Modeling the Distribution of Infectious Diseases Using Parallel Calculations

Lesia Mochurad [a], Natalya Kustra [a]

[a] *Lviv Polytechnic National University, Lviv, 79013, Ukraine*

Abstract
Global warming and related climate changes are the subjects of intensive research as they affect the social, economic, and medical aspects of human life. Large-scale climate change: melting glaciers and the spread of new infections are likely to have a negative impact on the world economy. The spread of infectious diseases is indeed relevant, but the amount of data needed to model the spread is large, and therefore the processing time of this data is large. Therefore, the use of parallel algorithms will speed up the solution to this problem, and, accordingly, faster obtaining the results of modeling the spread of infectious diseases. A parallel algorithm for modeling the spread of infectious diseases based on OpenMP technology has been developed and has been investigated; the benefits of using such a feature as multi-core personal computers have been analyzed in the paper. Parallel modeling uses a graph of an imaginary city. The results of speed and acceleration of the parallel algorithm, which indicates the optimization of the computational process have been analyzed. You can increase the acceleration rate, as well as direct the efficiency to one by varying the number of processor cores.

Keywords
An influence of global warming, a mathematical model, a parallel algorithm, OpenMP technology, acceleration.

1. Introduction

In light of recent world events, people are more interested than ever in the spread of infectious diseases. A complex global health care system has been developed to repel known and unknown threats of infectious diseases. However, the world continues to face long-standing, emerging, and recurring threats of infectious diseases. These threats vary widely in severity and probability. They also have different consequences for morbidity and mortality, as well as for social and economic factors. The question is whether the current global health care system can provide effective protection against the dynamic array of threats of infectious diseases that have called into question recent outbreaks of Ebola, Zika, Dengue, Middle East respiratory syndrome, severe acute respiratory syndrome, influenza and the so-called coronavirus COVID-19. This can be due to many factors, including rapid population growth in areas with weak health care systems, urbanization, globalization, civil conflict, and the possibility of human-animal disease transmission. There is also a potential risk of outbreaks of viruses resulting from laboratory accidents or deliberate biological attacks.

Also, one of the potential risks of virus outbreaks is climate change or so-called global warming. Fourier transform is used in the process of mathematical modeling and analysis of the latter. This operation is also used in listening to electromagnetic signals of artificial origin. In particular, NASA [1, 2] addressed this problem by supporting the SETI project [3]. His idea is to listen to electromagnetic signals of artificial origin. If there is a developed civilization somewhere, it could evolve enough to invent the radio, the television, or any other system that works with electromagnetic

IntelITSIS'2022: 3rd International Workshop on Intelligent Information Technologies and Systems of Information Security, March 23–25, 2022, Khmelnytskyi, Ukraine
EMAIL: lesia.i.mochurad@lpnu.ua (L. Mochurad), Nataliia.O.Kustra@lpnu.ua (N. Kustra).
ORCID: 0000-0002-4957-1512 (L. Mochurad), 0000-0002-3562-2032 (N. Kustra).

waves. Sooner or later, the signals generated by such civilizations will reach the Earth, where we will be able to recognize them. Of course, it is unlikely that any meaningful set of information for millions of light-years will reach us undistorted, Information from radio telescopes is used not only in the SETI project but also for radio astronomy research: observation of radio galaxies, quasars, pulsars, supernova remnants, radio radiation of the Sun, stars, planets, etc., black holes. Parallel calculations are also used for this purpose. They make it possible to obtain the frequency spectrum from the electromagnetic flux much faster than during sequential processing. Calculations can even be accelerated so much that we get the result in real-time.

With the rapid accumulation of processed data in the simulation of a problem, there is a need to apply the parallelization of the computational process, to be able to make decisions in real-time. In particular, parallel calculations are considered as a tool for modeling complex phenomena and processes, as well as in solving cumbersome scientific and engineering problems [4-6]: in the study of atmospheric phenomena, environmental pollution, solving problems of geology, seismology; applied and nuclear physics, electrical engineering, particle physics, materials science, nanotechnology, microelectronics; research in the fields of biotechnology, ecology, population biology, virology, genetics, pharmacology, and molecular chemistry.

That is why there is currently a growing interest in parallel computing in various fields, including medicine [7, 8]. Scientists and engineers are doing their best to model or predict the possible spread of viral diseases in the future. This section will create a model for the spread of infectious diseases within a single city using parallel calculations. This is an important step in understanding and researching how a person may respond to different treatments or attempts to predict a pandemic. The main task is to create a simplified model of infectious disease and show how the disease spreads among the population. You can learn more about algorithms and modeling methods in the following sections.

2. Related Works

One of the natural phenomena for the analysis and modeling of which parallel calculations are used is global warming. Global warming is a very important issue for life on the planet today. Weather conditions are changing, and it seems that human activity is one of the main reasons. Since the beginning of the industrial revolution, the burning of fossil fuels has increased unnatural emissions of carbon dioxide into the atmosphere. Carbon dioxide is a greenhouse gas that absorbs infrared radiation generated by the reflection of sunlight on the Earth's surface, capturing heat in the atmosphere. Global warming and related climate change are the subjects of intensive research because they affect the social, economic, and medical aspects of human life. One of the most interesting works in this field was the article [9]. This work investigates the impact of global warming on infectious diseases and its future prospects. Global warming has different effects on human health. The main indirect consequences are infectious diseases. Although the impact on human health varies, depending on the location of the countries concerned and the socio-economic situation. Among infectious diseases, aquatic, foodborne and infectious diseases transmitted with food are the three main categories that are predicted to be most prevalent. Infections of infectious diseases such as malaria and dengue fever are mainly due to the expansion of infected areas and an increase in the number and activity of infected mosquitoes. The number of cases of diarrhea transmitted by water and food is growing. Implementing measures to adapt to the effects of global warming is the most practical action we can take. It is generally accepted that the impact of global warming on infectious diseases is not yet apparent at this time in East Asia. However, these effects will manifest themselves in one form or another if global warming continues to progress in the future [9]. The following conclusions were drawn from this scientific work: development of vaccines and vaccination, development of new drugs, creation of monitoring and control programs; the forecast of the epidemic and the development of preventive measures are quite important tools in the fight against the epidemic.

In the conclusions, many researchers suggest that climate change has a negative impact on human health, including infectious diseases. However, it should be noted that the level of impact of climate change on human health will vary from region to region depending on various factors, such as social infrastructure and the establishment of countermeasures. This complicates the interpretation of research results. Understanding the impact of climate change on human health has progressed greatly in recent years; however, it is true that much more research and data are needed to further understand the effects of climate change in detail.

There are other important issues to consider, especially when studying the effects of infectious diseases. The number of patients with infectious diseases is influenced by many factors. There are differences in "contagiousness" among the strains of each pathogen. Thus, the amount of symptomatic infection may vary depending on the level of "infectivity" of the dominant strains of the pathogen. In addition, changes in the number of cases depend to a large extent on the accuracy of surveillance and reporting systems, which have not yet been established in many developing countries. Thus, the study of the impact of global warming on infectious diseases should take into account multiple biological, sociological, and economic factors.

Also, a very important article in this study was the scientific work [10]. The authors have developed a computerized algorithm that estimates "who transmits to whom", that is, the most likely routes of transmission during an outbreak of a human-transmitted disease. This algorithm uses basic information about the natural history of the disease, population structure and chronology of the observed symptoms. To assess the effectiveness of the algorithm, the authors built a simulator with parameters describing the disease and population to simulate accidental outbreaks of influenza. The performance of the algorithm was compared with three reference methods that simulated how a person behaves in such situations. For any size of outbreak, the algorithm provided the majority of cases for which the source that transmitted the infection was identified. The authors also demonstrated the applicability of the algorithm for describing influenza outbreaks in nursing homes.

In this work [11], the authors present a categorization framework for categorizing multiscale models of infectious disease systems. The categorization framework consists of five integration frameworks and five criteria.

The use of MPI technology for parallel calculations is well covered in [12, 13]. In these works, the author explores all the advantages and disadvantages of using MPI in the study of algorithms for solving problems that require large data processing.

Also expanding the topic of parallel simulations, we drew attention to the article [14]. Preventing and combating epidemics in human or computer networks is a major issue. This work presents the implementation of an algorithm that simulates the spread of the epidemic in networks using CUDA technology. The spread of epidemics on the network is modeled by a discrete SIR model ("can be infected" – infected – recovered). This implementation allows you to select a starting point and control the spread of the epidemic in each cycle. Compared to a regular CPU implementation, a CUDA implementation achieves approximately 10x faster runtime, which is important for running tests on large networks. The implementation was tested on real social networks consisting of more than 5 million nodes. Therefore, this implementation may be of practical importance in the analysis of an epidemic that extends to large networks.

In [15, 16] the use of OpenMP technology in modeling big data processing problems was investigated. The advantages of using such a property as high-fluidity and multi-nuclear architecture of modern personal computers are shown. This article will also investigate the use of OpenMP in parallelization of the algorithm for modeling the spread of infectious diseases.

3. Proposed methodology

Modeling the spread of infectious diseases using parallel computations is an important step in studying how populations can respond to different treatments or to predict a pandemic. The focus of the study in this section is a simplified model of an infectious disease spread by the population. People move randomly, which can be mathematically described by a graph, the vertices of which represent the points of contact. The spread of the disease is possible when an infected and healthy person are in the same place. This section will develop a parallel algorithm for such modeling using OpenMP technology [17].

This approach is based on the statement: for any period of time it is true to say that the number of people who joined the array of patients is equal to the number of people who have ceased to be healthy.

β – the probability of infecting a healthy person in contact with a sick person,

$n_S(t) = |S(t)|$ – the number of healthy individuals at time t,

$n_i(t) = |I(t)|$ – the number of patients.

The mathematical model for our sequential algorithm were the formulas for the spread of the disease of the following type:

$\frac{dn_s(t)}{dt} = -\beta n_s(t) n_i(t);$

$\frac{dn_i(t)}{dt} = \beta n_s(t) n_i(t).$

Obviously, when $n_i + n_s = N = const$, we have:

$\frac{dn_s(t)}{dt} = -\beta n_s(t)(N - n_s(t));$

$\frac{dn_i(t)}{dt} = \beta(N - n_i(t)) n_i(t).$

Here the multiplication $n_s(t)\big(N - n_s(t)\big) = \big(N - n_i(t)\big) n_i(t) = n_s(t) n_i(t)$ – is equal to the number of contacts per unit period of time if each healthy person is in contact with each patient. The first line describes the number of those who fell ill (left part of the equation) as the share of those who became infected during contact (right part of the equation). In the second equation, the new number of patients (left part of the equation) is signed as the part of those infected by contact (right part of the equation).

The main disadvantage of the approach is the assumption that the sick and healthy are evenly distributed in space. As a result, β is a constant for all participants in the social network, and it is supposed that contacts occur between all couples (sick, healthy). Also, the model makes predictions at the level of quantity and does not make any assumptions about who will get sick in the next moment.

The system of equations presented above can be rewritten for the case when at any given time a healthy person has approximately (k) contact with patients, and therefore (k) opportunities to become infected.

$\frac{dn_s(t)}{dt} = -\beta(k) n_s(t)(N - n_s(t));$

$\frac{dn_i(t)}{dt} = -\beta(k) n_i(t)(N - n_i(t)).$

Three algorithms were considered. The basis of our study was a known sequential non-optimized algorithm. In order to improve and speed up the program, we optimized it by adding some changes: exit from the closed for loop, which compared if a person with a suspected disease was infected; skipping iterations for already infected people. We also developed a parallel algorithm, the purpose of which was to parallelize the bypass of an undirected acyclic graph, record.

More about algorithms:

- Sequential (without optimization): algorithm for traversing an undirected acyclic graph, using path search tables ("hash tables" that store optimal routes from one vertex to another to avoid recalculating the same paths), C ++ 11, STD libraries, STL and Boost Graph.
- Sequential (optimized): non-directional acyclic graph algorithm using path search tables, C++ 11, STD, STL and Boost Graph libraries. Some optimizations were introduced: exit from the closed for loop, which compared if a person suspected of having the disease became infected; skipping iterations for already infected people.
- Parallel: non-directional acyclic graph algorithm using path search tables, C++ 11, STD, STL, Boost Graph and OpenMP libraries with most functions. The above-mentioned optimizations were also applied.

Due to an OpenMP issue with the GCC compiler on Ubuntu Linux, the experiments were performed on a Windows 10 x64 computer using Visual Studio and the Intel C++ Compiler update. The first versions of the code program were implemented using MSBuild and Visual Studio. In addition, the Intel VTune application was used to debug the program.

Parallel modeling uses a graph (Figure 1), the form of which can be seen in Table 1. A graph has a certain number of nodes and edges and does not change during the simulation process, instead people can move along the edges between its nodes.

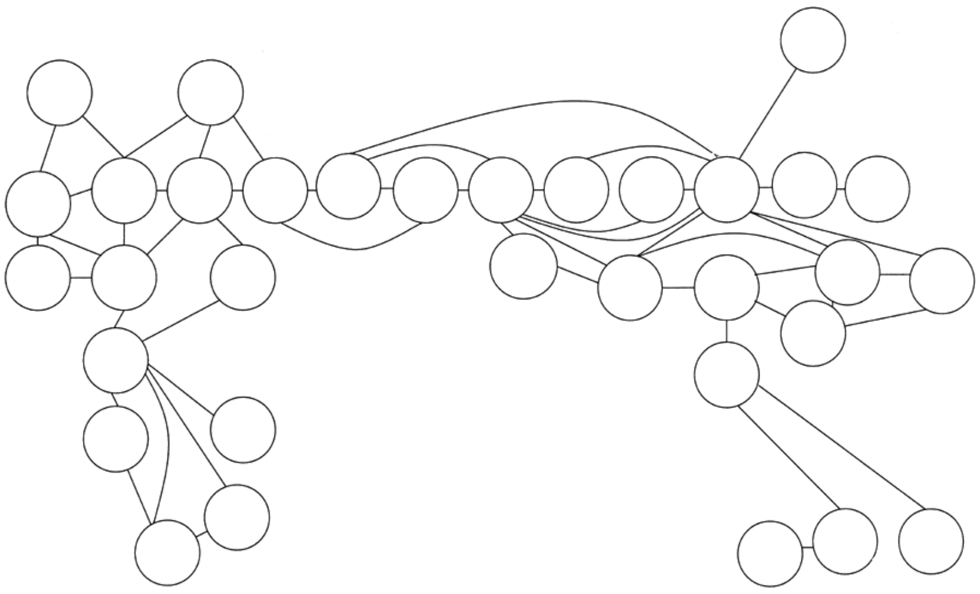


Figure 1: A fragment of a graph of an imaginary city, where nodes indicate certain places of gathering of people, and edges – ways of movement of people between these places

Two-way search was used to bypass the graph. Unlike other traversal algorithms that search for a path from point A to point B, two-way search allows you to run a reverse traversal, when it is possible to search in parallel from A to B, and from B to A. Accordingly, the traversal is divided into two subgraphs. The algorithm is executed in parallel in two vertices A and B and has the following form:

1. Start from vertex A. Execute BFS (v): = 1. Include vertex v in the queue.
2. Perform an estimate of the heuristic function (Euclidean distance).
3. Consider the vertex that is at the beginning of the queue; let this be the vertex of x. If BFS numbers are already defined for all vertices adjacent to vertex x, go to step 5, otherwise go to step 4.
4. Let {x, y} be an edge in which the BFS number (y) is not defined. Mark this edge with a bold solid line, define BFS (y) as the next BFS number, include the vertex in the queue, and go to step 3.
5. Exclude vertex x from queue. If the paths intersect, write our path to a file. Otherwise, go to step 3.

Normal BFS has a complexity $O(b^d)$, and two-way search $O(b^{d/2} + b^{d/2}) \Rightarrow O(b^{d/2})$, which gives much better efficiency.

The simulation performs the following steps:

1. Create a path search table with neighboring nodes for each node of the graph.

2. Repeat for all eras (steps):

(a) Accidentally move all persons.

(b) For all persons, if a person is infected and meets another person who is suspected of having the disease, check that the person with suspicion is infected.

(c) Moving on to the next era: Check people for the number of people infected in each era and mark them as cured if the disease threshold is exceeded.

3. We collect statistics about the current era: the share of infected people and contacts between people.

Simulation output:

- number of people;
- number of threads;
- number of epochs (simulation steps);
- the graph is generated or downloaded from a file;
- number of infected;
- export graph;
- the results of the epidemic.

All places where people are are stored in an undirected acyclic graph.

The Individual class has the following structure:

- One person can be infected, cured, infect others.
- The try_infect method uses uniform distribution and returns a real number. The move method uses the same method, but returns an integer. This random number is used to select the next random adjacent location to move the person. The location can be either the same node of the graph, or another node to which it is connected at a distance of 1 node from the current node.

The IndividualParameters structure determines a person's probability of becoming infected and the period of infection in the epoch.

The GraphHandler class has only static methods:

- Showing the results of the epidemic;
- Export results to csv;
- Generation of undirected graphs;
- Read graphs from files;
- Location of persons on the graph;
- Generate a path search table for neighboring nodes.

The get_node_neighborhood_lookup_map method scans the graph and returns a table that connects each vector location to neighboring locations. It uses a specialized Boost iterator that bypasses all nodes/vertices. For each node, an iterator with Boost can run on all adjacent vertices. Adjacent indices are stored inside the vector. The method of obtaining random faces generates a vector of faces of a certain size. This assigns them a random location.

The get_location_undirected_graph_from_file method can process an imaginary city graph file and generate an undirected location graph. It analyzes the file using Boost. This method reads two size_t values from each line from the file. A search table that is smaller (4 bytes) than 8 bytes size_t can be used. As will be shown later, this will help speed up the execution of simulations for both serial and parallel versions, especially when the number of simulated individuals exceeds 10.000.

The tuple vector is used to store epoch-making simulation statistics. Each epoch is represented by 3 integers: "number of contacts", "number of infected" and "number of recovered" persons. One motorcade is stored for each era.

The show_epidemic_results method can display information for each era:

- Percentage of contacts between persons (part of all people who have been infected at least once).
- The peak of the epidemic in percent.
- The peak of the epidemic.
- Number of contacts infected and recovered at the end of the simulation.

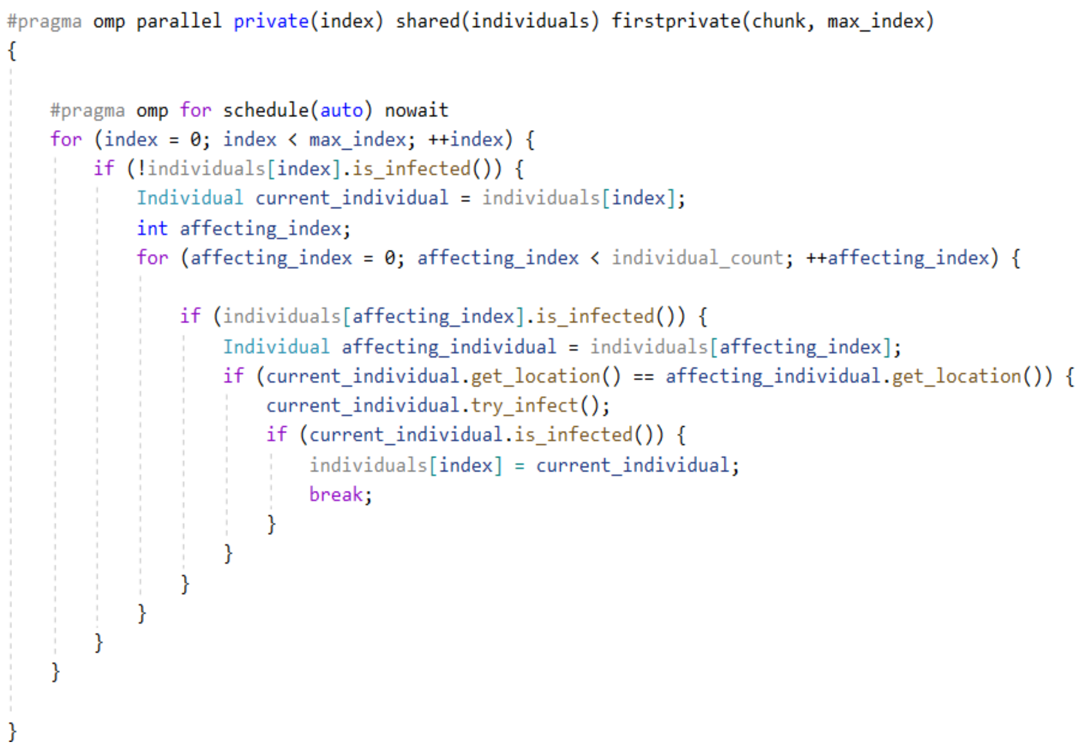
To check the simulation data, there is an assert_epidemic_results method that does this.

Some optimizations for the sequential algorithm were applied in the "infection phase":

- Checking whether another person will be infected: first you need to check whether the person is already infected. Second, check if two people are in the same place. The second operation is "more expensive" in terms of performance, and this cost can be completely missed.
- When you try to infect a person, you need to check if they are really infected. If so, the inspection cycle for that person should be stopped. You no longer need to check the infection between this person and any other person.

Initially, double-floating-point exact numbers were used, but later all of these numbers were replaced with the same ones, but with one decimal place, as they can be scaled better. In addition, they use less memory (64 vs. 32 bits). Achieving as many threads as possible was the most defining factor in program parallelization. During program execution, it was found that a call to obtain the size() of the vector would significantly impair performance. The scalar variable size_t inside the parallel section solved this problem.

In the Figure 2 below you can see part of the parallel algorithm of the program, which parallels the comparison when a healthy person comes into contact with a sick person, whether he is really infected. Simultaneous updating of the vector of patients was also paralleled.



```
#pragma omp parallel private(index) shared(individuals) firstprivate(chunk, max_index)
{

    #pragma omp for schedule(auto) nowait
    for (index = 0; index < max_index; ++index) {
        if (!individuals[index].is_infected()) {
            Individual current_individual = individuals[index];
            int affecting_index;
            for (affecting_index = 0; affecting_index < individual_count; ++affecting_index) {

                if (individuals[affecting_index].is_infected()) {
                    Individual affecting_individual = individuals[affecting_index];
                    if (current_individual.get_location() == affecting_individual.get_location()) {
                        current_individual.try_infect();
                        if (current_individual.is_infected()) {
                            individuals[index] = current_individual;
                            break;
                        }
                    }
                }
            }
        }
    }

}
```

Figure 2: Fragment of the parallel algorithm by OpenMP means

The omp_get_wtime method is used to determine the time of both serial and parallel algorithms. “Benchmark” can be used to perform several simulations simultaneously and output the results in CSV (Table 1, Table 2).

Table 1
The results of performance testing “Benchmark”

execution _time	execution _type	thread _count	individual _count	node _count	edge _count	total _epochs	epoch _timestamp	repeat _count
1.01414e+07	openmp	1	503138	152506	170294	30	1	1
6.17857e+06	openmp	2	503138	152506	170294	30	1	1
5.17666e+06	openmp	3	503138	152506	170294	30	1	1
4.13226e+06	openmp	4	503138	152506	170294	30	1	1

Each data column represents the execution time, the execution type (synchronous or parallel), the number of threads, the number of people, the number of nodes, the number of edges, the number of epochs (steps), the number of the epoch from which began modeling, and the number of repetitions of the experiment with these parameters, respectively. Before each execution, the reset_input method will re-initialize all input simulation parameters.

In general, the factor that usually slows down parallel execution is the synchronization of elements. These elements are usually collections of shared memory objects that can be read and written simultaneously by many different threads. This introduces the need to use atomic and critical sections, but at the same time it slows down the program.

Table 2
The results of advanced performance testing "Benchmark"

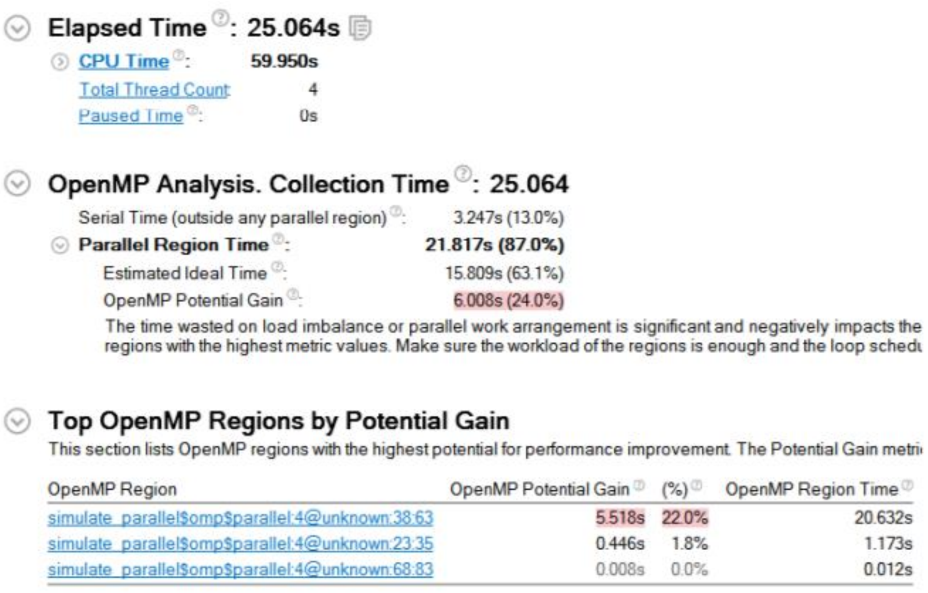
execution _time	executio n _type	threa d _coun t	individua l _count	node _count	edge _count	total _epoch s	epoch _timestam p	repeat _coun t

1.42438e+06	serial	1	100000	152506	170294	30	1	1
403151	openmp	1	100000	152506	170294	30	1	1
261820	openmp	2	100000	152506	170294	30	1	1
223662	openmp	3	100000	152506	170294	30	1	1
192600	openmp	4	100000	152506	170294	30	1	1
125.261 th most common	serial	1	50	152506	170294	30	1	1
321.829	serial	1	500	152506	170294	30	1	1
6529.42	serial	1	5000	152506	170294	30	1	1
376779	serial	1	50000	152506	170294	30	1	1

The following OpenMP features have been applied:

- Synchronization was used to a minimum and this was achieved by changing the algorithm so that each thread could write to different elements, ie in the "infection phase": since we change only individuals with indices of each thread, there is no need for an atomic/critical section for the vector element, recorded with only one thread.
- Local thread variables are used to minimize access to global memory.
- External for loops are paralleled.
- Implemented the ability to save local thread variables at the end of each cycle (also implemented a check to see if it is necessary – for example, for future comparisons)
- Private modifiers are used for the indexes of the cycles of each thread, as well as firstprivate for other temporary variables.
- Changed the number of iterations of the loop, which can be processed by each thread using Schedule (static, chunk). In practice, the use of Schedule (auto) showed a significant acceleration (see Figure 3).

Used reduction: + in the results collection phase. This preprocessor directive eliminates the need to use the atomic section on common variables of the number of infected, recovered, and the number of contacts that are updated by all threads at the same time. Compared to the atomic section, updating variables with the reduction directive has been more productive.



Elapsed Time: 25.064s

CPU Time:	59.950s
Total Thread Count	4
Paused Time:	0s

OpenMP Analysis. Collection Time: 25.064

Serial Time (outside any parallel region):	3.247s (13.0%)
Parallel Region Time:	21.817s (87.0%)
Estimated Ideal Time:	15.809s (63.1%)
OpenMP Potential Gain:	6.008s (24.0%)

The time wasted on load imbalance or parallel work arrangement is significant and negatively impacts the regions with the highest metric values. Make sure the workload of the regions is enough and the loop schedu

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metri

OpenMP Region	OpenMP Potential Gain	(%)	OpenMP Region Time
simulate_parallel$omp$parallel:4@unknown:38:63	5.518s	22.0%	20.632s
simulate_parallel$omp$parallel:4@unknown:23:35	0.446s	1.8%	1.173s
simulate_parallel$omp$parallel:4@unknown:68:83	0.008s	0.0%	0.012s

Figure 3: Screenshot of the Intel VTune Amplifier program that was used with Visual Studio

So, the acceleration of the parallel algorithm compared to the sequential non-optimized algorithm using the parallel method was 24%.

4. Results

Test machine configuration. So the test machine was an x64 laptop:
● Processor: 1x CPU 3.0 GHz Intel Core i5 480M, 4-core, 4-thread
● Memory: 8GB DDR3 1066MHz 64bit
● GPU: Intel HD Graphics (Generation 1)
● Hard disk: OCZ Vertex 150, 240GB SSD drive
● OS: Microsoft Windows 10 x64
And also the x64 laptop:
● Processor: Intel Celeron N4000 (1.1 - 2.6 GHz), 2 cores, 2 threads
● Memory: 4 GB RAM
● GPU: Intel UHD 600
● Hard disk: 500 GB HDD
● OS: Microsoft Windows 10 x64
Compilers / Libraries:
● Intel C ++ Compiler.
● OpenMP
Parameters:
● Age step: 1 (day)
● Number of epochs: 30 + 1
● Number of people: 50, 100, 500, 1000, 5000, 10000, 50000, 100000
● Streams: 1, 2, 3, 4
● Graph file: test.edges (Table 3)
● Number of nodes in the test graph: 152506
● Number of edges in the test graph: 170294 (undirected)
● Initial number of infected: 15

The test.edges file (see Table 3) is a set of vertices, each with its own unique identifier. A pair of vertices describe the relationship between them.

Table 3
View of the test.edges file

№	id_1	id_2

1	708864979	708865058
2	708864979	708864885
3	708864979	708865225
4	1470917308	26263088
5	1470917308	2773443163
6	1470917309	1470917034
7	1470917309	27734414305
8	1470917309	1270915518
9	1470917309	654621604
10	654162161	1470917302
11	654162161	1470917728
12	14790017300	1470917865
13	14790017300	1718443526
14	14790017302	1718443525
15	14790017302	26862406
16	14790027304	2773443160
17	14790027304	2772443660
18	314432202	2773443165
19	314432203	262636088
20	314432204	262637078
21	314432303	262630988

The simulation time for more than 10 000 people is growing at an exponential rate (see Figure 5). However, the more the "population" grows, the greater the acceleration (see Figure 4). Also, the advantage of using 4 threads instead of three is already visible in simulations for 50 000 people and more (see Figure 6).

The parallel algorithm gives acceleration from only 500 people (see Figure 5). Finally, the general conclusion is that even minor code optimizations have a strong effect on the execution time of the algorithm (even sequential) (see Figure 7).

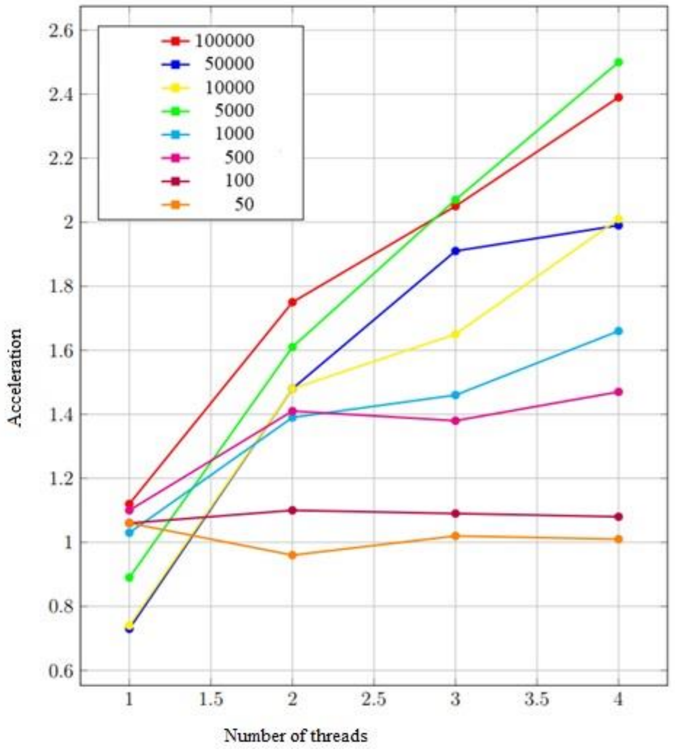


Figure 4: Comparison of acceleration results to model the spread of infectious disease in different numbers of threads using OpenMP for different numbers of people

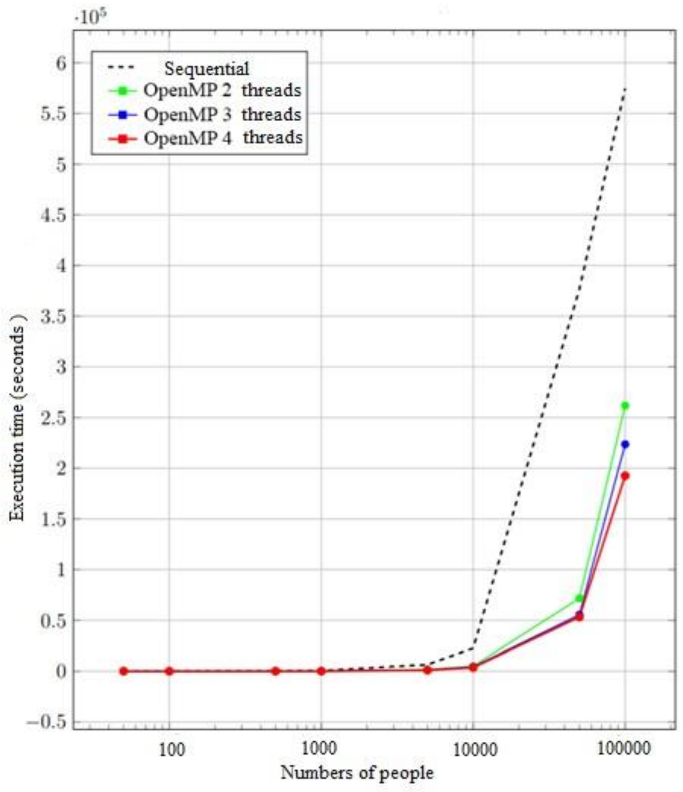


Figure 5: Comparison of the results of the execution of sequential and parallel algorithms for different numbers of people

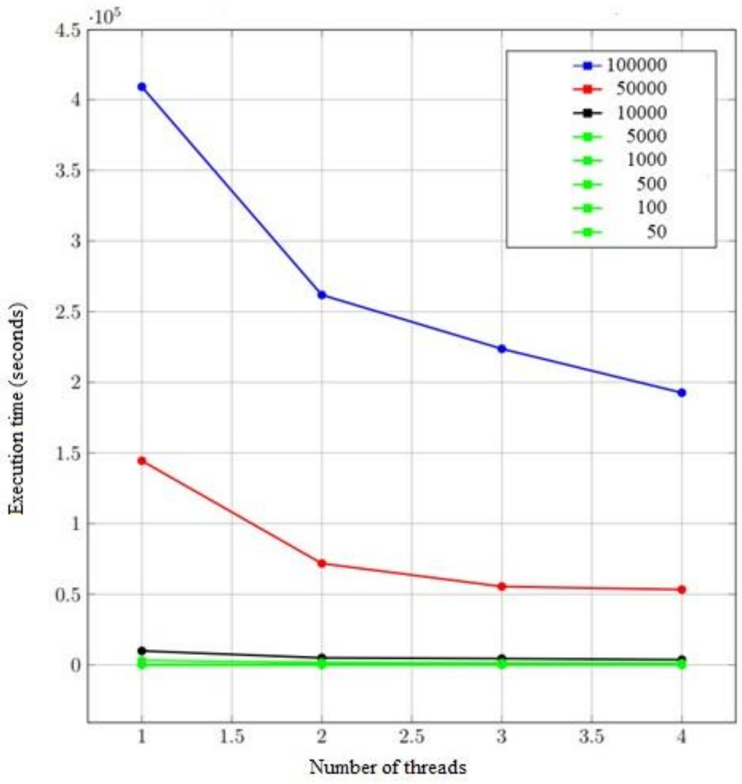


Figure 6: Execution time of parallel algorithm depending on different number of threads and persons

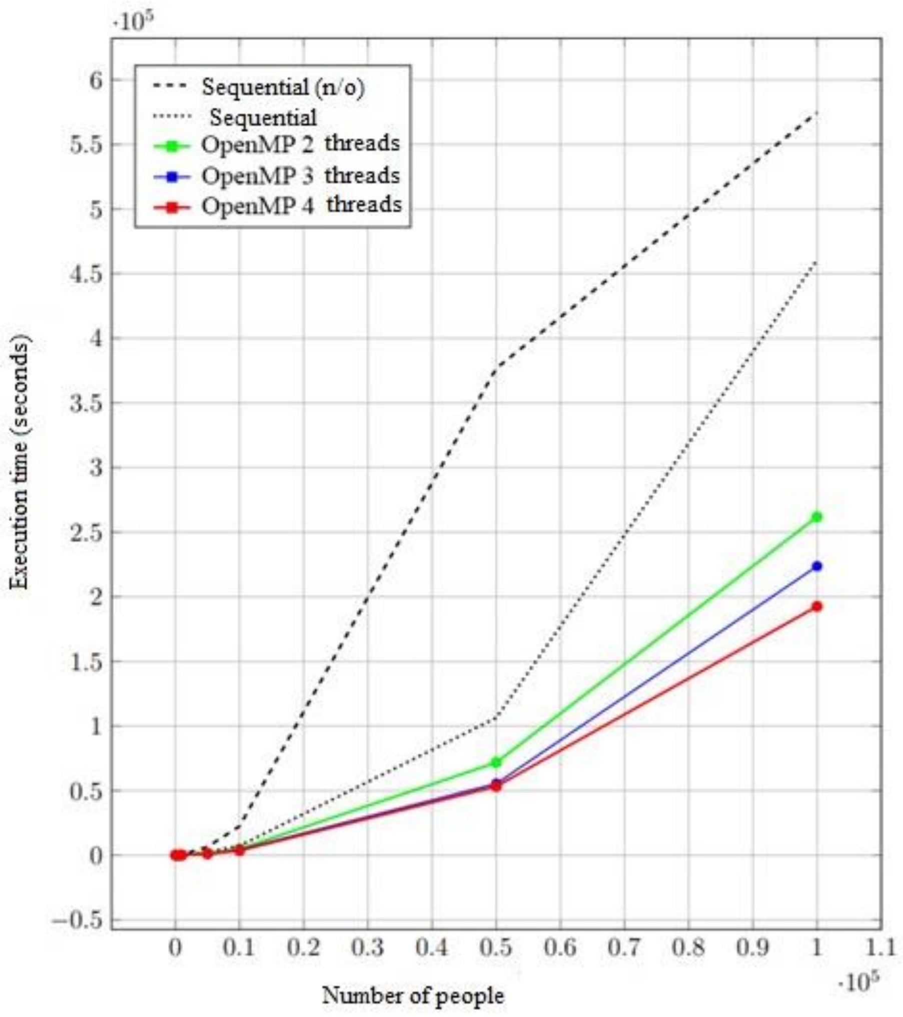


Figure 7: Execution time of all algorithms (consecutive non-optimized, sequential optimized and parallel) depending on the number of people

Table 4 shows the acceleration $S_p(n)$ [18] for different numbers of people and threads for a 4-core processor.

Table 4
Acceleration for 4 cores

Number of people	1 thread	2 threads	3 threads	4 threads
50	1.05	0.96	1.02	1.01
100	1.05	1.11	1.09	1.08
500	1.1	1.41	1.38	1.47
1000	1.02	1.39	1.43	1.67
5000	0.91	1.63	2.09	2.5
10000	0.74	1.48	1.65	2.02
50000	0.71	1.48	1.92	1.97
100000	1.13	1.75	2.05	2.38

After that, we get an acceleration of 2.5 times for 4 cores.

Table 5 shows the efficiency $E_p(n)$ [4] for different numbers of people and threads for a 4-core processor.

Table 5
The efficiency for 4 cores

Number of people	1 thread	2 threads	3 threads	4 threads
50	0.2625	0.24	0.255	0.2525
100	0.2625	0.2775	0.2725	0.27
500	0.275	0.3525	0.345	0.3675
1000	0.255	0.3475	0.3575	0.4175
5000	0.2275	0.4075	0.5225	0.625
10000	0.185	0.37	0.4125	0.505
50000	0.1775	0.37	0.48	0.4925
100000	0.2825	0.4375	0.5125	0.595

Table 6 shows the acceleration for a 2-core processor.

Table 6
Acceleration for 2 cores

Number of people	1 thread	2 threads	3 threads	4 threads
50	0.86	0.64	0.79	0.71
100	0.87	0.98	0.81	0.78
500	0.83	1.12	1.24	1.29
1000	0.93	1.01	1.14	1.37
5000	0.69	1.42	1.86	2.23
10000	0.51	1.17	1.39	1.91
50000	0.41	1.2	1.71	1.74
100000	1.02	1.53	1.79	2.11

We can conclude that when using calculations in a 2-core processor, the acceleration obtained by parallel algorithm optimization is close to 2.

Table 7 shows the efficiency for a 2-core processor.

Table 7
The efficiency for 2 cores

Number of people	1 thread	2 threads	3 threads	4 threads
50	0.215	0.16	0.1975	0.1775
100	0.2175	0.245	0.2025	0.195
500	0.2075	0.28	0.31	0.3225
1000	0.2325	0.2525	0.285	0.3425
5000	0.1725	0.355	0.465	0.5575
10000	0.1275	0.2925	0.3475	0.4775
50000	0.1025	0.3	0.4275	0.435
100000	0.255	0.3825	0.4475	0.5275

As a result of numerical experiments, we have been achieved the results presented in Table 8.

Table 8
View of the out.csv file with the results of modeling the spread of infectious diseases using OpenMP

epoch	hitcount	infectedcount	recoveredcount
0	16	16	0
1	23	23	0
2	44	44	0
3	70	70	0
4	95	95	0
5	121	121	0
6	144	144	0
7	174	158	16
8	194	171	23
9	218	174	44
10	244	174	70
11	258	163	95
12	267	146	121
13	280	136	144
14	288	114	174
15	299	105	194
16	306	88	218
17	316	72	244
18	326	68	258
19	334	67	267
20	338	58	280
21	342	54	288
22	347	48	299
23	355	49	306
24	361	35	316
25	366	32	334
26	367	29	338
27	369	27	342
28	373	26	347
29	375	20	355

Each column of data represents the epoch, the number of contacts, the number of infected people, and the number of recovered people, respectively. From here a certain dependence between infected/recovered people can be deduced (see Figure 8).

5. Conclusion

In this paper has been simulated how dangerous infectious diseases spread among the population of an “imaginary” city. Our model was simplified: it was believed that the city is isolated from external influences, i.e. the population of the city is in contact only with each other; the physical characteristics of people were not taken into account; simulation was performed until all residents became ill and acquired immunity. A simplified model gives better results after simulating the spread of infection and does not require very powerful computing resources. The relevance of this topic can be seen even now: the world is suffering from the global pandemic of the coronavirus COVID – 19. Due to strong changes in the climate of our planet, predicting the spread of viruses is becoming increasingly difficult. In the future, due to the increase in average annual temperature, viruses, which mainly existed in the tropics, will spread to new areas. Scientists have also noted that melting glaciers lead to the emergence of a large number of ancient viruses that are not known to modern science.

Various IT technologies, including parallel computing and modeling, come to the rescue to predict the spread of infectious diseases in the future. With their help, researchers accelerate and optimize existing algorithms to obtain faster and more accurate results.

The developed parallel algorithm provides a significant advantage for users with less powerful computing systems who seek to study the process by modeling the above. With OpenMP technology, we have reduced program execution time and increased support for various types of hardware. This makes it possible to get the result in real-time.

The proposed approach can be used to optimize the solution of problems described in [19-21]. In general, the research conducted in this work can be continued by expanding the criteria for modeling and choosing other technologies of distributed and parallel computing, i.e. taking into account the non-simplified model of the spread of infectious diseases. And given the fact that the city may not be isolated from external influences, that is, the city population may be in contact with the population of other cities; it must be taken into account the physical characteristics of people and so on.

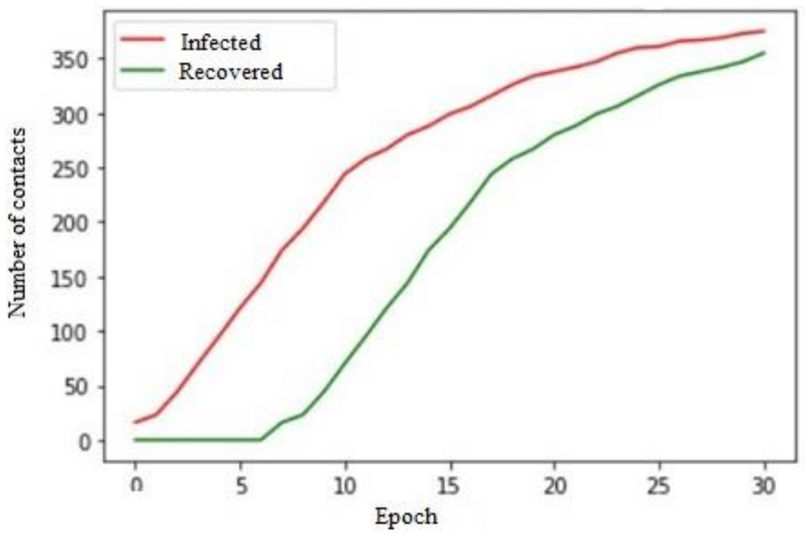


Figure 8: The results of modeling the spread of the disease using OpenMP

6. References

[1] Yadigar Sekerci, Adaptation of species as response to climate change: Predator-prey mathematical model, J. AIMS Mathematics, 2020, 5(4): 3875-3898. doi: 10.3934/math.2020251.

[2] A. Buis, Study Confirms Climate Models are Getting Future Warming Projections Right, January 9, 2020, NASA's Jet Propulsion Laboratory. URL:http://climate.nasa.gov/news/2943/study-confirms-climate-models-are-getting-future-warming-projections-right/.

[3] K.I. Kellermann, E.N. Bouton, S.S. Brandt, Is Anyone Out There? In: Open Skies. Historical & Cultural Astronomy, Springer Cham, 2020. doi:10.1007/978-3-030-32345-5_5.

[4] L. Mochurad, N. Kryvinska, Parallelization of Finding the Current Coordinates of the Lidar Based on the Genetic Algorithm and OpenMP Technology J. Symmetry 13, 666 (2021). doi:10.3390/sym13040666.

[5] Nugool Sataporn, Worasait Suwannik, and Montri Maleewong, Parallel Algorithms of Well-Balanced and Weighted Average Flux for Shallow Water Model Using CUDA, J. Modelling and Simulation in Engineering, volume 2021, article ID 9534495, 18 pages. doi:10.1155/2021/9534495.

[6] J. Brozovsky, R. Cajka, Z. Neuwirthova, Parallel Code Execution as a Tool for Enhancement of the Sustainable Design of Foundation Structures, J. Sustainability13, 1145 (2021). doi: 10.3390/su13031145.

[7] L. Mochurad, M. Yatskiv, Simulation of a Human Operator's Response to Stressors under Production Conditions, in: Proceedings of the 3rd International Conference on Informatics & Data-Driven Medicine. Växjö, Sweden, November 19 - 21, 2020, pp. 156-169.

[8] L. Mochurad, Ya. Hladun, Modeling of Psychomotor Reactions of a Person Based on Modification of the Tapping Test, J. International Journal of Computing, 20(2) (2021). doi: 10.47839/ijc.20.2.2166.

[9] I. Kurane, The effect of global warming on infectious diseases, J. Osong Public Health Res Perspect 1(1) (2010). doi:10.1016/j.phrp.2010.12.004.

[10] N. Lapidus, F. Carrat, WTW – an algorithm for identifying "who transmits to whom" in outbreaks of interhuman transmitted infectious agents, J. Am. Med. Inform. Assoc. 17(3) (2010) 348-353. doi:10.1136/jamia.2009.002832.

[11] W. Garira, A complete categorization of multiscale models of infectious disease systems, Journal of Biological Dynamics, 11:1 (2017), 378-435, doi:10.1080/17513758.2017.1367849.

[12] L. Primavera, E. Florio, A Hybrid MPI-OpenMP Parallel Algorithm for the Assessment of the Multifractal Spectrum of River Networks, J. Water 13, 3122 (2021). doi:10.3390/w13213122.

[13] J.M. Ahn, H. Kim, J.G. Cho, T. Kang, Y.-S. Kim, J. Kim, Parallelization of a 3-Dimensional Hydrodynamics Model Using a Hybrid Method with MPI and OpenMP, J. Processes 9, 1548 (2021). doi:10.3390/pr9091548.

[14] Mile Sikic, CUDA implementation of the algorithm for simulating the epidemic spreading over large networks. Conference: MIPRO 2012 – 35th International Convention on Information and Communication Technology, Electronics and Microelectronics – Proceedings, 2012, 4 p.

[15] Seonmyeong Bak, Colleen Bertoni, Swen Boehm et al. OpenMP application experiences: Porting to accelerated nodes, Parallel Computing, Volume 109, 102856, (2022), doi: 10.1016/j.parco.2021.102856.

[16] J. Zhao, M. Zhang and H. Yang, "Code Refactoring from OpenMP to MapReduce Model for Big Data Processing," 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 2019, pp. 930-935, doi: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00186.

[17] T. Perciano, C. Heinemann, D. Camp, B. Lessley, & E.W.Bethel, Shared-Memory Parallel Probabilistic Graphical Modeling Optimization: Comparison of Threads, OpenMP, and Data-Parallel Primitives, High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings, 12151(2020), pp. 127–145. doi:10.1007/978-3-030-50743-5_7.

[18] S. Bak, C. Bertoni, S. Boehm and etc, OpenMP application experiences: Porting to accelerated nodes, J. Parallel Computing, 109(2022), 12 p.

[19] I. Izonin, R. Tkachenko, N. Kryvinska, Kh. Zub, O. Mishchuk, T. Lisovych, Recovery of incomplete IoT sensed data using high-performance extended-input neural-like structure, Procedia Computer Science, volume 160 (2019), pp. 521–526.

[20] Lujic, V. De Maio and I. Brandic, Adaptive Recovery of Incomplete Datasets for Edge Analytics, 2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), (2018), pp. 1-10, doi: 10.1109/CFEC.2018.8358726.

[21] Y. Zhang, W. Yu, L. He, et al., Signals classification based on IA-optimal CNN, Neural Comput & Applic 33**,** 9703–9721 (2021). https://doi.org/10.1007/s00521-021-05736-x.