A Binary Array Asynchronous Sorting Algorithm with Using Petri Nets

A A Voevoda¹, D O Romannikov¹

¹ Novosibirsk State Technical University, 20, Karla Marksa ave., Novosibirsk, 630073, Russia

E-mail: voevoda@ucit.ru

Abstract. Nowadays the tasks of computations speed-up and/or their optimization are actual. Among the approaches on how to solve these tasks, a method applying approaches of parallelization and asynchronization to a sorting algorithm is considered in the paper. The sorting methods are ones of elementary methods and they are used in a huge amount of different applications. In the paper, we offer a method of an array sorting that based on a division into a set of independent adjacent pairs of numbers and their parallel and asynchronous comparison. And this one distinguishes the offered method from the traditional sorting algorithms (like quick sorting, merge sorting, insertion sorting and others). The algorithm is implemented with the use of Petri nets, like the most suitable tool for an asynchronous systems description.

1. Introduction

Nowadays, we need a growing amount of calculation resources for a plenty of different tasks. They are: machine learning (and particular neuron networks [1, 2]), internet services, medical researches and others.

For solution of this task, scientists and the whole industry moves in several directions: the first one is to use a parallelization technique. There is a strong trend for using Graphic Processing Unit (GPU) as a device with ability to calculate a lot of parallel processes. In [3], GPU was used for calculation of Boltzmann equation, in machine learning [1, 2, 4-6], GPU is a new trend for neural networks calculus [4-6]. The second trend is using Petri nets as a tool that can be applied for parallel and asynchronous processes description [7]. Particularly, in [8-10] Petri nets were applied for description of industrial automatic control systems of a water utilities. They were used for modeling communication between payphones and a center of their control. So, it is an important and actual task that should be solved for a sustainable development.

In the paper, we want to propose an algorithm of sorting an array of binary numbers using Petri nets. The main requirement of the algorithm is a maximum level of parallelization and asynchronization that is used for making the algorithm as fast as possible.

2. The algorithm

The main idea of the algorithm is to divide an entire array into independent pairs of adjacent elements and make their comparison in parallel. Let us consider it in more details in figure 1. Initially, all places do not contain any tokens, except for the start place and the places that represent numbers in a binary
form. Work of a network starts with a transition of tokens from the start place to $ei$-th places (each place represents numbers availability). Since the $fi$ places do not contain tokens, the transition from $ei$ and $ei+1$ is available for an operation. As a result, a token goes to the $di$ places that enables a comparison of adjacent elements with the help of the comparison sections. These sections are organized in a form of a table where columns represent elements in the form of discharges, and rows represent the same discharges of the elements. Each comparison section is linked by three inputs (comparison, swap and readiness) and outputs (need to compare next discharge, need to swap next discharge, elements are compared). Final transitions (need to compare next discharge and swap next discharge outputs) of the comparison sections are combined on places $ci$ and via transition $Bi$ tokens go to $ei$, $ei+1$ places to show that elements are not in a comparison, to $fi$ to block this pair of elements for the next comparison, and to $ai$, $ai-1$ (if they exist) - to unlock these pairs for the next comparisons. In case the left element is lesser than the right one, the token via the $bi$ transition goes to the same places except for $ai$, $ai-1$ in cause there is no demand to compare pairs of adjacent elements if they were not changed.

![Figure 1. A common scheme of an algorithm of an array sorting using Petri nets](image)

In figure 2, a section of array elements comparison is displayed. It contains two kinds of tokens: “stars” that show commands and “circles” that show an existence of a binary value. If places $p1$, $p2$ contain a token that has a value of one, in the opposite case – zero. The section starts its work at one of the three commands: the first one is “compare i-th elements” means that elements of comparison are
required. In this case one of transitions of comparison ("="","","<") is available due to transition from place \( p7 \) to these transitions. If \( p7 \) does not contain a token, then ("="","","<") transitions are blocked. Here we consider that the "=" transition is available if \( p1, p3 \) places contain circle tokens whether both of them do not contain anything, ">" is available if \( p1 \) contains token and \( p2 \) does not, and "<" is available in the opposite case.

**Figure 2.** A section of array’ s elements comparison

The second command “Swap i-th elements” firstly shifts the star token to \( p2 \), and after through transitions the places exchange their tokens. The last command “Elements’ ve compared” is used in case a comparison at above levels has already finished. So, it is like a bypass through all other sections.

3. An example
Let us consider an algorithm work by the following example of the array: \([9, 0, 1, 2, 3, 4, 5, 6, 7, 8]\). Conditional tacts are shown in table 1. Needless to say that on every conditional tact different pairs of adjacent elements can be formed. On the first conditional tact, the following pairs of elements were chosen: \(\{0, 1\}, \{3, 4\}, \{5, 6\} \text{ and } \{7, 8\}\). On the next conditional tact, the same pairs cannot be chosen; so, the next pairs are: \(\{0, 9\}, \{1, 2\}, \{4, 5\} \text{ and } \{6, 7\}\). As a result, the array will be sorted by 9 conditional tacts.

**Table 1.** An example of the algorithm applying

| # of conditional tact | Array elements     |
|-----------------------|--------------------|
| 0                     | \([9, 0, 1, 2, 3, 4, 5, 6, 7, 8]\) |
| 1                     | \([9, 0, 1, 2, 3, 4, 5, 6, 7, 8]\) |
| 2                     | \([0, 9, 1, 2, 3, 4, 5, 6, 7, 8]\) |
| 3                     | \([0, 1, 9, 2, 3, 4, 5, 6, 7, 8]\) |
It is important to mention that on the different starts, the algorithm can show different numbers of conditional tacts, i.e. Petri nets allow choosing different transitions and there is no guarantee that the optimal pairs will be chosen. A bubble sort (that was used as a prototype for the algorithm) requires about 50 tacts.

4. Future discussions
An idea of principles of parallelization and asynchronization using of Petri nets can be developed and applied to complex algorithms like a merge sorting [11]. A scheme of such applying is shown in figure 3 where an array is displayed as a spiral (it is required to show that each element has four neighbors (left, right, bottom, down)).

![Figure 3. An idea of applying parallelization and asynchronization principles to a merge sorting algorithm](image)

In this scheme, a basic structure is proposed. It consists of a logic of blocking that looks pretty the same as the logic of blocking in figure 1 (it also can contain several changes related to the fact that elements will be compared not only with left and right neighbors, but also with bottom and upper). The scheme of logic blocking allows (or disable) comparison sections, which can compare elements, to work (they are represented here as arrows from a big spiral coil to a small one) in four directions (left, right, bottom, down). After the spiral tokens go to the collection section (the same as ci places in figure 1) and from it back to the logic of blocking. Potentially, this scheme can work faster than the proposed algorithm.

5. Conclusion
In conclusion we want to say that an asynchronous algorithm was proposed. This algorithm allows sorting binary elements in parallel and asynchronously and as result it may significantly increase the sorting speed. For example, if an array contains 100 elements (it also means that a number of simultaneously compared elements is in the interval of 33 to 50 (including)), then it can be sorted by
150 tacts in the worst case. In the best case (when an array is sorted, the algorithm executes in 2 or 3 tacts depending on how pairs would be made).

The next steps of this work are applying principles of parallelization and asynchronization (that were shown in the paper) for the merge sort algorithm.

6. Acknowledgments
The reported study was funded by RFBR (grant 2014/138), the name of the project: “New structures, models and algorithms for the breakthrough technical systems management based on high-tech results of intellectual activity”.

References
[1] Haykin S 2009 Neural Networks and Learning Machines (Pearson Education) p. 938
[2] Bishop C 2007 Pattern Recognition and Machine Learning (Springer) 729 p
[3] Calore E, Gabbana A, Kraus J, Pellegrini E, Schifano S F, Tripiccione R. 2016 Massively parallel lattice–Boltzmann codes on large GPU clusters Parallel Computing 58 1–24
[4] Wang J, Ren Y, Zheng D, Wu Q. 2007 A Machine-Learning Based Load Prediction Approach for Distributed Service-Oriented Applications Lecture Notes in Computer Science 4487 462-465
[5] Lin M, Chen Q, Yan S. 2014 Network In Network
[6] Voevoda A A, Polubinskiy V L, Romannikov D O. 2016 Sort the array of integers with using a neural network Scientific Bulletin of NSTU 2(63) 151-157
[7] Yefremov A A 2005 Failure-repair processes simulation for parallel redundant configurations using stochastic Petri nets Proc. of 9th Russian-Korean International Symposium on Science and Technology, KORUS-2005 1 738-740
[8] Voevoda A A, Markov A V, Romannikov D O. 2014 Software development: design using UML diagrams and Petri nets on the example of PCS pumping station Proceedings SPIIRAS 3 (34) 218-232
[9] Korotikov S. 2007 Using color hierarchical Petri net for UML diagrams consistency checking on the phase of RMCC software design vol. 2 (27) (Science bulletin of the NSTU) pp. 203–206
[10] Selvesyuk N I, Ostrovskiy A S, Aristov R S, Gladkikh A A. 2016 Object-oriented design of a neural network to automate the process of computer architecture determination in the information security problems Science bulletin of the NSTU 1 (63) 133–145
[11] Cormen T, Leiserson C, Rivest R, Stein C. 2009 Introduction to Algorithms (The MIT Press) p. 1328