

**THE EFFICIENCY EXPLORATION OF PARALLEL WAVE ROUTING ALGORITHM
WITH GPU COMPUTING COMPARED TO CPU**

The present paper concerns the issues of speeding up the execution time of the modified reverse wave routing algorithm in a software-defined network of large size. The parallel version of the algorithm is executed on the predefined network sizes with the same edge probability on a multi-core CPU and GPU separately, partly on a multi-core GPU and partly on a multi-core CPU. The exploration results of the parallel algorithm help to define the most suitable way of algorithm computing in networks of different sizes.

Keywords: modified inverse wave algorithm, CPU, GPU, software-defined network.

Fig.: 4. Tabl.: 3. Bibl.: 3.

Relevance of the research topic. The modified inverse wave algorithm is an effective traffic engineering method for software-configured networks, as it reduces the time complexity of forming multiple paths and reduces reconfiguration time. However, the execution time of the algorithm increases significantly in large networks. This research considers the application of graphic processor technology to improve the performance of modified reverse wave routing algorithm in a large mobile network.

Target setting. The research target is to speed up the execution time of the routing algorithm in software-configured networks of large size by using GPU computing.

Actual scientific researches and issues analysis. Many scientific papers in the field of mobile networks are devoted to solving the problem of choosing an optimal algorithm for execution in large networks [1], [2], [3]. As powerful GPUs become more available and suitable for massively parallel computing, performing parallel processing of the algorithm on the GPU can solve the problem of speeding up the routing execution in a scalable network. Today there are many scientific works devoted to choosing CPU, GPU or CPU+GPU implementation that provides minimum execution time for different applications [4].

Uninvestigated parts of general matters defining. This article is devoted to the parallelization of the reverse wave routing algorithm and exploration of its execution efficiency in three cases including separate execution with GPU, CPU, and partly on GPU and partly on CPU to improve the algorithm performance characteristics in large mobile networks.

The research objective. The main task is to use the technology of graphic processors to make the reverse wave algorithm find paths in the large networks faster.

The statement of basic materials. First of all, the possibility of performing 'for' cycle iterations in parallel can be used to improve execution time of the algorithm. This is possible because the routers of the current wave can be computed separately and the results of their calculations can be combined to form the next set of routers and so on. Besides, the factors increasing the execution time of the parallel algorithm version will include the number of iterations and the maximum number of operations of minimum delay metrics change of adjacent nodes on each iteration.

1. Set initial number of routers $W_1 = \{R_n\}$;
2. $D_i = 0$;
3. $J = 0$; *can be executed in parallel*
4. for $j=j+1$ step 1 form the routers set $W_{j+1} = \{R_i | i=1, \dots, k\}$ adjacent to the routers set from $W_j = \{R_i | i=1, \dots, k\}$, where k - the sum of the degrees of the routers set $W_j = \{R_i | i=1, \dots, k\}$;
5. if $W_{j+1} = \emptyset$ then go to 10 do
6. for $i=1$ step 1 до k calculate $Z_i \{ V_m, V_b, M_{i,m}, d_i \}$
7. if $d_j > D_i$ then $D_i = d_j$
8. end;
9. go to 4
10. end.

Fig. 1. Pseudo code of the parallel algorithm

Then, the parallel wave algorithm execution on a multi-core CPU and a multi-core GPU depends on its implementation with the use of special libraries and data structures that fit specific architecture needs.

Experiments. Firstly, the proposed parallel algorithm was executed on a multi-core CPU only. The CPU characteristics included 4 processors Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz.

To implement the algorithm on the CPU, parallel processing tools in the Python programming language were used, namely, the multiprocessing module, which supports the process generation using the API. Due to the possibility of bypassing the Global Interpreter Lock (GIL), this module allows full use of several processors on the user's computer. With the use of a multiprocessing library in Python, processes are generated by creating a Process object and then calling the start() method. This package also includes special data types for exchange between processes. Table 1 shows the results of running the algorithm with CPU depending on the number of nodes from 100 to 1000 in a random connected graph with a step of 100 with an edge probability of 0.01:

Table 1

Table of algorithm execution results with CPU

t, sec	1,49	2,07	15,89	20,06	21,9	36,15	60	74,87	101,86	112,43
N	100	200	300	400	500	600	700	800	900	1000

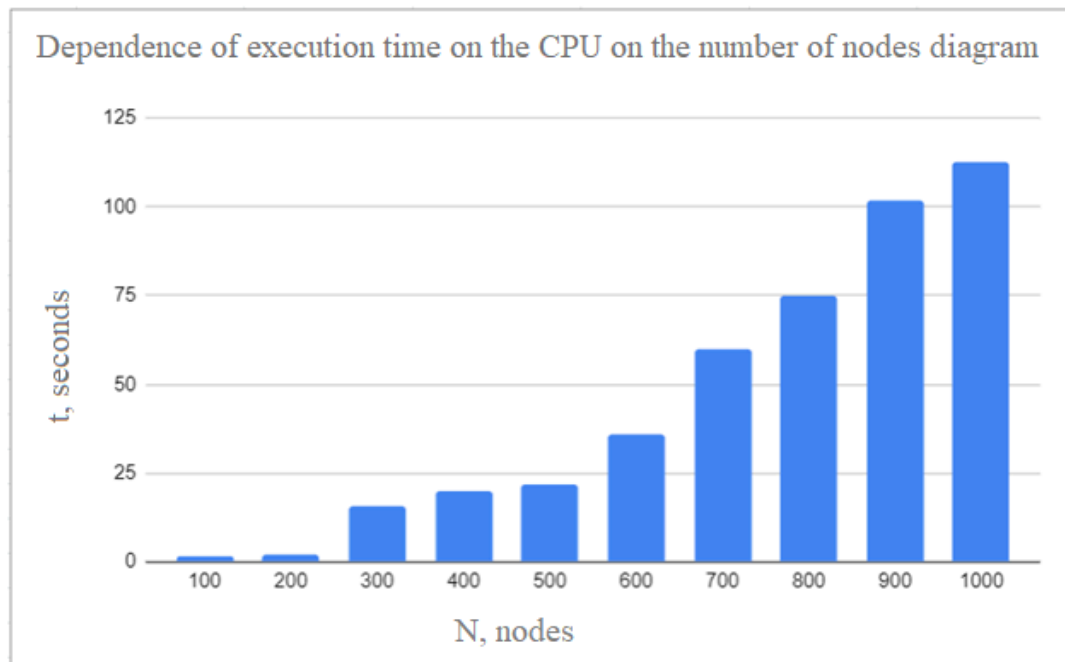


Fig. 2. Dependence of execution time on the CPU with nodes number

As can be seen from the results of execution at a given computing power obtained a small execution time with the number of nodes of the graph from 100 to 200, but then with an increasing nodes number the execution time of the algorithm does not give optimal results. It can be explained by the impossibility to process all the nodes in parallel because of an insufficient number of processors, and because the number of levels of the graph also increases, which also increases the execution time of the algorithm.

Secondly, the parallel algorithm was executed on a multi-core GPU. It was decided to use the CUDA simulator from the cudatoolkit package in Python that implements the functions of one GPU device with a capacity of 5.2 which is sufficient for writing kernel functions with GPU support. This cudatoolkit package also includes GPU-accelerated libraries and the CUDA runtime for the Conda ecosystem and the Numba library tools that support CUDA GPU programming by directly compiling a limited subset of Python code into CUDA kernels and device

functions according to the CUDA execution model. Kernels written in Numba seem to have direct access to NumPy arrays. NumPy arrays are transferred between CPU and GPU automatically. The algorithm was executed on the same range of graph nodes number from 100 to 1000 and using the same coefficient of edge probability in a graph that equaled 0.01.

Table 3 shows the results of running the algorithm with GPU depending on the number of nodes from 100 to 1000 in a random connected graph with a step of 100 with an edge probability of 0.01:

Table 2

Table of algorithm execution results with GPU

t, sec	0,07	0,16	0,73	0,87	0,94	1,78	2,21	3,31	3,85	4,13
N	100	200	300	400	500	600	700	800	900	1000

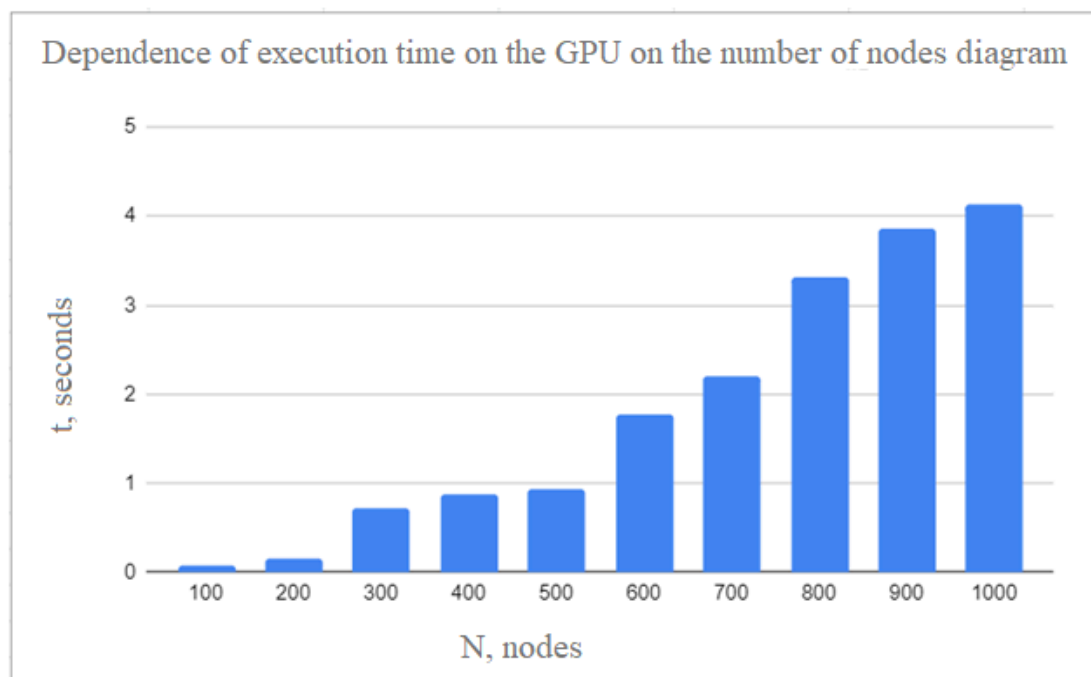


Fig. 3. Dependence of execution time on the GPU with nodes number

As can be seen from the results of execution at these graph sizes, there is a gradual increase in the execution time of the algorithm, which generally gives good execution results even at a graph size of 1000 nodes. Execution time

intervals with the number of nodes from 100 to 200, from 300 to 500, from 600 to 700, and from 800 to 1000 give approximately similar execution times.

Then, the algorithm was executed partly on a multi-core GPU and partly on a multi-core CPU in ratio proportion of fifty-fifty using the same characteristics of CPU and GPU. To implement partial parallelization of the algorithm on CPU and GPU in a 50/50 percentage ratio, a pre-implemented functionality was used to parallelize the algorithm on CPU and GPU, so that half of the graph is processed on the CPU and the other half with all its associated data transferred for processing on the GPU. Table 3 shows the results of running the algorithm partly on CPU and GPU depending on the number of nodes from 100 to 1000 in a random connected graph with a step of 100 with an edge probability of 0.01:

Table 3

Table of algorithm execution results partly on CPU and GPU

t, sec	1,71	3,9	5,03	11,51	16,26	18,32	22,41	26,9	41,53	63,85
N	100	200	300	400	500	600	700	800	900	1000

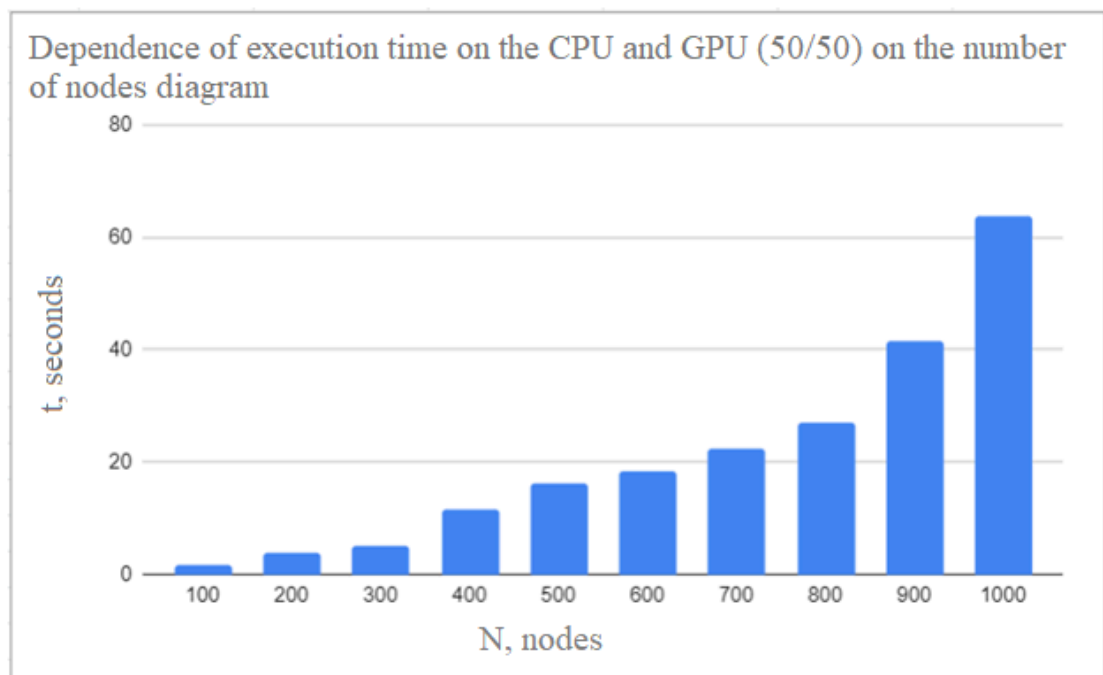


Fig. 4. Dependence of execution time on the CPU and GPU with nodes number

This combination gives good results of parallelization at the number of nodes from 100 to 300. After that, there is a significant increase in execution time at each subsequent interval, which may be due to an insufficient number of cores on the CPU and time spent on data transfer from CPU to GPU and vice versa.

To sum it up, the results from all three experiments are given in the form of a bar chart on Fig 5.

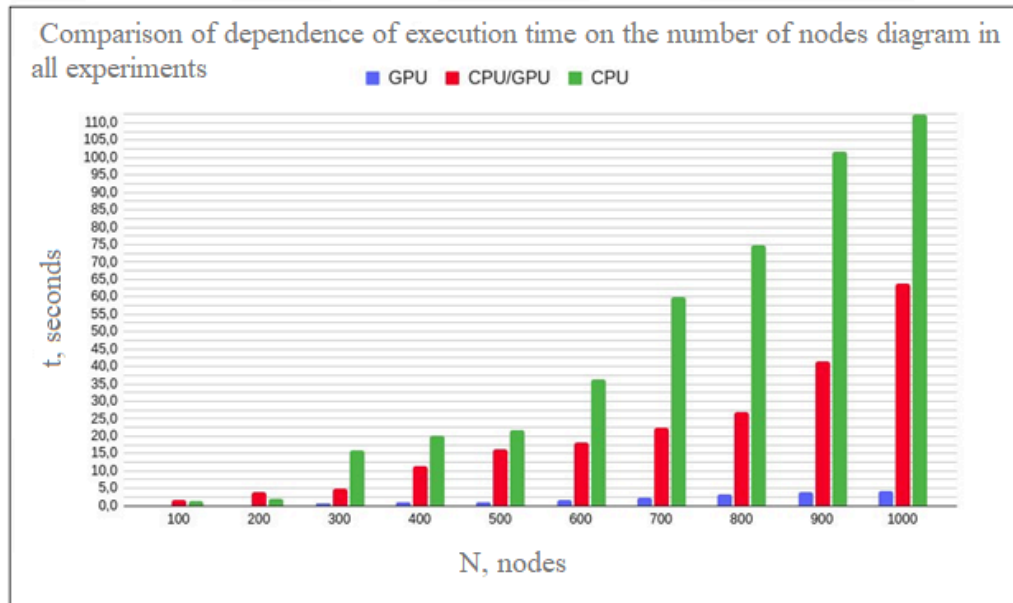


Fig. 5. Comparison of execution time results from all experiments

Conclusions. It has been proven that with an increasing number of graph nodes the algorithm execution time increases, and the best execution time is provided on the GPU, due to the architecture, because GPUs have enough cores to process large amounts of data in parallel, which in this case is determined by the number of vertices. Due to the insufficient number of cores, the execution time on the CPU is much longer, as all vertices in the queue are not processed in parallel by each core, but are distributed to available cores and wait for their execution sequentially. When running the algorithm partly on the CPU and partly on the GPU, we have better results than those obtained when running only on the CPU, although for graphs with less than 200 nodes it is more profitable to apply

the algorithm only on the CPU than partly on the CPU and GPU associated with additional time delays for data transfer from CPU to GPU.

References

1. Kulakov Y. et al. Traffic engineering in a software-defined network based on the decision-making method //Восточно-Европейский журнал передовых технологий. – 2019. – №. 2 (9). – С. 23-28.
2. Kulakov, Y., Kogan, A., “The method of plurality generation of disjoint paths using horizontal exclusive scheduling“, The advanced science journal. Issue 10. Volume 2014. ISSN 2219-746X. DOI: 10.15550/ASJ.2014.10. Pp. 16-18.
3. Kulakov Y. et al. Load Balancing in Software Defined Networks Using Multipath Routing //International Conference on Computer Science, Engineering and Education Applications. – Springer, Cham, 2020. – С. 384-395.
4. Rusanova O.V.,Korochkin A.V.,Shevelo O.P. Classification of scheduling problems for modern parallel computer systems// Збірник тез доповідей XII Міжнародної науково-технічної конференції «Комп’ютерні системи та мережні технології,м.Київ,28-30 березня 2019р.-К.:НАУ,2019.-С.102-103.

AUTHORS

Yurii Kulakov (supervisor) – professor, Department of Computer Engineering, National Technical University of Ukraine "Igor Sykorsky Kyiv Polytechnic Institute".

Olga Rusanova (supervisor) – associate professor, Department of Computer Engineering, National Technical University of Ukraine "Igor Sykorsky Kyiv Polytechnic Institute".

Iryna Hrabovenko - student, Department of Computer Engineering, National Technical University of Ukraine "Igor Sykorsky Kyiv Polytechnic Institute".

Yuliia Hrabovenko - student, Department of Computer Engineering, National Technical University of Ukraine "Igor Sykorsky Kyiv Polytechnic Institute".

Email: yuliia.hrabovenko@gmail.com