DOI: 10.15514/ISPRAS-2019-31(3)-17

«Life» in Tensor: Implementing Cellular Automata on Graphics Adapters

N.A. Shalyapina, ORCID: 0000-0001-8742-4903 <nat.shalyapina@gmail.com>
M.L. Gromov, ORCID: 0000-0002-2990-8245 <maxim.leo.gromov@gmail.com>
National Research Tomsk State University,
36 Lenin Avenue, Tomsk, 634050, Russia

Abstract. This paper presents an approach to the description of cellular automata using tensors. This approach allows to attract various frameworks for organizing scientific calculations on high-performance graphics adapter processors, that is, to automatically build parallel software implementations of cellular automata. In our work, we use the TensorFlow framework to organize computations on NVIDIA graphics adapters. As an example cellular automaton we used Conway's Game of Life. The effect of the described approach to the cellular automata implementation is estimated experimentally.

Keywords: Cellular Automata; Conway's Game of Life; Tensor

For citation: Shalyapina N.A., Gromov M.L. «Life» in Tensor: Implementing Cellular Automata on Graphics Adapters. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 3, 2019. pp. 217-228. DOI: 10.15514/ISPRAS-2019-31(3)-17

«ЖИЗНЬ» в тензорах: реализация клеточных автоматов на видеокартах

Н.А. Шаляпина, ORCID: 0000-0001-8742-4903 <nat.shalyapina@gmail.com>
М.Л. Громов, ORCID: 0000-0002-2990-8245 <maxim.leo.gromov@gmail.com>
Национальный исследовательский Томский государственный университет,
634050, Россия, г. Томск, пр. Ленина, д. 36

Аннотация. В данной статье представлен подход к описанию клеточных автоматов с использованием тензоров. Такой подход позволяет привлекать различные фреймворки для организации расчетов на высокопроизводительных графических видеокартах, т.е. для автоматического построения параллельных программных реализаций клеточных автоматов. В нашей работы мы используем фреймворк TensorFlow для организации вычислений на графических видеокартах NVIDIA. В качестве примера клеточного автомата мы рассмотрели игру «Жизнь». Эффект от описанного подхода к программной реализации клеточных автоматов оценён экспериментально.

Ключевые слова: клеточный автомат; игра «Жизнь»; тензор

Для цитирования: Шаляпина Н., Громов М. «ЖИЗНЬ» в тензорах: моделирование клеточных автоматов с помощью видеокарт. Труды ИСП РАН, том 31, вып. 3, 2019 г., стр. 217-228 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(3)-17

1. Introduction

The use of automata in description of a dynamic systems' behavior has been known for a long time. The key point of this approach to the description of systems is a representation of the object under study in the form of a discrete automatic device – automaton (State Machine or Transition System).

Under the influence of input sequences (or external factors) an automaton changes its state and produces reactions. There are many types of such automata: the Moore and Mealy machines [1], the cellular automaton [2], and others. The knowledge of the features of the object under study can provide enough information to select the appropriate type of automaton for the object's behavior description. In some cases, it is convenient to use an infinite model. But finite models are mostly common. In the latter case, the sets of states, input actions (or states of the environment), and output reactions are finite.

Our work deals with *cellular automata* (CA). The theory of cellular automata began to take shape quite a long time ago. The work of John von Neumann [3] might be considered as the first work of the cellular automata theory. Today, a large number of studies devoted to cellular automata are known [4, 5]. Note that a major part of these works is devoted to the simulating of spatially distributed systems in physics, chemistry, biology, etc. [6]. The goal of the simulation is to find the states of the cells of a CA after a predetermined number of CA cycles. The resulting set of states in some way characterizes the state of the process or object under study (fluid flow rate at individual points, concentration of substances, etc.). Thus, the task of simulating a certain process or object by a cellular automaton can be divided into two subtasks. First, the researcher must select the parameters of the automaton (the dimension of the grid of cells, the shape of the cells, the type of neighborhood, etc.). And secondly, programmatically implement the behavior of the selected cellular automaton. Our work is focused on the second task – the software implementation of the cellular automaton.

In itself, the concept of a cellular automaton is quite simple and the idea of software implementation is obvious. However, the number of required calculations and the structure of these calculations suggest the use of modern supercomputers with a large number of cores and supporting large-block parallelism. In this case, the cell field of the automaton is divided into separate blocks. Processing of blocks is done in parallel and independently from each other. At the end of each processing cycle, the task of combining the processing results of each block arises. This problem was solved in [7] in the original way. The experimental study in [7] of the efficiency of parallelization was carried out on clusters with 32 and 768 processors. Despite the high effectiveness of this approach, it has some issues. First, this approach assumes that a researcher has an access to a cluster. Supercomputers are quite expensive and usually are the property of some collective access center [8]. Of course, after waiting a certain time in the queue, access to the cluster is possible. But another difficulty arises: a special skill is needed to write parallel programs in order to organize parallel sections of the program correctly. And this leads to the fact that it takes a certain number of experiments with the program to debug it before use. The latter means multiple times of waiting in a queue for a cluster, which, of course, delays the moment of launching actual (not debugging) experiments with cellular automata.

We offer an alternative approach for software implementation of cellular automata, which is based on the use of modern graphics adapters. Modern graphics adapters are also well-organized supercomputers, consisting of several specialized computational cores and allowing execution of operations in parallel. Compared to clusters, graphics adapters are available for a wide range of users and we believe that their capabilities are enough to implement cellular automata. In addition, there are special source development kits or frameworks (for example, ThensorFlow [9]) that can exploit multi-core graphics adapters and help a researcher quickly and efficiently create a software product, without being distracted by thinking about parallelizing data flows and control flows. In this paper, we demonstrate an approach to implementation of cellular automata on graphics adapters based on TensorFlow.

In order to use this tool, we propose to describe the set of states of an automaton cells' by the main data structure used in this framework, namely, the *tensor*. Then we describe the process of evolution of the automaton in terms of tensor operations. A well-known cellular automaton, the Conway's Game of Life, is used as a working example.

The paper is structured as follows. Section 3 presents the basic concepts and definitions concerning the theory of cellular automata. Section 3 provides a description of the game Conway's Game of

Life, its features and rules of operation. Section 4 is devoted to a detailed presentation of the proposed approach for software implementation of cellular automata on graphics adapters. The results of computer experiments with the implementation of the Conway's Game of Life and comparison with the results of a classical sequential implementation are presented in section 5.

2. Preliminaries

The Moore machine (finite, deterministic, fully defined) is a 6-tuple $A = \langle S, \hat{s}, I, O, \varphi, \psi \rangle$, where S is the finite nonempty set of states of the machine with a distinguished initial state $\hat{s} \in S$, I is the finite set of input stimuli (input signals), O is a finite set of output reactions (output signals). $\varphi: S \times I \to S$ is a fully defined transition function, $\psi: S \to O$ is a fully defined function of output reactions. If at some moment of time the Moore machine $\langle S, \hat{s}, I, O, \varphi, \psi \rangle$ is at the certain state $s \in S$ and the input signal $i \in I$ arrives, then the machine changes its state to the state $s' = \varphi(s, i)$, and the signal $o = \psi(s')$ appears at its output. The machine starts its operation from the initial state $\hat{s}$ with the output signal $\psi(\hat{s})$. It is important to note that originally Moore defined the machine so that the output signal of the machine is determined not by the final state of the transition, but by the initial one (i.e. in the definition above instead of $o = \psi(s')$ should be $o = \psi(s)$). However, for our purposes it is more convenient to use the definition we have specified.

Let $\mathbb{Z}$ be the set of integers. Consider the set of all possible integers pairs $(i, j) \in \mathbb{Z} \times \mathbb{Z}$. With each pair (i, j) we associate some finite set of pairs of integers $N_{i,j} \subseteq \mathbb{Z} \times \mathbb{Z}$, called *the neighborhood of the pair* (i, j). Pairs of $N_{i,j}$ will be called *neighbors* of the pair (i, j). The sets $N_{i,j}$ must be such that the following rule holds: if the pair (p, q) is the neighbor of the pair (i, j), then the pair $(p - k, q + l)$ is the neighbor of the pair $(i + k, j + l)$, where k and l are some integers. Note that the cardinalities of all neighborhoods coincide and the sets will have the same structure. For convenience, we assume that all neighbors from $N_{i,j}$ are enumerated with integers from 1 to $|N_{i,j}|$, where $|N_{i,j}|$ is the cardinality of the set $N_{i,j}$. Then we can talk about the *first*, *second*, etc. *neighbor* of some pair (i, j). If the pair (p, q) is the n-th neighbor of the pair (i, j), then the pair $(p - k, q + l)$ is the n-th neighbor of the pair $(i + k, j + l)$.

Consider the set of Moore machines of the form $A_{i,j} = \langle S, \hat{s}_{i,j}, S^{|N_{i,j}|}, S, \varphi, \psi \rangle$ such that $\psi(s) = s$. Here i and j are some integers, B^n is the n-th Cartesian power of the set B. The machines corresponding to the neighbors of the pair (i, j) are called *neighbors* of the machine $A_{i,j}$. Neighboring machines will be numbered as well as the corresponding neighboring pairs (that is, the first neighbor, the second, etc.). We specifically note that (i) for each machine $A_{i,j}$ the set of states is the same, i.e. S; (ii) for each machine $A_{i,j}$, the set of output signals coincides with the set of states, that is, also S; (iii) as an output signal, the machine gives its current state; (iv) all machines have the same transition function and the same function of output reaction; (v) as an input signal, machines take tuples of states (of their neighbors), the number of elements in the tuple coincides with the number of neighbors, that is, equals to $|N_{i,j}|$; (vi) machines differ only in their initial states. Let at a given time moment the current state of the first neighbor of the machine $A_{i,j}$ is equal to s_1, the state of the second neighbor is s_2, ..., the state of the n-th neighbor is s_n, where $n = |N_{i,j}|$. Then the tuple $(s_1, s_2, ..., s_n)$ is the input signal of the machine $A_{i,j}$ at this very moment. All machines accept input signals, change their states and provide output signals simultaneously and synchronously. That is, some global clock signal is assumed.

The resulting set $\{A_{i,j} | (i, j) \in \mathbb{Z} \times \mathbb{Z}\}$ of the Moore machines is called a *two-dimensional synchronous cellular automaton* (or simply *cellular automaton* – CA). Each individual Moore machine of this set will be called a *cell*. The set of states of all cells the CA at a given time moment will be called *the global state of the cellular automaton at this time moment*.

The transition rules of cells from one state to another (the function φ), the type of neighborhood of the cells (the sets $N_{i,j}$), the number of different possible cell states (the set S) define the whole variety of synchronous two-dimensional cellular automata.

For clarity, one can draw cellular automata on the plane. For this, the plane is covered with figures. Coverage can be arbitrary, but of course, it is more convenient to do it in a regular way. Classic covers are equal squares, equal triangles and equal hexagons. The choice of one or another method of covering the plane is dictated by the original problem a CA is used for and the selected set of neighbors. Next, the cover figures are assigned to the cells of the cellular automaton in a regular manner. For example, let the plane be covered with equal squares, so that each vertex of each square is also the vertex of the other three squares of the coverage (fig. 1a). Choose the square of this coverage randomly and associate it with the cell $A_{0,0}$. Let the cell $A_{i,j}$ be associated with a certain square. Then we associate the cell $A_{i+1,j}$ with the square on the right, the cell $A_{i-1,j}$ with the square on the left, the cell $A_{i,j+1}$ with the square above, and the cell $A_{i,j-1}$ with the square below (fig. 1b). Cell states will be represented by the color of the corresponding square (fig. 1c)

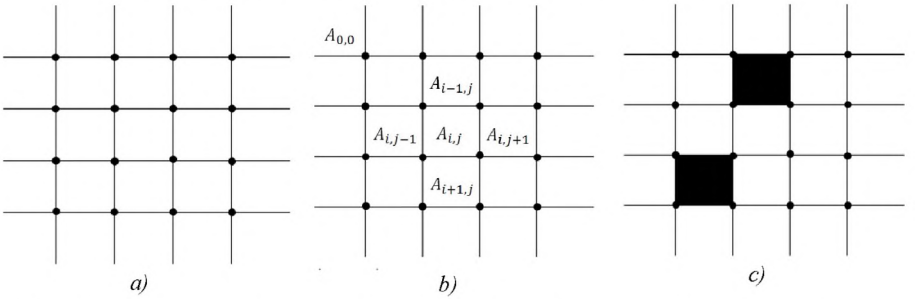

Fig. 1. A CA represented on a plane covered by equal squares: a) the coverage of the plane; b) association of the cells with the squares; c) colour representation of cells' states (for the case |S|=2, «white» – state 0, «black» – state 1)

The resulting square based representation of a CA on a plane is classical one. In our work we consider only this representation.

For the square based representation of a CA, the neighborhoods shown in fig. 2 are the most common.

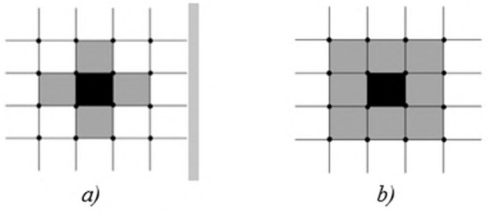

Fig. 2. The neighborhood (grey cells) of a cell (the black one) by a) von Neumann, b) Moore Geometric figures

If a given cellular automaton models a process (for example, heat transfer), then the various global initial states $\{s_{i,j} | (i,j) \in \mathbb{Z} \times \mathbb{Z}\}$ of the cellular automaton correspond to different initial conditions of the process. According to the definition of cellular automata introduced by us, the set of cells in it is infinite. However, from the point of view of practice, especially in the case of an implementation of a cellular automaton, a set of cells have to be made finite. In this case, some of the cells lack some neighbors. Therefore, for them the set of neighbors and the transition function are modified. Such modifications determine the boundary conditions of the process being modeled.

3. Conway's Game of Life

In the 70s of the 20th century, the English mathematician John Conway proposed a cellular automaton called the Conway's Game of Life [10].

The cells of this automaton are interpreted as biological cells. The state «0» corresponds to the «dead» cell, and the state «1» – «alive». The game uses the Moore's neighborhood (Fig. 2b), i.e. each cell has 8 neighbors. The rules for the transition of cells from one state to another are as follows:

- if a cell is «dead» and has three «alive» neighbors then it becomes «alive»;
- if a cell is «alive» and has two or three «alive» neighbors then it remains «alive»;
- if a cell is «alive» and has less than two or more than three «alive» neighbors then it becomes «dead».

For the convenience of perception, the behavior of each cell of the cellular automaton Conway's Game of Life can be illustrated using the transition graph (fig. 3).

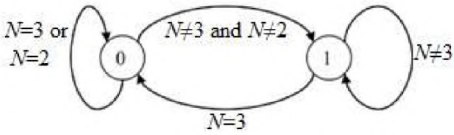

Fig. 3. Cell Transition Graph of the Conway's Game of Life, where N is the number of «alive» neighbors
Geometric figures

Despite the simplicity of the functioning of the automaton, it is an object for numerous studies, since the variation of the initial configuration leads to the appearance of various images of its dynamics with interesting properties. One of the most interesting among them are moving groups of cells – gliders. Gliders not only oscillate with a certain periodicity, but also move through the space (plane). Thus, as a result of experiments, it was established that on the basis of gliders logical elements AND, OR, NOT can be built. Therefore any other Boolean function can be implemented. It was also proved that using the cellular automata Conway's Game of Life it is possible to emulate the operation of a Turing machine.

4. Features of Conway's Game of Life Parallel Implementation

According to our definition, a set of states of a cell is finite. It is obvious that, in this case, without loss of generality, we can assume that the set of states is the set of integers from 0 to $|S|$ - 1, where $|S|$ – is the cardinality of the set of states. Therefore, the global state of the cellular automaton can be represented as a matrix $\boldsymbol{A}$. The element $\boldsymbol{A}_{i,j}$ of this matrix is equal to the current state of the cell $\boldsymbol{A}_{i,j}$. We call the matrix $\boldsymbol{A}$ *the matrix of the global state* of the cellular automaton. If there are no restrictions on the number of cells, then matrix $\boldsymbol{A}$ will be infinite. As have already been mentioned, the number of cells has to be limited from a practical point of view, that is, it is necessary to somehow choose the finite subset of cells. After that, only selected cells are considered. In this case, the ability to describe the global state of the cellular automaton by the matrix is determined by which cells are selected. We assume that the following set of cells is selected: $\{A_{i,j} | (1 \le i \le m) \wedge (1 \le j \le n)\}$, where m and n – two fixed natural numbers. In this case, the global state matrix is obtained naturally.

Since we use the TensorFlow framework for implementation of a CA, we should work with concepts defined in it. The main data structure in TensorFlow is a multidimensional matrix which in terms of this framework is called a tensor. However, in many cases, such a matrix may not correspond to any tensor. The tensor in the n-dimensional space must have n^{p+q} components and is represented as $(p + q)$-dimensional matrix, where (p, q) is the rank of the tensor. And, for example, a 2 by 3 matrix does not follow these restricions. But the convenience of data manipulation provided by the framework justifies some deviations from strictly defining the tensor. Therefore, in the case when we are talking about the software implementation of a cellular automaton using TensorFlow, we will

consider the notion of the global state matrix of a CA and the notion of *the global state tensor of a CA* as equivalent.

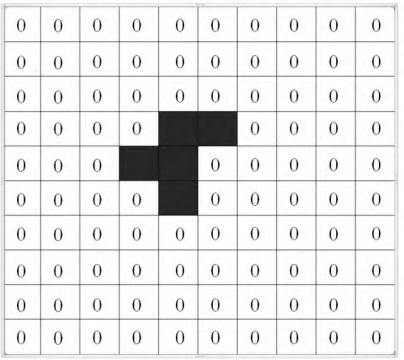

Fig. 4. Some initial global state of the finite state machine for the Conway's Game of Life

Thus, the evolution of the global state of a cellular automaton can be represented (within TensorFlow) as a transformation of the components of the global state tensor. Such a transformation will be called *the evolution of the tensor*.

Thus, the logic of the transition of the cellular automaton from a given global state to the next global state will be described using operations on tensors. In particular, for the software implementation of Conway's Game of Life in our work such operations are the convolution of tensors and the "restriction" of the components value. Let us consider a small example.

Let some initial global state of the cellular automaton (fig. 4) be given.

Black cells are a «alive» cell (state 1), zero means that the cell is «dead» (state 0). The corresponding tensor of the global state has the form (1):

$$\boldsymbol{T} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

The next state of a cell of the cellular automaton of the Conway's Game of Life depends on the number of living neighbors of this cell. We suggest using convolution to count the number of living neighbors of a cell. Since set of neighbors in the Conway's Game of Life are specified by the Moore neighborhood, the convolution kernel will have the form (2):

$$\boldsymbol{S} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0{,}5 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (2)$$

Note the special role of the element $\boldsymbol{S}_{22} = 0{,}5$. This element corresponds to the cell for which the number of living neighbors is calculated. Let the number of living neighbors of a certain dead cell be calculated. Then it will turn out to be integer because component $\boldsymbol{S}_{22}$ will be multiplied by the state of the dead cell (and it is equal to 0), and in the sum the number $\boldsymbol{S}_{22}$ will not participate. It will turn out to be half-integer in the case when the number of living neighbors of a living cell is calculated. This is important when the cell has two living neighbors. Then the dead cell must remain dead, and the living cell must live. That is, if after the convolution the counted number of living

neighbors turns out to be 2 (the cell is dead, it has 2 living neighbors), then in its place should be 0 in the tensor of the global state of the cellular automaton in the next cycle. If, after convolution, the counted number of living neighbors is 2.5 (the cell is alive, and it has 2 neighbors), then in its place should be 1 in the tensor of the global state of the cellular automaton in the next cycle.

Constructing a convolution with the kernel $\boldsymbol{S}$ of the tensor $\boldsymbol{T}$, we obtain the new tensor $\boldsymbol{C}$, where at the intersection of the i-th row and j-th column there is an element corresponding to the number of living neighbors for the cell $\boldsymbol{A}_{i,j}$. Note that we obtain a tensor $(m-2)\times(n-2)$ when constructing a convolution with a kernel of size 3×3 of an arbitrary tensor of the size $m\times n$. In order to save the initial dimensions of the global state tensor of a cellular automaton, we will set the elements in the first and last row and in the first and last column of the global automaton tensor to 0. We will append these zero rows and columns to the result after the convolution is completed. Appended elements in the formula (3) are highlighted in gray. The mentioned fact suggests that some of the subsequent computations are superfluous (namely computations on the appended elements). The amount of extra computations for the global state tensor with dimensions $m\times n$ will be $(2m-2)+(2n-2)$. Then, the part of extra computations in the amount of useful computations is $\frac{(2m-2)+(2n-2)}{(m-1)(n-1)} = O(\frac{1}{m}+\frac{1}{n})$.

$$C=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 3,5 & 2,5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3,5 & 4,5 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 2,5 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (3)$$

Taking into account the agreement on the half-integer value of the number of living neighbors, the integer part of the value of the tensor component $\boldsymbol{C}$ determines the number of living neighbors of the cell, and the presence of the fractional part means that the cell was alive in the previous step.

According to the rules of the Conway's Game of Life it is necessary to transform the tensor $\boldsymbol{C}$ in order to determine the global state of the cellular automaton in the next step. Components with values in the range [2.5, 3.5] should take the value 1 (cells are alive). The remaining components should become 0 (cells are dead). Among the classical operations on tensors there is no operation that would allow to express the required transformation. However, the framework used in our work was created primarily for the problems of the theory of artificial intelligence, namely, for implementation of neural networks. The data flow there is the flow of tensors (a tensor as an input, a tensor as an output). Computational elements, that change data, are layers of the neural network.

So, for example, in our case for the convolution we use a two-dimensional convolution layer with the kernel $\boldsymbol{S}$ (formula (2)). Any tool for neural network implementation ought to have the special type of layers – activation layers (layer of non-linear transformations). These layers calculate activation functions (some non-linear functions) of each element of the input tensor and put the result into the output tensor. TensorFlow offers a standard set of non-linear activation functions. In addition, it is possible to create custom activation functions. We built our own activation function based on a function from a standard set of functions, called a *Rectified Linear Unit (ReLU)*. The function *ReLU* is defined as follows (formula (4)). Its graph is shown in fig. 5:

$$ReLU = max(0, x) \quad (4)$$

Taking into account the required transformation of the components of the tensor $\boldsymbol{C}$ described above, we suggested the function presented in (5):

$$\delta = ReLU\big(4(x-2{,}125)\big) - ReLU\big(4(x-2{,}125)\big) - ReLU\big(4(x-2{,}125)\big) + ReLU\big(4(x-2{,}125)\big) \quad (5)$$

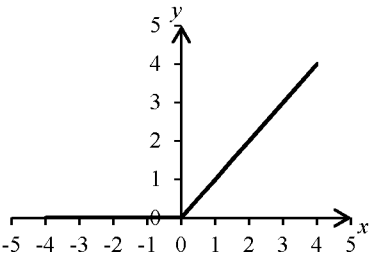

Fig. 5. Diagram of ReLU function

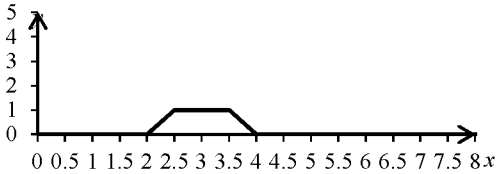

Fig. 6. Diagram of the transition function of the Conway's Game of Life

As a result of applying the function δ to each component of the tensor C, the tensor of the global state of the cellular automaton will take the following form (formula (6)).

$$T' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (6)$$

Thus, the software implementation of the Conway's Game of Life using TensorFlow is a two-layer neural network. The first layer is convolutional, with the kernel from formula (2). The second layer is the activation layer with the activation function from formula (5).

5. Experimental results

We have implemented the described approach for the cellular automaton of the Conway's Game of Life in Python. Since there was no one in our group familiar with TensorFlow, but we have some experience in Keras [11], the implementation was built using Keras as a kind of wrapper over TensorFlow. Keras is a high level interface to various low-level artificial intelligence libraries, including TensorFlow.

The resulting program was launched on a graphics adapter with CUDA support. For comparison with the classical implementation of the cellular automaton of the Conway's Game of Life on a uniprocessor system, we used the implementation of [12].

R-pentamino located in the middle of the field (fig. 4) was used as the initial global state of the cellular automaton of the Conway's Game of Life in the experiments.

We took a square game field (the matrix of the global state of the cellular automaton) with dimensions $m \times m$, where m varied from 10 to 350 with the step 10. For each m, we calculated 1000 subsequent global states of the cellular automaton. The execution time was measured. The

calculations were repeated 10 times. Time was averaged. All experiments were conducted on a computer with the following characteristics: Intel Core i5-3470@3.2 GHz CPU, 8 GB RAM, Windows 7-x64 OS, GeForce GTX 650 Ti graphics adapter (1024 MB RAM, 928 MHz base frequency, 768 CUDA cores).

Dependency diagrams of the program execution time on the «length» of the square field side m of the game are shown in fig. 7 and 8. We also built regressions. The regression curves are shown in fig. 7 and 8 as well. A second-degree polynomial was chosen as the regression hypothesis.

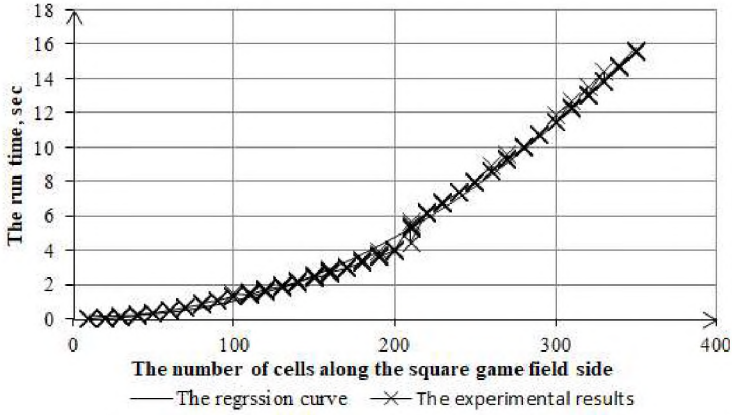

Fig. 7. Results of experiments with a single-threaded implementation

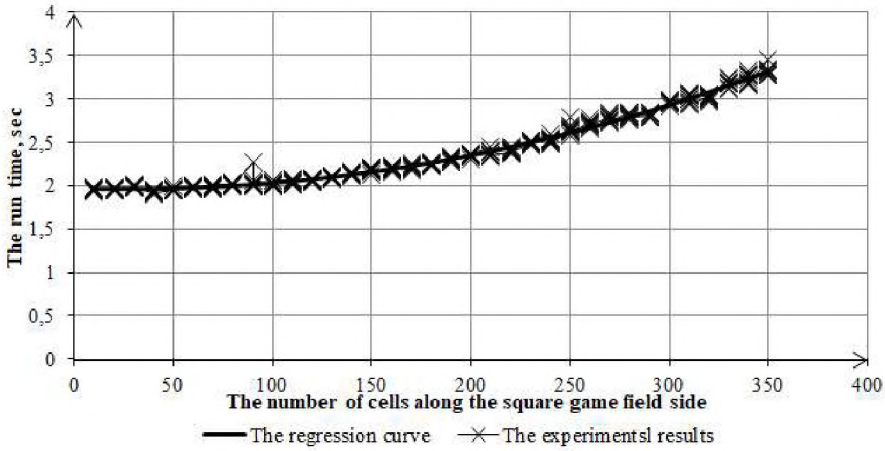

Fig. 8. Results of experiments with CUDA (Keras+TensorFlow) implementation

It can be noted that for small values of m, the execution time of a single-threaded program is smaller than the execution time of the multiprocessor (the graphics adapter) implementation proposed by us. However, as m grows, the situation changes and the proposed multiprocessor implementation begins to outperform the classical single-threaded implementation. We associate this with the overhead of transferring data from the computer's general RAM to the graphics adapter's RAM and returning the result from the graphics adapter's memory to the computer's memory. When the dimensions of the game field of the Conway's Game of Life are small, the time of actual calculations of the global

states of the cellular automaton is much less than the time of transmission of information. As the field size grows, the computation time of the cellular automaton state becomes significant and the multiprocessor implementation on the graphics adapter begins to outrun the single-threaded speed.

Obviously, the dependence of the execution time of programs on the "length" m of the square field side of the Conway's Game of Life must be parabolic. With the growth of m, the number of cells grows as m^2, each cell needs to be processed once per cycle. Therefore, the number of operations must be of the order of m^2. According to the obtained results we constructed regression polynomials of the second degree. Regression curves are in good agreement with experimental data (Fig. 7, 8). It may seem that for a multithreaded implementation the dependency should be different. However, we note that when the number of cells becomes much more than the number of cores in a multi-core system (in our case, the graphics adapter had 768 cores), then processing will be performed block by block: first comes one block of 768 cells, then another, etc. Thus, m^2/K operations will be done, where K is the number of cores, that is, also of the order of m^2.

6. Conclusions

In this paper, a tensor approach to the software implementation of cellular automata is described and programmatically implemented. The approach is focused on launching programs on multi-core graphics adapters. The program is implemented in Python using TensorFlow and Keras as an interface to TensorFlow. TensorFlow allows automatically generate and run multi-threaded programs on multi-core graphics adapters.

The effectiveness of using the developed approach was shown during a series of computer experiments. For the experiments the Conway's Game of Life was chosen. If the number of cells in the automaton is less or equal to the number of cores, then the maximum acceleration can be observed. If the number of cells exceeds the number of cores, then the parallel sections of the program are executed sequentially. This means that with a very large size of the playing field the type of dependence will be parabolic when using a graphics adapter. The latter is confirmed by regression analysis.

References / Список литературы

[1]. Harris D., Harris S. Digital Design and Computer Architecture. Morgan Kaufmann, 2012, 721 p.
[2]. Toffolli T., Margolus N. Cellular Automata Machines. MIT Press, 1987, 279 p.
[3]. von Neumann J. Theory of Self-Reproducing Automata. University of Illinois Press, 1966, 403 p.
[4]. Bandman O. Simulation Spatial Dynamics by Probabilistic Cellular Automata. Lecture Notes in Computer Science, vol. 2493, 2002, pp. 10–19
[5]. Malinetski G.G., Stepantsov M.E. Simulation of diffusion processes by means of cellular automata with Margolus neighborhood. Computational Mathematics and Mathematical Physics, 1998, vol. 38, no. 6, pp. 973-975.
[6]. Weimar J.R. Cellular Automata for Reaction-Diffusion Systems. Parallel Computing, vol. 23, no. 11, 1999, pp. 1699–1715.
[7]. Medvedev Yu.G. Development and Research of a Three-Dimensional Cellular Automaton Model of a Viscous Fluid Flow. PhD thesis, Novosibirsk, 2005, 108 p (in Russian). / Медведев Ю.Г. Разработка и исследование трехмерной клеточно-автоматной модели потока вязкой жидкости. Диссертация на соискание ученой степени кандидата технических наук, Новосибирск, 2005 г., 108 стр.
[8]. Computing Cluster «SKIF Cyberia». Available at: https://cyberia.tsu.ru, accessed 12.05.2019 (in Russian) / Вычислительный кластер СКИФ Cyberia.
[9]. TensorFlow. Available at: https://www.tensorflow.org, accessed 12.05.2019.
[10]. Gardner M. The Fantastic Combinations of John Conway's New Solitaire Game "Life". Scientific American, vol. 223, no 4, 1970, pp. 120–123.
[11]. Keras: The Python Deep Learning library. Available at: https://keras.io, accessed 12.05.2019.
[12]. Implementation of the Game "Life" using C++. Available at: https://code-live.ru/post/cpp-life-game, accessed 12.05.2019.

Информация об авторах / Information about authors

Наталия Андреевна ШАЛЯПИНА получила степень магистра радиофизики в 2018 г. В Национальном исследовательском Томском государственном университете, Томск, Россия. В настоящее время она готовит диссертацию на соискание степени кандидата физико-математических наук по направлению Информатика и вычислительная техника. Область интересов – клеточные автоматы, моделирование.

Natalia Andreevna SHALYAPINA received the M.S. degrees in radiophysics from National Research Tomsk State University, Tomsk, Russia. She is currently pursuing the Ph.D. degree in the field of Information and Computer Engineering. Research interests – cellular automata, simulating.

Максим Леонидович ГРОМОВ окончил радиофизический факультет Томского государственного университета и в 2004 году получил степень магистра радиофизики. В 2009 году защитил кандидатскую диссертацию. С 2009 года занимает должность доцента Томского государственного университета. Научные интересы связаны с дискретными моделями различных систем, обработки информации.

Maxim Leonidovitch GROMOV graduated from Radiphysics faculty of Tomsk State University and got master degree of Radiophysics in 2004. In 2009 he defended the PhD thesis in computer science. Since 2009 he holds the position of Associate Professor of Tomsk State University. Scientific interests are connected with dicrete models of different systems, information processing.