Efficiency Estimation for the Mathematical Physics Algorithms for Distributed Memory Computers

Igor Konshin[1,2]

[1] Dorodnicyn Computing Centre of FRC CSC RAS, Moscow 119333, Russia
[2] Marchuk Institute of Numerical Mathematics of the Russian Academy of Sciences, Moscow 119333, Russia
igor.konshin@gmail.com

Abstract. The paper presents several models of parallel program runs on computer platforms with distributed memory. The prediction of the parallel algorithm efficiency is based on algorithm arithmetic and communication complexities. For some mathematical physics algorithms for explicit schemes of the solution of the heat transfer equation the speedup estimations were obtained, as well as numerical experiments were performed to compare the actual and theoretically predicted speedups.

Keywords: Mathematical physics · Parallel computing · Parallel efficiency estimation · Speedup

1 Introduction

There are many works and Internet resources, which describe the properties of computational algorithms, parallel computing models, and also give recommendations on writing the most effective applications [1–3]. In this paper, an attempt is made to develop a constructive model of parallel computations, on the basis of which it is possible to predict the parallel efficiency of an algorithm implementation on a particular computing system.

When working on the shared memory computers, it is possible to construct such a model based on the Amdahl law (see [4,5]), while on distributed memory computers, it is necessary to take into account the parallel properties of both the implemented algorithm and computer system. In the present paper, we will perform a detailed analysis of the message transfer rate depending on the length of the message. If the message initialization time is ignored in the model, then it is possible to obtain more compact formulas for parallel efficiency estimates, while taking it into account, they become somewhat more complex, but also constructive and meaningful.

In this paper, we will focus our attention on the estimation of parallel efficiency for mathematical physics problems. In the scientific literature there are descriptions of a huge number of results on the achieved parallel efficiency for mathematical physics problems, but there are no theoretical estimates of what

efficiency could be achieved in practice, and thus a comparison of theoretical and actual ones.

This paper is organized as follows. Sections 2 and 3 give estimates of the algorithms parallel efficiency and their application to mathematical physics problems. Section 4 briefly describes the configuration of the computational cluster used. In Sections 5 and 6, the numerical experiments on combination of asynchronous interprocessor data exchanges and calculations, as well as the dependance of data transmission rate on the message length are studied in detail. In Section 7, the parallel efficiency estimates are refined on the basis of the results obtained, as well as their application to the mathematical physics algorithms is considered. Section 8 describes the results of numerical experiments, while the conclusion sums up the main results of the paper.

2 Estimates of the Algorithms Parallel Efficiency

To obtain an estimate of the parallel algorithm run time, the key point is to estimate the transmission rate of the message. Let us exploit for this purpose the widely used formula

$$T_c = \tau_0 + \tau_c L_c, \tag{1}$$

where τ_0 is the message initialization time, τ_c is the message transfer rate (i.e., the message transmission time for the unit message length), and T_c is the message transmission time for the message length L_c. A detailed study of the values of τ_0 and τ_c will be carried out later in Section 6, and now it will suffice for us to assume that the length of the messages in the algorithms under investigation is large enough and therefore, for simplicity, we can assume that $\tau_0 = 0$. This allows us to substantially simplify formula (1):

$$T_c = \tau_c L_c. \tag{2}$$

Let us estimate the speedup that can be achieved by using p processors in the implementation of some parallel algorithm. Let $T(p)$ be the time of solving the problem on p processors, then the speedup obtained using this algorithm will be expressed by the formula:

$$S = T(1)/T(p).$$

Following [4, 5], for further estimates we denote by L_a the total number of arithmetic operations of the algorithm, and by τ_a the execution time of one characteristic arithmetic operation. Similarly, let L_c be the total length of all messages, and τ_c is the already introduced time for transmitting a message of unit length. Then, the execution time of all arithmetic operations can be expressed by the formula $T_a = \tau_a L_a$, and the transmission time of all messages by $T_c = \tau_c L_c$.

Additionally, you can introduce a value

$$\tau = \tau_c/\tau_a, \tag{3}$$

which expresses the characteristic of the 'parallelism' property of the used computer, in other words meaning how many arithmetic operations can be performed during the transfer of one number to another processor.

Similarly, we introduce the value

$$L = L_c/L_a, \tag{4}$$

which expresses the characteristic of the 'parallelism' property of the algorithm under investigation, i.e. is the reciprocal of the number of arithmetic operations performed during the algorithm execution process to transfer of one number to another processor.

Now, in estimating the speedup, we can write:

$$\begin{aligned} S = S(p) = T(1)/T(p) = T_a/(T_a/p + T_c/p) = pT_a/(T_a + T_c) \\ = p/(1 + T_c/T_a) = p/(1 + (\tau_c L_c)/(\tau_a L_a)) = p/(1 + \tau L). \end{aligned} \tag{5}$$

In this connection, the estimate of the algorithm's parallel performance is written as follows:

$$E = S/p = 1/(1 + \tau L). \tag{6}$$

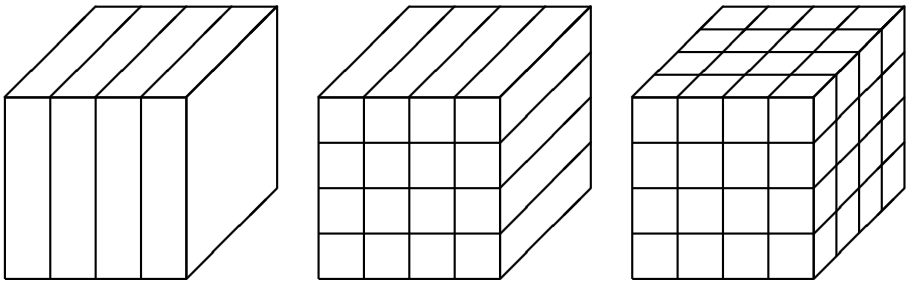
Remark 1. We note that a twofold decrease in the efficiency of the algorithm occurs when $L = 1/\tau$.

In papers [4, 5], a detailed discussion of the applicability of the obtained estimates (5)–(6) can be found, but we shall confine ourselves to explaining at first glance the strange fact that the formula for evaluating the efficiency of (6) does not include the number of processors used. In fact, the value of L depends on the total transmission length L_c, which, in turn, depends on the number of processors p.

3 Estimates for Mathematical Physics Algorithms

Now, following [6], we can proceed to the estimate of the parallel efficiency of mathematical physics algorithms. For simplicity, we restrict our consideration to explicit schemes used for nonstationary problems described by some finite-difference equations.

We consider the problem in a d-dimensional cube ($d = 1, 2, 3$) with side in n cells, the total number of d-dimensional cubic cells is equal to $N = n^d$. Let V denote the number of unknown functions per computational cell (for example, $V = 5$ for three velocities u, v, w, as well as pressure and temperature). To calculate the values at a new time step for each cell, it is required to know the values in the nearest neighboring cells, which means using the $(2d + 1)$-points d-dimensional discretization stencil (more complex discretizations, especially for equations with cross derivatives, can contain in the stencil up to 3^d points). We denote by C the average number of arithmetic operations per cell when calculating the solution at a new time step. Now, almost everything is ready for



$D = 1$, $p = r$, $L_a = CN/p$, $L_c \approx 2Vn^2$, $L \approx 2Vr/(Cn) \sim r/n$.

$D = 2$, $p = r \times r$, $L_a = CN/p$, $L_c \approx 4Vn^2/r$, $L \approx 4Vr/(Cn) \sim r/n$.

$D = 3$, $p = r \times r \times r$, $L_a = CN/p$, $L_c \approx 6Vn^2/r^2$, $L \approx 6Vr/(Cn) \sim r/n$.

Fig. 1. Distribution of data by processors for the three-dimensional problem of mathematical physics for r layers in each of one, two, and three directions.

estimating the speedup when solving the described problem on p processors, but the resulting speedup will depend essentially on the way the cells are assigned to the processors.

Figure 1 shows three different types of data distribution by processors, $D = 1, 2, 3$, using r layers in one, two, and three directions, respectively. The total number of processors in these cases is equal to $p = r^D$.

Assuming that $D \leqslant d$, we can begin to estimate the computational and communication costs.

The arithmetic cost per processor for all the cases under consideration is $L_a = CN/p = Cn^d/r^D$. Communication costs will depend on the number of boundary cells which need to be sent (received) on each of the processors, and will be $L_c = (2 - 2/r)DVn^{d-1}/r^{D-1}$. Note that $L_c = 0$ for $r = 1$ as required.

In this way, the main characteristic of the parallelism for the algorithm (4) is calculated as follows:

$$L = L_c/L_a = (2 - 2/r)DVn^{d-1}r^D/(Cn^2r^{D-1}) = (2 - 2/r)DVC^{-1} \cdot r/n \sim r/n. \tag{7}$$

This value is inversely proportional to the number of cells of the given processor in the direction of partitioning by processors.

Substituting the resulting expression for L into (6), we obtain

$$E = 1/(1 + (2 - 2/r)DVC^{-1}\tau \cdot r/n), \qquad S = pE \tag{8}$$

or to clarify the dependence on the original problem dimension d:

$$E(d, D) = 1/(1 + (2 - 2p^{-1/D})DVC^{-1}\tau \cdot p^{1/D}N^{-1/d}), \qquad S = pE(d, D). \tag{9}$$

To illustrate the specificity of (9), we give a short Table 1, which shows theoretical estimates of the parallel efficiency depending on the number of pro-

cessors p and the type of domain decomposition by processors D for the three-dimensional problem with the following set of parameters:

$$d = 3, \quad N = n^3, \quad n = 1000, \quad V = 5, \quad C = 30, \quad D = 1, 2, 3, \quad \tau = 10. \qquad (10)$$

The obtained efficiency values show the importance of using the right type of decomposition D, especially when utilizing a large number of processors.

4 INM RAS Cluster

Numerical experiments were performed on the cluster [7] of the Marchuk Institute of Numerical Mathematics of the Russian Academy of Sciences. Configuration of computational nodes from the 'x6core' segment used for calculations:

- Compute Node Asus RS704D-E6;
- 12 cores (two 6-core Intel Xeon processor X5650@2.67 GHz);
- RAM: 24 GB;
- Operating system: SUSE Linux Enterprise Server 11 SP1 (x86_64);
- Network: Mellanox Infiniband QDR 4x.

To build the code, we used the Intel C compiler version 4.0.1, with support for MPI version 5.0.3 [8].

5 Asynchronous Data Exchanges

The availability in the MPI standard the possibility of carrying out the asynchronous communications allows, after initializing the exchanges, without waiting for them to complete, immediately proceed with the calculations for which the data required is already available on the processor. The most of parallel computation guides [2] are permeated by the spirit of carrying out asynchronous data exchanges. However, the effect of overlapping calculations and data exchanges directly depends on specifics of architecture and implementation of MPI. Let we try to deal with this issue in more detail.

A special program was developed that implements test 1 from [9]. When carrying out the intensive calculations for the vector of numbers, there were sending the portions of processed values to another processor, which was used in its calculations at the next iteration. Asynchronous data exchanges were performed using the MPI_Isend and MPI_Irecv functions, as well as the MPI_Waitall function was used for completing all the data transmissions. The length of the vector was chosen to be $M = 2^{25}$, and the number of iterations was taken equal to 10. The number of portions to which the vector was partitioned before it was sent to another processor was chosen to be 1, 8, and 64. The calculations were performed on the INM RAS cluster [7] (see Section 4) in the 'x6core' segment.

The results of numerical experiments on asynchronous data exchanges are given in Table 2. Deviations at time measurements at various iterations were insignificant and were equal to from 1 to 5 percent. Let us consider the obtained

Table 1. Theoretical estimate of the parallel efficiency (9) using the parameters set (10).

p	$D=1$	$D=2$	$D=3$
1	1.00	1.00	1.00
10	0.97	0.98	0.98
64	0.82	0.95	0.96
729	0.29	0.85	0.92

Table 2. Time of test 1 on asynchronous data exchanges in the 'x6core' segment.

N_{drops}	$p=1$	$p=2$	$p=13$
1	15.42	17.18	18.52
8	15.43	15.58	15.78
64	15.43	15.63	15.77

results in more detail. Calculations and data exchanges were carried out at the first and last of the used processes. The parameter p means the number of cores used, thus, in the second column $p = 1$ the results of calculations without data exchanges are shown. The time measurements in this column are almost the same, we will use them as a reference point for our further observations. In the first line for $N_{\text{drops}} = 1$, we have a situation where all the data is sent in one portion, so the exchanges are synchronous and no overlap occurs with the calculations. The $p = 2$ column reflects the situation where two exchanging processes are physically located on one compute node and, despite the use of the MPI library, exchanges are actually conducted within the shared memory. One can see that due to synchronous exchanges ($N_{\text{drops}} = 1$), the total computation time increased by about 10%. The $p = 13$ column contains data when two exchanging processes are physically located on different compute nodes. Indeed, one computing node of the 'x6core' segment consists of two 6-core processors, and since calculations and exchanges are performed in the first and last core, the communications were carried out through the interprocessor network. Due to synchronous exchanges ($N_{\text{drops}} = 1$), the total computational time in this case increased by about 20%. With the increase in the number of portions N_{drops}, the total computational time decreases, tending to the respective time on 1 processor.

The obtained results allow to conclude that on the particular computer the effect of asynchronous exchanges is observed, but it is not significant and determinative.

It should also be noted that the theoretical consideration of the contribution of asynchronous exchanges to the overall solution time is extremely difficult, since, assuming that the delay time due to exchanges is negligible, the speedup of all algorithms should be considered linear (which is very far from reality even with an ideally uniform CPU usage). Our goal is to develop simple and constructive efficiency estimates for parallel applications for mathematical physics problems.

6 A Detailed Analysis of the Data Transmission Rate

To refine the estimates (5)–(6) and (9), it is necessary to perform a detailed analysis of the message transfer rate, taking into account the initialization time of the

message τ_0 from formula (1). Let us describe test 2 from [9], which investigates this dependence.

The described numerical experiment consisted in transmitting a message with the total length of $L_c = M = 2^m$ words of the type 'double', and the message was divided into $n_{c,i} = 2^i$ portions of length $L_{c,i} = L_c/n_{c,i} = 2^{m-i}$, $i = 0, ..., m$. For more statistical confidence, the test was repeated several times.

Thus $m + 1$ values of $T_{c,i}$ was obtained for the total message transmission time.

Let us derive the theoretical estimates of $T_{c,i}$, taking into account the time of initialization of the messages τ_0 in (1):

$$T_c(L_c) = T_c n_c = (\tau_0 + \tau_c L_c) n_c, \quad n_c = M/L_c. \tag{11}$$

For numerical experiments, the parameters $m = 25$ and $M = 2^m = 33554432$ were chosen, and the values $T_c(L_{c,i})$ for $L_{c,i} = 2^i$, $i = 0, ..., m$, were obtained for the 'x6core' segment. As the measured initialization time of the message we can take the value

$$\tau_0 = \max_{i=0,...,m} T_c(L_{c,i})/M = T_c(1)/M = 10.0/M \approx 3.0 \cdot 10^{-7}, \tag{12}$$

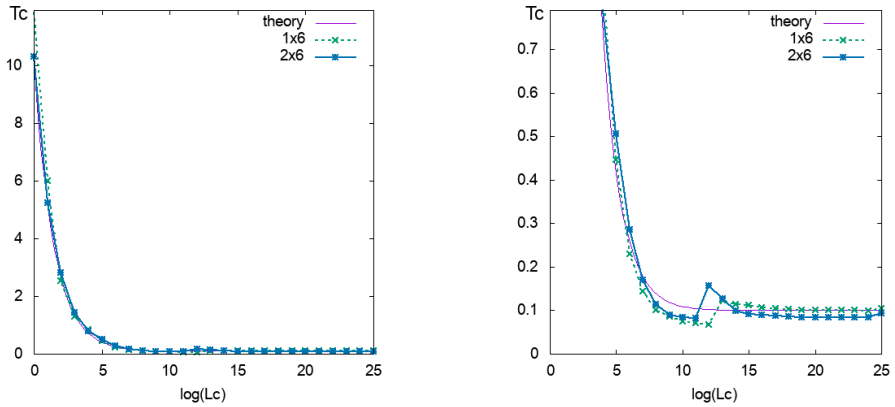
and as the average transmission rate of one number, we can use the value

$$\tau_c = \min_{i=0,...,m} T_c(L_{c,i})/M = 0.10/M \approx 3.0 \cdot 10^{-9}. \tag{13}$$

The values of the constants 10.0 and 0.10 were obtained from the described numerical experiment conducted in the 'x6core' segment, for simplicity, the values of the constants were rounded. You can see that for the values obtained, the ratio τ_0/τ_c is 100. In other words, we can conclude that the time $T_c(0)$ of transmitting an empty message will be only 2 times less than the time $T_c(100)$ of transferring 100 numbers of the type 'double'.

Figure 2 presents the results of comparing the experimental data with the theoretical ones, calculated from (11). The measurement data are statistically reliable: the average statistical deviation for most measurements is less than 1–5% for 10 repetitions of the test described. In Fig. 2a it can be seen that the theoretical curve practically coincides with the experimental data, which indicates the rationality of the assumptions made regarding the transmission rate, in spite of the significant rounding-off of the constants (12)–(13).

It is interesting to note that in Fig. 2b some retardation of transmission rate is observed when using a too long messages. This is probably related to the size of the internal buffers of the MPI library, which is selected when it is installed. In addition, for small messages with $L_c < 14$, the data transmission rate on one computing node is in most cases slightly more than for two nodes, and for $L_c \geqslant 14$, on the contrary. On average, the difference is about 10%. Probably, this effect is related to the specifics of MPI implementation and configuration of the nodes from the 'x6core' segment.



(a) (b)

Fig. 2. The total time for sending a message of 2^{25} double words by portions of length L_c. A theoretical estimate from (11) and actual calculations in the 'x6core' segment.

7 Estimation Refinement and its Usage for the Mathematical Physics Algorithms

Taking into account (1), we write down the time of data transmission:

$$T_c = n_c\tau_0/q + \tau L_c, \tag{14}$$

where L_c, as before, denotes the total length of all communications per processor, q is the number of overlapping subdomains (the necessary exchanges are performed once per q time steps), and n_c is the total number of communications per each processor. Note that in most algorithms n_c usually does not modified with increasing number of processors p.

We write down the time spent on arithmetic operations as follows:

$$T_a = (1+Q)\tau_a L_a, \tag{15}$$

where L_a, as before, denotes the total number of arithmetic operations performed per one processor, the new parameter Q expresses the proportion of the increase in the number of arithmetic operations, if the duplication of some arithmetic operations was performed. This duplication may be due to attempt of reduce the number or length of communications. If, as in the previously considered algorithms, there is no duplication, then $Q = 0$ and the formula (15) goes into the previously used one $T_a = \tau_a L_a$, while if the computations are duplicated, then Q takes some small positive value, depending on the algorithm features.

Using (14) and (15) in estimating the algorithm speedup, we obtain

$$\begin{aligned} S = S(p,\tau_0,Q) &= T(1)/T(p) = T_a(1)/(T_a(p)+T_c(p)) \\ &= \tau_a L_a/((1+Q)\tau_a L_a/p + n_c\tau_0/q + \tau_c L_c/p) \\ &= p/(1+\tau L + Q + pn_c\tau_0/(q\tau_a L_a)). \end{aligned} \tag{16}$$

Here, as before in formulas (3) and (4), the values τ and L mean the parallelism characteristics of the computer and the algorithm, respectively, while their product in the denominator characterizes the decrease in the efficiency due to data exchanges. The value Q, if it is different from 0, expresses the loss of speedup due to duplication of calculations; and the ratio of the delays in initializing of n_c transmissions $(n_c\tau_0/q)$ to the execution time of arithmetic operations on each processor $(\tau_a L_a/p)$ contributes to the speedup loss due to message initialization. The value τ_0, involved in the estimate (16), can be calculated rather accurately, as it was done in (12).

Let us turn again to the mathematical physics algorithms to estimate the remaining unknowns in (16). The number n_c of data transmissions per one time step will be equal or proportional to the number of subdomains connected to the specific processor. In the simplest case, in the notation of Section 3, depending on the method of decomposition of the domain D, we can take $n_c = 2D$.

Suppose now that instead of one layer of neighboring cells, we want to exchange q layers of cells at once, in order to perform exchanges in q times less often. This can lead to some reduction in the calculation time due to a decrease in the transmission initialization time, although it leads to some duplication of calculations. The total length of each transmission will increase by q times to qL_c, although the total length of transmissions for q time steps will remain practically unchanged. Note that duplicated computations will be performed at $(q-1)$ time steps exactly in the cells that participated in the transmissions, in this case the number of additional arithmetic operations being $L_c q(q-1)/2$. Thus, the estimate of the last unknown quantity in (16) is obtained:

$$Q = \tfrac{1}{2}q(q-1)L_c/L_a = \tfrac{1}{2}q(q-1)L. \tag{17}$$

Note that in the traditional 'high-communication no-extra-computation' scheme (HCNC) without duplicating the calculations (i.e., for $q = 1$), we get $Q = 0$. The case $q > 1$ corresponds to the 'low-communication high-extra-computation' scheme (LCHC) with $Q > 0$.

Remark 2. For further analysis of formula (17), suppose that the user is aimed to avoid double duplication of computations (i.e. $Q = 1$) and the associated double decrease in the computations efficiency, then it is sufficient to select the value of q not more than $\sqrt{2/L}$. Turn to formula (6), one can see that the same efficiency drop can be observed for the case $\tau L = 1$. Therefore, the previous restriction can also be rewritten as $q < \sqrt{2\tau}$. The computers encountered by the author are usually characterized by the parameter τ from the range from 10 to 30 (sometimes up to 100), therefore it is recommended to take $q < 5$.

Returning to the speedup estimate and substituting the values derived into (16), we obtain the final relation

$$S = p/\big(1+(\tau_{ca}+\tfrac{1}{2}q(q-1))(2-2p^{-1/D})DVC^{-1}p^{1/D}N^{-1/d}+2DC^{-1}q^{-1}pN^{-1}\tau_{0a}\big), \tag{18}$$

where $\tau_{ca} = \tau = \tau_c/\tau_a$ and $\tau_{0a} = \tau_0/\tau_a$ are denoted.

The impact on the speedup of various quantities has already been discussed, we note only that the computer-dependent quantity $\tau = \tau_c/\tau_a$ contributes to the second term of the denominator, and the ratio τ_0/τ_a is involved in the third term. We also note that the second term increases with increasing of q, while the third one decreases.

Remark 3. Rewriting (18) in the form

$$\begin{gathered} S = p/\big(1 + (\tau_{ca} + \tfrac{1}{2}q(q-1))C_1 + C_2\big), \\ C_1 = (2 - 2p^{-1/D})DVC^{-1}p^{1/D}N^{-1/d}, \quad C_2 = 2DC^{-1}q^{-1}pN^{-1}\tau_{0a}\big) \end{gathered} \tag{19}$$

we can take the derivative of S over q to estimate the optimal overlap size. In this way for $S'(q) = 0$ we obtain the cubic equation $2C_1q^3 - C_1q^2 - 2C_2 = 0$. For $q > 1$ there is only one root $q_* \approx (C_1/C_2)^{1/3}$, or

$$q_*^3 \approx C_1/C_2 = 2V^{-1}p^{1-1/D}N^{1/d-1}\tau_{0a}. \tag{20}$$

Substituting to (20) some reasonable values $D = d = 3$, $V = 5$, $p = 10^3$, $N = 10^6$, and $\tau_{0a} = 10^4$ we obtain $q_* \approx 3$.

At the end of this section, it should be noted that the estimates (9) and (18) can easily be generalized to the case of a region in the form of a rectangle or a parallelepiped. In the case of complex shape regions and/or arbitrary data distribution over processors, all the unknown values $\tau = \tau_c/\tau_a$ and $L = L_c/L_a$ can be easily calculated, for example, directly in the designed application just before the computations.

8 Results of Numerical Experiments

In Section 4 the specification of the computational cluster exploited for all numerical experiments made in this paper. We now turn to the description of the model problem to be solved.

As a model problem, the solution of the heat transfer equation in the d-dimensional cubic region $d \leqslant 3$ with the same number of cells in each direction was chosen. The cells in the computational domain were distributed over processors in D dimensions, $D \leqslant d$. The overlap of subdomains in q layers was considered, which allowed only one stage of data exchanges for q time steps. We focus on the standard finite-difference discretization of the heat transfer equation with an explicit scheme in time.

Figure 3 shows the results of computations for the model problem of dimension $100p \times 100 \times 100$, $d = 3$, $D = 1$, $p = 64$, using the subdomain overlap of size $q = 1, ..., 6$. The number of time steps was equal to 120. It can be noted that the minimal solution time is obtained for the case $q = 3$, which is in full agreement with the theoretical estimates made in Section 7.

Figures 4 and 5 contain the results for two problems of dimension $432 \times 432 \times 432$ and $512 \times 512 \times 512$, respectively. We analyzed all possible one-,

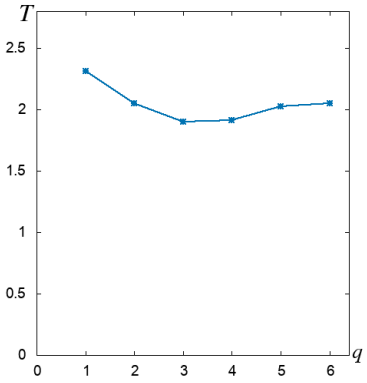


Fig. 3. The solution time depending on the subdomain overlap size $q = 1, ..., 6$.

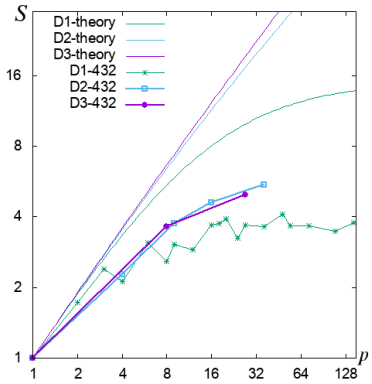


Fig. 4. Speedup for the $432 \times 432 \times 432$ problem for $D = 1, 2, 3$.

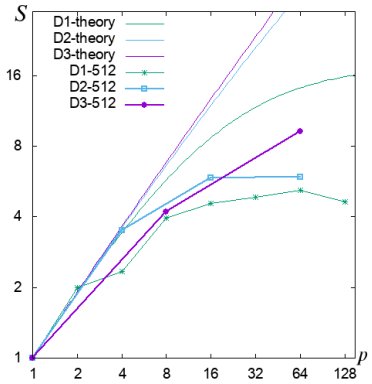


Fig. 5. Speedup for the $512 \times 512 \times 512$ problem for $D = 1, 2, 3$.

two- and three-dimensional ($D = 1, 2, 3$) distributions over processors, for all variants of processor ratio, in which all subdomains have the same size. We considered a conventional version with a minimum overlap of size $q = 1$, with 120 time steps. Theoretical estimates based on formula (18) give a slightly more optimistic forecast of the algorithm speedup, but it is clear that the use of a larger dimension for distribution over processors in most cases provides greater speedup, as it follows from the theoretical estimates obtained in Sections 3 and 7.

9 Conclusion

Constructive estimates for the algorithm parallel efficiency are obtained, which include the characteristics of the parallelism of the computer and the parallelism of the algorithm itself. In addition, the estimates are obtained that take into account the transmission initialization time, as well as the ability to group

messages. For explicit discretization schemes applied to mathematical physics problems, estimates are obtained that, in addition to the geometric parameters of the problem and the type of cell distribution over processors, also include the number of unknown functions per cell, as well as the number of arithmetic operations per cell at one time step. The dependence of the message transfer rate on message length is analyzed in detail, and also the expediency of performing calculations at the same time as the asynchronous communications. For the sample heat transfer problem, a direct comparison of the experimental results with the theoretical estimates on the parallel efficiency was performed. Confirmed the conclusion about the profitability of using the higher dimension type of data distribution over processors.

Acknowledgements. The theoretical part of this work has been supported by the Russian Science Foundation through the grant 14-11-00190. The experimental part was partially supported by RFBR grant 17-01-00886.

References

1. AlgoWiki: Open encyclopedia of algorithm properties. URL: http://algowiki-project.org (accessed: 15.04.2018)
2. Voevodin V.V., Voevodin Vl.V.: Parallel computing. BHV-Petersburg, St. Petersburg, 2002 (in Russian)
3. Gergel V.P., Strongin R.G.: Fundamentals of parallel computing for multiprocessor computer systems. Publishing house of the Nizhny Novgorod State Univ., Nizhny Novgorod, 2003 (in Russian)
4. Konshin I.N., Parallel computational models to estimate an actual speedup of analyzed algorithm. In: Russian Supercomputing Days: Proc. of the Int. Conf. (September 26-27, 2016, Moscow, Russia). Moscow State University, Moscow, 2016, pp. 269-280 (in Russian) http://2016.russianscdays.org/files/pdf16/269.pdf
5. Konshin I., Parallel computational models to estimate an actual speedup of analyzed algorithm. In: Vol. 687 of Communications in Computer and Information Science (Ed. Vl. Voevodin), Springer, 2017, 304–317
6. Konshin I.N., Parallelism in computational mathematics. International Summer Supercomputer Academy. Track: Parallel Algorithms of Algebra and Analysis and Experiments of Supercomputer Modeling. MSU, Moscow, 2012. URL: http://academy2012.hpc-russia.ru/files/lectures/algebra/0704_1_ik.pdf (accessed: 15.04.2018) (in Russian)
7. INM RAS cluster. URL: http://cluster2.inm.ras.ru (accessed: 15.04.2018) (in Russian)
8. MPI: The Message Passing Interface standard. http://www.mcs.anl.gov/research/projects/mpi/ (accessed: 15.04.2018)
9. Bajdin G.V.: On some stereotypes of parallel programming. Vopr. Atomn. Nauki Tekhn., Ser. Mat. Model Fiz. Prots., 2008, No. 1, 67–75 (in Russian)