**I.K. Beisembetov, T.T. Bekibaev, B.K. Assilbekov,
U.K. Zhapbasbayev, B.K. Kenzhaliev**

APPLICATION OF GPU IN THE DEVELOPMENT OF 3D HYDRODYNAMICS SIMULATORS FOR OIL RECOVERY PREDICTION

1. INTRODUCTION

The efficiency of hydrocarbons recovery is one of the major objectives of Petroleum companies during the fields development. Great number of studies, tests and computer calculations required before the wells will be drilled and development scheme will be chosen. Computer model of hydrocarbon field is created and analyzed by modern software before and during the field exploitation. This work made routinely by the most of developers. Rule of thumb: oil can be recovered only once, but simulation can be made infinitely. Thus, software that calculate different scenarios of field development (new wells, capital investments, etc.) play important role in Petroleum Industry. One of the hydrodynamic modeling extensions is rapid calculations of recovery forecasting and history matching. Increasing number of cells in the simulation grid significantly slow down the calculations. Fast simulation on commercial software (Eclipse (Schlumberger), Tempest MORE (Roxar), VIP (Landmark), tNavigator (RFDynamics), etc.) is based on parallel computing on CPU cores of HPC (High Performance Computers) and PC using MPI (Message Passing Interface) and OpenMP (Open Multi-Processing). This paper reviews the parallel computing of hydrodynamics on GPU (Graphical Processing Unit) of PC. It is one of main differences between our 3D hydrodynamic simulator and existing analogues.

A great while, GPU was used with CPU only for their designated purpose – accelerate the calculations of graphics. But with increasing computation power of GPU, developers tend to apply it for CPU calculations. Nowadays, libraries that allow running the applications on the graphical processors have been developed, and the concept of GPU usage in calculations was widely spreaded in various areas of science, from astrophysics to financial risks calculations [1–5].

Modern GPU consists of hundreds of processing cores and in some instances overcome the calculation power of CPU hundred times. These works research the calculation capabilities of GPU in oil recovery problems.

2. MATHEMATICAL MODEL

This paper reviews the hydrodynamic modeling of oil reservoir with pressure above bubble point pressure eliminating the gas from liberation. Temperature influence wasn't noticed in calculations. Thus, fluid flow in porous media could be expressed in following equations [6–8]:

$$\nabla \cdot \left[\lambda_o K \left(\nabla p_o - \gamma_o \nabla z \right) \right] = \frac{\partial}{\partial t} \left[\frac{\phi (1 - S_w)}{B_o} \right] + q_o \tag{1}$$

$$\nabla \cdot \left[\lambda_w K \left(\nabla p_o - \nabla p_{ow} - \gamma_w \nabla z \right) \right] = \frac{\partial}{\partial t} \left[\frac{\phi S_w}{B_w} \right] + q_w \tag{2}$$

where:

K – absolute permeability,
λ_o, λ_w – mobility of oil and water phases,
ϕ – porosity,
S_w – water saturation,
p_o – oil phase pressure,
q_o, q_w – oil and water sink/sources,
B_o, B_w – oil and water formation volume factor,
z – depth,
γ_o, γ_w – oil and water specific gravity,
t – time,
p_{ow} – capillary pressure.

Equations (1) and (2) are solved using no flux boundary conditions.

3. NUMERICAL TECHNIQUES

Equations (1)–(2) is discretized using IMPES method [6–8]. Values of pressure are calculated by SOR method. Water saturation field is defined explicitly after pressure is calculated. Finally, capillary pressure and transmissibility between neighbor blocks are computed and applied for next time step.

4. PROGRAM REALIZATION ON GPU WITH CUDA

In order to use the calculation capabilities of GPU effectively, it is necessary to find the most time-consuming parts of code and adopt them by CUDA technology. Most of time in our program is spent in jobs associated with 3D arrays that have significant size. That work with cells of various 3D arrays was paralleled on GPU. On the figure below (see Fig. 1), flowchart of CUDA-based program realization is shown.

CheckSchTime() – procedure, that inspects work of wells (user-defined schedules and modes). If condition of well work switching occurs, GPU data relating to wells will update.

PrePoissonKernel() – procedure that runs on the GPU (CUDA kernel). It computes coefficients needed to solve the equation of pressure, also calculates necessary data for time step.

PoissonIterationKernel() – kernel, that executes the next iteration of the Poisson, and determines the residual for each thread block. The procedure determines the value of the pressure in each cell of model.

SaturatonCalcKernel() – kernel, that computes the values of water saturation in each cell, using the new values of pressure. Also the necessary parameters for well (flow rate, bottom hole pressure, water cut, etc.) are defined.

GetStratumDatas() – procedure, that's extract the field distribution of the saturation, pressure, temperature from the DRAM of video card. It takes considerable time and is performed only at user-defined time steps.

Besides, in PrePoissonKernel() и PoissonIterationKernel() parallel reduction is implemented in order to compute values needed for time step and residual for each thread block. The final reduction with relation to the blocks is calculated sequentially on the CPU.

One of key differences between GPU and CPU is memory organization and its application. It is known, CUDA architecture has complex memory structure. In order to achieve the highest acceleration, it is necessary to accurately distribute data on various memory types. A part of data, that occupy a large amount of memory, is stored as arrays in global memory of the GPU. Each cell in the array (I,J,K) stores the value for the corresponding cell of the hydrodynamic model of the reservoir. This data are listed below:

- Main data – pressure, water saturation. They changes at each time step.
- The coefficients for the Poisson's equation. Computed in the procedure PrePoissonKernel() and used to determine the pressure at each iteration PoissonIterationKernel().
- Immutable data – depth, the transmissibility of cells.

PVT and saturation function tables occupies a small volume and stored in constant memory for quick access. These tables are used repeatedly to determine the pressure and saturation for each computational cell of the reservoir.

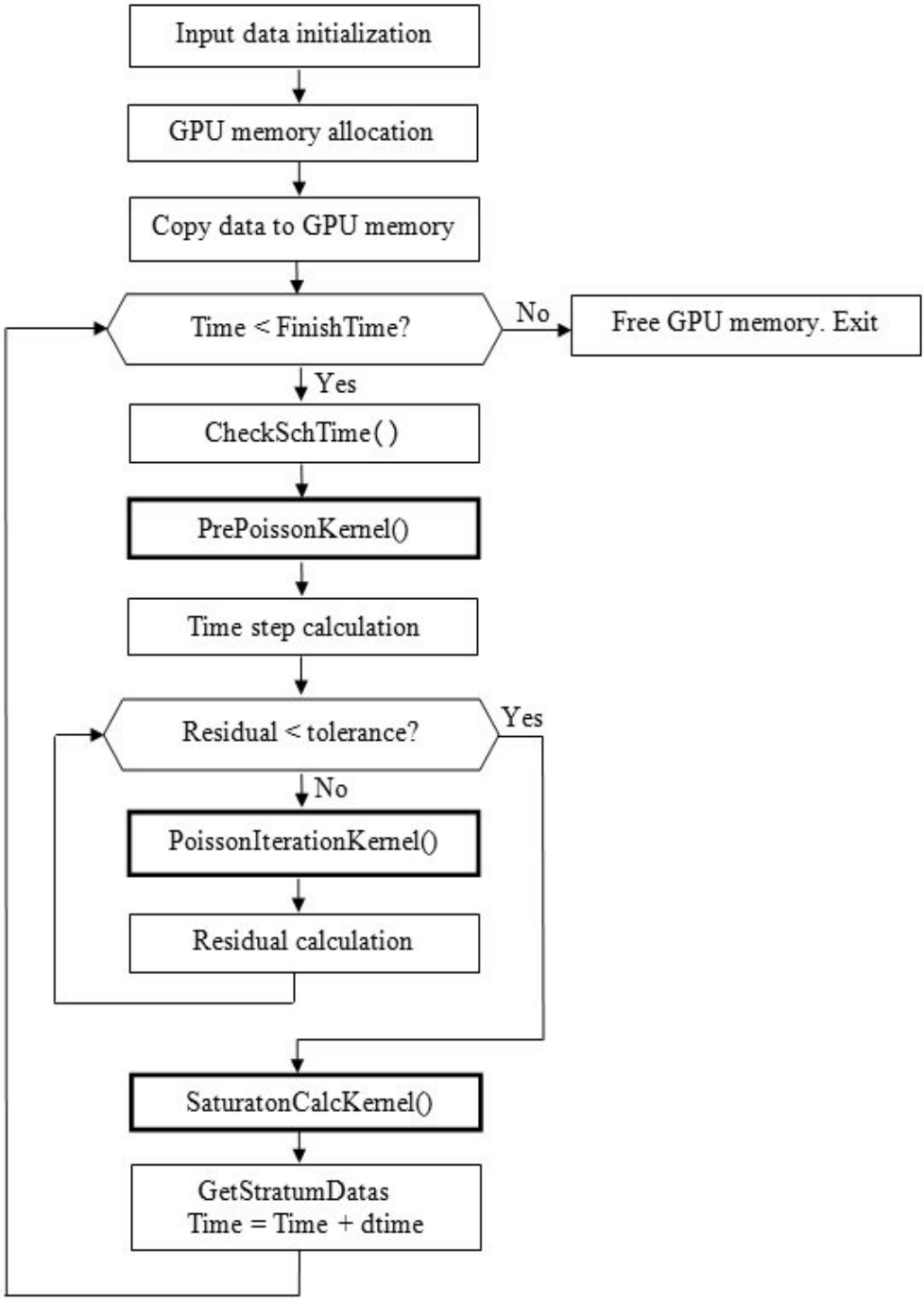

Fig. 1. Flowchart representing a program realization using CUDA

Well data include the values of flow rates of fluids and liquids, water cut, bottomhole pressures for each well. The data stored in the page-locked host memory to read it in CPU procedure and to write in GPU kernel.

Data of well management include the type and value of work mode, also data of the well perforation. It stored in constant memory. At change of a well mode, values of data are updated in GPU memory from CPU procedure.

Shared memory is used to store intermediate variables in the kernel. And also, thread blocks load a part of data from global to shared memory for faster use in the current kernel.

One of important aspects of algorithm creation was the choice of division of settlement area between threads. For this purpose, it was necessary to define dimension and the sizes of the block of thread. There was a choice between three-dimensional and two-dimensional division grids (Fig. 2). At three-dimensional division one thread processes one cell of model, at two-dimensional – a column of cells along an axis Z. Indexes of cells (I,J,K) are calculated concerning index of the block and index of thread.

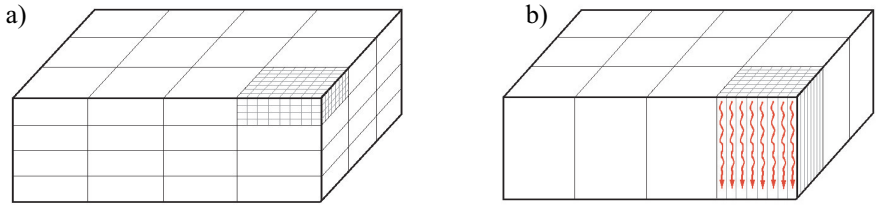

Fig. 2. Splitting of computational cells into thread blocks: a) 3D case; b) 2D case

At calculation of each value cell, values of neighbor cells (Fig. 3a) are necessary. For example, in iterative formula of the Poisson's equation data access looks like:

$$p_{ijk}^{l+1} = Ap_{i+1\,jk}^{l} + Bp_{i-1\,jk}^{l+1} + Cp_{ij+1k}^{l} + Dp_{ij-1k}^{l-1} + Ep_{ijk+1}^{l} + Fp_{ijk-1}^{l+1} + RHS.$$

Because of data access features, in order to calculate the two-dimensional region of size NX×NY it is necessary to download NX*NY + 2*NX + 2*NY values (Fig. 3b), for three-dimensional region of size NX×NY×NZ – NX*NY*NZ + 2*NX*NY + 2*NX*NZ + 2*NY*NZ values (Fig. 3c).

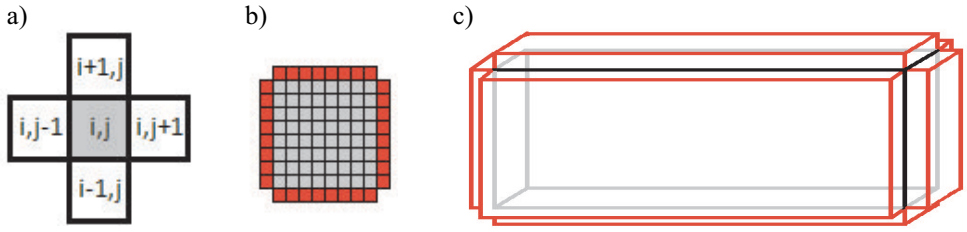

Fig. 3. Additional cells (red) needed to load for thread block

Obvious at first sight, the three-dimensional configuration (Fig. 2a) was less effective. This configuration does not allow to contain in block a large number of cells (the restriction on the number of threads), which leads to the small size of the 3D thread block and, consequently, a greater ratio of „side" loads from global memory. For example, if the block has size of 8×8×8, „side" loading will be $\frac{6*8*8}{8*8*8} = 66.66\%$. In contrast with this, in the two-dimensional block 16×16 (Fig. 3b) such loadings occupy $\frac{4*16}{16*16} = 25\%$, and in case $32\times16 - \frac{2*32+2*16}{32*16} = 18.75\%$. Extra „side" loadings – time cost access to global memory and branching in a code at which one threads load „side" parts and others stand idle. Besides, such two-dimensional splitting gives the chance to use effectively the shared memory. Therefore application of the two-dimensional splitting is more preferable.

Comparing thread blocks of size 16×16, 32×16 and 32×8, it should be noted that at global memory allocation it is necessary to make array padding to satisfy the coalescing condition (association of transaction to memory) when half-warps access to DRAM. 32×8 or 32×8 tiling requires more memory offset than 16×16 tiling. Because of the smaller sizes, the thread block 16×16 is better scaled under the various sizes of a grid of model along an axis X (or Y). However, the blocks 32×16 or 32×8 are more suitable for video cards with compute capability 2.x because in this architecture a warp (not a half-warp) request to global memory, i.e. access takes one transaction, in contrast of this, warp in 16×16 tiling needs 2 transactions.

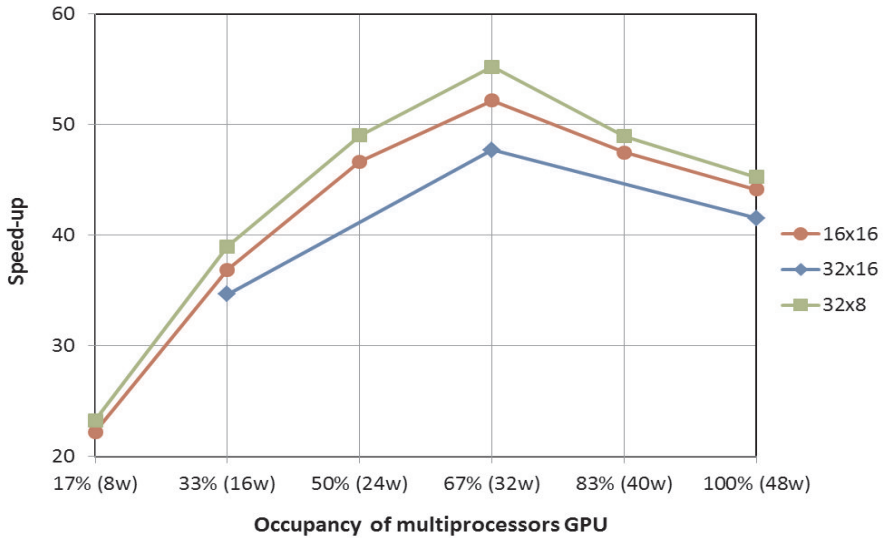

Fig. 4. Acceleration at various occupancy of SM and sizes of thread block

Also, it is necessary to take into account the occupancy of streaming multiprocessor (SM). It means number of thread blocks, that one SM can execute, and depends on number of active warps in SM. In this work it was experimentally established that on computing capability 2.x GPU the program gives the highest acceleration when from 48 warps (which can be executed on one SM), are active only 32 warps, in other words optimal occupancy is 67% for various block sizes (Fig. 4). In present algorithm, maximum occupancy of SM is

not effective. This is due to best latency hiding in process of warp access to global memory and a large number of registers on a thread for intermediate calculations by contrast to the maximum occupancy.

5. NUMERICAL EXPERIMENTS. RESULTS AND DISCUSSIONS

In case 1, calculations took place on the field test model and was aimed to find the influence of cells distribution on acceleration (various cells number or NX, NY, NZ proportion by X, Y, Z axis at constant total number of cells, „grid size ratios"). In case 2, calculations was made on East Moldabek field model. Below, models descriptions are presented.

Calculations took place on Intel Core i7 2600 CPU (3.40 GHz, DRAM Frequency 667 MHz) and GeForce GTX 560 GPU (memory clock – 2004 MHz, processor clock – 1620 MHz, compute capability – 2.1, number of SMs – 7 with 336 cores).

Case 1. Homogenous cells with permeability of 2900 mD and 33% porosity. Initial pore pressure – 100 bars, water saturation – 20%. Heterogeneity of characteristics through reservoir space does not affect on software effectiveness. Model has 2 wells: 1 production well and 1 injection well, located on opposite corners. (see Fig. 5).

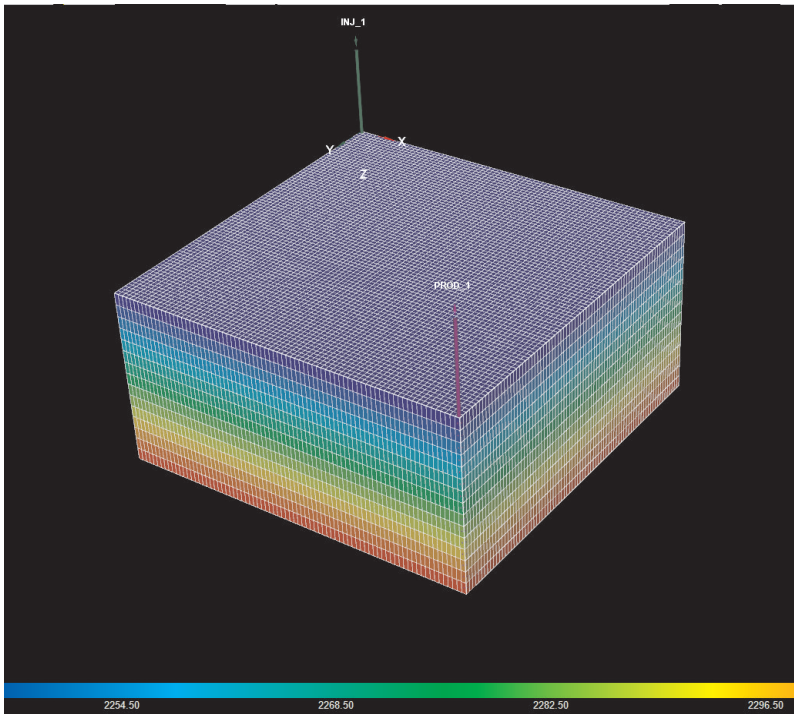

Fig. 5. Test model for case 1

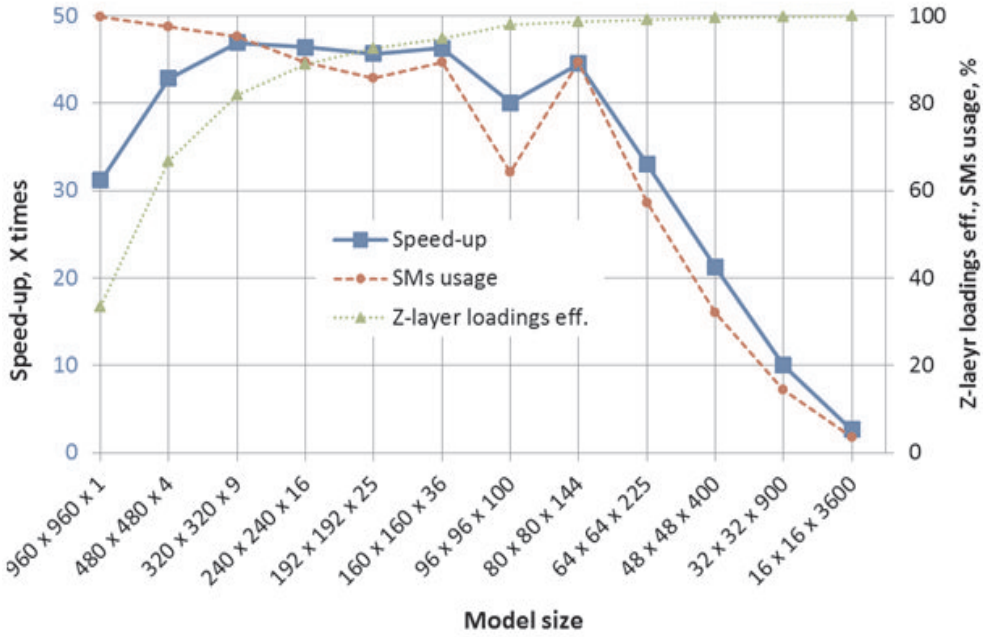

Fig. 6. The dependence of the acceleration and „grid size ratios"

On the figure (Fig. 6), relationship between NX, NY, NZ proportion at constant cells number and calculations acceleration is shown. Thread block of 16x16 was used. In all experiments, total number of cells was 921 600. Comparison made at NZ values of 1 to 3600 and NX*NY*NZ = 921 600.

Two factors influence on acceleration at this experiments.

First, it is usage of streaming multiprocessors, i.e. distribution of thread blocks to all SMs. Our GPU has 7 SM, which one can execute 4 thread blocks, i.e. for one certain step they can process 28 blocks. Therefore, it is important, that in splitting of model cells into thread blocks its number will be divisible by 28. For example, the model of size NX = 96, NY = 96 splits into 36 blocks 16×16, and at such block number on the first step all of the 7th multiprocessor will work, counting 28 blocks, on the second step the remained 8 blocks will be computed by only 2 SMs, and the others 5 SMs will stand idle. In other words, SMs usage will be $\frac{36}{2*28} = 64\%$. Obviously, the usage is more, so the acceleration on GPU is higher. If the number of blocks is much less 28, only few SMs will be used on the first step, i.e. the conveyor from 7 SMs will work not for full efficiency. And as a result, speed-up will dramatically decrease at small NX and NY values. It should be noted that divisibility on 28 plays of block number is important only at small NX, NY.

At great values of NX, NY (i.e. at a large number of blocks) divisibility on 28 is not so important and does not essentially influence on acceleration because there will be a set of steps of processing of 28 blocks, and unoccupied SMs can remain only on the last step, and as a whole use of all multiprocessors will tend to 100%.

The second factor affecting the speed-up is related to the NZ. The matter is that for calculation values on a layer K (by Z-axis), values on layers K-1 and K+1 are needed, additional layers above and below model are entered for this purpose. When next layer starts to be computed in GPU, layer K+1 becomes K, and previous layer K becomes K-1, the values are stored on shared memory. Therefore, each time to calculate new layer, there is no need to load three layers, it is enough to load only K+1, and the others already stored from last iterations by K. As a result, only NZ+2 layers are loaded from global memory to calculate NZ layers. It is known that, accesses to DRAM are very slowly carried out on GPU. Therefore the number of transaction to DRAM is critical. At small values NZ, amount of „superfluous" transactions essentially in comparison with NZ. For example, for calculation model with NZ=1, loading of 3 layers is required, for NZ = 2 – 4 layers, and for NZ = 100 – 102 layers. Ratios of the calculated layers number to loaded are 33%, 50% and 98% accordingly. Obviously, if this indicator is higher, the acceleration will be higher. It explains less speed-up of GPU calculation of models with small NZ.

So, at identical number of grid cells, model with the balanced sizes NX, NY, NZ has the highest speed-up on GPU program. Acceleration considerably decreases, if NX, NY is too small.

Figure 7 shows a dependence of the acceleration and total number of grid cells in hydrodynamic model, where $NX = NY\frac{16}{3}NZ$. In this case total number of cells changed from 96000 to 6144000 and thread block 16×16 was used.

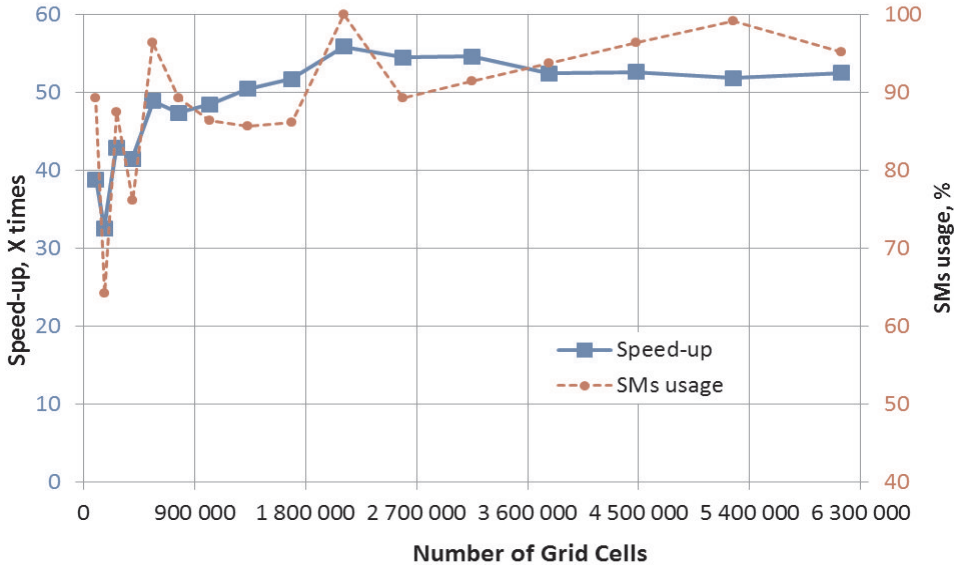

Fig. 7. Dependence of the acceleration and total number of grid cells

Figure shows, with increase in total number of cells acceleration grows, after 2 million cells acceleration is stabilized and equal to 52–55 times. Figure oscillation, especially in the beginning, can be explained by various coefficient of SMs usage that was discussed before.

Small speed-up at low amount of cells is explained by smaller fraction of parallel part of program comparing with the sequential part. (Parallel part: various values calculation of model 3D grid; sequential: well mode recalculation and limits verification, kernel launch, results output).

Maximum speed-up of 60.4 times was taken on grid of 2 million and more cells, using the thread block 32×8.

It is necessary to mention those factors which negatively affect acceleration at the current implementation of the program:

1) Divisibility of NX, NY by sizes of thread blocks (16×16, 32×16, 32×8). If NX or NY is divisible by block size, NX and NY will be rounded up, amount of block will increase and in some boundary blocks threads will be idle. For example, model with NX×NY=48×48 has 9 blocks, model with NX×NY = 48×49 has 12 blocks (such as 48×64). However, for large NX, NY is becoming less important.
2) Amount of wells. Wells increase time cost access to DRAM (some well parameters stored in global memory) and branching in a code. Branching cannot be effectively paralyzed in GPU.
3) Existence of inactive cells. The block, which has inactive cells, is computed for the same time, as well as the block, which has all active cells. Whereas in consecutive realization, inactive cells reduce number of iteration at looping on a three-dimensional grid that reduces time of consecutive calculation.

Case 2. In this case there is considered field model East Moldabek (see Fig. 8). The model consists of 91476 (36×77×33) grid cells. Formation volume factor B_w of water, water viscosity μ_w and porosity ϕ relations from pressure are set up in following equations for East Moldabek field:

$$B_w = 1.02\left[1 + c_w(p_w - 20) + 0.5c_w^2(p_w - 20)^2\right]^{-1} \tag{3}$$

$$B_w\mu_w = 1.029\left[1 + c_\mu(p_w - 20) + 0.5c_\mu^2(p_w - 20)^2\right]^{-1} \tag{4}$$

$$\phi = \phi_o\left[1 + c_\phi(p_o - 39) + 0.5c_\phi^2(p_o - 39)^2\right] \tag{5}$$

where:

B_w = barsa^{-1},
c_μ = barsa^{-1},
c_ϕ = barsa^{-1}.

East Moldabek field oil and water density at standard conditions are 889.5 and 1000.1 kg/m^3 respectively. Horizontal permeability varies from 503.2 mD to 4240.1 mD, while vertical permeability ranges are 53 mD and 420 mD. Porosity changes in range of 17–40%.

The Figure 9 shows the initial water saturation distribution. Dependence of relative phase permeability and capillary pressure from water saturation is represented in Figure 10. Figure 11 shows the oil PVT properties.

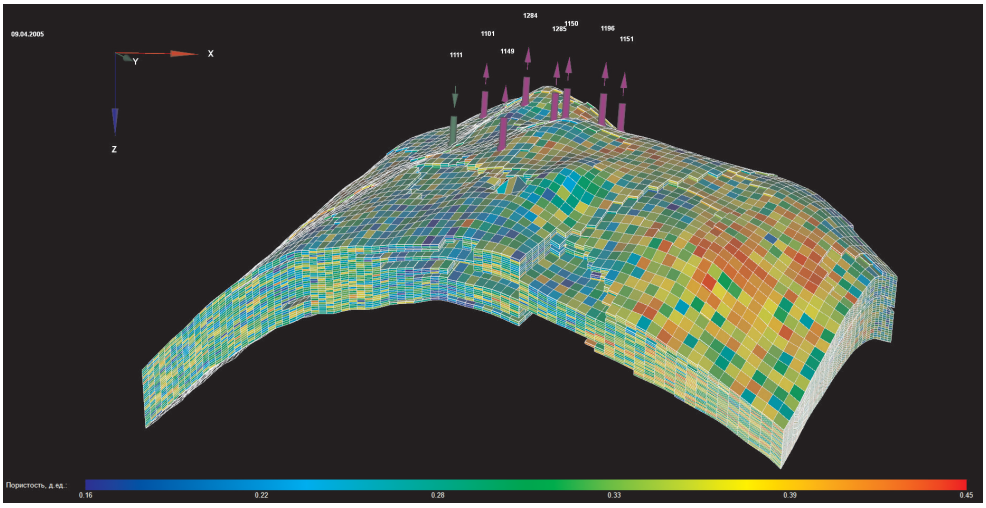

Fig. 8. Porosity distribution of East 1Moldabek field

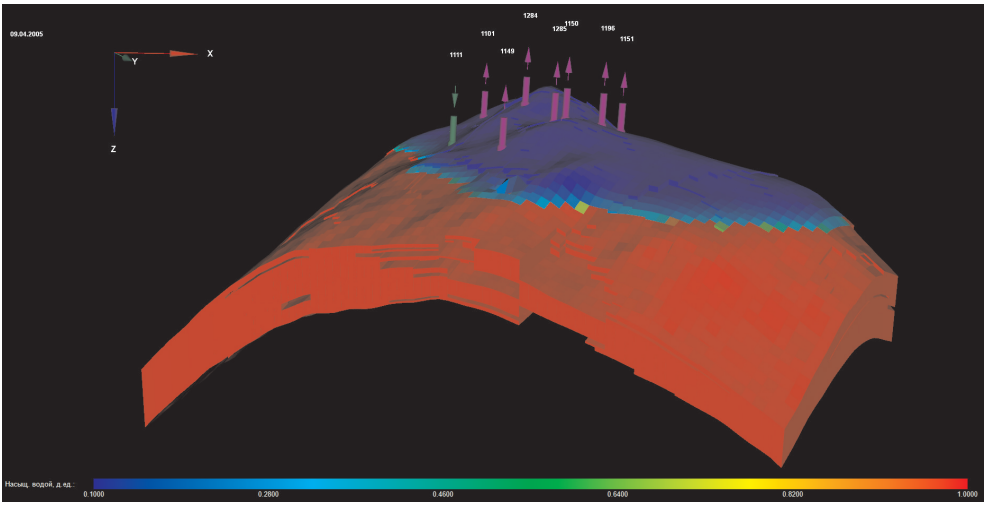

Fig. 9. Initial water saturation distribution of East Moldabek field

The carried out calculation consisted of 100 iterations on time that totally composes 450 days of forecast. Operating time of the parallel version of the program is 5.3 sec., and gave acceleration in 33 times in comparison of the unparallel version of the program.

Calculations results of two versions were completely identical. Smaller acceleration in comparison with the maximum achieved acceleration in 60.4 times, is explained by the small size of model and as a result, an underloading of all sreaming multiproccers of GPU.

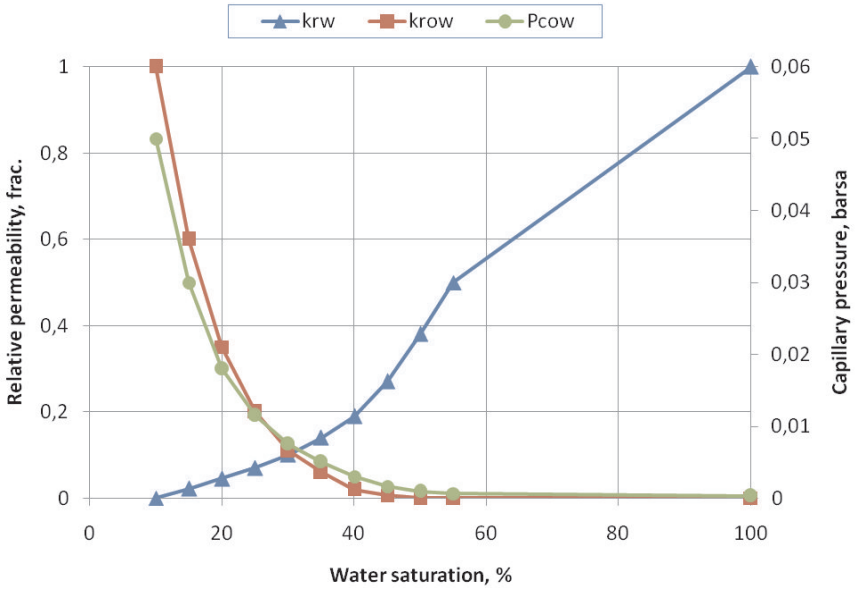

Fig. 10. Relative permeabilities and capillary pressure for case 2

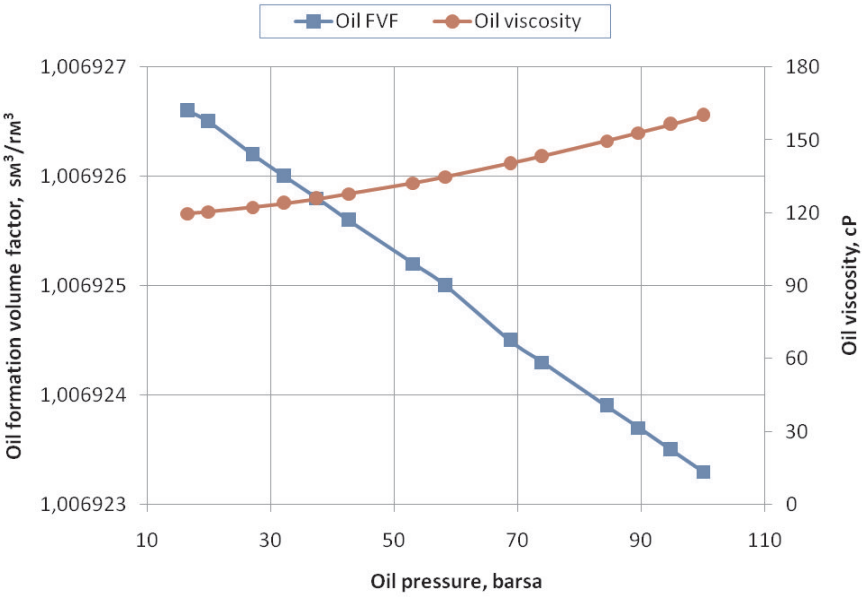

Fig. 11. Oil PVT properties for case 2

6. CONCLUSIONS

In this work transposition and optimization of the program of hydrodynamic modeling from the consecutive version to parallel version on GPU using CUDA technology is carried out. For parallel calculations running video card GeForce GTX 560 was used, consecutive calculations were spent on the central processor Intel Core i7-2600.

Process of transposition consisted of separation of algorithm parts which can be carried out with the maximum degree of parallelism, and rewriting these parts for GPU kernels. For these purposes there are chosen parts of a code, which are related to calculations of all cells values in a three-dimensional grid of model, because computation this cells occupied the main part of time in the consecutive program, especially at the big sized model. Input data of calculation were divided into various classes of CUDA memory, in dependence of their volume, frequency and need of use. All model cells have been split into two-dimensional regions of thread blocks, each thread of which calculate cells with indexes from (I,J,1) to (I,J,NZ). Two-dimensional model splitting showed up more effectively as against three-dimensional splitting. It was found that the optimum size of thread blocks for acceleration – 32×8, however the 16×16 tiling is better scaled under the various sizes of model grid and only slightly inferiors in performance. Therefore for further calculations the block 16×16 was used.

A series of calculations on the test model (with different ratios NX, NY, NZ relative to each other and the total number), and the model of the real field East Moldabek were executed. Analyzing the results of the calculations, identifying the main factors affecting the performance, it is possible to predict the acceleration value of the model in its size (Figs 6, 7). Calculations show that the highest speed-up on GPU is reached on models with large grid size (2 million cells and more). A large number of wells and inactive cells negatively influences on acceleration. Incomplete loading of GPU multiprocessors essentially reduces performance that occurs at calculation of models with small values of NX, NY. The maximum achieved speed-up with GeForce GTX 560 is 60.4 times. It is fair to expect that on more productive GPU of the Tesla series having more processing cores and more memory bandwidth, acceleration of calculation will be still much higher.

So, it is found out, that algorithm of the IMPES method perfectly to approach under architecture of CUDA. Considerable acceleration (from 25 to 60.4 times) was achieved. Acceleration of water flooding calculations on GPU can make competitive alternative of use of expensive high-performance clusters based of CPU.

REFERENCES

[1] Buatois L., Caumon G., Levy B.: *Concurrent number cruncher: a GPU implementation of a general sparse linear solver*. International Journal of Parallel, Emergent and Distributed Systems, vol. 24, Issue 3, June 2009, pp. 205–223.

[2] http://www.seismiccity.com/Technologies.html.

[3] http://www.hanweckassoc.com/.
[4] http://www.ks.uiuc.edu/Research/namd/.
[5] Zverev Ye., Novozhilov Yu., Mikhalyuk D.: *Acceleration of engineering calculations with GPU NVIDIA Tesla in ANSYS Mechanical.* Engineering Technical Journal, 2011, pp. 33–38.
[6] Aziz K., Settari A.: *Petroleum Reservoir Simulation.* Elsevier, New York 1979, 406 p.
[7] Chen Z., Huan G., Ma Yu.: *Computational Methods for Multiphase Flows in Porous Media*. SIAM, Philadelphia 2006, 549 p.
[8] Fanchi J.R.: *Principles of Applied Reservoir Simulation*. Second Edition. Gulf professional publishing, Houston 2001, 357 p.