Int. J. Nonlinear Anal. Appl. 12 (2021) No. 1, 351-364
ISSN: 2008-6822 (electronic)
http://dx.doi.org/10.22075/ijnaa.2021.4809

# Comparative Analysis Of Parallel Algorithms For Solving Oil Recovery Problem Using CUDA and OpenCL

Timur Imankulov[a,b], Beimbet Daribayev[b], Saltanbek Mukhambetzhanov[b,*]

[a] *Yessenov University, Aktau, Kazakhstan*
[b] *Al-Farabi Kazakh National University, Almaty, Kazakhstan*

*(Communicated by Haydar Akca)*

**Abstract**

In this paper the implementation of parallel algorithm of alternating direction implicit (ADI) method has been considered. ADI parallel algorithm is used to solve a multiphase multicomponent fluid flow problem in porous media. There are various technologies for implementing parallel algorithms on the CPU and GPU for solving hydrodynamic problems. In this paper GPU-based (graphic processor unit) algorithm was used. To implement the GPU-based parallel ADI method, CUDA and OpenCL were used. ADI is an iterative method used to solve matrix equations. To solve the tridiagonal system of equations in ADI method, the parallel version of cyclic reduction (CR) method was implemented. The cyclic reduction is a method for solving linear equations by repeatedly splitting a problem as a Thomas method. To implement of a sequential algorithm for solving the oil recovery problem, the implicit Thomas method was used. Thomas method or tridiagonal matrix algorithm is used to solve tridiagonal systems of equations. To test parallel algorithms personal computer installed Nvidia RTX 2080 graphic card with 8 GB of video memory was used. The computing results of parallel algorithms using CUDA and OpenCL were compared and analyzed. The main purpose of this research work is a comparative analysis of the parallel algorithm computing results on different technologies, in order to show the advantages and disadvantages each of CUDA and OpenCL for solving oil recovery problems.

*Keywords:* CUDA, OpenCL, Cyclic Reduction, ADI, Oil Recovery Problem

*Corresponding author
*Email addresses:* imankulov.timur@gmail.com (Timur Imankulov), beimbet.daribayev@gmail.com (Beimbet Daribayev), mukhambetzhanov_@mail.ru (Saltanbek Mukhambetzhanov)

*Received:* 10 October 2020 *Revised:* 25 January 2021

## 1. Introduction

Hydrodynamic modelling of processes in oil reservoirs is one of the complex fluid mechanics problems. This is due to the fact that flow processes in underground reservoirs can be very complex. It is necessary to take into account phase transitions, chemical transformations, temperature effects, degassing processes with a drop in pressure at the bottomhole and in the reservoir, etc. When modeling the filtration of multiphase multicomponent fluids, the properties of phases can vary depending on the composition of the phases, temperature (in oil reservoirs they can reach up to 200 C) and pressure (in oil reservoirs can reach up to 150 MPa).

The oil phase consists of hydrocarbon components that range from the lightest (methane) to the heaviest (bitumen). A large number of components can be grouped into pseudo-components. Knowing the physical parameters of the pseudo-component (molecular weight, critical pressure, critical temperature, compressibility, density, viscosity, thermal conductivity, and specific heat), it is possible to simulate the process of multicomponent filtration.

The level of development of computing technology today highly determines the pace of technical progress and success in solving fundamental scientific problems, among which the class of problems is generally recognized as Grand challenges: these are fundamental scientific or engineering problems with a wide range of applications, the effective solution of which is possible only with the use of powerful computing resources (supercomputer) with a performance of hundreds of Gflops and higher. One of these tasks is modeling the optimal strategy for the development of oil and gas fields, the creation and use of permanent geological and technological models for managing the development of oil and gas fields, requiring processing of huge amounts of information.

All the world's oil and gas companies are already using numerical modeling to some extent when developing new fields and determining the optimal method for extracting resources. The oil and gas industry are one of the leaders in high-performance resources - the growth of computing power in this composition is up to 100% per year. In total, 10% of the world's TOP 500 supercomputers are used for computing geophysical problems.

The role of high-performance computing in the oil and gas industries growth up every year. The progress of modern GPUs has opened up possibilities for computing these kinds of tasks. Given this fact, a parallel algorithm of ADI method [28] on modern GPUs to solve the multiphase, multicomponent fluid flow problem in porous media were considered. ADI is a finite-difference numerical method for solving parabolic, hyperbolic and elliptic equations and is widely used in scientific and engineering fields [5, 12, 24, 27]. In the ADI method, each numerical step is divided into several substeps based on a spatial dimension of the problem, and the system of linear equations is solved implicitly in one direction and explicitly in other direction. In the ADI method, at each substep, equations have a tridiagonal structure and can be solved using the Thomas method [40].

To accelerate this kind of parallel algorithms, platforms with non-specialized GPGPU (general-purpose computing on graphics processing units) are excellent. The GPU achieves high performance by running more than a thousand threads at the same time, and each of them processing different computing blocks. In various research fields, there have been many successful GPU-based implementations, such as medical image analysis [18] and computational fluid dynamics [43, 44]. With the advent of high-performance computing in GPUs, parallel algorithms have been developed for solving systems of tridiagonal equations [42, 41, 10, 11, 13, 22, 33, 32]. Zhang and et al. first implemented parallel cyclic reduction (PCR) algorithm [42], and then proposed a hybrid CR-PCR algorithm. The PCR-Thomas hybrid algorithm was proposed by N. Sugar [33]. In each direction of the ADI method, CR algorithms were used as a parallel algorithm for solving systems of tridiagonal equations. Its solver achieves high-performance with a huge number of available systems.

The systems of tridiagonal equations were first implemented on GPUs in the work of M. Kass and et al. to perform effective depth of field blurring [20]. Their implementation was based on CR and was written in GPU shading languages. S. Sengupta and et al. implemented CR using CUDA technology and applied it to modelling shallow water flows in real time [21, 35]. Also, we published the results of oil problems [2] and their parallel implementations with CUDA on mobile devices [1, 17]. To implement the parallel ADI method, the parallel versions of existing CR algorithm using CUDA [19] and OpenCL [8] were used. The main reason for choosing this framework is its heterogeneity. OpenCL is an open standard for cross-platform parallel programming of various processors found on personal computers, servers, mobile devices and embedded platforms. OpenCL provides a standard interface for parallel computing using task and data parallelism. While OpenCL can interact with a large number of devices natively, this does not mean that the code will run optimally on all of them without any effort. For solving the sequential algorithm, Thomas method was implemented [39].

## 2. Mathematical Model

The compositional model is widely used in the oil and gas industry. These basic equations may seem easy, but there are certain challenges in solving them. Let us consider a mathematical model of multicomponent filtration of a three-phase liquid in a porous medium.

We have considered a mathematical model of multiphase, multicomponent fluid flow in a porous media. The model takes into account 3 phases (phase number - $\alpha$ : $\alpha$ = o - oil phase, $\alpha$ = g - gas phase, $\alpha$ = w - water phase). A component number in phase $\alpha$ will be denoted by $j$. The continuity equation for a separate component with the number $j$ contained in the phase $\alpha$ in differential form and the phase velocities $\overrightarrow{\boldsymbol{U}}_{\boldsymbol{\alpha}}$ according to Darcy's law has following form [6, 16]:

$$\frac{\partial}{\partial t}\left(\sum_{\alpha} m\rho_{\alpha}S_{\alpha}C_{\alpha j}\right)+div\left(\sum_{\alpha}\rho_{\alpha}C_{\alpha j}\overrightarrow{U}_{\alpha}\right)=div\left(mD_{\alpha j}S_{\alpha}grad\left(\rho_{\alpha}C_{\alpha j}\right)\right)$$

$$\alpha=\overline{w,\ o,\ g};\qquad j=\overline{1,n} \tag{2.1}$$

$$u_{\alpha}=\ -K\ \frac{k_{ra}\left(\mathrm{S}\right)}{\mu_{\alpha}}\ \nabla\ (p_{\alpha}-\ p_{\alpha}gz), \tag{2.2}$$

$$\alpha=w,o,g;$$

$$S_w+S_o+S_g=1$$

where, $\boldsymbol{m}$ – rock porosity, $\boldsymbol{S_{\alpha}}$ – phase saturation, $\overrightarrow{\boldsymbol{U}}_{\boldsymbol{\alpha}}-$ velocity vector of phase filtration, $\boldsymbol{D_{j\alpha}}-$ molecular diffusion coefficient of component $\boldsymbol{j}$ in phase $\boldsymbol{\alpha}$, $\boldsymbol{K}-$ absolute permeability, $\boldsymbol{k_{ra}}=\boldsymbol{k_{ra}}\left(\boldsymbol{S},\boldsymbol{T}\right)-$ relative phase permeability of $\boldsymbol{\alpha}$, $\boldsymbol{\mu_{\alpha}}=\boldsymbol{\mu_{\alpha}}\left(\boldsymbol{p_{\alpha}},\boldsymbol{T},\boldsymbol{C_{i\alpha}}\right)-$ phase viscosity, $\boldsymbol{p_{\alpha}}=\boldsymbol{p_{\alpha}}(\boldsymbol{S})\ -$ phase pressure, $\boldsymbol{\rho_{\alpha}}-$ phase mass density, $\boldsymbol{g}-$ acceleration of gravity, $\boldsymbol{z}-$ depth. The phase pressures are related by the following expressions:

$$p_o=p_w+p_{cow}(S_w),p_g=p_o+p_{cog}\left(S_g\right) \tag{2.3}$$

where $\boldsymbol{p_{cow}}(\boldsymbol{S_w},\boldsymbol{T})$, $\boldsymbol{p_{cog}}\left(\boldsymbol{S_g}\ ,\boldsymbol{T}\right)-$ capillary pressure at the border of water-oil (cow) and oil-gas (cog). These dependencies are considered to be specified saturation functions.

The oil phase is composed of hydrocarbon components ranging from lightest (methane) to heavier (bitumen). A large number of components can be grouped into pseudo-components. Knowing the physical parameters of the pseudo-component (molecular weight, critical pressure, critical temperature, compressibility, density, viscosity, thermal conductivity, and specific heat), it is possible to simulate the process of multicomponent filtration.

Let's write down the system of equations for 4 components. For each component, the equations of continuity will look like this:

$$\frac{\partial}{\partial t}\left(\rho_w S_w C_{w1} + \rho_o S_o C_{o1} + \rho_g S_g C_{g1}\right) +$$
$$+div\left(\rho_w C_{w1}\overrightarrow{U_w} + \rho_o C_{o1}\overrightarrow{U_o} + \rho_g C_{g1}\overrightarrow{U_g}\right) =$$
$$= div\left(mD_{w1}S_w grad\left(\rho_w C_{w1}\right) +\right.$$
$$\left. + mD_{o1}S_o grad\left(\rho_o C_{o1}\right) + mD_{g1}S_g grad\left(\rho_g C_{o1}\right)\right)$$
$$\frac{\partial}{\partial t}\left(\rho_w S_w C_{w2} + \rho_o S_o C_{o2} + \rho_g S_g C_{g2}\right) +$$
$$+div\left(\rho_w C_{w2}\overrightarrow{U_w} + \rho_o C_{o2}\overrightarrow{U_o} + \rho_g C_{g2}\overrightarrow{U_g}\right) =$$
$$= div\left(mD_{w2}S_w grad\left(\rho_w C_{w2}\right) +\right.$$
$$\left. + mD_{o2}S_o grad\left(\rho_o C_{o2}\right) + mD_{g2}S_g grad\left(\rho_g C_{o2}\right)\right)$$
$$\frac{\partial}{\partial t}\left(\rho_w S_w C_{w3} + \rho_o S_o C_{o3} + \rho_g S_g C_{g3}\right) +$$
$$+div\left(\rho_w C_{w3}\overrightarrow{U_w} + \rho_o C_{o3}\overrightarrow{U_o} + \rho_g C_{g3}\overrightarrow{U_g}\right) =$$
$$= div\left(mD_{w3}S_w grad\left(\rho_w C_{w3}\right) +\right.$$
$$\left. + mD_{o3}S_o grad\left(\rho_o C_{o3}\right) + mD_{g3}S_g grad\left(\rho_g C_{o3}\right)\right)$$
$$\frac{\partial}{\partial t}\left(\rho_w S_w C_{w4} + \rho_o S_o C_{o4} + \rho_g S_g C_{g4}\right) +$$
$$+div\left(\rho_w C_{w4}\overrightarrow{U_w} + \rho_o C_{o4}\overrightarrow{U_o} + \rho_g C_{g4}\overrightarrow{U_g}\right) =$$
$$= div\left(mD_{w4}S_w grad\left(\rho_w C_{w4}\right) +\right.$$
$$\left. + mD_{o4}S_o grad\left(\rho_o C_{o4}\right) + mD_{g4}S_g grad\left(\rho_g C_{o4}\right)\right)$$

*Table 1: Composition of water, oil and gas*

| Component number, $j$ | Gas | Oil | Water |
|---|---|---|---|
| 1 | - | - | + |
| 2 | + | - | - |
| 3 | + | + | - |
| 4 | - | + | - |

Table 1 shows the compositional compositions of the gas, water and oil phases. Component number 1 is contained only in the aqueous phase, component 2 is contained only in the gas phase, component 3 is included in the gas and oil phases, and component number 4 is contained only in the oil phase. Hence, the following expressions follow:

$$\begin{gathered} C_{w1} = 1, \\ C_{w2} = 0, \\ C_{w3} = 0, \\ C_{w4} = 0, \\ C_{o1} = 0, \\ C_{o2} = 0, \\ C_{g1} = 0, \\ C_{g4} = 0, \\ C_{o3} + C_{o4} = 1, \\ C_{g2} + C_{g3} = 1. \end{gathered} \tag{2.4}$$

Taking into account (2.4), the system of equations for 4 components can be rewritten as follows:

$$\begin{gathered} m\frac{\partial}{\partial t}\left(\rho_w S_w\right) + div\left(\rho_w \overrightarrow{U_w}\right) = 0 \\ m\frac{\partial}{\partial t}\left(\rho_g S_g C_{g2}\right) + div\left(\rho_g C_{g2}\overrightarrow{U_g}\right) == \ div(mD_{g2}S_g \ grad\left(\rho_g C_{g2}\right)) \\ m\frac{\partial}{\partial t}\left(\rho_o S_o C_{o3} \ + \ \rho_g S_g C_{g3}\right) \\ +div\left(\rho_o C_{o3}\overrightarrow{U_o} + \ \rho_g C_{g3}\overrightarrow{U_g}\right) \\ = div(mD_{o3}S_o grad\left(\rho_o C_{o3}\right) \\ +mD_{g3}S_g grad\left(\rho_g C_{g3}\right)) \\ m\frac{\partial}{\partial t}\left(\rho_o S_o C_{o4}\ \right) + div\left(\rho_o C_{o4}\overrightarrow{U_o}\right) = div(mD_{o4}S_o grad\left(\rho_o C_{o4}\right)) \end{gathered} \tag{2.5}$$

The relationship between pressure, temperature and phase volume is determined by the equation of state. The thermodynamic behaviour of liquids is determined by cubic equations of the van der Waals type (equations of state of Peng-Robinson, Redlich-Kwong, Soave-Kwong, Zudkevich-Ioffe, etc.) [25, 29, 4, 31, 30, 36, 45].

The distribution of each component of the hydrocarbon phase between the two phases is a stable equilibrium state, which can also be said as the Gibbs free energy for a compositional system:

$$\boldsymbol{f_{mo}}\left(\boldsymbol{p_o}, \boldsymbol{x_{1o}}, \boldsymbol{x_{2o}}, \ldots, \boldsymbol{x_{N_c o}}\right) = \boldsymbol{f_{mg}}\left(\boldsymbol{p_g}, \boldsymbol{x_{1g}}, \boldsymbol{x_{2g}}, \ldots, \boldsymbol{x_{N_c g}}\right)$$

where $\boldsymbol{f_{mo}}$ and $\boldsymbol{f_{mg}}$ are functions of the fugacity of the $m$-th component in the gas and liquid phases, respectively.

After several simple arithmetic operations, equation (2.5) is reduced to the form (in the two-dimensional case):

$$\frac{\boldsymbol{\partial}}{\boldsymbol{\partial x}}\left[\left(\boldsymbol{M_x}\right)\frac{\boldsymbol{\partial P_g}}{\boldsymbol{\partial x}}\right] = \ \boldsymbol{R(x, t, P_{cow}, P_{cog}, S_w, S_o, S_w)}$$

which is further solved by the sweep method.

The relative permeabilities can be determined by Stone's approximation in a modified version of Aziz and Settari:

$$\overline{S_\alpha} = \frac{S_\alpha - S_{\alpha r}}{1 - S_{wr} - S_{or} - S_{gr}}, \qquad \alpha = w, o, g.$$

$$k_{rw}\left(\overline{S}_w\right) = \begin{cases} \overline{S}_w^{1/2}\left[1 - \left(1 - \overline{S}_w^{n/(n-1)}\right)^{(n-1)/n}\right]^2, & 0 < \overline{S}_w < 1, \\ 1, & \overline{S}_w \geq 1, \\ 0, & \overline{S}_w \leq 0; \end{cases}$$

$$k_{rg}\left(\overline{S}_g\right) = \begin{cases} \overline{S}_g^{1/2}\left[1 - (1 - \overline{S}_g)^{n/(n-1)}\right]^{2(n-1)/n}, & 0 < \overline{S}_g < 1, \\ 1, & \overline{S}_g \geq 1, \\ 0, & \overline{S}_g \leq 0; \end{cases}$$

$$k_{ow}\left(\overline{S}_w\right) = \begin{cases} (1 - \overline{S}_w)^{1/2}\left[1 - \overline{S}_w^{n/(n-1)}\right]^{2(n-1)/n}, & 0 < \overline{S}_w < 1, \\ 1, & \overline{S}_w \leq 0, \\ 0, & \overline{S}_w \geq 1; \end{cases}$$

$$k_{og}\left(\overline{S}_o\right) = \begin{cases} \overline{S}_o^{1/2}\left[1 - \left(1 - \overline{S}_w^{n/(n-1)}\right)^{(n-1)/n}\right]^2, & 0 < \overline{S}_o < 1, \\ 1, & \overline{S}_o \geq 1, \\ 0, & \overline{S}_o \leq 0; \end{cases}$$

$$k_n\left(\overline{S}_w, \overline{S}_o\right) = \begin{cases} \frac{\overline{S}_o k_{ow}\left(\overline{S}_w\right) k_{og}\left(\overline{S}_o\right)}{\left(1-\overline{S}_w\right)\left(\overline{S}_w+\overline{S}_o\right)}, & \overline{S}_w < 1,\ \overline{S}_w + \overline{S}_o < 1, \\ 0, & \textit{in other case}; \end{cases}$$

where $\overline{\boldsymbol{S}}_{\boldsymbol{\alpha}}$ - effective phase saturation, $\boldsymbol{S}_{\boldsymbol{\alpha r}}$ - critical phase saturation, $n = 3.25$.

Capillary pressures are described by Parker's approximate model:

$$p_{cow}\ (S_w) = p_o - p_w = \frac{1}{\zeta}\left(\overline{S}_w^{n/(n-1)} - 1\right)^{1/n},$$

$$p_{cog}\ (S_g) = p_g - p_o = \frac{1}{\zeta}\left((1 - \overline{S}_g)^{n/(n-1)} - 1\right)^{1/n}$$

## 3. Why GPU?

Let's take a quick look at some of the significant differences between the areas and features of the CPU and video card applications.

The CPU is initially adapted for solving general problems and works with arbitrarily addressable memory. Programs on the CPU can directly access any linear and homogeneous memory cells.

This is not the case for the GPU. There are as many as 6 types of memory in CUDA. You can read from any physically accessible cell, but not all cells can be written. The reason is that a GPU is a specific device for specific purposes anyway. This limitation was introduced to increase the speed of certain algorithms and reduce the cost of equipment.

An age-old problem in most computing systems is that memory is slower than a processor. CPU manufacturers solve this problem by introducing caches. The most frequently used areas of memory are placed in cache or cache memory, operating at the frequency of the processor. This allows you

to save time when accessing the most frequently used data and load the processor with the actual calculations.

Note that caches are actually transparent to the programmer. Both during reading and writing, data does not go directly to RAM, but passes through caches. This allows, in particular, to quickly read a certain value immediately after writing.

The GPU also has caches, and they are also important, but this mechanism is not as powerful as on the CPU. Firstly, not all memory types are cached, and secondly, caches are read-only.

On the GPU, slow memory accesses are hidden by using parallel computing. While some tasks are waiting for data, others are working, ready for calculations. This is one of the main principles of CUDA, which can greatly increase the performance of the system as a whole.

## 4. CUDA architecture

The CUDA program, called the kernel, is executed by thousands of threads grouped into blocks. Blocks are placed in a grid and scheduling in any of the available cores, which automates the scaling of different architectures. The smallest number of threads that are scheduled are called warps. All threads within the warp execute the same instruction simultaneously. The size of the warp is implementation-defined and has to do with shared memory organization, data access patterns, and data control [15, 37, 9, 34, 14, 7].

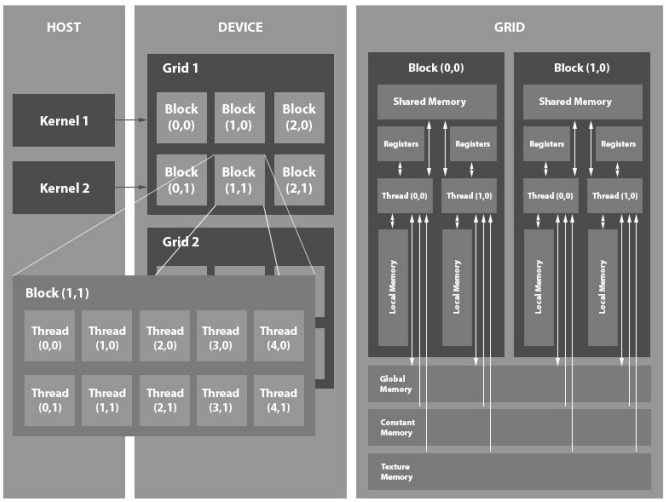


Figure 1: Architecture and memory structure of CUDA

The memory hierarchy is organized into read-only global, persistent and texture memory, with special functions such as caching or data prefetching. These memories are available to all threads, each thread has its own local and private memory. There are mechanisms for synchronizing threads within a block, but not among different blocks. Because of this, limited data cannot be shared between blocks (Figure 1). This becomes problematic when a thread needs data that was created outside of its block.

## 5. OpenCL architecture

The OpenCL specification moves away from architectural differences by presenting any platform as a host system associated with one or more devices. Each device consists of one or more compute units, which can include several processing elements. All calculations take place in the processing

elements of the device. It is obvious that such a general model can be used to describe any system, including central and graphics processors, as well as various types of accelerators. The control application runs on the host system and issues OpenCL commands to organize computing on devices. The processing elements of one computational module can execute the instruction stream as SIMD *(Single Instruction, Multiple Data)* units or as SPMD *(Single Program, Multiple Data)* units.

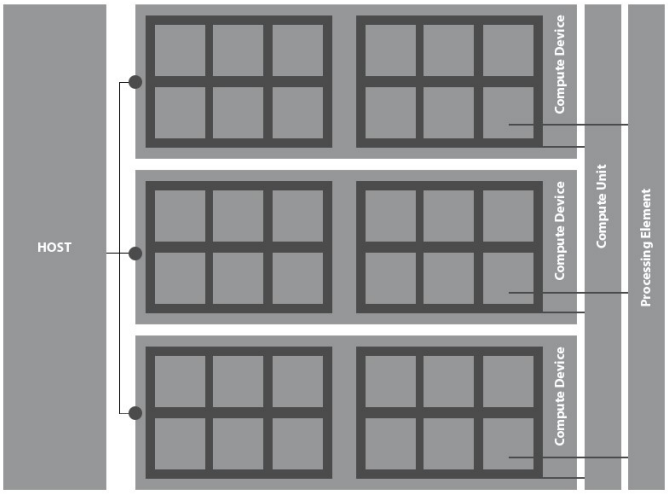


Figure 2: OpenCL platform model

An OpenCL-based application includes two components: kernels, which run on OpenCL devices, and a host program, which runs on the host system and organizes the execution of the kernels (Figure 2). The OpenCL execution model defines each component. Execution of kernels on the device [26, 38, 3, 23]. Before starting the kernel, called index space is defined. An instance of the kernel is executed for each point in a given space and is called a work-item. The corresponding point in index space specifies its global identifier. Each piece of work follows a specific path within the same code, receiving unique data as input. This is how data parallelism is implemented, which is the main use case for OpenCL.

Work items are combined into work-groups, which provide a coarser-grained decomposition of the index space. Work groups are also assigned unique identifiers that have the same dimension as the index space. Within one work group, each item is assigned a unique local identifier. Thus, a work item is defined in two ways: by its global identifier, or by a combination of a local identifier and a work group identifier.

## 6. Parallel algorithms

To implement the sequential algorithm Thomas method was used. Thomas method is a Gaussian exception in the case of a tridiagonal system. The algorithm has two stages, forward elimination and backward substitution. In the first step, we find the coefficients as follows:

$$\alpha_1 = \frac{c_1}{b_1},\ \alpha_i = \frac{c_i}{b_i - \alpha_{i-1}a_i},\ i = 2, 3, \ldots, n$$

$$\boldsymbol{\beta_1 = \frac{d_1}{b_1}, \beta_i = \frac{d_i - \beta_{i-1}a_i}{b_i - \alpha_{i-1}a_i}, i = 2, 3, \ldots, n}$$

At the second stage, all unknowns from the last to the first element are found:

$$\boldsymbol{x_n = \beta_n},\ \boldsymbol{x_i = \beta_i - \alpha_i x_{i+1}},\ \boldsymbol{i = n-1, \ldots, 2, 1}$$

The algorithm is sequential and takes *2n* computation steps, since the computation of $\boldsymbol{\alpha_i}$, $\boldsymbol{\beta_i}$ and $\boldsymbol{x_i}$ depends directly on the result of the previous computation of $\boldsymbol{\alpha_{i-1}}$, $\boldsymbol{\beta_{i-1}}$ and $\boldsymbol{x_{i+1}}$.

To implement the parallel ADI method, CR algorithm was chosen. The CR algorithm consists of two stages: forward reduction and backward substitution. In forward stage, the system is successively reduced to a smaller system with half the number of unknowns until a system of one unknown is reached. In backward stage are successively found remaining unknowns, using the previously obtained values.

At each step of the forward substitution, we update all equations with an even index in parallel with equation *i* of the current system as a linear combination of equations *i*, *i+1* and *i-1*, so that we only display a system of unknowns with even indices. Equation i has the form $\boldsymbol{a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i}$. Updated values of $\boldsymbol{a_i}$, $\boldsymbol{b_i}$, $\boldsymbol{c_i}$ **and** $\boldsymbol{d_i}$ are found by the following formulas:

$$a_i^{'} = -a_{i-1}k_1,\ b_i^{'} = b_i - c_{i-1}k_1 - a_{i+1}k_2$$

$$c_i^{'} = -c_{i+1}k_2,\ d_i^{'} = d_i - d_{i-1}k_1 - d_{i+1}k_2$$

$$k_1 = \frac{a_i}{b_{i-1}},\ k_2 = \frac{c_i}{b_{i+1}}$$

At each step of the backward stage, we simultaneously solve all $\boldsymbol{x_i}$ with odd indices by substituting the solved values $\boldsymbol{x_{i-1}}$ and $\boldsymbol{x_{i+1}}$ in equation *i*:

$$x_i = \frac{d_i^{'} - a_i^{'} x_{i-1} - c_i^{'} x_{i+1}}{b_i^{'}}$$

Note that for simplicity, in the above description, we neglect the special processing of the last equation and the first unknown, respectively, in two phases of the algorithm. In addition, we solve a system of two equations between the two stages of the algorithm. Figure 3 shows the relationship scheme of the algorithm for a system with eight equations.

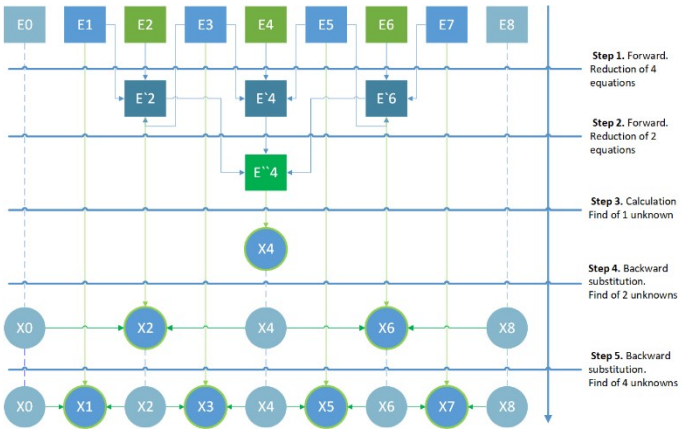


Figure 3: Relationship scheme of the CR algorithm for a system with eight equations

The parallel CR algorithm and the sequential Thomas method perform a series of operations that are linear in the number of unknowns. The Thomas method performs *8n* operations, while CR performs *17n* operations. However, on a parallel computer with *n* processors, CR requires *2log2n* steps, while the Thomas method requires *2n* steps.

For parallel implementation of the ADI method on the GPU, OpenCL and CUDA were used. For all implementations, we copy the data once from the CPU to the GPU and store all the data in the GPU memory until we return the result to the CPU.

Shared memory provides much greater throughput and lower latency than global memory, and therefore it is advisable to preload data from global memory into shared memory. The use of small sizes of shared memory provides higher load and better performance, allowing you to run more threads when shared memory is a limiting factor. Below we describe the kernel implementations for CR algorithm.

First, we set the kernel execution parameters. The number of blocks is equal to the amount of the system, since each individual system is mapped to one block. The number of threads in one block is equal to half the number of equations for each system, since we start by updating only equations with even indices in the first algorithmic step. All variables in memory are stored with single precision (float) with a floating point.

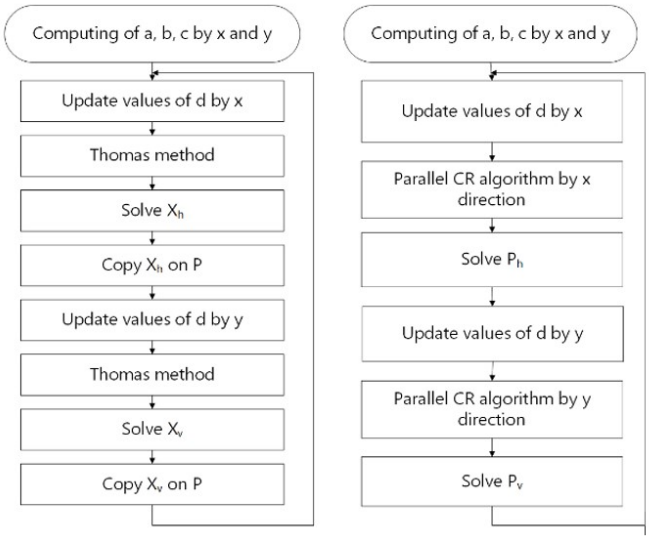


Figure 4: Data flow diagram for implementing the ADI method: a) sequential algorithm b) parallel algorithm

We use five global arrays to store three diagonals of the matrix (*alist*: lower diagonal, *blist*: main diagonal, *clist*: upper diagonal), one right side of the *dlist* and one solution vector *xlist*. The amount of shared memory that we allocate for each block or each three-diagonal system is equal to *system_size * 5 * sizeof (float)* bytes.

The forward stage executes *log2(system_size/2)* algorithmic steps. All five arrays *alist, blist, clist, dlist* and *xlist* are shared memory arrays with data loaded from global memory. The step begins with 1 and doubles the value of each algorithmic step. The number of active threads decreases by half at each algorithmic step.

At the beginning of the backward stage, we solve the system of equations with two equations obtained at the final stage of the forward stage. Then we execute the *log2(system_size/2)* algorithmic steps to solve all the other unknowns. According to the algorithm, we double the number of active threads at each algorithmic step.

The general scheme of the sequential ADI method is shown in Figure 4 (a). All arrays are linearly stored in physical memory in both horizontal and vertical directions. The elements of $P$ are updated in each direction, and it is immediately used to compute the new linearly stored right term in another direction, so the result of the linear solution $x$ from Thomas method is mapped back to the two-

dimensional array $P$. Since arrays are stored urgently in a C/C++ program, this strategy is good for computing the vertical right element, but not so good for doing it in the horizontal direction.

On personal computers (PCs) with graphics accelerators, where the computing parts of the program are uploaded to the GPU via the PCI Express bus, the program must minimize data transfer between the host and the device. Therefore, the best scenario is that data is copied to the device only once at the beginning and saved until the end of the computing. The general scheme of the parallel ADI method implementation on the GPU is shown in Figure 4 (b).

## 7. Performance tests

For testing the parallel ADI method using the two-dimensional multiphase filtration problem of a multicomponent fluid in porous media at different grid sizes. To test the program, we used a PC with an eight-core Intel Core i7-3770 processor with a clock frequency of 3.4 GHz and 8 GB of memory, Nvidia RTX 2080 graphic card with 8 GB of video memory, and Windows 10 operating system. To implement the parallel and sequential algorithm, C++ programming language was used. The task parameters are set as follows: the grid size ($N$) is uniform in both directions with $h = \Delta x = \Delta y = 1 / (N\text{-}1)$, and the numerical time step $\Delta t$ is 0.001, and *time* is 0.1, therefore – the number of iterations over time is 100. The size of the grid must be equal to a power of two plus one. The size of the block depends on the size of $N$ and the GPU compute capability, because of this there is a limit on the size of the grid. When measuring performance, computing time was used instead of floating-point operations per second (FLOPS), since different parallel algorithms have different numbers of FLOPS. The test results in Figure 5 shows that the OpenCL program is 16-25 times faster than a sequential program. Although the OpenCL program is inferior in performance to the CUDA program, it is not tied to NVIDIA processors.

*Table 2: Computing time in seconds*

| | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|
| Sequential algorithm | 0,352 | 1,131 | 3,472 | 12,165 | 45,818 |
| OpenCL algorithm | 0,018 | 0,041 | 0,138 | 0,515 | 2,778 |
| CUDA algorithm | 0,013 | 0,031 | 0,091 | 0,323 | 1,600 |

As a result, we compared the computation time of three programs: a sequential program implemented using the Thomas method and parallel programs implemented on OpenCL and CUDA (Table 2). Testing results shows that parallel programs running on a GPU are 16-37 times faster than a sequential program (Figure 5).

*Table 3: Speedup of parallel algorithms*

| | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|
| OpenCL | 19,556 | 27,585 | 25,159 | 23,621 | 16,493 |
| CUDA | 27,077 | 36,484 | 38,154 | 37,663 | 28,636 |

From Table 3 we can see that a parallel CUDA program performs faster than a parallel program implemented on OpenCL. Figure 6 shows that the speedup by increasing the mesh size decreases. This is due to the overhead in synchronizing data between the host and device. In the parallel program on the device, memory is allocated for *51* two-dimensional arrays of the float type. The

approximate required memory for these arrays is *51 * system_size * system_size * sizeof (float)*. This parameter must be considered before each start.

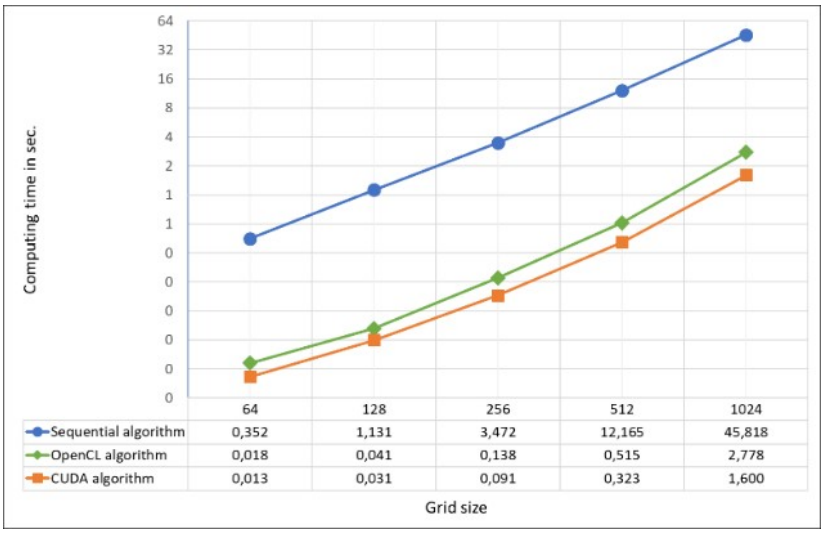


Figure 5: Comparative graph of computing time results on various grid sizes

Given the above limitations, we need to implement the distribution of the data of one system into several blocks to remove the restriction of the dimension of the computational grid, test this parallel program on mobile devices, develop an algorithm for running parallel programs on multiple GPUs.

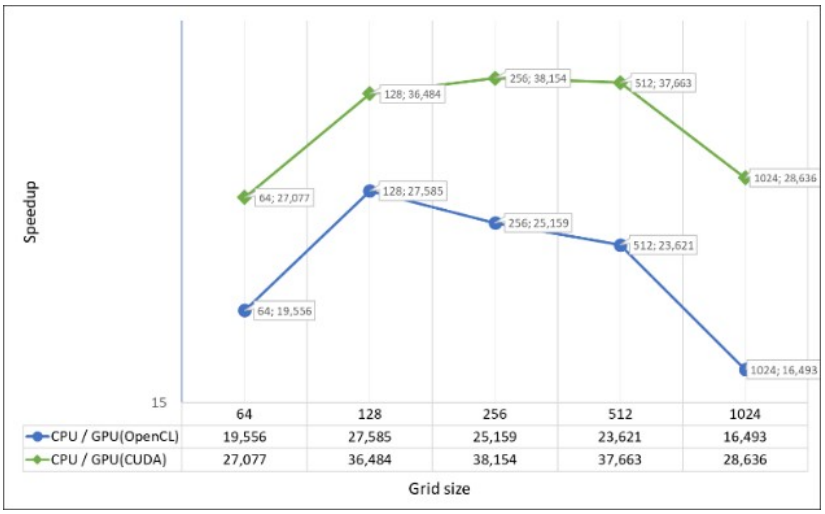


Figure 6: Speedup of parallel algorithms (CUDA and OpenCL) on various grid sizes

## 8. Conclusions

As a result of this work, the parallel ADI method was developed to solve the multiphase, multicomponent fluid flow problem in porous media on modern GPU. When implementing the ADI method, for the parallel solving tridiagonal equations systems in each direction, the cyclic reduction method using the CUDA and OpenCL were used. To achieve effective acceleration, a reduced shared memory size in each computing block was used. The test results of the program showed that the OpenCL program runs 16-25 times and on CUDA 26-37 times faster than the sequential program.

Due to this acceleration and heterogeneity of OpenCL, the developed program runs on various devices (mobile devices, FPGA and etc.). All algorithms have memory limitations. Given these limitations, the next step in the work is:

- implementation of the distribution of data from one system to several CUDA blocks to remove the restrictions on the dimension of the computational grid
- testing developed algorithms on mobile devices, FPGA and etc.
- implementation of hybrid algorithms for solving the multiphase, multicomponent fluid flow problem in porous media
- parallel algorithm development on multi-GPU.

## 9. Acknowledgements

This work was carried out as part of project No. AP05130366 "Development of an intelligent high-performance information system for the analysis of oil recovery enhancement technologies iFields-II" due to grant funding from the Ministry of Education and Science of the Republic of Kazakhstan.

## References

[1] D.Zh. Akhmed-Zaki, B.S. Daribayev, T.S. Imankulov and O.N. Turar, *High-performance computing of oil recovery problem on a mobile platform using CUDA technology*, Eurasian J. Math. Comput. Appl. 5 (2017) 4–13.

[2] D.Zh. Akhmed-Zaki, T.S. Imankulov, B. Matkerim, B.S. Daribayev, K.A. Aidarov and O.N. Turar, *Large-scale simulation of oil recovery by surfactant-polymer flooding*, Eurasian J. Math. Comput. Appl. 4 (2016) 12–31.

[3] R. Banger and K. Bhattacharyya, *OpenCL Programming by Example*, Packt Publishing, 2013.

[4] J. Bear, *Dynamics of Fluids in Porous Media*, New York: Dover, 1972.

[5] V. Casulli and R. Cheng, *Semi-implicit finite difference methods for three-dimensional shallow water flow*, Int. J. Numerical Meth. Fluids 15 (2005) 629–648.

[6] Z. Chen, *Reservoir Simulation: Mathematical Techniques in Oil Recovery*, Society for Industrial and Applied Mathematics, 2007.

[7] J. Cheng, M. Grossman and T. McKercher, *Professional CUDA C Programming*, Wrox, 1st edition, 2014.

[8] Sh. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (Applications of GPU Computing)*, 1st edition, Morgan Kaufmann, 2012.

[9] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (Applications of GPU Computing)*, Morgan Kaufmann, 1st edition, 2012.

[10] A. Davidson and J. Owens, *Register packing for cyclic reduction: a case study*, Proc. Fourth Workshop on General Purpose Processing on Graphics Processing Units, ACM, 2011.

[11] A. Davidson, Y. Zhang and J. Owens, *An auto-tuned method for solving large tridiagonal systems on the GPU*, Parallel & Distributed Processing Symposium (IPDPS), IEEE Int. (2011) 956–965.

[12] N. Ellner and E. Wachspress, *Alternating direction implicit iteration for systems with complex spectra*, SIAM J. Numerical Anal. 28 (1991) 859–870.

[13] D. Goddeke and R. Strzodka, *Cyclic reduction tridiagonal solvers on GPUs applied to mixed-precision multigrid*, Parallel Distributed Syst. IEEE Trans. 22(1) (2011) 22–32.

[14] J. Han and B. Sharma, *Learn CUDA Programming: A beginner's guide to GPU programming and parallel computing with CUDA 10.x and C/C++*, Packt Publishing, 2019.

[15] N. Hecquet, *Parallel Calculus in CUDA: Heat equation, Black and Scholes Model*, Éditions universitaires européennes, 2019.

[16] T.S. Imankulov, D.Zh. Akhmed-Zaki, B.S. Daribayev and et al., *Intellectual system for analysing thermal compositional modelling with chemical reactions*, 16th European Conference on the Mathematics of Oil Recovery, 2018.

[17] T.S. Imankulov, D.Zh. Akhmed-Zaki, B.S. Daribayev and O.N. Turar, *HPC Mobile Platform for Solving Oil Recovery Problem*, Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2016). 2 (2016) 595–598.

[18] B. Jang, D. Kaeli, S. Do, H. Pien, *Multi GPU implementation of iterative tomographic reconstruction algorithms*, IEEE Int. Symp. Biomedical Imaging: From Nano to Macro. (2009) 185–188.

[19] D.R. Kaeli, P. Mistry, D. Schaa, D. Zhang, *Heterogeneous Computing with OpenCL 2.0*, 3rd Edition, Morgan Kaufmann, 2015.
[20] M. Kass, A. Lefohn, J.D. Owens, *Interactive depth of field using simulated diffusion*, Technical Report 06-01, Pixar Animation Studios, 2006.
[21] M. Kass, G. Miller, *Rapid, stable fluid dynamics for computer graphics*, Computer Graphics (Proceedings of SIGGRAPH 90), (1990) 49—57.
[22] H. Kim, S. Wu, L. Chang, W. Hwu, *A scalable tridiagonal solver for GPUs*, Parallel Processing (ICPP), Int. Conf. IEEE. (2011) 444–453.
[23] J. Kowalik, T. Puzniakowski, *Using OpenCL: Programming Massively Parallel Computers (Advances in Parallel Computing)*, IOS Press, Har/Cdr edition, 2012.
[24] I. Lindemuth, J. Killeen, *Alternating direction implicit techniques for two-dimensional magnetohydrodynamic calculations*, Journal of Computational Physics. 13 (1972) 181–208.
[25] J.J.Martin, *Cubic Equations of State*, EC Fundamentals, 1979.
[26] A. Munshi, *OpenCL Programming Guide*, Addison-Wesley Professional, 1st edition, 2011.
[27] T. Namiki, *A new FDTD algorithm based on alternating-direction implicit method*, Microwave Theory and Techniques, IEEE Transactions on 47. (1999) 2003–2007.
[28] D. Peaceman and H. Rachford, *The numerical solution of parabolic and elliptic differential equations*, Journal of the Society for Industrial & Applied Mathematics. 3 (1955) 28–41.
[29] K.S.Pederson and P.L. Christensen, *Phase Behavior of Petroleum Reservoir Fluids*, CRC Press, 2008.
[30] D.Y. Peng and D.B. Robinson, *A new two-constant equation of state*, Industrial and Engineering Chemistry Fundamentals. 15 (1976) 59—64.
[31] R.C. Reid, J.M. Prausnitz and T.K. Sherwood, *The Properties of Gases and Liquids 3rd edition*, New York: McGraw-Hill, 1977.
[32] N. Sakharnykh, *Effcient tridiagonal solvers for ADI methods and fluid simulation*, NVIDIA GPU Technology Conference, 2010.
[33] N. Sakharnykh, *Tridiagonal solvers on the GPU and applications to fluid simulation*, GPU Technology Conference, 2009.
[34] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional, 1st edition, 2010.
[35] S. Sengupta, M. Harris, Y. Zhang and J.D. Owens, *Scan primitives for GPU computing*, Graphics Hardware, ACM, New York. (2007) 97-–106.
[36] G. Soave, *Equilibrium constants from a modified Redlich-Kwong equation of state*, Chem. Eng. Sci. 27 (1972) 1197—1203.
[37] D. Storti and M. Yurtoglu, *CUDA for Engineers: An Introduction to High-Performance Parallel Computing*, Addison-Wesley Professional, 1st edition, 2015.
[38] R. Tay, *OpenCL Parallel Programming Development Cookbook*, Packt Publishing. 2013.
[39] L.H. Thomas, *Elliptic Problems in Linear Differential Equations over a Network*, Watson Sci. Comput. Lab Report. 1949.
[40] W. Wu, *Computational river dynamics*, CRC. 2007.
[41] Y. Zhang, J. Cohen, A. Davidson and J. Owens, *A hybrid method for solving tridiagonal systems on the GPU*, GPU Computing Gems Jade Edition, 2011.
[42] Y. Zhang, J. Cohen and J. Owens, *Fast tridiagonal solvers on the GPU*, ACM Sigplan Notices 45 (2010) 127–136.
[43] Y. Zhang and Y. Jia, *Parallelization of implicit CCHE2D model using CUDA programming techniques*, World Envir. Water Resour. Cong. (2013) 1777–1792.
[44] Y. Zhang and Y. Jia, *Parallelized CCHE2D model with CUDA Fortran on graphics process units*, Comput. Fluids 2013 (2013) 359–368.
[45] D. Zudkevitch and J. Joffe, *Correlation and prediction of vapor-liquid equilibria with the Redlich-Kwong equation of state*, Amer. Inst. Chem. Engin. J. 16 (1970) 112—199.