Check for updates

Stage-Parallel Implicit Runge–Kutta Methods Via Low-Rank Matrix Equation Corrections

Fabio Durastante[1] · Mariarosa Mazza[2] (ORCID)

Abstract

Implicit Runge–Kutta (IRK) methods are highly effective for solving stiff ordinary differential equations (ODEs) but can be computationally expensive for large-scale problems due to the need of solving coupled algebraic equations at each step. This study improves IRK efficiency by leveraging parallelism to decouple stage computations and reduce communication overhead, specifically we stably decouple a perturbed version of the stage system of equations and recover the exact solution by solving a Sylvester matrix equation with an explicitly known low-rank right-hand side. Two IRK families—symmetric methods and collocation methods—are analyzed, with extensions to nonlinear problems using a simplified Newton method. Implementation details, shared memory parallel code, and numerical examples, particularly for ODEs from spatially discretized PDEs, demonstrate the efficiency of the proposed IRK technique.

Keywords Runge–Kutta · Stage-Parallel Method · Low-rank Sylvester Matrix Equation

1 Introduction

Implicit Runge–Kutta (IRK) methods are a widely used technique for the numerical integration of ordinary differential equations (ODEs):

$$\begin{cases} M\mathbf{y}'(t) = f(\mathbf{y}(t), t), & t \in [0, T], \\ \mathbf{y}(0) = \mathbf{y}_0, \end{cases} \quad \begin{array}{l} \mathbf{y} : [0, T] \to \mathbb{R}^N, \\ f : \mathbb{R}^N \times [0, T] \to \mathbb{R}^N, \\ \mathbf{y}_0 \in \mathbb{R}^N, \ M \in \mathbb{R}^{N \times N}, \ N \in \mathbb{N}, \ T > 0. \end{array} \quad (1)$$

Unlike explicit methods, which compute the solution at the next time-step using only information from previous stages, an IRK method with s stages involves solving a system of

---

F. Durastante, M. Mazza: have contributed equally to this work.

✉ Mariarosa Mazza
  mariarosa.mazza@uniroma2.it

  Fabio Durastante
  fabio.durastante@unipi.it

[1] Department of Mathematics, University of Pisa, Largo Bruno Pontecorvo, 5, Pisa 56127, PI, Italy

[2] Department of Mathematics, University of Rome "Tor Vergata", Via Della Ricerca Scientifica, 1, Rome 00133, RM, Italy

algebraic equations at each step:

$$M\mathbf{y}^{(n+1)} = M\mathbf{y}^{(n)} + h \sum_{i=1}^{s} b_i \mathbf{k}_i^{(n)}, \quad n = 0, \ldots, n_t, \quad h = \frac{T}{n_t}, \quad n_t \in \mathbb{N}, \tag{2}$$

where the stages $\mathbf{k}_i^{(n)}$ for the approximation $\mathbf{y}^{(n+1)} \approx \mathbf{y}(t_{n+1})$ are given by

$$\mathbf{k}_i^{(n)} = f\left(\mathbf{y}^{(n)} + h \sum_{j=1}^{s} a_{i,j} \mathbf{k}_j^{(n)}, t_n + c_i \tau \right), \quad i = 1, \ldots, s, \tag{3}$$

with $t_n = nh$. The Runge–Kutta method is fully defined by the coefficients $a_{i,j}$, weights b_i, and nodes c_i, for $i, j = 1, \ldots, s$.

IRK methods are particularly suitable for stiff ODEs, as explicit methods can become unstable or require impractically small time-steps. By considering the interdependence of intermediate solution values within each time-step, IRK methods offer enhanced stability and accuracy. However, these benefits come at the cost of solving a computationally expensive system of algebraic equations at each step. This challenge is especially pronounced when IRK methods are applied to semi-discretized partial differential equations (PDEs), where the resulting systems can be of large scale.

To address this computational burden, IRK methods can benefit significantly from parallelism. In particular, parallelization across the stages enables simultaneous computation of multiple intermediate solutions, reducing overall computational time. This approach has been explored in the literature for various classes of RK methods, see, e.g., [1–8]. For instance, [3–5] propose a diagonal iteration method to solve the stage equations (3), which facilitates parallelization through the solution of block-diagonal systems. Similarly, [6] investigates the sparsity structure of the stage matrix $A = (a_{i,j})_{i,j}$ in (3) using concepts from sparse linear system solvers, decoupling stage dependencies as much as possible. For parallel explicit methods, including Parallel Explicit RK and RK-Nyström methods, we refer to [9–11]. Parallelism can also be leveraged by employing preconditioned distributed iterative solvers to independently solve the equations arising at each stage [7, 8, 12–14].

In this work, we aim to solve system (3) via the following two-step parallel approach:

1. **Perturbation and Decoupling:** In the first step, we perturb the coefficients $a_{i,j}$ in (3), decoupling the solution. This perturbation enables parallel computation of a modified set of stages. Of course, the resulting solution is affected by an error due to the perturbation.
2. **Correction via Sylvester Equation:** To recover the exact stages, we solve a Sylvester matrix equation with a known low-rank term [15]. This sequential step uses scalable and efficient Krylov projection methods.

Once these two steps are completed, we compute the approximation of the ODE solution at the new time-step using (2). The concept of splitting the procedure into two steps is inspired by the approach in [16], later expanded in [17], which were developed for constructing parallel-in-time methods for multistep linear methods. In this work, however, instead of using the perturbation to decouple the problem via the Fast Fourier Transform, we leverage the hidden structural properties of the stage matrix A. The resulting approach is particularly effective because the number of stages is independent of the problem size, hence we achieve the decoupling-diagonalization by computing the spectral decomposition of an $s \times s$ matrix in s^3 operations. This cost is smaller than either $n_t \log_2(n_t)$, incurred in using FFT-based strategy, or the n_t^3 cost of a direct approach. For all the three choices the time-stepping cost needs to be added. The construction supports two types of parallelism: distributed memory

parallelism, implemented using MPI subcommunicators, and shared memory parallelism. In this discussion, we will focus on the shared memory model of computation.

The remainder of this article is structured as follows: In Section 2, we introduce the general framework for a generic IRK method applied to linear problems. We then analyze and construct two prominent families of IRK methods: symmetric methods (Section 3), collocation methods (Section 4). In Section 5, we extend our analysis to nonlinear problems using a simplified Newton method for (1). Section 6 details the implementation and accompanying code and provides numerical examples, including ODEs arising from spatially discretized PDEs. Section 7 concludes the work with suggestions for future research.

Notation

We will denote vectors in bold face, matrices with capital letters (either latin, greek or calligraphic). The $\| \cdot \|$ denotes the two-norm, while $\| \cdot \|_F$ denotes the Frobenius norm. The vectors of the canonical basis will be denoted by $\mathbf{e}_j$, while $\mathbf{1}_s$ denotes the column vector of all ones of size s. The identity matrix of dimension m is denoted by I_m. We call vec $(\cdot)$ the operator stacking the columns of its matrix argument one below the other. The symbol "$\otimes$" denotes the Kronecker product.

2 Parallelism Across the Stages for Linear Problems

To succinctly represent the generic IRK method (2)-(3) and the parallelism across the stage idea we employ the Butcher tableau:

$$
\frac{\mathbf{c} \mid A}{\mathbf{b}^\top} =
\begin{array}{c|ccc}
c_1 & a_{1,1} & \dots & a_{1,s} \\
\vdots & \vdots & \ddots & \vdots \\
c_s & a_{s,1} & \dots & a_{s,s} \\
\hline
 & b_1 & \dots & b_s
\end{array}. \tag{4}
$$

To describe the stage-parallel version, we consider a case in which the ODE system is given by the semi-discretization in space of a PDE. We examine a linear PDE of the form

$$
\begin{cases}
\frac{\partial u}{\partial t} + \mathcal{L}[u] = s(\mathbf{x}, t), & (\mathbf{x}, t) \in \Omega \times (0, T], \quad \Omega \subseteq \mathbb{R}^d, \\
u(0) = u_0(\mathbf{x}), & , \quad d \in \mathbb{N}, \\
\mathcal{B}[u(\mathbf{x}, t)] = 0, & (\mathbf{x}, t) \in \partial\Omega \times [0, T],
\end{cases} \tag{5}
$$

where $\mathcal{L}$ and $\mathcal{B}$ are, respectively, a differential operator involving only spatial derivatives, and a boundary operator.

After a discretization scheme is applied to (5) we can rewrite it in the form (1) as

$$
\begin{cases}
M\mathbf{y}'(t) = -L\mathbf{y}(t) + \hat{\mathbf{f}}(t), & M, L \in \mathbb{R}^{N \times N}, \\
\mathbf{y}(0) = \mathbf{y}_0,
\end{cases} \tag{6}
$$

where the matrices M, L and the vector function $\hat{\mathbf{f}}$ collect the discretization of the operators $\mathcal{L}$, $\mathcal{B}$, and of the source term. Applying the generic IRK method with s stages in (4) yields

$$
\begin{cases}
M\mathbf{y}^{(n+1)} = M\mathbf{y}^{(n)} + h \sum_{i=1}^{s} b_i \mathbf{k}_i^{(n)}, & n = 0, \dots n_t, \\
M\mathbf{k}_i^{(n)} = -L\mathbf{y}^{(n)} - h \sum_{j=1}^{s} a_{i,j} L \mathbf{k}_j^{(n)} + \hat{\mathbf{f}}(t_n + c_i h), & i = 1, \dots, s,
\end{cases}
$$

where the coefficients b_i and c_i are the nodes and the weights of the Butcher tableau (4). By considering the matrix formulation and setting

$$K^{(n)} = [\mathbf{k}_1^{(n)}|\dots|\mathbf{k}_s^{(n)}] \in \mathbb{R}^{N \times s},$$

we can rewrite the second equation as

$$MK^{(n)} = -L\mathbf{y}^{(n)}\mathbf{1}_s^\top - hLK^{(n)}A^\top + F^{(n)}, \tag{7}$$

where

$$F^{(n)} = [\hat{\mathbf{f}}(t_n + c_1 h)|\dots|\hat{\mathbf{f}}(t_n + c_s h)] \in \mathbb{R}^{N \times s}.$$

If we rewrite (7) in a linear system format we find

$$(I_s \otimes M + hA \otimes L)\,\text{vec}\left(K^{(n)}\right) = -(\mathbf{1}_s \otimes L)\mathbf{y}^{(n)} + \text{vec}\left(F^{(n)}\right). \tag{8}$$

If we now assume that A is diagonalizable with $A = X\Lambda X^{-1}$ we restate the system as

$$(X \otimes I_N)\,(I_s \otimes M + h\Lambda \otimes L)\left(X^{-1} \otimes I_N\right)\text{vec}\left(K^{(n)}\right) = -(\mathbf{1}_s \otimes L)\mathbf{y}^{(n)} + \text{vec}\left(F^{(n)}\right),$$

which can then be solved [15, §4.3] as

$$\begin{cases} \mathbf{r} = -(X^{-1}\mathbf{1}_s \otimes L)\mathbf{y}^{(n)} + \left(X^{-1} \otimes I_N\right)\text{vec}\left(F^{(n)}\right), \\ (I_s \otimes M + h\Lambda \otimes L)\,\mathbf{z} = \mathbf{r}, \\ \text{vec}\left(K^{(n)}\right) = (X \otimes I_N)\,\mathbf{z}. \end{cases} \tag{9}$$

This requires the solution of decoupled linear systems to advance the method in the time-step, i.e., we first compute $\mathbf{r}$ and then solve the systems

$$\left[M + h\lambda_j L\right]\mathbf{z}^{(j)} = [\mathbf{r}]_{(j-1)N+1:jN}, \qquad j = 1, \dots, s, \tag{10}$$

where $[\,\cdot\,]_{p:q}$ extracts the entries from the pth to the qth, while finally computing the stages as

$$K^{(n)} = ZX^\top, \qquad Z = \left[\mathbf{z}^{(1)}|\cdots|\mathbf{z}^{(s)}\right].$$

To write this version we assumed that the Butcher Tableau matrix A was diagonalizable. But when is this the case, particularly when is the eigenvector matrix X well-conditioned? In general, this is not the case, and this has led to the development of several types of preconditioners for the system of stages (8) that exploit the real Schur decomposition [12, 13] or the SVD factorization [14] of the matrix A.

In the next two Sections 3 and 4, we will show how it is possible to perturb the matrix A via a low-rank update to get a diagonalizable matrix for symmetric and collocation IRK methods, respectively. In both cases the solution of the original problem from the perturbed one will be recovered by solving a Sylvester matrix equation.

3 Symmetric Runge–Kutta Schemes

An RK scheme is called *symmetric* if it remains invariant under the reflection t to $1 - t$ in the direction of integration [18], i.e., employing the notation in (4), if it is such that

$$\mathbf{c} + J\mathbf{c} = \mathbf{1}_s, \ \mathbf{b} = J\mathbf{b}, \ JAJ + A = \mathbf{1}_s \mathbf{b}^\top,$$

for J the exchange matrix, i.e., the matrix J such that $J[v_1, \dots, v_{s-1}, v_s]^\top = [v_s, v_{s-1}, \dots, v_1]^\top$; see [19, §5] for further information.

Table 1 Symmetric Runge–Kutta schemes with s stages and order $2s$

$s = 1$			$s = 2$		
$1/2$	$1/2$		$3-\sqrt{3}/6$	$1/4$	$(3-2\sqrt{3})/12$
	1		$3+\sqrt{3}/6$	$(3+2\sqrt{3})/12$	$1/4$
				$1/2$	$1/2$

$s = 3$			
$5-\sqrt{15}/10$	$5/36$	$(10-3\sqrt{15})/45$	$(25-6\sqrt{15})/180$
$1/2$	$(10+3\sqrt{15})/45$	$2/9$	$(10-3\sqrt{15})/45$
$(5+\sqrt{15})/10$	$(25+6\sqrt{15})/180$	$(10+3\sqrt{15})/45$	$5/36$
	$5/18$	$4/9$	$5/18$
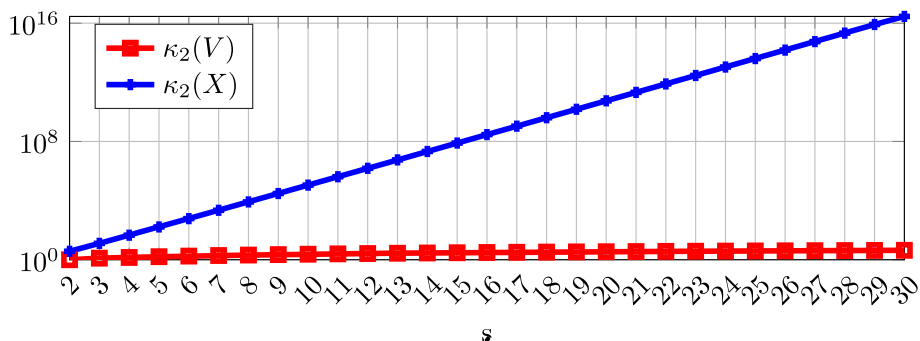
Fig. 1 We use the QZ algorithm to compute the left eigenvector basis for S, $SV = VD$, and A, $AX = X\Lambda$, for an increasing number of stages $s = 2, \ldots, 30$, and then measure their condition number

Example 1 Here we build an example of symmetric IRK scheme. To this end we take **b**, and the **c** as the weights and nodes of the Gauss-Legendre quadrature formula, while the $a_{i,j}$ need to satisfy the sum and moment conditions [20, §II.7 p.208] and can be computed as integrals of the Lagrange polynomials

$$a_{i,j} = \int_0^{c_i} \ell_j(c) \, dc, \qquad \ell_j(c) = \prod_{\substack{k=1 \\ k \neq j}}^{s} \frac{c - c_k}{c_j - c_k}. \tag{11}$$

The schemes for the low order values of s are given in Table 1, while for the scheme of a higher order the accompanying code[1] can be employed.

We recall that a matrix S is called centroskew [21] (oftentimes skew-centrosymmetric) if

$$JSJ = -S,$$

thus the matrix A of a symmetric RK scheme is a rank-one correction of a centroskew matrix, i.e., $S = A - \mathbf{1}_s \mathbf{b}^\top / 2$ with $JSJ = -S$ and $A = S + \mathbf{1}_s \mathbf{b}^\top / 2$. This choice causes an improvement in the conditioning of the eigenvector basis as s increases; see Fig. 1 in which we compare the conditioning of the diagonalization for the A of the IRK Gauss scheme and the one of the related S varying s.

To make use of this property, we start again from the matrix equation (7), and separate its solution into two steps, one for solving the system with a centroskew matrix and one for

---

[1] See the functions in the `schemes.jl` source code in the GitHub repository Cirdans-Home/SP_IRK.jl.

correcting it with a low-rank matrix, namely

$$
\begin{aligned}
MK^{(n)} + hLK^{(n)}A^\top &= -L\mathbf{y}^{(n)}\mathbf{1}_s^\top + F^{(n)} \\
M\hat{K}^{(n)} + hL\hat{K}^{(n)}S^\top &= -L\mathbf{y}^{(n)}\mathbf{1}_s^\top + F^{(n)}
\end{aligned}
\Bigg| - \tag{12}
$$

$$
ME^{(n)} + hLE^{(n)}A^\top = -\tfrac{h}{2}[L\hat{K}^{(n)}\mathbf{b}]\mathbf{1}_s^\top \Big| \tag{13}
$$

in which $E^{(n)} = K^{(n)} - \hat{K}^{(n)}$, and (12) can be solved through the diagonalization and parallelization procedure (9), while (13) is a matrix equation with a known term of rank 1 and for which, usually, the size of L and M is much larger than the size of A. We can employ a single-sided Krylov projection method with Galerkin condition [15] to solve this equation; see the pseudocode given in Algorithm 1.

Data: Matrices $Z \in \mathbb{R}^{N \times N}$, $R \in \mathbb{R}^{s \times s}$, vectors $\mathbf{u} \in \mathbb{R}^N$, $\mathbf{v} \in \mathbb{R}^s$, tolerance ϵ, maximum iterations
 $k_{\max}$
Result: Approximate solution E to $ZE + ER = \mathbf{u}\mathbf{v}^\top$

Initialize $V_1 = \frac{\mathbf{u}}{\|\mathbf{u}\|}$, $\beta = \|\mathbf{u}\|$;
for $j = 1, 2, \ldots, k_{\max} - 1$ **do**
    /* Krylov subspace generation   */
    Compute $\mathbf{w} = Z\mathbf{v}_j$;
    for $i = 1, 2, \ldots, j$ **do**
        Compute $h_{i,j} = \mathbf{v}_i^\top \mathbf{w}$;
        Orthogonalize: $\mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$;
    end
    Compute $h_{j+1,j} = \|\mathbf{w}\|$;
    if $h_{j+1,j} = 0$ **then**
        break;
    end
    Normalize: $\mathbf{v}_{j+1} = \frac{\mathbf{w}}{h_{j+1,j}}$;
    Update $V_{j+1} = [\mathbf{v}_1|\cdots|\mathbf{v}_{j+1}]$;
    Form the matrix $H_j = V_j^\top Z V_j$;
    /* Projected problem solution   */
    Solve the projected Sylvester equation $H_j Y_j + Y_j R = \beta e_1 \mathbf{v}^\top$;
    Compute the residual: $\rho_j = |h_{j+1,j}| \|\mathbf{e}_j^\top Y_j\|_F$;
    if $\rho_j < \epsilon \times \|\mathbf{u}\|_F \|\mathbf{v}\|_F$ **then**
        break;
    end
end
Compute the approximate solution $E_j = V_j Y_j$;

Algorithm 1: One-sided Krylov projection method for the matrix equation $ZE + ER = \mathbf{u}\mathbf{v}^\top$ with Arnoldi iteration

---

Note in particular that the expansion of the projection space in Algorithm 1 requires only a matrix-vector product. In the case of a semi-discretized PDE, this operation reduces to applying the mass matrix discretization followed by solving a system involving the spatial operator. Specifically, this corresponds to computing $Z = h^{-1}L^{-1}M$. A more robust variant of Algorithm 1 replaces the polynomial Krylov subspace $\mathcal{K}_k(Z, \mathbf{u}) = \text{span}\{\mathbf{u}, Z\mathbf{u}, \ldots, Z^{k-1}\mathbf{u}\}$, constructed with respect to the matrix Z and the vector $\mathbf{u}$, with a rational-type Krylov subspace [22]. While it is often possible to design projection spaces where the rational components have poles optimized for the problem at hand, a reliable and effective compromise

between usability and efficiency is offered by the extended Krylov subspace [23], defined as $\mathcal{E}_{2k}(Z, \mathbf{u}) = \text{span}\{\mathbf{u}, Z\mathbf{u}, Z^{-1}\mathbf{u}, Z^2\mathbf{u}, Z^{-2}\mathbf{u}, \ldots, Z^{k-1}\mathbf{u}, Z^{-k+1}\mathbf{u}\}$. This space captures information from both Z and Z^{-1}, and is expanded by alternating their application to the current basis vectors. In this case, the cost of expansion increases due to the need to solve systems involving the matrix Z. In our setting, this corresponds to solving linear systems with the mass matrix M, which is typically less computationally demanding than solving systems with the spatial operator L.

Remark 1 In the case where L is singular, the solution procedure becomes more involved. It requires working with the Krylov subspace generated by the matrix pencil (M, hL) and, at the projection level, employing a specialized dense solver; see, e.g., [24] for a discussion of the corresponding LAPACK routines.

The computational cost of the described procedure is expressed as $s \times c_L + c_S$, where c_L represents the cost of solving the linear systems involved in the block diagonalization of (12), and c_S corresponds to the cost of solving the Sylvester equation in (13). By employing n_p processes to simultaneously solve the step in (12), the cost is reduced to $\frac{s}{n_p} \times c_L + c_S$, with an additional communication overhead arising from parallel loop reduction.

Comparing this approach with methods that rely on constructing a block-diagonalizable preconditioner for solving the entire system in (8) within a Krylov method, we observe a key advantage: this approach avoids the need to construct a basis for a space of size $s \times N$. Instead, it operates directly on matrices and spaces of size N, thus reducing memory and computational complexity.

4 Collocation Methods

In RK collocation methods, the solution over a single step is approximated by a polynomial, and the coefficients of this polynomial are determined by enforcing that the differential equation holds at specific points within the interval, called collocation points. The accuracy and stability of the method depend on the choice and distribution of these points. A special subset of these methods is the Gauss, Radau, and Lobatto collocation methods, which correspond to the Gauss–Legendre—the method introduced in Example 1—Radau, and Lobatto quadrature points, respectively. This corresponds to the selection of the $\mathbf{b}$ and $\mathbf{c}$ as the weights and nodes of the Gauss–Legendre, Gauss–Radau, and Gauss–Lobatto quadrature formulas, respectively. For all these methods we can construct a representation in which the Butcher tableau matrix A, built as in (11), is written as a normal matrix plus a low-rank correction, so that we can exploit the idea we explored for the case of symmetric RK methods. In the following subsection we build such a decomposition by means of the $\mathcal{W}$ transform.

4.1 The $\mathcal{W}$ Transformations

Let $\{P_\ell\}_{\ell \geq 0}$ be the sequence of scaled and shifted degree ℓ Legendre polynomials on the interval $[0, 1]$ [25, P. 27], i.e., the sequence of orthonormal polynomials with respect to the $\omega(x) = 1$ measure on the $[0, 1]$ interval

$$\int_0^1 P_p(x) P_q(x)\, \omega(x)\, \mathrm{d}x = \delta_{p,q} = \begin{cases} 1, & p = q, \\ 0, & p \neq q, \end{cases}$$

and let $\{c_i\}_{i=1}^s$ be the nodes of the related Gauss-Legendre formula. We define the $\mathcal{W}_s \in \mathbb{R}^{s \times s}$ matrix

$$(\mathcal{W}_s)_{i,j} = w_{i,j} = P_{j-1}(c_i), \quad i = 1, \ldots, s, \qquad j = 1, \ldots, s. \tag{14}$$

Then the following result for the Gauss method of order $2s$ holds true.

Theorem 1 ([26, Theorem 5.6]) *Let A be the coefficient matrix for the Gauss method of order $2s$ from (11) and $\mathcal{W}_s$ the matrix in (14), then*

$$\mathcal{W}_s^\top A \mathcal{W}_s = \begin{bmatrix} \frac{1}{2} & -\xi_1 & & & \\ \xi_1 & 0 & -\xi_2 & & \\ & \xi_2 & \ddots & \ddots & \\ & & \ddots & 0 & -\xi_{s-1} \\ & & & \xi_{s-1} & 0 \end{bmatrix} = X_s, \quad \xi_k = \frac{1}{2\sqrt{4k^2 - 1}}.$$

From Theorem 1 it follows that we can use the matrix $\mathcal{W}_s$ to transform problem (7) into a form in which we have in place of the matrix A the matrix X_s that is written as the sum of an antisymmetric matrix—and therefore normal—and a rank 1 correction. While this is not a significant addition in the case of the Gauss method, for which we can use the construction discussed in Section 3, this allows us to obtain effective writing for our diagonalization plus correction solution approach in other cases as well.

Consider an IRK method with tableau (4), and let us define $\mathcal{B} = \text{diag}(\mathbf{b})$. Then the matrix [26, pp.77–84]

$$X_s = \mathcal{W}_s^\top \mathcal{B} A \mathcal{W}_s = \begin{bmatrix} \frac{1}{2} & -\xi_1 & & & & \\ \xi_1 & 0 & -\xi_2 & & & \\ & \xi_2 & \ddots & \ddots & & \\ & & \ddots & 0 & -\xi_{s-2} & \\ & & & \xi_{s-2} & 0 & \zeta_{s-1,s} \\ & & & & \zeta_{s,s-1} & \zeta_{s,s} \end{bmatrix}, \quad \xi_k = \frac{1}{2\sqrt{4k^2 - 1}}. \tag{15}$$

The three coefficients $\zeta_{s-1,s}$, $\zeta_{s,s-1}$ and $\zeta_{s,s}$ are given in Table 2 for different collocation methods. Moreover, we define

$$\mathcal{D} = \mathcal{W}_s^\top \mathcal{B} \mathcal{W}_s = \text{diag}(1, 1, \ldots, 1, d_s), \tag{16}$$

for which the d_s is also given in Table 2.

Let us do the change of variables $Z^{(n)} = K^{(n)} \mathcal{W}_s^{-T}$ for the equation of stages (7), and multiply on the right the equation by $\mathcal{B} \mathcal{W}_s$, hence

$$M Z^{(n)} \mathcal{W}_s^\top \mathcal{B} \mathcal{W}_s + h L Z^{(n)} \mathcal{W}_s^\top A^\top \mathcal{B} \mathcal{W}_s = -L y^{(n)} \mathbf{b}^\top \mathcal{W}_s + F^{(n)} \mathcal{B} \mathcal{W}_s,$$

giving the transformed generalized Sylvester matrix-equation

$$\begin{cases} M Z^{(n)} \mathcal{D} + h L Z^{(n)} X_s^\top = -L y^{(n)} \mathbf{b}^\top \mathcal{W}_s + F^{(n)} \mathcal{B} \mathcal{W}_s, \\ K^{(n)} = Z^{(n)} \mathcal{W}_s^\top. \end{cases}$$

To apply the same method adopted for the symmetric schemes in (12)-(13), we need to explicitly define a low-rank transformation that diagonalizes the system, i.e., makes X_s

Table 2 Coefficients of the $\mathcal{W}_s$-transformed Butcher tableau (15)-(16) for different IRK methods. The last column gives the rank of the correction needed to write X_s as a skewsymmetric and hence normal matrix. Whenever $d_s \neq 1$, an additional rank 1 correction is needed to rewrite $\mathcal{D}$ as the identity

Method	$\zeta_{s,s-1}$	$\zeta_{s-1,s}$	$\zeta_{s,s}$	d_s	Rank
Gauss	ξ_{s-1}	$-\xi_{s-1}$	0	1	1
Radau IA	ξ_{s-1}	$-\xi_{s-1}$	$1/4s-2$	1	2
Radau IIA	ξ_{s-1}	$-\xi_{s-1}$	$1/4s-2$	1	2
Lobatto IIIA	$\frac{2s-1}{s-1}\xi_{s-1}$	0	0	$\frac{2s-1}{s-1}$	2
Lobatto IIIB	0	$-\frac{2s-1}{s-1}\xi_{s-1}$	0	$\frac{2s-1}{s-1}$	2
Lobatto IIIC	$\frac{2s-1}{s-1}\xi_{s-1}$	$-\frac{2s-1}{s-1}\xi_{s-1}$	$\frac{2s-1}{(2s-2)(s-1)}$	$\frac{2s-1}{s-1}$	2
Lobatto IIIC*	$\frac{2s-1}{s-1}\xi_{s-1}$	$-\frac{2s-1}{s-1}\xi_{s-1}$	$-\frac{2s-1}{(2s-2)(s-1)}$	$\frac{2s-1}{s-1}$	2
Lobatto IIID	$\frac{2s-1}{s-1}\xi_{s-1}$	$-\frac{2s-1}{s-1}\xi_{s-1}$	0	$\frac{2s-1}{s-1}$	1

skew-symmetric, and ensures the diagonal $\mathcal{D}$ is corrected to the identity. This is achieved by setting[2]:

$$
\hat{X}_s = X_s + \begin{bmatrix} -1/2 & & & 0 \\ 0 & & & \vdots \\ \vdots & & 0 & \\ 0 & -\zeta_{s,s-1} - \zeta_{s-1,s} \\ 0 & -\zeta_{s,s} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \vdots \\ \vdots & 0 \\ 0 & 1 \end{bmatrix}^\top = X_s + C_1 C_2^\top,
$$

$$
I_s = \mathcal{D} + (1 - d_{s,s})\mathbf{e}_s\mathbf{e}_s^\top.
$$

We solve the block-diagonal linear system with $\hat{X}_s$ in place of X_s and I_s in place of $\mathcal{D}$:

$$
\left(I_s \otimes M + h\hat{X}_s \otimes L \right) \mathrm{vec}\left(\hat{Z}^{(n)} \right) = \mathrm{vec}\left(-L\mathbf{y}^{(n)}\mathbf{b}^\top \mathcal{W}_s + F^{(n)}\mathcal{B}\mathcal{W}_s \right),
$$

where $\hat{X}_s$ is diagonalizable as $\hat{X}_s = Q\Lambda Q^H$ with $Q^H Q = I_s$. Consequently, we can rewrite:

$$
(Q \otimes I_N)(I_s \otimes M + h\Lambda \otimes L)(Q^H \otimes I_N) \mathrm{vec}\left(\hat{Z}^{(n)} \right) = \mathrm{vec}\left(-L\mathbf{y}^{(n)}\mathbf{b}^\top \mathcal{W}_s + F^{(n)}\mathcal{B}\mathcal{W}_s \right).
$$

The correction matrix equation can then be written as:

$$
\begin{array}{r|}
MZ^{(n)}\mathcal{D} + hLZ^{(n)}X_s^\top = -L\mathbf{y}^{(n)}\mathbf{b}^\top \mathcal{W}_s + F^{(n)}\mathcal{B}\mathcal{W}_s & - \\
M\hat{Z}^{(n)} + hL\hat{Z}^{(n)}\hat{X}_s^\top = -L\mathbf{y}^{(n)}\mathbf{b}^\top \mathcal{W}_s + F^{(n)}\mathcal{B}\mathcal{W}_s & = \\
\hline
ME^{(n)}\mathcal{D} + hLE^{(n)}X_s^\top = (1 - d_{s,s})M\hat{Z}^{(n)}\mathbf{e}_s\mathbf{e}_s^\top + hL\hat{Z}^{(n)}C_2 C_1^\top &
\end{array}
\qquad (17)
$$

where $E^{(n)} = Z^{(n)} - \hat{Z}^{(n)}$. The right-hand side is now a matrix of rank two or three, depending on the linear independence of the vectors forming the three dyads.

Remark 2 To obtain the minimal rank version of the right-hand side of (17), it is sufficient to collect the three vectors together and apply the Gram-Schmidt procedure to the columns and rows respectively to discard the only possible linearly dependent vector. To solve the

---

[2] For all cases in Table 2 except Lobatto IIIB for which we substitute $C_2 = [\mathbf{e}_1, \mathbf{e}_s]$ with $C_2 = [\mathbf{e}_1, \mathbf{e}_{s-1}]$, observe also that for the Lobatto IIIA and IIIB $\zeta_{s,s} = 0$. This choice avoids a zero eigenvalue in $\hat{X}_s$.
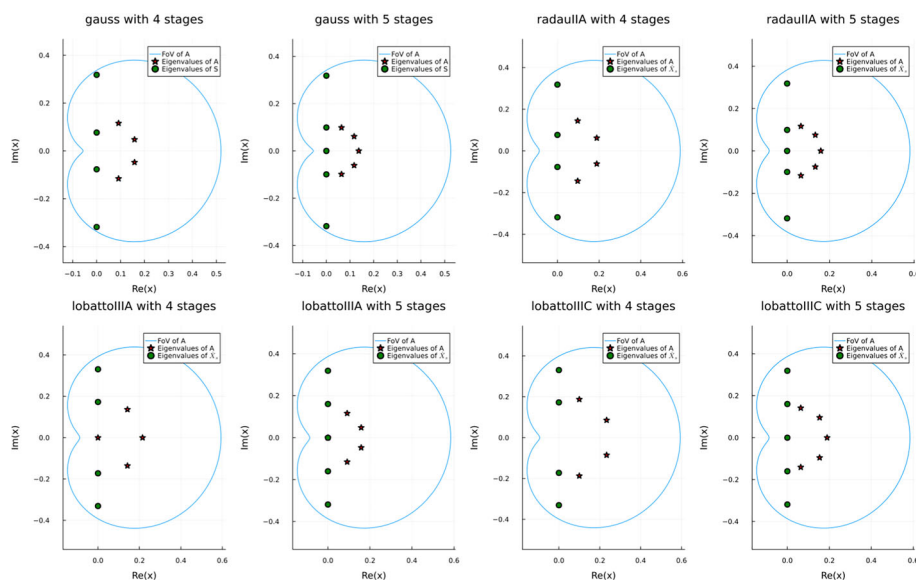
Fig. 2 Visualization of the eigenvalues ($\star$) and the field of values of the Butcher matrix A for various Runge–Kutta schemes, along with the eigenvalues of the perturbed matrices S and $\hat{X}_s$ ($\circ$) for stage numbers $s = 4, 5$

matrix equation (17) it is then possible to apply the Algorithm 1 in which the Krylov space is replaced by the block Krylov space, i.e., block Arnoldi [27, Algorithm 6.22], with block size equal to the rank of the right-hand side.

Remark 3 Note that for an odd number of stages s, the center-skew matrix S or the skew-symmetric matrix $\hat{X}_s$ will always have a zero eigenvalue by construction; see the examples in Fig. 2. When the mass matrix is either the identity matrix or a more general non-singular matrix, choosing an odd number of stages does not pose any issues for the solvability of the shifted systems.

However, if the considered case involves a differential-algebraic equation (DAE) or a singular mass matrix, one of the systems to be solved using this approach will have a non trivial kernel.

5 The Nonlinear Case: the Simplified Newton Method

In the nonlinear case, the differential equation in (5) is substituted by

$$\begin{cases} \frac{\partial u}{\partial t} + \Theta[u, t] = 0, & (\mathbf{x}, t) \in \Omega \times (0, T], \quad \Omega \subseteq \mathbb{R}^d, \\ u(0) = u_0(\mathbf{x}), & , \quad d \in \mathbb{N}, \qquad (18) \\ \mathcal{B}[u(\mathbf{x}, t)] = 0, & (\mathbf{x}, t) \in \partial\Omega \times [0, T], \end{cases}$$

for $\Theta[\cdot, \cdot]$ a—possibly non autonomous—nonlinear operator involving partial derivatives of the function u. After a suitable space discretization, we come back to the case

$$\begin{cases} M\mathbf{u}'(t) = -\Theta(\mathbf{u}, t), & t \in (0, T], \\ \mathbf{u}(0) = \mathbf{u}, \end{cases} \qquad M \in \mathbb{R}^{N \times N}, \ \Theta : \mathbb{R}^N \times [0, T] \to \mathbb{R}^N.$$

To solve the stage equations in (3) we need to apply a suitable linearization procedure to it, typically a Newton-like or a Picard iteration. In the Newton case, we start by introducing the auxiliary functions

$$\hat{\Theta}_i(\mathbf{k}_i^{(n)}) \equiv M\mathbf{k}_i^{(n)} - M\mathbf{u}^{(n)} + h \sum_{j=1}^{s} a_{i,j} \Theta\left(\mathbf{k}_j^{(n)}, t_n + c_j h\right) = 0, \qquad i = 1, \ldots, s, \quad (19)$$

and thus the vector function $\hat{\Theta}(\mathbf{k}, t) = [\hat{\Theta}_1, \ldots, \hat{\Theta}_s]^\top$. From the solution of (19) we obtain the expression for the stages with which we advance of a time-step by

$$M\mathbf{u}^{(n+1)} = M\mathbf{u}^{(n)} - h \sum_{i=1}^{s} b_i \Theta(\mathbf{k}_j^{(n)}, t_n + c_i h). \qquad (20)$$

The full Newton iteration for the approximation of the zeros of (19) is given by

$$\begin{cases} \text{Given } \kappa^0 = [\mathbf{k}_1^{(n),0}, \ldots, \mathbf{k}_s^{(n),0}], & \\ J_{\hat{\Theta}}(\kappa^p)\mathbf{d}^p = \hat{\Theta}(\kappa^p), & p \geq 0 \\ \kappa^{p+1} = \kappa^p - \mathbf{d}^p, & \end{cases} \qquad (21)$$

for $J_{\hat{\Theta}}(\kappa^p)$ a suitable approximation of the Jacobian of the vector function $\hat{\Theta}$ at the current step, and which can be expressed as

$$J_{\hat{\Theta}}(\kappa^p) = \left(A^{-1} \otimes M + h \begin{bmatrix} J_{\Theta}^{(1)} & & \\ & \ddots & \\ & & J_{\Theta}^{(s)} \end{bmatrix} \right) (A \otimes I_N)$$

where $J_{\Theta}^{(i)}$, $i = 1, \ldots, s$, is the Jacobian of the Θ function with respect to the $\mathbf{k}_i^{(n)}$ variables.

To be able to return to the linear case, we must assume that all Jacobian matrices are computed for the same time-step, that is, that the system is autonomous, or close to being so; i.e., we build what is called a *simplified Newton* method [7, 28]. In such a case we build the single matrix J_Θ for all the stages and use

$$J_{\hat{\Theta}} = (I_s \otimes M + hA \otimes J_\Theta),$$

in (21); e.g., as in [29], we can select J_Θ as the arithmetic average of the $J_\Theta^{(i)}$, $i = 1, \ldots, s$. Now, to solve for the Newton direction we apply the correction procedure. In summary, to advance from time-step t_n to time-step t_{n+1} we use (20), where $\mathbf{k}_i^{(n)}$ is obtained from the simplified Newton method applied to $\hat{\Theta}(\mathbf{k}_i^{(n)}) = 0$, i.e., we select $\kappa^0 = [\mathbf{k}_1^{(n),0}, \ldots, \mathbf{k}_s^{(n),0}]$ and solve

$$J_{\hat{\Theta}}\mathbf{d}^p = \hat{\Theta}(\kappa^p),$$

$$(I_s \otimes M + hA \otimes J_\Theta)\, \mathbf{d}^p = \hat{\Theta}(\kappa^p).$$

To simplify the discussion, we assume working with a symmetric scheme as in Section 3, and solve instead

$$(I_s \otimes M + hS \otimes J_\Theta)\, \hat{\mathbf{d}}^p = \hat{\Theta}(\kappa^p),$$

for $S = A - \mathbf{1}_s \mathbf{b}^\top / 2$, by block-diagonalization

$$(X \otimes I_N)\, (I_s \otimes M + h\Lambda \otimes J_\Theta)\, (X^{-1} \otimes I_N) \hat{\mathbf{d}}^p = \hat{\Theta}(\kappa^p),$$

$$(I_s \otimes M + h\Lambda \otimes J_\Theta)\,\mathbf{z} = (X^{-1} \otimes I_N)\hat{\mathbf{\Theta}}(\boldsymbol{\kappa}^p), \tag{22}$$
$$\hat{\mathbf{d}}^p = (X \otimes I_N)\mathbf{z}.$$

If we denote by Δ_p, $\hat{\Delta}_p$ and θ the matricization of size $N \times s$ of the vectors $\mathbf{d}^p$, $\hat{\mathbf{d}}^p$ and $\hat{\mathbf{\Theta}}(\boldsymbol{\kappa}^p)$ respectively, then we recover the Newton direction $\mathbf{d}^p$ via the solution of the Sylvester equation

$$
\begin{array}{r|l}
M\Delta_p + hJ_\Theta\Delta_p A^\top = \theta & - \\
M\hat{\Delta}_p + hJ_\Theta\hat{\Delta}_p S^\top = \theta & = \\
\hline
ME_p + hJ_\Theta E_p A^\top = -\frac{h}{2}\left(J_\Theta\hat{\Delta}_p\mathbf{b}\right)\mathbf{1}_s^\top &
\end{array}, \tag{23}
$$

with $E_p = \Delta_p - \hat{\Delta}_p$. Hence the Newton update is computed as

$$\boldsymbol{\kappa}^{p+1} = \boldsymbol{\kappa}^p - \mathrm{vec}\left(\hat{\Delta}_p + E_p\right).$$

The procedure is analogous for the collocation methods discussed in Section 4.

6 Implementation Details and Numerical Examples

The approach discussed in this work is implemented in Julia and available at the GitHub repository Cirdans-Home/SP_IRK.jl. Our implementation takes full advantage of Julia's shared-memory parallelism to accelerate critical components of the computation. One of the central tasks is the parallel computation of LU factorizations required for solving the linear systems that appear in equations (10), and (22). This is accomplished by concurrently factorizing matrices of the form $M + h\lambda_j L$—respectively $M + h\lambda_j J_\Theta$ and J_Θ a suitable approximation of the Jacobian. The operation is encoded as follows:

```
1  lhs = fetch.([Threads.@spawn factorize(Mass + h*ev[j]*L) for
       j in 1:s])
```

In this snippet, the LU factorization of each matrix is computed in a separate asynchronous task using `Threads.@spawn`, and the resulting factorizations are aggregated into the array `lhs` via `fetch`. For autonomous linear problems, where the matrix remains unchanged over time, this factorization is computed just once and then reused for all time-steps. This step is efficiently handled by the external UMFPACK library [30] interfaced through Julia.

In addition to the factorization, the code also exploits multi-threading for assembling the right-hand side (RHS) of the linear systems and for applying the computed factorizations to obtain the solutions. The assembly of the RHS involves evaluating the ODE vector field at modified time points, and the corresponding loop is parallelized using `Threads.@threads` as shown below for the linear case (6):

```
1  Threads.@threads for j in eachindex(c)
2      @inbounds F[:,j] = f(ti + c[j]*h)
3  end
```

Here, the use of `@inbounds` eliminates array bounds checking, thereby reducing overhead since the dimensions of F have already been verified. Following the assembly, the precomputed LU factorizations are applied to solve the linear systems in parallel:

```
1  Threads.@threads for j in eachindex(lhs)
```

```
2        @inbounds K[:,j] = lhs[j] \ r[:,j]
3    end
```

Each thread independently applies the corresponding factorization from `lhs` to the RHS vector segment in `r`, storing the resulting solution in `K` while again employing `@inbounds` for performance optimization. It is important to note that while multi-threading significantly accelerates the computation by exploiting parallel execution, there is an inherent overhead associated with creating and managing the threaded environment. This overhead stems from the cost of thread creation, task scheduling, which may become non-negligible, particularly for smaller problem sizes or less intensive computational tasks. Therefore, when applying parallelization, it is critical to balance these overhead costs against the performance gains to ensure that the overall efficiency is maintained. It is also worth mentioning that another bottleneck for parallelism arises in the correction step following the solution of the linear systems, where the Sylvester matrix equation (13)—respectively (17) and (23)—is solved. This step, due to its inherent sequential nature, limits further speedups that might be obtained through parallelization. Furthermore, the maximum number of threads that can be effectively utilized is constrained by the number s of stages in the underlying IRK method, as each stage represents an independent computation whose total number sets the upper bound for concurrent execution.

6.1 Numerical Examples

We examine two differential problems to demonstrate the effective application of the strategy discussed earlier. Our focus lies on a framework that addresses PDEs using the longitudinal method of lines. This approach begins by discretizing the spatial domain, converting the PDE into a system of ODEs, which can be linear or nonlinear. These ODEs are subsequently solved using the proposed integration methods. The numerical experiments contained in this section are conducted on a single node of the Toeplitz cluster, located at the Green Data Center of the University of Pisa. The node is equipped with an AMD EPYC 7763 64-Core Processor CPU, featuring 2 threads per core, 64 cores per socket, 2 sockets, and a total of 2 TB of RAM, which makes it particularly suitable for testing the multi-threaded programming model approach discussed above.

We analyze two specific cases: a linear heat transport problem in Section 6.1.1, and a nonlinear wave problem in Section 6.1.2. These examples test the methods introduced in Section 4 and 5, respectively.

6.1.1 An Heat Transport Problem

We solve a finite element discretization of the heat equation on the 3D domain depicted in Fig. 3 with mixed Dirichlet and Neumann boundary conditions. Dirichlet boundary conditions are applied on the outer sides of the holed prism $\Gamma_{\mathfrak{D}}$, while non-homogeneous Neumann conditions are applied to the three internal boundaries of the holes (Γ_α for the triangular hole, Γ_γ for the square hole, and Γ_β for the circular hole). Homogeneous Neumann boundary conditions are applied in the remaining portion of the boundary Γ_W—hence $\Gamma_{\mathfrak{N}} = \Gamma_\alpha \cup \Gamma_\gamma \cup \Gamma_\beta \cup \Gamma_W$:
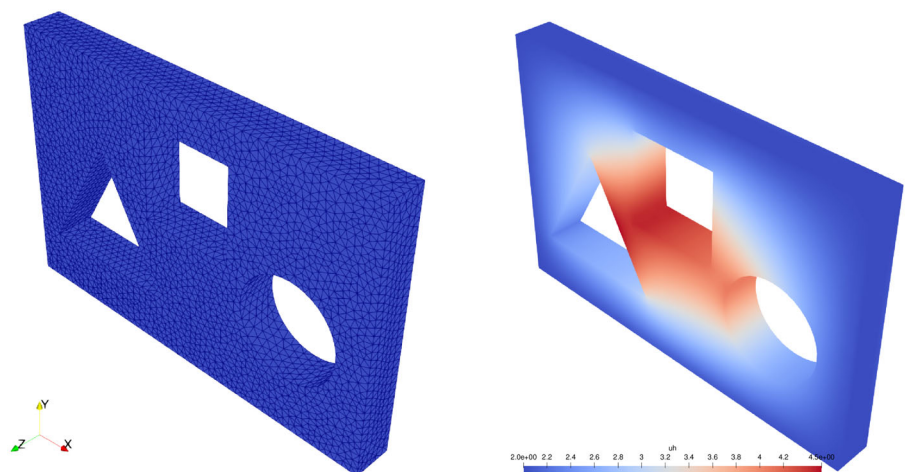
(a) Domain with an example mesh for the heat transport problem (24).

(b) Solution at $T = 1$ for the choice of parameters in (25).

Fig. 3 Heat transport problem from Section 6.1.1

$$\begin{cases} \frac{\partial u}{\partial t} - \nabla^2 u = f(\mathbf{x}, t), & \text{in } \Omega \times (0, T], \\ u = g_1(\mathbf{x}), & \text{on } \Gamma_{\mathfrak{D}} \times (0, T], \\ \nabla u \cdot \vec{\mathbf{n}} = g_2(\mathbf{x}), & \text{on } \Gamma_{\mathfrak{N}} \times (0, T], \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}), \end{cases} \quad (24)$$

where $\vec{\mathbf{n}}$ is the outward pointing normal with respect to the Neumann boundaries; see Fig. 3a for a depiction of the domain.

To arrive at the semi-discrete formulation in (6) we employ the Finite Element Gridap.jl library [31, 32] and select

$$f(\mathbf{x}, t) = (1.5 - x_1)(1 - x_2)(1 - x_3) + \frac{1}{2} \sin(2\pi t), \quad g_1(\mathbf{x}) = 2, \quad g_2(\mathbf{x}) = 3. \quad (25)$$

For all the following examples we set $T = 1$, and $h = 10^{-2}$; see Fig. 3b for a depiction of the solution.

Fig. 4a illustrates the elapsed time for the solution of the Gauss IRK method using a stage-parallel approach. As the number of threads increases, a noticeable reduction in elapsed time is observed, highlighting the effectiveness of parallelizing across stages. However, the rate of improvement diminishes beyond a certain point, suggesting that the overhead caused by the handling of the multi-thread environment in Julia and the solution of the associated matrix equation starts to dominate at higher thread counts. This behavior is further quantified in Fig. 4b, which shows the parallel speedup achieved. The speedup increases significantly to around $3\times$ with the number of threads, but again, exhibits sublinear scaling as the number of computing units grows. To better investigate the numerical behavior of the procedure, in Fig. 5 we focus on the solve time for the number of stages equal to the number of threads, i.e., to the diagonal of Fig. 4a. The left axis reports the total elapsed time measured in seconds, while the right axis shows the Krylov iteration count. We observe that the number of iterations performed by the polynomial Krylov method described in Algorithm 1 grows with s. Indeed,

(a) Elapsed time (s)   (b) Speedup
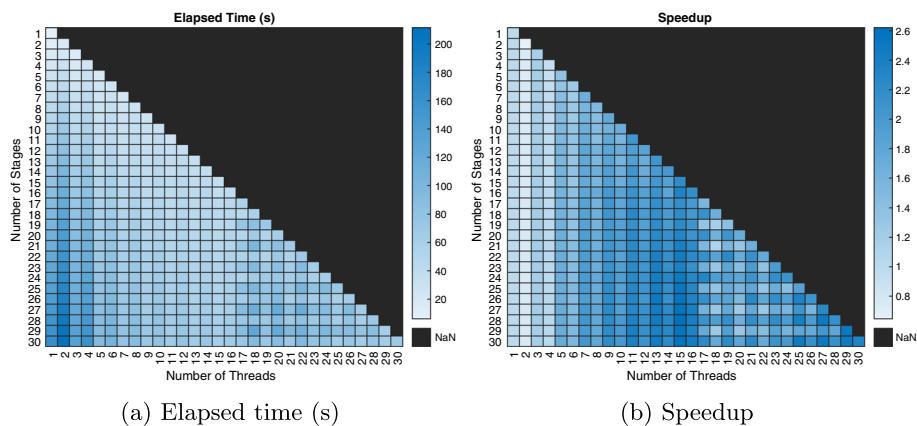
Fig. 4 Elapsed time and speedup for the heat transport problem in Section 6.1.1 obtained by employing the stage-parallel procedure with the Gauss symmetric scheme and number of stages and threads running from 1 to 30
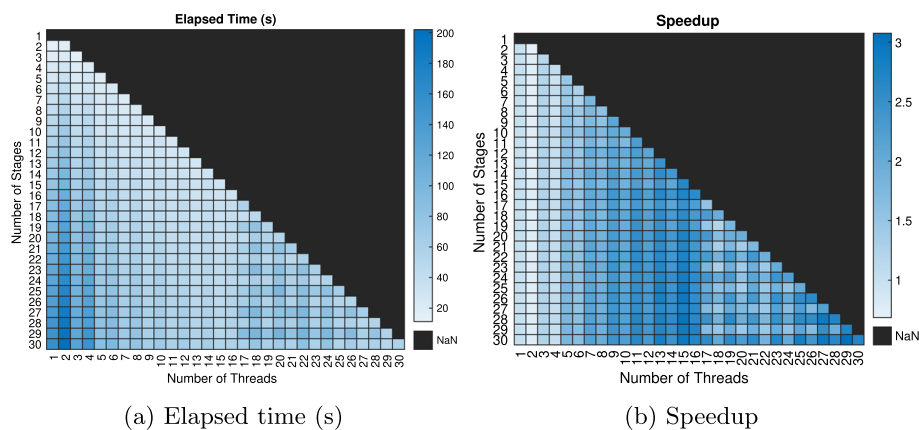

s : number of stages / threads
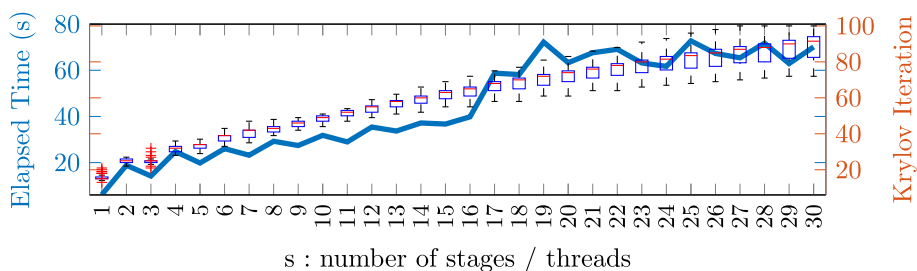
Fig. 5 Elapsed time and iteration scaling with respect to the number of stages and equal number of threads for the solution of the Sylvester equation (13) in the case of the Gauss scheme, implemented as a symmetric scheme

if we interpret the scaling as a weak scaling with s, we observe that increasing the problem size from $2N$ to $30N$ shows $3.7\times$ increase in computational time. Overall, the results confirm that the stage-parallel strategy provides a practical mean to enhance computational efficiency for time integration in stiff systems.

We repeat the experiment using the same configuration as before, but replacing the Gauss method with the Radau IIA method, which is based on its interpretation as a collocation method (Section 4). The elapsed time and computed speedup are presented in Fig. 6, and can be compared with the results shown in Fig. 4. The behavior in both cases is analogous, with the Radau IIA scheme[3] achieving a slightly higher maximum speedup compared to the Gauss method. This improvement can be attributed to the more efficient solution of the Sylvester equation (17) in the Radau IIA case, as opposed to the Gauss case where the alternative form (13) was used instead. See Fig. 7 for the Radau IIA results, and compare them with the corresponding Fig. 5 for the Gauss case.

---

[3] Note that for Radau IIA the scheme with $s = 1$ is meaningless.

(a) Elapsed time (s)   (b) Speedup

Fig. 6 Elapsed time and speedup for the heat transport problem in Section 6.1.1 obtained by employing the stage-parallel procedure with the Radau IIA collocation scheme and number of stages running from 2 to 30 and threads running from 1 to 30

s : number of stages / threads
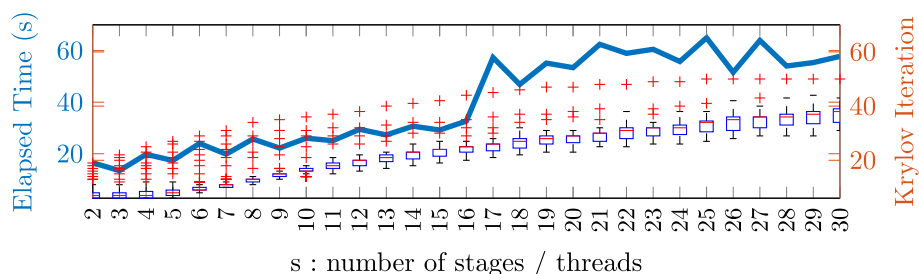
Fig. 7 Elapsed time and iteration scaling with respect to the number of stages and equal number of threads for the solution of the Sylvester equation (17) in the case of the Radau IIA scheme, implemented as a collocation scheme

6.1.2 A Nonlinear Wave Equation

We next consider the nonlinear wave equation from [29, §5.2],

$$
\begin{cases}
\partial_{tt}u = \partial_{xx}u + \beta u^2, & (x, t) \in \left(-\frac{1}{2}, \frac{1}{2}\right) \times (0, 1), \\
u\left(-\frac{1}{2}, t\right) = u\left(\frac{1}{2}, t\right) = 0, & t \in (0, 1), \\
u(x, 0) = e^{-100x^2}, \quad u_t(x, 0) = 0, & x \in \left(-\frac{1}{2}, \frac{1}{2}\right).
\end{cases}
\tag{26}
$$

The parameter $\beta > 0$ represents the strength of the nonlinearity. We semi-discretize in space the equation by employing centered finite differences as in [29] with a mesh-size Δx, which results in the system of ODEs of the form (18)

$$
\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} - \begin{bmatrix} O & I \\ B & O \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} - \begin{bmatrix} 0 \\ \beta \mathbf{u}^2 \end{bmatrix} = \mathbf{0}, \quad B = \frac{1}{\Delta x^2} \begin{bmatrix} 1 & 0 & & & 0 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 0 & & & 0 & 1 \end{bmatrix},
$$

(a) $u(x,t)$, $x \in [1/2, 1/2]$, $t \in [0,1]$ (b) $u_x(x,t)$, $x \in [1/2, 1/2]$, $t \in [0,1]$
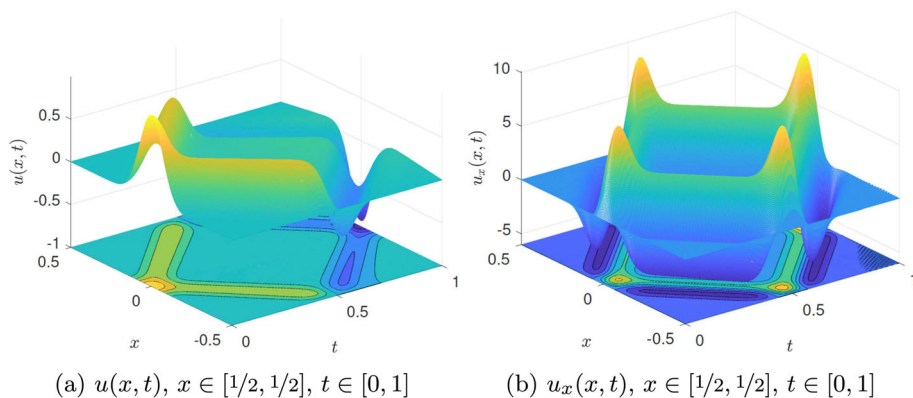
**Fig. 8** Solution of the nonlinear wave equation (26) with $\beta = 10$, $u(x,0) = \exp(-100x^2)$, $u_x(x,0) = 0$ obtained with $\Delta x = 1/127$ and $h = 1/1270$ using the symmetric Gauss scheme with $s = 4$ stages

(a) 5 stages   (b) 15 stages   (c) 25 stages
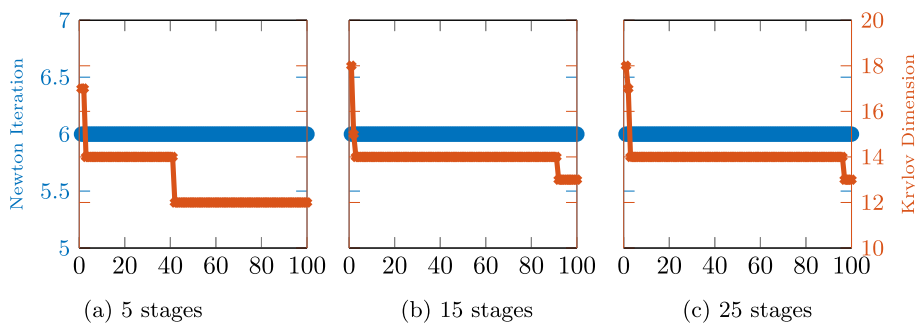
Fig. 9 Iteration count for the simplified Newton method and average dimension of the extended rational Krylov subspace for the correction matrix equation (23)

where the operation $\mathbf{u}^2$ should be understood componentwise; see Fig. 8 for a depiction of the solution of (26) with $\Delta x = 1/127$ and $h = 1/1270$ obtained using the symmetric Gauss scheme with $s = 4$ stages. To have both a larger problem size and moving towards the direction of a more realistic problem, we consider the same test illustrated in Fig. 8 discretized on a finer spatial mesh with $\Delta x = 1/1023$ and set the time-step size to $h = \Delta x/10$. The simulation is carried out over 100 time-steps to collect performance data, ensuring that all simulations reach the same final time independently of s. We employ the Gauss integration scheme with varying numbers of stages s, and initially focus on two metrics: the number of simplified Newton iterations required per time-step, and the average dimension of the Krylov subspace used in solving the correction Sylvester matrix equations. In particular, we utilize a variant of Algorithm 1 that employs the extended Krylov subspace method [23], where the subspace is expanded by two basis vectors at each iteration. As shown in Fig. 9, the number of simplified Newton iterations—targeting a convergence tolerance of 10^{-10}—remains stable across time-steps and is largely unaffected by the number of Gauss stages. The average dimension of the Krylov subspace exhibits some variability, but the fluctuations remain well-contained throughout the simulation. Fig. 10 illustrates the performance comparison between the threaded and sequential implementations of the method as a function of the number of stages which is equal to the number of threads. The left axis reports the total elapsed time
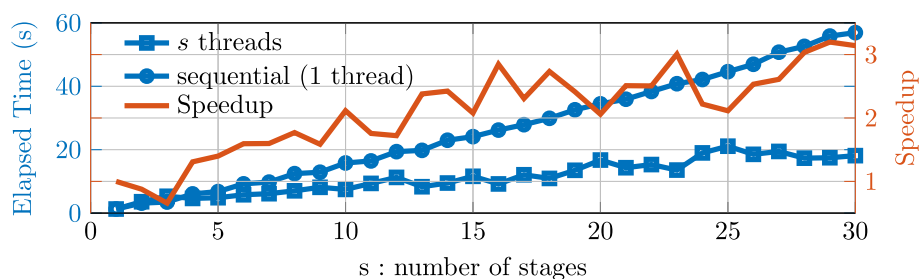
Fig. 10 Performance in terms of elapsed times and speedup of the threaded version against the sequential implementation for varying number of threads

measured in seconds, while the right axis shows the speedup. As expected, the execution time increases with the number of stages due to the higher computational cost per time-step. However, the threaded implementation consistently outperforms the sequential counterpart, achieving substantial speedup, particularly in the mid-to-high stage range. Notably, a speedup close to $3\times$ is observed for several stage counts, indicating effective parallel scalability of the threaded solver. The variability in speedup at higher stage counts may be attributed to increasing synchronization overhead or memory contention effects in the parallel execution. For low stage counts, however, the threaded implementation shows no significant performance improvement. This is primarily due to the limited amount of parallel work available in these configurations, where the overhead of thread management and synchronization outweighs the computational benefits. From a weak-scaling perspective, the behavior of the threaded implementation as s increases is encouraging: to an increase from $2N$ to $30N$ corresponds an increase in time of $5\times$. Although the total work per time-step grows with s, the threaded runtime exhibits only a moderate increase in elapsed time, suggesting that the solver sustains parallel efficiency even as the computational load per step increases.

7 Conclusions and Future Extensions

We have shown an approach for building stage-parallel IRK methods for high stage counts together with a feasible implementation in a multithreaded environment. The numerical results on two classical benchmark problems are promising and highlight the applicability of the method to more realistic settings. Among the future plans, there is to investigate in detail the construction of iterative methods for the solution of block diagonal systems that arise from the procedure of fast diagonalization of the equations for the stages. Moreover, we aim to extend the construction in two directions. First to encompass extension of Runge–Kutta methods for conservative problems which can be expressed in an analogous matrix format [33], secondly towards cases where, beyond handling the stages of the Runge–Kutta method, we also solve all time-steps simultaneously. On the implementation side, we plan to incorporate this strategy into the *Parallel Sparse Computation Toolkit* [34] to fully leverage a parallel computing environment. This approach enables the distribution of matrices across multiple processes and allows for the overlapping of stage computations, thereby enhancing overall performance.

Data Availability Data sharing is not applicable to this article as no new datasets were generated during the current study.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Code availability The code discussed here is implemented in the Julia package available at the GitHub repository Cirdans-Home/SP_IRK.jl.

References

1. Bellen, A.: Parallelism across the steps for difference and differential equations. In: Numerical Methods for Ordinary Differential Equations (L'Aquila, 1987). Lecture Notes in Math., vol. 1386, pp. 22–35. Springer, Berlin (1989). https://doi.org/10.1007/BFb0089229
2. Bellen, A., Vermiglio, R., Zennaro, M.: Parallel ODE-solvers with stepsize control. J. Comput. Appl. Math. **31**(2), 277–293 (1990). https://doi.org/10.1016/0377-0427(90)90170-5
3. Houwen, P.J., Sommeijer, B.P.: Iterated Runge-Kutta methods on parallel computers. SIAM J. Sci. Statist. Comput. **12**(5), 1000–1028 (1991). https://doi.org/10.1137/0912054
4. Houwen, P.J., Sommeijer, B.P., Couzy, W.: Embedded diagonally implicit Runge-Kutta algorithms on parallel computers. Math. Comp. **58**(197), 135–159 (1992). https://doi.org/10.2307/2153025
5. Houwen, P.J., Sommeijer, B.P.: Preconditioning in parallel Runge-Kutta methods for stiff initial value problems. vol. 28, pp. 17–31 (1994). https://doi.org/10.1016/0898-1221(94)00183-9
6. Iserles, A., Nørsett, S.P.: On the theory of parallel Runge-Kutta methods. IMA J. Numer. Anal. **10**(4), 463–488 (1990). https://doi.org/10.1093/imanum/10.4.463
7. Jay, L.O.: Inexact simplified Newton iterations for implicit Runge-Kutta methods. SIAM J. Numer. Anal. **38**(4), 1369–1388 (2000). https://doi.org/10.1137/S0036142999360573
8. Munch, P., Dravins, I., Kronbichler, M., Neytcheva, M.: Stage-parallel fully implicit Runge-Kutta implementations with optimal multilevel preconditioners at the scaling limit. SIAM J. Sci. Comput. **46**(2), 71–96 (2024). https://doi.org/10.1137/22M1503270
9. Sommeijer, B.P.: Explicit, high-order Runge-Kutta-Nyström methods for parallel computers. Appl. Numer. Math. **13**(1–3), 221–240 (1993). https://doi.org/10.1016/0168-9274(93)90145-H
10. Cong, N.H.: Note on the performance of direct and indirect Runge-Kutta-Nyström methods. J. Comput. Appl. Math. **45**(3), 347–355 (1993). https://doi.org/10.1016/0377-0427(93)90053-E

11. Cong, N.H., Strehmel, K., Weiner, R., Podhaisky, H.: Runge-Kutta-Nyström-type parallel block predictor-corrector methods. Adv. Comput. Math. **10**(2), 115–133 (1999). https://doi.org/10.1023/A:1018930732643

12. Southworth, B.S., Krzysik, O., Pazner, W., De Sterck, H.: Fast solution of fully implicit Runge-Kutta and discontinuous Galerkin in time for numerical PDEs, Part I: the linear setting. SIAM J. Sci. Comput. **44**(1), 416–443 (2022). https://doi.org/10.1137/21M1389742

13. Southworth, B.S., Krzysik, O., Pazner, W.: Fast solution of fully implicit Runge-Kutta and discontinuous Galerkin in time for numerical PDEs, Part II: nonlinearities and DAEs. SIAM J. Sci. Comput. **44**(2), 636–663 (2022). https://doi.org/10.1137/21M1390438

14. Leveque, S., Bergamaschi, L., Martínez, A., Pearson, J.W.: Parallel-in-time solver for the all-at-once Runge-Kutta discretization. SIAM J. Matrix Anal. Appl. **45**(4), 1902–1928 (2024). https://doi.org/10.1137/23M1567862

15. Simoncini, V.: Computational methods for linear matrix equations. SIAM Rev. **58**(3), 377–441 (2016). https://doi.org/10.1137/130912839

16. Kressner, D., Massei, S., Zhu, J.: Improved ParaDiag via low-rank updates and interpolation. Numer. Math. **155**(1–2), 175–209 (2023). https://doi.org/10.1007/s00211-023-01372-w

17. Gander, M.J., Palitta, D.: A new ParaDiag time-parallel time integration method. SIAM J. Sci. Comput. **46**(2), 697–718 (2024). https://doi.org/10.1137/23M1568028

18. Scherer, R., Türke, H.: Reflected and transposed Runge-Kutta methods. BIT **23**(2), 262–266 (1983). https://doi.org/10.1007/BF02218447

19. Hairer, E., Lubich, C., Wanner, G.: Geometric Numerical Integration. Springer Series in Computational Mathematics, vol. 31, p. 515. Springer, Berlin (2002). https://doi.org/10.1007/978-3-662-05018-7

20. Hairer, E., Nørsett, S.P., Wanner, G.: Solving Ordinary Differential Equations. I, 2nd edn. Springer Series in Computational Mathematics, vol. 8, p. 528. Springer, Berlin (1993). https://doi.org/10.1007/978-3-540-78862-1

21. Collar, A.R.: On centrosymmetric and centroskew matrices. Quart. J. Mech. Appl. Math. **15**, 265–281 (1962). https://doi.org/10.1093/qjmam/15.3.265

22. Berljafa, M., Güttel, S.: Generalized rational Krylov decompositions with an application to rational approximation. SIAM J. Matrix Anal. Appl. **36**(2), 894–916 (2015). https://doi.org/10.1137/140998081

23. Simoncini, V.: A new iterative method for solving large-scale Lyapunov matrix equations. SIAM J. Sci. Comput. **29**(3), 1268–1288 (2007). https://doi.org/10.1137/06066120X

24. Kågström, B., Poromaa, P.: LAPACK-style algorithms and software for solving the generalized Sylvester equation and estimating the separation between regular matrix pairs. ACM Trans. Math. Software **22**(1), 78–103 (1996). https://doi.org/10.1145/225545.225552

25. Gautschi, W.: Orthogonal Polynomials: Computation and Approximation. Numerical Mathematics and Scientific Computation, p. 301. Oxford University Press, New York (2004). https://doi.org/10.1093/oso/9780198506720.001.0001

26. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations. II, 2nd edn. Springer Series in Computational Mathematics, vol. 14, p. 614. Springer, Berlin (1996). https://doi.org/10.1007/978-3-642-05221-7

27. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn., p. 528. Society for Industrial and Applied Mathematics, Philadelphia, PA (2003). https://doi.org/10.1137/1.9780898718003

28. Liniger, W., Willoughby, R.A.: Efficient integration methods for stiff systems of ordinary differential equations. SIAM J. Numer. Anal. **7**, 47–66 (1970). https://doi.org/10.1137/0707002

29. Gander, M.J., Wu, S.-L.: A diagonalization-based parareal algorithm for dissipative and wave propagation problems. SIAM J. Numer. Anal. **58**(5), 2981–3009 (2020). https://doi.org/10.1137/19M1271683

30. Davis, T.A.: Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. ACM Trans. Math. Software **30**(2), 196–199 (2004). https://doi.org/10.1145/992200.992206

31. Badia, S., Verdugo, F.: Gridap: an extensible Finite Element toolbox in Julia. J. open source softw. **5**(52), 2520 (2020). https://doi.org/10.21105/joss.02520

32. Verdugo, F., Badia, S.: The software design of Gridap: a finite element package based on the Julia JIT compiler. Comput. Phys. Commun. **276**, 108341 (2022). https://doi.org/10.1016/j.cpc.2022.108341

33. Brugnano, L., Iavernaro, F.: Line Integral Methods for Conservative Problems. Monographs and Research Notes in Mathematics, p. 222. CRC Press, Boca Raton, FL (2016). https://doi.org/10.1201/b19319

34. D'Ambra, P., Durastante, F., Filippone, S.: Parallel sparse computation toolkit[formula presented]. Software Impacts **15** (2023) https://doi.org/10.1016/j.simpa.2022.100463. Cited by: 1; All Open Access, Gold Open Access, Green Open Access