IRSTI 27.35.57 https://doi.org/10.26577/JPCSIT.2023.v1.i2.05

E. Makhmut*, T.S. Imankulov

Al-Farabi Kazakh National University, Kazakhstan, Almaty
*e-mail: erlanmahimut@gmail.com

COMPARATIVE ANALYSIS OF REDUCTION ALGORITHMS IN CUDA FOR PRESSURE EQUATION

Abstract. This study aims to develop and analysis of different reduction algorithms in CUDA parallel computing technology for calculating pressure distribution in porous media. The parallel model is developed and enough to analyze a computation time of pressure. The main purpose of this study is a comparative analysis of each reduction algorithm computing results on CUDA technologies, in order to show the results of each reduction algorithm. The computing results of each reduction algorithms were compared and analyzed.

Key words: Buckley–Leverett, water flooding, CUDA, reduction, GPU.

Introduction

In oil recovery engineering, using water flooding, secondary recovery technique inject water into oil reservoir has been becoming one of the most important techniques in EOR fields, which provides 40-60% recovered oil [1] , where water and oil are immiscible. The special case of one-dimensional, immiscible, two-phase flow was theoretically investigated in 1942 [2]. Water flooding process can be seen mathematically using Buckley- Leverett model. Using the numerical approaches, the field will be discretizing into several points. Large amount of points will be used in order to obtain the convergence solution. These points will increase computational cost, so in order to reducing the time computational cost, the high performance computing technology CUDA will be required.

CUDA (Compute Unified Device Architecture) is a parallel programming model, which is a bridge between CPU and GPU, not only responsible for to transfer data, but also to realize the parallel computing, because of GPU provides much higher instruction throughput and memory bandwidth. [3-7] shows the information of the suggested optimization and programming model for CUDA architecture. By taking advantage of the throughput and low latency of GPU, like other applications, in oil reservoir fields have attempted to increase the speedup of their computations. In [8] developed a parallel algorithm using CUDA technology for three-dimensional modeling for the reservoir. In [9] authors made comparative analyses of execution time for three-dimensional Poisson equation, and concluded that using a mobile device is possible to conduct computations of complex mathematical models in real time. Paper [10] presented a parallel implementation for a Forward Reservoir Simulation (FRS). As the result of the work, the proposed model proves that the CUDA-based parallel simulator implementation of FRS enables solving an 82 times larger problem than the serial one.

In [11], the authors developed a parallel model for the Kahan Summation algorithm using CUDA parallel reduction methods. The results demonstrated that optimizing the algorithm led to a decrease in execution time and an increase in speedup.

Paper [12] investigated parallel data processing in a hybrid CPU+GPU system, 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 multi-core CPU+GPU systems. The authors obtained results that can be utilized for implementing building blocks for data stream frameworks, particularly focusing on multi CUDA stream communication optimization.

In [13], the paper provided an overview of the CUDA programming model, discussing concepts such as threads, thread blocks, grids, and memory hierarchy. It emphasized the importance of maximizing parallelism and exploiting memory bandwidth to achieve optimal performance in CUDA applications. The authors compared P systems with other parallel paradigms and proposed simulating parallel classical architectures using P systems. The paper specifically focused on simulating the CUDA architecture for solving the parallel reduction problem.

In [14], the paper presented an interesting investigation into parallel reduction operations and proposed a GPU-based parallel approach to improve their performance. The authors effectively highlighted the significance of reduction operations in various computational problems and provided a clear explanation of the concept. The work achieved a commendable 2.8x speedup compared to a serial implementation, showcasing the effectiveness of the presented approach.

In this work, to speed up of the application, used different types of reduction algorithms in CUDA for the computation of the maximum value of new and old pressure, and analyzed the computation time and speedup.

2 Mathematical model.

Water and oil are incompressible fluid in a porous medium reservoir. Therefore, two phases (water, oil) are considered in the model. Mass conservation equation for water and oil phases are following:

$$m\frac{\partial S_w}{\partial t} + div(\vec{v}_w) = 0; \quad (1)$$

$$m\frac{\partial S_o}{\partial t} + div(\vec{v}_o) = 0; \quad (2)$$

$$S_o + S_w = 1, \quad (3)$$

where, m is porosity, S_o and S_w are oil and water saturation, $\vec{v}_w, \vec{v}_o$ are velocity of water and oil. The Darcy's law expresses velocities:

$$\vec{v}_i = -k\frac{f_i(S)}{\mu_i}\nabla p; \quad i=o,w. \quad (4)$$

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:

$$f_w(S_w) = S_w^2, \quad f_o(S_o) = (1-S_o)^2. \quad (5)$$

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

$$div(\vec{v}_w) + div(\vec{v}_o) = 0. \quad (6)$$

The initial condition at $t=0$ is given below:

$$S|_{t=0} = S_0, \; P|_{t=0} = P_0. \quad (7)$$

The boundary conditions:

$$S|_{x=0} = S_{inj}, \; \frac{\partial s}{\partial x}|_{x=1} = 0, \; P|_{x=0} = P_{inj}, \; P|_{x=1} = P_{prod}. \quad (8)$$

To solve the (1)-(8) 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 and capillary pressure 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 displacement is nonlinear. To solve pressure equation (6) Jacobi method was used.

For pressure:

$$P_i^{t+1} = \frac{M_{i+\frac{1}{2}}P_{i+1}^t + M_{i-\frac{1}{2}}P_{i-1}^t}{M_{i+\frac{1}{2}} + M_{i-\frac{1}{2}}}; \quad (9)$$

where,

$$M_{i+\frac{1}{2}} = \frac{M_i + M_{i+1}}{2}; \quad M_{i-\frac{1}{2}} = \frac{M_i + M_{i-1}}{2};$$

CUDA programming model for oil displacement problem.

In this work, the CUDA technology is used for the realizing of the parallelization. Figure 1(a) describe the process of initialization and call kernel. The main problem of this step is that initialization of device variables used for computation of pressure. Firstly, define corresponding variables, and then allocate memory for these variables. Secondly, copy the corresponding data from host to device to be computationally amenable. One thing to note that before "call kernel" operation, block number and block size must be considered, it directly associated with the number of threads in the kernel. It defined by following process, for one-dimensional case:

```
dim3 threadsPerBlock(number_of_threads);
dim3 numBlocks((threadsPerBlock.x +N-1) / threadsPerBlock.x);
```

kernel<<< numBlocks, threadsPerBlock >>>(parameters).

Thirdly, to call "kernel", described in Figure 1(b). Finally, copy the result from device to host, and output the result.

Once the "call kernel" function is connected, the computation of pressure is implemented, as described in Figure 1(b). In the "call kernel" function, the following operation is realized:

1) For the control of an independent thread, the global ID is calculated. As following equation: global_id=threadIdx.x + BlockIdx.x*BlockDim + GridIdx.x*GridDim.

2) For finding max value of pressure, the thread ID is calculated, and the shared memory is defined.

3) The value of pressure in the new period is calculated using equation (9).

4) The subtraction of new and old value of pressure is calculated, and the result is saved in shared memory for finding max value.

5) The max value in each block is calculated using reduction algorithm.

6) The biggest max value of blocks is calculated using atomic operation.

7) The old pressure value is updated by new pressure value.

8) The received biggest *max* value is compared with computation accuracy, the computation cycle (3)-(7) continues until the *max* is less than computation accuracy, otherwise the computation cycle ends.

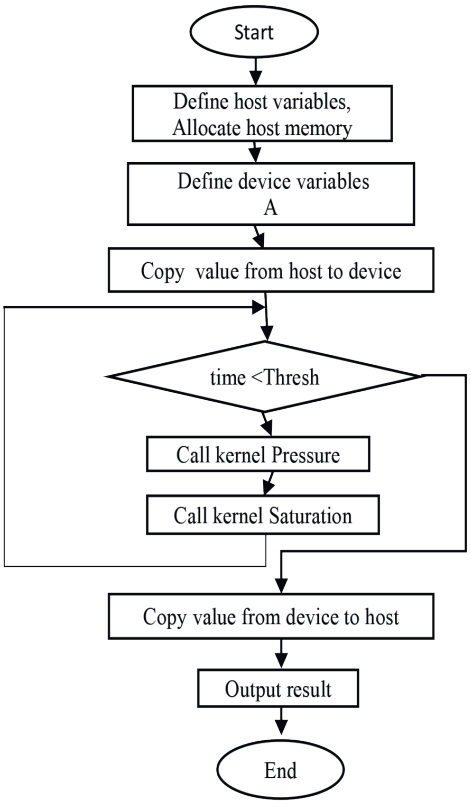

Figure 1(a) – CUDA-parallelization

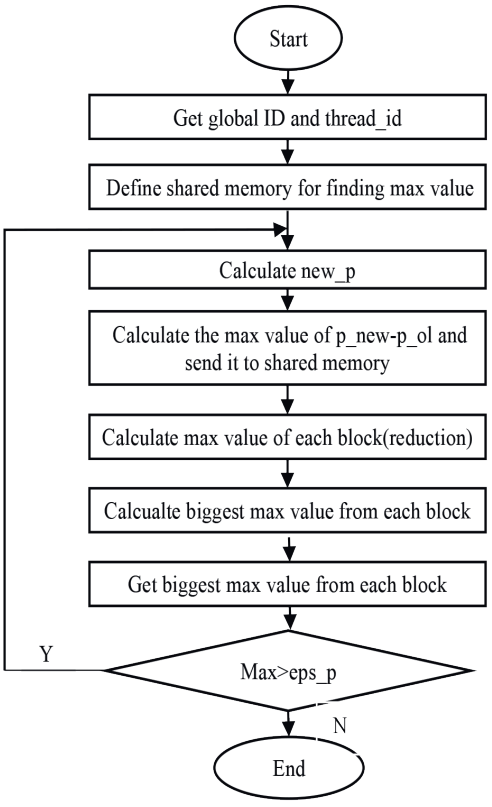

Figure 1(b) – The computation of pressure in CUDA

Reduction algorithms in CUDA

In this model, it must be considered how to implement the computation of the *max* value because the reduction algorithm is used for this operation. It is in some degree impact on the performance of the CUDA application. How to strive to reach the peak performance of GPU is related to choosing

the right metrics in CUDA, such as GFLOP/s for compute-bound kernels or Bandwidth for memory-bound kernels. Because of the reduction operation have very low arithmetic intensity, it should strive for peak bandwidth. In this work, different versions of reduction is constructed, and that is present different performance concerns in CUDA. Each test was performed on a GTX 1060Ti using CUDA 7.0 and a Linux operating system.

Figure 2 describes the detailed process of the interleaved addressing method for parallel reduction. The max data accessed by adjacent threads in the block is evaluated. At the beginning, stride is 1, and every thread will be attended to compute part of the reduction. The value of stride will be doubled as further steps are taken. With the increasing the value of stride, the number of threads involved in the computation was decreased. Although this reduction algorithm enhanced the performance significantly, but the problem is that there are not adjacent threads are participated in the computation, it highly divergent, and wraps are very inefficient.

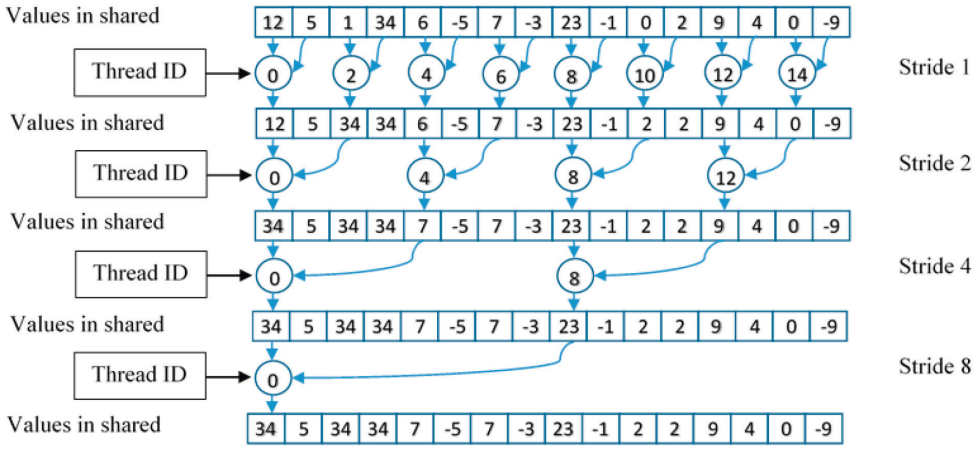

Figure 2 – Interleaved addressing method using for the computation of max value of pressure

To avoid this problem, non-divergent branch is considered. Rather than the first reduction algorithm, this approach is achieved that these adjacent threads are participated in the computation, as you can see in Figure3, in each step these threads ID is increased sequentially. In some degree, it increased the performance, but the same bank is visited by different threads at the same time, which is called shared memory bank conflicts.

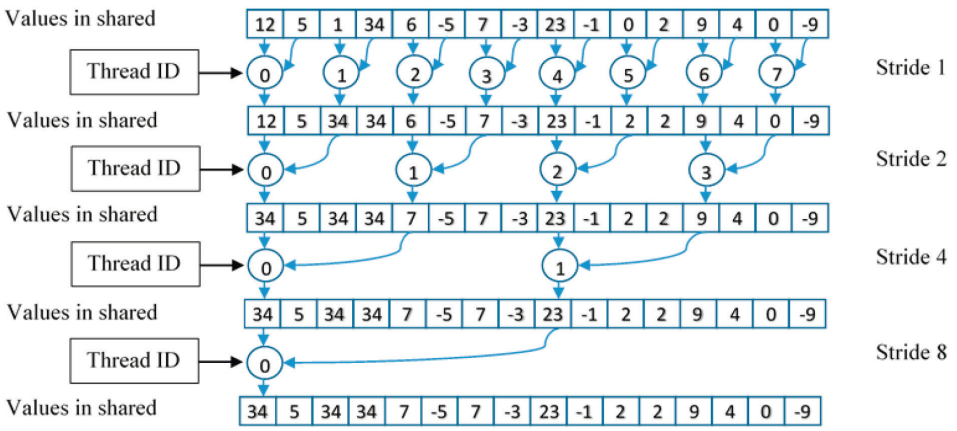

Figure 3 – Interleaved non-divergent addressing method using for the computation of max value of pressure

The sequential addressing reduction algorithm is successfully avoided this problem. Figure 4 illustrates the process of this algorithm. Not only these adjacent threads are participated, but also these threads keep their id with the corresponding memory address. For example, the first address is visited by thread0, and the second address is visited by thread1, and so on. This algorithm presented a significant enhancement in the performance rather than first one.

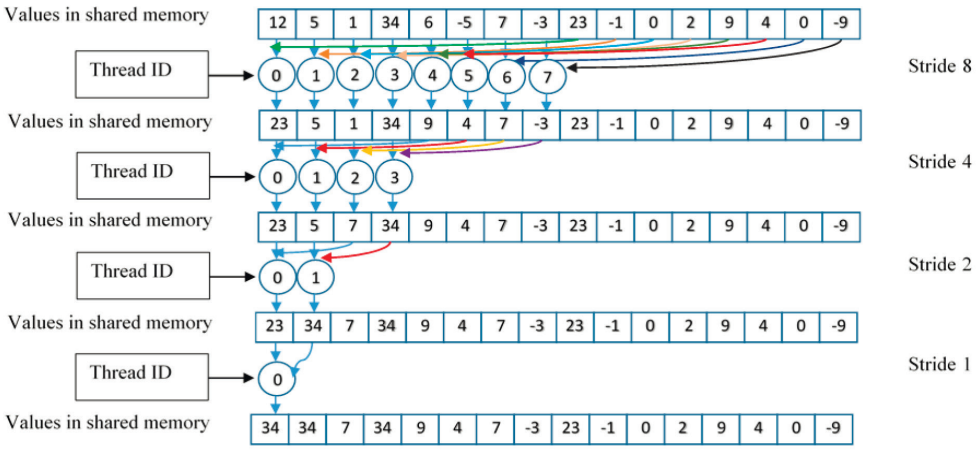

Figure 4 – Sequential addressing method using for the computation of max value of pressure

Even if it gives the higher performance from the above illustrated algorithm, but these reduction algorithms have low arithmetic intensity, and there has a possible bottleneck is caused by instruction overhead from the address arithmetic and loop. In an inner loop, after the computation of the max value, synchronization operation is implemented. This latency also causes the performance decrease. So partial avoid synchronization, a method is considered that it only unrolls the last six iterations of the inner loop, is called unrolling the last warp. Because within a wrap, instructions are synchronous. Without unrolling, all wraps execute every iteration of the loop and if statement.

The number of iterations is another important element in reduction operation, if it is known before the compile time, the above reduction algorithm could be changed into complete unrolling. In modern GPU, the block size is up to 1024, fortunately. Normally, block size is getting the value 8, 16, 32, 64, 128, 256, 512, and 1024, is sticking to power-of-2 operation, and for a fixed size of block, complete unrolling reduction is easily implemented. So, the number of iteration must be fixed with the size of block at compile time. This template supported by NVIDIA CUDA programming technology.

Results

Table 1 shows the input parameters of CUDA parallel model.

Table 1 – Parameters of model

Parameters	Value
1	2
Absolute permeability	0.001
Water viscosity	0.09
Oil viscosity	0.3
Porosity	0.2
P_{inj}	0.5

1	2
P_{init}	0.3
P_{prod}	0.1
S_{inj}	1.0
S_{init}	0.001

Execution time of serial model is shown in Table 2. Execution time of serial model is shown in Table 2. Execution time is increased as the number of points is increased. These results are only used for the experiment.

Table 2 – Serial execution time (for pressure)

Number of element	Execution time(s)
2^{14}	6.501
2^{16}	27.985
2^{18}	120.383
2^{20}	536.222
2^{22}	2221.43
2^{23}	10960

In this part, the performance of parallel version of above reduction algorithms are presented. In GPU device, when block size is 512, it gives best result compared with other sizes (64, 128, 256, 1024). So, in this work, 512 block size is used for all tests. Table 3 shows the execution time of pressure using different reduction algorithms.

Table 3 – The execution time of pressure used different reduction algorithm

Number of element	Interleaved addressing-divergent		Interleaved addressing – non divergent		Sequential addressing		Unroll last warp		Complete unroll	
	Execution time	Speedup	Execution time	Speedup	Execution time	Speed up	Execution time	Speedup	Execution time	Speed up
2^{14}	2.076	3.1	2.075	3.1	2.061	3.2	1.893	3.43	1.898	3.43
2^{16}	6.439	4.3	6.345	4.4	6.306	4.44	5.663	4.94	5.631	4.97
2^{18}	23.35	5.2	22.84	5.3	22.90	5.3	20.34	5.92	20.17	5.97
2^{20}	92.04	5.8	91.39	5.9	90.32	5.94	80.69	6.65	80.04	6.7
2^{22}	373.5	6.0	368.7	6.03	366.9	6.05	325.2	6.83	322.25	6.90
2^{23}	749.2	14.6	718.6	15.25	711.9	15.4	654.9	16.74	643.9	17.0

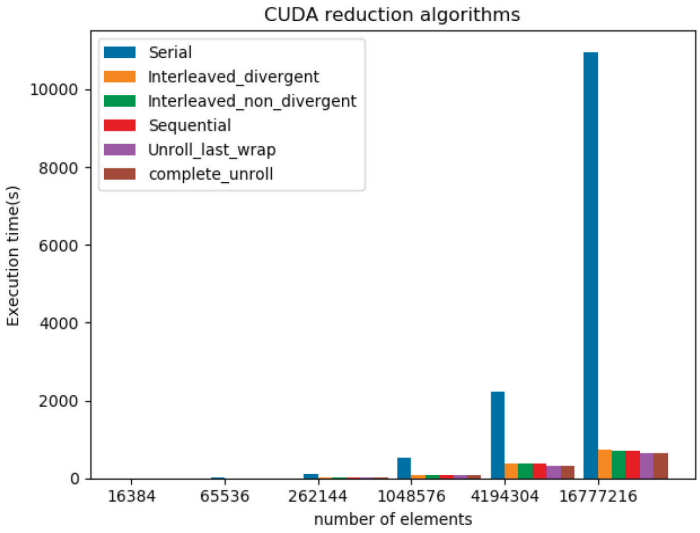

Figure 5 – Computation time serial and different types of parallel reduction algorithm

With the using of the different techniques of threads in shared memory, the execution time is increased, this case is presented in Table 3.

Figure 5 shows the parallel execution time of CUDA using different types of reduction algorithms, and 6 shows the speedup of these reduction algorithms on computation of pressure.

To reach the peak performance of GPU, first, it must be avoiding highly divergence of threads. Secondly, as possible as to avoid bank conflicts in the shared memory. Thirdly, as possible as to use instruction within a wrap, because there is no any __syncthreads() operation. Finally, to know how to unrolling block sizes at compile time. These problems are solved systematically in this work, execution time is gradually reduced as shown in Figure 5, and speedup is gradually increased, as shown in Figure 6.

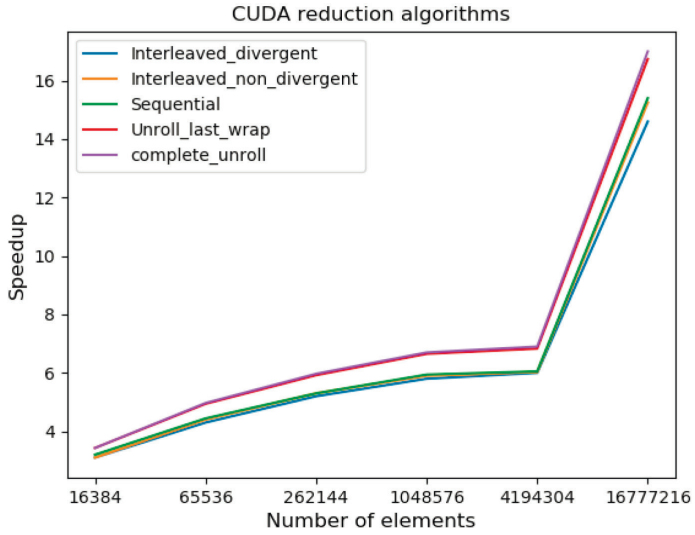

Figure 6 – Speedup of each parallel reduction algorithm

Conclusions

In this work, for the oil displacement problem, the Buckley-Leverett method is considered. For the solution of numerical model, Jacobi method is applied. The serial and a high-performance computing algorithm is created. We developed a high-performance computing model using different reduction algorithms in CUDA. The results of each reduction algorithm is analyzed. The results show that the execution time or speedup of model is affected by the divergence of threads, bank conflicts, sequential of threads, synchronization, and a compile time, as shown in Table 3. With the increasing of the number of points, the speedup of each parallel algorithm is increased compared with serial one. Although the difference of the result of execution time between each reduction algorithm is very small, but these improvements is also increased speedup of these reduction algorisms.

Reference

1. Willhite, G. Paul [1986]. “Waterflooding.”, Society of petroleum engineers, Richardson, TX, Textbook Series Volume, 3: 1-7.

2. Buckley, S.E. and Leverett, M.C. (1942), Mechanism of Fluid Displacement in Sands, Transactions AIME, Vol.146, pp.107-116.

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

4. N. Wilt, The Cuda Handbook: A Comprehensive Guide to GPU Programming, Pearson Education, 2013.

5. C. Nvidia, Programming guide, ed, 2008.

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

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

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

9. 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.

10. 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), http://dx.doi.org/10.1016/j.cpc.2016.04.010

11. Adityo Mahardito, Adang Suhendra, Deni Tri Hasta. “Optimizing Parallel Reduction In Cuda To Reach GPU Peak Performance Optimizing Parallel Reduction In Cuda To Reach GPU Peak Performance”, Gunadarma University Repository(2018), oai:repository.gunadarma.ac.id:722

12. Paweł Czarnul, “Investigation of Parallel Data Processing Using Hybrid High Performance CPU + GPU Systems and CUDA Streams”, Computing and Informatics, Vol. 39, 2020, 510–536, doi: 10.31577/cai_2020_3_510

13. R. Ceterchi, M. Angel Martínez-del-Amor, and M.J. Pérez-Jiménez. “The Reduction Problem in CUDA and Its Simulation with P Systems”, Research Group on Natural Computing (2014).

14. Walid Jradia, Hugo do Nascimento and Wellington Martins. “A Fast and Generic GPU-Based Parallel Reduction Implementation”, Symposium on High Performance Computing Systems (WSCAD) (2018).