The Optimal Processor Cores' Number Research for the Parallel Cluster Multiple Labeling Technique

S. Yu. Lapshina*

(Submitted by A. M. Elizarov)

*Joint Supercomputer Center, Scientific Research Institute for System Analysis
of the Russian Academy of Sciences, Moscow, 119334 Russia*

Received May 7, 2020; revised May 30, 2020; accepted June 5, 2020

Abstract—The article is about the research of a optimum number of processor cores for launching the Parallel Cluster Multiple Labeling Technique in the course of conducting simulation experiments during multi-agent modeling of the spread of mass epidemics on modern supercomputer systems installed in the JSCC RAS. This technique may be used in any field as a tool for differentiating large lattice clusters, because it is given input in a format independent of the application. At the JSCC RAS, this tool was used to study the problem of the spread of epidemics, for which an appropriate multiagent model was developed.

DOI: 10.1134/S1995080220120240

Keywords and phrases: *multi-agent simulation, percolation's cluster, parallel cluster multiple labeling technique, high-performance computing systems, processor cores.*

1. INTRODUCTION

Infectious diseases remain one of the leading causes of premature death of people. This is mainly due to high mortality in developing countries. In industrialized countries, the availability of drugs and vaccines has led to increased confidence that the threat has almost been overcome.

However, the situation with the spread of the new coronavirus has shown that this is not so. On January 30, 2020, at a meeting of the World Health Organization (WHO) Emergency Committee, the outbreak of COVID-19 was recognized as a public health emergency of international concern. On March 11, WHO declared the coronavirus epidemic a pandemic. As of May 13, 2020, 4 170 424 cases of infection worldwide, 287 399 deaths were confirmed [1]. In China—84 458 cases, 4 644 deaths. In USA—1 322 054 cases of infection, 79 634 deaths. In Europe, the majority of cases occurred in Russian Federation—242 271 cases of infection, 2 212 deaths; in Spain—228 030 cases of infection, 26 920 deaths; in United Kingdom—226 467 cases of infection, 32 692 deaths.

When analyzing the spread of this epidemic (and the epidemics of past years), a tendency was noticed that at some point an abrupt transition occurs in the spread of the epidemic.

During modeling the processes of the spread of mass epidemics and pandemics, the effect often arises that insignificant changes in the values of one or more parameters (for example, the probability of infection of individual representatives) can lead to an abrupt change in the behavior of the entire population (a disease from a local and non-dangerous transition to the stage of a large-scale pandemic). One way to study such effects is to study the formation and growth of percolation clusters.

Studies of the properties of percolation clusters should be accompanied by the active use of a variety of computer modeling technologies, including multi-agent simulation technology [2–4].

The principal feature of multi-agent simulation in comparison with other technologies (aggregate, discrete-event, etc.) is the ability to identify and record the behavior properties of a complex system under the influence of the purely individualistic behavior of its representatives. Refining the obtained results in

*E-mail: lapshina@jssc.ru

Fig. 1. Scheme of the prototype.

order to improve the initial model sometimes requires a large number of resource-intensive simulation experiments (mainly on multiprocessor computing architectures), the use of specialized parallelization algorithms and the selection of optimal input parameters [5, 6].

In the multiagent epidemic spread model developed at the JSCC RAS—Branch of SRISA as part of the BIOCLUST software and hardware complex [7], the Hoshen—Kopelman's cluster multiple labeling technique (CML technique) is used [8].

2. THE SCHEME OF SIMULATION EXPERIMENTS

The scheme of simulation experiments carried out using multiprocessor computing systems based on the parallel CML technique consists of the steps shown in Fig. 1:

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 $p = 0.01 - \dots 1.00$ 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 Hoshen—Kopelman algorithm (labeling Load clusters). For each analyzed lattice, the CML technique is launched. 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 .

An important point in the work of the parallel CML technique 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.

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-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 javascript 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.

3. ANALYSIS OF THE PROCESSOR CORES' OPTIMAL NUMBER

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.

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 s;

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

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

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

- Broadwell section—376 s;
- Skylake section—417 s.

Minimum runtime:

- Haswell section—322 s on 128 cores;
- Broadwell section—361 s on 208 cores;
- Skylake section—371 s on 128 cores.

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 s on 128 cores.

Fig. 5. Calculation on MVS-100K.

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

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.

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.

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 JSCC. The minimum calculation time is shown by MVS-10P Tornado. For most supercomputers, the minimum counting time is achieved using 128–208 cores.

4. CONCLUSION

In the course of imitation experiments, a variant of the Parallel Cluster Multiple Labeling Technique for percolation Hoshen–Kopelman clusters related to the tag linking mechanism, which can also be used in any area as a tool for differentiating large-size lattice clusters, was used to be improved on a multiprocessor system. The article provides an estimate of the execution time of the Parallel Cluster Multiple Labeling Technique for Hoshen–Kopelman percolation clusters for various values of input parameters on high-performance computing systems installed in the JSCC RAS: MVS-10P MP2 KNL, MVS-10P OP, MVS-10P Tornado, MVS-100K.

ACKNOWLEDGMENTS

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.

FUNDING

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).

REFERENCES

1. World Health Organization Homepage. <https://www.who.int/>. Accessed May 14, 2020.
2. R. S. Kalikhman and Yu. A. Shebeko, “Simulating the growth of percolation clusters on computers with a strongly pronounced parallel architecture,” *Vychisl. Tekhnol., Sb. Nauch. Tr. SO RAN* **4** (10) (1995).
3. M. Kondratev, R. Ivanovskii, and L. Tsybalova, “Application of an agent approach to simulation modeling of the disease spread process,” *Nauch.-Tekh. Vedom. SPb Politekh. Univ.*, No. 2-2 (100), 189–195 (2010).
4. Yu. Yu. Tarasevich, *Percolation: Theory, Applications, Algorithms* (URSS, Moscow, 2002) [in Russian].
5. I. Utakaeva, “Simulation of the spread of epidemics based on the agent approach,” *Politem. Setev. El. Nauch. Zh. KubGAU* **121**, 1369 (2016). <http://ej.kubagro.ru/2016/07/pdf/85.pdf>. Accessed March 11, 2020.
6. M. Klinov, S. Lapshina, P. Telegin, and B. Shabanov, “Features of the use of multi-core processors in scientific computing,” *Vestn. UGATU* **16** (6), 25–31 (2012).
7. S. Yu. Lapshina, “High-performance computations in multi-agent simulation problems of percolation cluster behavior,” *Lobachevskii J. Math.* **40**, 341–348 (2019).
8. S. Yu. Lapshina, “Parallel cluster multiple labeling technique,” *Lobachevskii J. Math.* **40** (5), 555–561 (2019).