14th World Congress on Computational Mechanics (WCCM)
ECCOMAS Congress 2020)
Virtual Congress: 11-–15 January 2021
F. Chinesta, R. Abgrall, O. Allix and M. Kaliske (Eds)

PARALLEL NUMERICAL SOLUTION OF TWO-PHASE FLOW IN POROUS MEDIA ON NON-ORTHOGONAL GEOMETRIES: A PERFORMANCE STUDY USING DIFFERENT GPU ARCHITECTURES

VICTOR L. TEJA JUAREZ[1], LUIS M. DE LA CRUZ SALAS[2] AND BRUNO A. LOPEZ JIMENEZ[3]

[1] and [3] Engineering Faculty, Universidad Nacional Autonoma de Mexico
Ciudad Universitaria, CdMx, 04510
vleotejaj@unam.mx, balopezj@unam.mx

[2] Geophysics Institute, Universidad Nacional Autonoma de Mexico
Ciudad Universitaria, CdMx, 04510
luiggi@geofisica.unam.mx

Key words: Two-phase fluid flow, Newton-Raphson, GPU architectures, Performance

Abstract. A parallel numerical model for two phase flow (water and oil) in porous media on non-orthogonal geometries is solved by using different Graphics Processing Unit (GPU) architectures to carry out a comparison of the performance that can be reached by each of them. The mathematical model is based on the mass conservation transformed equations for water and oil phases, which results in two coupled non-linear partial differential equations (PDEs). The Finite Volume Method (FVM) is used to discretize the set of PDEs that govern this problem and the Newton-Raphson method is utilized to linearize and solve them simultaneously. Solution of the linear equations system is computationally expensive and requires a large amount of time as the number of unknowns increases. We take advantage of the current GPUs computing technology for constructing massive parallel numerical algorithms for modeling multi-phase flow in porous media [1, 2]. The construction of the Jacobian is directly done in the GPU, which reduces the information that needs to be exchanged between the CPU (Central Processing Unit) and the GPU. Libraries that include Krylov methods are used and tested. The numerical results indicate until 12x of speed up over a single CPU by applying the GPU parallelism with the different architectures tested in this study (Kepler, Pascal and Turing). Furthermore, this study also tries to identify which of these architectures is the best option according to our computing needs.

1 INTRODUCTION

Nowadays, oil remains as the main source of energy of the world. New production techniques are being applied to avoid declining of production in the oil fields. Most popular and oldest technique is the water injection, which consists in injecting water into a oil reservoir to displace oil and then produce it. Two phase fluid flow in porous media is a well known problem, mathematical and numerical model. Although it is a well known problem, almost all models found in the literature are based in finite differences method applied on an orthogonal Cartesian coordinate system with uniform or refined meshing in some places [3, 4, 5]. This limits solving complex domains with these type of models. On the other hand, models

based on the finite element method with non-orthogonal meshes create sparse matrices that are very complicated to solve, requiring highly complex solution algorithms, both for ordering of elements and for solving linear system equations [4]. Additionally, since hydrocarbons continue to be the main source of energy, faster calculations using from tens of thousands to millions of grid blocks are required to meet the research needs in the oil industry. Most of the research to speed up compute time is based on domain decomposition algorithms, which break the problem down into smaller problems that are solved across multiple processors using distributed memory [6, 7, 8]. However, there is a limitation to use this technique, since a computational cluster with tens until thousands of processors is needed to achieve the desired speed up. Shared memory algorithms are increasingly being incorporated for personal computers and workstations, which take advantage of the computing power of the multiple cores that a modern processor and GPUs could have, using this type of implementation it is not required to have access to a computing cluster to obtain good accelerations of numerical codes [9, 10, 11, 12].

In this work we develop a numerical simulator for water injection based on the mass conservation and fractional flow transformed phase equations and fully implicit time scheme discretization, this let us to solve partial differential equations on non-orthogonal geometries. A shared memory algorithm to use GPUs for both the construction of the Jacobian matrix and the solution of the linear system of equations is implemented.

2 MATHEMATICAL MODEL OF WATER INJECTION

An axiomatic formulation is used to derive the governing equations for water injection, this technique was introduced by Herrera and Pinder [13]. Equation (1) shows the mass balance into a porous media for a number phases equal to α.

$$\frac{\partial \phi \rho_\alpha S_\alpha}{\partial t} - \nabla \cdot \left(\bar{\bar{k}} \lambda_\alpha \left(\nabla p_\alpha - \rho_\alpha g \nabla z \right) \right) = q_\alpha \tag{1}$$

Considering two phases: the subscripts w and o for water and oil, respectively, this leads to:

$$\frac{\partial \phi \rho_w S_w}{\partial t} = \nabla \cdot \left(\frac{\bar{\bar{k}} k_{rw} \rho_w}{\mu_w} \nabla \Phi_w \right) + q_w \tag{2}$$

$$\frac{\partial \phi \rho_o S_o}{\partial t} = \nabla \cdot \left(\frac{\bar{\bar{k}} k_{ro} \rho_o}{\mu_o} \nabla \Phi_o \right) + q_o \tag{3}$$

where $\bar{\bar{k}}$ is the diagonal tensor of absolute permeability and $k_{r\alpha}$ is the relative permeability; the Greek letters Φ_α, ρ_α, μ_α are the potential, the density and the dynamic viscosity for the phase α ($\alpha = o, w$), respectively; the symbol ϕ represents the porosity of the porous media; the saturation and the source term for the phase α are represented by S_α and q_α, respectively. The mass balance equations are interrelated by the following mathematical expressions:

$$1 = S_o + S_w \tag{4}$$

$$\Phi_\alpha = p_\alpha - \rho_w g z \tag{5}$$

$$p_w = p_o - p_{cow} \tag{6}$$

$$p_{cow} = p_{cow}(S_w) \tag{7}$$

where p_α is the pressure of the phase α, g is the magnitude of gravity, z is the depth, p_{cow} is the capillary pressure for the oil-water as a function of S_w.

We apply fractional flow formulation to slightly decouple and make numerical solution of the equations (3), (2) more stable . Also we have considered incompressible fluids and porous media, then we can obtain the following equations [4]:

$$\phi\frac{\partial S_w}{\partial t}-\nabla\cdot\left[\bar{\bar{k}}\lambda_w\left(\nabla p_o-\frac{dp_c}{dS_w}\nabla S_w-\rho_w g\nabla z\right)\right]=\frac{q_w}{\rho_w}. \tag{8}$$

$$-\nabla\cdot\left[\bar{\bar{k}}\lambda\nabla p_o-\bar{\bar{k}}\lambda_w\frac{dp_c}{dS_w}\nabla S_w-\bar{\bar{k}}\left(\lambda_w\rho_w+\lambda_o\rho_o\right)g\nabla z\right]=\frac{q_w}{\rho_w}+\frac{q_o}{\rho_o}. \tag{9}$$

where new terms that appear are total mobility λ and water mobility λ_w. Equations (8) and (9) are known as water saturation a oil pressure equations, respectively.

2.1 Mathematical model in the computational domain

In order to solve the equations (8) and (9), we have transformed them from a physical domain to a computational domain as shown in figure 1 by using following mathematical expressions $\xi=\xi(x,y)$, $\eta=\eta(x,y)$, and $\tau=t$. The details of this transformation are described in reference [14].

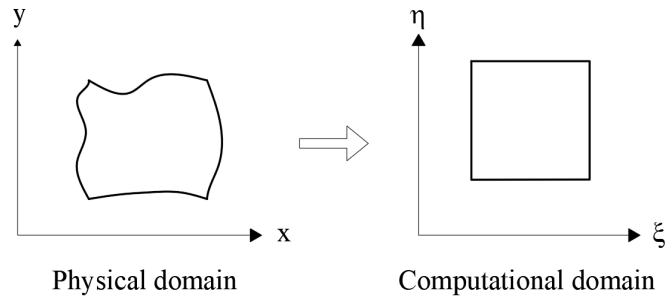


Figure 1: Mapping from irregular physical domain to a regular computational domain.

The final transformed equations are:

$$\begin{aligned}\phi\frac{\partial}{\partial\tau}\left(\frac{S_w}{J}\right) &= \frac{\partial}{\partial\xi}\left[A_{11}^{\alpha}\frac{\partial p_o}{\partial\xi}+A_{12}^{\alpha}\frac{\partial p_o}{\partial\eta}-B_{11}^{\alpha}\frac{\partial S_w}{\partial\xi}-B_{12}^{\alpha}\frac{\partial S_w}{\partial\eta}\right] \\ &+ \frac{\partial}{\partial\eta}\left[A_{21}^{\alpha}\frac{\partial p_o}{\partial\xi}+A_{22}^{\alpha}\frac{\partial p_o}{\partial\eta}-B_{21}^{\alpha}\frac{\partial S_w}{\partial\xi}-B_{22}^{\alpha}\frac{\partial S_w}{\partial\eta}\right]+\frac{S^{wq}}{J},\end{aligned} \tag{10}$$

$$\begin{aligned}&\frac{\partial}{\partial\xi}\left[A_{11}^{\alpha}\frac{\partial p_o}{\partial\xi}+A_{12}^{\alpha}\frac{\partial p_o}{\partial\eta}-B_{11}^{\alpha}\frac{\partial S_w}{\partial\xi}-B_{12}^{\alpha}\frac{\partial S_w}{\partial\eta}\right] \\ &+ \frac{\partial}{\partial\eta}\left[A_{21}^{\alpha}\frac{\partial p_o}{\partial\xi}+A_{22}^{\alpha}\frac{\partial p_o}{\partial\eta}-B_{21}^{\alpha}\frac{\partial S_w}{\partial\xi}-B_{22}^{\alpha}\frac{\partial S_w}{\partial\eta}\right]+\frac{S^{oq}}{J}=0.\end{aligned} \tag{11}$$

where S^{wq} and S^{oq} are the source terms, tensors A_{ij} and B_{ij} equation are defined as:

$$\begin{aligned}
A_{11}^{\alpha} &= \tfrac{1}{J}\left(a_{11}^{\alpha}\xi_x^2 + a_{22}^{\alpha}\xi_y^2\right), & B_{11}^{\alpha} &= \tfrac{1}{J}\left(b_{11}^{\alpha}\xi_x^2 + b_{22}^{\alpha}\xi_y^2\right), \\
A_{12}^{\alpha} &= \tfrac{1}{J}\left(a_{11}^{\alpha}\xi_x\eta_x + a_{22}^{\alpha}\xi_y\eta_y\right), & B_{12}^{\alpha} &= \tfrac{1}{J}\left(b_{11}^{\alpha}\xi_x\eta_x + b_{22}^{\alpha}\xi_y\eta_y\right), \\
A_{21}^{\alpha} &= \tfrac{1}{J}\left(a_{11}^{\alpha}\xi_x\eta_x + a_{22}^{\alpha}\xi_y\eta_y\right), & B_{21}^{\alpha} &= \tfrac{1}{J}\left(b_{11}^{\alpha}\xi_x\eta_x + b_{22}^{\alpha}\xi_y\eta_y\right), \\
A_{22}^{\alpha} &= \tfrac{1}{J}\left(a_{11}^{\alpha}\eta_x^2 + a_{22}^{\alpha}\eta_y^2\right), & B_{22}^{\alpha} &= \tfrac{1}{J}\left(b_{11}^{\alpha}\eta_x^2 + b_{22}^{\alpha}\eta_y^2\right),
\end{aligned} \tag{12}$$

where J and Greek letters ξ_x, ξ_y, η_x and η_y represent the Jacobian and the transformation metrics, respectively. The remained terms a_{ii} and b_{ii} contain the rock and fluids properties, these terms are expressed as $a_{ii}^{\alpha} = k_{ii}\lambda_{\alpha}$ and $b_{ii} = k_{ii}\lambda_{\alpha}\frac{dp_c}{dS_w}$.

3 NUMERICAL MODEL

Most of literature concerning the numerical implementation of two phase fluid flow model are based on Finite Differences (FD), Finite Volumes (FV) and Finite Elements (FE) methods are the general framework for numerical simulation [4, 5]. For solving the mathematical model of water injection numerically, we have to choose a strategy to minimize numerical oscillations and to reproduce the physics of the problem. We have selected Finite Volume Method to discretize and Newton-Raphson Iteration to linearize and solve equations (10) and (11).

3.1 Finite Volume Method

In order to discretize equations (10) and (11) by applying FV, first we have to integrate the equations with respect to the time and the control volume shown in figure 2, for both equations we obtain:

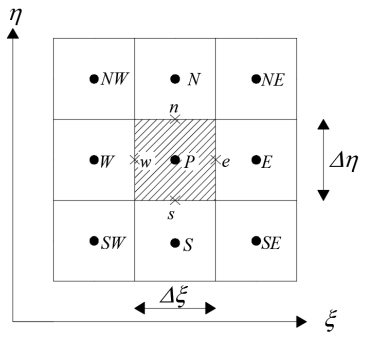


Figure 2: Elementary control volume in the computational domain.

$$\begin{aligned}
\int_n^{n+1}\int_{dA}\Bigg\{\phi\frac{S_w}{J}\left(\frac{Q}{J}\right) \;-\; & \frac{\partial}{\partial\xi}\left[A_{11}^{\alpha}\frac{\partial p_o}{\partial\xi} + A_{12}^{\alpha}\frac{\partial p_o}{\partial\eta} - B_{11}^{\alpha}\frac{\partial S_w}{\partial\xi} - B_{12}^{\alpha}\frac{\partial S_w}{\partial\eta}\right] \\
+\; & \frac{\partial}{\partial\eta}\left[A_{21}^{\alpha}\frac{\partial p_o}{\partial\xi} + A_{22}^{\alpha}\frac{\partial p_o}{\partial\eta} - B_{21}^{\alpha}\frac{\partial S_w}{\partial\xi} - B_{22}^{\alpha}\frac{\partial S_w}{\partial\eta}\right] - \frac{S^{wq}}{J}\Bigg\}d\xi d\eta d\tau = 0
\end{aligned} \tag{13}$$

$$\int_n^{n+1}\int_{dA}\left\{\begin{array}{rl} & \frac{\partial}{\partial\xi}\left[A_{11}^{\alpha}\frac{\partial p_o}{\partial\xi}+A_{12}^{\alpha}\frac{\partial p_o}{\partial\eta}-B_{11}^{\alpha}\frac{\partial S_w}{\partial\xi}-B_{12}^{\alpha}\frac{\partial S_w}{\partial\eta}\right] \\ + & \frac{\partial}{\partial\eta}\left[A_{21}^{\alpha}\frac{\partial p_o}{\partial\xi}+A_{22}^{\alpha}\frac{\partial p_o}{\partial\eta}-B_{21}^{\alpha}\frac{\partial S_w}{\partial\xi}-B_{22}^{\alpha}\frac{\partial S_w}{\partial\eta}\right]+\frac{S^{oq}}{J}\end{array}\right\}d\xi d\eta d\tau=0 \tag{14}$$

to evaluate remained terms of the integrands, we take in to account the following considerations: 1) a backward Euler approximation with a fully implicit scheme are used, 2) the permeability tensor is diagonal and 3) the space derivatives are approximated by using finite differences. After applying mentioned considerations, finally we obtain the residuals for both equations:

$$R_w = R_w(S_w, p_o) \tag{15}$$

$$R_o = R_o(S_w, p_o) \tag{16}$$

3.2 Newton Method

For applying the Newton-Raphson method we select p_o and S_w as primary variables. Thus, the system of equations to solve has the following form:

$$\begin{pmatrix}\frac{\partial R_o^k}{\partial p_o} & \frac{\partial R_o^k}{\partial S_w}\\ \frac{\partial R_w^k}{\partial p_o} & \frac{\partial R_w^k}{\partial S_w}\end{pmatrix}\begin{pmatrix}\delta p_o^{k+1}\\ \delta S_w^{k+1}\end{pmatrix}=\begin{pmatrix}-R_o^k\\ -R_w^k\end{pmatrix}, \tag{17}$$

Matrix on the left of equation (17) is the Jacobian and superscript k is used to indicate the Newtonian iteration. It should be noted Jacobian has four blocks of nine diagonal each one, due to the nature of the discretization method proposed (see figure 3). This facilitates the ordering of the Jacobian matrix compared to a finite element discretization.

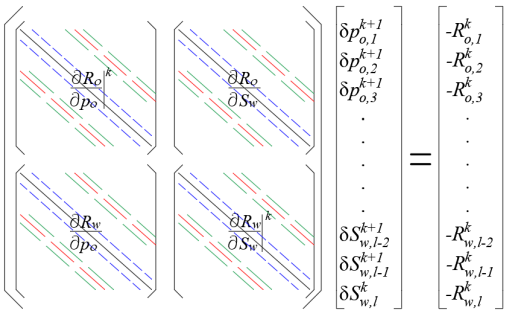


Figure 3: Schematic representation of linear system equations.

4 COMPUTATIONAL IMPLEMENTATION

As first step, we implemented the algorithm to be executed in an ordinary CPU to ensure proper working and for comparison purposes. We write our codes using the C++ and Compute Unified Device Architecture (CUDA) languages. Pseudocode of the main algorithm is shown in figure 4, which was used for

```
01 /* Initialization of all variables and initial data,
02 construction of all arrays to store field variables
03 and matrices and vectors of linear system */
04  WHILE (Time_step <= Total_time) DO
05       WHILE ( Delta_Sw < epsilon & Iteration < Max_iter) DO
06           Calculate oil components of Jacobian and Ro
07           Calculate water components of Jacobian and Rw
08           Build Jacobian matrix using the CRS format
09           Solve the linear system of equations using BICGSTAB
10       END WHILE
11    Update old variables
12   IF (!(Time_step % Frequency)) THEN
13      Save or print primary variables (po and Sw)
14   END IF
15  END WHILE
16 /* Free memory and finalize simulation */
```

Figure 4: General pseudo-code.

both CPU and GPU implementation. First three lines are similar to any standard numerical code to solve partial diferential equations. In the fourth line begins the main numerical solution loop. First "while" sentence is used to control time loop and second one is for solving the equations through Newtonian iteration. Lines from 11 to 14 allow us to update and save primary variables. Finally, lines 15 and 16 are used to go out of main loop and to free memory, respectively. The CPU version code uses the Eigen library was used to solve linear system equations [15].

For the GPU implementation we use CUDA and the CUSP Library, which provides a high-level interface for manipulating sparse matrices and solving sparse linear systems [16]. It should be emphasized that in this case two kind of memories are used: global memory to store the primary variables (S_w and p_o), the properties of the porous media (k, ϕ , etc) and some other important arrays of the simulation; constant memory is used to store constant values, i.e, the viscosities, conversion factors and some tables of properties. Keeping this in mind, we have paralysed the calculation of Jacobian, as an example of this method, we show an extract of CUDA-kernel for computing the Jacobian block corresponding to the residual $\partial R_o/\partial p_o$ and its derivatives [12]. Before used any CUDA-kernel, all necessary data have to be copied to the global memory of the GPU for its correct execution. Analyzing the figure 5, first thing to do is to determine the thread index, see lines 2 and 3. Using this index it is possible that each thread of the block in the grid, execute the operations defined in lines 5-11 concurrently. Line 4 is required to assure that the index is inside of the limits of the arrays. Lines 5 to 7 consist of device kernel functions, which can only be called by the GPU. Finally, lines 9-11 are calculations that run in parallel within the kernel.

5 NUMERICAL RESULTS

As a way to validate our numerical code, in this section we present a comparison with a well known problem called Five Spot pattern. Also numerical experiments with three different GPU architectures were carried out to compare the performance between them.

```
01 __global__ void jacobianCoeff_RoPo2D (*physical parameters as arguments) {
02 int i = threadIdx.x + blockIdx.x*blockDim.x;
03 int j = threadIdx.y + blockIdx.y*blockDim.y;
04 if(i>0 && i < nx-1 && j > 0 && j < ny-1){
05    interpolatePermeability(k, k_e, k_w, k_n, k_s);
06    interpolateSaturation(Sw, Sw_e, Sw_w, Sw_n, Sw_s);
07    calculate_kro(Sw, kro(Sw),kro_e, kro_w, kro_n, kro_s);
08    /* ... Some other calculations ... */
09    R(Po)=Tr_e*(Po[i+1,j])+Tr_w*(Po[i-1,j])+...;
10    R(Po+deltaPo)=Tr_e*(Po[i+1,j]+deltaPo)+Tr_w*(Po[i-1,j]+deltaPo)+...;
11    Block_RoPo2D[2D(i,j)]=firstOrderDerivative(R(Po), R(Po+deltaPo), deltaPo);
12  }
```

Figure 5: Kernel function to compute a Jacobian block $\partial R_o/\partial p_o$ by using the GPU.

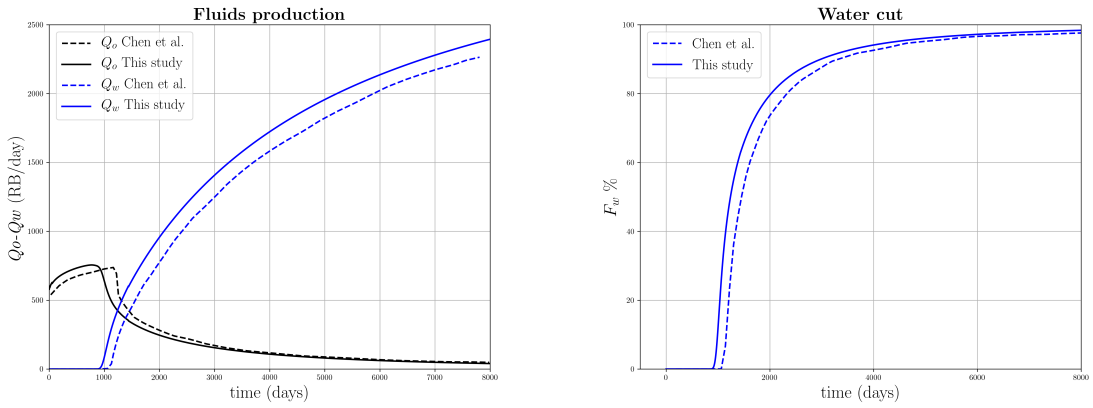
5.1 Numerical model validation

The Five Spot pattern describes the displacement of oil by water in an isotropic domain, in which four producer wells are located at the corners of a rectangular domain and one injector well is at the center of the domain. We consider only one quarter of the domain due to symmetry. In this model we consider the follow assumptions: displacement occurs at a bidimensional orthogonal domain, porous media is isotropic, effects of capillary pressure are taking into account, there are one source (injector well) and one sink (producer well) and gravity forces are neglected.

The petrophysical parameters to carry out the simulation can be obtained from Chen et al. [4]. The numerical parameters were selected as follows: to evaluate the derivatives within the Jacobian blocks we chose increments of $\Delta S_w = 1 \times 10^{-4}$ for water saturation and $\Delta p_o = 0.1$ for the oil pressure were used and a fixed time step of 1 day was chosen. Figure 6 a) shows the production of water and oil in reservoir barrels units (RB) and figure 6 b) shows the fractional flow of the water to analyze when water cut occurs and the percentage of this in the production fluids. It can be seen that results reported by Chen and those obtained in this work have a similar qualitatively behavior. The small difference observed in figures 6, may be due to implementations of the numerical method, since Chen used an improved IMPES with 5 neighboring points for the discretization, while in this work a fully implicit scheme with the Newton Raphson method and 9 neighboring points are used.

5.2 Results for non-orthogonal geometries

We ran our simulator in two different geometries with the same parameters data used at the five spot pattern. The only parameter that changed to carry out these simulations is the shape of the domain and consequently its mesh. We have made these changes to show our simulator can be used in non-orthogonal geometries. As a first step the numerical code was tested by using a trapezoidal domain (see figure 7 a)). We have selected this domain to analyze the behavior of the simulator in the quasi-orthogonal domains. Water saturation profile S_w for 500 days simulation time is shown in the figure 7 b). It can be seen that water saturation increases after five hundred days of continuous injection, that means oil in this part of domain already been swept. The graphs obtained for both fluids production and the water phase fractional flow are shown in figures 7 c) and 7 d), respectively. Water cut occurs around 1500 days, that is when the oil production begins to decline.



(a) (b)

Figure 6: Results for: a) water and oil production $Q_w - Q_o$ in reservoir barrels (RB); b) fractional flow $F_w(\%)$.

We tested the simulator in another non-orthogonal domain that is shown in figure 8 a). This domain has a more complex geometry compared with the first one. As in the example above, figure 8 b) shows water saturation profile S_w for 200 days simulation time, it can be seen saturation profile behavior is as expected. Also, fluids production and the water phase fractional flow are presented in figures 8 c) and 8 d). In this example, water cut takes place around 400 days, almost 1,000 days earlier than in previous case. Therefore, through these test examples it can be argued that our code is capable of dealing with non-orthogonal domains and reproducing the physical behavior of the water injection model in porous media adequately.

5.3 Performance tests

Numerical experiments were carried out on four different GPUs. First GPU is the Tesla K40m that is a professional graphics card based on Kepler architecture for clusters and workstations launched in 2013. Second GPU is the Quadro P400, this is a low end professional card based on Pascal architecture focus on graphics performance for professional applications launched in February 2017. Third GPU is the RTX 2060 is a performance gaming graphics card based on Turing architecture which incorporates ray tracing technology. The last one GPU is the RTX 2070 with Max-Q design is the power saving variant of the mobile RTX 2070 and it can be founded in gaming laptops, last two mentioned GPUs were launched in 2019. Likewise, the serial code for a single CPU was also executed in a workstation with a single processor Intel Xeon Silver 4210 and 16 Gigabytes of RAM with the aim of having a starting point in the acceleration achieved.

In order to analyze the performance of the numerical code, we selected the trapezoidal domain since it is simpler than second domain to increase mesh size. The numerical parameters were the same as in the cases presented previously, only total time simulation was changed to 5 days and a value of 0.1 day time step was chose. Table 1 shows the average computation time for each Newton-Raphson iteration. As we can see Jacobian computation time increases when mesh size increases. In these results Xeon CPU is slower than any GPU show in the table. There are even several magnitude orders of difference

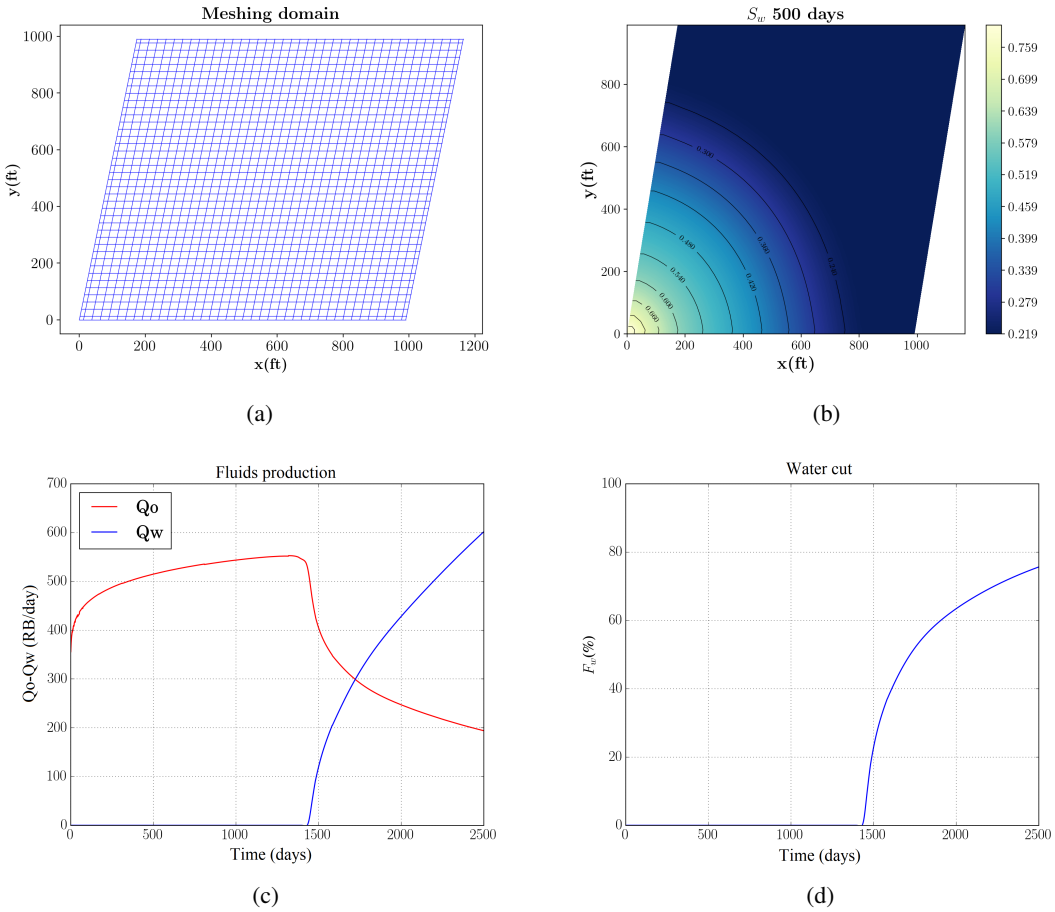


Figure 7: Results obtained for the first non orthogonal domain.

between CPU versus GPU computation times, for example, taking the Tesla K40m GPU as a reference and considering the smallest mesh size, speed up of 184x are obtained and if the larger mesh size is considered speed up increases to 18,653x. Note that only a single core of Xeon processor is being used. This result was obtained because all mathematical operations to calculate the Jacobian are independent of each other, and it is a considerable save of computation time taking into account this procedure has to be repeated every Newton iteration.

On the other hand, if the computation times obtained for the GPUs are compared, we realize they remain in the same magnitude order, with the RTX2070Max Q being the fastest and the Tesla K40m is the slowest in this comparison. Another point to note is that the P400 outperforms the RTX2060, being the first of an older architecture and besides it has reduced features than the second one.

Most of computation time is consumed by the solution of the system of linear equations. For solving the linear system equations GMRES solver is used. Computation times are showed in Table 2. Results indicate that Xeon CPU is faster than any GPU shown in table when linear system is small (20,000 unknowns). When the unknowns increase from 20,000 to 180,000 the Xeon processor solves linear

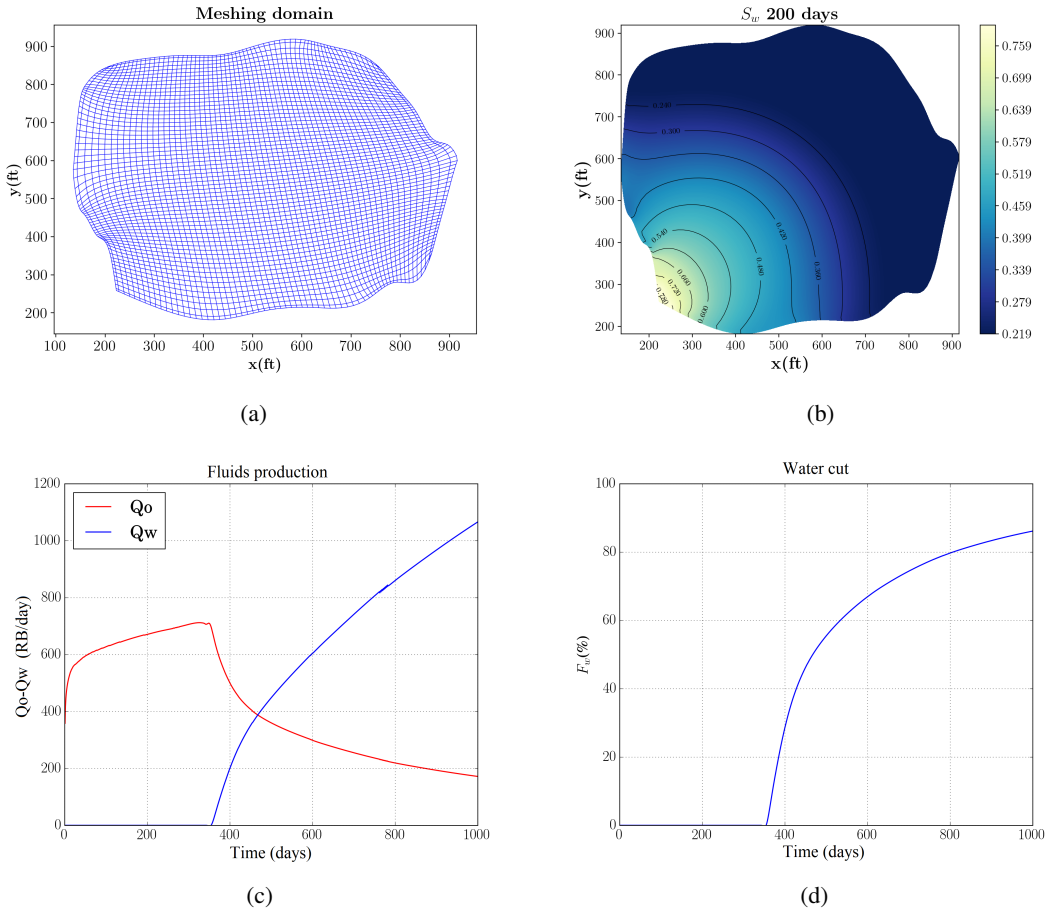


Figure 8: Results obtained for the second non orthogonal domain.

Table 1: Average computation time obtained for the calculation and construction of the Jacobian.

Mesh size	Xeon S. 4210	Tesla K40m	P400	RTX2060	RTX2070Max-Q
Nx×Ny	(s)	(s)	(s)	(s)	(s)
100×100	0.0175	9.51×10^{-5}	4.22×10^{-5}	6.43×10^{-5}	2.93×10^{-5}
200×200	0.0734	5.54×10^{-5}	4.48×10^{-5}	5.52×10^{-5}	3.21×10^{-5}
300×300	0.1648	7.89×10^{-5}	4.74×10^{-5}	5.39×10^{-5}	3.46×10^{-5}
500×500	0.4809	1.0×10^{-4}	5.17×10^{-5}	6.14×10^{-5}	3.43×10^{-5}
1,000×1,000	1.949	1.04×10^{-4}	*memory error*	6.23×10^{-5}	3.49×10^{-5}

system faster than two GPUs shown in the table (Tesla K40m and P400). But when the number of unknowns increase to 5×10^5, GPUs are much faster than CPU, for example in the case of the Tesla K40m we got 4.5x of speed up and for the RTX2070 12x of speed up are obtained.

Carrying out a comparison among the GPUs, it is observed once again the fastest is the RTX2070Max-Q. This difference is more noticeable compared to the GPUs of two previous architectures (Kepler and

Table 2: Average computation time obtained to solve linear system equations by using GMRES.

Unknowns	Xeon S. 4210	Tesla K40m	P400	RTX2060	RTX2070Max-Q
	(s)	(s)	(s)	(s)	
20,000	1.01	38.27	5.02	4.75	3.89
40,000	7.55	40.30	17.32	5.52	5.48
180,000	19.46	44.37	34.01	8.22	8.04
500,000	226.78	52.02	87.58	19.27	18.77
2×10^6	2557.33	96.74	*memory error*	61.86	61.55

Pascal) and it is highlighted even more when solving the smaller linear system, since when it is larger, only 1.57x speed up is obtained compared to the Tesla K40m.

6 CONCLUSIONS

A GPU based fully implicit simulator to solve two-phase flow in porous media capable of deal with non-orthogonal geometries has been presented. Our implementation was validated with a Five Spot benchmark. Our GPU strategy of parallelization allows to reduce the computation time both to build the Jacobian and to solve linear system equations. Simulator was tested with four different GPUs. Results show that the Tesla K40m with Kepler architecture is still a good choice to speed up a numerical code from hundreds of thousands to a couple of million cells. Also we have tested a Quadro P400 Pascal architecture low end card. This GPU was able to perform calculations with a quarter million of cells and half million of unknowns, it is also faster in small calculations than the Tesla K40m. It must be taken into account the P400 is mounted on a workstation and it provides video signal so most of its memory is occupied by this task. We used two GPUs with Turing architectures with similar run time results among them, however the fastest was the RTX2070max Q, this result is as expected, since the latter has more features in the hardware than the RTX2060. Through the analysis carried out in this work, it can be concluded that current GPU architectures are better in performance than previous ones. However, older architectures GPUs can still be used to speed up a numerical code as they are better in performance than a single core of a modern processor.

ACKNOWLEDGMENTS

This work was supported by UNAM-PAPIIT IA106820. We thank DGTIC UNAM for providing access to the computer cluster Miztli, through project number LANCAD-UNAM-DGTIC-065 under the direction of Dr. Ismael Herrera Revilla. Also, this work was performed as a part of collaborative research to Engineering Faculty and Geophysics Institute both from Universidad Nacional Autonoma de Mexico.

REFERENCES

[1] Beisembetov I.K., and Bekibaev T.T., Assilbekov B.K., Zhapbasbayev U.K. and Kenzhaliev B.K. Application of GPU in the development of 3D hydrodynamics simulators for oil recovery prediction. *AGH Drilling, Oil, Gas.* (2012) **29**: 75–88.

[2] Trapeznikova M.A., Churbanova N.G., Lyupa A.A., and Morozov D.N. Simulation of multiphase flows in the subsurface on GPU-based supercomputers. *Parallel Computing: Accelerating Computational Science Engineering (CSE), Advances in Parallel Computing* (2015) **25**:324–333.

[3] Ertekin, Turgay and Abou-Kassem, Jamal Hussein and King, Gregory R. *Basic applied reservoir simulation.* Society of Petroleum Engineers Richardson. Vol. VII., (2001).

[4] Chen, Zhangxin and Huan, Guanren and Ma, Yuanle. *Computational methods for multiphase flows in porous media.* SIAM. First Ed., (2006).

[5] Abou-Kassem, Jamal Hussein and Farouq-Ali, SM and Islam, M Rafiq *Petroleum Reservoir Simulations*. Elsevier. First Ed. (2013).

[6] Ma Y. and Chen Z. Parallel computation for reservoir thermal simulation of multicomponent and multiphase fluid flow. *Journal of Computational Physics.* (2004) **201(1)**:224–237.

[7] Dogru A.H., Fung L.S., Middya U., Al-Shaalan T.M., Pita J.A., HemanthKumar K., Su H., Tan J.C., Hoy H., Dreiman W. A next-generation parallel reservoir simulator for giant reservoirs. *In SPE/EAGE Reservoir Characterization & Simulation Conference.* (2009).

[8] Wang K., Liu H., and Chen Z. A scalable parallel black oil simulator on distributed memory parallel computers. *Journal of Computational Physics.* (2015) **301**:19–34.

[9] Yu, Song and Liu, Hui and Chen, Zhangxin John and Hsieh, Ben and Shao, Lei and others. GPU-based parallel reservoir simulation for large-scale simulation problems. *SPE Europec/EAGE Annual Conference.* (2012.)

[10] de la Cruz, Luis Miguel and Monsivais, Daniel. Parallel numerical simulation of two-phase flow model in porous media using distributed and shared memory architectures *Geofísica internacional.* (2014) **53:1**: 59–75.

[11] McClure, James E. and Prins, Jan F. and Miller, Cass T. A novel heterogeneous algorithm to simulate multiphase flow in porous media on multicore CPU–GPU systems. *Computer Physics Communications* (2014) **185(7)**:1865–1874.

[12] Teja-Juárez, V. Leonardo and de la Cruz, Luis. A Graphic Processing Unit (GPU) based implementation of an incompressible two-phase flow model in porous media *Geofísica internacional.* (2018) **57(3)**: 205–222.

[13] Herrera, Ismael and Pinder, George F. *Mathematical modeling in science and engineering: An axiomatic approach*. John Wiley & Sons. First Ed., (2012).

[14] Maliska, C. R. *Transfêrencia de Calor e Mecânica de Fluidos Computacional*. Livros Técnicos e Cientificos. First Ed., (1995).

[15] Gaël Guennebaud and Benoît Jacob and others. Eigen v3. *http://eigen.tuxfamily.org.* (2010).

[16] Maia, F and Dalton, S and others. Generic Parallel Algorithms for Sparse Matrix and Graph Computations. *http://cusplibrary.github.io/.* (2016).