IRSTI 30.19.33 DOI: https://doi.org/10.26577/JMMCS2023v119i3a9

E. Makhmut[*], **T.S. Imankulov**, **B.S. Daribayev**
Al-Farabi Kazakh National University, Kazakhstan, Almaty
[*]e-mail: erlanmahimut@gmail.com

PARALLEL IMPLEMENTATION OF MUSKAT-LEVERETT EQUATION USING CUDA

Capillary pressure plays a crucial role in waterflooding by influencing the displacement of oil by water in reservoir rocks. It is influenced by factors such as pore size distribution, wettability, and pore connectivity. Understanding and accounting for capillary pressure in the design and implementation of waterflooding operations can lead to improved oil recovery from reservoirs. In this work, to investigate the effects of capillary pressure in the waterflooding process in porous media, a one-dimensional numer-ical model is proposed, and the execution time of the serial model is computed. In the serial model, the absolute permeability, water and oil viscosity are considered as constant. In order to speed up the execu-tion time of the serial model, the high-performance computing technology CUDA is used, and the re-sults (execution time and speedup) on different threads are calculated. The results of serial and CUDA parallel models for the effects of capillary pressure are presented and analyzed.
Key words: HPC, CUDA, waterflooding, Capillary pressure, Saturation.

Е. Махмут[*], Т.С. Иманкулов, Б.С. Дарибаев
әл-Фараби атындағы Қазақ ұлттық Университеті, Қазақстан, Алматы қ.
[*]e-mail: erlanmahimut@gmail.com

CUDA технологиясын пайдаланып Маскет-Леверетт теңдеуін параллелді жүзеге асыру

Қабат жыныстарындағы мұнайдың сумен ығысуына әсер ету үшін капиллярлық қысым шешуші рөл атқарады. Оұан кеуектер көлемінің таралуы, сулану және кеуектердің қосылуы сияқты факторлар әсер етеді. Сумен ығыстыру операцияларын жобалау және жүзеге асыру кезінде капиллярлық қысымды ойластыру қабаттардан мұнай алуды жақсартуға әкелуі мүмкін. Бұл жұмыста, сумен ығыстыру процесінде кеуекті ортадағы капиллярлық қысымның әсерін зерттеу үшін бір өлшемді сандық модель ұсынылады және сериялық модельдің орындалу уақыты есептеледі. Сериялық модельде абсолютті өткізгіштік, су және майдың тұтқырлығы тұрақты мән алады деп қарастырылды. Сериялық модельдің орындалу уақытын жылдамдату үшін CUDA жоғары есептеу технологиясы қолданылады және әртүрлі ағындар бойынша нәтижелер (орындалу уақыты мен жылдамдығы) алынды. Капиллярлық қысымның әсеріне байланысты сериялық және CUDA параллель модельдерінің нәтижелері ұсынылды және талдау жасалынды.

Түйін сөздер: HPC, CUDA, су тасқыны, капиллярлық қысым, қанықтылық.

Е. Махмут[*], Т.С. Иманкулов, Б.С. Дарибаев
Казахский национальный университет имени аль-Фараби, Казахстан, г. Алматы
[*]e-mail: erlanmahimut@gmail.com

Параллельная реализация уравнения Маскета-Леверетта с использованием CUDA

Капиллярное давление играет решающую роль при заводнении, влияя на вытеснение нефти водой в пористых средах. На него влияют такие факторы, как распределение пор по размерам, смачиваемость и связность пор. Понимание и учет капиллярного давления при проектировании и реализации операций заводнения может привести к повышению нефтеотдачи пластов.

В данной работе для исследования влияния капиллярного давления в прецессии заводнения в пористых средах предлагается одномерная численная модель и вычисляется время выполнения последовательной модели. В последовательной модели абсолютная проницаемость, вязкость воды и нефти считаются постоянными. Для ускорения времени выполнения последовательной модели используется технология высокопроизводительных вычислений CUDA, а результаты (время выполнения и ускорение) рассчитываются на разных потоках. Представлены и проанализированы результаты последовательной и параллельных моделей CUDA для эффектов капиллярного давления.

Ключевые слова: Высокопроизводительные вычисления, CUDA, заводнение, капиллярное давление, насыщенность.

1 Introduction

Waterflooding [1] is a widely used method in the field of oil recovery. It involves injecting water into an oil reservoir to displace and recover additional oil. As mentioned earlier, the Buckley-Leverett [2] equation is commonly employed to model the two-phase flow of water and oil in porous media during waterflooding processes. One important aspect of the Buckley-Leverett equation is that it can exhibit a phenomenon known as shock waves or discontinuities in the so-lution. These occur when there is a sudden change in the saturation profile, resulting in sharp interfaces between the fluids. The presence of capillary pressure can smooth out these interfaces to some extent, but they may still be presence. In the production of oil engineering, engineers have made simplifying assumptions associated with the movement of water into the oil phase. However, it is important to acknowledge that simplifying assumptions are often made regarding the interaction between water and oil phases during waterflooding. These assumptions may not always reflect the complexities of real-world situations. In reality, there are functional relationships that exist among saturation, capillary pressure, and other factors between the oil and water phases. These functional relationships describe the interplay between saturation and capillary pressure, considering factors such as the wettability of the reservoir rock, interfacial tension between oil and water, and the presence of other fluids or contaminants. These relationships capture the effects of capillarity and multiphase interactions on fluid flow behavior. So, in this work, the Muskat-Leverett equation is considered.

In the past, various capillary pressure-saturation models have been developed and correlated from laboratory experiments conducted under equilibrium conditions. These models aim to describe the relationship between capillary pressure and saturation of the wetting phase in a porous medium.

Static capillary pressure-saturation relationships, such as the ones referenced in [3], have been widely used in mathematical studies and numerical simulations of multiphase flow in porous media. These relationships provide a means to incorporate the effects of capillary pressure into the governing equations and capture the behavior of fluid displacement.

It's important to note that while these static capillary pressure-saturation models have been widely used and provide valuable insights, they are empirical in nature and may have limitations when applied to real reservoir conditions. Factors such as hysteresis, dynamic effects, and heterogeneity of the porous medium can influence the capillary pressure-saturation relationship, and more sophisticated models may be required to capture these complexities accurately.

However, recent studies in soil physics have highlighted limitations in using laboratory-measured capillary pressure as an accurate representation of capillary pressure in cases involving large velocities or when the fluid content is in motion. These findings have led to the development of new aspects and theories in two-phase flow, challenging the classical capillary pressure-saturation relationship and proposing the concept of dynamic capillary pressure.

The theoretical studies referenced in [4-9] have contributed to the understanding of dynamic capillary pressure and its implications in two-phase flow modeling. These studies have proposed different formulations and approaches to describe the dynamic capillary pressure-saturation relationship, aiming to incorporate the influence of flow rate, fluid motion, and non-equilibrium conditions.

In recent years, high-performance computing technologies enable faster computations, improved modeling capabilities, and enhanced understanding of complex phenomena, leading to more efficient exploration, production, and reservoir management strategies. Especially, the application of HPC technologies in the oil and gas industry has led to significant advancements in computational capabilities and modeling accuracy. By leveraging parallel computing and utilizing technologies like OpenMP, MPI, and CUDA, researchers can perform complex calculations in a fraction of the time it would take with traditional computing methods. This, in turn, facilitates better decision-making, optimization of oil and gas production processes, and improved resource management.

CUDA is a parallel programming model that bridges the gap between CPUs and GPUs. CUDA enables developers to harness the parallel processing power of GPUs, which offer higher instruction throughput and memory bandwidth [10-14] compared to CPUs. This advantage makes GPUs well-suited for accelerating computations in various fields, including oil reservoir modeling.

In [15], the authors describe the development of a parallel algorithm using CUDA technology for three-dimensional reservoir modeling. By leveraging the capabilities of CUDA, the researchers aimed to increase the speedup of their computations and enhance the modeling capabilities in the oil reservoir field. The parallel algorithm likely utilized the parallel architecture of GPUs to distribute the computational load across multiple cores, enabling faster calculations and improved performance.

In [16] authors suggest that it is possible to conduct computations of complex mathematical models, specifically the three-dimensional Poisson equation, in real-time using mobile devices. The authors conducted comparative analyses of execution time and likely demonstrated the feasibility of leveraging the computational power of mobile devices to perform real-time computations. This work highlights the potential benefits of parallel computing, specifically CUDA, in the context of computational modeling in the oil and gas industry. The use of parallel implementations can lead to significant improvements in execution time, allowing for real-time computations on mobile devices and the ability to handle larger and more complex problems in reservoir simulation.

In [17], a parallel implementation for a Forward Reservoir Simulation (FRS) is presented. The authors developed a CUDA based parallel simulator for FRS, which allows for solving significantly larger problems compared to a serial implementation. The results indicate that the proposed CUDA based parallel implementation enabled solving a problem 82 times larger than the serial implementation. This demonstrates the effectiveness of CUDA in achieving substantial speedup and scalability in reservoir simulation, thereby enabling a more detailed and comprehensive analysis of reservoir behavior.

In [18] authors investigated parallel data processing in a hybrid CPU+GPU system, with a specific focus on utilizing multiple CUDA streams to overlap communication and computations. The paper analyzed the performance and performance-to-power consumption ratio of multi-stream data processing on modern multicore CPU+GPU systems. The authors obtained results that can be used to implement building blocks for data stream frameworks, particularly emphasizing multi CUDA stream communication optimization. This research likely contributes to optimizing data processing and achieving efficient utilization of hybrid CPU+GPU systems.

In order to accelerate the execution time of the serial model, in this work, the High-performance computing technology CUDA is applied, and the execution time and speedup of the CUDA parallel model are calculated and analyzed.

2 Mathematical model

The mass conservation equations for water and oil phases in a porous medium reservoir can be described as follows:

Water Phase: The mass conservation equation for the water phase, also known as the water saturation equation, represents the conservation of water mass within the reservoir. It considers the change in water saturation with respect to time and the flow of water through the porous medium. The equation can be written as:

$$m\frac{\partial S_w}{\partial t} + \mathrm{div}(\overrightarrow{v}_w) = 0, \tag{1}$$

Oil Phase: The mass conservation equation for the oil phase, also known as the oil saturation equation, represents the conservation of oil mass within the reservoir. Similar to the water phase equation, it considers the change in oil saturation with respect to time and the flow of oil through the porous medium. The equation can be written as:

$$m\frac{\partial S_o}{\partial t} + \mathrm{div}(\overrightarrow{v}_o) = 0, \tag{2}$$

$$S_o + S_w = 1. \tag{3}$$

Capillary pressure: The capillary pressure can be defined as the difference between the pressure in the wetting phase (e.g., water) and the pressure in the non-wetting phase (e.g., oil) at a given location within the reservoir. It can be expressed as:

$$P_o = P_o - P_w, \tag{4}$$

where, m is porosity, S_o and S_w are oil and water saturation, v_w, v_o are velocity of water and oil. P_c is the capillary pressure, P_o is the pressure in the oil (non-wetting) phase, and P_w is the pressure in the water (wetting) phase.

The Darcy's law expresses velocities:

$$\overrightarrow{\vartheta}_i = -k\left(\frac{f_i(S)}{\mu_i}\right)\frac{\partial P_i}{\partial x}, \qquad i = o, w; \tag{5}$$

where, k is the absolute permeability, μ_i is the viscosity of oil and water, $f_i(S)$ is the relative permeability expressed by following equations:

$$f_w(S_w) = S_w^2\,, \qquad f_o(S_o) = (1 - S_o)^2, \tag{6}$$

The pressure equation, expressed by equation (1) and (2), is following:

$$\frac{\partial}{\partial x}\left(-k\frac{f_w(S)}{\mu_w} - k\frac{f_o(S)}{\mu_o}\frac{\partial P_w}{\partial x}\right) - \frac{\partial}{\partial x}\left(-k\frac{f_o(S)}{\mu_o}\frac{\partial P_c}{\partial x}\right) = 0, \tag{7}$$

The initial condition at t=0 is given below:

$$S\big|_{t=0} = S_0, \qquad P\big|_{t=0} = P_0.$$

The boundary conditions:

$$S\big|_{x=0} = S_{\text{inj}}, \qquad \frac{\partial s}{\partial x}\bigg|_{x=1} = 0, \qquad P\big|_{x=0} = P_{\text{inj}}, \qquad P\big|_{x=1} = P_{\text{prod}}.$$

To solve the (1)-(7) system of equation, we considered the following assumptions:
– The flow is linear, horizontal and of constant thickness;
– The flow is isothermal, incompressible and obeys Darcy's law;
– Water and oil are immiscible;
– Gravity effects are negligible;
– The porosity is assumed constant;
– The density of water and oil are negligible.

3 Numerical model

The above mathematical model for oil recovery is nonlinear. To solve the Muskat-Leverett equation (7) Jacobi method was used.

For pressure (7):

$$P_{w(i)}^{(t+1)} = \frac{M_{(i+\frac{1}{2})}P_{w(i+1)}^t + M_{(i-\frac{1}{2})}P_{w(i-1)}^t - M1_{(i+\frac{1}{2})}P_{c(i+1)}^t + M1_{(i-\frac{1}{2})}P_{c(i-1)}^t}{M_{(i+\frac{1}{2})} + M_{(i-\frac{1}{2})}} \tag{8}$$

where,

$$M_{(i+\frac{1}{2})} = \frac{M_i + M_{(i+1)}}{2}\,; M_{(i-\frac{1}{2})} = \frac{M_i + M_{(i-1)}}{2}\,; M = M1 + M2\,;$$

$$M1 = -k\frac{f_{w(s)}}{\mu_w}\,; M2 = -k\frac{f_{0(s)}}{\mu_o}\,; M1_{(i+\frac{1}{2})} = \frac{M1_i + M1_{(i+1)}}{2}\,; M1_{(i-\frac{1}{2})} = \frac{M1_i + M1_{(i-1)}}{2}\,.$$

For saturation:

$$S_i^{(t+1)} = S_i^t + \frac{\Delta t}{m\Delta x^2}\left[K_{(i+\frac{1}{2})}(P_{(i+1)}^t - P_i^t) - K_{(i-\frac{1}{2})}(P_i^t - P_{(i-1)}^t)\right] \tag{9}$$

where,

$$K_{(i+\frac{1}{2})} = \frac{K_i + K_{(i+1)}}{2}\,; K_{(i-\frac{1}{2})} = \frac{K_i + K_{(i-1)}}{2}\,; K_i = -k\frac{f_w(S)}{\mu_w}.$$

For capillary pressure:

$$P_{c(i)}^t = a\left(\frac{0.072}{S_{w(i)}^t}\right) - \frac{S_{w(i)}^t}{2} + 0.391\,, \tag{10}$$

where,

$$a = 3.5 \times 10^{-6}.$$

4 CUDA programming model for Muskat-Leverett equation

In this work, CUDA technology is utilized for parallelization purposes. Figure 1 presents the process of initialization and calling the kernel. The primary challenge in this step is the initialization of device variables required for pressure computation. The process involves two main steps. Firstly, the corresponding variables are defined, specifying their data types and sizes. This step ensures that the necessary variables are available for computation. Secondly, memory is allocated on the device (GPU) for these variables. This memory allocation step reserves the required space on the GPU to store the variables and perform computations. Following memory allocation, the corresponding data is copied from the host (CPU) to the device. This data transfer operation enables the GPU to access and process the data efficiently.

It's important to note that before the "call kernel" operation, the block number and block size must be considered. The block number and block size are directly associated with the number of threads in the kernel. The block size represents the number of threads within a block, and the block number determines the total number of blocks to be executed. For the one-dimensional case, the block number and block size are determined by following a specific process, which is defined by the following process, for one-dimensional case:

```
dim3 threadsPerBlock(number\_of\_threads);
dim3 numBlocks((threadsPerBlock.x + N - 1) / threadsPerBlock.x);
kernel<<< numBlocks, threadsPerBlock >>>(parameters);
```

However, in general, the block size and block number are chosen based on the problem's characteristics, GPU architecture, and desired parallelism.

Thirdly, after the initialization and memory allocation steps described earlier, the "kernel"is called as shown in Figure 2. The kernel is a CUDA function that runs in parallel on the GPU. It performs the actual computations for the desired algorithm or operation.

The kernel function is designed to be executed by multiple threads in parallel. Each thread is responsible for performing a specific portion of the computation, typically operating on

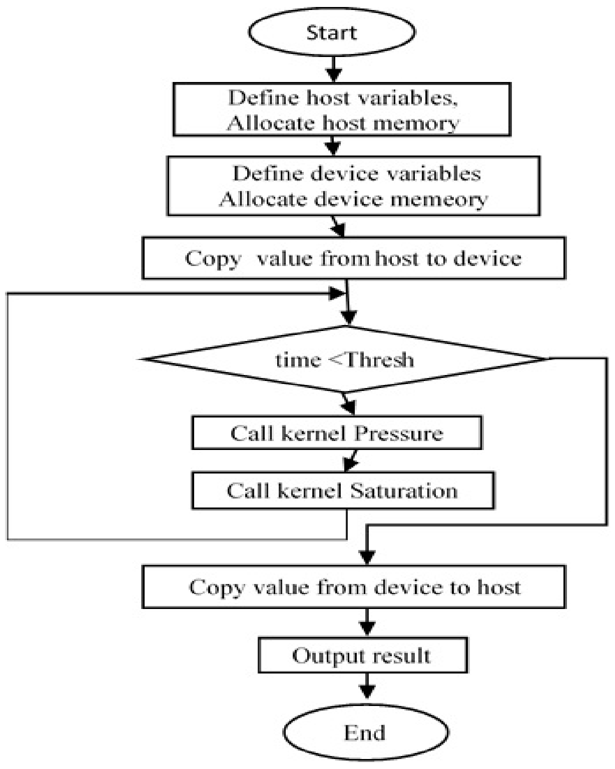

Figure 1: CUDA parallelization

different data elements. The block number and block size determined earlier play a crucial role in defining the number and organization of threads that will execute the kernel.

Once the “call kernel” function is invoked, the computation of pressure is carried out according to the process described in Figure 1. The "call kernel"function includes the following operations:

(1) Calculation of Global ID: Each thread in the GPU needs to have a unique identifier to control its independent execution. The global ID is calculated using the equation:

globalid = threadIdx.x + blockIdx.x * blockDim.x + gridIdx.x * (blockDim.x * gridDim.x).

This formula ensures that each thread is assigned a distinct ID based on its position with-in the grid of blocks and threads.

(2) Finding the Maximum Value of Pressure: The thread ID is calculated, and shared memory is defined to facilitate communication between threads within a block. This shared memory will be used to store intermediate results.

(3) Calculation of New Pressure: The value of pressure in the new time step is calculated using Equation (9), which likely represents the numerical scheme or algorithm used for the pressure update.

(4) Subtraction and Saving in Shared Memory: The difference between the new and old pressure values is computed, and the result is saved in shared memory. This step is performed to find the maximum value among the pressure differences calculated by the threads within a block.

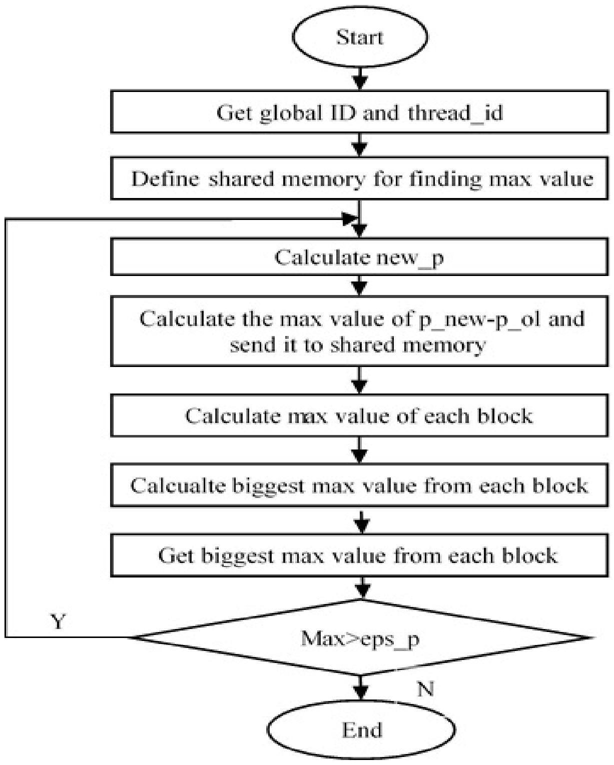
Start
Get global ID and thread_id
Define shared memory for finding max value
Calculate new_p
Calculate the max value of p_new-p_ol and send it to shared memory
Calculate max value of each block
Calcualte biggest max value from each block
Get biggest max value from each block
Y
Max>eps_p
N
End

Figure 2: The computation of pressure in CUDA

(5) Calculation of Maximum Value within blocks: A reduction algorithm is employed to determine the maximum value among the pressure differences within each block. This process involves iterative reduction steps to combine and compare values until the maximum value is obtained.

(6) Calculation of Maximum Value among blocks: Using atomic operations, the biggest maximum value obtained from each block is determined. Atomic operations ensure that multiple threads can safely access and update a shared variable without conflicts.

(7) Updating the Old Pressure Value: The old pressure value is updated with the newly calculated pressure value, as part of the iterative process for solving the pressure equation.

(8) Comparison with Computation Accuracy: The received biggest maximum value is compared with a predefined accuracy threshold. If the maximum value is less than the desired accuracy, the computation cycle (steps 3 to 7) continues. Otherwise, if the maximum value exceeds the accuracy threshold, the computation cycle is repeated until the desired accuracy is achieved.

This process iteratively calculates the pressure values until convergence, where the maximum change in pressure falls below the specified accuracy. The steps involve parallel computations by multiple threads within blocks and synchronization using shared memory to find the maximum values efficiently.

5 Results

Table 1 presents the input parameters of the CUDA parallel model. These various parameters affect the behavior and performance of the parallel model.

Table 1: Parameters of the model

Parameters	**Description**	**Value**
k	Absolute permeability	0.000001
μ_w	Water viscosity	0.09
μ_o	Oil viscosity	0.3
m	Porosity	0.2
P_{inj}	Injected pressure	0.5
P_{init}	Initial pressure	0.3
P_{pord}	Production of pressure	0.1
S_{inj}	Injected saturation	1.0
S_{init}	Initial saturation	0.001

Table 2 displays the execution time of the serial model. The table demonstrates how the execution time of the serial model varies as the number of points is increased. It indicates that as the number of points increases, the execution time also increases. This observation suggests that the computational complexity of the problem grows with the number of points, resulting in longer execution times for larger problem sizes. That is obtained by executing a serial algorithm in a workstation with an 11the Gen Intel(R) CORE(TM) i9-11900KF.

Table 2: The execution time of the sequential model.

Number of elements	**Execution time(s)**
2^{12}	4.2624
2^{13}	8.5646
2^{14}	16.6913
2^{15}	34.9052
2^{16}	70.6724
2^{17}	140.431
2^{18}	287.398
2^{19}	590.7
2^{20}	1208.34
2^{21}	2418.37

The performance evaluation of the parallel algorithms is presented in this section. Specifically, the focus is on assessing the execution time and speedup achieved by utilizing CUDA parallelization techniques. In the GPU device, various block sizes were tested, including 64, 128, 256, 512, and 1024. Among these block sizes, it was observed that a block size of 1024 yielded the best results in terms of performance when compared to the others.

Based on this observation, the block size of 1024 was chosen as the preferred configuration for all the tests conducted in this work. This consistent block size allows for fair and meaningful comparisons of the execution time and speedup across different parallel algorithms.

Table 3 provides a summary of the results obtained from the CUDA parallel algorithms. It includes the execution time for each block size, as well as the corresponding speedup achieved compared to the serial implementation. These results are obtained by the NVDIA Ge-Force RTX 2080 Ti.

Tabel 3: The execution time and speedup of the CUDA parallel algorithm.

number of elements	Execution time (128)	Speedup (128)	Execution time (256)	Speedup (256)	Execution time (512)	Speedup (512)	Execution time (1024)	Speedup (1024)
2^{12}	2.99885	1.42	4.21201	1.0	7.18712	0.59	13.2427	0.3
2^{13}	3.72914	2.3	4.59255	1.87	7.50429	1.14	13.7876	0.6
2^{14}	6.32544	2.64	5.3485	3.12	7.8867	2.12	13.9358	1.2
2^{15}	11.7902	2.96	9.62928	3.63	8.54534	4.1	14.3246	2.4
2^{16}	23.323	3.0	17.8835	3.95	15.8657	4.45	14.9749	4.7
2^{17}	49.1264	2.86	33.3089	4.2	28.8945	4.86	28.58582	4.9
2^{18}	100.428	2.86	65.9505	4.1	55.8267	5.15	55.452	5.2
2^{19}	203.329	2.91	128.392	4.6	110.739	5.33	108.783	5.4
2^{20}	412.112	2.93	254.881	4.74	215.897	5.6	214.71	5.6
2^{21}	818.841	2.95	505.169	4.79	423.209	5.7	413.065	5.86

By analyzing the execution time and speedup values presented in Table 3, the effectiveness of the CUDA parallel algorithms can be assessed. These results provide insights into the performance gains achieved through parallel computing and help validate the benefits of utilizing CUDA technology for accelerating the computations in the oil reservoir simulation domain.

Figure 3 shows the execution time of serial and CUDA parallel algorithm with a block size equal to 128.

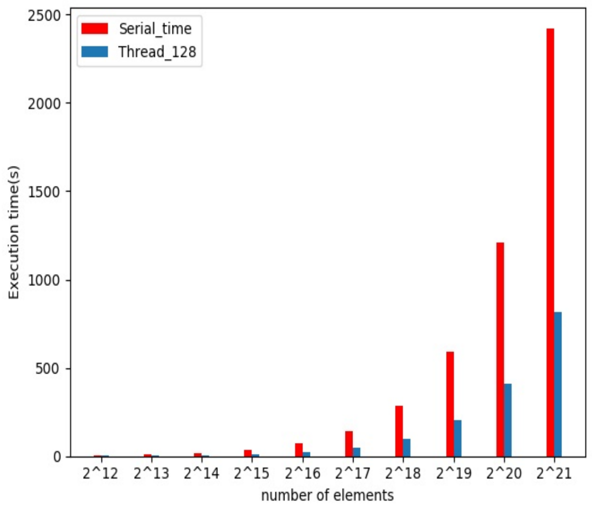

Figure 3: Computation time of serial and CUDA parallel algorithm

From Figure 3, it is evident that the CUDA parallel algorithm performs equally fast as the serial model as the number of elements increases.

Figure 4 visualizes the parallel execution time of the CUDA parallel algorithms on different block sizes. On the other hand, Figure 6 showcases the speedup achieved by the parallel algorithms in the computation of the Muskat-Leverett equation.

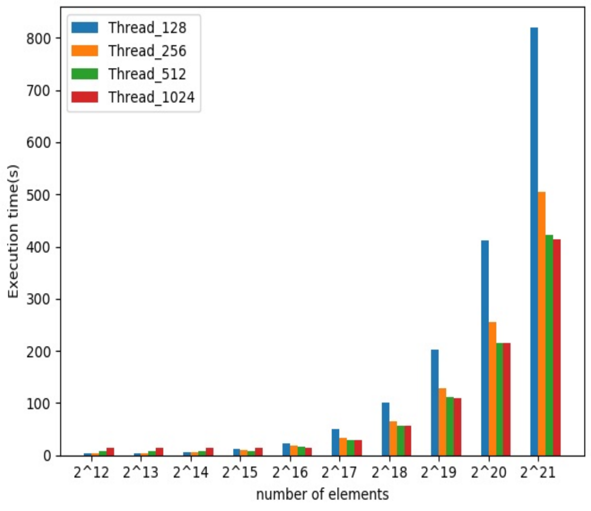

Figure 4: Computation time of different block sizes of parallel algorithm

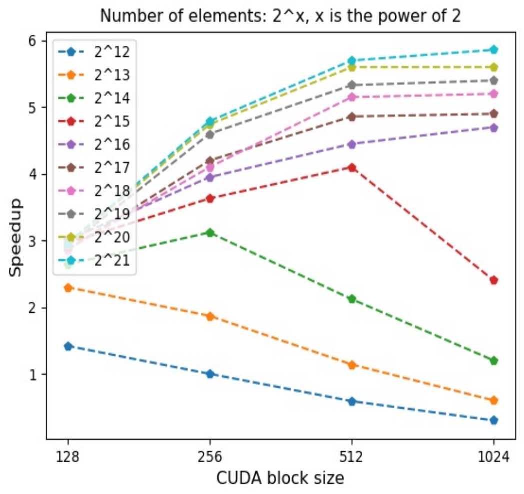

Figure 5: Speedup of parallel algorithm

From Figure 4, the CUDA parallel algorithm with the larger block size performs equally fast as the number of elements increases. Notably, when the block size is set to 1024, it provides the most optimal results compared to other block sizes. Initially, smaller block sizes (128, 256) exhibit good speedup when compared to larger block sizes (512, 1024). However, as the number of elements increases, the larger block sizes progressively yield better and improved results, shown in Figure 5.

By referring to Figure 4 and Figure 5, researchers and readers can gain insights into the efficiency and effectiveness of the CUDA parallel algorithms specifically applied to solving the Muskat-Leverett equation. These figures provide a visual representation of the parallel execution time and the resulting speedup, allowing for a comprehensive assessment of the performance benefits and scalability achieved through parallelization.

6 Conclusions

This work focuses on solving the oil displacement problem using the Muskat-Leverett equation. The numerical model is solved using the Jacobi method. In this work, the serial algorithm and a CUDA parallel algorithm are developed. The results obtained from the parallel algorithms are thoroughly analyzed. The analysis reveals that the execution time and speedup of the model are influenced by various factors, such as thread divergence, block size, synchronization, and compile time. These factors play a crucial role in optimizing the performance of parallel algorithms.

Table 3 presents the findings regarding the impact of these factors on the execution time or speedup. By examining the table, researchers and readers can gain insights into the performance characteristics and scalability of the parallel algorithms about the number of points in the computational domain.

Notably, the results demonstrate that with an increasing number of points, the speedup achieved by the parallel algorithm improves in comparison to the serial version. This indicates that the parallel algorithms effectively leverage the computational capabilities of parallel computing platforms, resulting in enhanced performance and reduced execution time.

Acknowledgments

This research was supported by the Grant No. AP19576314 “Development of high-performance hybrid algorithms for solving oil displacement problems based on classical (GPU) and quantum computing” of Ministry of Science and Higher Education of the Republic of Kazakhstan.

References

[1] Paul Willhite G. (1986)., "Waterflooding." , *Society of petroleum engineers, Richardson, TX, Textbook Series Volume,* 3: 1-7.

[2] Ming Yang, Cunliang Chen, Yu Wang, Xiaohui Wu, Dong Ma,Buckley S.E., Leverett M.C. (1942) Mechanism of Fluid Displacement in Sands , *Transactions AIME,* Vol.146, pp.107-116.

[3] Brooks R. H., Corey A. T. , Hydraulic properties of porous media , Hydrology Paper 3 (1964), 27.

[4] Gray W., Hassanizadeh S.,"Paradoxes and Realities in Unsaturated Flow Theory" , *Water Resources Research,* 27 (1991), 1847–1854.

[5] Gray W., Hassanizadeh S., "Unsaturated Flow Theory Including Interfacial Phenomena" , *Water Resources Research* 27 (1991), 1855–1863.

[6] Hassanizadeh S., Gray W., "Thermodynamic basis of capillary pressure in porous media" , *Water Resources Research* 29 (1993), 3389–3406.

[7] HassanizadehS., Celia M., Dahle H.,"Dynamic Effect in the Capillary Pressure-Saturation Relationship and its Impacts on Unsaturated Flow" , *Vadose Zone Journal* 1 (2002), 38–57.

[8] Das D., Hassanizadeh S., Rotter B., Ataie-Ashtiani B.,"A Numerical Study of MicroHeterogeneity Effects on Upscaled Properties of Two-Phase Flow in Porous Media" , *Transport in Porous Media* 56 (2004), 329–350.

[9] Beliaev, Hassanizadeh S., "A Theoretical Model of Hysteresis and Dynamic Effects in the Capillary Relation for Two-phase Flow in Po-rous Media" , *Transport in Porous Media* 43 (2001), 487–510.

[10] NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110, 2012, Available: http://www.nvidia.com/content/PDF/kepler/NVIDIA-KeplerGK110-ArchitectureWhitepaper.pdf.

[11] Wilt N., The Cuda Handbook: A Comprehensive Guide to GPU Programming , *Pearson Education,* 2013.

[12] C. Nvidia , *Programming guide, ed,*2008.

[13] Sanders J., Kandrot E., CUDA by Example: An Introduction to General-Purpose GPU Programming , *Addison-Wesley Professional,* 2010.

[14] Kirk D.B., Wen-mei W.H., Programming Massively Parallel Processors: A Hands-On Approach , Newnes, 2012.

[15] Beisembetov I. K., Bekibaev T. T., Assilbekov B. K., Zhapbasbayev U. K., Kenzhaliev B. K.,"High-performance computing in oil recovery simulation based on CUDA." , *Proceedings of 10th World Congress on Computational Mechanics. Sao-Paulo, Brazil(2012).*

[16] Akhmed-Zaki D.Zh., Daribayev B.S., Imankulov T.S., Turar O.N, "High-performance computing of oil recovery problem on a mobile platform using cuda technology" , *Eurasian Journal Of Mathematical And Computer Applications* , Volume 5, Issue 2 (2017) 4–13.

[17] Ayham Zaza, Abeeb A. Awotunde, Faisal A. Fairag, Mayez A. Al-Mouhamed, "A CUDA-based parallel multi-phase oil reservoir simulator" , Computer Physics Communications (2016),

[18] Pawel Czarnul, "Investigation of Parallel Data Processing Using Hybrid High-Performance CPU + GPU Systems and CUDA Streams" , *Computing and Informatics* , Vol. 39, 2020, 510–536.