Least-squares space-time formulation for advection-diffusion problem with efficient adaptive solver based on matrix compression

Marcin Łoś[1] [0000−0002−8426−6345], Paulina Sepúlveda[2] [0000−0002−7146−2240], Mateusz Dobija[3,4] [0000−0003−4557−3534], and Anna Paszyńska[3] [0000−0002−0716−0619]

[1] Institute of Computer Science, AGH University of Science and Technology, Al. Mickiewicza 30, Kraków, Poland
los@agh.edu.pl

[2] Instituto de Matemáticas, Pontificia Universidad Católica de Valparaíso, Casilla 4059, Valparaiso, Chile
paulina.sepulveda@pucv.cl

[3] Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, ul. prof. Stanisława Łojasiewicza 11, Kraków, Poland
anna.paszynska@uj.edu.pl
mateusz.dobija@doctoral.uj.edu.pl

[4] Doctoral School of Exact and Natural Sciences, Jagiellonian University

Abstract. We present the hierarchical matrix compression algorithms to speed up the computations to solve unstable space-time finite element method. Namely, we focus on the non-stationary time-dependent advection dominated diffusion problem solved by using space-time finite element method. We formulate the problem on the space-time mesh, where two axes of coordinates system denote the spatial dimension, and the third axis denotes the temporal dimension. By employing the space-time mesh, we avoid time iterations, and we solve the problem "at once" by calling a solver once for the entire mesh. This problem, however, is challenging, and it requires the application of special stabilization methods. We propose the stabilization method based on least-squares. We derive the space-time formulation, and solve it using adaptive finite element method. To speed up the solution process, we compress the matrix of the space-time formulation using the low-rank compression algorithm. We show that the compressed matrix allows for quasi-linear computational cost matrix-vector multiplication. Thus, we apply the GMRES solver with hierarchical matrix-vector multiplications. Summing up, we propose a quasi-linear computational cost solver for stabilized space-time formulations of advection dominated diffusion problem.

Keywords: Finite element method · Space-time formulation · Isogeometric analysis · H-matrices · Matrix compression · SVD.

1 Introduction

With increasing supercomputer power, the space-time finite element method is becoming more and more popular. The method employs an n-dimensional space-time mesh, with $n-1$ axes corresponding to the spatial dimension, and one axis corresponding to the temporal dimension. One of the advantages of the method is the fact that we can refine the computational mesh in space-time domain. The space-time formulation does not process a sequence of computational meshes from consecutive time moments. We formulate and solve the problem on one big mesh, and we can simultaneously refine the mesh to improve the quality of the solution in space and time.

The problem of developing stabilization methods for space-time finite element is a very important scientific topic nowadays. There are several attempts do develop stabilized FEM solver. Different methods have been employed for this purpose. Paper [1] employs space-time stabilized formulation using an adaptive constrained first-order system with the least squares method. Another space-time discretization for the constrained first-order system least square method (CFOSLS) are discussed in [2]. It is also possible to employ Discontinuous Petrov-Galerkin method for the stabilization of the space-time formulation, as it is illustrated for the Schrödinger equation in [5] and for the acoustic wave propagation in [6, 7]. The least-square finite element method has been applied for the stabilization of the parabolic problem in [8].

The most crucial aspect when developing the space time formulations is the computational cost of the solver [10]. Paper [3] summarizes different fast solvers for space-time formulations. In [4] the authors discuss the applications of the algebraic multigrid solvers for an adaptive space-time finite-element discretization in 3D and 4D.

The hierarchical matrices have been introduced by Hackbush [11, 12]. They employ the low-rank compression of matrix blocks to speed up the solution process.

In this paper, we present the hierarchical matrix compression algorithms to speed up the computations of difficult, unstable space-time finite element method together with the stabilization method. The described approach is used to solve the stabilized space-time formulations of advection dominated diffusion problem with quasi-linear computational time.

The novelties of our paper are:

- We consider advection-dominated diffusion transient problem formulated in space-time finite elements with the stabilization based on the least squares method.
- We employ adaptive finite element algorithm implemented in FeniCS library refining the elements in the space-time domain.
- We introduce the idea of the hierarchical matrices for the space-time formulation. We generate and compress the matrices resulting from the finite element method discretization using the truncated SVD algorithm applied for blocks of the global matrix.

– We show that the hierarchical matrices can be processed by the GMRES iterative solver one order of magnitude faster than regular matrices.

2 Model problem

As a model problem, we study the advection-diffusion equation over a domain:

$$\begin{aligned}&\partial_t \phi = \varepsilon \Delta\phi - \beta \cdot \nabla u + f,\\ &\phi(x,0) = u_0 \quad \textit{for } x \in \Omega,\\ &\phi(x,t) = 0 \quad \textit{for } (x,t) \in \partial\Omega \times [0,T].\end{aligned}$$

For small $\varepsilon / \|\beta\|$, the problem is advection-dominated and the standard Galerkin method encounters stability issues.

3 Space-time formulation

The space-time formulation we employ is a first-order formulation based on the idea of a constrained least squares problem (CFOSLS), and has been first introduced in [2].

3.1 First-order formulation

Let $\Omega_T = \Omega \times (0,T)$ denote the space-time domain, and $\Gamma_S = \partial\Omega \times (0,T)$ denote the spatial boundary. We start by writing the equation in the divergence form

$$\partial_t \phi + \operatorname{div}_{\mathrm{x}} \underbrace{(-\varepsilon \nabla \phi + \beta \phi)}_{\mathcal{L}\phi} = f, \tag{1}$$

which allows us to reformulate it as

$$\operatorname{div}_{\mathrm{x,t}} \underline{\boldsymbol{\sigma}} = f, \tag{2}$$

where $\underline{\boldsymbol{\sigma}} = (\mathcal{L}\phi, \phi)$ and $\mathcal{L}\phi = (\mathcal{L}_x\phi, \mathcal{L}_y\phi)$, and $\operatorname{div}_{\mathrm{x,t}}$ denotes the full space-time divergence operator (as opposed to the spatial divergence $\operatorname{div}_{\mathrm{x}}$). Introducing $\underline{\boldsymbol{\sigma}}$ as a new unknown, we can rewrite our equation as a first-order system

$$\begin{cases} \operatorname{div}_{\mathrm{x,t}} \underline{\boldsymbol{\sigma}} = f, \\ \underline{\boldsymbol{\sigma}} - \begin{bmatrix} \mathcal{L}\phi \\ \phi \end{bmatrix} = 0, \end{cases} \tag{3}$$

where $\underline{\boldsymbol{\sigma}} \in R = H(\operatorname{div}, \Omega_T)$, $\phi \in V = \{v \in H^1(\Omega_T) : v|_{\Gamma_S} = 0\}$.

For convenience, let us separate components of $\underline{\boldsymbol{\sigma}}$ as $\underline{\boldsymbol{\sigma}} = (\boldsymbol{\sigma}, \sigma_*)$, where σ_* is a scalar function.

3.2 Variational formulation

Let

$$J(\underline{\boldsymbol{\sigma}},\phi)=\frac{1}{2}\left\|\underline{\boldsymbol{\sigma}}-\begin{bmatrix}\mathcal{L}\phi\\ \phi\end{bmatrix}\right\|^2=\frac{1}{2}\left\|\boldsymbol{\sigma}-\mathcal{L}\phi\right\|^2. \tag{4}$$

The solution of the system (3) is also a solution of the following minimization problem:

$$\min_{(\underline{\boldsymbol{\tau}},\omega)\in R\times V} J(\underline{\boldsymbol{\tau}},\omega) \quad \text{subject to } \operatorname{div}_{\mathrm{x,t}}\underline{\boldsymbol{\tau}}=f, \tag{5}$$

since $J(\underline{\boldsymbol{\sigma}},\phi)=0$. Applying the Lagrange multipliers method to this constrained minimization problem, we search for the critical points of the functional

$$G(\underline{\boldsymbol{\sigma}},\phi,\lambda)=\frac{1}{2}\left\|\boldsymbol{\sigma}-\mathcal{L}\phi\right\|^2+\frac{1}{2}\left\|\sigma_*-\phi\right\|^2+(\operatorname{div}_{\mathrm{x,t}}\underline{\boldsymbol{\sigma}}-f,\lambda), \tag{6}$$

where $\lambda\in L^2(\Omega_T)$ is the Lagrange multiplier. In this formulation, we need to find $(\underline{\boldsymbol{\sigma}},\ \phi,\ \lambda)\in\mathbf{W}$ such that for all $(\underline{\boldsymbol{\tau}},\ \omega,\ \mu)\in\mathbf{W}$

$$\begin{aligned}&(\boldsymbol{\sigma}-\mathcal{L}\phi,\boldsymbol{\tau}-\mathcal{L}\omega)+(\sigma_*-\phi,\tau_*-\omega)\\&+(\operatorname{div}_{\mathrm{x,t}}\underline{\boldsymbol{\tau}},\lambda)+(\operatorname{div}_{\mathrm{x,t}}\underline{\boldsymbol{\sigma}}-f,\mu)=0.\end{aligned} \tag{7}$$

where $\mathbf{W}=R\times V\times L^2(\Omega_T)$. To make the structure of the resulting discrete matrix more apparent, we can rewrite the above equation as

$$\begin{aligned}
(\phi,\omega)+(\mathcal{L}\phi,\mathcal{L}\omega)-(\sigma_*,\omega)\quad-(\boldsymbol{\sigma},\mathcal{L}\omega)\qquad\qquad\qquad &=0,\\
-(\phi,\tau_*)\quad+(\sigma_*,\tau_*)\qquad\qquad\qquad+(\lambda,\partial_t\tau_*)\quad &=0,\\
-(\mathcal{L}\phi,\boldsymbol{\tau})\qquad\qquad\qquad+(\boldsymbol{\sigma},\boldsymbol{\tau})\quad+(\lambda,\operatorname{div}_{\mathrm{x}}\boldsymbol{\tau}) &=0,\\
+(\partial_t\sigma_*,\mu)+(\operatorname{div}_{\mathrm{x}}\boldsymbol{\sigma},\mu)\qquad\qquad\qquad &=(f,\mu)\,.
\end{aligned}$$

3.3 Discrete problem

We approximate the independent variables ϕ,σ_x, and σ_y using quadratic B-splines. Let us define $\{u_i\}$ the B-spline basis functions and $\phi^{\mathrm{h}},\sigma_*^{\mathrm{h}},\sigma_x^{\mathrm{h}},\sigma_y^{\mathrm{h}},\lambda_x^{\mathrm{h}}$ the corresponding vectors of coefficients of the B-spline expansion of ϕ,σ_x,σ_x, respectively. We can write the system in the following matrix structure

$$\begin{bmatrix}M+K & -M & -L_x^T & -L_y^T & 0\\ -M & M & 0 & 0 & A_t^T\\ -L_x & 0 & M & 0 & A_x^T\\ -L_y & 0 & 0 & M & A_y^T\\ 0 & A_t & A_x & A_y & 0\end{bmatrix}\begin{bmatrix}\phi^h\\ \sigma_*^h\\ \sigma_x^h\\ \sigma_y^h\\ \lambda^h\end{bmatrix}=\begin{bmatrix}0\\0\\0\\0\\f^h\end{bmatrix}$$

where M represents the mass matrix such that $(M)_{ij}:=(u_i,u_j)_{L^2}$, (A_γ), is such that $(A_\gamma)_{ij}=(\partial_\gamma u_i,u_j)$ and L_γ such that $(L_\gamma)_{ij}:=(\mathcal{L}_\gamma u_i,u_j)$. Moreover, f^{h} is also the vector of coefficients related to the expansion of f in the B-spline basis.

4 Matrix compression

The core of the low-rank matrix compression is the SVD algorithm, illustrated in Figure 1. The matrix A is decomposed into UDV, namely the matrix of "columns" U, the diagonal matrix of singular values D, and the matrix of "rows" V.

$$A = UDV, \quad [U,D,V] = SVD(B),$$
$$U \in \mathcal{M}^{n\times n}, D - \text{diagonal } m\times n, V \in \mathcal{M}^{m\times m}.$$

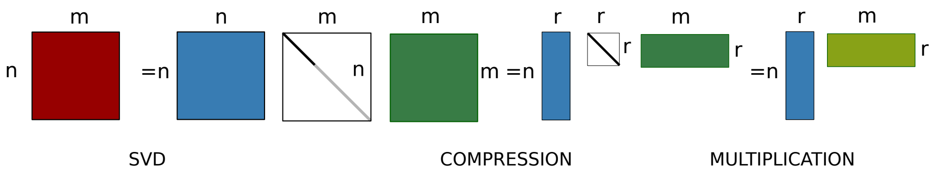

Fig. 1. SVD algorithm for the low-rank matrix compression.

The entries of D (singular values) are sorted in descending order. The diagonal values less than the compression threshold δ are removed together with corresponding columns of U and rows of V. The entries of D (singular values) are sorted in descending order. The diagonal values less than the compression threshold δ are removed together with corresponding columns of U and rows of V. As the result we obtain the low-rank compressed matrix $\mathcal{H}_\mathcal{A}$, where $s = rank$ $\mathcal{H}_\mathcal{A} = \max\{\text{i} : d_{ii} > \delta\}$. The matrix $\mathcal{H}_\mathcal{A}$ is the best approximation of A in the Frobenious norm among all the matrices of rank s.

The compression is performed in a recursive way, as expressed by Algorithms 1 and 2. We partition the matrix recursively into blocks, we check how many singular values are larger than the prescribed δ. If the compression of the block with δ results in viewer singular values than the prescribed threshold b, we stop the recursion and store the sub-matrix in a compressed way. Otherwise, we continue with the recursive partitions.

Standard LAPACK subroutine dgesvd for the SVD computations has time complexity $\mathcal{O}(N^3)$ for a square matrix. We, however, employ the truncated SVD that computes only r singular values, having the complexity of $\mathcal{O}(N^2r)$. Thus, the compression of the space-time matrix has a time complexity of a similar order as the matrix r vectors multiplication.

The compression of the space-time matrix results in a structure presented in Figure 2.

5 Compressed matrix-vector multiplication

We illustrate in Figure 3 the process of multiplication of a compressed matrix by s vectors. There are two cases to consider.

Algorithm 1 compress_matrix

```
Require: $A \in \mathcal{M}^{m \times n}$, $\delta$ compression threshold, $b$ maximum rank
 1: if $A = 0$ then
 2:     create new node $v$; $v.rank \leftarrow 0$; $v.size \leftarrow size(A)$; return $v$;
 3: end if
 4: $[U, D, V] \leftarrow SVD(A)$; $\sigma \leftarrow diag(D)$;
 5: $rank \leftarrow card\left(\{i \colon \sigma_i > \delta\}\right)$;
 6: if $rank < b$ then
 7:     create new node $v$; $v.rank \leftarrow rank$;
 8:     $v.singularvalues \leftarrow \sigma(1 : rank)$;
 9:     $v.U \leftarrow U(*, 1 : rank)$;
10:     $v.V \leftarrow D(1 : rank, 1 : rank) * V(1 : rank, *)$;
11:     $v.sons \leftarrow \emptyset$; $v.size \leftarrow size(A)$;
12:     return $v$;
13: else
14:     return $process\_matrix(A, \delta, b)$;
15: end if
```

Algorithm 2 process_matrix

```
Require: $A \in \mathcal{M}^{m \times n}$, $\delta$ compression threshold, $b$ maximum rank
 1: $v \leftarrow create\_node()$
 2: $A_{11} \leftarrow A(1 : \frac{m}{2}, 1 : \frac{n}{2})$
 3: $A_{12} \leftarrow A(1 : \frac{m}{2}, \frac{n}{2} + 1 : n)$
 4: $A_{21} \leftarrow A(\frac{m}{2} + 1 : m, 1 : \frac{n}{2})$
 5: $A_{22} \leftarrow A(\frac{m}{2} + 1 : m, \frac{n}{2} : n)$
 6: $n_1 \leftarrow$ compress_matrix$(A_{11}, \delta, b)$
 7: $n_2 \leftarrow$ compress_matrix$(A_{12}, \delta, b)$
 8: $n_3 \leftarrow$ compress_matrix$(A_{21}, \delta, b)$
 9: $n_4 \leftarrow$ compress_matrix$(A_{22}, \delta, b)$
10: $v.sons \leftarrow [n_1, n_2, n_3, n_4]$
11: return $v$
```

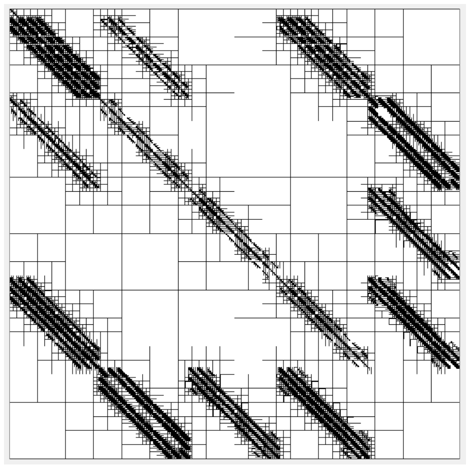

Fig. 2. Exemplary compressed matrix

– The first case is located at the leaves of the compressed matrix, where we multiply the compressed submatrix by a corresponding part of the vector. This is illustrated on the left panel in Figure 3. In this case, the computational cost of matrix-vector multiplication with compressed matrix and s vectors is $\mathcal{O}(rms + rns)$, when $n = m = N \gg r$ it reduces to $\mathcal{O}(Nrs)$

– The second case is related to the multiplication of a matrix compressed into four SVD blocks by the vector partitioned into two blocks. This is illustrated on the right panel in Figure 3. We employ the recursive formula $\begin{bmatrix} C_2 * (C_1 * X_1) + D_2 * (D_1 * X_2) \\ E_2 * (E_1 * X_1) + F_2 * (F_1 * X_2) \end{bmatrix}$. The computational cost is $\mathcal{O}(Nrs)$.

The pseudocode is illustrated in Algorithm 3.

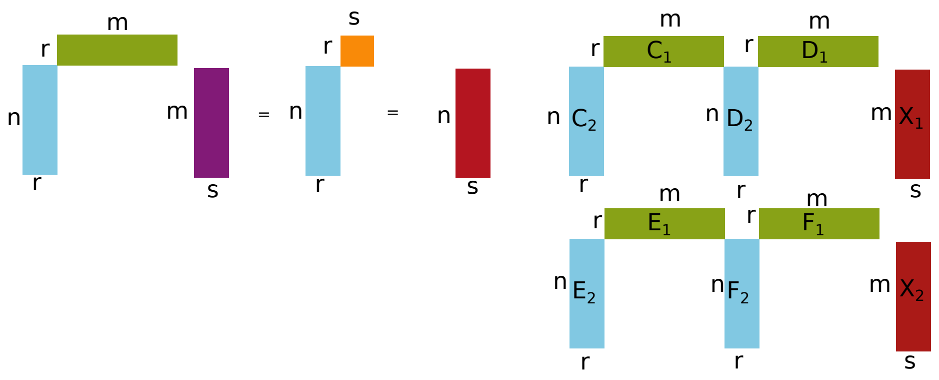

Fig. 3. SVD-compressed matrix multiplication

Algorithm 3 matrix_vectors_multiply

```
Require: node $v$, Compressed matrix $A(v) \in \mathcal{M}^{m\times n}$, $Y \in \mathcal{M}^{n\times c}$ vectors to multiply
1: if $v.sons = \emptyset$ then
2:     if $v.rank = 0$ then
3:         return $zeros(size(A).rows)$;
4:     end if
5:     return $v.U * (v.V * X)$;
6: end if
7: $rows = size(X).rows$;
8: $X_1 = X(1 : \frac{rows}{2}, *)$; $X_2 = X(\frac{rows}{2} + 1 : size(A).rows, *)$;
9: $C_1 = v.son(1).U; C_2 = v.son(1).V$;
10: $D_1 = v.son(2).U; D_2 = v.son(2).V$;
11: $E_1 = v.son(3).U; E_2 = v.son(3).V$;
12: $F_1 = v.son(4).U; F_2 = v.son(4).V$;
13: return $\begin{bmatrix} C_2 * (C_1 * X_1) + D_2 * (D_1 * X_2) \\ E_2 * (E_1 * X_1) + F_2 * (F_1 * X_2) \end{bmatrix}$;
```

The critical from the point of view of the computational cost is the structure of the compressed matrix. If we have the structure of the matrix as presented in Figure 4, we have the quasi-linear multiplication cost. Namely, at each level, we have 2 leaves and 2 interior nodes

$$C(N) = \underbrace{2C(N/2) + 2\mathcal{O}(Nrs/2)}_{\text{multiplication}} + \underbrace{\mathcal{O}(N)}_{\text{addition}}, \quad C(N_0) = \mathcal{O}(N_0 rs)$$
$$\Rightarrow C(N) = \mathcal{O}(N \log N)$$

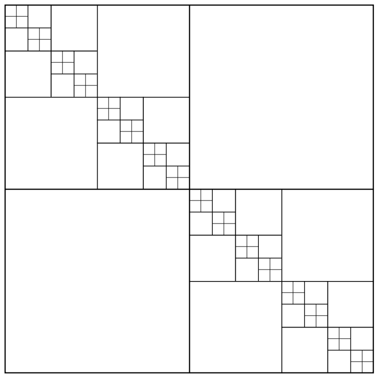

Fig. 4. Optimal structure of the compressed matrix

Fortunatelly, the structure of the space-time problem matrix has this optimal shape in several sub-blocks, see Figure 2.

6 GMRES solver

We employ the GMRES iterative solver with hierarchical matrix-vector multiplication, illustrated in Algorithm 4.

Algorithm 4 Pseudo-code of the GMRES algorithm

```
Require: $\mathcal{H}_A$ compressed matrix, $b$ right-hand-side vector, $x_0$ starting point
    Compute $r_0 = b - \mathcal{H}_A x_0$
    Compute $v_1 = \frac{r_0}{\|r_0\|}$
    for $j = 1, 2, ..., k$
    Compute $h_{i,j} = (\mathcal{H}_A v_j, v_i)$ for $i = 1, 2, ..., j$
    Compute $\hat{v}_{j+1} = \mathcal{H}_A v_j - \sum_{i=1,...,j} h_{i,j} v_i$
    Compute $h_{j+1,j} = \|\hat{v}_{j+1}\|_2$
    Compute $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
    end for
    Form solution $x_k = x_0 + V_k y_k$ where $V_k = [v_1 ... v_k]$, and $y_k$ minimizes $J(y) = \|\beta e_1 - \hat{H}_k y\|$ where $\hat{H} = \begin{bmatrix} h_{1,1} & h_{1,2} \cdots h_{1,k} & & \\ h_{2,1} & h_{2,2} \cdots h_{2,k} & & \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & h_{k,k-1} & h_{k,k} \\ 0 & \cdots & 0 & h_{k+1,k} \end{bmatrix}$
```

7 Numerical results

For the numerical tests, we use the model problem (2) on a regular domain $\Omega \times (0,T) = (0,1)^3$ with $\beta = (0, 0.3)$, $\varepsilon = 10^{-5}$, no forcing ($f = 0$), and the initial state u_0 given by $u_0(x) = \psi(10\|x - c\|)$, where $c = (0.5, 0.5)$ and

$$\psi(r) = \begin{cases} (1-r^2)^2 & \text{for } r \leq 1, \\ 0 & \text{for } r > 1. \end{cases}$$

As a result, the initial state is zero except for a small region in the center of the domain. Tests were performed using basis functions of degree $p = 1$ for all the discrete spaces.

Starting with the coarse initial mesh, we perform adaptive mesh refinements using the value of $J(\underline{\boldsymbol{\sigma}}_h, \phi_h)$ as the error indicator and the Dörfler marking criterion [9] with $\theta = 0.5$. The improvement of the solutions at particular adaptive iterations is denoted in Figure 6. Finally, the sequence of generated adaptive space-time meshes is presented in Figure 7. We also present in Table 1 the convergence of the adaptive space-time finite element method, as well as the execution times for the solver without the compression, and after the compression, for a sequence of adaptive grids. Figure 5 presents execution times for the solver without the compression, and after the compression, for a sequence of adaptive grids.

Table 1. Results of the adaptive mesh refinement process

level	DoFs	$J(\boldsymbol{\sigma} U_h, \phi_h)$	solver for A [s]	solver for $\mathcal{H}_A$ [s]
0	3019	0.0137	0.046	0.005
1	11540	0.0039	0.567	0.076
2	25963	0.0052	4.280	0.774
3	63033	0.0040	14.573	2.573
4	163755	0.0031	31.190	5.048
5	359658	0.0023	101.335	20.120
6	730953	0.0017	279.838	45.606

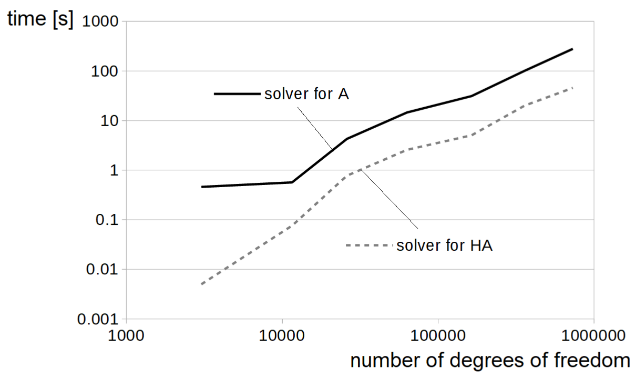

Fig. 5. Execution times for solver without the compression, and after the compression, for a sequence of adaptive grid.

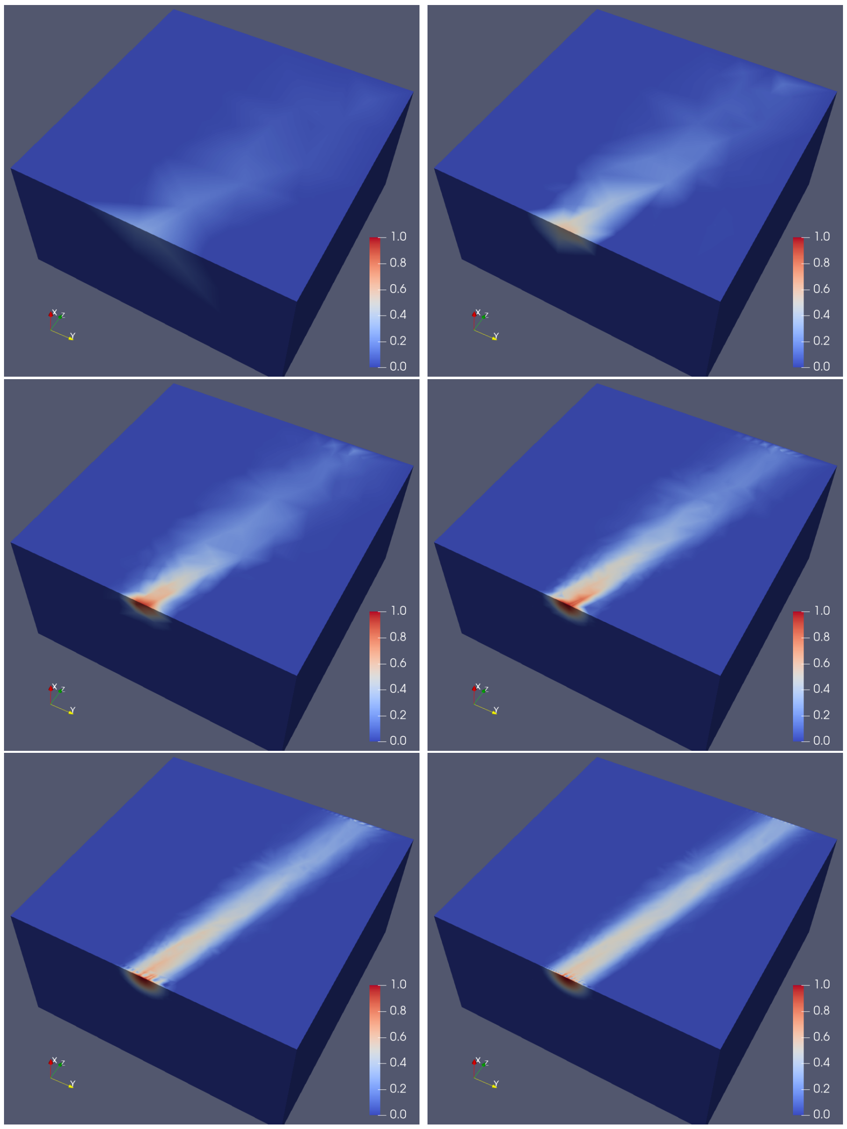

Fig. 6. Solutions in the six consecutive refinement steps

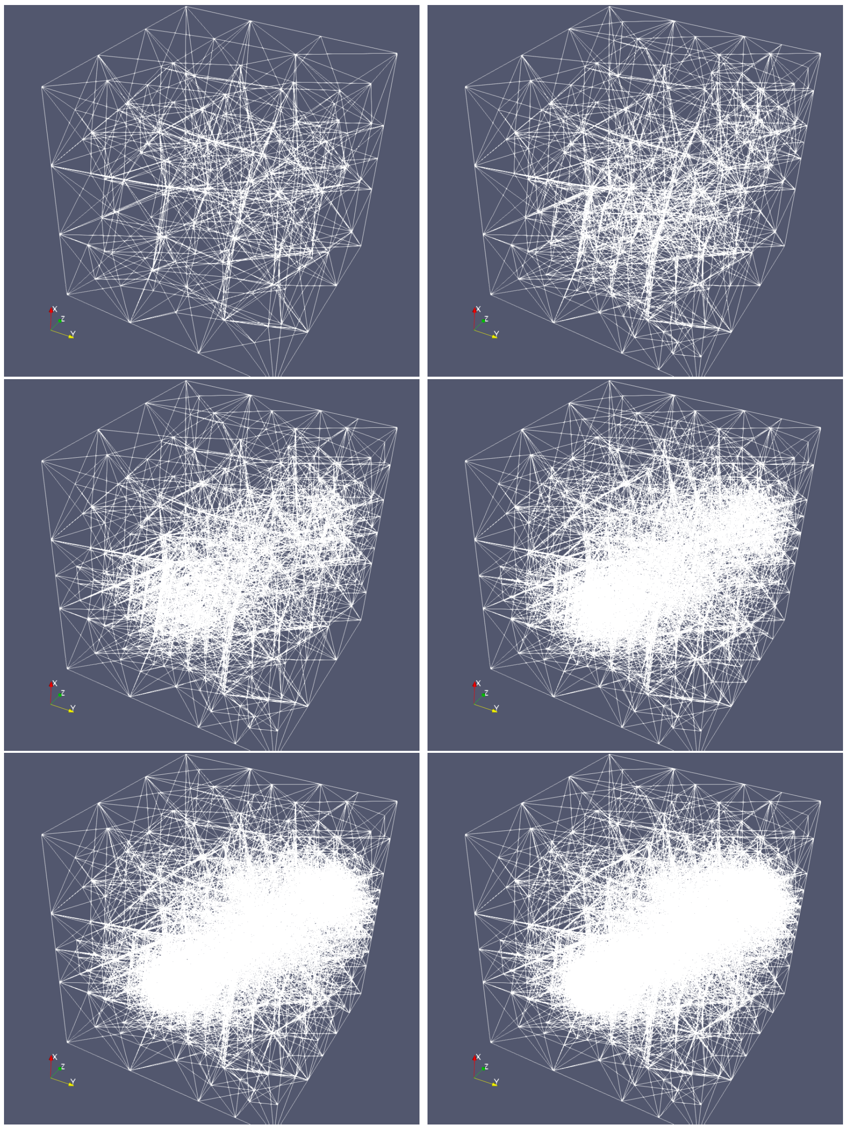

Fig. 7. Mesh in the six consecutive refinement steps

8 Conclusions

In this paper, we considered the space-time formulation of the advection dominated diffusion problem. The stabilization was obtained by introducing the constrained minimization problem with Lagrange multipliers. The obtained system of linear equations was discretized with B-spline basis functions. Next, we employed the low-rank compression of the discrete form of the space-time matrix. The obtained hierarchical form of the space-time matrix has several blocks that has been compressed into a hierarchical "diagonal" form. The hierarchical matrix can be multiplied by the vector in a $\mathcal{O}(NlogN)$ computational cost. This allows to construct a GMRES solver with hierarchical matrices that is one order of magnitude faster than the GMRES solver with the original sparse matrix. Future work may include research on a proper perconditioner for the GMRES solver. For example, we may investigate if the deep compression of the matrix, with a large compression threshold, can be applied as an efficient preconditioner for the iterative solver.

Acknowledgements Research project supported by the program “Excellence initiative - research university" for the AGH University of Science and Technology.

References

1. Schafelner, A., Vassilevski, P. S.:Numerical results for adaptive (negative norm) constrained first order system least squares formulations. In: Recent Advances in Least-Squares and Discontinuous Petrov–Galerkin Finite Element Methods, Computers & Mathematics with Applications 95, 256–270, (2021)
2. Voronin, K., Lee, C. S., Neumuller, M., Sepulveda, P., Vassilevski, P. S.: Space-time discretizations using constrained first-order system least squares (CFOSLS), Journal of Computational Physics **373** 863–876 (2018)
3. Neumuller, M.: Space-time methods: Fast solvers and applications, Ph.D. thesis, Graz University of Technology, Institute of Applied Mathematics (2013).
4. Steinbach, O., Yang, H.: Comparison of algebraic multigrid methods for an adaptive space–time finite-element discretization of the heat equation in 3d and 4d, Numerical Linear Algebra with Applications **25** (3) e2143 (2018)
5. Demkowicz, L., Gopalakrishnan, J., Nagaraj, S., Sepulveda, P.: A spacetime DPG method for the Schrodinger equation, SIAM Journal on Numerical Analysis **55** (4) 1740–1759 (2017)
6. Gopalakrishnan, J. , Sepulvedas, P.: A space-time DPG method for the wave equation in multiple dimensions, De Gruyter, Berlin, Boston, Chapter 4 117–140 (2019)
7. Ernesti, J., Wieners, C.: A space-time discontinuous Petrov–Galerkin method for acoustic waves, De Gruyter, Berlin, Boston, Chapter 3 89–116 (2019)
8. Fuhrer, T., Karkulik, M.:Space-time least-squares finite elements for parabolic equations, Computers & Mathematics with Applications 9281 27–36 (2021)
9. Dorfler,W.: A convergent adaptive algorithm for Poisson's equation, SIAM Journal on Numerical Analysis **33** (3) 1106–1124 (1996)

10. Skotniczny, M., A. Paszyńska, A., Rojas, S., Paszyński, M.:Complexity of direct and iterative solvers on space-time formulations versus time–marching schemes for h-refined grids towards singularities arXiv:2110.05804 1–36 (2022)
11. Hackbusch. W.: A sparse matrix arithmetic based on H-matrices. Part I: introduction to H -matrices. Computing 62 89–108 (1999)
12. Hackbusch. W.: Hierarchical matrices: algorithms and analysis. Springer (2015)