XLIII CILAMCE
21–25, November, 2022
FOZ DO IGUAÇU – PR – BRAZIL

Numerical solution of non-isothermal flow in heavy oil reservoirs using parallel computing

Ralph Alves Bini da Silva Almeida[1], Grazione de Souza[1], Helio Pedro Amaral Souto[1]

[1]*Rio de Janeiro State University, Polytechnic Institute*
25 Bonfim Street, Vila Amélia, Zip-code:28625570, Nova Friburgo, RJ, Brazil.
rabsalmeida@iprj.uerj.br, gsouza@iprj.uerj.br, helio@iprj.uerj.br

Abstract. In this work, we have used the OpenACC to parallelize a reservoir simulator aiming to simulate two-dimensional non-isothermal flows in a heavy oil reservoir. We have considered the production scenario with a vertical well and two static heaters. We have also applied the Control Volume Finite Difference Method to discretize flow and energy governing equations. We have chosen the Conjugate Gradient Method to solve the systems of algebraic equations to obtain pressure and temperature fields along with an operator splitting method. In the study of computational performance, we have employed different computational meshes and achieved speedup values greater than eight.

Keywords: Heavy oil. High-performance computing. OpenACC. Reservoir simulation. Thermal recovery.

1 Introduction

We can apply miscible, chemical, miscible displacement, biological or thermal methods in enhanced oil recovery. Specifically, when it comes to thermal techniques, we can cite approaches such as combustion in situ [1], heated steam injection [2], or reservoir heating through static equipment [3]. In the latter, considered in this work, we introduce an apparatus into the porous medium, and a heating process occurs without the need to inject fluids into the reservoir. Heating a reservoir favors flow by reducing the oil viscosity, as in the case of heavy oil reservoirs (viscosities ranging from 20 to 400 $\times$ 10^{-3} Pa.s) [4].

As a possible application, we can mention the case of single-phase flow in a heavy oil reservoir heated through static heating wells (Figure 1). As is well known, in this problem, the flow and the heat transfer are governed by nonlinear partial differential equations. Analytical solutions, when available, can only be determined for particular simplified cases. Therefore, we resort to numerical methods to obtain approximate solutions in the context of oil reservoir simulation.

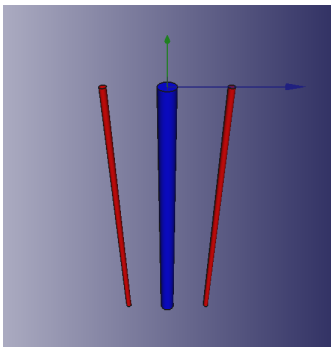

Figure 1. Producing well (in blue) and two static heating wells (in red)

We frequently use numerical simulations of flow in reservoirs to evaluate the operational scenarios, helping to choose the best production strategy. In this sense, the oil & gas industry has used high-performance computing tools to obtain results with a lower computational cost. Parallel computing consists, in general, of the use of

hardware and specific programming techniques that can enable the reduction of the time required for the execution of computer programs when compared to the respective execution time using serial codes [5]. Parallelization techniques include, for example, the use of programming applying Message Passing Interface (MPI), Open Multi-Processing (OpenMP), Open Accelerators (OpenACC), Compute Unified Device Architecture (CUDA), or hybrids of these. As the hardware of computers evolves, more complex engineering problems are likely to be solved, although the computational effort expended in solving these problems tends to be very high. Conversely, we can achieve improvements in computational performance by parallelizing the entire numerical code, or part of it, thus implying the use of architectures enabling distributed and parallel execution [6].

Within the context already presented, this work has as its goal the implementation of a computationally efficient simulator. For this purpose, we use the OpenACC to parallelize part of the numerical code. Specifically, we have decided to modify the numerical code of the method chosen to solve the systems of linearized algebraic equations and the calculation of transmissibilities. These systems arise from the discretization process, and we use them to obtain the pressure and temperature fields during heavy oil recovery in a reservoir heated by static elements.

2 Physical-mathematical Modeling

Introducing the Formation Volume Factor ($B = \rho_{osc}/\rho_o$) and using the Darcy law, we can write the equation that expresses the mass balance for the porous media oil flow as [7, 8]

$$\underbrace{V_b\left(\frac{\phi c_o}{B^0}+\frac{\phi^0 c_\phi}{B}\right)}_{\equiv\Gamma_p}\frac{\partial p}{\partial t}-\underbrace{V_b\left(\frac{\phi c_{oT}}{B^0}+\frac{\phi^0 c_{\phi T}}{B}\right)}_{\equiv\Gamma_T}\frac{\partial T}{\partial t}-V_b\nabla\cdot\left(\frac{\mathbf{k}}{B\mu}\nabla p\right)-q_{sc}=0, \tag{1}$$

where ρ_o is the oil density, ϕ is the porosity, μ_o is the oil viscosity, $\mathbf{k}$ is the absolute permeability tensor (here considered diagonal), p is the pressure, V_b is the bulk volume, q_{sc} is a volumetric source term, sc indicates the standard conditions for pressure and temperature (p_{sc} and T_{sc}), and the superscript "0" indicates the reference conditions. We have neglected the gravitational effect in obtaining eq. (1).

Here, we also consider that

$$B=B^0\left[1+c_o\left(p-p^0\right)-c_{oT}\left(T-T^0\right)\right]^{-1} \qquad \text{and} \qquad \phi=\phi^0\left[1+c_\phi\left(p-p^0\right)-c_{\phi T}\left(T-T^0\right)\right], \tag{2}$$

where B^0 and ϕ^0 are, respectively, the formation volume factor and the porosity in the reference conditions of pressure, p^0, and temperature, T^0. The terms c_o and c_{oT} represent the compressibility and the expansion thermal coefficient of the oil, while c_ϕ and $c_{\phi T}$ are the compressibility and the expansion thermal coefficient of the rock, respectively. Moreover, for the oil viscosity, we use the correlation for heavy oil:

$$\mu=a\exp\left(\frac{b}{T-T_{ref,\mu}}\right), \tag{3}$$

where a and b are oil dependent and $T_{ref,\mu}$ is a reference temperature [4].

As the initial condition, we set $p(x,y,t=0)=p_{ini}(x,y)=p_{inic}$, where p_{ini} is the initial pressure before production and heating start. Concerning boundary conditions, no-flow conditions are chosen, and thus $(\partial p/\partial x)_{x=0,L_x}=(\partial p/\partial y)_{y=0,L_y}=0$, where L_x and L_y are the reservoir lengths.

Besides, considering a well-reservoir coupling [9],

$$q_{sc}=-J_w\left(p-p_{wf}\right), \tag{4}$$

where J_w is the productivity index and p_{wf} is the wellbore pressure.

On the other hand, from the energy balance, without assuming the local thermal equilibrium hypothesis ($T_r \neq T_o$) [10],

$$\frac{\partial}{\partial t}\left[(\rho c_p)T\right] - \nabla\cdot(\boldsymbol{\kappa}\nabla T) = \frac{q_H}{V_b} + \frac{\rho_o h_o q_{sc}}{V_b} - \nabla\cdot(\rho_o h_o \mathbf{v}_o), \tag{5}$$

where $(\rho c_p) = \phi\rho_o c_{p_o} + (1-\phi)\rho_r c_{p_r}$ represents the porous media thermal capacity, T is the average temperature of the reservoir, such as $(\rho c_p)T = \phi\rho_o c_{p_o} T_o + (1-\phi)\rho_r c_{p_r} T_r$, T_r and T_o are rock and oil temperatures, h_o is the oil enthalpy, q_H is a heat source term, and $\boldsymbol{\kappa} = \left[\phi\kappa_o + (1-\phi)\,\kappa_r\right]\mathbf{I}$ is the effective thermal dispersion tensor (neglecting the tortuosity and hydrodynamic dispersion terms [10]). The source term q_H represents the energy transfer from the heating wells to the reservoir.

We propose the initial condition for temperature $T(x,y,t=0) = T_{ini}(x,y) = T_{inic}$, where T_{inic} is the initial temperature before the production and heating begin. As boundary conditions we use $(\partial T/\partial x)_{x=0,L_x} = (\partial T/\partial y)_{y=0,L_y} = 0$.

3 Numerical methodology

We use the Control Volume Finite Difference (CVFD) method and a centered block mesh to obtain the numerical solution of governing equations [11]. We determine it in the nodes of the computational mesh, located in the centers of the cells (blocks), where n_x and n_y are the number of cells in the x- and y-directions, respectively. The integer index i and j represent the cell numbers in the respective$x$- and y-directions, while fractional index $i \pm 1/2$ and $j \pm 1/2$ indicate the cell faces. For the two-dimensional flow, using a fully time-implicit formulation, it is possible to obtain the final discretized form of eq. (1) as [8]

$$\begin{aligned} &\mathbb{T}_x\Big|_{i+1/2,j}^{n+1}\left(p_{i+1,j}^{n+1} - p_{i,j}^{n+1}\right) - \mathbb{T}_x\Big|_{i-1/2,j}^{n+1}\left(p_{i,j}^{n+1} - p_{i-1,j}^{n+1}\right)\mathbb{T}_y\Big|_{i,j+1/2}^{n+1}\left(p_{i,j+1}^{n+1} - p_{i,j}^{n+1}\right) \\ -\ &\mathbb{T}_y\Big|_{i,j-1/2}^{n+1}\left(p_{i,j}^{n+1} - p_{i,j-1}^{n+1}\right) = \frac{(\Gamma_p)_{i,j}^{n+1}}{\Delta t}\left(p_{i,j}^{n+1} - p_{i,j}^{n}\right) + \frac{(\Gamma_T)_{i,j}^{n+1}}{\Delta t}\left(T_{i,j}^{n+1} - T_{i,j}^{n}\right) + (q_{sc})_{i,j}^{n+1} \end{aligned} \tag{6}$$

where we have used centered difference approximations and Euler approximations backward in time, $n+1$ is the time level in which the pressures are unknown, and we have also introduced the transmissibility in the x-direction given by

$$\mathbb{T}_{x,i\pm\frac{1}{2},j}^{n+1} = \left(\frac{A_x k_x}{\mu B \Delta x}\right)_{i\pm\frac{1}{2},j}^{n+1}, \tag{7}$$

knowing that we use a harmonic mean to determine the area and permeability in the position $i \pm 1/2, j$ from the known values in i, j and $i \pm 1, j$, while we employ an arithmetic mean for fluid properties [11]. We can use a similar procedure and obtain an equivalent expression for the transmissibility in the y-direction.

We can use the term $(q_{sc})_{i,j}^{n+1}$ in eq. (4) to include the wellbore pressure. In this case, we have

$$(q_{sc})_{i,j}^{n+1} = -\left(J_w\right)_{i,j}^{n+1}\left[p_{i,j}^{n+1} - (p_{wf})_{i,j}^{n+1}\right], \tag{8}$$

where the productivity index is given by

$$(J_w)_{i,j}^{n+1} = \left[2\pi\sqrt{k_x k_y}L_z/B\mu\,\ln\left(r_{eq}/r_w\right)\right]_{i,j}^{n+1}, \tag{9}$$

where r_w is the radius of the producing well and we calculate the equivalent radius, r_{eq}, using the equation [9]

$$r_{eq} = 0,28\left[\frac{\sqrt{\sqrt{(k_y/k_x)}\,(\Delta x)^2 + \sqrt{(k_x/k_y)}\,(\Delta y)^2}}{\sqrt[4]{(k_y/k_x)} + \sqrt[4]{(k_x/k_y)}}\right]_{i,j}. \tag{10}$$

From the energy balance (eq. (5)) we obtain its corresponding discrete form [8]

$$\mathbb{K}_x\Big|_{i+1/2,j}^{n+1}(T_{i+1,j}-T_{i,j})^{n+1}-\mathbb{K}_x\Big|_{i-1/2,j}^{n+1}(T_{i,j}-T_{i-1,j})^{n+1}\,\mathbb{K}_y\Big|_{i,j+1/2}^{n+1}(T_{i,j+1}-T_{i,j})^{n+1}$$
$$-\mathbb{K}_y\Big|_{i,j-1/2}^{n+1}(T_{i,j}-T_{i,j-1})^{n+1}=V_b\left[\frac{(\rho c_p)}{\Delta t}\right]_{i,j}^{n+1}\left(T_{i,j}^{n+1}-T_{i,j}^{n}\right)+(\rho_o h_o q_{sc})_{i,j}^{n+1}+\Phi_{i,j}^{n+1}+(q_H)_{i,j}^{n+1}\,, \quad (11)$$

where

$$\mathbb{K}_x\Big|_{i\pm1/2,j}^{n+1}=\left(\frac{A_x\kappa_x}{\Delta x}\right)_{i\pm1/2,j}^{n+1}\qquad\text{and}\qquad\mathbb{K}_y\Big|_{i,j\pm1/2}^{n+1}=\left(\frac{A_y\kappa_y}{\Delta y}\right)_{i,j\pm1/2}^{n+1}, \quad (12)$$

$$\Phi_{i,j}^{n+1}=\rho_{osc}[(h_o\mathbb{T}_x)_{i-(1/2),j}^{n+1}(p_{i,j}-p_{i-1,j})^{n+1}-(h_o\mathbb{T}_x)_{i+(1/2,j)}^{n+1}(p_{i+1,j}-p_{i,j})^{n+1}]$$
$$+\rho_{osc}[(h_o\mathbb{T}_y)_{i,j-(1/2}^{n+1}(p_{i,j-1}-p_{i,j})^{n+1}-(h_o\mathbb{T}_y)_{i,j+(1/2}^{n+1}(p_{i,j+1}-p_{i,j})^{n+1}]. \quad (13)$$

Equations (6) and (11) are linearized using Picard's method [12]. Here, the solution of the linearized algebraic systems demands the determination of hundreds of thousands of unknowns. Therefore, due to the size of these systems, it becomes necessary to use a large amount of computational memory and resources with high processing speed for their numerical resolution. In this work, we utilize the Conjugate Gradient method [13] for this purpose.

4 Parallel processing using OpenACC

In the field of parallel computing, one of the most popular resources available is the OpenACC. When applying it in shared memory architectures, it is necessary to use three components: the compilation directives, the execution library, and the environment variables [14]. In general, during the execution of the numerical code a directive will initiate the parallelization of a part of the computational code, distributing a sequence of tasks among several threads. Then the threads will perform the assigned tasks separately. OpenACC also allows for parallelization using graphics cards employing uncomplicated programming compared to CUDA. This last feature represents the main reason for adopting OpenACC, that is, the possibility of having the best performance of GPUs to run numerical codes due to a large number of cores available and dedicated memory.

Parallelization using OpenACC demands the correct installation of the compilers and the graphics card, besides a thorough analysis of the computational code to define which parts or functions of the code request more computational effort. Also, we must propose the directives, clauses, and flags for a successful compilation of the numerical code [15]. Here, we utilize the version of the community edition for Linux (based on Debian) of PGI Compiler 20.4 (NVIDIA HPC SDK Version 20.11). For readers who have already used the OpenMP, we should point out that the commands and directives are similar. Therefore, the user will experience a sense of familiarity.

After defining the code region to be parallelized, it is necessary to select among the different options, the directives, and clauses that are the most appropriate. In this context, we added the following directive: *#pragma acc parallel loop* to parallelize a loop. A fundamental directive is *data*. We employ it when we want to copy, for example, vectors and matrices that we stored in the memory accessed by the CPU to the memory of the GPU or vice versa. Regarding the *data* directive, we utilize respectively *copy* and *create* clauses to transfer data to the GPU and CPU memories.

To parallelize loops of type *for* it is possible to replace the directive *parallel loop* with *kernels*. This last directive allows the compiler to choose the safest parallelization strategies. Besides, it can verify whether parallelization is feasible. Therefore, it deprives the programmer of his prerogatives of choice. We have used clauses such as *private* and *reduction* associated with the *parallel loop* directive. The first specifies that each loop iteration has its copy of the variables listed. The reduction clause works similarly to the private clause, and the compiler generate a private copy of the variables. However, there is a reduction at the end of the parallel region execution of all private copies into a single final result. Possible reduction operations are addition, multiplication, maximum and minimum, for example, *reduction(+:sum)*.

Recent advances make it possible, depending on the OpenACC version, to use the GNU Compiler to compile parts of the computer codes. Nevertheless, the most widespread and consolidated use of OpenACC recommends

that we perform the compilation with the PGI Compiler [16]. For that, we must use compilation-specific flags so that the code executes tasks in parallel on CPUs or GPUs.

After modifying the computational code, written in C language, using OpenACC directives and clauses, we have to type one of the following commands in a Linux terminal: *pgcc -acc -ta=nvidia:managed -fast -Minfo=all -lm code.c -o code_parallel* or *pgcc -acc -ta=multicore -fast -Minfo=all -lm code.c -o code_parallel*. The "pgcc" command indicates that the PGI must compile the source code, while the "-acc" flag tells the compiler that it must consider the OpenACC directives and clauses in the code. Besides, the flag *-ta=nvidia:managed* indicates that the graphics card will execute the code. Now, if we utilize *-ta:multicore*, the CPUs' threads must perform the execution, as when we use OpenMP. Also, the allocation in memory using *Malloc* and *Calloc* will be replaced by routines that allocate memory in a unified way, such as, for example, the *cudaMallocManaged*. Finally, the flag *-fast* has the function of optimizing the code, and *-Minfo=all* displays as much information as possible on the terminal [15].

5 Numerical Results

In the context of tertiary recovery of hydrocarbons, we have chosen to study the non-isothermal two-dimensional flow of heavy oil. Further, we enhance the production using static heaters. The properties and parameters used can be seen in Table 1, which represents the default case adopted. So these values are used in all simulations unless otherwise specified. All the simulation runs have been accomplished in an Intel(R) Xeon(R) Silver 4210 CPU @ 2.20 GHz with 48 threads. Although it is possible to run using graphics cards, we chose only to use the CPUs threads to perform the simulations in this initial work.

Table 1. Default case

Parameter	Value	Parameter	Value	Parameter	Value
a	0.2 Pa · s	$k_x = k_y$	0.02 μm^2	x_1	2,725 m
b	600 K	$L_x = L_y$	6,400 m	x_2	3,675 m
B^0	1.3 m^3/(std m^3)	L_z	40 m	$y_1 = y_2$	3,200 m
c_o	7.25 $\times$ 10 $^{-7}$ kPa^{-1}	n_H	2	κ_o	0.45 W/(m · K)
c_{oT}	7.25 $\times$ 10 $^{-7}$ K^{-1}	$p_{inic} = p^0$	6,900 kPa	κ_r	3.5 W/(m·K)
c_{po}	2,100 J/(kg · k)	q_H	20 kW	ρ	2,500 kg/m^3
$c_{p\phi}$	1,200 J/(kg· K)	t_{max}	60 days	ϕ^0	0.2
c_{vo}	1,800 J/(kg·k)	$tol_1 = tol_2$	1.0 $\times$ 10 $^{-6}$ kPa or K	Δt_{ini}	0.1 day
c_ϕ	4.35 $\times$ 10 $^{-7}$ kPa^{-1}	$T_{inic} = T^0$	330 K	Δt_{max}	1.0 day
$c_{\phi T}$	4.35 $\times$ 10 $^{-7}$ K^{-1}	T_{ref}	500 K		
$F_{\Delta t}$	1.1	$x_{prod} = y_{prod}$	3,200 m		

We must also provide some additional information concerning the data presented. In Table 1, (x_1,y_1) and (x_2,y_2) are the coordinates indicating where we have placed the two heaters, and we represent the number of heaters by n_H. Furthermore, we have positioned the producing well at the coordinates (x_{prod},y_{prod}). The maximum production time is t_{max}, the initial time increment is Δt_{ini}, and the time step growth rate is $F_{\Delta t}$, such that $\Delta t^{n+1} = F_{\Delta t}\Delta t^n$. Its growth occurs until the value reaches Δt_{max}, then it remains unchanged until the end of the simulation.

We have considered that the reservoir is a parallelepiped with dimensions L_x, L_y, and L_z, the last being much smaller than the others. We present in Table 2 for this domain geometry the number of cells used in the meshes in x- and y-directions (n_x and n_y), as well as the total number of corresponding cells.

We should point out that we have performed a mesh refinement study and found that the numerical method is convergent [8]. Furthermore, we haven't found any difference between the results obtained with the serial and parallelized versions of the numerical code [8]. Thus, it was evident that the results obtained in the simulations performed with parallel versions were in no way compromised. Furthermore, we have attested its superposition with those obtained from the serial version.

Table 2. Computational grids

Mesh	1	2	3	4	5	6
$n_x = n_y$	47	93	185	369	737	1473
Total of cells	2,209	8,649	34,225	136,161	543,169	2,169,729

Having defined the six computational meshes, we performed all the simulations varying the number of cells and threads to determine the respective speedup values, and we can see them in Table 3.

Table 3. Speedup and runtimes for the different meshes and number of threads

Threads	Mesh 1	Mesh 2	Mesh 3	Mesh 4	Mesh 5	Mesh 6
Serial	0.631	2.229	11.986	74.601	494.116	3495.587
5	0.427	0.974	3.837	19.39	96.471	581.953
10	0.541	1.028	3.387	15.645	72.101	439.387
15	0.682	1.153	3.255	14.265	67.148	418.519
20	0.797	1.420	3.412	13.750	63.669	406.573
25	0.779	1.535	3.622	14.375	66.385	481.769
30	0.811	1.594	4.128	14.886	64.801	401.054
35	0.807	1.658	4.155	14.636	62.704	411.262
40	0.893	1.760	4.492	15.043	64.556	418.268
Maximum speedup	1.478	2.289	3.682	5.426	7.880	8.716

From the observation of the results, it is clear that we have achieved our objective. The parallelization of the Conjugate Gradient method and the transmissibilities calculation allowed us to obtain a gain of around 870% with Mesh 6. We can also highlight that the reduction in execution time reached values of approximately 550 and 790% for Meshes 4 and 5. Therefore, considering we have used only CPUs, we understand that these initial results are satisfactory.

6 Conclusions

First of all, we would like to emphasize that the parallelization did not affect the accuracy of the results. We have verified this fact by comparing them with those obtained with the non-parallelized version of the numerical code.

As a consequence of modifications we have carried out in the numerical simulator, it was possible to obtain gains in computational performance. The speedup has increased when we have refined the computational meshes. As already advanced, there was a tendency to raise the speedup as the total number of cells in the computational meshes augmented. As a result, the simulations using Mesh 6 presented the highest speedup. However, we understand that, depending on the hardware, we cannot affirm that this behavior for even more refined meshes is maintained if we increase the number of cells and threads.

In future perspectives, we intend to carry out simulations using graphics cards and not only CPUs. By the way, this is one of the advantages of using OpenACC, which allows running the codes on both CPUs and GPUs. Of course, depending on the hardware specifications, we expect the gains in computational efficiency to be higher with the use of graphics cards.

Acknowledgements. The authors gratefully thanks Rio de Janeiro State University, Carlos Chagas Filho Foundation for Research Support of the State of Rio de Janeiro (FAPERJ), and Coordination for the Improvement of Higher Education Personnel (CAPES) - Finance Code 001 for their support.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] Y. Chen, W. Pu, X. Liu, Y. Li, M. A. Varfolomeev, and J. Hui. A preliminary feasibility analysis of in situ combustion in a deep fractured-cave carbonate heavy oil reservoir. *Journal of Petroleum Science and Engineering*, vol. 174, pp. 446 – 455, 2019.

[2] K. Mohammadi and F. Ameli. Toward mechanistic understanding of fast sagd process in naturally fractured heavy oil reservoirs: Application of response surface methodology and genetic algorithm. *Fuel*, vol. 253, pp. 840 – 856, 2019.

[3] E. Rangel-German, J. Schembre, C. Sandberg, and A. Kovscek. Electrical-heating-assisted recovery for heavy oil. *Journal of Petroleum Science and Engineering*, vol. 45, n. 3, pp. 213 – 231, 2004.

[4] M. Rousset. *Reduced-order modelling for thermal simulation*. PhD thesis, Stanford University, 2010.

[5] D. V. Leshchinskiy, A. V. Starchenko, E. A. Danilkin, and S. A. Prohanov. Parallel implementation of a numerical method for solving a three-dimensional transport equation for a mesoscale meteorological model. *Procedia Computer Science*, vol. 178, pp. 47 – 54. 9th International Young Scientists Conference in Computational Science, YSC2020, 05-12 September 2020, 2020.

[6] K. Jaquie. Extensão da ferramenta de apoio à programação paralela (f.a.p.p.) para ambientes paralelos virtuais. Master's thesis, Universidade de São Paulo, São Carlos, 1999.

[7] J. Dyrdahl. Thermal flow in fractured porous media and operator splitting., 2014.

[8] R. A. B. S. Almeida. Numerical solution of non-isothermal flow in heavy oil reservoirs applying parallel computing. Master's thesis, Universidade do Estado do Rio de Janeiro, Nova Friburgo. In portuguese, 2021.

[9] D. W. Peaceman. Interpretation of well-block pressures in numerical reservoir simulation with nonsquare grid blocks and anisotropic permeability. *Society of Petroleum Engineers Journal*, vol. 23, n. 3, pp. 531–543, 1983.

[10] C. Moyne, S. Didierjean, H. Amaral Souto, and da O. Silveira. Thermal dispersion in porous media: one-equation model. *International Journal of Heat and Mass Transfer*, vol. 43, n. 20, pp. 3853–3867, 2000.

[11] T. Ertekin, J. Abou-Kassem, and G. King. *Basic Applied Reservoir Simulation.* Richardson, USA: Society of Petroleum Engineers, 2001.

[12] H. M. Nick, A. Raoof, F. Centler, M. Thullner, and P. Regnier. Reactive dispersive contaminant transport in coastal aquifers: Numerical simulation of a reactive henry problem. *Journal of Contaminant Hydrology*, vol. 145, pp. 90–104, 2013.

[13] Y. Saad. *Iterative Methods for Sparse Linear Systems. 2. ed.* Philadelphia: SIAM, 2003.

[14] M. Sulzbach. Programação paralela híbrida para CPU e GPU: Uma avaliação do OpenACC frente a OpenMP e CUDA. Master's thesis, Universidade Federal de Santa Maria, Santa Maria, 2014.

[15] K. C. A. Moreira. Anotação automática de código com diretivas OpenACC. Master's thesis, Universidade Federal de Minas Gerais, Belo Horizonte, 2015.

[16] J. Y. Kim, J.-S. Kang, and M. Joh. GPU acceleration of MPAS microphysics WSM6 using OpenACC directives: Performance and verification. *Computers & Geosciences*, vol. 146, pp. 104627, 2021.