European Society of Computational Methods in Sciences and Engineering (ESCMSE)

Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM)
vol. 19, no. 1-2, 2026, pp. 49-55
ISSN 1790–8140

Parallel numerical algorithms for solving two-dimensional elliptic problem

M. A. Sultanov[a,b], **E. N. Akimova**[1,c,d], **V. E. Misilov**[c,d], **B. T. Sarsenov**[a,b], **Y. Nurlanuly**[a,b]

[a] Institute of Mathematics and Mathematical Modeling,
Pushkin Street 125, Almaty 050010, Kazakhstan
[b] Department of Mathematics, Faculty of Natural Science,
Khoja Akhmet Yassawi International Kazakh-Turkish University,
B. Sattarhanov Street 29, Turkestan 160200, Kazakhstan
[c] Krasovskii Institute of Mathematics and Mechanics,
S. Kovalevskaya Street 16, Ekaterinburg 620077, Russia
[d] Institute of Radioelectronics and Information Technology,
Ural Federal University, Mira Street 19, Ekaterinburg 620002, Russia

Received October, 2025; accepted in revised form October, 2025

Abstract: This work is devoted to the construction of efficient parallel algorithms and development of parallel code for solving the two-dimensional stationary elliptic differential equation. The numerical algorithm is based on a finite difference scheme. After discretization and application of this scheme, we obtain a large system of linear algebraic equations with a block-tridiagonal matrix. To solve this equation, we apply the matrix sweep method. In this work, we have developed a parallel code for graphics processors using the CUDA technology and cuBLAS and cuSOLVER libraries. We have achieved a 5-fold speedup in comparison with the earlier CPU code.

Keywords: Partial differential equation; elliptic problem; finite difference scheme, matrix sweep method, CUDA

Mathematics Subject Classification: 35J25; 65Y05

PACS: 02.30.Jr; 02.60.Lj

1 Introduction

Elliptic partial differential equations are a basis of many mathematical models in various field of physics, including heat problems, electrostatic, elastisity, etc [1, 2]. Practical application usually require solving these equation by numerical methods [4], such as finite difference [5], finite element, or finite volume methods [6]. This leads to a necessity for solving large systems of linear algebraic

[1]Corresponding author. E-mail: aen15@yandex.ru

equations (SLAE). Thus, developing and implementing efficient parallel algorithms for such systems is an important task.

Previously, we have developed efficient parallel algorithms and codes for solving forward and inverse problems for various types of two-dimensional differential equations [7, 8]. These algorithms use the matrix sweep (block elimination) methods to solve a SLAE with block-tridiagonal matrix.

In this work, to solve a boundary value problem for a two-dimensional linear second-order elliptic equation, we develop an efficient implementation of the matrix sweep algorithm for graphics processors using the NVIDIA CUDA technology and cuBLAS and cuSOLVER libraries. We conduct a series of numerical experiments to study the performance of our code and compare it with the earlier implementation based on the Intel MKL library for central processors.

2 Problem Statement and Numerical Method

Consider an elliptical equation in divergence form

$$-\nabla \cdot (M \nabla u) + qu = 0,$$

where M is a symmetric positive definite matrix.

In our work, we consider a two-dimensional stationary diffusion problem with variable coefficients $M = \begin{pmatrix} p(x_1) & 0 \\ 0 & r(x_1) \end{pmatrix}$. The problem takes the following form:

$$\begin{aligned} &r(x_1)u''_{x_2x_2}(x_1,x_2) + [p(x_1)u'_{x_1}(x_1,x_2)]'_{x_1} - q(x_1)u(x_1,x_2) = 0, \\ &\xi_1 < x_1 < \xi_2, \quad \zeta_1 < x_2 < \zeta_2 \end{aligned} \tag{1}$$

with the following boundary conditions

$$\begin{cases} u(\xi_1, x_2) = f_1(x_2), & \zeta_1 \le x_2 \le \zeta_2, \\ u(\xi_2, x_2) = g_1(x_2), & \zeta_1 \le x_2 \le \zeta_2, \\ u(x_1, \zeta_1) = f_2(x_1), & \xi_1 \le x_1 \le \xi_2, \\ u(x_1, \zeta_2) = g_2(x_1), & \xi_1 \le x_1 \le \xi_2, \end{cases} \tag{2}$$

where $f_1(x_2)$, $f_2(x_1)$, $g_1(x_2)$, $g_2(x_1)$ are continuous functions, $q(x_1)$ is continuous, $p(x_1)$ and $r(x_1)$ are continuously differentiable, and $r(x_1) > 0$, $p(x_1) > 0$, $q(x_1) \ge 0$.

To solve the boundary problem of finding the unknown function $u(x_1, x_2)$, we apply a finite difference scheme. We construct a mesh with $(M+1)$ and $(N+1)$ points and spacings $\Delta x_1 = h_1 = (\xi_2 - \xi_1)/M$ and $\Delta x_2 = h_2 = (\zeta_2 - \zeta_1)/N$. The mesh nodes are denoted as $x_{1,i} = \xi_1 + ih_1$, $x_{2,j} = \zeta_1 + jh_2$, $i \in \{0, 1, ..., M\}$, $j \in \{0, 1, ..., N\}$, and the values of discretized solution as $Y_{i,j} = u(x_{1,i}, x_{2,j})$. Also, denote the coefficients

$$\begin{aligned} a_i &= -\frac{p_{i-1} + p_i}{2h_1^2}, \\ b_i &= -\frac{p_i + p_{i+1}}{2h_1^2}, \\ c_i &= -\frac{2r_i}{h_2^2} - \frac{p_{i-1} + 2p_i + p_{i+1}}{2h_1^2} - q_i, \\ d_i &= -\frac{r_i}{h_2^2}. \end{aligned}$$

Applying the second-order approximation, we construct a finite difference scheme, which leads to a SLAE with block-tridiagonal matrix A in form

$$AY = F, \tag{3}$$

$$A = \begin{bmatrix} C_1 & -B_1 & & & & & \\ -A_2 & C_2 & -B_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & -A_j & C_j & -B_j & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -A_{N-2} & C_{N-2} & -B_{N-2} \\ & & & & & -A_{N-1} & C_{N-1} \end{bmatrix},$$

$j \in \{1, 2, ..., N-1\}$.

Its blocks $A_j,\ B_j,\ C_j$ have the following structure

$$A_j = B_j = \begin{bmatrix} d_1 & & & & & & \\ & d_2 & & & & & \\ & & \ddots & & & & \\ & & & d_i & & & \\ & & & & \ddots & & \\ & & & & & d_{M-2} & \\ & & & & & & d_{M-1} \end{bmatrix},$$

$$C_j = \begin{bmatrix} c_1 & -b_1 & & & & & \\ -a_2 & c_2 & -b_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & -a_i & c_i & -b_i & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -a_{M-2} & c_{M-2} & -b_{M-2} \\ & & & & & -a_{M-1} & c_{M-1} \end{bmatrix},$$

$i \in \{1, 2, ..., M-1\},\ j \in \{1, 2, ..., N-1\}$.

Sought vector Y of values $Y_{i,j}$ is reshaped from two dimensions to a "flat" form in a row-major order

$$Y = \Big[Y_1, Y_2, ..., Y_{N-1}\Big]^T,$$

$$Y_1 = \Big[Y_{1,1},\ Y_{2,1}, ...,\ Y_{M-2,1},\ Y_{M-1,1}\Big];$$

$$Y_2 = \Big[Y_{1,2},\ Y_{2,2}, ...,\ Y_{M-2,2},\ Y_{M-1,2}\Big];$$

$$...$$

$$Y_{N-1} = \Big[Y_{1,N-1},\ Y_{2,N-1}, ...,\ Y_{M-1,N-1}\Big].$$

Right-hand side vector F is constructed similarly with respect to the boundary values

$$F = \Big[F_1, F_2, ..., F_{N-1}\Big]^T;$$

$$F_1 = \Big[-\frac{r_1}{h_2^2}Y_{1,0} - \frac{p_0+p_1}{2h_1^2}Y_{0,1},\ -\frac{r_2}{h_2^2}Y_{2,0},\ ...,$$
$$-\frac{r_{M-2}}{h_2^2}Y_{M-2,0},\ -\frac{r_{M-1}}{h_2^2}Y_{M-1,0} - \frac{p_{M-1}+p_M}{h_1^2}Y_{M,1}\Big];$$

$$F_2 = \Big[-\frac{p_0+p_1}{2h_1^2}Y_{0,2},\ 0,\ ...,\ 0,\ -\frac{p_{M-1}+p_M}{2h_1^2}Y_{M,2}\Big];$$

$$...$$

$$F_{N-1} = \Big[-\frac{r_1}{h_2^2}Y_{1,N} - \frac{p_0+p_1}{2h_1^2}Y_{0,N-1},\ -\frac{r_2}{h_2^2}Y_{2,N},\ ...,$$
$$-\frac{r_{M-1}}{h_2^2}Y_{M-1,N} - \frac{p_{M-1}+p_M}{2h_1^2}Y_{M,N-1}\Big].$$

For the stability and convergence analysis of this scheme, we can apply the discrete maximum principle and other results from [3].

Theorem 2.1 *Consider a difference scheme*

$$a_iY_{i-1,j} + b_iY_{i+1,j} + d_iY_{i,j-1} + d_iY_{i,j+1} - c_iY_{i,j} = 0 \tag{4}$$

for $i \in \{1, 2, ..., M-1\}$, $j \in \{1, 2, ..., N-1\}$ *with Dirichlet boundary conditions* (2).

Let $p, r \in C^1[0,1]$, $q \in C[0,1]$ *and exact solution* $u \in C^4([0,1] \times [0,1])$.

Then, the scheme (4)*:*

- *approximates the original problem with the second order of accuracy* $O(h_1^2 + h_2^2)$*;*
- *is unconditionally stable with respect to boundary conditions and the following estimate for its solution* $\widetilde{Y}$ *is valid:*

$$\left\|\widetilde{Y}\right\| \le \max\left\{ \|f_1\|, \|f_2\|, \|g_1\|, \|g_2\| \right\};$$

- *converges to the exact solution with the second order of convergence:*

$$\left\|\widetilde{Y} - Y\right\|_{C(\overline{\omega_h})} = O(h_1^2 + h_2^2).$$

Thus, we need to solve SLAE (3) with block-tridiagonal matrix to obtain a numerical solution of boundary value problem (1).

3 SLAE Solver and Parallel Implementation

In our work, to solve SLAE (3), we apply the direct block elimination (or matrix sweep) method [4].

The algorithm is a follows. First, we perform a forward sweep and find supplementary matrices α_j and vectors β_j

$$\begin{aligned} &\alpha_1 = C_1^{-1}B_1, \quad \alpha_j = (C_j - A_j\alpha_j)^{-1}B_j, \quad j = 2, 3, ..., N-2, \\ &\beta_1 = C_1^{-1}F_1, \quad \beta_j = (C_j - A_j\alpha_j)^{-1}(F_j + A_j\beta_{j-1}), \quad j = 2, 3, ..., N-1. \end{aligned} \tag{5}$$

To find the solution, we perform a backward substitution pass

$$Y_{N-1} = \beta_{N-1}, \quad Y_j = \alpha_j Y_{j+1} + \beta_j, \quad j = (N-1), (N-2), ..., 2, 1. \tag{6}$$

The stability of this algorithm is established in [4].

The parallel algorithm is based on parallelization of vector-matrix operations using Intel Math Kernel Library [9] and cuBLAS Libraries [10, 11]. Computations by formulae (5)–(6) are bound to be sequential with respect to index j. Thus, the parallel algorithm may be based only on the optimization and parallelization of the matrix operations involved.

In work [8], we have implemented this algorithm for multicore central processors. This code utilizes the Intel MKL library for optimized parallel implementation of matrix routines, namely, 'gemv' and 'gemv' from the BLAS API for matrix-vector and matrix-matrix multiplications, and 'getrf' and 'getri' from the LAPACK API for the LU factorization and matrix inversion, respectively.

In the present work, we have implemented algorithm (5)–(6) for NVIDIA graphics processors using the CUDA technology. For basic matrix operations, we have used 'copy', 'axpy', 'gemv', and 'gemm' routines from the cuBLAS library. For matrix inversion, we adapted a combination of 'getrf' and 'getrs' implementations from the cuSOLVER library.

4 Numerical Experiments and Discussion

We have performed a series of numerical experiments to compare the performance of two implementations, CPU- and GPU-based one. For these experiments, we have used a workstation with a 16-core Intel i9-12900K processor and NVIDIA GeForce RTX 4090 GPU.

As a test problem, we consider an elliptic equation

$$\begin{aligned} &u_{x_2x_2}(x_1, x_2) + [(1 + x_1)u_{x_1}(x_1, x_2)]_{x_1} - (3 + x_1)u(x_1, x_2) = 0, \\ &0 < x_1 < 1, 0 < x_2 < 1, \end{aligned} \tag{7}$$

with boundary conditions

$$\begin{cases} u(0, x_2) = e^{x_2}, \quad 0 \le x_2 \le 1, \\ u(1, x_2) = e^{1+x_2}, \quad 0 \le x_2 \le 1, \\ u(x_1, 0) = e^{x_1}, \quad 0 \le x_1 \le 1, \\ u(x_1, 1) = e^{1+x_1}, \quad 0 \le x_1 \le 1. \end{cases} \tag{8}$$

The exact solution is given

$$u(x_1, x_2) = e^{x_1+x_2}, \ 0 \le x_1 \le 1, \ 0 \le x_2 \le 1.$$

A series of experiments was conducted for this problem with various grid sizes to compare the performance of the CPU and GPU implementations. The double precision number format was used in both implementation.

Table 1 contains the computing times for both implementations. In the fourth column, we list the speedup coefficient between the CPU and GPU codes. It also includes the values of the absolute error $\delta_\infty(h_1) = \left\| u(x_1, x_2) - \overline{u_{h_1}(x_1, x_2)} \right\|_\infty$ between exact and approximate solution $\overline{u_{h_1}(x_1, x_2)}$.

Table 1: Results of numerical experiments

Grid size $M \times N$	Computing time, CPU [seconds]	Computing time, GPU [seconds]	Speedup	Absolute error
258×514	1.06	1.22	0.87	$2.9 \cdot 10^{-4}$
514×514	7.65	3.33	2.29	$1.4 \cdot 10^{-4}$
1026×514	54.68	10.8	5.05	$7.2 \cdot 10^{-5}$

Let us discuss these results. In our parallel implementation, the performance is mainly driven by the optimization and parallelization of operations on matrix blocks. The size of these blocks is equal to number M. Number N defines the number of blocks, and the number of iterations in algorithm (5)–(6). So, this number does not affect the performance of parallel implementation.

So, we have used a large enough fixed N and varied number M. As the results demonstrate, at low matrix size, the times for CPU and GPU implementations are almost equal. The higher size we use, the more speedup we obtain for the GPU implementation in comparison with the CPU one. This is expected from the architecture of the graphics processors with numerous multiprocessor units, to achieve better efficiency at larger matrix size.

In the future, we plan to adapt and implement the parallel matrix sweep method [8] on graphics processors, taking advantage of both matrix operations for large size blocks and using CUDA streams to perform asynchroneous computations for decomposed sweep iterations, conducting multiple matrix operations simultaneously.

5 Conclusion

In this work, we have developed and implemented an efficient parallel algorithm for numerical solution of a boundary value problem for a two-dimensional elliptic differential equation. The algorithm consists in applying a finite difference scheme and the matrix sweep method to solve the resulting SLAE. We have developed a parallel code for NVIDIA graphics processors that implements this algorithm. The code is based on the CUDA technology and the cuBLAS and cuSOLVER libraries. We have conducted a series of numerical experiments to study the performance of our code in comparison with our earlier CPU code that utilizes the Intel MKL library. A speedup of 5 times was achieved.

In the future, we plan to adapt the parallel matrix sweep method to develop and implement more efficient parallel algorithm and use it as a part of algorithm for solving inverse problems for the considered elliptic equation.

Acknowledgment

Authors M. A. Sultanov, B. T. Sarsenov, and Y. Nurlanuly were financially supported by the Science Committee of the Ministry of Science and Higher Education of the Republic of Kazakhstan (Grant №AP23487362). Authors E. N. Akimova and V. E. Misilov received no external funding.

The Authors wish to thank the Reviewers for their careful reading of the manuscript and their fruitful comments and suggestions.

References

[1] O. A. Ladyzhenskaya and N. N. Ural'tseva. Linear and Quasilinear Elliptic Equations. Academic Press. 1968.

[2] D. Gilbarg and N. S. Trudinger. Elliptic partial differential equations of second order, Berlin, Heidelberg, New York and Tokyo: Springer. 1983.

[3] A. A. Samarskii and A. V. Gulin. Stability of Difference Schemes. Nauka: Moscow, 1973) [in Russian].

[4] A. Samarskii and E. Nikolaev. Numerical Methods for Grid Equations, Volume I: Direct Methods. Birkhäuser: Basel, Switzerland, 1989.

[5] J. C. Strikwerda. Finite difference schemes and partial differential equations. SIAM. 2004.

[6] V. P. Il'In. *Finite difference and finite volume methods for elliptic equations [In Russian].* 2001.

[7] M. A. Sultanov, E. N. Akimova, V. E. Misilov, and Y. Nurlanuly. Parallel direct and iterative methods for solving the time-fractional diffusion equation on multicore processors // Mathematics. 2022, Vol. 10, Iss. 3. Art. no. 323. DOI: 10.3390/math10030323

[8] E. N. Akimova, M. A. Sultanov, V. E. Misilov, and Y. Nurlanuly. Parallel algorithm for solving the inverse two-dimensional fractional diffusion problem of identifying the source term // Fractal and Fractional. 2023, Vol. 7, Iss. 11. Art. no. 801. DOI: 10.3390/fractalfract7110801

[9] Intel Corporation. oneAPI Math Kernel Library. `https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html`

[10] NVIDIA Corporation. cuBLAS: Basic Linear Algebra on NVIDIA GPUs. `https://developer.nvidia.com/cublas`

[11] NVIDIA Corporation. cuSOLVER: Direct Linear Solvers on NVIDIA GPUs. `https://developer.nvidia.com/cusolver`