CONFERENCE PROCEEDINGS

17-19 October 2018, Almaty, Kazakhstan

www.aict.info/2018

SESSION 3. HIGH PERFORMANCE COMPUTING AND MACHINE LEARNING

Implementation of the Two-Dimensional Elliptic Equation Model in LuNA Fragmented Programming System

Beimbet Daribayev
Al-Farabi Kazakh National University
Almaty, Kazakhstan
beimbet.daribaev@gmail.com

Vladislav Perepelkin
ICMMG SB RAS
Novosibirsk, Russia
perepelkin@ssd.sscc.ru

Danil Lebedev
Al-Farabi Kazakh National University
Almaty, Kazakhstan
danil.lebedev.0881@gmail.com

Darkhan Akhmed-Zaki
University of International Business
Almaty, Kazakhstan
darhan_a@mail.ru

***Abstract*— Numerical modeling of phenomena on modern supercomputers using finite difference methods requires development of complex application software capable of adapting to both hardware configuration and the simulation process in order to provide satisfactory efficiency. In the paper two parallel programs, which employ the alternating-triangular method for solving of Dirichlet problems for the Poisson equation in unit square, are studied. The first program is a conventional one, based on Message Passing Interface (MPI), while the second one is developed with LuNA programming system. LuNA automates construction of numerical programs for multicomputers. Scalability and efficiency of both programs are compared.**

Keywords—fragmented programming; LuNA; numerical solution; MPI; parallel program; alternating-triangular method

I. INTRODUCTION

Numerical simulation by finite difference methods with the use of supercomputers is increasingly encountering questions of the effectiveness of algorithms and the parallel programs that implement them. To implement models with a large number of points, it is necessary to use algorithms that allow efficient and scalable parallel implementation. In particular, programs that use centralized management and/or storage of data, data transmission for a long distance (in the sense of network topology) have a poor scalability.

An example of an algorithm that allows an effective and scalable implementation is an alternating-triangular method with a Chebyshev set of parameters. It is a rapidly convergent two-layer iterative method used to solve the difference elliptic equations arising in the approximation of second-order elliptic equations.

However, parallel implementation of such algorithms is a complex task of system parallel programming, because the program requires synchronization of individual computing processes, data transmission over the network, and the like.

To reduce the complexity of developing parallel programs, tools are used that partially automate their design. These systems include LuNA [1], PaRSEC [2-3], Charm++ [4].

The aim of this paper is a comparative study of the properties of parallel programs implementing alternating-triangular method used to solve the model two-dimensional elliptic equation. Two parallel programs are compared, one of which is obtained using the traditional approach (based on the MPI message interface) and the other using the LuNA fragmented programming system.

The second section of the article contains the necessary definitions and description of the LuNA system. The third section describes the statement of the problem, the applied algorithm and the scheme of its parallel implementation. The fourth section presents the results of numerical experiments and a comparative analysis of their productivity.

II. LUNA FRAGMENTED PROGRAMMING SYSTEM

In automation systems for the construction of parallel programs, computation models other than the widely used model of interacting sequential processes are used, in particular, in the standard MPI (Message Passing Interface). This is due to the fact that different models of computation have different properties. For example, the dynamic migration of an MPI process from one node to another as a method of load balancing is almost impossible because of the need to transfer all process memory over the network, possible open file descriptors, and the like problems that arise from the computation model used in MPI.

The LuNA system uses a computation model, called a fragmented program (FP). In this model, the task data is

represented as a set of individual units, called data fragments (DF). DF's are immune and are variables of a single assignment. The DF values can have both a base type (integer, real, etc.), and a compound type (a fragment of a grid, a vector, etc.). The calculations of the problem are set by a set of processes, each of which is associated with a set of input and output DF's and calculates the output DF values from the input values. CF has no side effects.

The computational process consists in the fact that CF's for which the values of all their input DFs are known and the values of the output are unknown are fulfilled, which leads to the calculation of new DF's. As a consequence, new CF's can be executed, etc. The computing process ends when none of the CF's can be executed, or the values of all required DF's (DF output programs) are obtained.

Such a computation model has several advantages in terms of simplifying the creation of parallel programs with the required properties.

1) In practice, CF's are implemented as calls to conventional sequential procedures, which allows, in essence, to be limited to sequential programming when developing a parallel program. This also allows you to re-use the accumulated baggage of consecutive programs.

2) The FP is not tied to the configuration of the multicomputer. The CF can be placed in the memory of any node, transmitted over the network to another node. The CF can be executed on any node, provided that all its input DFs are present in the memory of this node at the time of execution, and there is enough free space for the DF outputs. This makes it possible to automatically distribute the DF and CF to the nodes of the multicomputer as the program executes, thereby ensuring a uniform load on the calculator.

3) The programmer does not need to synchronize processes, monitor the availability of data or release memory - these functions are assumed by the executive system.

As a result, the programmer can develop parallel programs without experiencing the difficulties inherent in parallel programming.

The model also has drawbacks, which reduce to the fact that the effective implementation of the FP is a complex optimization task. In practice, this means a relatively low efficiency of FP performance. High performance can be achieved either for a limited class of applications, or by further manual optimization of the performance of the FP (for which the LuNA system has appropriate means).

III. Problem formulation

The solution of the model two-dimensional elliptic equation is considered. These equations cover a wide class of applied problems. To solve the difference elliptic equations arising when approximating second-order elliptic equations, two-layer iterative methods are often used. One of the rapidly converging two-layer iterative methods is the alternating-triangular method with the Chebyshev set of parameters proposed by A.A. Samarskij [6]. For parallel and fragmented implementation, a parallel variant of the alternating-triangular method was chosen to solve elliptic equations on the processor lattice [7].

We consider the Dirichlet problem for the Poisson equation in the unit square.

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=-f(x,y),0<x<1,0<y<1 \qquad (1)$$

and with initial conditions

$$u(0,y)=u(1,y)=u(x,0)=u(x,0)=0 \qquad (2)$$

The right-hand side f was chosen from the condition that the exact solution of problem (1) be the function $u=32x(1-x)(1-y)$.

For the numerical solution of problem (1), an alternating-triangular method (ATM) was used with a Chebyshev set of parameters τ_k. For the number of iterations in this method the estimate $n_0(\varepsilon)=O(\sqrt{N}\ln(2/\varepsilon))$, where ε - required relative error, N - the number of nodes of the difference grid in one spatial direction.

In the ATM, the difference operator approximating the original problem is represented in the form:

$$B=(D+\omega_0 R_1)D^{-1}(D+\omega_0 R_2)$$

where

$$R_1+R_2=R,\ R_1=R_2,\ R>0,\ D=D^*>0.$$

We assume that $A=A^*>0$ operator A energy equivalent to the operator R:

$$c_1 R\le A\le c_2 R,$$

and the ratio c_2/c_1 not too large.

As an a priori information in the ATM, δ and Δ are used from the inequalities:

$$\delta D\le R,\ R_1 D^{-1}R_2\le 0.25\Delta R.$$

It is proved [6] that the optimal value of ω_0 is

$$\omega_0=2/\sqrt{\delta\Delta},$$

and for the iteration number for the Chebyshev set of parameters τ_k and the optimal value ω_0, the estimate

$$n\ge n_0(\varepsilon)=\frac{\ln(2/\varepsilon)}{2\sqrt{2}}\sqrt[4]{\frac{\Delta}{\delta}}\sqrt{\frac{c_2}{c_1}}$$

With this value of n, we have the following estimate $\left\|y^n-u\right\|_F\le\left\|y^0-u\right\|_F$, where $F=A,B$ or $AB^{-1}A$. In ATM, the operator D given by the diagonal matrix is chosen from the condition of minimum number of iterations.

In the present paper, $R=-\Lambda$, where

$$\Lambda y = y_{\bar{x}_1 x_1} + y_{\bar{x}_2 x_2}$$

The following difference operators are used in the classic version of ATM:

$$D = E,\ R_1 y = y_{\bar{x}_1} / h_1 + y_{\bar{x}_2} / h_2,\ R_2 y = y_{x_1} / h_1 + y_{x_2} / h_2.$$

With this choice of operators, the parameter values are defined as follows:

$$\delta = \sum_{\alpha=1}^{2} 4h_\alpha^{-2} \sin^2(\pi h_\alpha / 2l_\alpha),\ \Delta = \sum_{\alpha=1}^{2} 4h_\alpha^{-2}$$

where l_α, h_α - the size of the calculated area and the step of the difference grid in the direction α.

To find y^{k+1}, we use the following algorithm [6]:

Step 1. $r^k = Ay^k - f;$

Step 2. $(E + \omega_0 R_1)\overline{w}^k = r^k;$

Step 3. $(E + \omega_0 R_2)w^k = \overline{w}^k;$

Step 4. $y^{k+1} = y^k - \tau_{k+1} w^k.$

As the τ_{k+1}, the Chebyshev set of parameters is used.

IV. A PARALLEL VERSION OF THE ALTERNATING-TRIANGULAR METHOD ON THE PROCESSOR GRID

As a parallel version of the ATM, the algorithm proposed in [7] was used. Its feature is that the operators R_1 and R_2 are represented as the sum of two operators.

$$R_1 = R_1^1 + R_1^2,\ R_2 = R_2^1 + R_2^2$$

where

$$R_1^\alpha y = \begin{cases} y_{\bar{x}_\alpha} / h_\alpha, i_\alpha = N_{i_\alpha} 2n_\alpha + 1, \ldots, N_{i_\alpha}(2n_\alpha + 1) - 1, n_\alpha = 0,1,\ldots, M_{1_\alpha} \\ -y_{x_\alpha} / h_\alpha, i_\alpha = N_{i_\alpha}(2n_\alpha + 1) + 1, \ldots, N_{i_\alpha}(2n_\alpha + 2) - 1, n_\alpha = 0,1,\ldots, M_{2_\alpha} \\ -y_{\bar{x}_\alpha x_\alpha} - y / h_\alpha^2, i_\alpha = N_{i_\alpha} 2n_\alpha, n_\alpha = 1,\ldots, M_{1_\alpha} \\ y / h_\alpha^2, i_\alpha = N_{i_\alpha}(2n_\alpha + 1), n_\alpha = 0,1,\ldots, M_{2_\alpha} \end{cases}$$

$$R_2^\alpha y = \begin{cases} -y_{x_\alpha} / h_\alpha, i_\alpha = N_{i_\alpha} 2n_\alpha + 1, \ldots, N_{i_\alpha}(2n_\alpha + 1) - 1, n_\alpha = 0,1,\ldots, M_{1_\alpha} \\ y_{\bar{x}_\alpha} / h_\alpha, i_\alpha = N_{i_\alpha}(2n_\alpha + 1) + 1, \ldots, N_{i_\alpha}(2n_\alpha + 2) - 1, n_\alpha = 0,1,\ldots, M_{2_\alpha} \\ y / h_\alpha^2, i_\alpha = N_{i_\alpha} 2n_\alpha, n_\alpha = 1,\ldots, M_{1_\alpha} \\ -y_{\bar{x}_\alpha x_\alpha} - y / h_\alpha^2, i_\alpha = N_{i_\alpha}(2n_\alpha + 1), n_\alpha = 0,1,\ldots, M_{2_\alpha} \end{cases}$$

where

$N_{i_\alpha} = N_\alpha / N_{k_\alpha},\ N_{k_\alpha} \geq 3, \alpha = 1,2,$

$M_{1_\alpha} = M_{2_\alpha} = 0.5(N_{k_\alpha} - 2)$ if N_{k_α} is even,

$M_{1_\alpha} = 0.5(N_{k_\alpha} - 1), M_{2_\alpha} = 0.5(N_{k_\alpha} - 3)$ if N_{k_α} is odd.

It's obvious that $R_1 + R_2 = R = -\Lambda > 0.$

The calculation for each iteration consists of four stages analogous to the classical ATM.

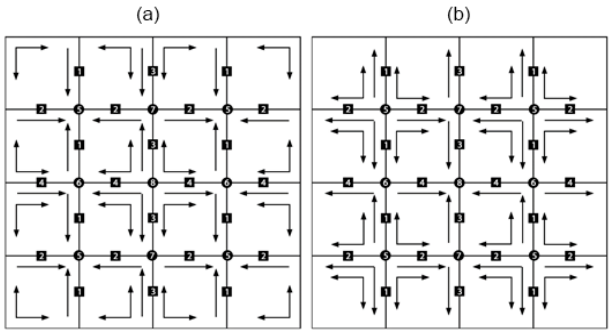

Figure 1. Calculation scheme for a parallel ATM algorithm

The first stage can be performed independently on each process if the values of y^k corresponding to the subregion are determined and the values of the border points from neighboring subregions are determined.

$$r^k = Ay^k - f\ .$$

The second and third stage of the calculations are shown in Fig. 1 (a) and (b), respectively, and proceed according to the following scheme.

The second stage of calculations begins at point 8, then this calculation continues on lines 3 and 4 in the directions indicated by the arrows. The calculation then occurs at interior points of the geometric subregions. After calculation at internal points, the calculation takes place at points 6 and 7 and on lines 1 and 2 in the directions indicated by arrows and ends at points 5. For calculations at points 6,7 and on lines 1,2 and at point 5, it is necessary to exchange information between neighboring processors.

$$\overline{w}^k = (D + \omega_0 R_1)^{-1} r^k\ .$$

Parallel implementation of the third stage of calculations occurs similarly (Fig. 1 (b)). The calculation starts at points 5, then occurs on lines 7,2, then at the interior points of the subregions, then at points 6,7 and on lines 3,4 and ends at point 8. The necessary information exchange is carried out between the neighboring processors.

$$w^k = (D + \omega_0 R_2)^{-1} \overline{w}^k.$$

It should be noted that in each processor the values of w^k at the border points of neighboring sub-regions are either forwarded during the calculation or can be directly calculated (with the exception of a number of corner points). This approach will reduce the number of shipments. Therefore, in the fourth stage, in each processor, y^{k+1} can be computed at the interior points of the subdomain, the points of internal boundaries, and at points adjacent to internal boundaries located in adjacent subareas.

$$y^{k+1} = y^k - \tau_{k+1} w^k.$$

Calculations at all stages occur at each iteration *k* before the method converges.

V. RESULTS

Serial, parallel program using the MPI standard, and a fragmented program were implemented. The testing was carried out for a different number of points with respect to spatial coordinates and different number of processors for parallel implementation. Sequential, parallel and LuNA programs were run on the SSCC cluster. A parallel program was run on 8 processes per node. The LuNA program started in 8 threads per node mode. The number of nodes changed depending on the decomposition of the area. The purpose of testing was to determine the effectiveness of various implementations of the numerical algorithm under consideration. The test results are shown in Fig. 2.

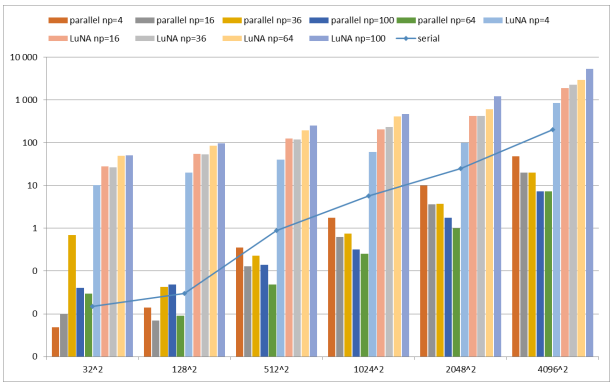

Figure 2. Calculation time (in seconds), depending on the number of grid points

Fig. 2 shows that the parallel implementation of the algorithm is the fastest, depending on the number of processes. At the 32x32 task size, the smallest counting time for a parallel implementation, which was started on 4 processes. With the increase in the size of the task when running parallel implementation on a different number of processes, the minimum calculation time is reached on a larger number of processes. So, with the task size 128x128, the minimum calculation time for the number of processes is 16. Also, when the number of processes increases without changing the size of the area, the calculation time increases. This is due to the fact that the counting time is reduced, but more time is spent on data transmission.

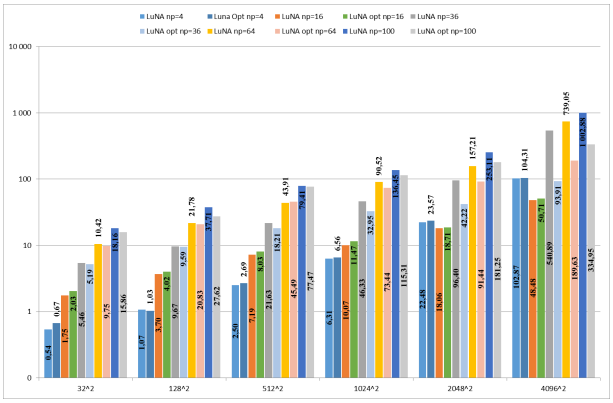

Figure 3. Calculation time of LuNA and optimal LuNA programs (in seconds), depending on the number of grid points

Since the fragmented program showed a not very good result, its optimization was carried out, which consisted in the fact that the data fragments obtained two-dimensional coordinates so that the system algorithms could more efficiently distribute the computations. It can be seen from the Fig. 3 that the difference between the initial fragmented program and the optimized program is practically absent in the small size of the problem. This is due to the fact that the time spent on computing is very small, and the main time is spent on communication. Also, on a small number of processes, there is practically no difference due to the fact that all calculations take place on the same node and the program is practically running in the shared memory mode, due to which the data exchange takes place very quickly. With the increase in the size of the task and the number of processes, the computation time of the optimized program is less than the original one.

VI. CONCLUSIONS AND FUTURE WORKS

The efficiency and scalability of two parallel programs implementing the alternating-triangular method for solving the Dirichlet problem for the Poisson equation in the unit square were investigated. The study showed that MPI-program has higher efficiency and scalability than the LuNA-program. The LuNA program was easier to develop and debug due to the lack of the need to program the binding to the calculator (distribution of subtasks by cluster nodes, etc.). In addition, when optimizing a fragmented program, the computation time was reduced. In the future, it is planned to develop a fragmented program on direct control for the parallel algorithm under consideration.

ACKNOWLEDGMENT

This work was financially supported by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan, grant No. AP05134651.

REFERENCES

[1] V.E. Malyshkin, V.A. Perepelkin, LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // Proceedings of the 11th International Conference on Parallel Computing Technologies, LNCS 6873, Springer, 2011, pp. 53-61.

[2] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault and J. Dongarra, PaRSEC: exploiting heterogeneity to enhance scalability, IEEE Comput Sci Eng, vol. 15(6), pp. 36-45, 2013.

[3] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Herault, J. Kurzak J. Langou, P. Lemarinier, H. Ltaeif, P. Luszczek, A. YarKhan and J. Dongarra, Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA, Proceedings of the IPDPS 2011 Workshops, IEEE, pp. 1432-1441, 2011.

[4] (2015) Charm++. [Online]. Available: http://charm.cs.uiuc.edu/

[5] (2015) NAMD: Scalable molecular dynamics library. [Online]. Available: http://www.ks.uiuc.edu/Research/namd/

[6] A.A. Samarskij, Teorija raznostnyh sistem, M.: Nauka, 1989.

[7] O.Ju. Miljukova, B.N. Chetverushkin, Parallel'nyj variant poperemenno-treugol'nogo metoda, Zh. vychisl. matem. i matem. fiz., 1998, tom 38, nomer 2, pp. 228–238.