High-Performance Computational and Information Technologies for Numerical Models and Data Processing

Darkhan Akhmed-Zaki, Madina Mansurova,
Timur Imankulov, Danil Lebedev, Olzhas Turar,
Beimbet Daribayev, Sanzhar Aubakirov,
Aday Shomanov and Kanat Aidarov

Additional information is available at the end of the chapter

Abstract

This chapter discusses high-performance computational and information technologies for numerical models and data processing. In the first part of the chapter, the numerical model of the oil displacement problem was considered by injection of chemical reagents to increase oil recovery of reservoir. Moreover the fragmented algorithm was developed for solving this problem and the algorithm for high-performance visualization of calculated data. Analysis and comparison of parallel algorithms based on the fragmented approach and using MPI technologies are also presented. The algorithm for solving given problem on mobile platforms and analysis of computational results is given too. In the second part of the chapter, the problem of unstructured and semi-structured data processing was considered. It was decided to address the task of n-gram extraction which requires a lot of computing with large amount of textual data. In order to deal with such complexity, there was a need to adopt and implement parallelization patterns. The second part of the chapter also describes parallel implementation of the document clustering algorithm that used a heuristic genetic algorithm. Finally, a novel UPC implementation of MapReduce framework for semi-structured data processing was introduced which allows to express data parallel applications using simple sequential code.

Keywords: fragmented algorithm, high-performance visualization, computational algorithms on mobile platforms, MPI, unstructured and semi-structured data processing, n-gram extraction, MapReduce framework

1. Introduction

With development of computer technology level and high-performance systems across the world, efficiency of solving problems in the field of fundamental and engineering research is increasing. Annual development of mathematical models allows to study physical and chemical processes in greater detail. Modern numerical methods are also being developed for solving applied problems and an amount of calculations are increasing. In this regard, using high-performance computing and computational technologies to solve applied problems with each year becomes more relevant.

In the middle stages of the development of high-viscosity oil fields, the problem of decreasing oil recovery becomes an issue. Increasing oil recovery of reservoirs remains one of urgent tasks at the moment. Methods of injecting polymers and surfactants into an oil reservoir are currently widely used in the oil industry and are considered as one of the effective methods for increasing the oil recovery of reservoirs [1, 2]. Therefore, the problem of oil displacement process by polymer and surfactant flooding was perceived as being a task for given working group.

Parallel implementation of the oil displacement problem and applied method appears to be complex problem of system parallel programming because it requires to provide synchronization of separate computational processes, network data transfer, etc. In order to decrease complexity of such parallel programs, technology of fragmented programming and its implementation called LuNA (Language for Numerical Algorithms) were adopted [3].

Visualization is an integral part of the analysis during the processing of the scientific data. It has a significant role in large-scale computational experiments on modern high-performance engines. The amount of data obtained in such calculations can reach several terabytes. Such system requires a well-designed and implemented client-side visualization module taking into account its client orientation. So such programming module was applied using modern visualization technology Vulkan API [4].

Nowadays full computational potential of mobile devices almost not used because of devices being idle for extended periods during a day. There are number of projects such as Berkeley Open Infrastructure for Network Computing (BOINC) which use excessive computational capabilities of PCs and mobile devices across the globe [5]. While provisioning services for its customers as integrator of numerous computational resources for solving their problems, the processing itself was conducted using only CPUs. Many recent mobile devices are equipped with powerful GPUs generally used for 3D graphics rendering. By efficient usage of mobile GPUs, one can achieve much more performance from a single device therefore increasing overall productivity of such integrational computations. This task requires the mobile software installed to be able to use capabilities offered by GPUs. Following researchers studied issues and possibilities related to exploit GPU capabilities of mobile devices in integrated computations: Zhao [6], Montella et al. [7].

Because of the rapid progress on computer-based communications and information dissemination, large amounts of data are daily generated and available in many domains. The purpose of the research presented in the second part of the chapter is to develop models and algorithms for unstructured and semi-structured data processing using high-performance parallel and distributed technologies.

Today huge amount of information are being associated with the web technology and the internet. To gather useful information from it, these text has to be categorized. Text categorization is a very important technique for text data mining and analytics. It is relevant to discovery various different kinds of knowledge. It is related to topic mining and analysis. It is also related to opinion mining and sentiment analysis, which has to do with knowledge discovery about the observer, the human sensor. The observer based on the content they produce can be categorized. The indexing influences the ease and effectiveness of a text categorization system [8]. The simplest indexing is formed by treating each word as a feature. However, words have properties, such as synonymy and polysemy. These have motivated attempts to use more complex feature extraction methods in text categorization tasks. If a syntactic parse of text is available, then features can be defined by the presence of two or more words in particular syntactic relationships. Nowadays authors [9–11] have used phrases (n-grams), rather than individual words, as indexing terms. In this work, the task was also addressed to n-gram text extraction which is a big computational problem when a large amount of textual data is given to process. In order to deal with such complexity, there was a need to adopt and implement parallelization patterns.

The chapter also focuses on research related to the application of genetic algorithm for document clustering. Genetic algorithms make it possible to take into account peculiarities of the search space by adjusting the parameters and selecting the best solutions from the solutions obtained by the population [12–14]. Clustering algorithm is based on the assessment of the similarity between objects in a competitive situation. Since clustering problem solution requires large computational resources parallelization on the stage of genetic algorithm for setting the coefficients in the formula of similarity measures was performed, as well as on the stage of data clustering.

MapReduce technology has shown a great potential in dealing with large-scale data processing problems [15, 16]. Such batch-oriented MapReduce systems as Apache Hadoop, however, lacks efficiency in dealing with iterative problems. The main bottleneck can be attributed to slow disk operations arising in data storage after current iteration in a distributed file system. Number of solutions that deal with that problem has been proposed in a literature, including ones that propose novel techniques that optimize loops [17] and ones that try to keep static data [18]. Recently introduced novel approaches rely mostly on in-memory processing mechanisms [19, 20]. Also some types of data parallel problems require efficient communication between parallel workers in order to be able to implement specific nature of the data exchange patterns. In such a way, it is necessary to consider other parallel programming models that can be effectively combined with MapReduce.

Partitioned global address space (PGAS) model presents an interesting approach to deal with data communication problem. In PGAS model, a global memory is divided among threads with different choices of memory to thread mappings. Several works introduced different approaches to implement MapReduce functionality in a frame of PGAS model. For example, in [21], authors introduce a design of MapReduce system based on using unified parallel C that belongs to a family of PGAS languages. In that approach collective operations for data exchange are employed. A different implementation of MapReduce based on X-10 parallel programming language of PGAS family uses hashmap data structure to deal with data exchange task [22].

2. Mathematical and computer modeling of 3D oil displacement process in porous media

2.1. Mathematical model of polymer and surfactant flooding

In general processes of oil displacement by chemical reagents controlled by complex physical and chemical processes. Therefore, exact simulation of such processes using numerical methods produces a number of certain issues. Therefore, the mathematical model of two-phase flow in porous media has the following assumptions: (1) flow is incompressible; (2) gravitational forces and capillary effects are neglected and (3) two-phase flow (water and oil) obeys Darcy's law.

Taking into account the foregoing assumptions, a system of equation was written for two-phase flow in porous media, which contains mass conservation equations for water and oil phases, the Darcy's law, and the equation for the transfer of concentration and salt in the reservoir [23, 24].

Mass conservation equations can be written as follows:

$$m\frac{\partial S_w}{\partial t} + div(\boldsymbol{v_w}) = q_1 \tag{1}$$

$$m\frac{\partial S_o}{\partial t} + div(\boldsymbol{v_o}) = q_2 \tag{2}$$

where m is the porosity, S_w, S_o are the water and oil saturations, q_1, q_2 are the source or sink. Porous medium saturated with fluids: $S_w + S_o = 1$.

Velocities of each phases is given by Darcy's law:

$$\boldsymbol{v_i} = -K_0\frac{f_i(s)}{\mu_i}\nabla P, \quad i = w, o \tag{3}$$

where $f_i(s)$, μ_i is the relative permeability and viscosity for phase i, P is the pressure, K_0 is the permeability tensor.

Polymer, surfactant, salt and heat transport equations are given by:

$$m\frac{\partial}{\partial t}(c_p s_w) + \frac{\partial a_p}{\partial t} + div(\boldsymbol{v_w} c_p) = div(mD_{pw} s_w \nabla c_p) \tag{4}$$

$$m\frac{\partial}{\partial t}(c_{sw}s_w + c_{so}s_o) + \frac{\partial a_{surf}}{\partial t} + div(\boldsymbol{v_w}c_{sw} + \boldsymbol{v_o}c_{so}) = div(mD_{sw}s_w\nabla c_{sw} + mD_{so}s_o\nabla c_{so}) \quad (5)$$

$$m\frac{\partial}{\partial t}(c_s s_w) + div(\boldsymbol{v_w}c_s) = 0 \quad (6)$$

$$\frac{\partial}{\partial t}[(1-m)C_r\rho_r T + m(C_w s_w \rho_w + C_0 s_0 \rho_0)T] + div(\rho_w C_w \boldsymbol{v}_w T + \rho_0 C_0 \boldsymbol{v}_0 T) = div(D\nabla T),$$

$$D = (1-m)\lambda_0 + m(\lambda_1 s_w + \lambda_2 s_0) \quad (7)$$

where c_p, c_s is the concentrations of polymer and salt in water phase, $c_{sw,}$ c_{so} are the concentration of surfactant in water and oleic phases, a_p, a_{surf} are the adsorption functions, D_{pw}, D_{sw}, D_{so} are the diffusion coefficients, C_w, C_o, C_r are the specific heat of water, oil and rock, ρ_w, ρ_o, ρ_r are the water, oil and rock densities, λ_0, λ_1, λ_2 are the thermal conductivity coefficients.

Initial conditions:

$$s_w|_{t=0} = s_{w0},\ c_{pw}\big|_{t=0} = c_{p0},\ c_s|_{t=0} = c_{s0},\ c_{sw}|_{t=0} = c_{sw0}, \quad c_{so}|_{t=0} = c_{so0},$$

$$T|_{t=0} = T_p,\ a_{surf0}\big|_{t=0} = a_{surf0}, \quad a_p\big|_{t=0} = a_{p0} \quad (8)$$

Boundary conditions:

$$\left.\frac{\partial s_w}{\partial n}\right|_{\partial\Omega} = 0; \qquad \left.\frac{\partial P}{\partial n}\right|_{\partial\Omega} = 0; \qquad \left.\frac{\partial T}{\partial n}\right|_{\partial\Omega} = 0; \left.\frac{\partial c_{pw}}{\partial n}\right|_{\partial\Omega} = 0;$$

$$\left.\frac{\partial c_{pw}}{\partial n}\right|_{\partial\Omega} = 0; \quad \left.\frac{\partial c_{sw}}{\partial n}\right|_{\partial\Omega} = 0; \quad \left.\frac{\partial c_s}{\partial n}\right|_{\partial\Omega} = 0 \quad (9)$$

The following viscosity dependence on injected reagent concentrations and temperature was used:

$$\mu_a = \mu_w\Big[1 + \Big(\gamma_1 c_p + \gamma_2 c_p^2 + \gamma_3 c_{sw} + \gamma_4 c_{sw}^2\Big)c_s^{\gamma_5} - \gamma_6\big(T - T_p\big)\Big] \quad (10)$$

$$\mu_o = \mu o_o\big[1 - \gamma_7\big(T - T_p\big)\big] \quad (11)$$

where $\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6, \gamma_7$ are the constants, μo_o is the initial viscosity of oil phase, T_p is the reservoir temperature. The imbibition relative permeability curve for water/oil flow is given by

$$f_w(S_w) = S_w^{3.5};\ f_o(S_w) = (1 - S_w)^{3.5} \quad (12)$$

The process of displacement of oil by polymer and surfactant solutions described through developed mathematical model. First oil reservoir filled with surfactant solution is driven by conventional water. After that polymer solution injected in order to control the slug which improves volumetric sweep efficiency. This procedure followed by injection of an ordinary water flow. The amount of surfactant, polymer and water injected must be computed through developing mathematical model describing distributing of pressure and temperature, saturation of each phase, chemical concentrations of the process flowing within a reservoir. Reservoir dimensions and shape described within mathematical model as three-dimensional computational domain (**Figure 1a**).

The numerical solution of Eqs. (1)–(12) based on finite difference method and explicit/implicit scheme. The algorithm for constructing a solution is reduced to the following. The temperature of the reservoir and the injected water, the initial oil saturation of the reservoir, the initial pressure distribution, the technological and physical parameters of the reservoir are set. The values of saturation, pressure, concentration and temperature are solved according to the explicit Jacobi scheme on the basis of the finite difference method in the three-dimensional grid [25] (**Figure 1b**).

2.2. Fragmented algorithm

For solving the three-dimensional fluid flow problem, the method with stabilizing correction was used [26]. For implementation of parallel algorithm initial area ($Nx \times Ny \times Nz$) divided to subdomains. At first division, division made by z-axis where number of subdomains depends on number of processes, size of subarea equals Nz/size+2 shadow edges.

After that computations for the first and the second intermediary step were made in order to find pressures and saturations by algorithm described in previous section. Then the second subdivision of initial domain was done by y-axis and compute values of gas pressure by third step of the method. After the third step, boundaries for all variables were exchanged and compute first step of the method for further time step. At the end of this step domain was made subdivision again but already by x-axis and compute the second and the third steps. After that subdivision was made on by z-axis again, exchange shadow edges and start to compute first and second intermediary steps as shown in **Figure 2**.

Advantage of such scheme of initial three-dimensional domain division at computing two intermediary steps is the possibility to solve independently at each process by sequential sweep [27] for which there is no efficient parallel algorithm. But there are global communications after each second intermediary step which appears when initial domain is divided.

Testing conducted on “MVC” Supercomputer of Unified Supercomputer center of the Russian Academy of Sciences [28] which include nodes with two Xeon E5-2690 processors,

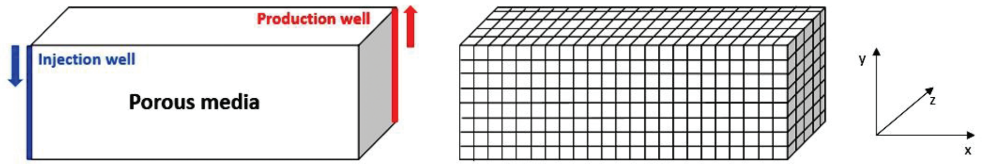

Figure 1. Computational domain (a) and computational grid (b).

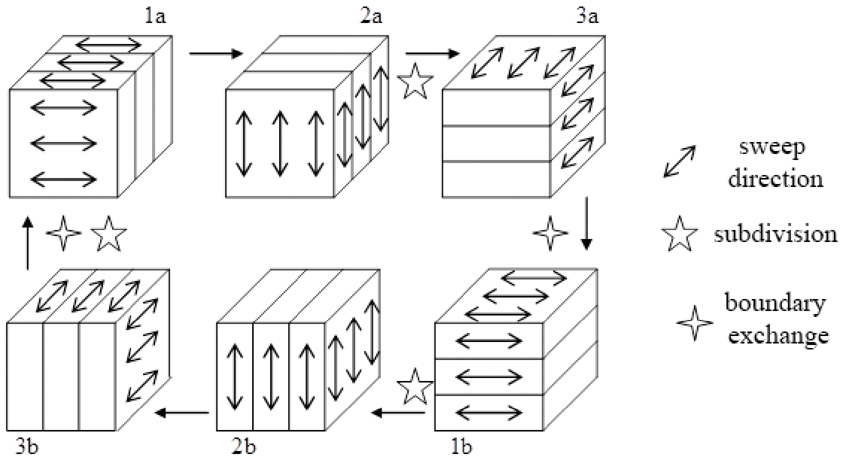

Figure 2. Scheme of computations for three-dimensional fluid flow problem.

communicational and transport network based on FDR Infiniband and 64 Gb of Operating Memory for each node. MPI MPICH3 and GCC compiler used.

Series of experiments allowed to define weak scalability of the implementation. That is for different problem sizes increase in problem size was proportional to number of computational nodes. Ideally, computation time must be equal for every experiment because number of computations in each node is approximately the same. In reality, increased time leads to increase in communication length and size of transferred values.

As show in **Figure 3** (x-axis shows number of processes and appropriate domain size), MPI implementation possesses best efficiency because unlike fragmented program it does not have overhead expenses belonging to LuNA system algorithms [29]. But efficiency does not reach 100% because of existing global communications appearing at decomposition of the domain in a process. Moreover, it can be noted that LuNA implementation approximately 200 time slower than MPI while LuNA with manual setting (LuNA-Fw) approximately 10 times slower.

From **Figure 3**, it can be seen that with increase of the problem size execution time for the sequential program disappears. This related to the fact that program data no longer fits to a memory of a single node while parallel and fragmented programs still do.

2.3. High-performance visualization

Let us consider the visualization module. Highly optimized visualization API with cross-platform support is needed. Previously only the OpenGL standard can be such tool. However, OpenGL has a number of limitations, mainly related to its high-level implementation. Because of this, it cannot use advantages of processors from different manufacturers.

At the moment, a new low-level visualization standard, the ideological continuation of OpenGL, Vulkan API [4] is rapidly developing. It also contains a functional for parallelizing

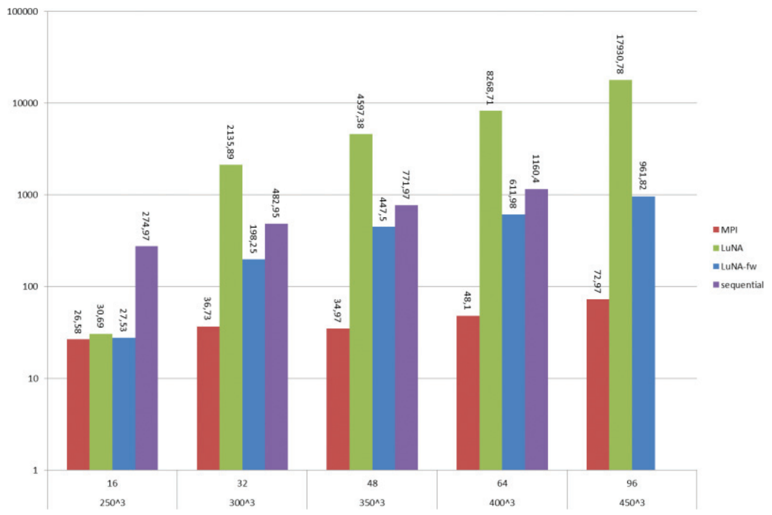

Figure 3. Weakly scalable testing. Dependence of computation time from the size of a problem and processes.

CPU-side computations and provides multiple performance improvements by reducing the number of CPU addressing using a technology similar to AMD Mantle [30]. Vulkan is a cross-platform response from the Khronos Group to the latest DirectX 12 Direct3D standard [31] developed by Microsoft and released with the Microsoft Windows 10 operating system.

The Vulkan toolkit is extremely low-level and most settings are manually configured. The main reason of the high performance shown by Vulkan compared to OpenGL is due to the decrease of the dependence of video processor on the CPU. Indeed, in the old visualization tools, the drawing of each animation frame was each time run directly from the CPU. Thus, after each rendering iteration and presentation to the screen, a signal was sent to the CPU, after which the video processor waited the completion of current CPU commands before launching the next iteration. In other words, CPU and GPU were synchronized on each call of the render function.

In an application that uses the Vulkan API, parts that flow on the CPU and the video processor are generally unsynchronized. Synchronization at the moments of necessity is controlled by the application itself, not by the driver and the library, as it was in OpenGL.

The demonstration of the implemented module is shown in **Figure 4**.

To test the capabilities of this visualizer, special test models obtained from the models presented above were used (**Figure 5**). To simplify the creation of model, colors of the cells were generated randomly. From this basic model (**Figure 5a**), using a special generator, a similar model was created consisting of a much larger number of polygons and active cells. Test models were generated by splitting each polygon of the base model.

The model shown in **Figure 5b** has a surface consisting of 62,078,400 polygons. The geometry of this model occupies 1420 MB or 1.387 GB on graphics memory. One can interact with this

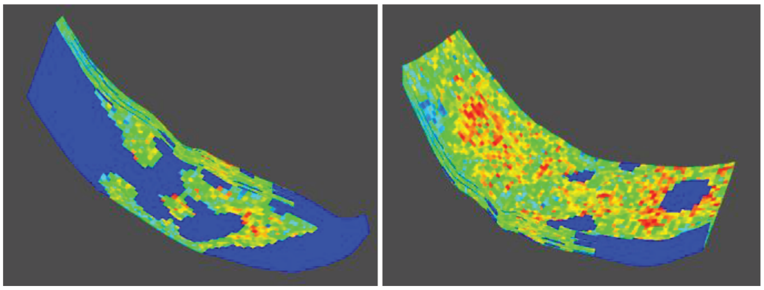

Figure 4. Demonstration of the visualization module using the Vulkan API (different colors represent the values of permeability along the Ox axis: red color for maximum values and blue for minimum).

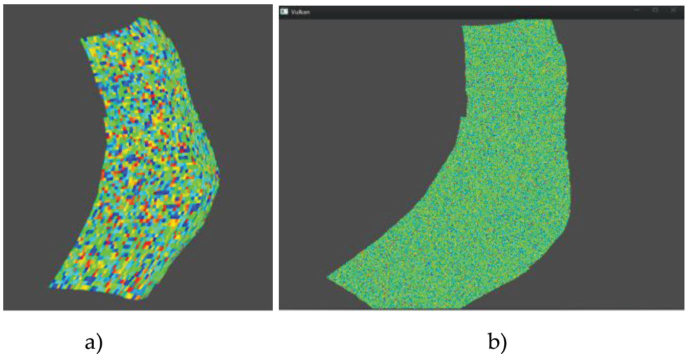

Figure 5. Basic model containing 67,165 active cells (a) coarse mesh; (b) finer mesh.

model in real time, since the rendering is done at a frequency of 51–53 frames per second. At the moment, Vulkan standard allows to significantly optimize the performance of graphics applications due to special technologies for working with data and video card resources at a low level.

By using the described technology, a new version of the information system visualizer shows a significant increase in drawing performance. The presented results of the rendering speed can theoretically correspond to the models with hundreds of millions of computational cells.

2.4. Computational algorithms on mobile platforms

2.4.1. Creation high-performance software on mobile devices

Nowadays, rapid grows of number of mobile devices pushed mobile industry to the very top of the global technological market making it one of the most important areas of public services. Huge interest in mobile market from common population set technological trend of mobile industry to a fastest possible route. CPUs and especially GPUs present in modern mobile devices being absolutely separate computational units can be used as a parts of heterogeneous computational platform combining ordinary servers and other alternative connected devices with purpose of integral computation. Recent models of mobile devices equipped with GPUs supporting nVidia CUDA technology. This technology can be used to implement parallel versions of conventional algorithms further allowing to solve computational intensive tasks. The mobile nature of computational devices allows to exploit them directly at oil field even if there no wireless connection. In case if there is access to digital network they can be a part of heterogeneous computational cluster.

This section describes oil displacement problem in order to test the parallel algorithm on GPU, which uses a shared memory for storage of a grid node values and comprises of various comparative tests focusing on effectiveness of mentioned algorithms. Oil displacement problem by polymer and surfactant injection taking into account temperature effects. The computer model described the complex real industrial problem of oil recovery [32, 33].

2.4.2. A parallel algorithm using CUDA technology on a mobile platform

Let us make assumption that GPU grids chosen for allocating program data contain several blocks. Every block represented in a three-dimensional form and program data copied from the global memory to the shared memory of a GPU during computation. After relocation of data into shared memory it cannot be used again. Therefore, it will be copied back from the global memory which usually appears to be slowest one. It means that copying data from the global memory to GPUs shared memory four times creates inefficiency. Other issue is that each subdomain requires data from its neighbors to continue computation. This creates situation where data will be copied from the global memory every time the boundary layer data needed [34].

To tackle abovementioned issue, the kernel function introduced into parallel implementation of an algorithm. Using kernel function allows data to be declared in a shared memory according to a size of a problem. Every thread within a single block has access to the shared memory only. The performance of the shared memory much higher than of the global memory. This allows to avoid loading data from the global memory every time the boundary data required which leads to noticeable increase in performance. The parallel algorithm within the kernel function works as follows: a temporary array declared in the shared memory where an output of a calculation will be stored; at first this array represent a copy of an input array; after that values at edges of a given array will be replaced by boundary values from neighboring blocks. Every time algorithm requires input data values it gets them from the shared memory instead of the global memory. High performance of the shared memory significantly speeds up total computation. One must be careful when working with boundary values from neighboring block because wrong choice of indices lead to incorrect output data.

Conducting mobile computations on problems related to real-world technological processes recently became popular topic among science and industry. Game industry actively utilize computaional capabilities of recent mobile devices by developing games with high-performance graphical data processing. Other examples are image recognition and machine learning in mobile cloud services. One can easily notice rapid grows of computational capabilities of modern mobile devices. Recent developments in this industry like nVidia Tegra X1 chipset possesses 256

GPU cores based on nVidia Maxwell architecture [35]. This chipset has extraordinary computational performance potential up to 1 TFlops which is comparable with performance of small supercomputer. Such situation undoubtedly expands area of problems solving with such devices to a new height.

2.4.3. Results and comparison of computational experiments

In **Figure 6**, the ratio of calculating time for solving oil recovery problem on the PC and the mobile device can be seen. By increasing the grid size, the time ratio decreases.

Figure 7 demonstrates the application for the mobile device on the base of the model of oil displacement process by polymer and surfactant taking into account the salinity and temperature of the reservoir in porous medium.

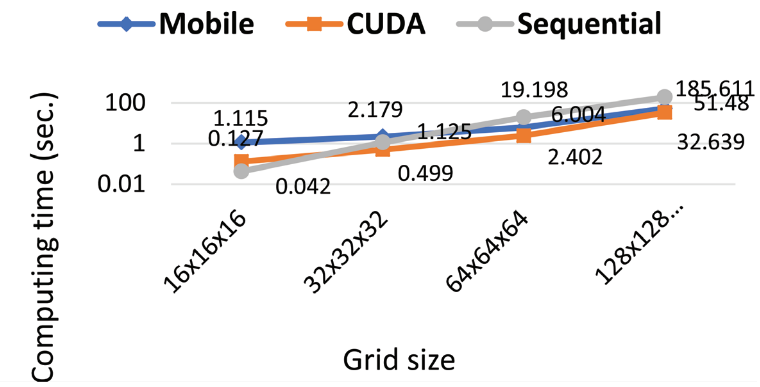

Figure 6. Computing times of mobile device and PC (polymer and surfactant flooding).

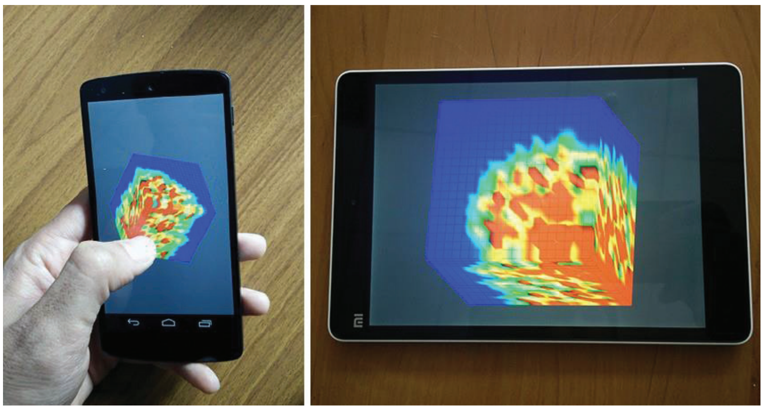

Figure 7. Demonstration of the mobile application results.

The prototype of hydrodynamic simulator developed for high-performance computations on mobile devices and uses existing industrial file formats of a known foreign software companies (Schlumberger, Roxar, etc.). This means that computation results of developed simulator can be backward compatible with file formats used in other software products by these companies. Advantage of a mobile application before traditional one is that the mobile app allows users get results of a computation being located directly on the field.

3. High-performance information technologies for data processing

3.1. Comparison of distributed computing approaches to complexity of n-gram extraction

Nowadays there are several HPC frameworks and platforms that can be used for the distributed computing text processing. n-Gram extracting task was implemented on three platforms: (1) MPJ Express, (2) Apache Hadoop, and (3) Apache Spark. Moreover two different kinds of the input datasets were used: (i) small number of large textual files and (ii) large number of small textual files. Experiments were conducted with each of the HPC platform, each experiment uses both datasets and the experiment repeats for a set of different file sizes. The speedup and efficiency among MPJ Express, Apache Hadoop, and Apache Spark were computed. The guidelines for choosing the platform could be provided based on the following criteria: kind of dataset (i) or (ii), dataset size, granularity of the input data, priority to reliability, or speedup. The contributions from our work include:

- Comprehensive experimental evaluation on English Wikipedia articles corpora;
- Time and space comparison between implementations on MPJ Express, Apache Hadoop, and Apache Spark;
- Detailed guidelines for choosing platform.

The n-gram feature extraction was conducted from the Wikipedia articles corpora. The corpora size is 4 gb and it is consists of more than 200000 articles, each article's size is approximately 20 kb. Furthermore all dataset was divided into six subsets: 64, 256, 512, 1024, 2048 and 4096 Mb, where each subset is divided into two sets: (i) a large number of small textual files and (ii) a small number of large textual files. The articles were kept as is for data set (i), whereas articles were concatenated into bigger files for data set (ii).

Our goal is to extract n-gram from all articles and from each article separately. So the full n-gram model was considered to be all extracted n-grams, where $n \in [1, k]$ and k are the length of longest sentence in the dataset. Further the method that is described by Google in their paper [36] was used and improvements suggested by work [37] were considered. Both algorithms are based on MapReduce paradigm. Method proposed by [37] optimized memory consumption overall performance, but at the same time rejecting not frequent n-grams. The reason of using Google's proposed algorithm is because our goal is to obtain full n-gram model.

As a result the algorithm for our goal of n-gram extraction was adopted from the individual articles. Our method operates with sentences, text of the articles is represented as set of sentences S, where $S = (S_1, S_2, \ldots, S_n)$ and each sentence S_n is a list of words $S_n = (W_1, W_2, \ldots, W_m)$, where W_m is a single word.

The functions sliding(), *map*(), and *reduce*() were implemented. Function *map*() takes list of sentences S and for each S_i executes sliding() function with the parameter $n = (0, 1, 2, \ldots, m)$, where n is size of slides (n-grams) that function will produce and m is number of words W in sentence S_i. Function *reduce*() takes output of *map*() function, which is the list of n-grams (list of list of words) and count similar ones. As a results it returns list of objects (n-gram; v), that is usually called Map, where v is the frequency of particular n-gram in the text. This approach provide ability to execute independent *map*() and avoid communication between nodes until *reduce*() stage.

For our experiments the cluster of 16 nodes was used, each node has the same characteristics. More details about cluster and frameworks could be found in [38] work. **Figure 8** shows results of the conducted experiments. It is clear that parallelization reveal good efficiency and speedup on all three HPC platforms. During our experiments, Apache Hadoop shows low speed and efficiency for a large number of small files. Researches [39] show that Apache Hadoop works faster if input data is represented as a few big files instead of many small files. This is because of HDFS design, which was developed for processing big data streams. Readings of many small files leads to many communications between nodes, many disk head movements and as a consequence leads to extremely inefficient work of HDFS. Details of the comparison could be found in the work [38].

3.2. Parallel text document clustering based on genetic algorithm

This section describes parallel implementation of the text document clustering algorithm. The algorithm is based on evaluation of the similarity between objects in a competitive situation, which leads to the notion of the function of rival similarity. While attributes of bibliographic description of scientific articles were chosen as the scales for determining similarity measure. A genetic algorithm is developed to find the weighting coefficients which are used in the formula of similarity measure. To speed up the performance of the algorithm, parallel computing technologies are used. Parallelization is executed in two stages: in the stage of the genetic algorithm, as well as directly in clustering. The parallel genetic algorithm is implemented with the help of MPJ Express library and the parallel clustering algorithm using the Java 8 Streams library. The results of computational experiments showing benefits of the parallel implementation of the algorithm are presented.

3.2.1. Clustering using the function of rival similarity

FRiS-Tax algorithm described in [40] is chosen as a clustering algorithm. The measure of rival similarity is introduced as follows. In the case of the given absolute value of similarity m(x, y) between two objects, the rival similarity of object a with object b on competition with c is calculated by the following formula:

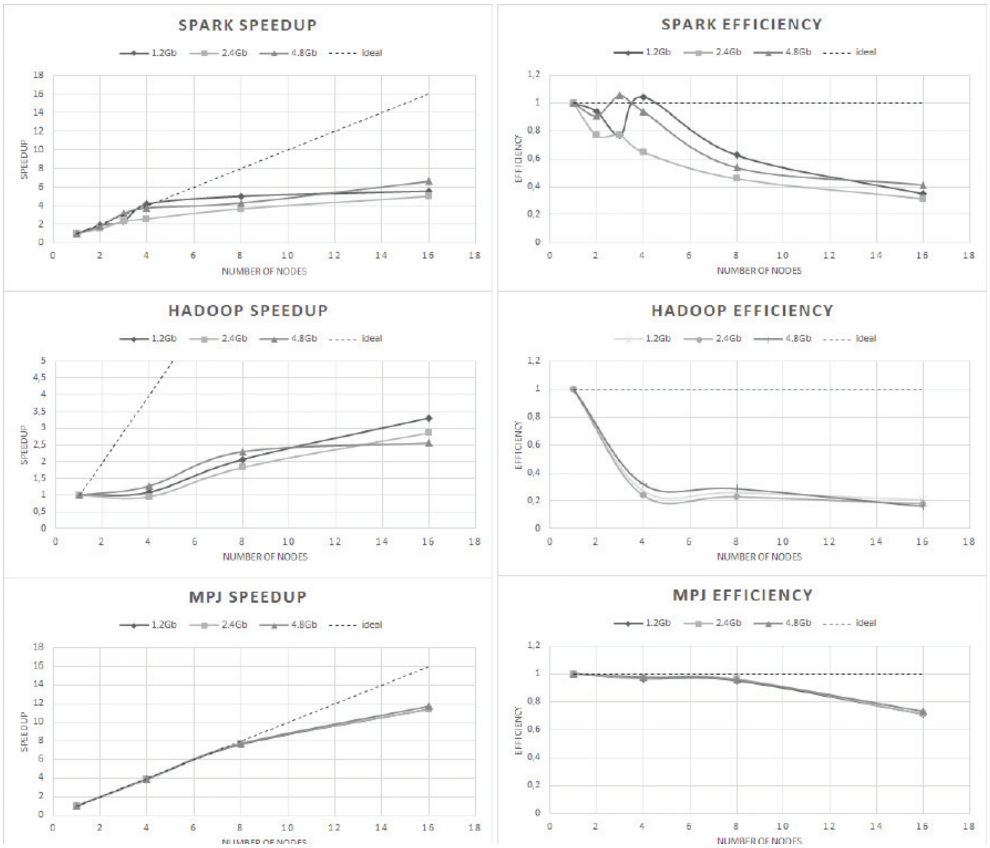

Figure 8. Speedup and efficiency of each platform.

$$F_{b/c}(a) = \frac{\mathrm{m(a, b) - m(a, c)}}{\mathrm{m(a, b) + m(a, c)}} \tag{13}$$

where *F* is called a function of rival similarity or FRiS-function. To measure similarity, the attributes of the bibliographic descriptions of scientific articles were proposed to be taken as scales.

The year of issue; code UDC; key words; authors; series; annotation; title were chosen as attributes of division of articles from bibliographic databases into clusters. A genetic algorithm was developed to choose weighting coefficients which are used in the formula of similarity measure (Eq. (13)). The use of genetic algorithm allows automating the search for the most acceptable weighting coefficients in the formula of similarity measure.

3.2.2. Genetic algorithm for adjustment of coefficients in the formula of similarity measure

To create the initial population of genetic algorithm and its further evolution, it is necessary to have an ordered chain of genes or a genotype. For this task, a chain of genes has a fixed length equal to 13 and presents a set of parameters made up on the basis of attributes of bibliographic description of documents.

In genetic algorithms, the individuals entering the population are presented by ordered subsequent genes or chromosomes with coded in them sets of the problem parameters.

At the stage of selection, the parents of the future individual are determined with the help of methods Roulette Selection, Tournament Selection, and Elitism Selection. The survived individuals take part in reproduction. For crossover operator, the following methods are used: One point crossover, Two point crossover, Uniform crossover, and Variable to Variable crossover. The stage of mutation is necessary not to let the solution of the problem get into a local extremum. It is supposed that, after the crossover is completed, part of the new individuals undergo mutations. In our case, 25% of all individuals are selected which are subjected to mutation. In this work, the quality of the obtained clusters is evaluated using the Purity and Root mean square deviation measures of estimation.

3.2.3. Development of the parallel clustering algorithm

Parallelization is carried out in two stages of the algorithm of clustering. The first stage is occurred during the selection of individuals in the genetic algorithm when clustering is performed with different sets of weighting coefficients. The program is written in Java, and this stage of the parallel algorithm is performed using MPJ Express. Secondly, it is directly in the course of performing the clustering algorithm.

The load test revealed the two slowest stages in the clustering algorithm. They are the methods of finding the first centroid called pillar and finding the next pillar, which are doing $N^*(N\text{-}1)$ and $N^*(N\text{-}1)^*M$ operations, where N is the number of articles and M is the number of already found pillars. To accelerate these methods, the technology *Java* 8 *Streams* was used. Since repeated (N-1) and (N-1)*M times operations in methods finding first and finding next pillar, respectively, are simple and their result need to be summarized at the end, it is reasonable to implement here *parallel*() method of *Java* 8 (**Figure 9**).

For clustering and performance analysis, the journal “Bulletin of KazNU” of 2008–2015 was used as initial data. Sampling includes 95 pdf documents. The total number of articles is 2837. The choice of the initial data is conditioned by the fact that all documents were divided into series (mathematics, biology, philosophy, etc.) and further divisions do not cause difficulties, when using measures of similarity based on only bibliographic descriptions or titles of the articles. In order to evaluate the quality of division of sampling, this body was divided into clusters with the help of an expert into the problem domain.

The time of execution was determined as follows. The measurements of the time of clustering processes were made for the clusters being formed on one computer node and several computer nodes for parallel realization. **Figures 10** and **11** present acceleration and efficiency of parallel realization. As it is seen in the constructed diagrams, with the increase in the number of processes, acceleration increases to a certain value which is related to the expenditure of communication. The most optimum number of processes proved to be eight at which the maximum value of acceleration was observed but the highest value of efficiency was achieved with 4 processes.

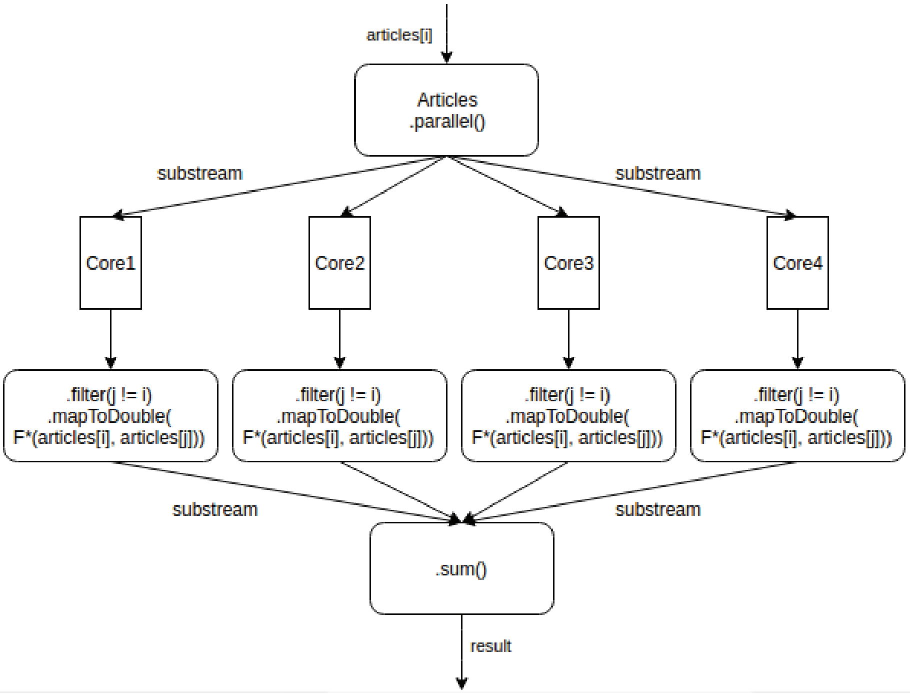

Figure 9. Stream parallelization on 4-core processor, find first pillar.

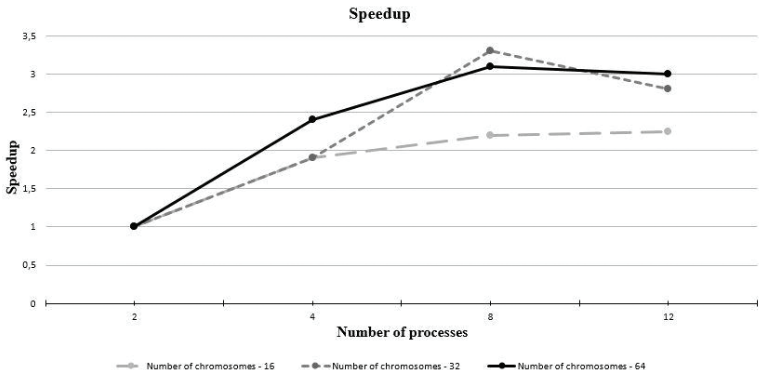

Figure 10. Speedup of parallel clustering algorithm.

It can be concluded that the use of the genetic algorithm allowed to determine the values of attributes at which clustering of documents gives the best results [41].

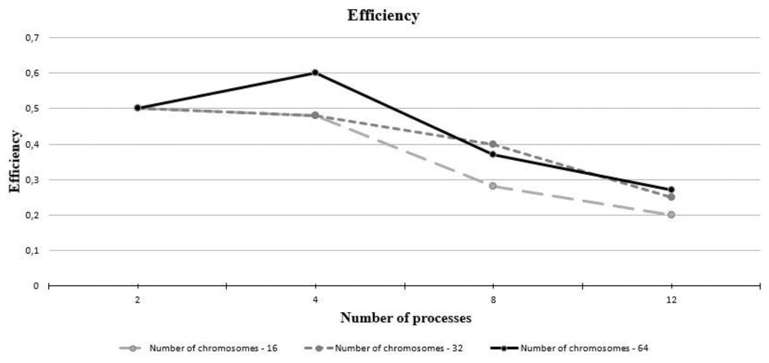

Figure 11. Efficiency of parallel clustering algorithm.

3.3. PGAS approach to implement MapReduce framework based on UPC language

In Section 3.1, the important role of the MapReduce paradigm and distributed file systems in large data processing tasks was emphasized. The weak side of distributed file systems is the considerable time spent on performing read and write operations. In this chapter, an approach to implement MapReduce functionality was described using partitioned global address space model (PGAS). PGAS is a parallel programming model in which memory is divided among threads with certain affinity rules. The affinity is a property that tells how memory is distributed among threads. In some sense, PGAS is considered to be a model that shares the properties of both shared and distributed memory models. The memory is divided in such a way that each thread controls some portion of shared memory region and a private memory which is used to store local to that thread variables. The obvious benefits of using such a model are:

- Transparent view of shared memory by each thread;
- No need to use low-level message passing techniques to exchange data between threads.

The implementation of MapReduce using PGAS approach is based on using hashmap data structure. Hashmap data structure is used to store key/value pairs generated during MapReduce execution. The main idea is that array containing hashmap entries is created in a global shared memory space. Array is distributed in such a way that each array entry correspond to exactly one thread. Since, hashmap is located in a global shared memory region each thread can view and modify/read the hashmap entries of the other threads. This way data exchange for the MapReduce (see **Figure 12**) can easily be implemented by just exchanging and distributing corresponding key/value pairs among different threads. For each thread to decide set of keys to be processed in reduce stage the problem of key distribution was formulated.

The problem of key distribution among threads after map stage has been stated in the following way:

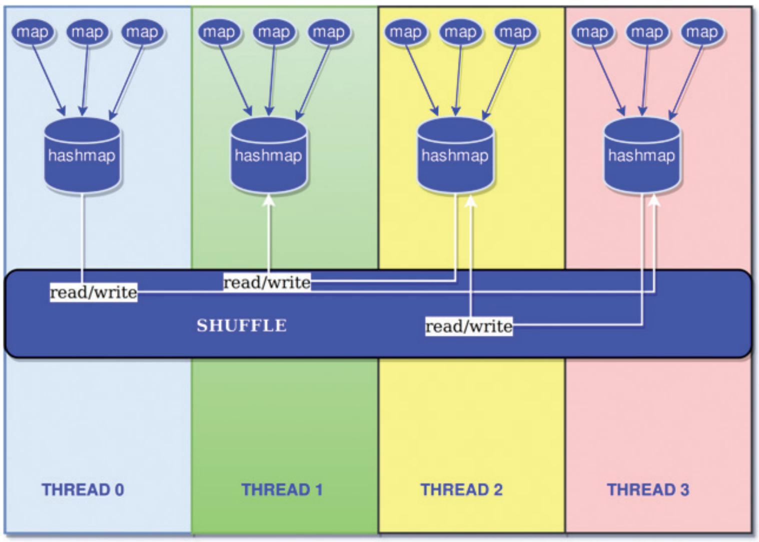

Figure 12. Data exchange mechanism for MapReduce using PGAS approach.

$$min \sum_{i=0}^{threads-1} \sum_{j=1}^{keys} x_{ij} \times cost_{ij} \tag{14}$$

$$x_{ij} \in \{0,1\} \tag{15}$$

$$min\left(\max_{i,\, j=0..threads-1} \left| load_i - load_j \right| \right) \tag{16}$$

$$load_i = \sum_{t=0}^{threads-1} \sum_{j=1}^{keys} x_{ij} \times size_{tj} \tag{17}$$

Finding the cost of assigning key j to thread i is done by building the *cost* matrix. The quantity $cost_{ij}$ represents the cost of moving key j to thread i. This value is defined to be a number of elements with certain key to be moved from other threads to the thread with an index of i. Keys need to be distributed in such a way that Eqs. (14) and (16) are satisfied. Eq. (15) specifies the domain of x_{ij}. The value of x_{ij} is equal to zero when thread i is not assigned to process key j and $x_{ij} = 1$ otherwise. Load balancing function is defined in Eq. (16) and can be computed as a minimum of maximal difference of loads assigned to any pair of threads. Load for each thread i is defined in Eq. (17) [42].

Since the described problem of key distribution is proven to be NP-hard, finding the optimal distribution even for a small set of keys is a computationally very expensive task. Therefore, a heuristic genetic algorithm was used that tries to find a close approximation to the optimal result.

The MapReduce framework has been tested on WordCount application (see **Figure 13**). WordCount application is used to compute number of occurrences of each word in a collection of documents. This is a standard benchmark application to test performance of different parallel tools in Big Data domain. In **Figure 14** the results of Apache Hadoop versus MapReduce on UPC for WordCount application is presented.

```
void * map (string filename)
{
char * file_data;
file_data = read_file_contents (filename);
Vector tokens;
vector_init(&tokens);
Tokenize (file_data,&tokens);
for (int i = 0;i < tokens.size;i++)
{collect (vector_get (&tokens,i),1);}
free(file_data);}
```

```
void reduce (string key,shared [] vector_sh *values)
{
int i;
int cnt = 0;
for (i = 0;i < values- > size;i++)
{int v = vector_get_shared_copy (values,i);
cnt + =v;}
reduce_collect (key,cnt);}
```

The presented MapReduce framework was developed using UPC parallel programming language which belongs to a family of PGAS languages. The overall obtained performance

```
void * map (string filename)
{
char * file_data;
file_data = read_file_contents (filename);
Vector tokens;
vector_init(&tokens);
Tokenize (file_data,&tokens);
for (int i = 0;i<tokens.size;i++)
{collect (vector_get (&tokens,i),1);}
free(file_data);}
```

```
void reduce (string key,shared [] vector_sh
*values)
{
   int i;
      int cnt = 0;
      for (i = 0;i<values->size;i++)
      {    int v = vector_get_shared_copy
(values,i);
            cnt+=v;    }
      reduce_collect (key,cnt);}
```

Figure 13. Implementation of map and reduce functions for WordCount application.

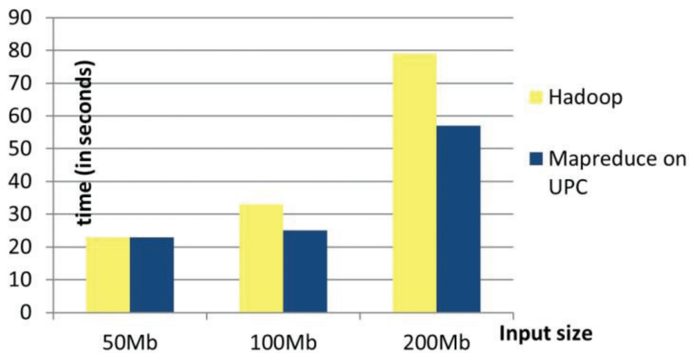

Figure 14. Apache Hadoop versus MapReduce on UPC.

and programmability benefits allow efficiently using this system for MapReduce based data processing tasks.

4. Conclusion

This chapter discusses high-performance computational and information technologies for numerical models and data processing. As a numerical model the oil recovery problem was considered. New fragmented algorithm was proposed, the algorithm for high-performance visualization and the algorithm on mobile platforms to solve this problem. Study of efficiency of applied algorithm implementations show that LuNA system appears to be less efficient than manual MPI implementation which justifies further development of LuNA functionality considering simplicity of software development with given system.

The described system contains the specialized visualization module implemented using Vulkan API. Given technology provides high-performance capabilities which were demonstrated using common desktop PC on a generated dataset.

The textual data processing problems as n-gram extraction and data clustering were also studied. In order to deal with computational complexity we had to adopt and implement parallelization patterns. Additionally, a new implementation of MapReduce framework was presented based on UPC language which provides functionality of combined MapReduce and partitioned global address space parallel programming models in a single execution environment which can be conveniently used in many complex workflows of data processing.

Author details

Darkhan Akhmed-Zaki, Madina Mansurova*, Timur Imankulov, Danil Lebedev, Olzhas Turar, Beimbet Daribayev, Sanzhar Aubakirov, Aday Shomanov and Kanat Aidarov

*Address all correspondence to: mansurova.madina@gmail.com

Al-Farabi Kazakh National University, Almaty, Kazakhstan

References

[1] Lake LW. Enhanced Oil Recovery. New Jersey: Prentice Hall Inc; 1989

[2] Sorbie KS. Polymer Improved Oil Recovery. Boca Raton: CRC Press; 1991

[3] Malyshkin V, Perepelkin V. Optimization methods of parallel execution of numerical programs in the LuNA fragmented programming system. The Journal of Supercomputing, Springer. 2012;**61**(1):235-248

[4] Vulkan. Industry Froged. Available from: https://www.khronos.org/vulkan/ [Accessed: 2017–05-01]

[5] BOINC - Open-Source Software for Volunteer Computing and Grid Computing. 2017. Available from: http://boinc.berkeley.edu/ [Accessed: 2017-05-01]

[6] Zhao D. Fast filter bank convolution for three-dimensional wavelet transform by shared memory on mobile GPU computing. The Journal of Supercomputing. 2015;**71**(9):3440-3455

[7] Montella R, Giunta G, Laccetti G, Lapegna M, Palmieri P, Ferraro C, Pelliccia V, Hong C, Spence I, Nikolopoulos D. On the virtualization of CUDA based GPU remoting on ARM and X86 machines in the GVirtuS framework. International Journal of Parallel Programming. 2017;**45**(5):1142-1163

[8] Lewis DD. Feature selection and feature extraction for text categorization. In: Proceedings of the Workshop on Speech and Natural Language. Harriman, New York: Association for Computational Linguistics; 1992. pp. 212-217

[9] Mikolov T, et al. editors. CoRR. 2013. arXiv: 1301.3781. Available from: http://arxiv.org/abs/1301.3781 [Accessed: 2017-05-01]

[10] Joulin A, et al. editors. Bag of Tricks for Efficient Text Classification. CoRR. 2016. arXiv: 1607.01759. Available from: http://arxiv.org/abs/1607.01759. [Accessed: 2017-05-01]

[11] Mikolov T, et al. editors. Distributed Representations of Words and Phrases and their Cmpositionality. CoRR. 2013. arXiv: 1310.4546. [Accessed: 2017-05-01]

[12] Whitley D, Sutton AM. Genetic algorithms. A survey of models and methods. In: Handbook of Natural Computing. Springer Berlin Heidelberg; 2012. pp. 637-671

[13] Bandyopadhyay S, Maulik U. Genetic clustering for automatic evolution of clusters and application to image classification. Pattern Recognition. 2002;**35**:1197-1208

[14] Gajawada S., D Toshniwal, N Patil, K Garg. Optimal clustering method based on genetic algorithm. In: Proceedings of the International Conference on Soft Computing for Problem Solving. December 20–22; 2011. pp. 295-303

[15] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM. 2008;**51**(1):107-113

[16] Dean J, Ghemawat S. Mapreduce: A flexible data processing tool. Communications of the ACM. 2010;**53**(1):72-77

[17] Bu Y, Howe B, Balazinska M, Ernst MD. The HaLoop approach to large-scale iterative data analysis. VLDB Journal. 2012;**21**(2):169-190

[18] Ekanayake J, Li H, Zhang B, Gunarathne T, Bae S-H, Qiu J, Fox G. Twister: A runtime for iterative MapReduce. In: Proceedings of the HPDC 2010. Chicago, IL, USA: ACM; 2010. pp. 810-818

[19] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. Boston, MA, USA: ACM; 2010. pp. 10-10

[20] Talbot JM, Yoo R, Kozyrakis C. Phoenix++: modular MapReduce for shared-memory systems. In: Proceedings of the Second International Workshop on MapReduce and its Applications. San Jose, California, USA: ACM; 2011. pp. 9-16

[21] Teijeiro C, Taboada GL, Tourino J, Doallo R. Design and implementation of MapReduce using the PGAS programming model with UPC. In: Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS '11). Tainan, Taiwan, 2011. pp. 196-203

[22] Dong H, Zhou S, Grove D. X10-enabled MapReduce. In: Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model (PGAS '10). New York, USA: ACM; 2010. p. 6

[23] Babalyan GA, Levy BI, Tumasyan AB, Khalimov EM. Oilfield development using surfactants. Moscow: Nedra; 1983. p. 98 (In Russian)

[24] Danaev N, Akhmed-Zaki D, Mukhambetzhanov S, Imankulov T. Mathematical modelling of oil recovery by polymer/surfactant flooding. Communications in Computer and Information Science. 2015:1-15

[25] Samarskii AA. Numerical methods. Moscow: Nauka; 1989. p. 432 (In Russian)

[26] Douglas J, Rachford HH. On the numerical solution of heat conduction problems in two and three space variables. Transactions of the American Mathematical Society. 1956;**82**(2): 421-439

[27] Samarskii AA. Theory of difference schemes: a tutorial. Moscow: Nauka; 1977. p. 656 (In Russian)

[28] Web site of the Interagency Supercomputer Center of the Russian Academy of Sciences. Available from: http://www.jscc.ru/scomputers.html [Accessed: 22.09.2017]

[29] Malyshkin VE, Perepelkin VA. LuNA fragmented programming system, main functions and peculiarities of run-time subsystem. In: Proceedings of the 11-th Conference on Parallel Computing Technologies, LNCS 6873. 2011. pp. 53–61

[30] AMD Mantle. Available from: http://www.amd.com/en-us/innovations/software-technologies/technologies-gaming/mantle [Accessed: 22.09.2017]

[31] DirectX 12. Available from: https://blogs.msdn.microsoft.com/directx/2014/08/12/directx-12-high-performance-and-high-power-savings/ [Accessed: 22.09.2017]

[32] Akhmed-Zaki DZ, Imankulov TS, Matkerim B, Daribayev BS, Aidarov KA, Turar ON. Large-scale simulation of oil recovery by surfactant-polymer flooding. Eurasian Journal of Mathematical and Computer Applications. 2016;**4**(1):12-31

[33] Akhmed-Zaki DZ, Daribayev BS, Imankulov TS, Turar ON. High-performance computing of oil recovery problem on a mobile platform using CUDA technology. Eurasian Journal of Mathematical and Computer Applications. 2017;**5**(2):4-13

[34] Cook Sh. CUDA Programming. A Developer's Guide to Parallel Computing with GPUs. Morgan Kaufmann: 2012. 600 p.

[35] NVIDIA Tegra X1 the new level of mobile performance. Available from: www.nvidia.com/object/tegra-x1-processor.html [Accessed: 22.09.2017]

[36] Brants T, Popat AC, Xu P, Och FJ, Dean J. Large Language Models in Machine Translation. In: Proceedings of the JCSSE; June 2007; Prague. pp. 858–867

[37] Berberich K, Bedathur S. Computing n-gram statistics in MapReduce. In: Proceedings of the 16th International Conference on Extending Database Technology (EDBT '13); 18–22 March 2013; Genoa. New York: ACM. pp. 101-112

[38] Aubakirov S, Trigo P, Ahmed-Zaki D. Comparison of distributed computing approaches to complexity of n-gram extraction. In: Proceedings of DATA 2016: 5th International Conference on Data Management Technologies and Applications. Lisbon: SCITEPRESS; 24-26 July. pp. 25-30

[39] Andrews BP, Binu A. Perusal on Hadoop small file problem. IJCSEITR. 2013;**3**(4):221-226

[40] Barakhnin VB, Nekhaeva VA, Fedotov AM. On the statement of the similarity measure for the clustering of text documents. Bulletin of Novosibirsk State University Series: Information Technology. 2008;**6**(1):3-9 (in Russian)

[41] Mansurova M, Barakhnin V, Aubakirov S, Khibatkhanuly E, Musina A. Parallel text document clustering based on genetic algorithm. In: Proceedings of the International Conference Mathematical and Information Technologies (MIT-2016); 28 August – 5 September 2016, Vrnjacka Banja. p. 218–232

[42] Shomanov AS, Akhmed-Zaki DZ, Mansurova ME. PGAS Approach to Implement Mapreduce Framework Based on UPC Language. In: Malyshkin V, editor. Parallel Computing Technologies. PaCT 2017. Lecture Notes in Computer Science. Vol. 10421. Cham: Springer. pp. 133-137