Analysis of Means of Simulation Modeling of Parallel Algorithms

Weins D.V. [1][0000-0003-3909-5249]✉, Glinskiy B.M.[2][0000-0002-0119-6370] and Chernykh I.G. [3][0000-0001-9564-1553]

[1] The Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia
vins@sscc.ru
[2] The Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia
gbm@opg.sscc.ru
[2] The Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia
chernykh@parbz.sscc.ru

Abstract. At the ICMMG, an integral approach to creating algorithms and software for exaflop computers is being developed. Within the framework of this approach, the study touches upon the scalability of parallel algorithms by using the method of simulation modeling with the help of an AGNES modeling system. Based on a JADE agent platform, AGNES has a number of essential shortcomings in the modeling of hundreds of thousands and millions of independent computing cores, which is why it is necessary to find an alternative tool for simulation modeling.

Various instruments of agent and actor modeling were studied in the application to modeling of millions of computing cores, such as QP/C++, CAF, SObjectizer, Erlang, and Akka. As a result, on the basis of ease of implementation, scalability, and fault tolerance, the Erlang functional programming language was chosen, which originally was developed to create telephony programs. Today Erlang is meant for developing distribution computing systems and includes means for generating parallel lightweight processes and their interaction through exchange of asynchronous messages in accordance with an actor model.

Testing the performance of this tool in the implementation of parallel algorithms on future exaflop supercomputers is carried out by investigating the scalability of the statistical simulation algorithm by the Monte Carlo methods on a million computing cores. The results obtained in this paper are compared with the results obtained earlier by using AGNES.

Keywords: Simulation Modeling, Actor Model, Scalability, Erlang

1 Introduction

According to the calculations given by D. Dongarra, the performance of supercomputers in exaflops will be reached by 2018-2020. Supercomputers will be able to serve 1 billion computing flows simultaneously. The number of cores will reach 100 million. The creation of exaflop supercomputers will require the development of parallel algorithms that can use tens and hundreds of millions of computing cores. The authors of the article develop an integral approach to developing algorithms and software for modern and future supercomputers. The approach is based on the technique for the development of algorithms and software for supercomputers of peta and exaflop levels, containing three related stages. The first stage is determined by co-design, which is understood as adapting the computational algorithm and mathematical method to a supercomputer architecture at all stages of the problem solution. The second stage is the development of preventive algorithms and software for the most promising supercomputers on the basis of simulation modeling for a given supercomputer architecture. The third stage is associated with estimating the energy efficiency of the algorithm for various implementations on this architecture or on various architectures [1].

This approach was tested on computationally complex problems of astrophysics, plasma physics, geophysics, and stochastic problems. The concept of co-design in the context of mathematical modeling of physical processes is understood as the construction of a physico-mathematical model of a phenomenon, numerical method, and parallel algorithm with its software implementation, which effectively uses the supercomputer architecture [2, 3].

An important component of the integral approach is simulation modeling, which allows investigating the scalability of a parallel algorithm on a given supercomputer architecture, determining the optimal number of computational cores to implement computations, and revealing bottlenecks in its execution.

The problem of modeling of scalable algorithms is not new - many groups of researchers in the world are engaged in it. Among foreign studies, we note the ones carried out in the US (University of Urbana-Champagne, Illinois). They are mainly engaged in estimating the performance of algorithms implemented with the use of MPI [4]. One of the main projects of this team is the BigSim project. The project is aimed at creating an imitation environment that allows the development, testing, and adjustment through the modeling of future generations of computers, while allowing for computer developers to improve their design solutions with a special set of applications. Among domestic studies, we note the ones conducted at the Ivannikov Institute for System Programming of the Russian Academy of Sciences (Moscow). This team developed a parallel program model that can be effectively interpreted on an instrumental computer, enabling fairly accurate prediction of the time of real execution of a parallel program on a given parallel computing system. The model is designed for parallel programs with explicit messaging, written in Java language with the use of the MPI library, and is included in the ParJava environment.

It is worth noting that both of the projects considered do not take into account (at least explicitly) the issues of fault tolerance in the execution of large programs, while the

use of tens of millions of computing cores at the same time is extremely urgent. The ParJava project, on the one hand, allows solving a wide range of problems for estimating the performance of parallel programs on promising computing systems, but, on the other hand, is tied to a specific programming language, which significantly reduces its capabilities.
The implementation of simulation modeling made it possible to investigate the scalability of algorithms for solving problems in astrophysics, plasma physics, geophysics, and problems using the Monte Carlo methods [1, 5]. The modeling was carried out on the basis of an AGNES multiagent system [6], with which it was possible to trace the behavior of algorithms up to several million computing cores. However, a further increase in the number of simulated computing cores proved to be difficult due to the limitations typical of the JADE platform, on the basis of which the AGNES system was built. Therefore, it became necessary to analyze other methods and tools, particularly based on an actor model. An actor model is a special technique for implementing agent modeling systems to reduce the overheads of agent communication [7].

2 Limitations of the AGNES Modeling System and Ways to Overcome Them.

In the study of the possibility to scale different algorithms to a large number of computational cores, the task of simulating the execution and communication of hundreds of thousands and millions of computational threads arose. The perfect approach for this purpose is an agent-based approach to modeling systems containing autonomous and interacting intellectual agents [8]. As a tool, the AGNES modeling system [7] was used, based on the JADE multiagent modeling platform [9]. This system had several advantages and disadvantages.

Using the JADE platform in the separation of the agent functionality into a set of behaviors makes it easy to parallelize the execution of independent behaviors. By default, the agent manager of each agent within the platform has its own flow, and all behaviors of the agent are carried out within this flow. However, the behavioral infrastructure itself introduces additional computational costs, especially in a single-flow performance. Also, significant overheads are introduced by the messaging system. It is extremely flexible and allows for standard means to transfer complex types of data, but all this at the expense of performance.

The JADE platform also provides for the partitioning of agents within the platform into containers that can reside on different hosts. If it is possible to minimize the exchange of messages between containers, then an increase in the number of hosts enhances the performance of such a system. However, in modeling systems where "all agents are connected to each other”, the system runs slower and slower.

In general, practice shows that the simultaneous launch of more than 1,000-2,000 simple interacting agents on a single average computing node causes a noticeable decrease in performance.

To overcome these shortcomings, various methods and techniques are used to run simulation models of algorithm behavior. The possibility of simulation of a large number of computational threads on a node requires that one agent simulates the behavior of a group of similar computational threads, each of which is an independent behavior of this agent and is executed in parallel. To minimize the exchange of messages between agents, the messages from different behaviors within the agent are formed into an array and forwarded to another agent as a single complex message. This allowed us to study the feasibility of scaling various algorithms into millions of computing cores.

Each algorithm under investigation has some special features. The special feature of algorithms for solving distributed statistical modeling problems using the Monte Carlo methods is the necessity for modeling an extremely large number of independent implementations. A feature of parallel grid methods in solving hyperbolic equations is the possibility of geometric decomposition of the computational domain and subsequent exchange of boundary values only between neighboring computing nodes. Below are the results of the study of the scalability of algorithms for solving the problems of astrophysics, plasma physics, and seismics and the problems using the Monte Carlo methods by means of the AGNES agent-based system (Fig. 1).

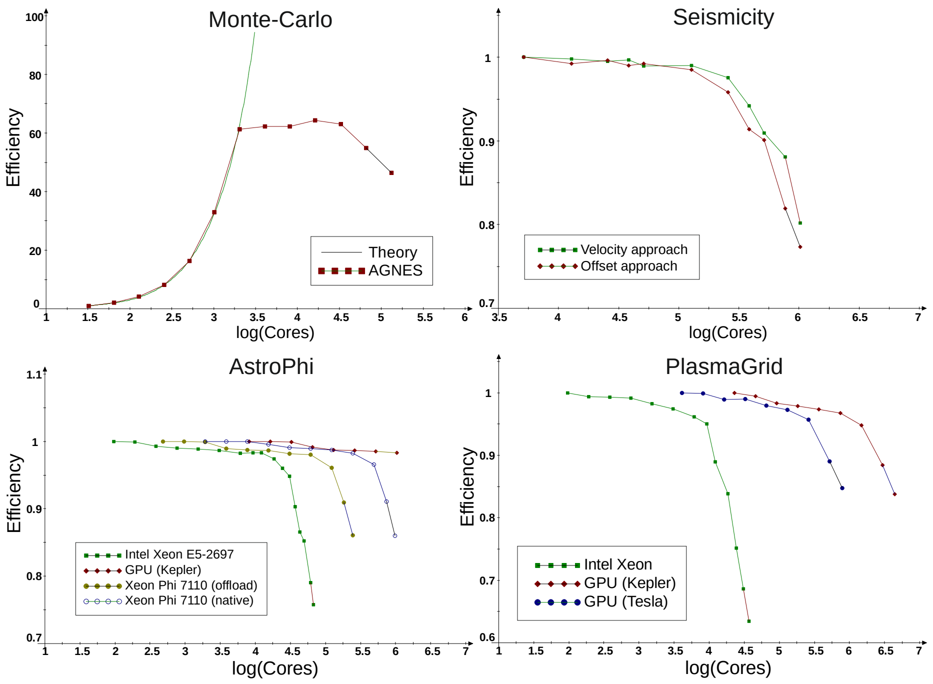

Fig. 1. Study of the scalability of different algorithms by means of the AGNES agent-based system

The upper section of the figure illustrates the results of the study of scalability of algorithms for solving the problem of direct statistical modeling by the Monte Carlo method and the seismic task in solving the problem in terms of displacement and stress velocities and in terms of displacements. Calculations were carried out on an

NKS-30T cluster with graphic accelerators [10]. The bottom section of the figure shows the results of the study of scalability of the astrophysical code and the physics code of plasma [11]. In this case, the scaling of algorithms on clusters with MPP architecture and using GPU and Phi is under study. It can be seen from the figure that about 1 million computing cores can be effectively used to solve these problems, but the efficiency drops sharply with a greater number of cores. A similar situation is observed in the solution of stochastic problems, where the computations are independent and there are practically no exchanges [5]. For parallel grid methods, a decrease in efficiency with rising number of computing cores is associated with an avalanche-like increase in message exchanges between the cores. For distributed statistical modeling, this decrease is due to the fact that the assembler node does not manage to process a large number of messages with intermediate data. The question arises: is this behavior of algorithms is primarily due to the calculation scheme or is it a feature of the chosen simulation modeling system?

In this regard, the task is to find an alternative modeling technique that helps eliminating the shortcomings of the JADE platform, as well as tools for working with it.

3 Analysis of Means of Simulation Modeling of Parallel Algorithms

3.1 Actor Model

An actor model is a mathematical model of parallel computations that treats the concept of "actor" as a universal primitive of parallel numerical calculation: in response to received messages, the actor can make local decisions, create new actors, send messages, and determine how to respond to following messages. It is not assumed that there is a certain sequence of the above-described actions, and all of them can be performed in parallel. The model was created in 1973, and it was used as a basis for understanding the calculus of processes and as a theoretical basis for a number of practical implementations of parallel systems [12].

The actor model is characterized by the inherent parallelism of calculations within one actor and between actors, dynamic creation of actors, inclusion of actors' addresses in messages, and also interaction only through direct asynchronous messaging without any restrictions on the order of arrival of messages.

The actor model has some characteristic distinctive features:

1. Unlimited indeterminism. There is no global state in the actor model.
2. Messages in the actor model are not necessarily buffered. This is its difference from the rest of the approaches to a model of simultaneous computations. In addition, messages in the actor model are simply sent, and there is no requirement for a synchronous handshake with the recipient.
3. Creating actors and inclusion of addresses of participants into messages means that the actor model has potentially variable topology in their relationships with each other. According to the communication model, the message does not have any mandatory fields, they can all be empty. However, if the sender of the message

wishes the recipient to have access to addresses that the sender does not have yet, the address should be sent in the message.

4. Unlike other approaches, based on the combination of sequential processes, the actor model was developed as a simultaneous model in its essence. As written in the theory of actor models, the sequence therein is a special case arising from simultaneous calculations.
5. The main innovation of the actor model is the introduction of the concept of behavior, defined as a mathematical function expressing the actor's actions when it processes messages, including determining a new behavior for processing the next message arrived. The behavior ensures the functioning of the mathematical model of parallelism and also frees the actor model from implementation details..

These and many other ideas introduced in the actor model are also used now in agent modeling systems. The actor model, in particular, is used in agent systems to minimize overheads in agent communication [7]. The key difference from agent modeling is that the system agent imposes additional restrictions on actors, usually requiring that they use commitments and goals. It is due to the advantages listed above that the use of the actor model to study the scalability of parallel algorithms seems very promising. Note that there are many papers on presentation and implementation of algorithms on the basis of the Actor model, but it has not yet been applied as a tool for investigating the scalability of parallel algorithms.

3.2 QP/C++, CAF and Sobjectizer

Let us consider tools and frameworks that allow implementing the concept of the actor model in C++. As C++ is a widely known native language, which makes it possible to switch easily from the lowest level close to hardware to a very high level, such as OOP and general programming. At the same time, this language is provided with a wide range of tools, books, and documentation.

Strictly speaking, there are not too many ready-made implementations of the actor model. Among popular and actively developing frameworks for C++, C++ Actor Framework (CAF) [13], QP/C++ [14], and SObjectizer (SO) [15] can be distinguished.

OpenSource project under the BSD license is C++ Actor Framework. Also known as CAF and libcppa, it is the most famous implementation of the actor model for C++. This is an easy-to-use framework with a specific syntax that maximally fully implements the principles of actor models. The scope of CAF is limited to the Linux platform. The CAF developers themselves describe it as the most productive framework. It also offers ready-made tools for creating distributed applications.

QP/C++ is a software product under a double license, designed to develop embedded software, including real-time systems and systems that can work directly on hardware implementation.

Actors in QP/C++ are called active objects and represent hierarchical finite-state machines. The code of actors can be typed in the form of ordinary C++ classes, and an actor can be drawn in a special tool for visual modeling and its code is generated automatically. Active objects in QP/C++ work on a context that QP allocates to them. Depending on the environment, active objects can work each on their own thread or they can share a common working context.

SObjectizer is an OpenSource project under the BSD license, has been developed since 2002, and is based on the ideas created and tested when building a small object-oriented SCADA system. SObjectizer is created specifically to simplify the development of multi-thread software in C++. Therefore, in SObjectizer, much attention is paid to compatibility. What SObjectizer does not currently provide is ready-made tools for constructing distributed applications.

Actors in SObjectizer are called agents. As in QP/C++, agents in SObjectizer are, as a rule, instances of individual C++ classes. Just like in QP / C++, agents are hierarchical finite-state machines. Just like in QP/C++, the working context for agents is provided by the framework. For this purpose, SObjectizer includes a dispatcher, which is a special entity that performs dispatching of agent events. SObjectizer strongly differs from the above-mentioned projects in that SObjectizer sends messages not directly to the recipient agents, but in mboxes (mailboxes). From an mbox, a message is delivered to those agents who are subscribed to it.

3.3 Erlang and Akka

When it comes to the actor model, one cannot help but mention Erlang, and, speaking of Erlang, one cannot help but talk about the actor model. Erlang is a functional programming language with strong dynamic typing, designed to create distributed computing systems [17]. It is developed and supported by Ericsson for writing programs for telephony. Erlang inherited its syntax and some concepts from the Prolog logical programming language. This language includes means of generating parallel lightweight processes and their interaction through the exchange of asynchronous messages in accordance with the actor model.

Erlang was purposefully designed for use in distributed, fault-tolerant, parallel real-time systems, for which, in addition to the language itself, there is a standard library of modules and library templates, known as an OTP framework. The program on Erlang is translated into bytecode, executed by virtual machines located on different nodes of a distributed computer network.

The popularity of Erlang began to grow due to the expansion of its application area (telecommunication systems) to highly loaded parallel distributed systems serving millions of WWW users, such as chats, content management systems, web servers, and distributed databases requiring scaling. A great number of products are developed on Erlang, and many companies use Erlang as a key tool.

Also speaking of the actor model, one cannot fail to mention the Akka framework for the Scala and Java languages [16]. Actors in Akka are instances of separate classes executed on JVM virtual machines located on different nodes of a distributed computer network. The actor in Akka consists of several interacting components: the actor's dispatcher and the actor itself. The dispatcher is responsible for placing messages in the queue leading to the mailbox of the actor and also orders this box to remove one or more messages from the queue (but only one at a time) and transfer them to the actor for processing. Last but not least: the actor is usually the only API that needs to be implemented, and it encapsulates the state and behavior. Akka does not allow direct access to the actor, so it guarantees that the only way to interact with the actor is through asynchronous messages.

Akka is widely used in the field of Web and online services (for example, Twitter and LinkedIn).

There are several key factors that explain the popularity of Erlang and Akka among modern developers:

- Simplicity of development. Using asynchronous messaging greatly simplifies the work when one has to deal with concurrent computing;
- Scaling. The actor model allows creating a huge number of actors, each of which is responsible for its particular task. The shared nothing principle and asynchronous messaging allow building distributed applications that are scaled horizontally as much as needed;
- Fault tolerance. A failure of one actor can be caught by other actors, who take appropriate actions to restore the situation. Also, if some lightweight process within Erlang VM performs division by zero, then Erlang VM simply closes one of these processes, and this will not affect the performance of other processes. However, if there is division by zero in one of the threads of a multithread application on C++, then the entire application crashes.

4 Experimental Study

To test the performance of the tools under consideration, a model of execution of the algorithm for solving the problem of distributed statistical modeling using the Monte Carlo methods is implemented on each of them. Unfortunately, due to problems with licensing, QP/C++ does not participate in this experiment. Such parameters as the amount of code for the description of actors, acceleration from parallelization, and the total simulation time are explored.

When simulating computations with the help of actors, threads-collectors and threads-calculators are simulated. Threads-calculators at each iteration of the calculation cycle determine the independent implementation and transmit the result of intermediate averaging to threads-collectors. The results obtained are averaged by thread-collectors. The calculation is carried out until the required level of the permissible relative statistical error is reached. Calculations on different threads-calculators are performed in an asynchronous mode. The calculation results are sent and received also in an asynchronous mode.

A comparison of the number of symbols needed to describe the actors/agents in the tools under consideration is shown in Figure 2. As can be seen from the figure, the most concise description is obtained in Erlang and Akka. This can be explained by the fact that very many mechanisms for memory distribution and allocation, balancing, and messaging are already implemented within these tools.

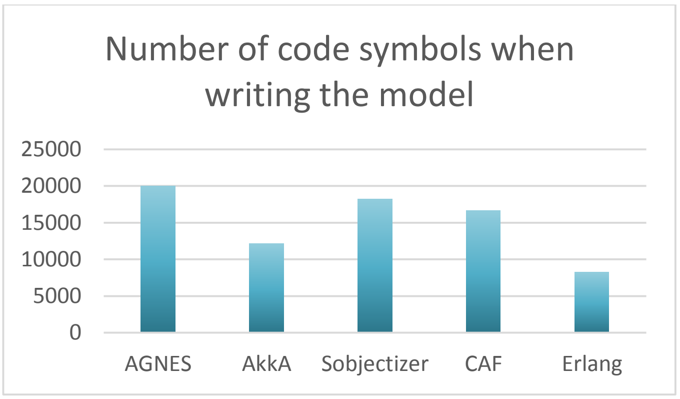

Fig. 2. Comparing the amount of programming code in the description of actors/agents in different tools

As is known, the theoretical acceleration for paralleling for statistical modeling methods is practically "ideal", i.e., the calculation time decreases in proportion to an increase in the computing nodes. This criterion is used to study the effectiveness of these computational algorithms.

We consider the resulting data on the scalability of the algorithm (Fig.3) and data on the execution time of the model (Fig. 4) for various tools. For clarity, the charts show the theoretical acceleration for this algorithm and the results obtained earlier with the help of AGNES.

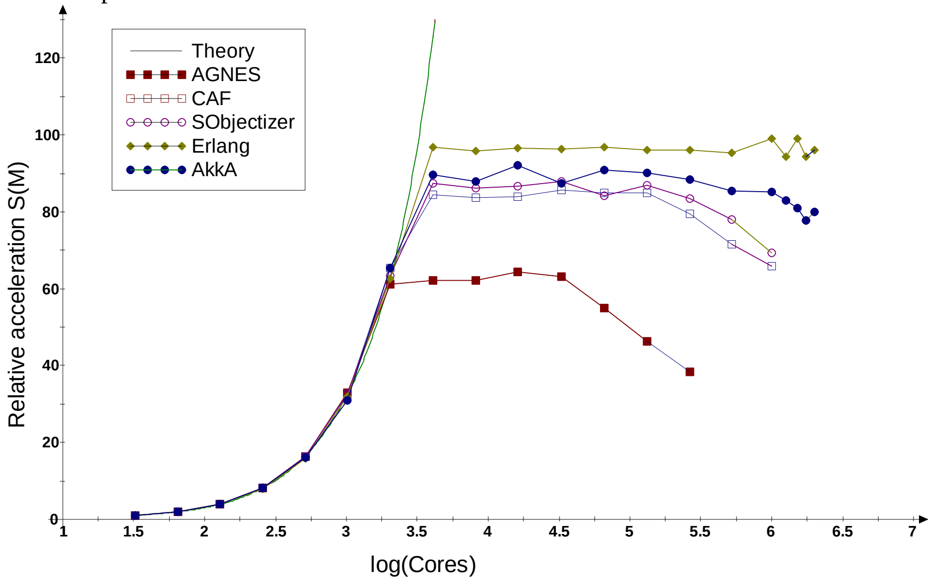

Fig. 3. Study of the scalability of the algorithm for solving the problem of direct statistical modeling using the Monte Carlo methods by means of various tools.

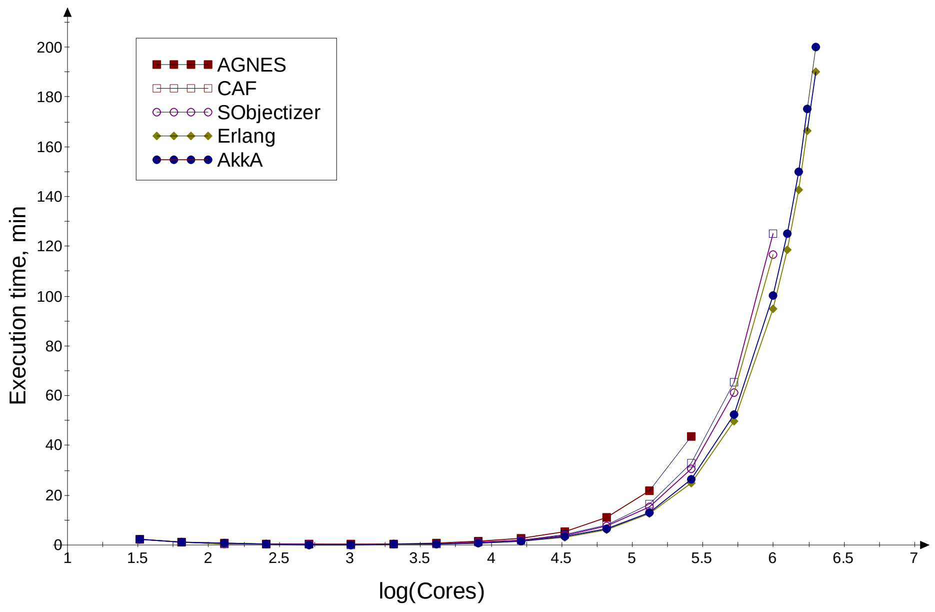

Fig. 4. Time for calculating the algorithm execution model on various tools.

Previously, similar behavior of the investigated algorithm (Fig.3) was explained this way: as soon as the time for sending and processing messages from all calculators after one iteration of calculations begins to exceed the time for this iteration, the acceleration gain slows down and the scaling is no longer observed on a certain number of the effect calculators (the line becomes parallel to the X axis). There are two possible ways to extend the effect of acceleration from the increasing number of cores. The first one is via hardware: reduction of the transmission and processing time of the message. The second one is algorithmic: construction of a hierarchical model for collecting intermediate results [5]. As can be seen from Fig. 3, the exchange of short, asynchronous messages in the actor model allowed us to obtain more accurate data on the scalability of the algorithm. The messaging model in AGNES introduced an additional delay in communications. Some difference between the tools implementing the actor model can be explained by various implementations of the messaging process. The fact that, in some implementations, there is a slowdown on the right part of the chart can be explained by the avalanche-like increase in incoming messages to the thread-collector that the message queue either cannot handle correctly or cannot handle at all. It is worth noting that, the desired (one million or more) number of actors failed to run on some tools.

It is possible to evaluate the performance of the tools implementing the actor model on the basis of Fig. 4. It is not for nothing that the Akka framework and the Erlang programming language are the recognized leaders in implementing the actor model because they showed the best performance. The CAF and SObjectizer tools are a little behind them.

5 Conclusion

As experiments have shown, the use of the actor model to simulate the execution of parallel algorithms allows one to get rid of a number of significant disadvantages of the JADE platform. The process of modeling of millions of computing cores is significantly lesser affected due to the use of simple and asynchronous messages. And the idea of using multiple lightweight (memory-wise) actors makes it possible to achieve the simulation of execution of required millions of computing cores.

Among the tools, libraries, and languages implementing the actor model, a special place is occupied by the Erlang programming language. In it, the concept of the actor model is most fully implemented. This was confirmed in the course of the experimental study of the performance of these tools.

The experimental research has shown that, due to such parameters as ease of implementation, scaling, and fault tolerance, the Erlang functional programming language is most suitable for investigating the scalability of algorithms using simulation modeling methods.

Acknowledgments

This work was supported by the Russian Foundation for Basic Research (Grants No. 16-07-00434, 18-37-00279, and 18-07-00757).

The Siberian Supercomputer Center of the Siberian Branch of the Russian Academy of Sciences (SB RAS) is gratefully acknowledged for providing supercomputer facilities.

References

1. Glinskiy B.M., Kulikov I.M., Chernykh I.G., Snytnikov A.V., Sapetina A.F., Weins D.V.: The integrated approach to solving large-size physical problems on supercomputers. RuSCDays 2017. CCIS. 2017. Vol. 793, pp. 278–289. DOI: 10.1007/978-3-319-71255-0_22.
2. Glinskiy B., Kulikov I., Snytnikov A., Romanenko A., Chernykh I.,Vshivkov V. Co-design of Parallel Numerical Methods for Plasma Physics and Astrophysics // Supercomputing Frontiers and Innovations. – 2014. – V. 1, I. 3. – P. 88-98.
3. Glinsky, B., Kulikov, I., Chernykh, I., Weins, D., Snytnikov, A., Nenashev, V., Andreev, A., Egunov, V., Kharkov, E. The Co-design of Astrophysical Code for Massively Parallel Supercomputers Proceedings of 16th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP) Lecture Notes in Computer Science, 2016, T: 10049 P: 342-353.

4. T. Hoefler, T. Schneider and A. Lumsdaine: "LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model
5. Glinsky B., Rodionov A., Marchenko M., Podkorytov D., Weins D. Scaling the Distributed Stochastic Simulation to Exaflop Supercomputers // Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications (HPCC-2012). 2012. P. 1131–1136.
6. Podkorytov, D., Rodionov, A., Choo, H.: Agent-based Simulation System AGNES for Networks Modeling: Review and Researching. Proc. of the 6th Int. Conference on Ubiquitous Information Management and Communication (ACM ICUIMC 2012), ISBN 978-1-4503-1172-4, pp. 115. ACM (2012) DOI: 10.1145/2184751.2184883.
7. Mueong J., Gul A. Agent framework services to reduce agent communication overhead in large-scale agent-based simulations // Simulation Modelling Practice and Theory. Vol. 14, i. 6, pp. 679-694.
8. Oren T. On the Synergy of Simulation and Agents: An Innovation Paradigm Perspective / T. Oren, L. Yilmaz // International J. of Intelligent Control and Systems. – 2009. – V. 14, №1. – P. 4 – 19.
9. JADE Homepage, http://jade.tilab.com/, last accessed 2018/04/10.
10. Boris M. Glinskiy, Anna F. Sapetina, Valeriy N. Martynov, Dmitry V. Weins, Igor G. Chernykh The Hybrid-Cluster Multilevel Approach to Solving the Elastic Wave Propagation Problem // PCT 2017, CCIS. 2017. Vol. 753, pp. 261–274. DOI: 10.1007/978-3-319-67035-5_19.
11. Boris M. Glinskiy, Igor M. Kulikov, Igor G. Chernykh, Alexey V. Snytnikov, Anna F. Sapetina, Dmitry V. Weins The Integrated Approach to Solving Large-Size Physical Problems on Supercomputers // Supercomputing. RuSCDays 2017. CCIS. 2017. Vol. 793, pp. 278–289. DOI: 10.1007/978-3-319-71255-0_22.
12. Карл Хьюитт, Питер Бишоп, Ричард Штайгер: Универсальный модульный формализм акторов для искусственного интеллекта. IJCAI, 1973
13. CAF - C++ Actor Framework, http://www.actor-framework.org/, last accessed 2018/04/10.
14. QP/C++: About QP/C++, http://www.state-machine.com/qpcpp/, last accessed 2018/04/10.
15. SObjectizer /Wiki /Home, https://sourceforge.net/p/sobjectizer/wiki/Home/, last accessed 2018/04/10.
16. AkkA, https://akka.io/, last accessed 2018/04/10.
17. Erlang Programming Language, http://www.erlang.org/, last accessed 2018/04/10.
18. F. Cesarini, S. Thompson. Erlang Programming. — O'Reilly Media, Inc., 2009. — 498 p. — ISBN 978-0-596-51818-9.