ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2021, Vol. 42, No. 11, pp. 2571–2579. © Pleiades Publishing, Ltd., 2021.

Scaling of Supercomputer Calculations on Unstructured Surface Computational Meshes

B. M. Shabanov[1*], A. A. Rybakov[1], S. S. Shumilin[1***], and M. Yu. Vorobyov[1****]**

(Submitted by A. M. Elizarov)

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

Received May 4, 2021; revised May 19, 2021; accepted May 29, 2021

Abstract—When solving complex problems of numerical modeling, computational meshes, containing hundreds millions of cells are quite often. Modern tasks even cross the line of billion cells. Workstations are unable to cope with such volume of data and computation. To perform computations of this volume we need to use supercomputer clusters consisting of many computational nodes interconnected by a high-speed communication network. In this case, it is necessary to perform the decomposition of the computational mesh into separate domains in order to ensure its parallel processing on all nodes of the cluster. These domains are distributed among the computational nodes of the supercomputer and are processed independently of each other. To efficiently perform calculations and scale them to a large number of computational nodes, it is necessary to develop efficient algorithms for decomposition of computational meshes that generate many domains with imposed requirements. We consider an hierarchical decomposition algorithm with the choice of the optimal criterion for dividing mesh into domains. As such a mesh we study an unstructured surface mesh used to calculate the processes of interaction of a volumetric body with the environment. Using this decomposition algorithm, supercomputer calculations are performed on the computing resources of JSCC RAS in order to measure the practical indicators of scalability of highly loaded applications.

DOI: 10.1134/S1995080221110202

Keywords and phrases: *supercomputer, unstructured surface computational mesh, decomposition, domain, high-performance computing, scaling of calculations.*

1. INTRODUCTION

Modern supercomputer applications are widely used in various spheres of life [1, 2] and they are extremely demanding on computational resources. For large tasks it is not possible to execute them on a separate computer (one microprocessor or one server) in reasonable time. There is a need to use supercomputer clusters for calculations, consisting of many computational nodes. In order to perform a task on a supercomputer, it is necessary to divide its computational domain into separate subdomains and process them in parallel [3].

To improve the efficiency of supercomputer applications within a computing node, various methods of data preparation and parallelization of execution for systems with shared memory are used [4]. Low-level optimizations of program code, such as vectorization [5], which can significantly increase the speed of application execution can also be very efficient. Of course, at the boundaries of domain contact, it becomes necessary to synchronize calculations, which is achieved by data exchange (for example, using MPI [6]).

[*] E-mail: shabanov@jscc.com
[**] E-mail: rybakov.aax@gmail.com
[***] E-mail: noisd@yandex.ru
[****] E-mail: nordmike@jscc.com

Thus, the execution of supercomputer calculations consists of two alternating steps: parallel processing of cells of the computational domain and data exchange at the boundaries of domains contact. The efficiency of supercomputer applications execution essentially depends on the quality of the computational mesh decomposition and its distribution over different computational nodes.

This article studies the problem of surface unstructured computational mesh decomposition for distribution between the nodes of homogeneous supercomputer cluster to increase the efficiency of calculations scaling. By homogeneous we mean a cluster consisting of the same computational nodes.

Take a surface mesh consisting of S cells. Let the supercomputer consist of n computational nodes with the same characteristics. We will also assume that the speed of data exchange between any two computational nodes is the same for all nodes. Extension of the problem of distributing the computational load to a heterogeneous supercomputer is achieved by entering weights for nodes and for data exchange channels, as described in [7].

If we denote the speed of cells processing on one computational node as a^{-1}, then the execution time of one iteration of calculations on one computational node can be written as $T_1 = aS$. Now let the computational domain be divided into n domains each containing S_i cells ($i = 1, n$). Let us denote by L_{ij} the number of edges that form the border between the domains S_i and S_j. We will assume that each domain is processed on its own computational node. Thus, all domains are processed in parallel, and the processing time for all cells is determined by the processing time for the largest domain. In addition to processing all cells, after a calculation iteration, it is necessary to exchange data between all pairs of domains along their boundaries. Let the speed of data transfer between nodes be determined as b^{-1}, and all exchanges are performed in parallel. Then the execution time of all exchanges is determined by the time of data exchange across the longest boundary. Based on this, it is possible to determine the total execution time of one iteration of calculations when executing on n computational nodes:

$$T_n = a \max_{i=1,n} S_i + b \max_{i,j=1,n} L_{ij}. \tag{1}$$

The criterion for optimization of computational mesh's decomposition is the reduction of time for performing calculations, that is, the value of T_n. The calculation execution time directly depends on the size of the largest domain, however, we will consider not the absolute size of the domain, but its deviation from the theoretical optimal value. Obviously, in the ideal case, during decomposition, all domains should have the size $\frac{S}{n}$, and the relative deviation from the ideal size in percentage can be calculated by the formula

$$D = 100\% \left(\frac{n}{S} \max_{i=1,n} S_i - 1 \right). \tag{2}$$

The second important criterion for the quality of the performed decomposition is the length of the longest boundary between pairs of domains. In this case, the absolute characteristic can be used, since the length of the boundary is rather difficult to predict, and in general, it has no theoretical minimum. Depending on the geometry of the mesh under consideration, the length of the boundary between domains can theoretically be zero. We will use the following simple metric as decomposition method quality characteristic:

$$L = \max_{i,j=1,n} L_{ij}. \tag{3}$$

Despite the fact that, in our assumptions, all data exchanges between domains are executed simultaneously, the total volume of all exchanges significantly affects the data exchange rate, so this parameter must also be taken into account. Let us introduce it in the following form. The total number of edges in the computational mesh remains unchanged regardless of the decomposition algorithm and number of domains. We denote the total number of edges in the mesh by E. Among these edges there are border edges that have only one incident cell, their number is also constant and equal to E_B (from the word "border"). The rest of the edges have two incident cells. If both cells incident to a certain edge belong to the same domain, then we call such an edge an inner edge of this domain. We denote their number by E_{INN} (from the word "inner"). Otherwise the edge enters the boundary between two domains and we call it an interdomain edge. E_{INT} is their number. There can be no other types of edges. Thus, the relation $E = E_B + E_{INN} + E_{INT}$ is fulfilled. As a parameter for evaluating the quality

of decomposition, we will consider the fraction of interdomain edges in the total number of mesh edges, that is, the value

$$I = 100\% \left(\frac{E_{INT}}{E} \right). \tag{4}$$

To assess the quality of the computational mesh decomposition, all three described parameters should be taken into account: D is the deviation of the biggest domain size from the ideal value, I is the fraction of interdomain edges in the total number of edges of the computational mesh, and L is the length of the longest boundary between pairs of domains. The lower the values of these criteria, the better the decomposition is and the more efficient calculations can be expected when running on a real machine.

2. PARALLELIZATION OF CALCULATIONS ON AN UNSTRUCTURED COMPUTATIONAL MESH

Consider a parallelization scheme for calculations associated with modeling the interaction of a volumetric body with the environment. In this case, calculations are carried out on the surface of an interacting body described by an unstructured surface computational mesh (MSU, Mesh Surface Unstructured) with triangular cells. In the considered calculation scheme on the body surface, various physical processes are considered, including surface deformation, the flow of a liquid film, and ice accretion, which ultimately should lead to remeshing of the surface. The calculation scheme is thus divided into several independent functional modules (the scheme of interaction of the modules is shown in Fig 1).

The first module (`msu-split`) is designed to load the computational mesh, decompose it into several domains and prepare it for solver. The physical solver `solver` is intended only for high-load computations, it interacts with a supercomputer cluster and operates on an arbitrary number of computational nodes. This is the most resource-demanding part of the application, which takes more than 99% of the estimated time, so it is implemented using MPI, OpenMP and vectorization of the program code. After carrying out the necessary calculations, the `solver` module returns the accumulated data from different time points, which are fed into the `msu-merge` data fusion module. The last link in the chain of calculations is the `msu-remesh` module, which receives a computational mesh as input (in general case, a set of computational meshes from different points of time) and remeshes it.

3. UNSTRUCTURED COMPUTATIONAL MESH DECOMPOSITION

Article [8] describes a parallel algorithm for geometric decomposition of mesh data. During the operation of this algorithm, the current domain is sequentially halved using a plane section. According to the logic of this algorithm, the initial head domain h is halved into a pair of domains h_l (left) and h_r (right), each of which is further halved, and so on up to any number of domains equal to a power of two.

We propose to extend this algorithm by introducing arbitrary criteria for dividing the current domain into a pair of smaller domains. First, consider the scheme of simple bisection of a domain using an arbitrary criterion by which the division is performed (Fig. 2).

Assume a set of cells of domain h and some function fun for extracting a feature from a cell. The first step is to calculate a set of features for all cells (f is features array)

$$f = \{fun(h_i) \mid h_i \in h\}.$$

Then sort f.

In the sorted set of features, the median value should be chosen (b is blade denotion):

$$b = median(f).$$

This value will be used to split the domain into two smaller domains (h_l is left subdomain, h_r is right subdomain) using two simple filters

$$h_l = \{h_i \ : \ h_i \in h, \ fun(h_i) < b\}, \quad h_r = \{h_i \ : \ h_i \in h, \ fun(h_i) \geq b\}.$$

After splitting a domain into two smaller subdomains, we can calculate a parameter that reflects the efficiency of splitting. We propose to use the length of the boundary between the two newly formed

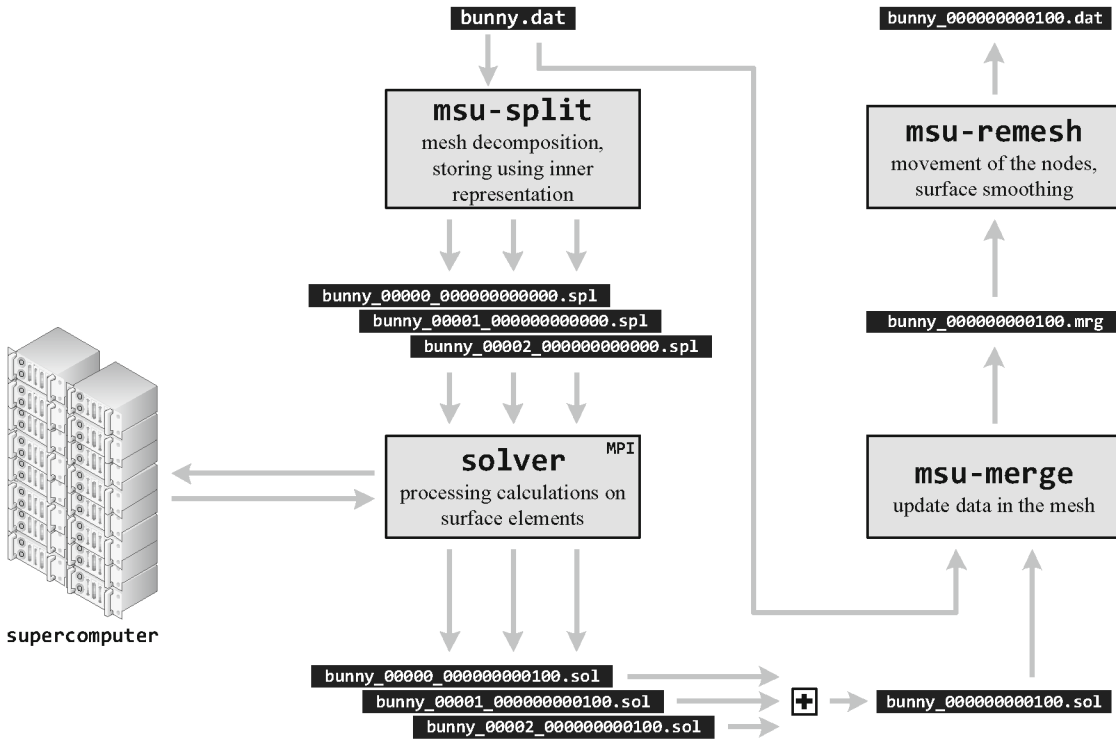

Fig. 1. Parallelization scheme for calculations on a supercomputer.

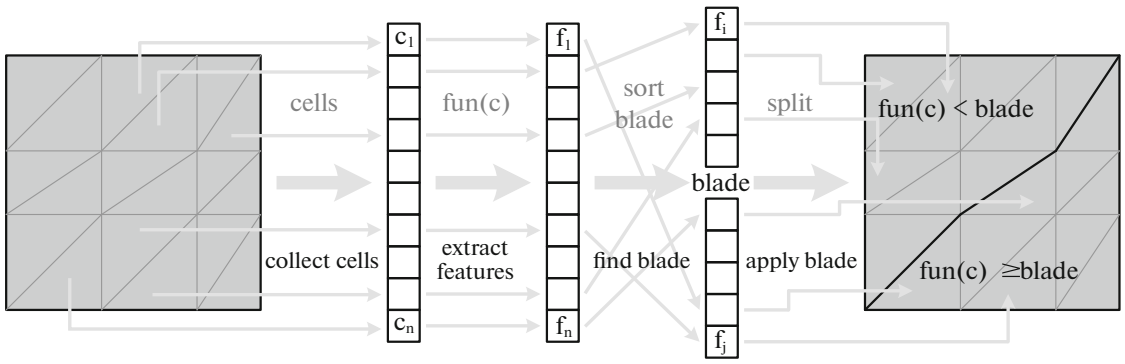

Fig. 2. A scheme of halving the domain according to the selected attribute.

domains as such a parameter. Thus, the splitting criterion depends on the function of calculating the attribute fun. In turn, this means that when performing a partition, it is not necessary to be limited to one function of calculating a feature. Instead, you can submit a list of functions, for each function calculate the parameter of quality of the partition and, as a result, stop at the function, which ultimately leads to the most efficient partition.

If we simply use the extraction of the three coordinates of the cells' centers as functions for calculating the feature of a cell, then we will obtain in its pure form an algorithm for the geometric decomposition of the mesh with the choice of longest coordinate for splitting. The results of applying this algorithm to several 3D models from [9] can be seen in Figs. 3 and 4.

By varying set of features for choosing those by which the domain can be partitioned, it is possible to perform geometric decomposition along the direction of any curve for which the cell projection is calculated. Decomposition using this method is not limited to geometric features only. Feature calculating functions can be used to analyze the physical data of cells, for example, to localize and isolate areas with increased pressure into separate domains.

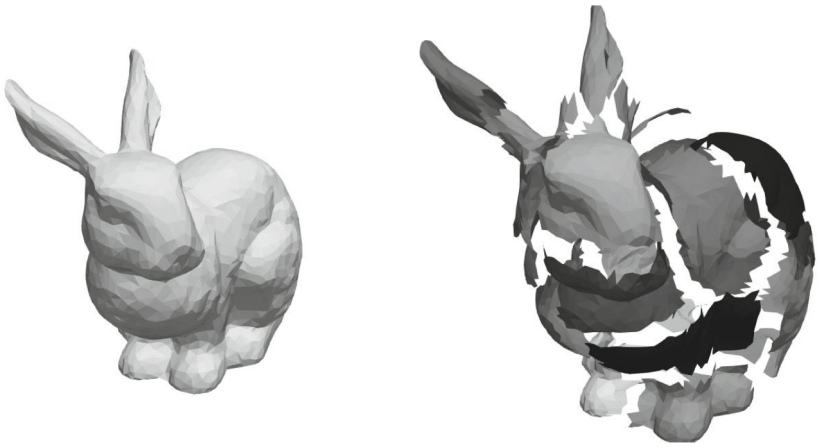

Fig. 3. An example of decomposition of a surface computational mesh using a hierarchical algorithm.

Fig. 4. Examples of surface computational mesh decomposition using a hierarchical algorithm.

Within the framework of this work, the described algorithm for decomposition of the surface unstructured computational mesh was applied to the surface meshes used to calculate icing of the aircraft surface [10, 11]. When calculating icing of an aircraft, the bulk of the computations relates to the processing of surface cells. Shared data between adjacent domains is collected on interdomain edges. The typical size of such meshes was about 10^5 cells. We considered both simply connected and not simply connected surfaces, as well as surfaces consisting of several zones isolated from each other.

It should be noted that the theoretical quality parameters D, L, I are quite low. The D parameter is practically zero, since at each step the domain is divided strictly in half. The deviations in the lengths of the boundaries between domains and the total length of the boundaries are also acceptable, despite the fact that the algorithm does not guarantee the formation of domains with minimal boundaries or even connected domains In the worst cases, domains of arbitrary shape and consisting of an arbitrary number of isolated parts of the surface may arise. Of course, a significant limitation of the algorithm is that it can be used to partition the surface only into the number of parts, which is a power of two, but the close-to-zero value of the computational load balancing quality index D allows in this case to neglect this drawback.

4. ORGANIZATION OF INTERPROCESS EXCHANGES

When performing calculations on a decomposed mesh, at each iteration of the computation, it is necessary to exchange data at each boundary between domains. In our case, when performing the decomposition of the computational mesh, the boundary between two domains is represented by an arbitrary set of interdomain edges. In this case, the boundary may be discontinuous, it may even consist of separate edges, therefore the order of interdomain edges in the sequence is not important when describing the boundary.

Figure 5 shows a scheme of organization of interprocess exchanges [12]. Two domains (we will conventionally call them left and right), processed in MPI processes numbered 0 and 1, are separated

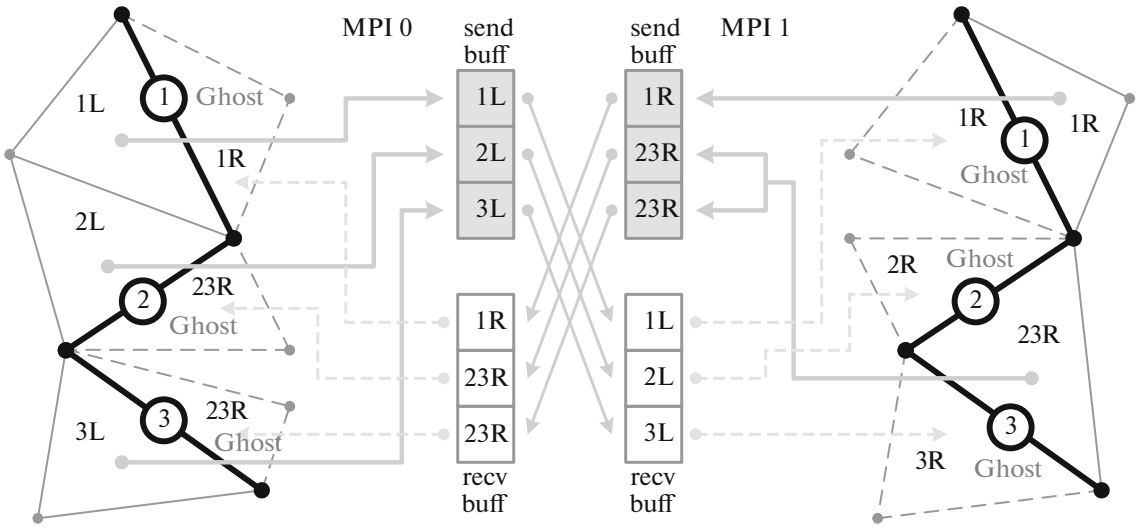

Fig. 5. Scheme of execution of MPI-exchanges across the border of two zones.

by a boundary consisting of three edges. In this case, there are three cells in the left domain that are adjacent to the considered boundary, in the right domain there are only two such cells (since cell 23R is adjacent to two edges of the boundary at once). To organize interprocess communications in each domain, ghost cells are created for each edge of the considered boundary, which are involved in calculating flows through the edges. At the same time, there is no need to recalculate physical values in ghost cells All data for ghost cells is obtained using MPI exchanges from real cells of a neighboring domain.

To transfer data from the real border cells of the domain to the ghost cells of the neighboring domain, buffers for sending and receiving data are organized in each of the two neighboring domains. The data exchange sequence is as follows (the diagram is shown in Fig. 5). First, data from real boundary cells is written to the corresponding send buffers, then asynchronous commands `MPI_Irecv` for receiving messages in the data receiving buffers are executed for all boundaries of the computational mesh. After that, commands for asynchronous sending of data `MPI_Isend` from the send buffers are also executed simultaneously for all boundaries of the computational mesh. Next, it is necessary to wait for the completion of all asynchronous data exchanges using the `MPI_Waitall` function. The last step, completing the exchange of data between neighboring domains, is the transfer of the received physical values from the data receiving buffers to the corresponding ghost cells.

Note that some data duplication may occur when using ghost cells. For example, in the presented diagram, cell 23R from the right domain corresponds at once to two ghost cells in the left domain. These cells contain the same data. This duplication of information is acceptable, since the data of the ghost cells is used only for reading to perform the calculation of flows across the domain boundary, therefore, in this case, there is no need to perform any synchronization of the same ghost cells.

5. EFFICIENCY OF SCALING SUPERCOMPUTER CALCULATIONS

To measure the scalability of computations on an unstructured surface computational mesh, a test surface of a streamlined three-dimensional body was used, containing about $2 \cdot 10^5$ nodes and $4 \cdot 10^5$ cells. In the cells, calculations were performed related to modeling the flow of a liquid film, solving the heat balance equations on the surface, as well as remeshing and smoothing the surface. To decompose the surface mesh, we used a simple hierarchical algorithm for halving domains, in which three coordinates of the center were taken as the features of the cells. At the same time, as a result, the criterion for choosing a specific coordinate for dividing a domain was to minimize the length of the boundary between two domains (this approach allows dividing a domain along the longest direction).

Homogeneous segments of the computing system of Joint Supercomputer Center of the Russian Academy of Sciences were used to measure the computational scalability [13]. In total, the calculations were carried out on four computational segments, the characteristics of the nodes of which are given in

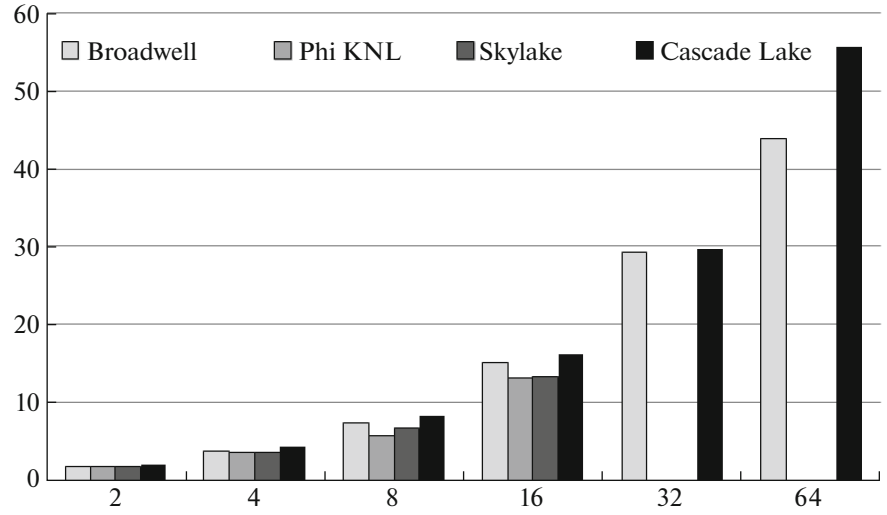

Fig. 6. Acceleration of computations on supercomputers at JSCC RAS when increasing the number of nodes.

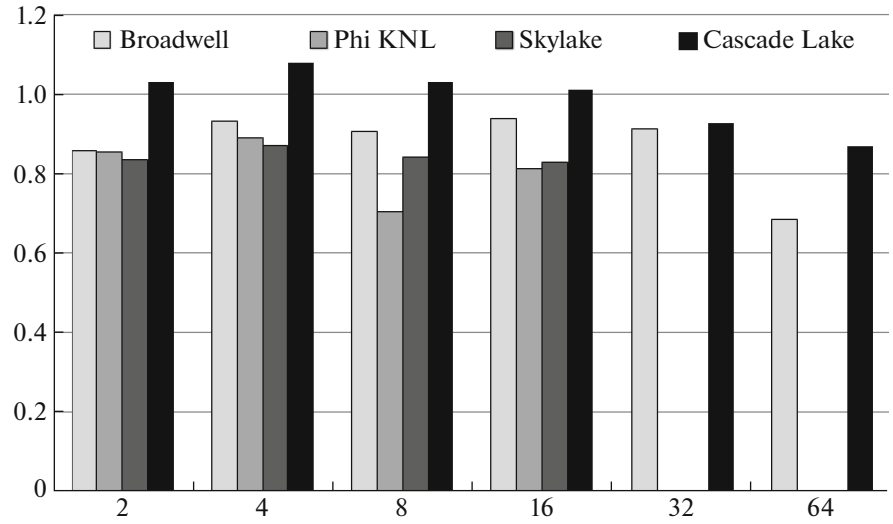

Fig. 7. Efficiency of scaling computations on supercomputers of JSCC RAS when increasing the number of nodes.

the Table 1. In this table, it can be noted that all Intel [14] microprocessors except Xeon Broadwell support the AVX-512 instruction set, which allows the use of special 512-bit vector registers for efficient code vectorization [15]. We should also highlight the computing nodes based on the Xeon Phi KNL microprocessor [16]. These microprocessors are distinguished by a huge number of computing cores, each of which is capable of executing up to 4 threads, which allows efficient parallelization of computational applications up to 288 threads on a single microprocessor.

The main goal of the launches was to measure the strong scalability of computations with full parallelization within computational nodes using OpenMP. That is, for all launches, the same surface was used (which was split over the required number of computational nodes), and all the threads available inside the computational nodes were also involved in the calculations.

When carrying out the calculations, the measurements were carried out independently for each computer system separately. Let us give a description of the quantities measured in the calculation process for one specific computing system. The time of the task execution on one computational node was used as the reference time: $t(1)$. We also measured the execution time of tasks for the number of computational nodes equal to a power of two (2, 4, 8, 16, 32, 64). In this case, the acceleration on the

Table 1. The configurations of the segments of the MVS-10P OP supercomputer, on which the computation scaling measurements were taken

Family of Intel microprocessors	Number of processors/cores/ threads per node	Frequency of microprocessor	RAM per node	Supports AVX-512
Xeon Broadwell	2/32/64	2.6 GHz	128 GB	no
Xeon Phi KNL	1/72/288	1.5 GHz	96 GB	yes
Xeon Skylake	2/36/72	3.0 GHz	192 GB	yes
Xeon Cascade Lake	2/48/96	3.0 GHz	192 GB	yes

number of nodes equal to i was considered the value of the value $s(i) = \frac{t(1)}{t(i)}$. Figure 7 shows diagrams of computation acceleration with an increase in the number of nodes for different computational systems.

In addition to calculating the direct speedup of code execution, the measurement of the scaling efficiency was performed. In this case, the computational scaling efficiency is understood as the value $e(i) = \frac{s(i)}{i}$. The physical meaning of it is as follows. We can assume that in the case of ideal parallelization of computations increasing the number of nodes by n times, the execution time decreases exactly by n times. Thus, in the case of ideal parallelization, $s(i) = i$, and $e(i) = 1$. The efficiency of scaling computations is a convenient indicator of the quality of creating executable parallel code and comparing different computing systems with each other. Note that super-linear scalability is quite possible (when the value of $e(i)$ rises above one) [17], but this is more an exception than an expected effect.

Figure 7 shows a diagram of the scaling efficiency of computations for various computational segments, depending on the number of computational nodes used. It can be seen that for all computing systems, the scaling efficiency varies in the region of 0.8–0.9, although on some launch configurations there are dips even in the region of 0.7. Launches with a low parallelization efficiency are usually associated with a spread in the processing time of MPI processes in their domains. Despite the fact that the algorithm for decomposition of the computational mesh used in this article provided a uniform distribution of cells across domains, the processing time of a cell itself strongly depends on its physical properties and can differ significantly. For this reason, balancing the computational load on different nodes is possible only in a dynamic mode [18], which was not done within the framework of this study. We also note the high efficiency of scaling computations for compute nodes based on Xeon Cascade Lake microprocessors. These processors are the most modern of all the equipment used in the described experiment.

6. CONCLUSION

The efficiency of scaling high-load computing is an important aspect when developing parallel applications and performing calculations. Today, given the complexity of scientific and engineering problems and the amount of data being processed, it is difficult to rely on the efficiency of local station or server. To conduct qualitative research using modern mathematical models, the use of supercomputers is required. To use them effectively, it is needed to be able to create applications that can run in parallel on many computing nodes. To assess the efficiency of running parallel applications, it is convenient to use an indicator called the efficiency of computing scaling, the proximity of which to 1 indicates that the approaches and methods used for organizing high-performance computing reasonably reflect the needs of the problem and hardware. This article describes various aspects that are critical to achieving high compute scaling efficiency. Among them are the decomposition of the computational mesh, the mechanism for organizing interprocess exchanges during the counting process, and building the entire chain of calculations into a single sequence of actions. To obtain practical results in the course of the study, the problem of calculating physical processes on the surface of a streamlined body was used. All the work was performed on an unstructured surface computational mesh. To carry out the launches, several segments of the computing system of JSCC RAS were used. The highest computation scaling efficiency among which was achieved on the segment based on Xeon Cascade Lake microprocessors.

FUNDING

The work has been done at the JSCC RAS as part of the state assignment for the topic 0580-2021-0016. The supercomputer MVS-10P OP (Broadwell, KNL, Skylake and Cascade Lake segments), located at the JSCC RAS, was used during the research.

REFERENCES

1. R. Fadeev, K. Ushakov, M. Tolstykh, R. Ibrayev, V. Shashkin, and G. Goyman, "Supercomputing the seasonal weather prediction," in *Supercomputing, Proceedings of the 5th Russian Supercomputing Days RuSCDays 2019*, Ed. by V. Voevodin and S. Sobolev, Vol. 1129 of *Commun. Comput. Inform. Sci.* (Springer Nature, Switzerland AG, 2019).
2. Y. Hu, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian, "Massively scaling seismic processing on Sunway TaihuLight supercomputer," IEEE Trans. Parallel Distrib. Syst. **31**, 1194–1208 (2020).
3. E. Golovchenko, E. Dorofeeva, I. Gasilova, and A. Boldareva, "Numerical experiments with new algorithms for parallel decomposition of large computational meshes," Adv. Parallel Comput. **25**, 441–450 (2014).
4. J. Dorris, J. Kurzak, and P. Luszczek, "Task-based Cholesky decomposition on Knights Corner using OpenMP," in *ISC High Performance,* Lect. Notes Comput. Sci. **9945**, 544–562 (2016).
5. B. Shabanov, A. Rybakov, and S. Shumilin, "Vectorization of high-performance scientific calculations using AVX-512 intruction set," Lobachevskii J. Math. **40** (5), 580–598 (2019).
6. V. Kalantzis, "Data analytics, accelerators, and supercomputing: The challenges and future of MPI," XRDS **23**, 50–52 (2017).
7. A. Rybakov, "Distribution of the computational load between the nodes of a heterogeneous computational cluster," Progr. Produkty Sist. Algoritmy **1**, 1–7 (2018).
8. E. Golovchenko, "Review of graph decomposition algorithms," KIAM Preprint No. 002 (Keldysh Inst. Appl. Math., 2020).
9. The Stanford 3D Scanning Repository. http://graphics.stanford.edu/data/3Dscanrep/. Accessed 2021.
10. W. Wright, P. Struck, T. Bartkus, and G. Addy, "Recent advances in the LEWICE icing model," SAE Technical Paper (2015).
11. Y. Bourgault, H. Beaugendre, and W. Habashi, "Development of a shallow-water icing model in FENSAP-ICE," J. Aircraft **37**, 640–646 (2000).
12. A. Rybakov, "Inner respresentation and crossprocess exchange mechanism for block-structured grid for supercomputer calculations," Program Produkty Sist. Algoritmy **8** (1), 121–134 (2017).
13. JSCC RAS Supercomputing Resources. http://www.jscc.ru/supercomputing-resources/. Accessed 2021.
14. *Intel 64 and IA-32 Architectures Software Developer's Manual* (Intel Corp., 2019), Combined Vols.: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4.
15. G. Savin, B. Shabanov, A. Rybakov, and S. Shumilin, "Vectorization of flat loops of arbitrary structure using instructions AVX-512," Lobachevskii J. Math. **41** (12), 2566–2574 (2020).
16. J. Jeffers, J. Reinders, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming, Knights Landing Edition* (Morgan Kaufmann, 2016).
17. L. Benderskiy, D. Lyubimov, and A. Rybakov, "Analysis of scaling efficiency in high-speed turbulent flow calculations on a RANS/ILES supercomputer using the high resolution method," Tr. SRISA RAS **7** (4), 32–40 (2017).
18. R. Van der Wijngaart, E. Georganas, T. Mattson, et al., "A new parallel research kernel to expand research on dynamic load-balancing capabilities," in *ISC High Performance,* Lect. Notes Comput. Sci. **10266**, 256–274 (2017).