Adv Syst Sci Appl 2017; **4**; 34-45
Published online at http://ijassa.ipu.ru/ojs/ijassa/article/view/257

Development of the Agent-based Demography and Migration Model of Eurasia and its Supercomputer Implementation

Valery. L. Makarov, Albert. R. Bakhtizin, Elena. D. Sushko, Gennady. B. Sushko
Central Economics and Mathematics Institute CEMI RAS, Moscow, Russia
E-mail: albert.bakhtizin@gmail.com

Abstract: In this work we describe the development of a scalable agent-based modelling framework for simulation of Eurasia population described in terms of demography, migration and transport flows. The simulated system will consist of agents representing individuals and sets of links to other agents, which represent the social interactions of individual. The individual agents in the model will participate in several independent processes, for which different sets of social links is important such as family and neighbors. As a base for our simulation system we have used a combination of a base native layer implemented using C++ language which uses MPI library, and Microsoft .NET platform as an environment for model code written in high-level C# programming language. To perform a load balancing of agents between processes the METIS/ParMETIS algorithms were used. These algorithms allow to split the graph of agents and links into parts of similar size with the least possible number of links between them. A number of numerical experiments were carried out for test model to estimate the influence of the parameters of the model on its performance and parallel scalability. For each combination of parameters a number of simulations were performed to average the results.

Keywords: agent-based modelling, demography, numerical modelling, parallel computing.

2 1. INTRODUCTION

In this work we describe the development of a scalable agent-based modelling framework for simulation of population of Eurasia described in terms of demography, migration and transport flows. The goal of the simulation is to describe the influence of large transport infrastructure projects on the development of population of the region.

The simulated system consists of agents who represent individuals and a set of links to other agents, which represent the social interactions of individual. The individual agents in the model participate in several independent processes, for which different sets of social links is important such as family and neighbors.

The agents of the system participate in two processes: 1) the process of reproduction of the population, and 2) the process of migration. In the first process they use messages exchange to search for the partner to form a family. In the second process the message exchange mechanism is used to obtain information about available jobs in different regions to determine the direction of migration.

There are several tools for high performance computing for ABM.

Microsoft Axum is a domain-specific concurrent programming language, based on the Actor model that was under active development by Microsoft between 2009 and 2011. It is an object-oriented language based on the .NET Common Language Runtime using a C-like syntax which, being a domain-specific language, is intended for development of portions of a software application that is well-suited to concurrency. But it contains enough general-purpose constructs that one need not switch to a general-purpose programming language (like C#) for the sequential parts of the concurrent components.

The main idiom of programming in Axum is an Agent (or an Actor), which is an isolated entity that is being executed in parallel with other Agents. In Axum parlance, this is referred to as the agents executing in separate isolation domains; objects instantiated within a domain cannot be directly accessed from another.

Agents are loosely coupled (i.e., the number of dependencies between agents is minimal) and do not share resources like memory (unlike the shared memory model of C# and similar languages); instead a message passing model is used. To coordinate agents or having an agent request the resources of another, an explicit message must be sent to the agent. Axum provides Channels to facilitate this.

The Axum project reached the state of a prototype with working Microsoft Visual Studio integration. Microsoft had made a CTP of Axum available to the public, but it was removed later. Although Microsoft decided not to turn Axum into a project, some of the ideas behind Axum are used in TPL Dataflow in .Net 4.5 (more information at [5]).

Repast for High Performance Computing (*Repast HPC*) 2.2.0, released on 30 September 2016, is a next generation agent-based modeling and simulation (ABMS) toolkit for high performance distributed computing platforms.

Repast HPC is based on the principles and concepts development in the Repast Simphony toolkit. Repast HPC is written in C++ using MPI for parallel operations. It also makes extensive use of the boost [2] library. Repast HPC is written in cross-platform C++. It can be used on workstations, clusters, and supercomputers running Apple Mac OS X, Linux, or Unix. Portable models can be written in either standard or Logo-style C++.

Repast HPC is intended for users with:

- Basic C++ expertise.
- Access to high performance computers.
- A simulation amenable to a parallel computation. Simulations that consist of many local interactions are typically good candidates.

Models can be written in C++ or with a “Logo-style” C++ [1].

CyberGIS Toolkit is a suite of loosely coupled open-source geospatial software components that provide computationally scalable spatial analysis and modeling capabilities enabled by advanced cyberinfrastructure. CyberGIS Toolkit represents a deep approach to CyberGIS software integration research and development and is one of the three key pillars of the CyberGIS software environment, along with CyberGIS Gateway and GISolve Middleware [3].

The integration approach to building CyberGIS Toolkit is focused on developing and leveraging innovative computational strategies needed to solve computing- and data-intensive geospatial problems by exploiting high-end cyberinfrastructure resources such as supercomputing resources provided by the Extreme Science and Engineering Discovery Environment and high-throughput computing resources on the Open Science Grid.

A rigorous process of software engineering and computational intensity analysis is applied to integrate an identified software component into the toolkit, including software building, testing, packaging, scalability and performance analysis, and deployment. This process includes three major steps:

1.Local build and test by software researchers and developers using continuous integration software or specified services;

2.Continuous integration testing, portability testing, small-scale scalability testing on the National Middleware Initiative build and test facility; and

3.XSEDE-based evaluation and testing of software performance, scalability, and portability. By leveraging the high-performance computing expertise in the integration team of the NSF CyberGIS Project, large-scale problem-solving tests are conducted on various supercomputing environments on XSEDE to identify potential computational bottlenecks and achieve maximum problem-solving capabilities of each software installation.

Pandora is a novel open-source framework created by the social simulation research group of the Barcelona Supercomputing Centre. This tool is designed to implement agent-based models and to execute them in high-performance computing environments. It has been explicitly programmed to allow the execution of large-scale agent-based simulations, and it is capable of dealing with thousands of agents developing complex actions.

Pandora has full Geographical Information System support, to cope with simulations in which spatial coordinates are relevant, both in terms of agent interactions and environment. The results of each simulation are stored in hierarchical data format (HDF), a popular format that can be loaded by most GIS. This feature is particularly useful, as we will also use GIS to analyze simulation results.

Pandora is complemented by Cassandra, a program developed to analyze the results generated by a simulation created with the library. Cassandra allows the user to visualize the complete execution of simulations using a combination of 2D and 3D graphics, as well as statistical figures (more information at [6]).

SWAGES [10], a distributed agent-based life simulation and experimentation environment that uses automatic dynamic parallelization and distribution of simulations in heterogeneous computing environments to minimize simulation times.

SWAGES allows for multi-language agent definitions, uses a general plug-in architecture for external physical and graphical engines to augment the integrated SimWorld simulation environment, and includes extensive data collection and analysis mechanisms, including filters and scripts for external statistics and visualization tools. Moreover, it provides a very flexible experiment scheduler with a simple, web-based interface and automatic fault detection and error recovery mechanisms for running large-scale simulation experiments [10].

A Hierarchical Parallel simulation framework for spatially-explicit Agent-Based Models (***HPABM*** [11]) is developed to enable computationally intensive agent-based models for the investigation of large-scale geospatial problems. HPABM allows for the utilization of high-performance and parallel computing resources to address computational challenges in agent-based models.

Within HPABM, an agent-based model is decomposed into a set of sub-models that function as computational units for parallel computing. Each sub-model is comprised of a subset of agents and their spatially-explicit environments. Sub-models are aggregated into a group of super-models that represent computing tasks. HPABM based on the design of super- and sub-models leads to the loose coupling of agent-based models and underlying parallel computing architectures. The utility of HPABM in enabling the development of parallel agent-based models was examined in a case study.

Results of computational experiments indicate that HPABM is scalable for developing large-scale agent-based models and, thus, demonstrates efficient support for enhancing the capability of agent-based modeling for large-scale geospatial simulation [11].

The growing interest in ABM among the leading players in the IT industry (Microsoft, Wolfram, ESRI, etc.) definitely shows the relevance of this instrument and its big future, while exponential growth of overall data volumes related to human functioning and the need for analytical systems to obtain new-generation data needed to forecast social processes, call for the use of supercomputer technologies.

In March 2011, an ABM was launched at the Lomonosov supercomputer to simulate the development of Russia's socio-economic system for the next 50 years [8]. The implemented ABM was based on the interaction of 100 mln agents who conditionally represented Russia's socio-economic milieu. The behavior of each agent was specified by a set of algorithms that described the agent's actions and interaction with other agents in the real world.

The ADEVS library for multiagent simulation, which the authors had already tested during the multisequencing of Russia's demographic model in 2011, showed itself quite well. In addition, the latest ADEVS versions support Java to a certain extent, which is also a plus. However, the ADEVS developers have not yet implemented multisequencing on

supercomputers except for the OpenMP technology for multiprocessors; therefore, our previous work required many updates for MPI support.

When multisequencing the previous, quite simple, model, it was fully rewritten in C++, which was superfluous: the pre- and postprocessing of data, as well as the creation of the initial state of the multiagent environment, are not time-critical operations. Usually, it is quite enough for supercomputers to multisequence only the algorithm's computing core, i.e., the phase of converting the population's state in this case.

The stages and methods of the efficient reflection of the computing core of a multiagent system on the architecture of the state-of-the-art supercomputer using the **S**upercomputer **T**echnology for **A**gent-o**R**iented **S**imulation (STARS) developed by the authors are analyzed in later article [9]. Analysis of the latest software technologies has shown that embeddable tools are being actively developed lately to execute Java programs that use the so-called Ahead-Of-Time (AOT) compilation. In this case, the result of the AOT compiler is a typical self-contained executable module that contains a machine code for the target platform. It is interesting to note that this approach is used in the new versions of the Android operating system, which, in our opinion, is not accidental: the efficiency of code execution is the main factor both for embeddable systems and for supercomputers. Experiments with a similar product – the Avian AOT compiler – allowed us to conclude that, first, it helps obtain a self-contained executable module as an MPI application for supercomputers; in addition, a random additional code, including initialization and binding to the MPI communication library, is easily implemented in C++; second, the operating speed of the obtained program module is close to the speed of the ADEVS operation. This made it possible to shift a large part of the work to the AOT compiler and to implement only the most necessary in C++, fixing the function of supporting accelerated stages with complex interagent communication to ADEVS.

3 2. THE MODEL DESCRIPTION

In this work we describe the development of an agent-based modelling framework for simulation of large scale societies which consists of large number of agents. The described technology is to be applied for the implementation of the large-scale agent based model of countries of Eurasia describing economy, migration and the results of implementation of large infrastructural projects.

The main types of agents in these simulations are individuals, enterprises, regions and governments. The main processes described by the model are demographical evolution of the society, education, jobs and career of individuals and the migration of the workforce according to changing economic conditions.

Individual agents in the system are described in terms of their age, sex, education level, income and the set of social links to other agents. This set of parameters defines the formation of families, awareness about working conditions in different areas and the possibility of the labor migration. The regions in the model are characterized by the transport connectivity graph, the level of economic development and the labor market conditions.

Due to that structure of the model agents form the following graph: each agent is linked with a dozen of other individuals in the same or other regions. An individual agent is also linked to his place of work and a region. Agents describing enterprises are linked with each other by trade contracts. The described structure of the model leads to formation of a large-scale graph of agents of different types which can be partitioned into connected blocks linked to regions.

The model uses a change in the transport connectivity graph as an input which leads to the change in economic activity in neighbor regions and the migration of the population. Starting from the implementation of the infrastructural project the living conditions are

changing: new jobs are created in neighbor areas which leads to a change in incomes and migration of workforce.

2.1 The technology

On the computational level the model should be scalable for system up to 10^9 agents. In order to perform efficient simulations of such systems the model should support running on modern supercomputers. To simplify the development of the model we use the high-level Microsoft .Net platform which has become available for running on supercomputers.

The most usual architecture of modern supercomputers is a cluster of multicore computing nodes connected by high-performance low-latency network. To run efficiently on such cluster the program should be split into multiple separate processes exchanging messages through the network. The most common way of writing such programs is to use C++/Fortran language and MPI library which are available on all supercomputers.

As a base for our simulation system we have used a combination of a base native layer implemented using C++ language which uses MPI library, and Microsoft .NET platform as an environment for model code written in high-level C# programming language. As most of modern supercomputers run on Linux operating system, we have decided to use Microsoft .Net Core and MONO [4] implementations of .NET platform available for this OS.

The choice of these technologies was determined by the following criteria:

1) The system has to be scalable across multiple computational cluster nodes (i.e. use resource of multiple nodes for speedup) therefore the multithreading calculation model was not suitable as it is limited to single cluster node.
2) The model should be easy to develop and maintain and therefore the high-level programming language C# was used.
3) The system should be efficient and therefore the native MPI library was used instead of TCP/IP Sockets or .Net libraries like Windows Communication Foundation as these technologies are not optimal for supercomputers and HPC applications. MPI libraries installed on each cluster computer are usually tuned for particular proprietary network system which is used on the cluster such as Infiniband.

On the level of individual agents the simulation of agent's internal state evolution, the formation of constant and temporary links between agents, message exchange and formation and destruction of agents in the system are supported. To carry out these simulations efficiently the system has to implement a dynamic load balancing mechanism for agents taking into account their links to neighbors. The proper description and taking into account of links of agents in the process of decomposition is crucial for reduction of network exchange traffic which is necessary for scalability of the model up to hundreds and thousands of CPU cores.

To perform a load balancing of agents between processes the METIS/ParMETIS [7] algorithms were used, which are commonly used for decomposition of big graphs (up to 10^9), computational grids and matrices. These algorithms allow to split the graph of agents and links into parts of similar size with least possible number of links between them. The algorithm can be applied recursively in order to calculate hierarchical splitting of the system in efficient way. The use of the algorithm allows both initial decomposition of the system and refinement of the decomposition in the process of calculation which is necessary to maintain load balancing as new agents are added to the system and some of old agents are being removed. The dynamic decomposition and redistribution of agents should allow us to use efficiently up to 1000 CPU cores.

2.2 Implementation of the model

The implementation of the agent-based simulation platform requires the proper definition of classes for agents, messages, model, time-steps and utility classes for file input and calculation of characteristics.

An efficient mechanism of message exchange between agents was implemented by means of message queue and native MPI collective operations. The message queue accumulates a buffer of messages to different processes and then uses MPI AllToAll exchange operation to deliver contents of messages. This operation delivers the buffers of arbitrary size from each MPI process to all other processes in most efficient way by splitting the buffer into chunks of optimal size for network transfer and hiding the latency of network operations by performing simultaneous several send and receive operations.

To use native operations with managed C# objects operations of binary serialization and deserialization of objects were implemented and C# wrappers for native functions were written.

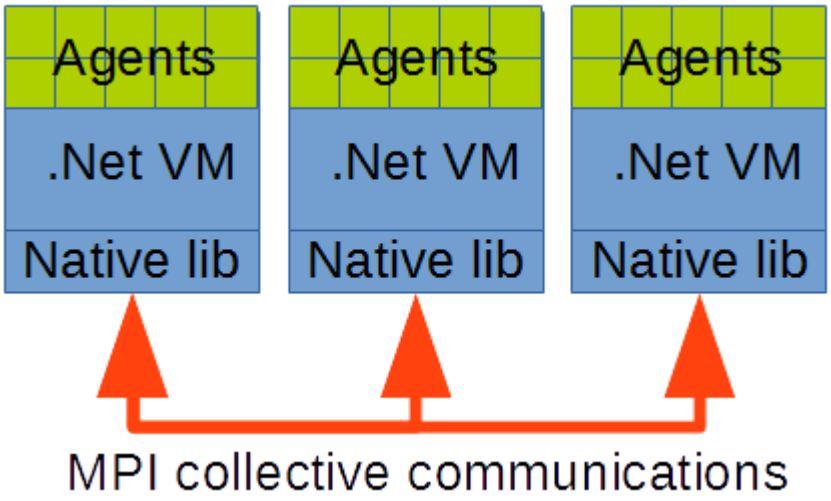

Fig. 1. The agents of the model are implemented using C# programming language and run in .Net virtual machine. The interaction between VM instances is implemented through the native library and MPI library.

The implementation of native wrappers for C++ library was carried out using InteropServices library and “DllImport” annotations in managed C# code. In Fig. 1 the scheme of interaction of model agents through native library and MPI operation is shown.

2.3 The test model description

In order to test the message delivery mechanism the test model was implemented and a set of simulations were carried out to estimate the scalability of the designed model. The model is characterized by the following parameters:

The total number of agents ***N.***

The ratio of agents participating in message exchange ***S.***

Message exchange intensity ***I***.

Total simulation time ***T***.

The number of MPI processes ***U***.

As it is usual for all MPI programs the program starts as a set of separate processes running the same program. At the initial stage of the model the initial set of agents is created with the number of agents ***N***. Each agent has the only numerical parameter - the number of messages it should send at each simulation time (***A***). The set of agents is distributed over all MPI processes, i.e. each MPI process creates only N/M agents according to its process identifier. The process with index 0 creates agents 0...N/M, the process with index 1 creates agents N/M+1...2N/M and so on.

On each simulation step for each agent of the system the random number is generated which determines if the agent will participate in message sending process (the probability of the event is ***S/N***).

If the agent is sending messages on this simulation step, that the random number of messages ***A*** is generated with uniform distribution of probability between 1 and ***I***. After that

the agent sends ***A*** messages to agents with random numbers and then receives replies. Each simulation step consists of 5 stages:

1. The loop over all agents on the current process, execution of the performStep method of each agent which results in generation of random messages according to corresponding sending probabilities. All messages are put into outgoing message queue.
2. After the generation of all initial messages the method sendReceiveMessages is called which initiates the exchange of the parts of message queue between all MPI processes using collective AllToAll operation. Each process is sending data buffers to all other processes and receives corresponding buffers from all other processes. At this stage the messages in the queue are serialized into binary arrays, these arrays are transmitted into the native library which uses MPI library to perform the exchange, after that new buffers are transmitted to C# part of the program and messages are deserialized.
3. After the exchange of buffers and deserialization of all messages the delivery of messages to corresponding agents on each MPI process is performed which results to the generation of new set of reply messages.
4. The exchange of the parts of message queue between all MPI processes using collective AllToAll operation. Each process is sending data buffers to all other processes and receives corresponding buffers from all other processes.
5. The delivery of messages to corresponding agents on each MPI process.

The use of the delayed delivery of messages through the message queue allows us to optimize the message exchange which is now bound not to the latency of the network but to its bandwidth.

After processing all agents in the population, the output characteristics are calculated and put into output file. The following output parameters of the model were written:

1. Step number;
2. The average number of message recipients for agents participating in message exchange;
3. The total calculation wall time.

2.4 Messages exchange procedure

The message queue implemented in the program is a managed C# object which receives objects of abstract class Message from agents, each MPI process contains one instance of the message queue object. The simulation model consists of agents of different types and messages can also have different types derived from the Message class. For each message type the operations of reading and writing to binary stream are implemented which use BinaryReader and BinaryWriter classes of C# standard library. These operations encode and decode the type of message object, the numbers of the sender and receiver objects and all additional message data fields into binary format.

Each message queue contains a set of BinaryWriter objects which are used as buffers for outgoing messages. For each outgoing message put into queue the number of destination process is determined, that the message is written to the corresponding binary stream.

The message queue implements a sendReceiveMessages method which performs the delivery of all messages to the corresponding agents. In order to deliver messages message queue objects of all MPI processes exchange the corresponding message buffers, for that all BinaryWriter objects are written to MemoryStream object which generates an outgoing array of bytes. In order to deliver the outgoing array of bytes first the MPI_Alltoall function is used to exchange the sizes of all message buffers between all processes. After that the MPI_Alltoallv function is used to deliver parts of the outgoing byte array to the corresponding processes.

After the exchange of the message buffers each message queue has separate buffers with messages from all other processes, i.e. process 0 has K1 bytes from process 1, K2 bytes from process 2… For all of these buffers the BinaryReader objects are created and the deserialization process is started. The program reads messages from all incoming buffers and generates an array of Message objects. For each of these messages the receiver agent index is determined, and for the corresponding agent the notify method is called with the corresponding message object passed as argument. The process of the delivery of the message is illustrated in Fig. 2.

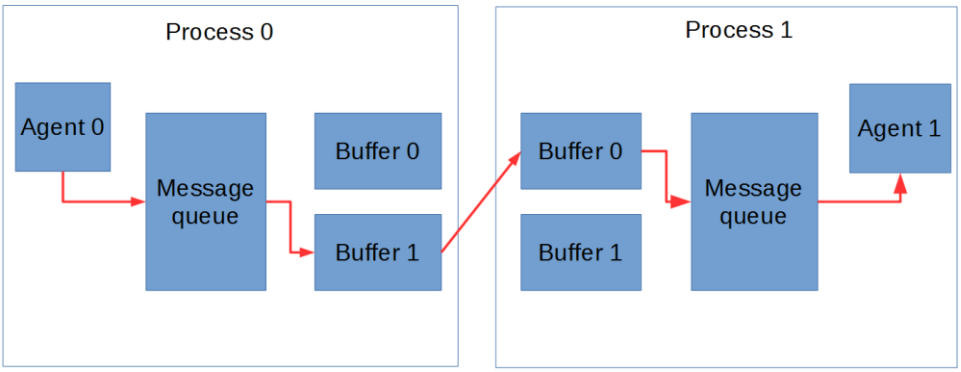

Fig. 2. The message sending procedure. The message from Agent 0 is transmitted first to Message queue of process 0, then the message is serialized to outgoing buffer for sending to process 1, then the buffers are exchanged between processes and the outgoing buffer number 1 becomes the incoming buffer number 0 on MPI process number 1, then the message is deserialized and delivered to Agent 1 by the message queue.

3. NUMERICAL EXPERIMENTS

A number of numerical experiments was carried out to estimate the influence of the parameters of the model on its performance and parallel scalability. For each combination of parameters a number of simulations was performed to average the results.

3.1 Test cluster configuration

To test the performance and scalability of the model we have used the cluster consisting of 4 dual-processor nodes using AMD Opteron 6172 (12 cores) processors. The total number of cores in each node was 24 and the same number of MPI processes on each node was used. The high-performance network (QDR Infiniband) was used to connect nodes of the cluster.

3.2 The results of the numerical experiments

To study the parallel efficiency of the model the following test configuration was used:

$\boldsymbol{N}$ = 10000000 (the number of agents)
$\boldsymbol{S}$ = 200000 (the number of agents sending messages on the simulation step)
$\boldsymbol{I}$ = 10 (the maximal number of messages for one agent on each step)
$\boldsymbol{T}$ = 3000 (the number of simulation steps)

The simulations were carried out for the number of MPI processes $\boldsymbol{M}$ = 1,24,48,96 and results of these simulations were compared in terms of parallel speedup (the ratio of total computation time for parallel and serial cases) and parallel efficiency (the speedup divided by number of CPU cores used).

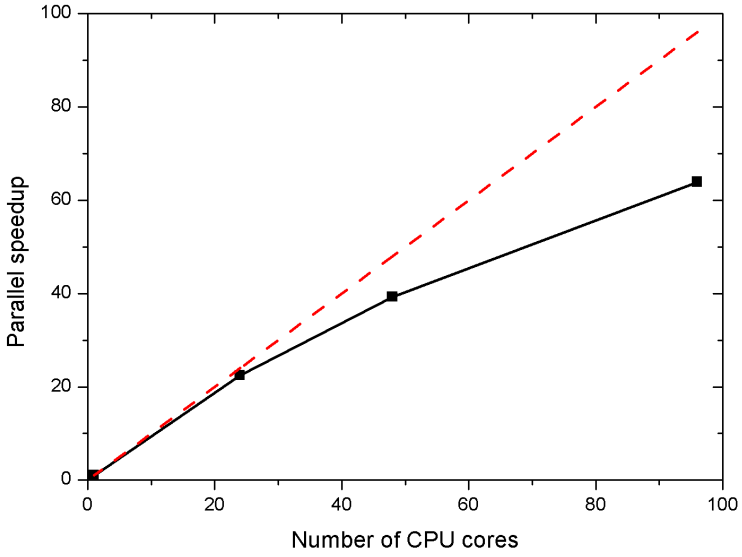

Fig. 3. The dependence of the parallel speedup of the test model on the number of CPU cores.

In Fig. 3 the dependence of the parallel speedup on the number of CPU cores is shown. The increase of the number of CPU cores leads to nearly linear speedup of the calculations. The use of 96 cores results to speedup of calculations by factor of 60 which means 65% efficiency of the cluster use. The dependence of the parallel efficiency on the number of CPU cores is shown in Fig. 4.

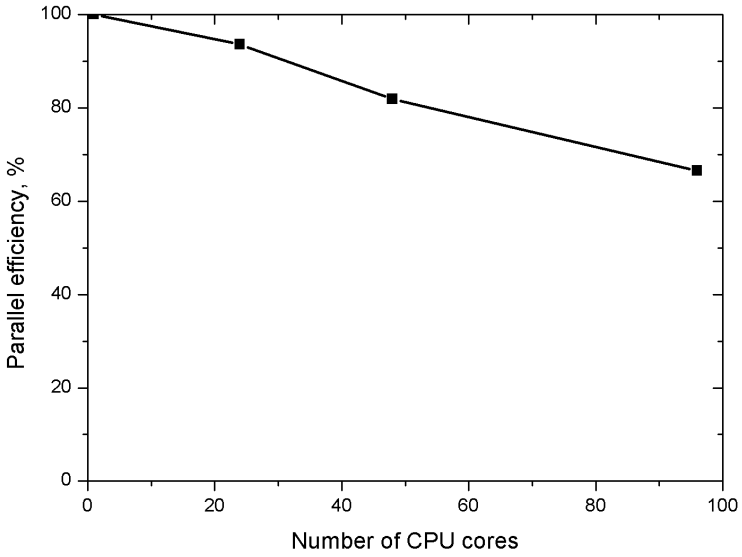

Fig. 4. The dependence of the parallel efficiency of the simulation on the number of MPI processes.

To estimate the influence of the parameters of the model on scalability the simulations were performed with parameter values $\boldsymbol{N}$ = 10000000, $\boldsymbol{S}$ = 200000, $\boldsymbol{T}$ = 3000 using $\boldsymbol{M}$ = 96 MPI processes with different values of message exchange intensity $\boldsymbol{I}$ = 30, 50, 100, 200, 1000.

The increase of intensity of message exchange leads to linear increase of the network traffic on each simulation step, and also increase of computations (random number generation) on each simulation step. In Fig. 5 the dependence of the speedup on message

exchange intensity shows that these factors are balanced and increase of intensity doesn't lead to degradation of parallel efficiency.

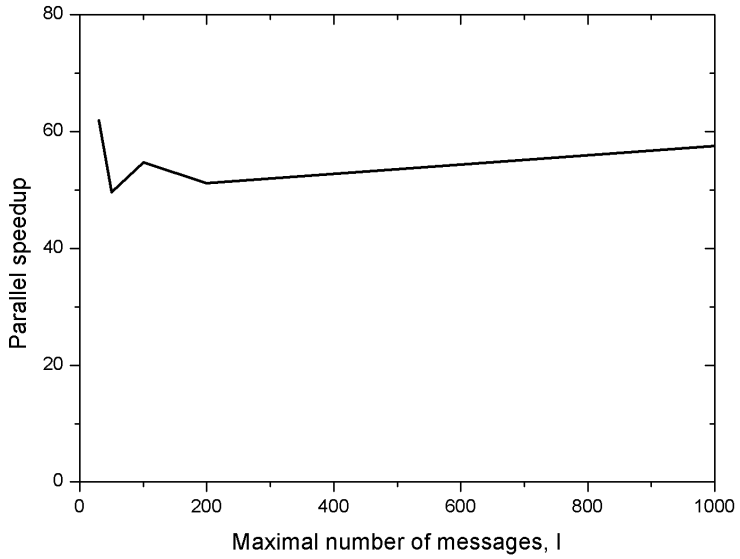

Fig. 5. The study of the influence of the message exchange intensity (**I**) on the scalability of the simulation.

To study the influence of parameter ***S*** the parameter ***I*** was fixed (***I*** = 10) and the calculations were carried out for the values ***S*** = 300, 500, 1000000, 10000000. The increase of the parameter S also leads to the linear increase of both network traffic and calculations of the random numbers, which shouldn't affect much the scalability. In the case of lower values of the parameter the size of the data is rather small and the speedup is determined more by the latency of the exchange network. This effect leads to the decrease of efficient for low values of ***S*** and much better efficiency for larger values of the parameter.

3.3 The influence of the interprocess communications

In the second variant of the test model the agents are divided into groups of size ***G***, the exchange of the messages is done only between agents inside the group. For that each agent has additional parameter the number of group N_G, which is defined at the beginning of the simulation to establish the uniform distribution of agents between groups. The use of such groups corresponds to the case of ideal decomposition of the graph of agents where most of the interprocess communications are removed.

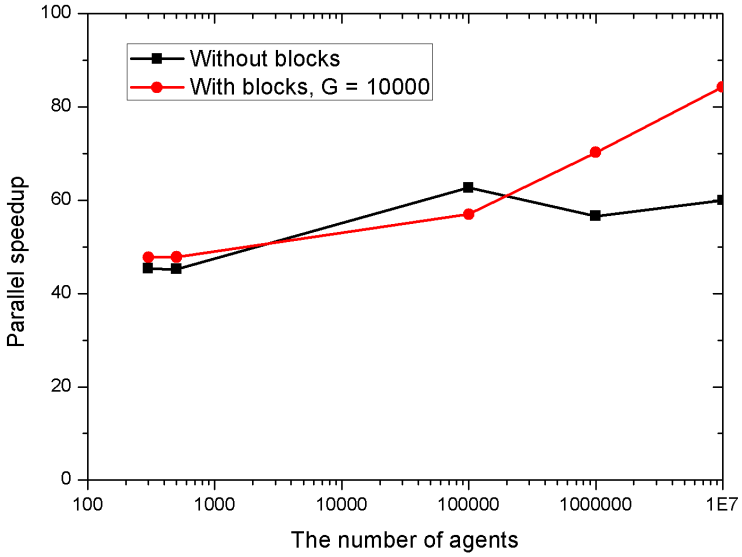

Fig. 6. The dependence of the parallel speedup on the number of agents participating in message exchange (**S**). The number of MPI processes **M** = 96.

In order to study this effect a set of simulations was performed and the dependence of the parallel speedup on the number of participating agents ***S*** was plotted. The results of these simulations are plotted in Fig. 6. The plot shows that the influence of the presence of blocks is much higher for high values of **S** when the number of agents participating in message exchange is high.

4. CONCLUSION

In this work a new framework for parallel calculations of agent-based models was presented and tested. The framework links the use of the high-level C# programming language and high-performance platform for messages exchange written using native C++ library and native MPI library of supercomputer. The provided results of the test simulations show good scalability of the program across multiple computational nodes.

ACKNOWLEDGEMENTS

This work was supported by the Russian Science Foundation (grant # 14-18-01968).

REFERENCES

[1] Collier N. (2013, August). *Repast HPC Manual.* [Online] Available: http://repast.sourceforge.net
[2] Boost C++ Libraries (2017) [Online]. Available: http://boost.org
[3] CyberGIS (2017) [Online]. Available: http://cybergis.cigi.uiuc.edu
[4] Mono (2017) [Online]. Available: http://www.mono-project.com
[5] Axum_(programming_language) (2017), [Online]. Available: https://en.wikipedia.org/wiki/Axum_(programming_language)
[6] Pandora: An HPC Agent-Based Modelling framework (2017), [Online]. Available: https://www.bsc.es/research-and-development/software-and-apps/software-list/pandora-hpc-agent-based-modelling-framework
[7] Karypis, G. & Kumar, V. (1995). *METIS-Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*. Technical Report.

[8] Makarov V.L., Bakhtizin A.R., Vasenin V.A., Roganov, V.A. & Trifonov I.A. (2011). Capacities of a supercomputer system for work with agent-based models, Programnaya Injeneriya [Software Engineering], 3, 2-14 [in Russian]

[9] Makarov, V.L., Bakhtizin, A.R., Sushko, E.D. et al. (2016) Supercomputer technologies in social sciences: Agent-oriented demographic models, *Her. Russ. Acad. Sci*, **86** (3), 248-257. doi:10.1134/S1019331616030047

[10] Scheutz, M., Connaughton, R., Dingler, A., & Schermerhorn, P. (2006). SWAGES - An Extendable Distributed Experimentation System for Large-Scale Agent-Based Alife Simulations. In *Proc. of Artificial Life X*, 412-419.

[11] Tang W. & Wang S. (2009). HPABM: A Hierarchical Parallel Simulation Framework for Spatially-explicit Agent-based Models, *Transactions in GIS* **13**(3)315-333.

