 : (n-1)$ **faça**
    $erro = erro + \left| x_i^{(k+1)} - x_i^{(k)} \right|$
  fim
  se $max(erro) < tol \quad para \quad 0 \leq i \leq (n-1)$ **então**
    $\mathbf{x} = \mathbf{x}^{(k+1)}$
    $k = k + 1$
  fim
  senão
    Se $k = nmax$, não houve convergência.
  fim
fim

terminar, com maior nível de detalhamento, a pressão no poço nos instantes iniciais de produção (Werneck, 2016), onde os gradientes de pressão são mais elevados (ou ocorrem mudanças na vazão no poço).

O computador usado nas simulações da Seção 6.3.1 foi um Dell PowerEdge T620 com sistema operacional OpenSuse Leap 15.1, apresentando as seguintes principais características:

- Máquina de arquitetura 64 bits,
- 2 Processadores Intel Xeon E5-2620 com 2 GHz de frequência básica e 6 núcleos (12 *threads*),
- Memória RAM de 16 GB.

Entretanto, em função do maior número de células das Malhas 6, 7 e 8 (Tabela 4), as simulações da Seção 6.3.2 exigiram uma máquina com uma capacidade maior de memória. Portanto, os resultados foram obtidos em um servidor PowerEdge R730 com sistema operacional CentOS 7.3.1611:

- Máquina de arquitetura 64 bits,
- 2 Processadores Intel Xeon E5-2620 v.3 com 2,4 GHz de frequência básica e 6 núcleos (12 *threads*),
- Memória RAM de 48 GB.

Para esta subseção específica, são apresentados os valores do tempo de execução médio ($\bar{t}_{exe}$), do seu desvio padrão (σ) e do *speedup* (S) para cada simulação, em função do número de *threads* (N_{thread}) e da malha empregada. Um total de cinco (5) execuções foi considerado na obtenção dos valores do tempo médio e do desvio padrão.

6.2 Refinamento de malha

A Tabela 2 mostra o número de células das diferentes malhas computacionais (n_x e n_y) empregadas no estudo do refinamento de malha. O número de células na direção z (n_z) é mantido constante e igual 1 para todas as simulações. Utiliza-se os valores padrão da Tabela 1 e considera-se que o escoamento é do tipo não-Darcy.

Os resultados foram obtidos com as versões serial e paralelizada do código numérico. Destaca-se o fato de que a versão serial do simulador já havia sido testada por de Souza (2013), que comparou os seus resultados com os obtidos com o simulador comercial IMEX (CMG, 2009) para escoamentos governados pela lei de Darcy

Tabela 1: Parâmetros para o caso de referência

Parâmetro	Valor	Unidade
b_K (Eq.(2))	250,0	psi
c_ϕ	$1{,}0\times10^{-6}$	psi^{-1}
k_x e k_y	$5{,}0\times10^{-6}$	Darcy
L_x e L_y	6.400	ft
L_z	40,0	ft
L_{wf}	40,0	ft
n_x e n_y	1.281	–
p_{ini} e p^0	6.000	psi
p_{sc}	14,65	psi
Q_{sc}	$-2{,}0\times10^{4}$	scf/dia
t_{max}	300	dia
tol	1×10^{-7}	psi
T	609,67	R
T_{sc}	519,67	R
R	10,73	$ft^3psi/R\ lbm-mol$
γ	0,6	–
$\delta_{\Delta t}$	1,5	–
Δt_{ini}	0,001	dia
Δt_{max}	20,0	dia
ϕ_{ini} e ϕ^0	0,12	–
τ	1,41	–

Tabela 2: Refinamento de malha

Malha	n_x	n_y
1	81	81
2	161	161
3	321	321
4	641	641
5	1.281	1.281

clássica. O estudo do refinamento de malha mostrou que os resultados, para a pressão no poço p_{wf}, ficaram cada vez mais próximos um dos outros à medida que a malha era refinada, indicando a convergência numérica. Em função da constatação da convergência numérica, a Malha 5 foi considerada ser suficientemente refinada para fornecer resultados acurados e foi adotada como padrão para as simulações da Seção 6.3.1.

Para a Malha 5, na Fig. 5 são apresentados os valores da pressão no poço calculados com as versões serial e paralelizada. Os resultados mostram que os mesmos valores são encontrados com as duas versões. Como comentário adicional do ponto de vista físico, vale ressaltar que como a permeabilidade considerada é baixa, nessas simulações não surgiram os efeitos de fronteira, os quais provocariam mudanças de inclinação na reta presente no gráfico de pressão para tempos finais de produção (de Souza, 2013).

6.3 Desempenho computacional

Analisa-se, agora, os resultados das simulações sob o ponto de vista do desempenho computacional.

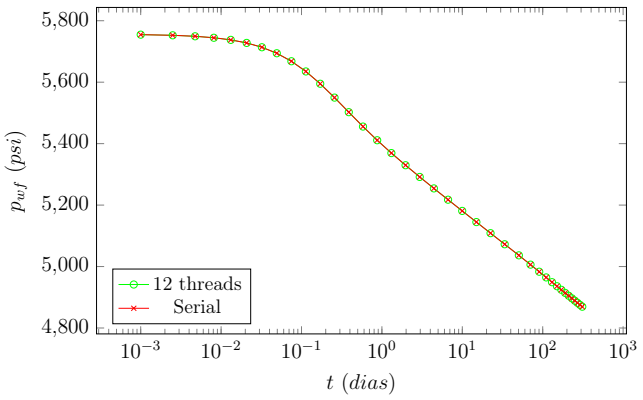

Figura 5: Pressão no poço em função do tempo

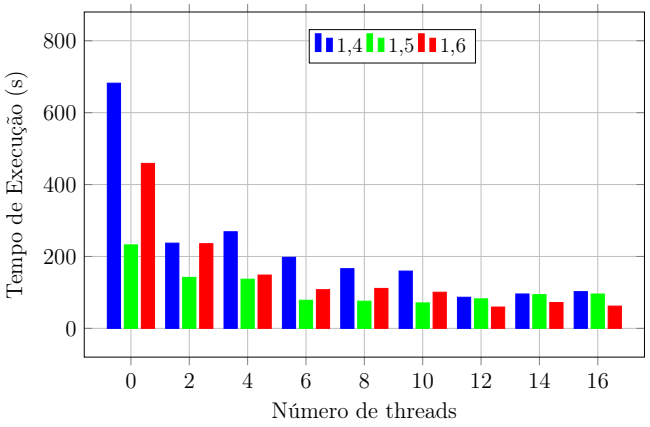

Figura 6: Tempos de execução para ω=1,4, 1,5 e 1,6

6.3.1 *Variação de parâmetros*

Todas as simulações foram realizadas, inicialmente, com uma malha composta de 1.281 × 1.281 células (n_x = n_y = 1.281), correspondente à Malha 5 da Tabela 2.

Apesar de não ter implicação física, o fator de relaxação ω possui um efeito direto no desempenho computacional, já que o número de iterações no método SOR, para se alcançar a convergência, dependente da escolha desse fator. O efeito da escolha de ω sobre o desempenho computacional pode ser observado na Fig. 6 (o número de *threads* nulo, nos gráficos, corresponde à execução em série), onde o tempo de simulação é menor quando $\omega = 1,5$. Reitera-se que há maneiras de se escolher um valor otimizado para ω (Ertekin et al., 2001), contudo, essa questão está fora do escopo deste trabalho.

A variação da permeabilidade absoluta, da densidade do fluido e da vazão também acarretam um esforço computacional diferenciado para cada caso. Avalia-se o desempenho em termos do tempo de execução, ao se utilizar a computação paralela, e os respectivos resultados são apresentados nas Figs. 7 a 9. Também fez-se um teste considerando a variação dos modelos de escoamento com e sem os efeitos oriundos do escorregamento do gás da Tabela 3 (Fig. 10).

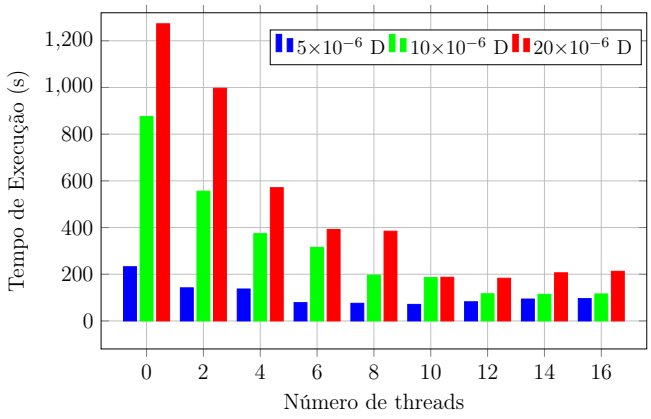

Figura 7: Tempos de execução para k_x=k_y=5×10^{-6} D, 10×10^{-6} D e 20×10^{-6} D

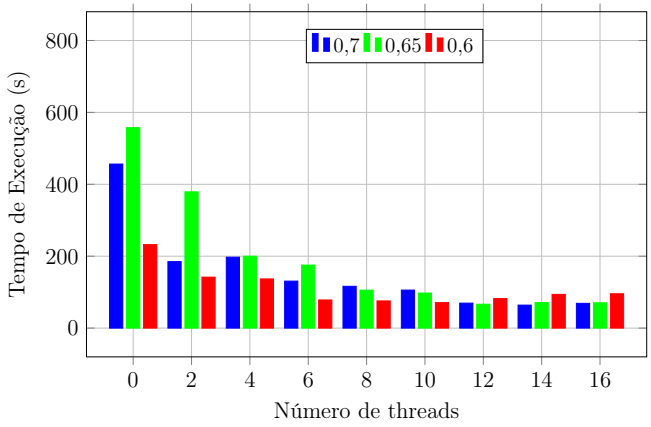

Figura 8: Tempos de execução para γ=0,7, 0,65 e 0,6

Dos testes realizados, nesta etapa, é possível observar que para um aumento na permeabilidade absoluta corresponde um aumento no tempo de execução. Como a permeabilidade influencia diretamente nos valores das transmissibilidades, ela introduz uma modificação nos elementos da matriz dos coeficientes do sistema de equações algébricas (Eq. (32)).

No caso da densidade, o valor intermediário, 0,65, levou aos maiores tempos de execução. Lembra-se que a densidade influencia no cálculo dos valores da viscosidade e do fator volume de formação do gás, sendo que a viscosidade afeta os valores das transmissibilidades, enquanto que B também altera os valores da transmissibilidade e do coeficiente Γ_p. Assim sendo, acredita-se que o comportamento encontrado para a variação da densidade é devido às não linearidades associadas ao problema.

A influência das não linearidades parece ter ocorrido também no caso dos testes para o aumento da vazão. No caso serial, a simulação com o valor de referência (intermediário), Q_{sc} = −20 × 10^3 scf/dia, teve o maior tempo de execução. Ressalta-se que diferentes valores da vazão conduzem a diferentes quedas de pressão, mudando o nível de pressão e alterando, por conseguinte,

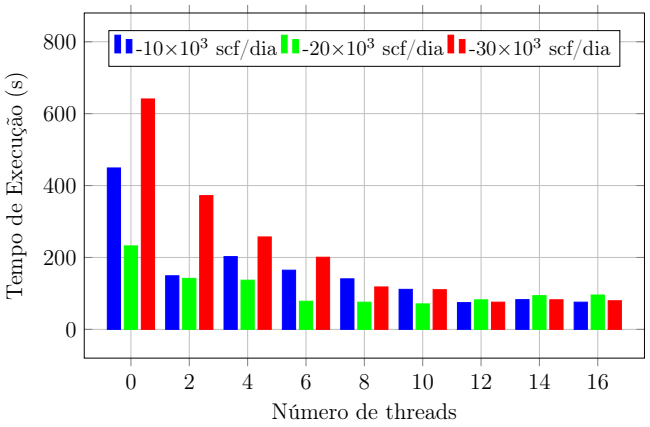

Figura 9: Tempos de execução para Q_{sc}=−10×10^3 scf/dia, −20×10^3 scf/dia e −30×10^3 scf/dia

os valores do coeficiente Γ_p, de μ e de B.

A Tabela 3 mostra os três modelos de cálculo da permeabilidade aparente utilizados nas simulações. O Modelo 1 considera que a permeabilidade aparente é simplesmente igual à permeabilidade absoluta. Já os Modelos 2 e 3 contemplam o fenômeno do escorregamento do gás, uma vez que os valores da permeabilidade aparente variam em função da pressão e do número de Knudsen (Chung et al., 2015), respectivamente.

Tabela 3: Modelos para a permeabilidade aparente

Modelo	Referência	$\mathbf{k}_a$
1	Ertekin et al. (2001)	$\mathbf{k}$
2	Li et al. (2016)	$f(K_n)\mathbf{k}$
3	Klinkenberg (1941)	$(1 + b/p)\,\mathbf{k}$

No caso das variações do modelo de escoamento, incluindo ou não o efeito do escorregamento, o uso da permeabilidade aparente variável aumenta a não linearidade e os maiores tempos de execução correspondem aos Modelos 2 e 3.

É interessante notar que para os testes variando a vazão e a densidade há a ocorrência de tempos de execução maiores para o uso de 4 *threads* do que para duas, o que certamente pode ser explicado em função das características intrínsecas ao modelo *fork-join* e às cláusulas de barreira (Werneck et al., 2019).

6.3.2 *Variação do número de células*

Em seguida, trata-se das análises de desempenho utilizando o *speedup* como parâmetro da medida do desempenho computacional, ou seja, a relação entre os tempos de simulação (médio) serial e em paralelo. Tendo esse parâmetro em vista, é possível estudar o desempenho em função da variação do número de *threads* utilizadas nas execuções para uma dada malha, assim como o desempenho em função da variação do número de células da malha computacional, para um dado nú-

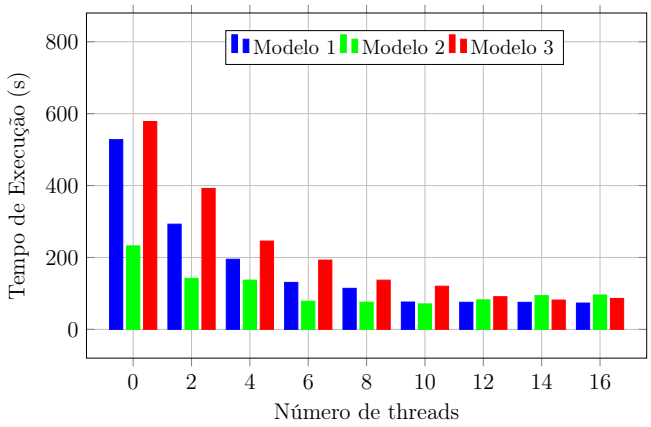

Figura 10: Tempos de execução empregando os Modelos 1, 2 e 3

mero de *threads*. Nesse caso, foram empregadas 8 (oito) diferentes malhas ao invés das cinco utilizadas no estudo do refinamento de malha e da Seção 6.3.1, de modo a se trabalhar com um número bem superior de células. Os valores de n_x e n_y podem ser vistos na Tabela 4.

Tabela 4: Malhas para os testes de *speedup*

Malha	n_x	n_y
1	81	81
2	161	161
3	321	321
4	641	641
5	1.281	1.281
6	2.561	2.561
7	5.121	5.121
8	10.241	10.241

Baseando-se nos resultados apresentados anteriormente, para o caso padrão, são fornecidas nas Tabelas 5 a 8 os valores do tempo de execução médio, do desvio padrão e do *speedup* para as oito malhas distintas. Nelas, o desempenho computacional pode ser analisado através das variações do *speedup* em função da malha e do número de *threads* empregados quando da execução do código. É possível observar que com o aumento do número de células aumenta também o número de cálculos a serem realizados, pois aumenta-se a ordem do sistema de equações a ser resolvido, com um consequente acréscimo no ganho do desempenho computacional quando utilizada a computação paralela. Deve-se levar em conta que nem todas as partes código numérico do simulador foram paralelizadas, de forma que ainda existem partes do código onde há um aumento do esforço computacional, resultante do aumento do número de nós da malha, e que não são influenciadas pela paralelização. Tal fato também ajuda a explicar a não linearidade do comportamento da variação do *speedup*.

Apesar do crescimento inicial do *speedup* à medida

Tabela 5: Tempo de execução médio ($\bar{t}_{exe}$), desvio padrão (σ) e *speedup* (S) para as Malhas 1 e 2

N_{thread}	Malha 1			Malha 2		
	$\bar{t}_{exe}$ (s)	σ	S	$\bar{t}_{exe}$ (s)	σ	S
Serial	0,16	0,001	-	0,65	0,002	-
2	0,15	0,004	1,07	0,53	0,007	1,24
4	0,12	0,001	1,38	0,43	0,016	1,50
6	0,11	0,002	1,45	0,39	0,015	1,66
8	0,11	0,001	1,47	0,37	0,007	1,76
10	0,11	0,002	1,43	0,39	0,019	1,68
12	0,13	0,009	1,26	0,37	0,010	1,74
14	0,16	0,016	1,03	0,40	0,018	1,65
16	0,19	0,014	0,86	0,39	0,005	1,67

Tabela 6: Tempo de execução médio ($\bar{t}_{exe}$), desvio padrão (σ) e *speedup* (S) para as Malhas 3 e 4

N_{thread}	Malha 3			Malha 4		
	$\bar{t}_{exe}$ (s)	σ	S	$\bar{t}_{exe}$ (s)	σ	S
Serial	2,61	0,004	-	10,59	0,021	-
2	1,97	0,045	1,32	8,47	0,014	1,25
4	1,74	0,063	1,50	7,29	0,078	1,45
6	1,42	0,050	1,84	6,31	0,169	1,68
8	1,33	0,029	1,97	5,75	0,053	1,84
10	1,28	0,019	2,04	5,60	0,382	1,89
12	1,34	0,083	1,95	5,47	0,354	1,94
14	1,34	0,027	1,95	5,79	0,047	1,83
16	1,31	0,017	2,00	5,70	0,141	1,86

que o número de *threads* é aumentado, nota-se que um valor máximo é atingido para um número de *threads* compreendido entre 8 e 12 (máximo de 7,43 com 12 *threads* e a Malha 8 para o SOR). Tal fato é conhecido e relatado na literatura (de Salles et al., 2018, Werneck et al., 2019) e se deve, entre outros fatores, ao modelo *fork-join*, cláusulas de barreira e ao possível desbalanceamento da distribuição de carga entre as *threads*. Portanto, para que melhores resultados sejam almejados, uma paralelização utilizando uma programação mais avançada com as diretivas e cláusulas do OpenMP seria necessária, levando em conta que o simulador numérico já existia na sua versão serial. Entretanto, este não era o objetivo primordial deste trabalho e, em termos práticos, os ganhos obtidos já são suficientes para que vários problemas de interesse da simulação de reservatórios sejam abordados e forneçam resultados próximos dos reais.

A última tabela (Tabela 9) corresponde a um estudo comparativo entre o desempenho do código numérico utilizando os três métodos estacionários de Jacobi, de Gauss-Seidel e SOR e empregando a Malha 7. Nela, verifica-se que os tempos de execução correspondem, na ordem decrescente, aos métodos de Jacobi, Gauss-Seidel e SOR. Após a paralelização desses respectivos métodos, essa ordem não é alterada, mas os maiores valores do *speedup* são obtidos para os métodos de Jacobi (8,35 com 12 *threads*) e SOR (6,16 com 12 *threads*). Em contrapartida, o máximo *speedup* atingido para o método Gauss-Seidel foi de 3,13 com 12 *threads*. Lembrando que esses valores correspondem aos obtidos com a Malha 7 e que valores superiores foram determi-

Tabela 7: Tempo de execução médio ($\bar{t}_{exe}$), desvio padrão (σ) e *speedup* (S) para as Malhas 5 e 6

N_{thread}	Malha 5			Malha 6		
	$\bar{t}_{exe}$ (s)	σ	S	$\bar{t}_{exe}$ (s)	σ	S
Serial	59,39	0,10	–	623,70	0,94	–
2	40,73	0,22	1,46	360,22	11,42	1,73
4	30,65	1,16	1,94	235,37	10,05	2,65
6	28,94	0,78	2,05	187,16	5,88	3,32
8	30,57	2,47	1,94	169,86	7,02	3,67
10	28,41	1,78	2,09	166,01	11,26	3,76
12	29,40	1,32	2,02	170,81	7,17	3,65
14	29,01	1,74	2,05	179,51	10,16	3,47
16	29,14	1,44	2,04	184,40	8,63	3,38

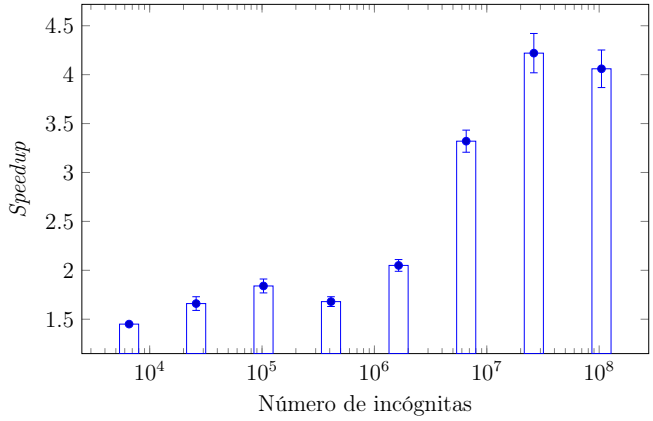

Figura 11: *Speedup* por número de incógnitas empregando 6 *threads*

nados para o método SOR com a Malha 8 (7,43 com 12 *threads*), conforme os valores da Tabela 8.

Ainda com respeito aos resultados da Tabela 9, deve-se chamar a atenção para o fato do comportamento *superlinear* encontrado com 2 *threads* para o método de Jacobi. De acordo com Chapman et al. (2008), a aceleração superlinear é um ganho de desempenho que leva a um valor do *speedup* que é maior que o número *threads* utilizadas. Segundo os autores, isso pode ser atribuído, principalmente, ao fato de que uma maior quantidade de memória *cache* está disponível ao se usar mais de um processador. Além disso, os valores de *speedup* aqui calculados consideram as médias dos tempos de execução considerando-se um total de cinco simulações.

As Figs. 11 e 12 mostram a variação do *speedup*, respectivamente, ao se utilizar 6 e 12 *threads* e diferentes números de células (número de incógnitas do sistema algébrico). Da análise conjunta das duas figuras, percebe-se que os valores mais altos do *speedup* foram atingidos quando usou-se 12 *threads*, o que faz sentido pelo maior número de recursos disponíveis para a execução do código em paralelo, considerando que os *speedups* máximos foram obtidos para o número de *threads* variando entre 8 e 12.

Nos dois gráficos das Figs. 11 e 12 é também possível se observar uma região de crescimento do *speedup*, seguida por uma queda (ou estagnação), sendo que para 6

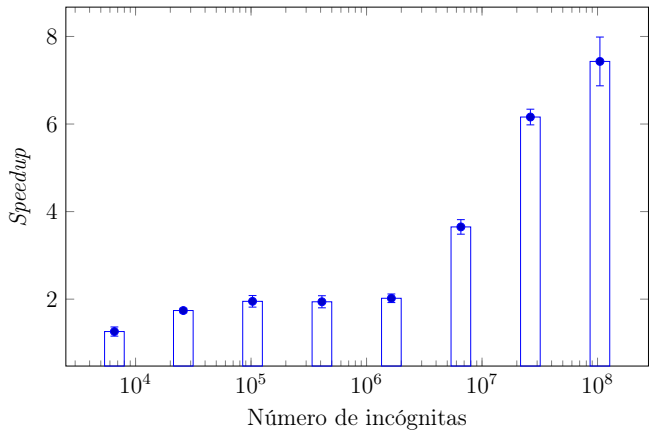

Figura 12: *Speedup* por número de incógnitas empregando 12 *threads*

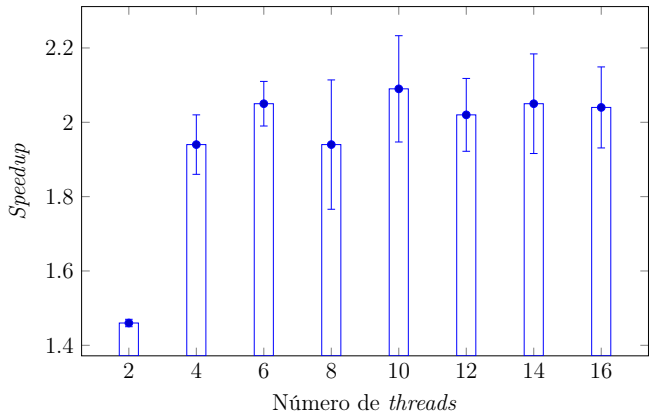

Figura 13: *Speedup* por número de *threads* para a Malha 5

threads existe mais uma queda trazendo o *speedup* para valores próximos aos das primeiras malhas avaliadas. Já com 12 *threads* o maior valor registrado ocorreu para a malha com o maior número de nós. Acredita-se que as oscilações, nesses gráficos, se devem (conforme já mencionado) ao balanço nos processos *fork-join*, de forma que para certos problemas a relação custo-benefício em utilizar o OpenMP depende de um bom ajuste entre as exigências dos problemas e os recursos mobilizados para a solução.

Por outro lado, é possível visualizar nas Figs. 13 a 16 o aumento do *speedup* em função da variação do número de *threads* para os quatro casos correspondentes aos maiores números de células, ou seja, às Malhas 5, 6, 7 e 8. Da análise dos resultados verifica-se que o ganho de desempenho não é linear e que de acordo com o aumento do número de *threads*, o *speedup* tendeu a uma quase estagnação (Malha 6) ou até mesmo a uma diminuição (Malha 5). As questões levantadas no parágrafo precedente também podem ser aplicadas aqui, como uma explicação para o comportamento reportado.

Tabela 8: Tempo de execução médio ($\bar{t}_{exe}$), desvio padrão (σ) e *speedup* (S) para as Malhas 7 e 8

N_{thread}	Malha 7			Malha 8		
	$\bar{t}_{exe}$ (s)	σ	S	$\bar{t}_{exe}$ (s)	σ	S
Serial	7.408,13	0,74	-	95.099,59	2.196,80	-
2	4.072,47	77,38	1,82	50.281,25	357,00	1,89
4	2.272,83	38,41	3,26	29.317,06	1.263,57	3,24
6	1.756,19	79,73	4,22	23.420,46	541,01	4,06
8	1.449,92	49,88	5,11	19.512,46	599,03	4,87
10	1.265,58	33,03	5,85	15.073,90	1.050,65	6,31
12	1.202,71	33,92	6,16	12.800,44	616,98	7,43
14	1.375,61	31,50	5,39	15.637,53	1.319,81	6,08
16	1.275,43	44,00	5,81	14.302,25	461,96	6,65

Tabela 9: Tempo de execução médio ($\bar{t}_{exe}$), desvio padrão (σ) e *speedup* (S) para os diferentes métodos iterativos e a Malha 7

N_{thread}	Jacobi			Gauss-Seidel			SOR		
	$\bar{t}_{exe}$ (s)	σ	S	$\bar{t}_{exe}$ (s)	σ	S	$\bar{t}_{exe}$ (s)	σ	S
Serial	159.314,42	184,34	-	14.617,58	15,77	-	7.408,13	0,74	-
2	79.286,33	281,57	2,01	9.338,71	10,08	1,57	4.072,47	77,38	1,82
4	42.427,51	150,67	3,75	6.429,67	6,94	2,27	2.272,83	38,41	3,26
6	30.654,25	108,86	5,20	5.743,35	6,20	2,25	1.756,19	79,73	4,22
8	24.514,34	87,06	6,50	5.100,06	5,50	2,87	1.449,92	49,88	5,11
10	21.241,84	75,44	7,50	4.845,89	5,23	3,02	1.265,57	33,03	5,85
12	19.085,77	67,78	8,35	4.665,35	5,03	3,13	1.202,71	33,92	6,16
14	26.643,09	94,62	5,98	4.923,38	5,31	2,97	1.375,61	31,50	5,39
16	22.932,09	81,44	6,95	4.297,16	5,32	2,97	1.275,43	44,00	5,81

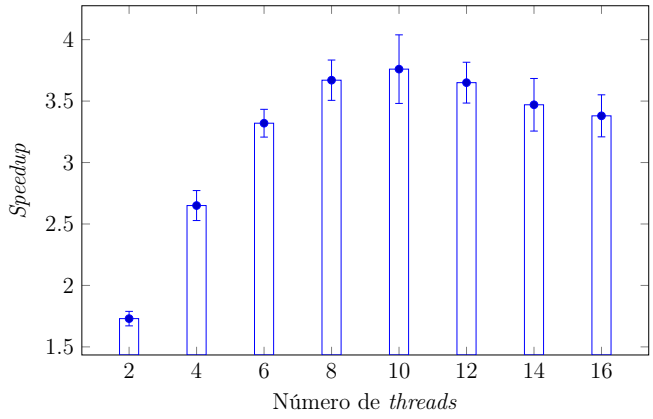

Figura 14: *Speedup* por número de *threads* para a Malha 6

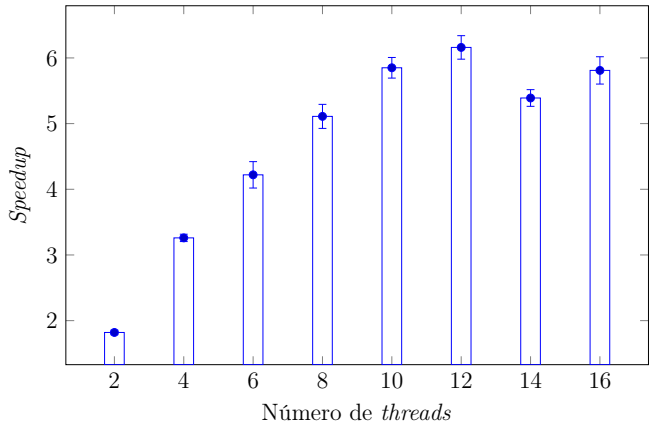

Figura 15: *Speedup* por número de *threads* para a Malha 7

7 Conclusões

Na busca da redução do tempo de execução de um simulador numérico, nem sempre o uso de um *hardware* mais potente é a melhor opção, seja por questões econômicas ou de desempenho. Neste trabalho, foi possível explorar a alternativa da paralelização via a API OpenMP para resolver de forma mais eficiente um problema físico complexo, a simulação do escoamento monofásico em um reservatório de gás natural. Vale ressaltar que o uso da computação paralela não altera, necessariamente, a complexidade do problema. Na prática, o ganho de desempenho não é linear ou equivalente ao de transformar um problema de complexidade $O(n^2)$ em $O(n)$. No entanto, foi possível se obter ganhos significativos com valores do *speedup* superior a 8 para o método de Jacobi e 6 para os métodos de Gauss-Seidel e SOR com a Malha 7.

No geral, verificou-se o crescimento do *speedup* em função do aumento do número de incógnitas do sistema de equações algébricas a ser resolvido (como consequência do maior número de nós da malha computacional) e, também, com o aumento do número de *threads* empregadas. Entretanto, como é sabido, o desempenho computacional está sujeito à ocorrência de oscilações e/ou estagnações que dependem das características in-

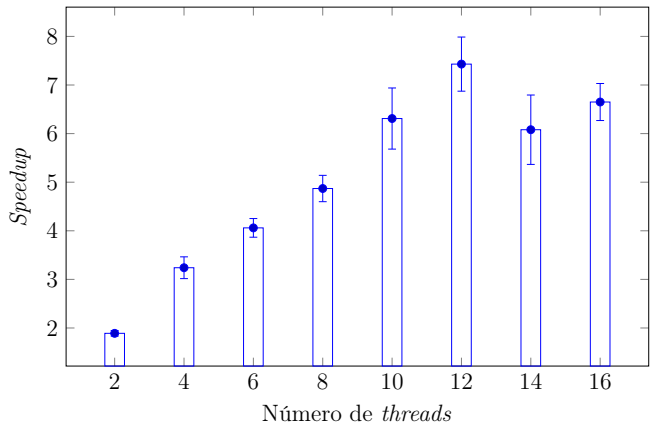

Figura 16: *Speedup* por número de *threads* para a Malha 8

trínsecas ao processo de *fork-join* e ao balanceamento da distribuição de tarefas entre as *threads*. Maiores ganhos poderiam, em teoria, ser obtidos mediante a reescritura do código computacional e o uso de uma programação mais avançada com o OpenMP. Entretanto, esse não era o objetivo principal deste estudo, que focou no uso direto das diretivas do OpenMP no programa de computador pré-existente.

Além das questões inerentes ao OpenMP, verificou-se que, conforme o esperado, a mudança dos parâmetros físicos pode afetar diretamente na variação do desempenho computacional, devido à influência dos mesmos na determinação dos valores dos elementos da matriz dos coeficientes associada ao sistema de equações algébricas que deve ser resolvido. Finalmente, cabe ressaltar que a solução paralelizada foi implementada de modo a preservar o comportamento fisicamente correto, obtidos com a versão serial, dos fenômenos estudados.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Código de Financiamento 001, e da FAPERJ.

Referências

Abou-Kassem, J. H., Farouq Ali, S. M. and Islam, M. R. (2006). *Petroleum Reservoir Simulation, A Basic Approach*, Gulf Publishing Company, Houston, USA.

Anderson, R. O. (1984). *Fundamentals of the petroleum industry*, University of Oklahoma Press, Oklahoma, United States of America.

ARB (2011). *OpenMP application program interface*, version 3.1 edn, Architecture Review Board.

Aziz, M. and Settari, A. (1990). *Petroleum Reservoir Simulation*, Elsevier Applied Science, New York, USA.

Bestok, A. e Karniadakis, G. E. (1999). A model for flows in channels, pipes, ans ducts at micro and nano scales, *Microscale Thermophysical Engineering* **3**: 43–77. https://doi.org/10.1080/108939599199864.

Carpen-Amarie, A., Hunold, S. and Träff, J. (2017). On expected and observed communication performance with mpi derived datatypes, *Parallel Computing* **69**: 98–117. https://doi.org/10.1145/2966884.2966905.

Chapman, B., Jost, G. and van der Pas, R. (2008). *Using OpenMP - Portable Shared Memory Parallel Programming*, The MIT Press, London, England.

Chen, Z., Huan, G. and Ma, Y. (2006). *Computational Methods for Multiphase Flows in Porous Media*, Society of Industrial and Applied Mathematics, Philadelphia, USA.

Chung, C. L., Freitas, M. M., Souza, G. and Souto, H. P. A. (2015). Numerical reservoir simulation of shale gas in the slip flow regime, *Ibero-Latin Congress on Computational Methods in Engineering, 36*, Rio de Janeiro.

CMG (2009). *IMEX, Advanced Oil/Gas Reservoir Simulator*, Computational Modeling Group.

Dandekar, A. Y. (2013). *Petroleum Reservoir Rock and Fluid Properties*, CRC Press, USA.

de Salles, R., Werneck, L., de Souza, G. and Amaral Souto, H. (2018). Aplicação de células inativas, Compressed Sparse Row e OpenMP na simulação numérica paralelizada de reservatórios de petróleo, *Revista Brasileira de Computação Aplicada* **10**(2): 64–79. https://doi.org/10.5335/rbca.v10i2.8055.

de Souza, G. (2013). *Acoplamento Poço-reservatório na Simulação Numérica de Reservatórios de Gás*, PhD thesis, Universidade Estadual do Norte Fluminense, Macaé, Brasil.

Dietrich, R., Schmitt, F., Grund, A. and Stolle, J. (2017). Critical-blame analysis for openmp 4.0 offloading on intel xeon phi, *Journal of Systems and Software* **125**: 381–388. https://doi.org/10.1016/j.jss.2015.12.050.

Dranchuk, P. and Abou-Kassem, J. (1975). Calculation of Z factors for natural gases using equations of state, *Journal of Canadian Petroleum Technology* **14**(3): 34–36. https://doi.org/10.2118/75-03-03.

Dyrdahl, J. (2014). *Thermal flow in fractured porous media and operator splitting*, Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Ertekin, T., Abou-Kassem, J. H. and King, G. R. (2001). *Basic Applied Reservoir Simulation*, SPE Textbook Series 7, Society of Petroleum Engineers, Richardson.

Florence, F. A., Rushing, J. A., Newsham, K. E. and Blasingame, T. A. (2007). Improved permeability prediction relations for low permeability sands, *Society*

of Petroleum Engineers Rock Mountain Oil & Gas Technology Symposium, Denver, Colorado, USA.

Halsey, T. C. (2016). Computational sciences in the upstream oil and gas industry, *Philos Trans A Math Phys Eng Sci.* **374**: 20150429. https://doi.org/10.1098/rsta.2015.0429.

Intel (2019). Developer guide for intel math kernel library for windows.

Islam, M. R., Moussavizadegan, S. H., Mustafiz, S. and Abou-Kassem, J. H. (2010). *Advanced Petroleum Reservoir Simulation*, first edn, Scrivener Publishing LLC., Salem, USA.

Jiang, J. and Younis, R. M. (2015). A multimechanistic multicontinuum model for simulating shale gas reservoir with complex fractured system, *Fuel* **161**: 333–344. https://doi.org/10.1016/j.fuel.2015.08.069.

Klinkenberg, L. J. (1941). The permeability of porous media to liquids and gases, *Drilling and Production Practice, American Petroleum Inst.* p. 200–213.

Lee, A., Gonzalez, M. and Eakin, B. (1966). The viscosity of natural gases, *Journal of Petroleum Technology, Transactions of AIME* **18**(8): 997–1000. https://doi.org/10.2118/1340-PA.

Li, D., Zhang, L., Wang, J. Y., Lu, D. and Du, J. (2016). Effect of adsorption and permeability correction on transient pressures in organic rich gas reservoirs: Vertical and hydraulically fractured horizontal wells, *Journal of Natural Gas Science and Engineering* **31**: 214–225. https://doi.org/10.1016/j.jngse.2016.02.033.

Nick, H. M., Raoof, A., Centler, F., Thullner, M. and Regnier, P. (2013). Reactive dispersive contaminant transport in coastal aquifers: Numerical simulation of a reactive Henry problem, *Journal of Contaminant Hydrology* **145**: 90–104. https://doi.org/10.1016/j.jconhyd.2012.12.005.

NVIDIA (2014). *NVIDIA CUDA C: Programming Guide*, version 6.5 edn, NVIDIA Corporation.

NVIDIA (2018). *OpenACC Getting Started Guide*, NVIDIA Corporation, Hillsboro, United States of America.

OpenMP Architecture Review Board (2018). OpenMP Application Programming Interface. Disponível em https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf.

Peaceman, D. W. (1978). Interpretation of well-block pressures in numerical reservoir simulation, *Society of Petroleum Engineers Journal* **18**(3): 183–194. https://doi.org/10.2118/6893-PA.

Peaceman, D. W. (1983). Interpretation of well-block pressures in numerical reservoir simulation with nonsquare grid blocks and anisotropic permeability, *Society of Petroleum Engineers Journal* **23**(3): 531–543. https://doi.org/10.2118/10528-PA.

Perni, P. H. R. (2002). *Um Estudo sobre Métodos Iterativos na Solução de Sistemas de Equações Provenientes do Método dos Elementos de Contorno*, PhD thesis, Universidade Federal do Rio de Janeiro.

Porto da Silveira, J. F. (2000). O problema do caixeiro viajante. Disponível em http://www.mat.ufrgs.br/~portosil/caixeiro.html.

Redondo, C. (2017). *A fast IMPES multiphase flow solver in porous media for reservoir simulation*, PhD thesis, Universidad Politécnica de Madrid Escuela Técnica Superior de Ingenieros Aeronáuticos.

Reis da Silva, A., da Silva Menezes, M. and de Castro Alves, E. V. (2013). Estudo do método de sobre relaxação sucessiva na resolução de sistemas de equações lineares. Anais do Congresso de Matemática Aplicada e Computacional.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*, 2 edn, Society of Industrial and Applied Mathematics, USA.

Werneck, L. F. (2016). *Implementação paralelizada de métodos de resolução de sistemas algébricos na simulação de reservatórios de gás*, Dissertação de mestrado, Universidade do Estado do Rio de Janeiro, Nova Friburgo.

Werneck, L. F., Freitas, M. M., de Souza, G., Jatobá, L. F. C. and Amaral Souto, H. P. (2019). An OpenMP parallel implementation using a coprocessor for numerical simulation of oil reservoirs, *Computational & Applied Mathematics* **38**: 33. https://doi.org/10.1007/s40314-019-0788-6.

Yakubov, S., Cankurt, B., Abdel-Maksoud, M. and Rung, T. (2013). Hybrid MPI/OpenMP parallelization of an Euler–Lagrange approach to cavitation modelling, *Computers and Fluids* **80**: 365–371. https://doi.org/10.1016/j.compfluid.2012.01.020.