Research Paper
Computational Methods for Differential Equations
http://cmde.tabrizu.ac.ir
Vol. 14, No. 3, 2026, pp. 1426-1441
DOI:10.22034/cmde.2025.65520.3017

A multigrid solver for subdiffusion equations

Mohadese Ramezani[1], **Reza Mokhtari**[1,2,*] **and Gundolf Haase**[1,2]
[1]Department of Mathematical Sciences, Isfahan University of Technology, Isfahan 84156-83111, Iran.
[2]Department for Mathematics and Scientific Computing, University of Graz, Graz, Austria.

Abstract

In this paper, we investigate the S3-FD method for solving time-fractional diffusion equations in both one-dimensional (1D) and two-dimensional (2D) spatial domains, achieving high-order temporal accuracy. We leverage the S3 formula, which has a temporal accuracy of $4-\alpha$, to approximate the Caputo fractional derivative of order $\alpha \in (0,1)$, and we employ the finite difference approach for spatial discretization. We develop a fully discrete scheme for both uniform and non-uniform spatial meshes. Our analysis begins with the 1D subdiffusion problem, where we employ the cyclic reduction method alongside OpenMP-based parallel programming to reduce computational costs. Leveraging this groundwork, we extend our technique to the 2D subdiffusion problem using a multigrid method and domain decomposition strategy paired with MPI programming. This innovative method yields an impressive temporal convergence order of $\mathcal{O}(\Delta t^{4-\alpha})$. The performance and efficiency of the proposed S3-FD algorithm are demonstrated through numerical experiments, highlighting its potential for large-scale fractional diffusion problems.

Keywords. S3 formula, Subdiffusion equation, Multigrid method, Domain decomposition, Parallel processing.
2010 Mathematics Subject Classification. 65F10, 65M55, 35R11.

1. Introduction

The following time-fractional diffusion problem is considered, with $0 < \alpha < 1$,

$$\begin{aligned} &D_t^\alpha u(\boldsymbol{x},t) - \Delta u(\boldsymbol{x},t) = f(\boldsymbol{x},t), && (\boldsymbol{x},t) \in \Omega \times (0,T], \\ &u(\boldsymbol{x},t) = g(\boldsymbol{x},t), && (\boldsymbol{x},t) \in \partial\Omega \times [0,T], \\ &u(\boldsymbol{x},0) = u_0(\boldsymbol{x}), && \boldsymbol{x} \in \Omega, \end{aligned} \tag{1.1}$$

where $\Omega \subset \mathbb{R}^d$ with $d = 1, 2$ and α is the anomalous diffusion exponent. D_t^α stands for the Caputo fractional derivative of order $\alpha \in (0,1)$ defined by

$$D_t^\alpha u(\cdot,t) = \frac{1}{\Gamma(1-\alpha)} \int_0^t \frac{\partial u(\cdot,s)}{\partial s} \, (t-s)^{-\alpha} \, \mathrm{d}s. \tag{1.2}$$

Several fully discrete schemes have been proposed in recent years for solving fractional partial differential equations (FPDEs) to improve accuracy in both time and space. Popular numerical methods for discretizing the time-fractional diffusion equation in space include finite difference methods (FDMs), finite element methods (FEMs), discontinuous Galerkin methods (DGMs), etc (see e.g. [7, 20, 21, 28] and references therein).

Many researchers have utilized piecewise interpolation methods to discretize the Caputo fractional derivative. The most widely used algorithm is the L1 formula, which is based on piecewise linear interpolation and has an error estimate of $\mathcal{O}(\Delta t^{2-\alpha})$ [10, 11]. Most methods proposed for solving time FPDEs with Caputo derivatives are based on the L1 formula. Zhao et al. [29] developed a technique for solving multi-term time-fractional diffusion equations using the finite element method in space and the L1 approximation in time. The modified L1 formula, developed by

Received: 18 January 2025; Accepted: 05 May 2025.
∗ Corresponding author. Email: mokhtari@iut.ac.ir.

Yan et al. [27], is used for solving time FPDEs with non-smooth data. Employing the L1 formula, Liu et al. [12] proposed a quadratic spline collocation method to solve fractional sub-diffusion equations with artificial boundary conditions. Gao et al. [4] and Alikhanov [1] introduced the L1-2 and L2-1$_\sigma$ approximations for the Caputo fractional derivative using quadratic interpolations, with an accuracy of order $\mathcal{O}(\Delta t^{3-\alpha})$. Due to the optimal error estimate of the L2-1$_\sigma$ formula, we could refer to [25] that has second-order convergence in solving fractional PDEs. Mokhtari et al. [14] established a high-order formula, called L1-2-3, for approximating the Caputo derivative. The stability and convergence analyses of FDM and FEM, along with L-type formulae (L1, L1-2, L1-2-3 formulae), were established in [15, 19] in solving fractional diffusion problems. Considering the global accuracy in solving subdiffusion equations, a new class of formulae based on B-spline interpolation, called S2 and S3, were constructed to approximate the Caputo derivative with $\mathcal{O}(\Delta t^{3-\alpha})$ and $\mathcal{O}(\Delta t^{4-\alpha})$ accuracy for $\alpha \in (0, 2)$ [18]. The stability and convergence analysis of the FDM based on the S2 formula in solving fractional subdiffusion equations was derived in [17], considering smooth and nonsmooth solutions. More recently, a second-order method has been studied [8] for solving generalized time-fractional diffusion equations with smooth solutions.

The fractional operators are nonlocal, often leading to large systems using discretization methods. Efficiently solving these systems using direct solvers like Gaussian elimination presents significant challenges. A robust space-time multigrid method based on the waveform relaxation technique was introduced in [5]. Tan et al. [24] developed an efficient numerical solver for anisotropic subdiffusion problems by using the L1 formula combined with collocation methods for time and space discretizations, respectively. In [16], a multigrid method for solving space-fractional diffusion equations with variable coefficients was extended. Additionally, Xu et al. [26] implemented an effective multigrid method for two-dimensional anisotropic fractional diffusion equations. Jiang and Bai [6] explored multigrid methods for time-fractional conservation laws. Zhou et al. [30] proposed a splitting characteristic finite difference domain decomposition scheme for nonlinear time-fractional MIM advection-diffusion equations, which enhances computational efficiency and scalability. A reduced conjugate gradient basis method for fractional diffusion was developed in [9], which improves computational efficiency by minimizing the solution space. In [23], a fast second-order numerical method for space-fractional advection-diffusion equations was introduced, utilizing a preconditioning iterative approach. Recently, Gan et al. [3] presented a preconditioning strategy to efficiently manage linear systems arising from high-order accurate schemes in time-fractional diffusion equations, significantly reducing computational complexity.

This paper proposes an efficient numerical method for fractional subdiffusion equations using the S3 formula for the Caputo derivative, achieving $\mathcal{O}(\Delta t^{4-\alpha})$ temporal accuracy, with finite difference spatial discretization on uniform and non-uniform meshes. One of the key challenges in implementing the S3 formula is its computational cost. In contrast to the S2 method, which can be implemented using a time-marching approach [17], the S3 method requires the construction and solution of a fully coupled, all-at-once system. This leads to a dense matrix formulation that incorporates both time and spatial discretizations, resulting in computational complexities on the order of $\mathcal{O}((mn)^2)$. Here, n denotes the total number of time steps, while m indicates the number of spatial discretization points. This formulation not only increases memory usage but also limits the scalability of direct solvers, making traditional methods impractical for large-scale problems. Our goal is to develop an efficient implementation that preserves the advantages of the S3 method while mitigating its computational costs. To tackle these challenges, we employ fast solvers, such as cyclic reduction and multigrid methods, which reduce the computational complexity to $\mathcal{O}(m \log(m) n^2)$. Additionally, we use advanced strategies to enhance solver performance, including symmetrizing the block matrix and implementing exact and preconditioned block-Jacobi iterations on the symmetrized system. These modifications allow us to maintain the excellent accuracy of the S3 formula while significantly improving computational efficiency, making it suitable for practical applications. Furthermore, we achieve parallelization using OpenMP and MPI, which enables our proposed method to be effectively applied to both one-dimensional and two-dimensional cases. This enhancement significantly improves scalability and efficiency for large-scale applications.

The paper is structured as follows. In section 2, we leverage the S3 formula for approximating the Caputo fractional derivative. Following that, in section 3, we combine the S3 formula with the finite difference method to solve a linear system and then attempt to solve it using a 1D multigrid solver based on cyclic reduction on both uniform and nonuniform spatial meshes. In section 4, our focus will be on solving the two-dimensional subdiffusion problem with

S3-FDM and proposing a parallel algorithm based on the multigrid method. We will also present some numerical examples to demonstrate the efficiency of the derived algorithms.

2. Time discretization

For any given integer $n, m > 0$, we set $\Delta t = 1/n$, $t_k = k\,\Delta t$ for $k = 0, 1, \ldots, n$, which constructs a partition over the interval $[0, 1]$ as $0 = t_0 < t_1 < \cdots < t_{n-1} < t_n = 1$. The S3 formula for approximating the Caputo derivative (1.2), as described in [18], is based on cubic spline interpolation and can be defined as follows

$$D_t^{\alpha,3} v(t)\big|_{t=t_k} := \frac{\Delta t^{-\alpha}}{\Gamma(4-\alpha)}\Big(\beta_{k+2} - \sum_{j=1}^{k+1} d_{k-j+2}^{(\alpha)}\,\beta_j - d_{k+2}^{(\alpha)}\,\beta_0\Big), \tag{2.1}$$

in which the coefficients $d_j^{(\alpha)}$ are derived by

$$d_j^{(\alpha)} = \begin{cases} 3a_0 - a_1 - 2b_0, & j = 1, \\ -3a_0 + 3a_1 - a_2 + b_0 - b_0', & j = 2, \\ a_{j-3} - 3a_{j-2} + 3a_{j-1} - a_j, & 3 \le j \le k-1, \\ a_{j-3} - 3a_{j-2} + 3a_{j-1} - b_j - b_j', & j = k, \\ a_{j-3} - 3a_{j-2} + 2b_{j-1}, & j = k+1 \\ a_{k-1} - b_k + b_k', & j = k+2, \end{cases}$$

therein

$$a_i = (i+1)^{3-\alpha} - i^{3-\alpha}, \quad b_i = (3-\alpha)i^{2-\alpha}, \quad b_i' = \frac{1}{2}(3-\alpha)(2-\alpha)i^{1-\alpha}, \quad i \ge 0.$$

In (2.1), β_j's are the coefficients related to the spline interpolant of degree 3 determined in the system of linear equations $\mathbf{M}\boldsymbol{\beta} = \mathbf{v}$, where

$$\mathbf{M} = \frac{1}{6}\begin{bmatrix} \frac{-3}{\Delta t} & 0 & \frac{3}{\Delta t} & \cdots & \cdots & 0 \\ 1 & 4 & 1 & \cdots & \cdots & 0 \\ 0 & 1 & 4 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 1 & 4 & 1 \\ 0 & \cdots & \cdots & \frac{-3}{\Delta t} & 0 & \frac{3}{\Delta t} \end{bmatrix}, \tag{2.2}$$

and $\mathbf{v} = [v'(t_0), v(t_0), \ldots, v(t_k), v'(t_k)]^T$ and $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_{k+1}, \beta_{k+2}]^T$.

Remark 2.1. The error estimates for the S3 formula in [18] were derived assuming that $v \in \mathrm{C}^4[0, T]$ to achieve a convergence order of $\mathcal{O}(\Delta t^{4-\alpha})$. However, the numerical results in the paper indicate that the method maintains a convergence order of $4 - \alpha$ even when the solution v belongs to the weaker regularity class $v \in \mathrm{C}^3[0, T]$. This implies that it may be possible to attain a higher convergence order than the L1-2 [4] and L2-1_σ [1] formulas under comparable conditions. Thus, it might be feasible to ease the regularity requirements for the method's convergence order without sacrificing performance. Consequently, future research will focus on rigorously proving convergence under these less restrictive conditions.

As mentioned in Remark 2.1, while the proposed method adopts similar regularity assumptions as the L1-2 and L2-1_σ formulas, it achieves a higher convergence order. However, it does not explicitly address weakly singular solutions. Additional strategies are often necessary to effectively address the complexities arising from singularities. Techniques such as graded meshes, which adaptively refine the temporal grid to concentrate computational effort in regions where singularities occur, can be incredibly beneficial [22]. Additionally, corrected schemes may offer modified approaches that enhance the accuracy of existing methods in the presence of complications [20]. Moreover, transformed methods

can be employed to simplify the analysis and treatment of singularities [17]. Nevertheless, refining the proposed method and conducting a rigorous theoretical analysis to deal with weak singular solutions necessitates a more in-depth investigation, which is beyond the scope of this paper.

3. One dimensional problem

In this section, we consider the following time-fractional diffusion problem in $\Omega = [a, b]$,

$$\begin{aligned} &D_t^\alpha u(x,t) - u_{xx}(x,t) = f(x,t), \qquad (x,t) \in \Omega \times (0,T], \\ &u(a,t) = \varphi(t), \quad u(b,t) = \phi(t), \qquad t \in (0,T], \\ &u(x,0) = g(x), \qquad x \in \Omega. \end{aligned} \tag{3.1}$$

To solve this problem numerically, we propose a finite difference-based approach using the S3 formula and finite difference method on both uniform and non-uniform spatial meshes, called the S3-FD method, which provides a high-order approximation for the Caputo fractional derivative. Then, we employ a 1D multigrid solver based on cyclic reduction to solve the resulting linear system efficiently.

3.1. **Equidistant spatial discretization.** Let $h = 1/m$ in order to define a mesh $\Omega_h \times \Omega_{\Delta t}$ in which $\Omega_h = \{x_i | x_i = i\,h, 0 \le i \le m\}$. We use the S3 formula (2.1) to approximate the fractional derivative and the second-order central difference denoted by $\delta_x^2 u_i^k$ for the second-order space derivative as

$$\frac{\partial^2 u(x_i,t_k)}{\partial x^2} = \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{h^2} + \mathcal{O}(h^2),$$

therefore, the difference scheme is as follows

$$D_t^{\alpha,3} u_i^k = \delta_x^2 u_i^k + f_i^k, \qquad 1 \le i \le m-1, 1 \le k \le n, \tag{3.2}$$

$$u_0^k = \varphi(t_k), \qquad u_m^k = \phi(t_k), \qquad 1 \le k \le n, \tag{3.3}$$

$$u_i^0 = g(x_i), \qquad 0 \le i \le m, \tag{3.4}$$

where $u_i^k = \sum_{j=0}^{k+2} \beta_i^j B_{j,3}(t_k)$ and $B_{j,3}$ is the B-spline function satisfying the following recursive formula

$$B_{j,m}(t) = \left(\frac{t - t_j}{t_{j+m} - t_j}\right) B_{m-1,j}(t) + \left(\frac{t_{j+m+1} - t}{t_{j+m+1} - t_{j+1}}\right) B_{m-1,j+1}(t), \quad m \ge 1,$$

with

$$B_{0,j}(t) = \begin{cases} 1, & t_j \le t < t_{j+1}, \\ 0, & \text{Otherwise.} \end{cases}$$

To ensure the uniqueness of the solution of finite difference scheme (3.2)–(3.4), we imposed the following conditions

$$u_t(x_i,0) \simeq \sum_{j=0}^{2} \beta^j(x_i) \frac{d}{dt} B_{j,3}(0), \qquad 0 \le i \le m, \tag{3.5}$$

$$u_t(x_i,t_n) \simeq \sum_{j=0}^{n+1} \beta^j(x_i) \frac{d}{dt} B_{j,3}(t_n), \qquad 0 \le i \le m. \tag{3.6}$$

We firstly suppose that the u_t is known in the t_0 and t_n, hence finding β_i^j leads to the following system

$$\mathbf{K}\boldsymbol{\beta} = \mathbf{f}, \tag{3.7}$$

with $\boldsymbol{\beta} = [\beta_0^0, \beta_0^1, \ldots, \beta_0^{n+2}, \beta_1^0, \beta_1^1, \ldots, \beta_1^{n+2}, \ldots, \beta_m^0, \beta_m^1, \ldots, \beta_m^{n+2}]^T$ and the following right hand side

$$\mathbf{f} = [\mathbf{f}_0; \mathbf{f}_1; \mathbf{f}_2; \ldots; \mathbf{f}_{m-1}; \mathbf{f}_m],$$

in which

$$\begin{aligned}
\mathbf{f}_0 &= [u_t(x_0,0), g(x_0), \varphi(t_1), \varphi(t_2), \ldots, \varphi(t_n), u_t(x_0,t_n)]^T,\\
\mathbf{f}_m &= [u_t(x_m,0), g(x_m), \phi(t_1), \phi(t_2), \ldots, \phi(t_n), u_t(x_m,t_n)]^T,\\
\mathbf{f}_i &= [u_t(x_i,0), g(x_i), f(x_i,t_1), f(x_i,t_2), \ldots, f(x_i,t_n), u_t(x_i,t_n)]^T, \quad 1 \le i \le m-1.
\end{aligned}$$

In addition, $\mathbf{K}$ is a block-three-diagonal matrix that is structured by

$$\begin{bmatrix}
\mathbf{M} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0}\\
\frac{-1}{h^2}\mathbf{B} & \mathbf{A} & \frac{-1}{h^2}\mathbf{B} & & \vdots\\
\mathbf{0} & \ddots & \ddots & \ddots & \mathbf{0}\\
\vdots & & \frac{-1}{h^2}\mathbf{B} & \mathbf{A} & \frac{-1}{h^2}\mathbf{B}\\
\mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{M}
\end{bmatrix}, \tag{3.8}$$

where $\mathbf{M}$ is given by (2.2), $\mathbf{B}$ is the same matrix $\mathbf{M}$ with the zero elements in the 1-th, 2-th and (N+3)-th row, i.e.,

$$\mathbf{B} := \operatorname{diag}\{0,0,1,\ldots,1,0\}\,\mathbf{M}, \tag{3.9}$$

and each row of matrix $\mathbf{A}$ is as follows

$$\mathbf{A}_i = \begin{cases} \mathbf{M}_i, & i = 1,2,N+3,\\ \dfrac{2}{h^2}\left(\mathbf{M}_i - \dfrac{h^2}{2}\dfrac{\Delta t^{-\alpha}}{\Gamma(4-\alpha)}\mathbf{D}_i\right), & \textit{Otherwise}, \end{cases}$$

therein

$$\mathbf{D} = \begin{bmatrix}
0 & 0 & 0 & 0 & & \cdots & \cdots & 0\\
0 & 0 & 0 & 0 & & \cdots & & \vdots\\
d_3 & d_2 & d_1 & -1 & 0 & & & \vdots\\
d_4 & d_3 & d_2 & d_1 & -1 & 0 & & \vdots\\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\
\vdots & & \ddots & \ddots & & \ddots & \ddots & 0\\
d_{n+2} & \cdots & \cdots & d_4 & d_3 & d_2 & d_1 & -1\\
0 & \cdots & \cdots & \cdots & 0 & 0 & 0 & 0
\end{bmatrix}. \tag{3.10}$$

In MATLAB: `D=diag([0,0,ones(1,numel(d)-2),0])*toeplitz([d;0],[d(1),-1,zeros(1,numel(d)-1)])`.

Finally, we can express $\mathbf{A}$ as

$$\mathbf{A} = \operatorname{diag}\left\{1,1,\frac{2}{h^2},\ldots,\frac{2}{h^2},1\right\}\left(\mathbf{M} - \frac{h^2}{2}\frac{\Delta t^{-\alpha}}{\Gamma(4-\alpha)}\mathbf{D}\right). \tag{3.11}$$

System (3.7) needs $\mathcal{O}((mn)^2)$ computational cost if the coefficient matrix is stored as a full dense matrix. Therefore, we use the cyclic reduction to solve this system more efficiently.

3.1.1. *Cyclic reduction.* Let us consider solving the block tridiagonal linear system (3.7) that each block is an $(n+3)\times(n+3)$ matrix and each vector $\boldsymbol{\beta}_i$ and $\mathbf{f}_i$ is of order $n+3$. To apply the cyclic reduction to this system, we follow this algorithm:

Let $m = 2^i, i \ge 2$, we fixed the first and last rows of the system. Then, the odd rows (numbering starts with 0) are to be eliminated and the new equation for even rows is obtained by multiplying the odd rows by $A^{-1}B$ and adding them to even rows.

TABLE 1. Maximum error norms and corresponding temporal orders of S3-FDM.

	$\alpha = 0.9$		$\alpha = 0.5$		$\alpha = 0.1$	
n	E_∞	$\mathcal{O}$	E_∞	$\mathcal{O}$	E_∞	$\mathcal{O}$
10	$1.505472e-04$	3.12	$1.473331e-04$	3.44	$1.566189e-04$	3.84
20	$1.731103e-05$	3.17	$1.352889e-05$	3.56	$1.093481e-05$	3.94
40	$1.925144e-06$	3.18	$1.144631e-06$	3.51	$7.105096e-07$	3.90
80	$2.119429e-07$	3.14	$1.007123e-07$	3.51	$4.746520e-08$	3.87
160	$2.398335e-08$		$8.810263e-09$		$3.252351e-09$	

This elimination yields the following reduced system of equations

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \frac{-1}{h^2}\mathbf{B}^{(1)} & \mathbf{A}^{(1)} & \frac{-1}{h^2}\mathbf{B}^{(1)} & & \vdots \\ \mathbf{0} & \ddots & \ddots & \ddots & \mathbf{0} \\ \vdots & & \frac{-1}{h^2}\mathbf{B}^{(1)} & \mathbf{A}^{(1)} & \frac{-1}{h^2}\mathbf{B}^{(1)} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{\beta}_0 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_{m-2} \\ \boldsymbol{\beta}_m \end{bmatrix} = \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{f}_2^{(1)} \\ \vdots \\ \mathbf{f}_{m-2}^{(1)} \\ \mathbf{f}_m \end{bmatrix},$$

where $\mathbf{B}^{(1)} = \mathbf{B}\mathbf{A}^{-1}\mathbf{B}$, $\mathbf{A}^{(1)} = \mathbf{A} - 2\mathbf{B}^{(1)}$ and for $j = 2, 4, \cdots, m-2$,

$$\mathbf{f}_j^{(1)} = \mathbf{f}_j + \mathbf{B}\mathbf{A}^{-1}\left(\mathbf{f}_{j+1} + \mathbf{f}_{j-1}\right).$$

Continuing this reduction the original system of Equations (3.7) will be reduced to the

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} \\ \frac{-1}{h^2}\mathbf{B}^{(i-1)} & \mathbf{A}^{(i-1)} & \frac{-1}{h^2}\mathbf{B}^{(i-1)} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{\beta}_0 \\ \boldsymbol{\beta}_{m/2} \\ \boldsymbol{\beta}_m \end{bmatrix} = \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{f}_{m/2}^{(i-1)} \\ \mathbf{f}_m \end{bmatrix}.$$

The remaining unknowns are then computed by the back substitution process.

3.1.2. *Numerical results.* Here, all computations are performed on an AMD Ryzen Threadripper 1950X 16-Core, 3.4 GHz with 128 GB of memory.

Example 3.1. We consider the subdiffusion problem (3.1) on $\Omega = (0, 1)$ where the source term given by $f(x,t) = e^x t^3 \left(\frac{\Gamma(4+\alpha)}{3!} - t^\alpha\right)$. The initial and boundary conditions are specified as $g(x) = 0$, $\phi(t) = t^{3+\alpha}$, and $\varphi(t) = et^{3+\alpha}$, with the exact solution expressed as $u(x,t) = e^x t^{3+\alpha}$. We initially solve this problem using the S3-FDM and present the temporal convergence order in Table 1. Although the exact solution $u(\cdot,t) \in C^3[0,T]$, Table 1 shows that we still observe a temporal convergence order of $4-\alpha$, highlighting the method's robustness in this setting.

We conduct simulations with various step sizes $m = 2^6$ to 2^{10}. We use the cyclic reduction method implemented in both MATLAB and C++ to compute the results. These results are then compared with the original MATLAB code for $\alpha = 0.9$, and the comparison is shown in Figure 1. Table 2 provides a detailed comparison of CPU times for a fixed number of time steps $n = 1024$, demonstrating the efficiency of different implementations. Furthermore, in Figure 2, we optimize the performance for larger step sizes $m = 2^{15}$ to 2^{20} by using C++ code with OpenMP parallelization. By varying the number of threads, we can significantly reduce the CPU time.

In addition, we measure the CPU time for a fixed step size $m = 2^{14}$ to evaluate the efficiency of the cyclic reduction method. The results, listed in Table 3, demonstrate significant performance improvements. For a more visual comparison, the CPU times are plotted in Figure 3, demonstrating the efficiency gains achieved through cyclic reduction. When dealing with larger time steps, $n = 10^{12}$ to 10^{14}, we assess the computational performance under

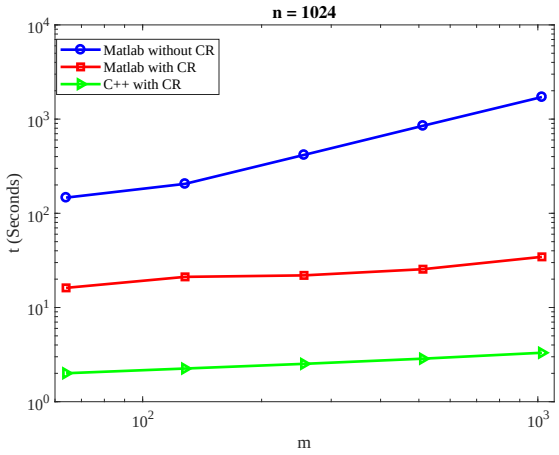

FIGURE 1. Comparing CPU-time between C++ and MATLAB code with different space discretizations.

TABLE 2. CPU times comparison, $n = 1024$.

m	MATLAB	MATLAB-CR	C++-CR
64	146.702629 sec.	16.160657 sec.	2.00483 sec.
128	205.356619	21.144071	2.24961
256	416.575058	21.925472	2.52272
512	849.021644	25.502045	2.86248
1024	1714.822793	34.488285	3.30851

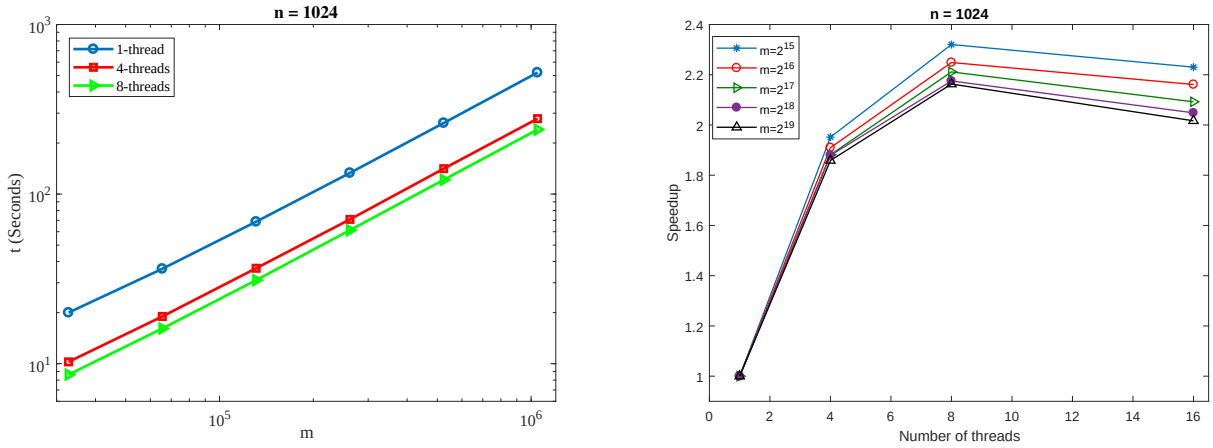

FIGURE 2. Run time with different number of threads (left) and strong speedup with different number of threads (right).

TABLE 3. CPU times comparison with the fixed $m = 2^{14}$.

n	MATLAB	MATLAB-CR	C++-CR
16	4.602478	0.469805	0.051962
32	9.910498	0.37234	0.0819394
64	29.934466	0.527201	0.17538
128	114.78865	1.155388	0.390583
256	1126.164315	3.528074	1.01247
512	3971.460558	15.447851	3.05244

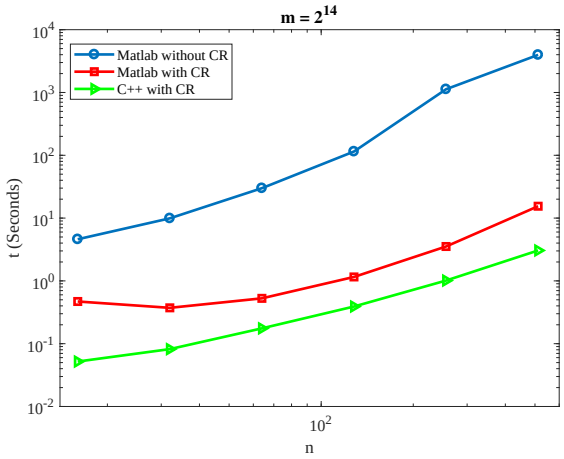

FIGURE 3. Comparing CPU-time between C++ and MATLAB code with different time steps.

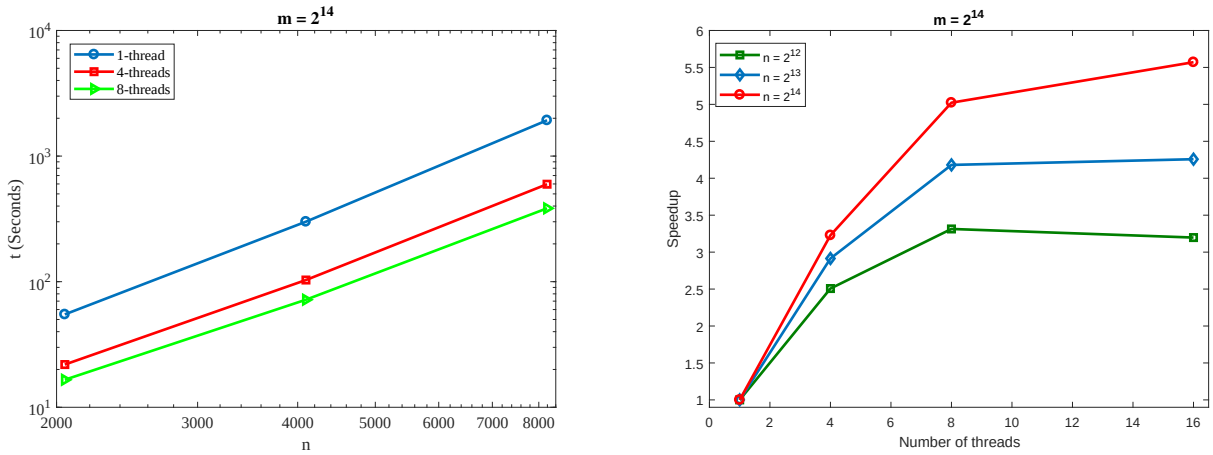

FIGURE 4. Run time with different number of threads (left) and speedup with different number of threads (right).

parallel programming. Figure 4 illustrates the CPU time required for these large-scale computations, showing that parallelization reduces computational time.

3.2. **Non-equidistant spatial discretization.** In numerical simulations, using a nonuniform mesh is a well-established technique that improves computational efficiency. Instead of uniformly refining the entire domain, this approach enables adaptive grid spacing: it uses finer meshes in areas with sharp gradients and coarser meshes in regions where the solution is smooth, optimizing computational costs. Although the current model does not necessarily require a nonuniform grid, incorporating this flexibility lays the groundwork for extending the method to more complex and practical problems. To this aim, let us define a mesh $\Omega_h \times \Omega_{\Delta t}$ in which $\Omega_h = \{x_i, 0 \le i \le m\}$ with $0 <= x_0 < x_1 < \cdots < x_i < x_{i+1} < \cdots < x_m = 1$. That defines a local discretization parameter $h_i := x_{i+1} - x_i$, $i = 0, \ldots, m-1$ resulting in the following approximation for the second derivative in space for $i = 1, \ldots, m-1$

$$\frac{\partial^2 u(x_i, t_k)}{\partial x^2} \approx \frac{2}{h_{i-1}h_i(h_i + h_{i-1})} \left(h_i u_{i-1}^k - (h_i + h_{i-1}) u_i^k + h_{i-1} u_{i+1}^k \right). \tag{3.12}$$

so the difference scheme is as follows

$$D_t^{\alpha,3} u_i^k = \delta_x^2 u_i^k + f_i^k, \quad 1 \le i \le m-1, 1 \le k \le n, \tag{3.13}$$

$$u_0^k = \varphi(t_k), \qquad u_m^k = \phi(t_k), \quad 1 \le k \le n, \tag{3.14}$$

$$u_i^0 = g(x_i), \quad 0 \le i \le m. \tag{3.15}$$

Imposing the conditions (3.5) and (3.6), we ensure the uniqueness of the solution of the finite difference scheme (3.13)–(3.15), leading to solving the system (3.7) with a different block-three-diagonal matrix $\mathbf{K}$ as follows

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \vdots & \ddots & \ddots & & & \vdots \\ & \frac{-2}{h_{i-2}(h_{i-1}+h_{i-2})}\mathbf{B} & \mathbf{A}_{i-1} & \frac{-2}{h_{i-1}(h_{i-1}+h_i)}\mathbf{B} & & \\ & & \frac{-2}{h_{i-1}(h_i+h_{i-1})}\mathbf{B} & \mathbf{A}_i & \frac{-2}{h_i(h_i+h_{i-1})}\mathbf{B} & \\ \vdots & & & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix}, \tag{3.16}$$

where $\mathbf{M}, \mathbf{B}$, and $\mathbf{D}$ are the same as before, and $\mathbf{A_i}$ is the following matrix

$$\mathbf{A}_i = \frac{2}{h_{i-1}h_i}\mathbf{M} - \frac{\Delta t^{-\alpha}}{\Gamma(4-\alpha)}\mathbf{D} = \frac{2}{h_{i-1}h_i}\left(\mathbf{M} - \frac{h_{i-1}h_i}{2}\frac{\Delta t^{-\alpha}}{\Gamma(4-\alpha)}\mathbf{D} \right).$$

except in the 1-th, 2-th and (N+3)-th row, that is the same as $\mathbf{M}$. So we write

$$\mathbf{A}_i = \underbrace{\operatorname{diag}\left\{1, 1, \frac{2}{h_{i-1}h_i}, \ldots, \frac{2}{h_{i-1}h_i}, 1\right\}}_{A_{L,i} :=} \underbrace{\left(\mathbf{M} - \frac{h_{i-1}h_i}{2}\frac{\Delta t^{-\alpha}}{\Gamma(4-\alpha)}\mathbf{D} \right)}_{\widetilde{A}_i :=}. \tag{3.17}$$

3.2.1. *Symmetrization of block matrix.* The block matrix $\mathbf{K}$ (3.16) can be symmetrized concerning the factors in front of the off-diagonal blocks, i.e., block matrix $K_{i-1,i}$ differs from $K_{i,i-1}$ only by the factors $\frac{1}{h_{i-1}+h_{i-2}}$ and $\frac{1}{h_i+h_{i-1}}$. Multiplying K from the left with $S_L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \operatorname{diag}\{h_i + h_{i-1}\}_{i=1}^{M-1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$ results in a re-scaled system of equations

$$K_S \underline{\beta} := S_L \cdot K \underline{\beta} = S_L \cdot \underline{f} =: \underline{f}_S \tag{3.18}$$

with the re-scaled system matrix K_S

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \vdots & \ddots & \ddots & & & \vdots \\ & \frac{-2}{h_{i-2}}\mathbf{B} & (h_{i-1}+h_{i-2})\mathbf{A}_{i-1} & \frac{-2}{h_{i-1}}\mathbf{B} & & \\ & & \frac{-2}{h_{i-1}}\mathbf{B} & (h_i+h_{i-1})\mathbf{A}_i & \frac{-2}{h_i}\mathbf{B} & \\ \vdots & & & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix}.$$

The $\mathbf{A_i}$ from (3.17) remains unchanged and the structure of the scaled block matrix is similar to a finite element discretization in space with linear test functions. Using the factorized $\mathbf{A_i}$ from (3.17) we can express K_S as

$$K_s = \begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \vdots & \ddots & \ddots & & & \vdots \\ & \frac{-2}{h_{i-2}}\mathbf{B} & (h_{i-1}+h_{i-2})\mathbf{A}_{L,i-1}\widetilde{\mathbf{A}}_{i-1} & \frac{-2}{h_{i-1}}\mathbf{B} & & \\ & & \frac{-2}{h_{i-1}}\mathbf{B} & (h_i+h_{i-1})\mathbf{A}_{L,i}\widetilde{\mathbf{A}}_i & \frac{-2}{h_i}\mathbf{B} & \\ \vdots & & & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix}. \tag{3.19}$$

3.2.2. *Exact block-Jacobi iteration of symmetrized system.* Solving $K_S\underline{\beta} = \underline{f}_S$ with the block matrix from (3.19) might contain the Jacobi iteration

$$\widehat{\underline{\beta}} := \underline{\beta} + \omega \mathrm{diag}(K_S)^{-1}\left(\underline{f}_S - K_s\underline{\beta}\right) \tag{3.20}$$

with

$$\mathrm{diag}(K_S)^{-1} = \begin{bmatrix} \mathbf{M}^{-1} & & & & \\ & \ddots & & & \\ & \widetilde{\mathbf{A}}_{i-1}^{-1}\mathbf{A}_{L,i-1}^{-1}\frac{1}{(h_{i-1}+h_{i-2})} & & & \\ & & \widetilde{\mathbf{A}}_i^{-1}\mathbf{A}_{L,i}^{-1}\frac{1}{(h_i+h_{i-1})} & & \\ & & & \ddots & \\ & & & & \mathbf{M}^{-1} \end{bmatrix}. \tag{3.21}$$

The multiplication by K_S in (3.20) can be implemented directly and cheaply via the fixed matrices $\mathbf{M}$ and $\mathbf{D}$ together with the cheap individual scaling (wrt. i) denoted by the diagonal matrix $\mathbf{A}_{L,i}$ (3.17) stored in a vector. But the exact inverse of the diagonal (3.21) requires the inversion (or system solve) of each $\widetilde{\mathbf{A}}_i$, i.e, it requires $\mathcal{O}(mn^3)$ arithmetic operations and $\mathcal{O}(mn^2)$ additional storage.

3.2.3. *Preconditioned block-Jacobi iteration of symmetrized system.* The expenses for iteration (3.20) can be reduced by using a preconditioner $C \approx \mathrm{diag}(K_S)$ that substitutes the expensive inversions of $\widetilde{\mathbf{A}}_i$ by the inversion of one matrix

$$\overline{C} := \mathbf{M} - \frac{\overline{h}}{2}\frac{\Delta t^{-\alpha}}{\Gamma(4-\alpha)}\mathbf{D},$$

with $\overline{h} := \max\{h_i\}$ such that the preconditioner can be defined as

$$\operatorname{diag}(K_S)^{-1} = \begin{bmatrix} \mathbf{M}^{-1} & & & \\ & \overline{C}^{-1}\mathbf{A}_{L,i-1}^{-1}\frac{1}{(h_{i-1}+h_{i-2})} & & \\ & & \overline{C}^{-1}\mathbf{A}_{L,i}^{-1}\frac{1}{(h_i+h_{i-1})} & \\ & & & \ddots \\ & & & & \mathbf{M}^{-1} \end{bmatrix}.$$

Now the additional storage is reduced to $\mathcal{O}(n^2)$ and we need $\mathcal{O}(n^3 + mn^2)$ arithmetic operations for inversion and application of the inverse. Finally the preconditioned Jacobi iteration

$$\underline{\widehat{\beta}} := \underline{\beta} + \omega C^{-1}\left(\underline{f}_S - K_s\underline{\beta}\right),$$

can be expressed for block row i

$$\begin{aligned} \underline{\widehat{\beta}}_i &:= \underline{\beta}_i + \omega\overline{C}^{-1}\mathbf{A}_{L,i}^{-1}\left(\underline{f}_S - K_s\underline{\beta}\right)_i \qquad \forall i = 1,\dots,M-1 \\ \text{resp.} \quad \underline{\widehat{\beta}}_i &:= \underline{\beta}_i + \omega\mathbf{M}^{-1}\left(\underline{f}_S - K_s\underline{\beta}\right)_i \qquad i \in \{0, M\}\ . \end{aligned}$$

4. Two dimensional problem

We consider (1.1) where $\Omega \in \mathbb{R}^2$ and f, g, u_0 are given functions. Let m_1, m_2, n be positive integers and define a mesh $\Omega_{h_x} \times \Omega_{h_y} \times \Omega_{\Delta t}$ such that $h_x = 1/m_1$, $h_y = 1/m_2$, $\Delta t = T/n$ and

$$\Omega_{h_x} = \{x_i \,|\, x_i = x_0 + ih_x, i = 0, 1, \dots, m_1\}, \quad \Omega_{h_y} = \{y_j \,|\, y_j = y_0 + jh_y, j = 0, 1, \dots, m_2\}.$$

We use the S3 formula and second-order central difference for approximating the Caputo and spatial derivatives, respectively, to construct the full difference scheme

$$\begin{aligned} &D_t^{\alpha,3}u_{i,j}^k = \delta_x^2 u_{i,j}^k + f_{i,j}^k, \\ &u^k|_{\partial\Omega} = g_{i,j}^k, \end{aligned} \tag{4.1}$$

for solving problem (1.1) in which $u(x_i, y_j, t_k) \simeq \sum_{l=0}^{k+2} \beta_{i,j}^l B_{l,3}(t_k)$, and for $0 \le i \le m_1$ and $0 \le i \le m_2$ we have

$$u'(x_i, y_j, 0) \simeq \sum_{l=0}^{2} \beta_{i,j}^l \frac{d}{dt}B_{l,3}(0), \quad u'(x_i, y_j, t_n) \simeq \sum_{l=0}^{n+1} \beta_{i,j}^l \frac{d}{dt}B_{l,3}(t_n).$$

Finding coefficient $\beta_{i,j}^l$ leads to solving the system $\mathbf{K}\boldsymbol{\beta} = \mathbf{f}$, where the block components of $\mathbf{f}$ are

$$\begin{aligned} \mathbf{f}_{\partial\Omega} &= [u'(x_i, y_j, 0), u_0(x_i, y_j), g(x_i, y_j, t_1), g(x_i, y_j, t_2), \dots, g(x_i, y_j, t_n), u'(x_i, y_j, t_n)]^T, \\ \mathbf{f}_{\Omega\setminus\partial\Omega} &= [u'(x_i, y_j, 0), u_0(x_i, y_j), f(x_i, y_j, t_1), f(x_i, y_j, t_2), \dots, f(x_i, y_j, t_n), u'(x_i, y_j, t_n)]^T, \end{aligned}$$

and

$$\boldsymbol{\beta} = [\beta_{0,0}^0, \beta_{0,0}^1, \dots, \beta_{0,0}^{n+2}; \beta_{1,0}^0, \beta_{1,0}^1, \dots, \beta_{1,0}^{n+2}; \dots; \beta_{m_1,m_2}^0, \beta_{m_1,m_2}^1, \dots, \beta_{m_1,m_2}^{n+2}]^T.$$

In addition, $\mathbf{K}$ is a block five-diagonal matrix. For example, if $m_1 = m_2 = 2$ the matrix $\mathbf{K}$ is structured by

$$\mathbf{K} = \begin{bmatrix} \mathbf{M} & & & & & & & & \\ & \mathbf{M} & & & & & & & \\ & & \mathbf{M} & & & & & & \\ & & & \mathbf{M} & & & & & \\ & \frac{-1}{h_y^2}\mathbf{B} & \mathbf{0} & \frac{-1}{h_x^2}\mathbf{B} & \bar{\mathbf{A}} & \frac{-1}{h_x^2}\mathbf{B} & \mathbf{0} & \frac{-1}{h_y^2}\mathbf{B} & \\ & & & & & \mathbf{M} & & & \\ & & & & & & \mathbf{M} & & \\ & & & & & & & \mathbf{M} & \\ & & & & & & & & \mathbf{M} \end{bmatrix},$$

in which $\mathbf{M}$, $\mathbf{B}$ and $\mathbf{D}$ are the same as 1D, and

$$\bar{\mathbf{A}} = \left(\frac{2}{h_x^2} + \frac{2}{h_y^2}\right)\mathbf{M} - \frac{\Delta t^{-\alpha}}{\Gamma(4-\alpha)}\mathbf{D},$$

except in the 1-th, 2-th and (n+3)-th row which is the same as $\mathbf{M}$. The matrix $\mathbf{K}$ is $\bar{n} \times \bar{n}$ with $\bar{n} = (n+3)(m_1+1)(m_2+1)$. Then solving the system $\mathbf{K}\boldsymbol{\beta} = \mathbf{f}$ using direct methods leads to high computational costs. In this case, we aim to use multigrid in 2D to reduce the computational costs.

4.1. **Multigrid method.** Let $\mathbf{e} = \boldsymbol{\beta} - \bar{\boldsymbol{\beta}}$ where $\bar{\boldsymbol{\beta}}$ is an approximation for $\boldsymbol{\beta}$. The idea of the multigrid method is to solve residual system $\mathbf{Ke} = \mathbf{r}$ on a coarse mesh instead of solving original system $\mathbf{K}\boldsymbol{\beta} = \mathbf{f}$ on a finer mesh, and update the finer solution with the interpolated coarse correction $\mathbf{e}$. Suppose that $i \geq 2$ and $m_1 = m_2 = 2^i$. We need two following operators:

- Prolongation: Let v^{2h} be in Ω_{2h}, then the linear interpolation

  $I_{2h}^h : \Omega_{2h} \to \Omega_h, \quad I_{2h}^h v^{2h} = v^h,$

  is defined by

$$\begin{aligned} v_{2i,2j}^h &= v_{i,j}^{2h}, \\ v_{2i+1,2j}^h &= \frac{1}{2}(v_{i,j}^{2h} + v_{i+1,j}^{2h}), \\ v_{2i,2j+1}^h &= \frac{1}{2}(v_{i,j}^{2h} + v_{i,j+1}^{2h}), \\ v_{2i+1,2j+1}^h &= \frac{1}{4}(v_{i,j}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h}), \quad i,j = 0,\ldots,\frac{m}{2}-1. \end{aligned}$$

- Full Restriction: the projection operator is defined by

  $I_h^{2h} : \Omega_h \to \Omega_{2h}, \quad I_h^{2h} v^h = v^{2h},$

  with

$$\begin{aligned} v_{i,j}^{2h} = v_{2i,2j}^h &+ \frac{1}{2}(v_{2i+1,2j}^h + v_{2i-1,2j}^{2h} + v_{2i,2j+1}^h + v_{2i,2j-1}^{2h}) \\ &+ \frac{1}{4}(v_{2i+1,2j+1}^h + v_{2i+1,2j-1}^h + v_{2i+1,2j-1}^h + v_{2i-1,2j-1}^h). \end{aligned}$$

Using the linear prolongation and full restriction, we can now formulate the multigrid algorithm:

In this algorithm, we use ω-Jacobi iteration as relaxation

$$\bar{\boldsymbol{\beta}}^{r+1} = \bar{\boldsymbol{\beta}}^r + \omega\,\mathbf{D}^{-1}(\mathbf{f} - \mathbf{K}\bar{\boldsymbol{\beta}}),$$

where $\mathbf{D}$ is a block-diagonal matrix constructed from block-diagonal components in a matrix $\mathbf{K}$. To parallelize Algorithm 1, we use C++ code with MPI using domain decomposition strategy [2]. Now, for the whole parallelizing

Algorithm 1: Recursive multigrid method.

Input: Suppose $\mathbf{K}\boldsymbol{\beta} = \mathbf{f}$ with initial guess $\bar{\boldsymbol{\beta}}$;
If Ω_h *has 9 points* **then**
  Solve system (Jacobi iterations);
else
  Step 1: Relaxation:
    using r iterations of Jacobi method for solving $\mathbf{K}_h\boldsymbol{\beta} = \mathbf{f}$ with initial guess $\bar{\boldsymbol{\beta}}$ and correcting the initial guess;
  Step 2: Residual Calculation ($\mathbf{r}_h := \mathbf{f} - \mathbf{K}_h\bar{\boldsymbol{\beta}}$);
  Step 3: Restriction:
    restricting $\mathbf{r}_h$ to Ω_{2h} ($I_h^{2h}\mathbf{r}_{2h} = \mathbf{r}_h$);
  Step 4: Approximation on Ω_{2h}:
    approximating the solution of $\mathbf{K}_{2h}\boldsymbol{e}_{2h} = \mathbf{r}_{2h}$ using multigrid method and finding $\bar{\mathbf{e}}_{2h}$;
  Step 5: Correction:
    interpolation and correction ($\bar{\boldsymbol{\beta}} = \bar{\boldsymbol{\beta}} + I_h^{2h}\bar{\mathbf{e}}_{2h}$);
  Step 6: Relaxation (Jacobi iterations);
end

TABLE 4. Errors and corresponding temporal orders for S3-FDM.

	$\alpha = 0.9$		$\alpha = 0.5$		$\alpha = 0.1$	
n	E_∞	$\mathcal{O}$	E_∞	$\mathcal{O}$	E_∞	$\mathcal{O}$
2^2	$2.78308e-4$	3.03	$1.70380e-4$	3.41	$1.39682e-5$	3.75
2^3	$3.40545e-5$	3.06	$1.60076e-5$	3.45	$1.03191e-6$	4.03
2^4	$4.07769e-6$	3.10	$1.46840e-6$	3.59	$6.28881e-8$	3.80
2^5	$4.72741e-7$		$1.21841e-7$		$4.54002e-9$	

Algorithm 1, we parallel all components including restriction, interpolation, relaxation, and approximating solution on a coarse mesh.

Remark 4.1. The time-fractional diffusion equation examined in this study is a crucial model for understanding anomalous diffusion processes. While this work focuses on the linear case, the proposed numerical framework is designed to be highly scalable and computationally efficient. This makes it a promising candidate for addressing more complex problems, including nonlinear subdiffusion equations. Nonlinear extensions typically require additional considerations, such as managing nonlinear terms, ensuring stability, and adapting iterative solvers.

4.2. **Numerical results.** Here, all computations are performed on a high-performance computing system with an AMD EPYC Processor featuring 64 Cores and 256 GB of memory.

Example 4.2. We focus on solving the fractional subdiffusion problem (1.1) over a square domain. We set up the problem using the exact solution $u(x, y, t) = t^{4+\alpha}(\exp(x) + \cos(y))$ and compute the numerical solution using the proposed S3-FD Method. To assess the performance and accuracy of the method, we present the maximum norm errors and convergence orders in Table 4, demonstrating that the proposed method achieves an accuracy of $\mathcal{O}(\Delta t^{4-\alpha})$. In the next phase of our numerical experiments, we implement the code using MPI parallel programming to take advantage of distributed computing resources. By increasing the number of processing cores and adjusting the spatial step size, we run the parallel code for a spatial discretization of $n = 2^6$. The corresponding runtimes are summarized in Table 5. Moreover, we observe a significant strong scaling speedup for large time and space step sizes when varying

TABLE 5. Scaleup with fixed time discretization $n = 2^6$.

m	cores	subdomains	Run time
2^6	1	1×1	4.13772 sec.
2^7	4	2×2	3.7974
2^8	16	4×4	4.37989
2^9	64	8×8	7.133

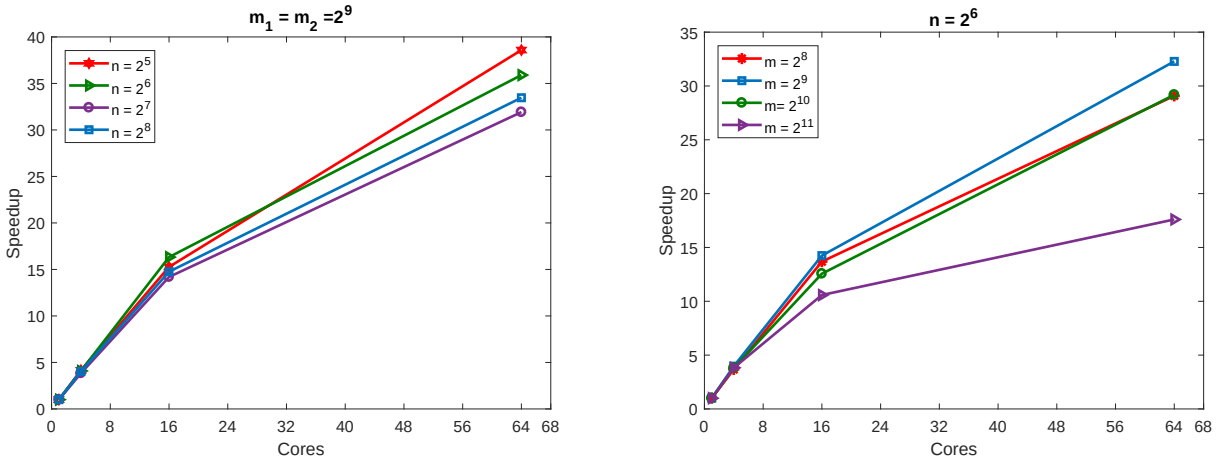

FIGURE 5. Strong speedup for various discretizations in space (left) and in time (right).

the number of cores, as illustrated in Figure 5. These results demonstrate the efficiency and scalability of our parallel implementation for solving the fractional subdiffusion problem.

5. CONCLUSION

In this paper, we introduce and analyze a high-order S3-FD method for solving time-fractional diffusion equations in one- and two-dimensional spatial domains. To tackle the computational challenges associated with solving the resulting large linear systems, especially in 2D cases, we have implemented efficient multigrid solvers. For 1D problems, we developed a cyclic reduction-based multigrid method with OpenMP parallelization for equidistant discretizations, utilizing a combination of block matrix symmetrization and preconditioned block-Jacobi iteration for non-equidistant cases. For the 2D domain, we extended our approach by implementing a parallel 2D multigrid solver using MPI programming, demonstrating the scalability and efficiency of the method for larger problems. Numerical results confirmed the effectiveness of the proposed algorithms in achieving high temporal accuracy while reducing computational costs. Future work will focus on extending the method to address nonlinear subdiffusion problems, exploring adaptive mesh refinement to manage boundary layers effectively and capture sharp gradients. Additionally, theoretical analysis will be conducted considering weak singular solutions.

CONFLICT OF INTEREST STATEMENT

The authors declare no potential conflict of interests.

REFERENCES

[1] A. A. Alikhanov, *A new difference scheme for the time fractional diffusion equation*, J. Comput. Phys., *280* (2015), 424–438.

[2] C. C. Douglas, G. Haase, and U. Langer, *A tutorial on elliptic PDE solvers and their parallelization*, Society for Industrial and Applied Mathematics, 2003.

[3] D. Gan, G. F. Zhang, and Z. Z. Liang, *An efficient preconditioner for linear systems arising from high-order accurate schemes of time fractional diffusion equations*, J. Appl. Math. Comput., *70* (2024), 5129–5151.

[4] G. H. Gao, Z. Z. Sun, and H. W. Zhang, *A new fractional numerical differentiation formula to approximate the Caputo fractional derivative and its applications*, J. Comput. Phys., *259* (2014), 33–50.

[5] X. Hu, C. Rodrigo, and F. J. Gaspar, *Using hierarchical matrices in the solution of the time-fractional heat equation by multigrid waveform relaxation*, J. Comput. Phys., *416* (2020), 109540.

[6] Y. Jiang and H. Bai, *Multigrid methods for time fractional conservation laws*, Numer. Algorithms, *97* (2024), 1301–1322.

[7] Y. Jiang, H. Chen, T. Sun, and C. Huang, *Efficient L1-ADI finite difference method for the two-dimensional nonlinear time-fractional diffusion equation*, Appl. Math. Comput., *471* (2024), 128609.

[8] A. Khibiev, A. Alikhanov, and C. Huang, *A second-order difference scheme for generalized time-fractional diffusion equation with smooth solutions*, Comput. Methods Appl. Math., *24* (2024), 101–117.

[9] Y. Li, L. Zikatanov, and C. Zuo, *A reduced conjugate gradient basis method for fractional diffusion*, SIAM J. Sci. Comput., *46* (2024), S68–S87.

[10] T. A. M. Langlands and B. I. Henry, *The accuracy and stability of an implicit solution method for the fractional diffusion equation*, J. Comput. Phys., *205* (2005), 719–736.

[11] K. Oldham and J. Spanier, *The fractional calculus: Theory and applications of differentiation and integration to arbitrary order*, Elsevier, 1974.

[12] J. Liu, H. Fu, and J. Zhang, *A QSC method for fractional subdiffusion equations with fractional boundary conditions and its application in parameters identification*, Math. Comput. Simul., *174* (2020), 153–174.

[13] S. Maji and S. Natesan, *Adaptive-grid technique for the numerical solution of a class of fractional boundary-value-problems*, Comput. Methods Differ. Equ., *12* (2024), 338–349.

[14] R. Mokhtari and F. Mostajeran, *A high order formula to approximate the Caputo fractional derivative*, Commun. Appl. Math. Comput., *2* (2020), 1–29.

[15] R. Mokhtari, M. Ramezani, and G. Haase, *Stability and convergence analyses of the FDM based on some L-type formulae for solving the subdiffusion equation*, Numer. Math. Theory Methods Appl., *14* (2021), 1–27.

[16] K. Pan, H. W. Sun, Y. Xu, and Y. Xu, *An efficient multigrid solver for two-dimensional spatial fractional diffusion equations with variable coefficients*, Appl. Math. Comput., *402* (2021), 126091.

[17] M. Ramezani and R. Mokhtari, *A novel high-order finite-difference method for the time-fractional diffusion equation with smooth/nonsmooth solutions*, Bull. Iran. Math. Soc., *48* (2022), 3987–4013.

[18] M. Ramezani, R. Mokhtari, and G. Haase, *Some high order formulae for approximating Caputo fractional derivatives*, Appl. Numer. Math., *153* (2020), 300–318.

[19] M. Ramezani, R. Mokhtari, and G. Haase, *Analysis of stability and convergence for L-type formulas combined with a spatial finite element method for solving subdiffusion problems*, Electron. Trans. Numer. Anal., (2022), 568–584.

[20] M. Ramezani, R. Mokhtari, and Y. Yan, *Correction of a high-Order numerical method for approximating time-fractional wave equation*, J. Sci. Comput., *100* (2024), 71.

[21] A. Singh, S. Kumar, and H. Ramos, *An efficient computational method based on exponential B-splines for a class of fractional sub-diffusion equations*, Comput. Methods Differ. Equ., *12* (2024), 719–740.

[22] M. Stynes, E. O'Riordan, and J. L. Gracia, *Error analysis of a finite difference method on graded meshes for a time-fractional diffusion equation*, SIAM J. Numer. Anal., *55*(2) (2017), 1057–1079.

[23] S. P. Tang and Y. M. Huang, *A fast preconditioning iterative method for solving the discretized second-order space-fractional advection–diffusion equations*, J. Comput. Appl. Math., *438* (2024), 115513.

[24] J. Tan and J. Liu, *An efficient numerical solver for anisotropic subdiffusion problems*, J. Comput. Appl. Math., *364* (2020), 112318.

[25] Y. Wang, N. An, and C. Huang, *Unconditional optimal error bounds of the fast nonuniform Alikhanov scheme for a nonlinear time-fractional biharmonic equation*, J. Appl. Math. Comput., *70* (2024), 4053–4071.

[26] Y. Xu, S. L. Lei, and H. W. Sun, *An efficient multigrid method with preconditioned smoother for two-dimensional anisotropic space-fractional diffusion equations*, Comput. Math. Appl., *124* (2022), 218–226.
[27] Y. Yan, M. Khan, and N. J. Ford, *An analysis of the modified L1 scheme for time-fractional partial differential equations with nonsmooth data*, SIAM J. Numer. Anal., *56* (2018), 210–227.
[28] S. Yeganeh, R. Mokhtari, and J. S. Hesthaven, *A local discontinuous Galerkin method for two-dimensional time fractional diffusion equations*, Commun. Appl. Math. Comput., *2* (2020), 689–709.
[29] Y. Zhao, Y. Zhang, F. Liu, I. Turner, Y. Tang, and V. Anh, *Convergence and superconvergence of a fully-discrete scheme for multi-term time fractional diffusion equations*, Comput. Math. Appl., *73* (2017), 1087–1099.
[30] Z. Zhou, S. Zhang, and W. Li, *The splitting characteristic finite difference domain decomposition scheme for solving time-fractional MIM nonlinear advection–diffusion equations*, J. Sci. Comput., *100* (2024), 49.