Multi-Agent Simulation of Epidemics' Distribution on Modern Supercomputers

Svetlana Lapshina

January 2, 2021

Multi-Agent Simulation of Epidemics' Distribution on Modern Supercomputers

Lapshina S.Yu. [1][0000-0002-2570-5811]

[1]Joint Supercomputer Center of the Russian Academy of Sciences - Branch of Federal State Institution «Scientific Research Institute for System Analysis of the Russian Academy of Sciences» (JSCC RAS - Branch of SRISA), Leninsky, 32 a, Moscow, Russia, 119334, jscc@jscc.ru

Abstract. The possibility of using modern supercomputers in solving the resource-intensive tasks of multi-agent modeling of the spread of mass epidemics based on the theory of growth of percolation clusters is considered in this article.

The determination of quarantine zones during the spread of epidemics is based on the multi-agent percolation model. It includes the formation of a lattice of interaction between representatives of the population, modeling the spread of the disease, collecting information about the population, implementing a parallel algorithm for multiple labeling of percolation clusters with a linking technique for labels, visualizing the results.

The article describes a variant of the algorithm for multiple labeling of Hoshen-Kopelman percolation clusters, improved for use on a multiprocessor system, and the current prototype of its implementation developed at the JSCC RAS - Branch of SRISA. The input to this algorithm is data in a format independent of the application. Therefore, it can be used in any field as a tool for differentiating large lattice clusters.

The article provides estimates of the execution time of the multiple labeling algorithm for Hochen-Kopelman percolation clusters for various input parameters on the four main high-performance computing systems installed in the JSCC RAS - Branch of SRISA.

Keywords: multi-agent simulation, theory of percolation, percolation’s cluster, epidemics' distribution, high-performance computing systems.

[1] The work was done within the framework of the state assignment on the topic «Research and development of methods and means of organizing high-performance computing, creating, processing, storing and distributing big data and digital content in distributed information and computing environments» (No. 0065-2019-0014).

1 Computer models of epidemic spread

The classical models of the spread of large-scale epidemics, based on integro-differential equations, are completely deterministic. However, the nature of epidemic processes is stochastic in nature. The process of development of these processes is influenced by a whole range of random factors that determine the random nature of the spread of the epidemic itself.

Models of computer simulation deserve special attention today, as they best meet the characteristics of studying the behavior of complex processes and phenomena that accompany the formation, development, and spread of mass epidemics and pandemics.

Simulation modeling refers to the development, execution on a computer of a software system that reflects the behavior and structure of the simulated object, process or phenomenon and the analysis of the results of computer experiments.

Simulation has significant advantages over analytical modeling when:

• relations between variables in the model are non-linear, and therefore analytical models are difficult or impossible to build;

• the model contains stochastic components;

• to understand the behavior of the system, visualization of the dynamics of the processes occurring in it is required;

• the model contains many parallel-functioning interacting components.

According to one of the classifications, simulation modeling includes four directions: modeling of dynamic systems, discrete-event modeling, system dynamics, and agent modeling.

1.1 Agent-based approach to the development of simulation models of epidemic spread and percolation theory

The agent approach in the development of simulation models is used in all cases when it is the individual behavior of objects that is essential in the system, and the integral characteristics and dynamics of the whole system are derived from these individual features. An agent is understood as a certain entity that has activity, autonomous behavior, can make decisions in accordance with certain rules, can interact with the environment and other agents, and can also change (evolve). Agent models are used to study decentralized systems, the dynamics of which are determined not by global rules by laws, but rather, these global rules and laws are the results of individual activity of an agent or group of agents. The purpose of the agent model is to get an idea of these global rules, based on the assumptions about the individual, private behavior of its active objects and the interaction of these objects in the system.

The logic of agent behavior and their interaction can not always be expressed by purely graphical means; here you often have to use program code.

When studying the behavior of complex systems, an abrupt and avalanche-like change in the characteristics of the behavior of the system under the influence of smooth changes in one or more basic parameters is often observed.

So, relatively small changes in the values of one or several parameters when studying the spread of mass epidemics (for example, the probability of infection) can lead to an abrupt change in the behavior of the entire system (a disease from a local stage takes on a planetary scale).

On January 30, 2020, at a meeting of the World Health Organization (WHO) Emergency Committee, the outbreak of the new Coronavirus was recognized as a public health emergency of international concern. WHO therefore have made the assessment that COVID-19 can be characterized as a pandemic.

As of September 21, 2020, 30 949 804 cases of infection worldwide, 959 116 deaths were confirmed [1]. In the USA - 6 703 698 cases of infection, 198 094 deaths, in the India - 5 487 580 cases, 87 882 deaths, in the Brazil - 4 528 240 cases, 136 532 deaths. In Europe, the majority of cases occurred in the Russian Federation - 1 109 595 cases of infection, 19 489 deaths.

When analyzing the spread of Coronavirus disease 2019 (COVID-19) and the epidemics of past years, it is clearly seen that at some point an abrupt transition occurs in the spread of the epidemic.

One of the interesting and effective tools for studying such situations is the method of constructing and analyzing the growth of percolation clusters.

In the theory of percolation, a number of rigorous statements are proved that describe processes on virtual lattices. To implement applications of large-scale percolation processes, the capabilities of modern supercomputer technologies are effectively used. Which in turn requires the development of appropriate models of computer simulation, including multi-agent [2-4], as well as specialized parallelization algorithms [5-6].

Using the provisions of percolation theory a working prototype for the implementation of the Hoshen-Kopelman's cluster multiple labeling technique (CML technique) [7] in C with the MPI library was developed at the JSCC RAS - Branch of SRISA. It allows you to select connected subgraphs (clusters) of a graph (lattice), as well as to find out which cluster a particular node of the lattice belongs to. The algorithm has been adapted for use on a multiprocessor system. This version of the algorithm coincides with the single-processor version with one exception: instead of passing through the entire lattice, each processor activates the CML technique only on the assigned group of nodes with the subsequent exchange of information between the processors.

The developed prototype was used to study the emergence and spread of large-scale epidemics [8]. To form the analyzed lattice, publicly available demographic data on the population of cities around the world and some assumptions about the intensity and nature of the interaction between people were used. To obtain information on the population of cities, an algorithm was developed for processing and analyzing population density maps (implementation in the java). Maps were borrowed from Google using the Google Maps API, and for the convenience of perceiving the results of simulation experiments, appropriate visualization tools were developed.

1.2 Schematic diagram of the prototype

The prototype of the implementation of the CML technique consists of the steps shown in Fig. 1:

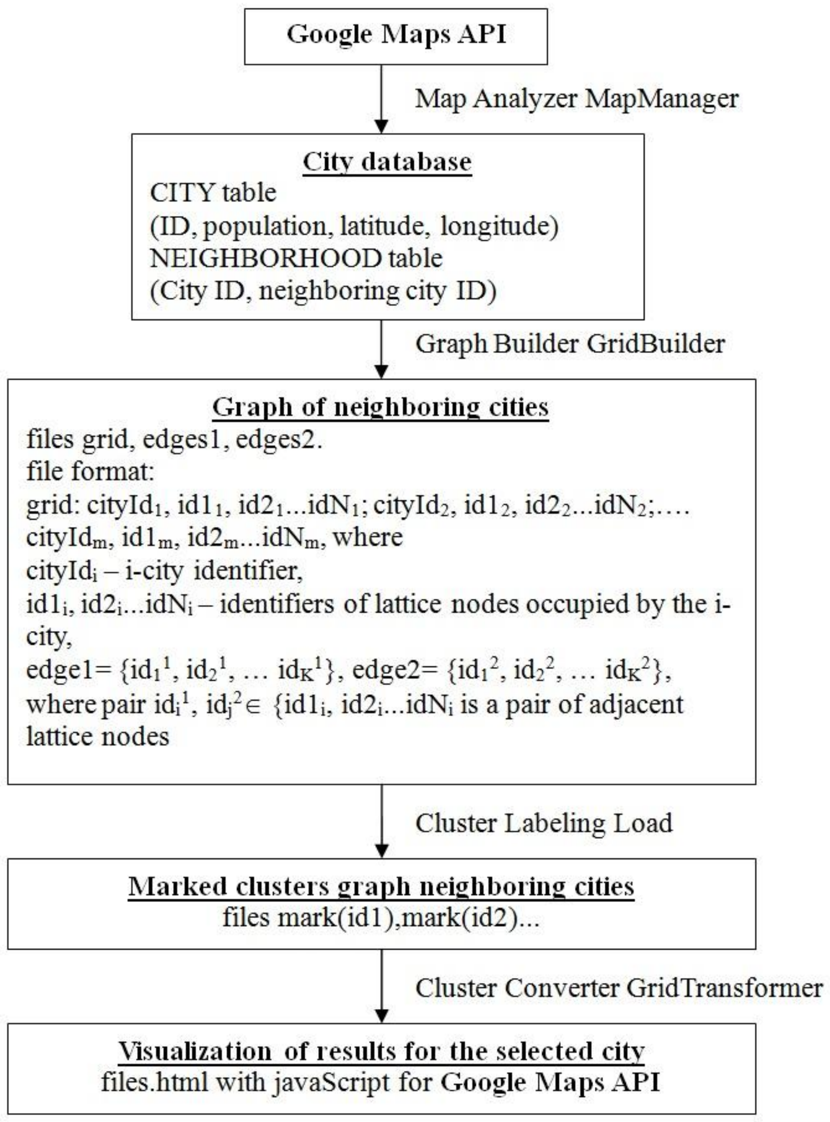
Google Maps API
Map Analyzer MapManager
City database
CITY table
(ID, population, latitude, longitude)
NEIGHBORHOOD table
(City ID, neighboring city ID)
Graph Builder GridBuilder
Graph of neighboring cities
files grid, edges1, edges2.
file format:
grid: cityId$_1$, id1$_1$, id2$_1$...idN$_1$; cityId$_2$, id1$_2$, id2$_2$...idN$_2$;….
cityId$_m$, id1$_m$, id2$_m$...idN$_m$, where
cityId$_i$ – i-city identifier,
id1$_i$, id2$_i$...idN$_i$ – identifiers of lattice nodes occupied by the i-city,
edge1= {id_1^1, id_2^1, … id_K^1}, edge2= {id_1^2, id_2^2, … id_K^2},
where pair id_i^1, $id_j^2 \in$ {id1$_i$, id2$_i$...idN$_i$ is a pair of adjacent lattice nodes
Cluster Labeling Load
Marked clusters graph neighboring cities
files mark(id1),mark(id2)...
Cluster Converter GridTransformer
Visualization of results for the selected city
files.html with javaScript for Google Maps API

Fig. 1. *Scheme of the prototype.*

Using the algorithm for collecting information about the population (MapManager map analyzer), data on cities in the world is collected and stored in a database in the format «city number, population, latitude, longitude».

The algorithm for generating a lattice of interaction between representatives of the population (the graph builder GridBuilder) creates the original lattice (implementation in java) and is stored in three files: the grid lattice and 2 edges1 and edges2 files, edges. For each value of the input variable parameter of the probability of infection in contact with the patient (parameter p = 0.01 -... 1.00 in increments of 0.01) of the original lattice, the analyzed lattice is formed in the main memory.

Next, the graph of neighboring cities is divided into related subsets by the Hochen-Kopelman algorithm (labeling Load clusters). For each analyzed lattice, the CML algorithm is launched. The algorithm can be divided into 3 stages:

1. Initialization: the first process loads the lattice into RAM from a file, converts it according to input parameters, distributes nodes into groups by processes and sends them. Other processes are waiting for their group of nodes. Having received such a group, processes distinguish from them a subgroup of externally connected nodes, i.e. nodes associated with nodes from groups of other processes. For each node in its group, initial label values are set in accordance with the absolute (within the entire lattice) node number. Each process creates a group of external nodes associated with the nodes of its group, and initializes them with labels of the connected nodes of its group.

2. The operation of the algorithm itself. Each process launches the MMK algorithm on its own group of nodes.

3. Exchange of information. It takes several steps until after the next exchange the labels of the nodes of all processes cease to change. The exchange of information can be divided into 3 stages: 1 - each process sends node label values from a subgroup of externally connected nodes to those processes with which these nodes are associated. 2 - each process takes label values from processes that have nodes associated with nodes of a given one. These values are assigned to node labels from a group of external nodes. If at least one of the node labels from the external group has been changed, then the process should repeat the exchange of information. All node labels of the group equal to the replaced labels of the external group must also be replaced. 3 - sending a message to the first process about whether or not this process should repeat the exchange of information with other processes. The first process receives similar information from all processes. If all processes do not need to be exchanged again, the first process sends a signal to everyone else to complete the work of the entire algorithm and begins the process of collecting data by the labels of their nodes. If at least one process has sent a message that it needs to continue the exchange, all processes will have to repeat the procedure for exchanging information.

As a result of labeling the clusters, an array of cluster labels was obtained (with indices from 1 to 100).

The Load cluster labeling program was launched at the Joint Supercomputer Center of the Russian Academy of Sciences - Branch of Federal State Institution «Scientific Research Institute for System Analysis of the Russian Academy of Sciences» on supercomputers with 48-304 processors. The average runtime of a program with an input parameter p from 0.01 to 1 in increments of 0.01 with constant values of t = 1, 3, 5, 10, 15, 20, 25, 30 days was about 5-10 minutes for each value t.

The next stage is a simulation experiment with a specific city (cluster converter GridTransformer). Visualization of the results obtained in the model is implemented using Google Maps Api.

The city — the source of infection — is selected, the number of infected nodes in this city was set, and for the data p and t, the calculated earlier array of cluster labels was loaded. After determining the labels of infected nodes of the selected city, the labels of all nodes from the loaded array are sequentially traversed, and if the label is "potentially infected", then the city in which the node with this label belongs is also declared potentially infected. Thus, a list of potentially infected cities is formed. This list is saved as java-script commands of the Google Maps API in an html file. Depending on the population, the infected city is marked with a red circle of a certain size. To visualize the results, the html file is launched in the browser.

2 Simulation experiments on supercomputers

An important point in the work of the CML algorithm is the correct selection of the number of processor cores on which the processing of the original lattice will be performed.

During the operation of the algorithm, it is loaded into the RAM of the node and, taking into account its large size, it would be logical to parallelize the process of its processing into a large number of parts.

But on the other hand, during the operation of the algorithm, it is necessary to exchange data between the boundary cells of the parts of the original lattice. And if there are too many such parts, then the data exchange time may exceed the allotted time limit for processing the task.

When running the algorithm on a supercomputer, it is necessary to find a balance between an increase in the number of requested computing cores and delays associated with the exchange of data between boundary cells.

2.1 Supercomputer Systems on which the simulation experiment was conducted

To study the optimal number of requested processor cores to run the algorithm, the Load cluster labeling program was launched in the JSCC with an input probability parameter p from 0.01 to 1 in increments of 0.01 with constant model times t = 30 days on 48-304 processor cores on the following supercomputers: MVS -10P OP, MVS-10P MP2, MVS-10P Tornado, MVS-100K.

MVS-10P OP is provided to users of the Center in the mode of collective access to three sections: Haswell, Broadwell and Skylake:

• Haswell (42 computing modules based on Intel Xeon E5-2697 v3 processors, 128 GB of RAM per module, peak module performance is 1.1648 TFLOPS, 1176 cores in a section);

• Broadwell (136 computing modules based on Intel Xeon E5-2697 v4 processors, 128 GB of RAM per module, peak module performance - 1.3312 TFLOPS, 4352 cores in a section);

• Skylake (58 computing modules based on Intel Xeon Gold 6154 processors, 192 GB of RAM per module, peak module performance is 3.456 TFLOPS, 2088 cores per section).

Common to installations on the MVS-10P OP is the use of the Intel Omni-Path low latency network as the communication medium.

MVS-10P MP2 KNL is a supercomputer of 38 computing modules based on Intel Xeon Phi 7290 processors, 96 GB of RAM per module, peak module performance is 3.456 TFLOPS, 2736 cores in the system.

MVS-10P Tornado is a supercomputer of 207 computing modules, each module has 2 Xeon E5-2690 processors, 64 GB of RAM, two Intel Xeon Phi 7110X coprocessors, peak module performance is 371.2 GFLOPS, 3312 cores in the system.

MVS-100K is a supercomputer of 110 computing modules based on Intel Xeon E5450 processors, 8 GB of RAM per module, peak module performance is 96 GFLOPS, 880 cores in the system.

2.2 Analysis of the optimal choice of the number of processor cores

Figure 2 shows a graph of the load program operating time versus the number of requested processor cores on various sections of the MVS-10P OP.

At MVS-10P OP, the average calculation time was:

• Haswell section - 360 sec;

• Broadwell section - 376 sec;

• Skylake section - 417 sec.

Minimum runtime:

• Haswell section - 322 sec on 128 cores;

• Broadwell section - 361 seconds on 208 cores;

• Skylake section - 371 sec on 128 cores.

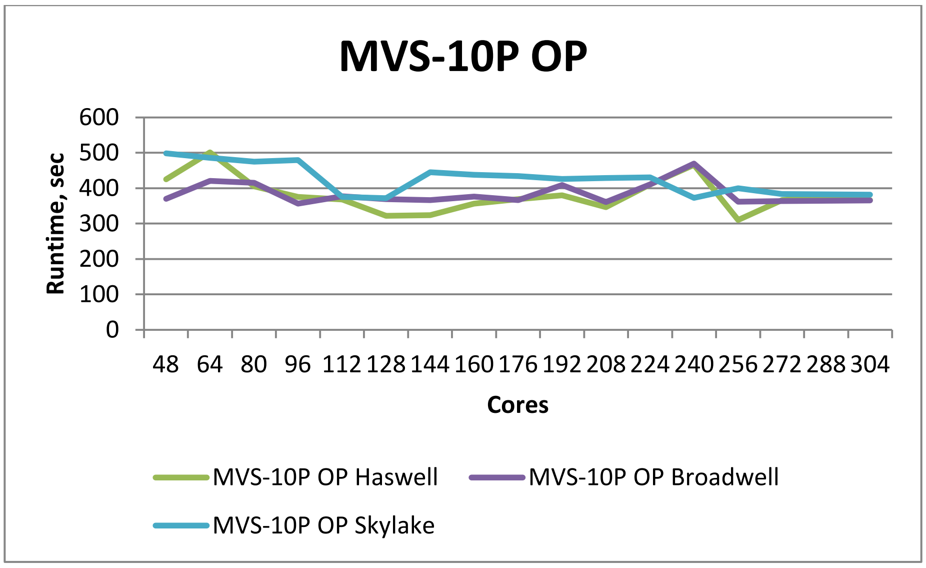

Fig. 2. Calculation on sections MVS-10P OP.

Figure 3 shows a graph of the load program operating time versus the number of requested processor cores on MVS-10P MP2 KNL. The average calculation time was 1201 seconds (almost three times longer than on any of the MVS-10P OP sections), the minimum launch time was 1172 seconds on 128 cores.

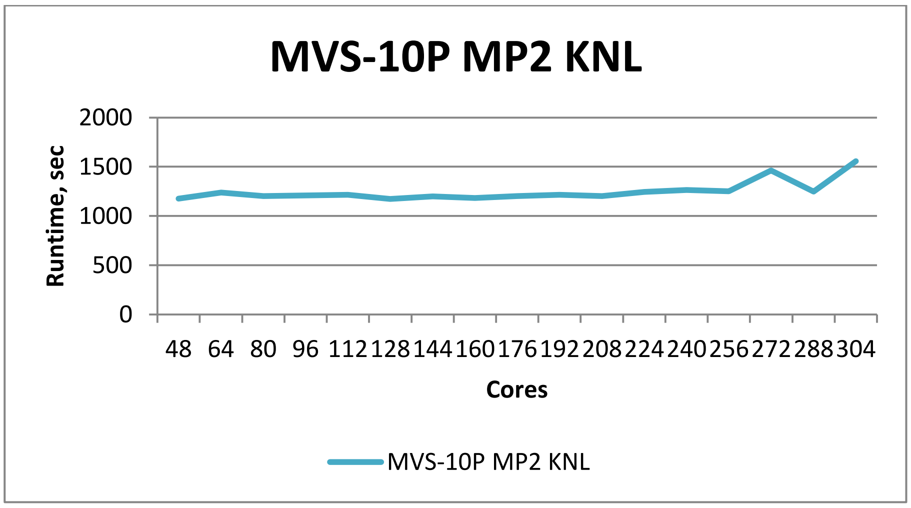

Fig. 3. Calculation on MVS-10P MP2 KNL.

Figure 4 shows a graph of the load program operating time versus the number of requested processor cores on the MVS-10P Tornado. The average calculation time was 263 seconds (approximately 25% less than on any of the MVS-10P OP sections), the minimum launch time was 235 seconds on 160 cores.

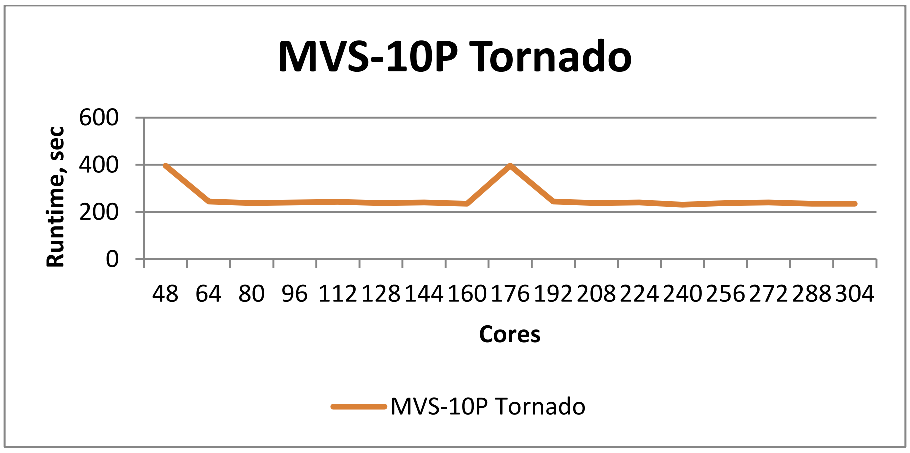

Fig. 4. Calculation on MVS-10P Tornado.

Figure 5 shows a graph of the load program operating time versus the number of requested processor cores on the MVS-100K. The average calculation time was 570 seconds (approximately 50% more than in the MVS-10P OP sections), the minimum launch time was 480 seconds on 128 cores.

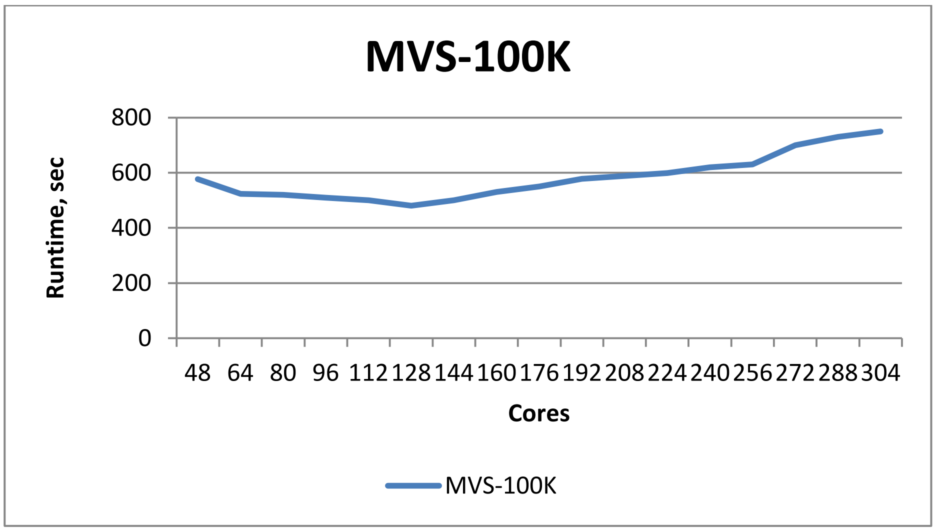

Fig. 5. Calculation on MVS-100K.

Figure 6 shows a summary graph of the load program operating time versus the number of requested processor cores on the main systems of the MSC RAS. The minimum calculation time is shown by MVS-10P Tornado. For most supercomputers, the minimum counting time is achieved using 128 - 208 cores.

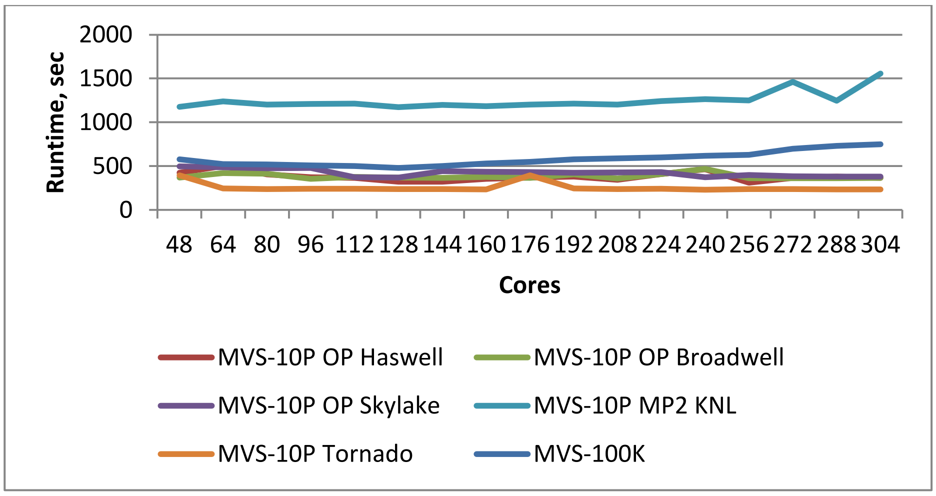

Fig. 6. A summary graph of the calculation time versus the number of requested processor cores.

The work was done at the Joint Supercomputer Center of the Russian Academy of Sciences within the framework of the state assignment on the topic «Research and development of methods and means of organizing high-performance computing, creating, processing, storing and distributing big data and digital content in distributed information and computing environments» (No. 0065-2019-0014) and in the framework of the project of the Russian Fund for Fundamental Research «Simulating the processes of spreading mass epidemics on high-performance supercomputer computer systems» (No. 19-07-00861).

The calculations were performed on the high-performance computing systems MVS-10P MP2 KNL, MVS-10P OP, MVS 10P Tornado, MVS-100K at JSCC RAS - Branch of SRISA.

References

1. World Health Organization Homepage https://www.who.int/, last accessed 2020/09/22.
2. Kalihman, R., Shebeko, Yu. Modeling the growth of percolation clusters on computers with a pronounced parallel architecture. Collection of scientific works of the Russian Academy of Sciences «Computational Technologies», Siberian Branch of the Russian Academy of Sciences., vol.4, issue 10 (1995).

3. Kondratev, M., Ivanovskij, R., Cybalova, L. Application of an agent approach to simulation modeling of the disease spread process. SPbU Scientific and Technical Bulletins. Science and Education, vol. 2, issue 2(100), pp. 189-195 (2010).
4. Tarasevich, Yu. Percolation: theory, applications, algorithms. M. URSS 2002. 64 p. (2002).
5. Utakaeva, I. Simulation of the spread of epidemics based on the agent approach. Scientific journal KubGAU, vol. 121(07) (2016). http://ej.kubagro.ru/2016/07/pdf/85.pdf, last accessed 2020/03/11.
6. Klinov, M., Lapshina, S., Telegin, P., Shabanov, B. Features of the use of multi-core processors in scientific computing. Bulletin of USATU, vol. 16, issue 6(51), pp. 25-31 (2012).
7. Lapshina, S. Parallel Cluster Multiple Labeling Technique. Lobachevskii Journal of Mathematics, vol. 40, issue 5, pp.555-561 (2019).
8. Lapshina, S. High-Performance Computations in Multi-agent Simulation Problems of Percolation Cluster's Behavior. Lobachevskii Journal of mathematics, vol. 40, issue 3, pp.341-348 (2019).