To appear in J.Comput.Appl.Math.

A block Krylov subspace implementation of the time-parallel Paraexp method and its extension for nonlinear partial differential equations

G.L. Kooij (g.l.kooij@utwente.nl)[a,*], M.A. Botchev[b,c], B.J. Geurts[a,d]

[a] *Multiscale Modeling and Simulation, Faculty EEMCS, University of Twente, Enschede, Netherlands*
[b] *Mathematics of Computational Science, Faculty EEMCS, University of Twente, Enschede, Netherlands*
[c] *Institute of Numerical Mathematics, Russian Academy of Sciences, Moscow, Russia*
[d] *Faculty of Applied Physics, Fluid Dynamics Laboratory, Eindhoven University of Technology, Eindhoven, Netherlands*

Abstract

A parallel time integration method for nonlinear partial differential equations is proposed. It is based on a new implementation of the Paraexp method for linear partial differential equations (PDEs) employing a block Krylov subspace method. For nonlinear PDEs the algorithm is based on our Paraexp implementation within a waveform relaxation. The initial value problem is solved iteratively on a complete time interval. Nonlinear terms are treated as a source term, provided by the solution from the previous iteration. At each iteration, the problem is decoupled into independent subproblems by the principle of superposition. The decoupled subproblems are solved fast by exponential integration, based on a block Krylov method. The new time integration is demonstrated for the one-dimensional advection-diffusion equation and the viscous Burgers equation. Numerical experiments confirm excellent parallel scaling for the linear advection-diffusion problem, and good scaling in case the nonlinear Burgers equation is simulated.

Keywords: parallel computing, exponential integrators, partial differential equations, parallel in time, block Krylov subspace

1. Introduction

Recent developments in parallel computing urge the design of new numerical methods, as well as a major revision of existing numerical algorithms [14]. To be efficient on massively parallel platforms, the algorithms need to employ all possible means to parallelize the computations. When solving partial differential equations (PDEs) with a time-dependent solution, an important way to parallelize the computations is, next to the parallelization across space, parallelization across time. This adds a new dimension of parallelism with which the simulations can be implemented. In this paper we present a new time-parallel integration method extending the Paraexp method [21] to nonlinear partial differential equations using Krylov methods and waveform relaxation.

Several approaches to parallelize the simulation of time-dependent solutions in time can be distinguished. The first important class of the methods are the waveform relaxation methods [6, 39, 45, 57], including the space-time multigrid methods for parabolic PDEs [8, 24, 30, 38]. The key idea is to start with an approximation to the numerical solution for the whole time interval of interest and update the solution, solving an easier-to-solve approximate system in time. The Parareal method [37], which attracted significant attention recently, is a prime example related to the class of waveform relaxation methods [19].

Parallel Runge–Kutta methods and general linear methods, where the parallelism is determined and restricted by the number of stages or steps, form another class of the time-parallel methods [8, 12, 53]. Time-parallel schemes can also be obtained by treating the time as an additional space dimension and solving for

[*] Corresponding author

values at all time steps at once [13, 58]. This approach requires significantly more memory and is used, e.g., in multidimensional tensor computations and (discontinuous) Galerkin finite element methods [9, 31, 36, 55]. Recently a "parallel full approximation scheme in space and time" (PFASST) was introduced, which is based on multigrid methods [17]. PFASST was observed to have a generally improved parallel efficiency compared to Parareal. Also, we mention parallel methods that facilitate parallelism by replacing the exact solves in implicit schemes by approximate ones [7], and parallel methods based on operator splitting [4]. Finally, there is the Paraexp method [21] for parallel integration of linear initial-value problems. This algorithm is based on the fact that linear initial-value problems can be decomposed into homogeneous and nonhomogeneous subproblems. The homogeneous subproblems can be solved fast with an exponential integrator, while the nonhomogeneous subproblems can be solved with a traditional time-stepping method.

In this paper we propose a time parallel method which is based on the matrix exponential time stepping technique and waveform relaxation. Our method adopts the Paraexp method within a waveform relaxation framework, which enables the extension of parallel time integration to nonlinear PDEs. The method is inspired by and relies on recent techniques in Krylov subspace matrix exponential methods such as shift-and-invert acceleration [22, 41, 52], restarting [1, 10, 16, 46, 51] and using block Krylov subspaces [2] to handle nonautonomous PDEs efficiently. The method also relies on a singular value decomposition (SVD) of source terms in the PDE, which is used to construct the block Krylov subspace. To improve the efficiency of the method, the SVD is truncated by retaining only the relatively large singular values. We show in theory and in practice that for a source term, that can be approximated by a smooth function, the singular values decay algebraically with time interval size. In that case, a truncated SVD approximation of the source term is adequate.

The contribution of this paper is more specifically as follows. First, to solve systems of linear ODEs, we propose an implementation of the Paraexp method, based on the exponential block Krylov (EBK) method, introduced in [2]. The EBK method is a Krylov method that approximates the exact solution of a system of nonhomogeneous linear ordinary differential equations by a projection onto a block Krylov subspace. Our Paraexp implementation does not involve a splitting of homogeneous and nonhomogeneous subproblems, which leads to a better parallel effiency. Second, we extend our EBK-based implementation of the Paraexp method to nonlinear initial-value problems. We solve the problem iteratively on a chosen time interval. In our case, the nonlinear term of the PDE is incorporated as a source term, which is updated after every iteration with the latest solution. Third, we show that our Paraexp-EBK (PEBK) implementation has promising properties for time-parallelization. The PEBK method can be seen as a special continuous-time waveform relaxation method, which is typically more efficient than standard waveform relaxation methods [6]. The PEBK method is tested for the one-dimensional advection-diffusion equation and the viscous Burgers equation in an extensive series of numerical experiments. Since we are interested in time-parallel solvers for large-scale applications in computational fluid dynamics (CFD), such as turbulent flow simulations, we also test our method with respect to the diffusion/advection ratio and grid resolution, reflecting high Reynolds-number conditions. For example, the parallel efficiency of the Parareal algorithm was found to decrease considerably with increasing number of processors for the advection equation [20, 48, 49]. In contrast, the EBK method was found to have excellent weak scaling for linear problems.

The paper is organized as follows. In Section 2 we describe the exponential time integration scheme and its parallelization. In Section 3, we present numerical experiments with the advection-diffusion equation, and with the viscous Burgers equation in Section 4. Finally, a discussion and conclusions are outlined in Section 5.

2. Exponential time-integration

Our time integration method for linear and nonlinear PDEs is based on the exponential block Krylov (EBK) method [2]. In this section, we first provide a brief description of the EBK method. Then, we extend the EBK method to integrate nonlinear PDEs in an iterative way. Finally, the parallelization of the time integrator is discussed.

2.1. Exponential block Krylov method

The EBK method is a time integrator for linear systems of nonhomogeneous ordinary differential equations (ODEs). Details of the method are given in [2]. We follow the *method of lines* approach [56], i.e., the PDE is discretized in space first. We start with linear PDEs. After applying a spatial discretization to the PDE, we obtain the initial value problem,

$$\begin{aligned}\mathbf{u}'(t) &= A\mathbf{u}(t) + \mathbf{g}(t),\\ \mathbf{u}(0) &= \mathbf{u}_0,\end{aligned} \tag{1}$$

where $\mathbf{u}(t)$ is a vector function $\mathbf{u}(t) : \mathbb{R} \to \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$ is a square matrix, and $\mathbf{g}(t) \in \mathbb{R}^n$ a time-dependent source term. The vector $\mathbf{u}$ can readily be identified with the solution to the PDE in terms of its values taken on a computational grid in physical space. In Section 2.3, we will introduce a more general term $\mathbf{g}(t, \mathbf{u}(t))$, which contains the nonlinear terms of the PDE. The matrix A is typically large and sparse, depending on the spatial discretization method. The dimension of the system, n, corresponds to the number of degrees of freedom used in the spatial discretization. Exponential integrators approximate the exact solution of the semi-discrete system (1), formulated in terms of the matrix exponential.

To treat general source terms, $\mathbf{g}(t)$ is approximated with a piecewise polynomial function. The polynomial approximation of $\mathbf{g}(t)$ is based on a truncated singular value decomposition (SVD) [3], which improves the efficiency of the EBK method. The source term is sampled at s points in time, $0 = t_1 < t_2 < \ldots < t_{s-1} < t_s = \Delta T$, in the integration interval $[0, \Delta T]$, and the source samples form the matrix

$$G = [\,\mathbf{g}(t_1) \quad \mathbf{g}(t_2) \quad \ldots \quad \mathbf{g}(t_s)\,] \in \mathbb{R}^{n\times s}. \tag{2}$$

We make a natural assumption that the typical number of time samples s, necessary for a good approximation of $\mathbf{g}(t)$, is much lower than n: $s \ll n$. Since we assume $s \ll n$, it is more economical to calculate the so-called thin SVD of G [23], instead of the full SVD, without loss of accuracy. To obtain the thin SVD of G, we first compute its thin QR factorization, $G = QR$, and then compute the SVD of R. In MATLAB the thin SVD can be calculated with `svd(G,0)`. In this case, the thin SVD of G is

$$G = \tilde{U}\tilde{\Sigma}\tilde{V}^T, \tag{3}$$

where $\tilde{\Sigma} \in \mathbb{R}^{s\times s}$ is a diagonal matrix containing the singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_s$, and $\tilde{U} \in \mathbb{R}^{n\times s}$, $\tilde{V} \in \mathbb{R}^{s\times s}$ are matrices with orthonormal columns. The thin SVD can be approximated by a truncation in which the $m < s$ largest singular values are retained. As seen from this truncated SVD, the samples of $\mathbf{g}(t)$ can be approximated by linear combinations of the first m columns of $\tilde{U}$, i.e.,

$$\mathbf{g}(t) \approx U\mathbf{p}(t). \tag{4}$$

where $U \in \mathbb{R}^{n\times m}$ is formed by the first m columns of $\tilde{U}$, $\mathbf{p}(t) \in \mathbb{R}^m$ is obtained by an interpolation of the coefficients in these linear combinations [3]. There are several possible choices for $\mathbf{p}(t)$, among which, cubic piecewise polynomials. Then, for a given ΔT, the approximation error in the source term, $\|\mathbf{g}(t) - U\mathbf{p}(t)\|$, can be easily controlled within a desired tolerance, depending on the number of samples in $[0, \Delta T]$, and the number of singular values truncated (see [2]).

The number of retained singular values, m, is equal to the block width in the block Krylov subspace. The efficiency of the EBK method therefore depends on how many singular values are required for an accurate approximation of the source term (4). In applications of the method this parameter m can be varied and practical convergence can be assessed. If the subinterval ΔT is small and the source function $\mathbf{g}(t)$ can be well approximated by a smooth function, then the singular values of the samples of $\mathbf{g}$ decrease rapidly, thus allowing for an efficient low rank approximation in (4). Furthermore, let the subinterval length ΔT be small in the sense that $(\Delta T)^2 \ll \Delta T$. Of course, the assumption that ΔT is small, is not always realistic and we comment on how it can be partially relaxed below in Remark 1.

Denote by $\mathbf{g}^{(j)}$ the jth derivative of $\mathbf{g}$ and assume that the constants

$$M_j := \max_{\xi\in[0,\Delta T]} \|g^{(j)}(\xi)\| \qquad j = 0, \ldots, n,$$

are bounded. Starting from a thin QR factorization of G, we can then show that the singular values of the samples decrease.

Consider the thin QR factorization of G, $G = QR$, where $Q \in \mathbb{R}^{n\times s}$ has orthonormal columns and $R \in \mathbb{R}^{s\times s}$ is an upper triangular matrix with nonnegative diagonal entries. As Theorem 1 in [25] states, the entries r_{jk} of R satisfy

$$|r_{jk}| \leq C_j M_{j-1}(\Delta T)^{j-1}, \quad k \geq j,$$

and $r_{jk} = 0$ for $k < j$ because R is upper triangular. In this estimate the constants C_j depend only on the points $t_1, \ldots, t_s$ at which the samples are computed and do not depend on $\mathbf{g}$ and ΔT.

Since G and R have the same singular values, we consider now the singular values of R. According to [29, 3.1.3],

$$\sigma_{j+1} \leq \|R_j\|_2, \qquad j = 1, \ldots, s-1, \tag{5}$$

where R_j is a matrix obtained from R by skipping j rows or columns, chosen arbitrarily. Since the entries in R monotonically decrease rowwise as $|r_{jk}| = O(\Delta T)^{j-1}$, to have the sharpest estimate in (5), we choose R_j to be the matrix R with the first j rows skipped. To bound the 2-norm of R_j we use [28, Chapter 5.6]

$$\|R_j\|_2 \leq \sqrt{\|R_j\|_1 \|R_j\|_\infty}$$

and note that for $j = 1, \ldots, s-1$

$$\|R_j\|_\infty \leq (s-j) C_j M_{j-1}(\Delta T)^{j-1}, \qquad \|R_j\|_1 = O(\Delta T)^{j-1}.$$

Thus, we obtain $\|R_j\|_2 = \sqrt{s-j}\, O(\Delta T)^{j-1}$, which, together with (5), yields the following result.

Theorem 1. *Let $\mathbf{g}(t) : \mathbb{R} \to \mathbb{R}^n$ be a smooth function such that the constants*

$$M_j := \max_{t\in[0,\Delta T]} \|g^{(j)}(t)\| \qquad j = 0, \ldots, n,$$

are bounded. Furthermore, let the subinterval length ΔT be small in the sense that $(\Delta T)^2 \ll \Delta T$. Then for the singular values σ_j, $j = 1, \ldots, s$, of the sample matrix

$$G = \begin{bmatrix} \mathbf{g}(t_1) & \mathbf{g}(t_2) & \ldots & \mathbf{g}(t_s) \end{bmatrix} \in \mathbb{R}^{n\times s}$$

holds

$$\sigma_{j+1} = \sqrt{s-j}\, O(\Delta T)^j, \qquad j = 1, \ldots, s-1. \tag{6}$$

Remark 1. *We may extend the result of Theorem 1 to the case of a large ΔT if the constants M_j are bounded more strongly or even decay with j. Indeed, if ΔT is large, we can consider the function $\tilde{\mathbf{g}}(\tilde{\xi}) := \mathbf{g}(t + \xi q)$, with q chosen such that $\tau := \Delta T/q$ is small. As the function $\tilde{\mathbf{g}}$ takes the same values for $\tilde{\xi} \in [0, \tau]$ as the function $\mathbf{g}$ for $\xi \in [t, t + \Delta T]$, Theorem 1 formally holds for $\tilde{\mathbf{g}}$ with ΔT replaced by τ and M_j multiplied by q^j. The coefficients in the O symbol in (6) may now grow with the powers of ΔT, thus rendering the result meaningless unless a stricter assumption on M_j is made.*

The truncated SVD approximation of the source term (4) facilitates the solution of the initial value problem (IVP) in (1) by the block Krylov subspace method [2]. Here, the block Krylov subspace at iteration l is defined as

$$\mathcal{K}_l(A, U) := \operatorname{span}\left\{U, AU, A^2U, \ldots, A^{l-1}U\right\}. \tag{7}$$

Furthermore, the block Krylov subspace method can be accelerated by the shift-and-invert technique [22, 41, 52] and, to keep the Krylov subspace dimension bounded, implemented with restarting [5, 10].

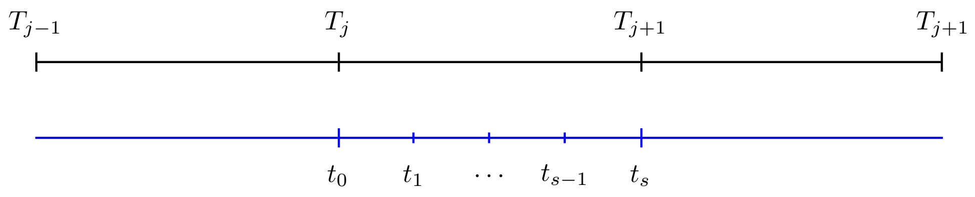


Figure 1: Time samples of the source term on the subinterval $[T_j, T_{j+1}]$.

2.2. Parallelization of linear problems

In this section, the parallelization of the linear ODE solution is discussed. As we will see, the method is equivalent to the Paraexp method, but there are differences in implementation leading to a better parallel efficiency (see Section 3.2). Linear IVPs can be solved parallel in time by using the principle of superposition. First, the time interval is divided into P non-overlapping subintervals,

$$0 = T_0 < T_1 < \ldots < T_P = T,$$

where P is also the number of processors that can be used in parallel. We introduce the shift of $\mathbf{u}(t)$ with respect to the initial condition, $\mathbf{u}(t) = \mathbf{u}_0 + \widehat{\mathbf{u}}(t)$. The shifted variable $\widehat{\mathbf{u}}(t)$ solves the IVP with homogeneous initial conditions,

$$\begin{aligned} \widehat{\mathbf{u}}'(t) &= A\widehat{\mathbf{u}}(t) + \widehat{\mathbf{g}}(t), \quad t \in [0, T], \\ \widehat{\mathbf{u}}(0) &= \mathbf{0}, \end{aligned} \tag{8}$$

where

$$\widehat{\mathbf{g}}(t) := A\mathbf{u}_0 + \mathbf{g}(t). \tag{9}$$

The source term is approximated using s time samples per subinterval, as illustrated in Fig. 1. The shifted IVP (8) can be decoupled into independent subproblems by the principle of superposition. To each subinterval $[T_{j-1}, T_j]$ we associate a part of the source term $\widehat{\mathbf{g}}(t)$, defined for $j = 1, \ldots, P$ as

$$\widehat{\mathbf{g}}_j(t) = \begin{cases} \widehat{\mathbf{g}}(t), & \text{for } T_{j-1} \le t < T_j, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

The expected parallel speedup is based on the observation that the solution to (8) is given by a variation-of-constant formula (see, e.g., [26]),

$$\widehat{\mathbf{u}}(T) = \int_0^T \exp\left((T-s)A\right) \widehat{\mathbf{g}}(s)\, ds = \sum_{j=1}^{P} \int_{T_{j-1}}^{T} \exp\left((T-s)A\right) \widehat{\mathbf{g}}_j(s)\, ds, \tag{11}$$

where the integrals on $[T_{j-1}, T]$ can be evaluated independently and in parallel. Here, we emphasize that the integration should be done for $[T_{j-1}, T]$ rather than for $[T_{j-1}, T_j]$ to account for the influence of the source term on the solution at later time. More precisely, by exploiting the linearity of the problem we decompose the IVP (8) into P independent subproblems,

$$\begin{aligned} \mathbf{v}_j'(t) &= A\mathbf{v}_j(t) + \widehat{\mathbf{g}}_j(t), \quad t \in [0, T], \\ \mathbf{v}_j(0) &= \mathbf{0}. \end{aligned} \tag{12}$$

where $\mathbf{v}_j(t)$ is referred to as a subsolution of the total problem. The subproblems are independent and can be solved in parallel. We integrate the subproblem individually with the EBK method. Observe that the source is only nonzero on the subinterval $[T_{j-1}, T_j)$. We follow the ideas of the Paraexp method [21] and note that the nonhomogeneous part of the ODE requires most of the computational work in the EBK method. An accurate SVD approximation of the source term (4) generally requires more singular values to be retained, increasing the dimensions of the block Krylov subspace (7). According to the principle of superposition, the solution of the original problem is then given as

$$\mathbf{u}(t) = \mathbf{u}_0 + \sum_{j=1}^{P} \mathbf{v}_j(t). \tag{13}$$

The summation of the subsolutions, $\mathbf{v}_j(t)$ is the only communication required between the parallel processes. The parallel algorithm is summarized in Fig. 2. In principle, this algorithm is identical to the Paraexp method [21]. The only practical difference is that in our implementation both the nonhomogeneous and the homogeneous part of the subproblems are solved by the EBK method. The original version of the Paraexp method assumes a convential time integration method to solve the "difficult" nonhomogeneous part, and a Krylov subspace method for exponential integration of the homogeneous part. Our implementation of the Paraexp method is compared numerically with the original one in Section 3.2.

Finally, additional parallelism could be exploited within the block Krylov subspace method itself. If the block size is m, then the m matrix-vector products can be executed entirely in parallel, see [35, 47]. This approach could be applied in combination with the parallelization described in this section.

```
Algorithm parallel time integration.
Given: $A$, $\mathbf{u}_0$, $\mathbf{g}(t)$, ...
Solve: $\mathbf{u}'(t) = A\mathbf{u}(t) + \mathbf{g}(t)$,   $\mathbf{u}(0) = \mathbf{u}_0$.
for $j = 1, \dots, P$ (in parallel)
  Solve nonhomogeneous part of the IVP (12), for $t \in [T_{j-1}, T_j]$.
  Solve homogeneous part of the IVP (12), for $t \in [T_j, T_P]$.
end for
Construct solution $\mathbf{u}(t)$, see (13).
```

Figure 2: The algorithm for solving linear ODE systems with the Paraexp exponential block Krylov (PEBK) method.

2.3. Treatment of nonlinearities

The EBK method, designed to solve a linear system of nonhomogeneous ODEs (1), can be extended to handle nonlinear systems of ODEs by including the nonlinearities in the source term. The system is then solved iteratively, in such a way that the iterand $\mathbf{u}_k(t)$ is updated on the entire interval $[0, T]$:

$$\mathbf{u}_k(t) \mapsto \mathbf{u}_{k+1}(t), \tag{14}$$

where k denotes the iteration index. We proceed in a few steps. First, the problem is reduced to the form of Eq. (1). The nonlinear IVP is therefore approximated by evaluating the source term with the current iterand, $\mathbf{u}_k(t)$. Because the current iterand $\mathbf{u}_k(t)$ is a known function, we can write the source term as an explicit function of time,

$$\mathbf{g}_k(t) := \mathbf{g}(t, \mathbf{u}_k(t)). \tag{15}$$

The resulting initial value problem is,

$$\begin{aligned} &\mathbf{u}'_{k+1}(t) = A\mathbf{u}_{k+1}(t) + \mathbf{g}_k(t), \\ &\mathbf{u}_{k+1}(0) = \mathbf{u}_0. \end{aligned} \tag{16}$$

This system is solved by the EBK method at each iteration until the solution has sufficiently converged. This approach is similar to applying Picard or fixed-point iterations [43] on the nonlinear term. We note that this approach, when we compute the low-rank approximation of Eq. (15) after every iteration, can be improved by using the "dynamical low-rank approximation" [33]. This would allow to reduce the computational work by reusing the low-rank approximations computed for the previous iterands. However, in the presented tests computing the low-rank approximation every time anew is cheap relatively to the other costs.

The convergence behaviour is improved by taking into account the Jacobian matrix, similar to a Newton–Raphson method. In this case, we introduce the average Jacobian matrix,

$$J_k := \frac{1}{T}\int_0^T \left[\frac{\partial \mathbf{g}_k}{\partial u_1} \dots \frac{\partial \mathbf{g}_k}{\partial u_n}\right] dt, \tag{17}$$

which is averaged over the interval of integration $[0, T]$, as the EBK method requires a constant matrix. The nonlinear remainder, with respect to the time-averaged state, is contained in the source term. Using this correction, we then find the recursive relation,

$$\begin{aligned} &\mathbf{u}_{k+1}'(t) = [A + J_k]\mathbf{u}_{k+1}(t) + \mathbf{g}_k(t) - J_k\mathbf{u}_k(t), \\ &\mathbf{u}_{k+1}(0) = \mathbf{u}_0. \end{aligned} \tag{18}$$

When converged, $\mathbf{u}_{k+1}(t) = \mathbf{u}_k(t)$, the terms containing J_k eventually disappear. For simplicity, we present the algorithm for a fixed number of iterations K. The convergence can of course also be checked based on the nonlinear residual. We define the nonlinear residual as follows,

$$\mathbf{r}_{k+1}(t) := A\mathbf{u}_{k+1}(t) + \mathbf{g}_{k+1}(t) - \mathbf{u}_{k+1}'(t). \tag{19}$$

This expression can be rewritten as

$$\begin{aligned} \mathbf{r}_{k+1}(t) &= \underbrace{[A + J_k]\mathbf{u}_{k+1}(t) + \mathbf{g}_k(t) - J_k\mathbf{u}_k(t) - \mathbf{u}_{k+1}'(t)}_{\approx 0} + \mathbf{g}_{k+1}(t) - \mathbf{g}_k(t) - J_k\left(\mathbf{u}_{k+1}(t) - \mathbf{u}_k(t)\right), \\ &\approx \mathbf{g}_{k+1}(t) - \mathbf{g}_k(t) - J_k\left(\mathbf{u}_{k+1}(t) - \mathbf{u}_k(t)\right). \end{aligned} \tag{20}$$

Assuming that the nonlinear term is Lipschitz continuous, we find an upper bound of the nonlinear residual,

$$\|\mathbf{r}_{k+1}(t)\| \leq L\|\mathbf{u}_{k+1}(t) - \mathbf{u}_k(t)\|, \tag{21}$$

where L is a Lipschitz constant, accounting for the contributions due to the Jacobian and the dependence of $\mathbf{g}$ on $\mathbf{u}$ as in (15). This means that in practice the convergence can be controlled by setting a certain tolerance for $\|\mathbf{u}_{k+1}(t) - \mathbf{u}_k(t)\|$.

Remark 2. *The iteration* (18) *is well known in the literature on waveform relaxation methods [38, 39]. Its convergence is given, e.g., in [5, 43] and, in case of an inexact iteration, in [6]. These results, in particular, show that the iteration* (18) *converges superlinearly on* finite *time intervals if the norm of the matrix exponential of $t(A + J_k)$ decays exponentially with t, i.e., the eigenvalues of $A + J_k$ lie in the left half-plane. This is typically true for the heat equation and similar parabolic problems such as advection-diffusion problems.*

2.4. Parallelization of nonlinear problems

Nonlinear IVPs are solved in an iterative way with the PEBK method. We follow the *waveform relaxation* approach [6, 34, 43], that is, the problem is solved iteratively on the entire time interval of interest, $[0, T]$. The original problem is decomposed into a number of independent, "easier" subproblems on $[0, T]$ that can be solved iteratively, and in parallel.

At each iteration the nonlinear term is treated as source term, which gives us a linear system of ODEs that can be integrated in parallel, see Section 2.2. In the case of nonlinear problems, we take the average of the Jacobian matrix on each subinterval, which gives us the piecewise constant function for $j = 1, \dots, P$:

$$J_k(t) := \frac{1}{T_j - T_{j-1}}\int_{T_{j-1}}^{T_j} \left[\frac{\partial \mathbf{g}_k}{\partial u_1} \dots \frac{\partial \mathbf{g}_k}{\partial u_n}\right] dt, \quad t \in [T_{j-1}, T_j]. \tag{22}$$

The IVP is then shifted with respect to the initial condition,

$$\begin{aligned} &\widehat{\mathbf{u}}'_{k+1}(t) = [A + J_k(t)]\widehat{\mathbf{u}}_{k+1}(t) + \widehat{\mathbf{g}}_k(t), \quad t \in [0, T], \\ &\widehat{\mathbf{u}}_{k+1}(0) = \mathbf{0}, \end{aligned} \tag{23}$$

where,

$$\widehat{\mathbf{g}}_k(t) := A\mathbf{u}_0 + \mathbf{g}_k(t) + J_k(t)[\mathbf{u}_0 - \widehat{\mathbf{u}}_k(t)]. \tag{24}$$

The shifted IVP (23) can be then solved in parallel. We follow an approach similar to the Paraexp method, as explained in Section 2.2. Here, we define

$$\widehat{\mathbf{g}}_{j,k}(t) := \begin{cases} \widehat{\mathbf{g}}_k(t), & \text{for } T_{j-1} \leq t < T_j, \\ 0, & \text{otherwise.} \end{cases} \tag{25}$$

Now, the problem is decomposed into P independent subproblems. The subsolutions $\mathbf{v}_j$ result from the convergence of $\mathbf{v}_{j,k}$ satisfying:

$$\begin{aligned} &\mathbf{v}'_{j,k+1}(t) = [A + J_k(t)]\mathbf{v}_{j,k+1}(t) + \widehat{\mathbf{g}}_{j,k}(t), \quad t \in [0, T], \\ &\mathbf{v}_{j,k+1}(0) = \mathbf{0}, \end{aligned} \tag{26}$$

which can be integrated with the EBK method. Here, we have introduced a second subindex k to indicate the jth subsolution at the kth iteration. The total solution can then be updated according to the principle of superposition,

$$\mathbf{u}_{k+1}(t) = \mathbf{u}_0 + \sum_{j=1}^{P} \mathbf{v}_{j,k+1}(t). \tag{27}$$

After each iteration, the solution is assembled, and the time-averaged Jacobian matrix is updated at each subinterval, according to Eq. (22). Note that for linear systems of ODEs, no iterations are required at all. The parallel algorithm is summarized in Fig. 5.

The parallel effiency of the EBK method is achieved by incorporating the parallelization within the iterations that are required to solve nonlinear IVPs. The differences between the serial and the parallel algorithm are illustrated as flow diagrams in Figures 3 and 4.

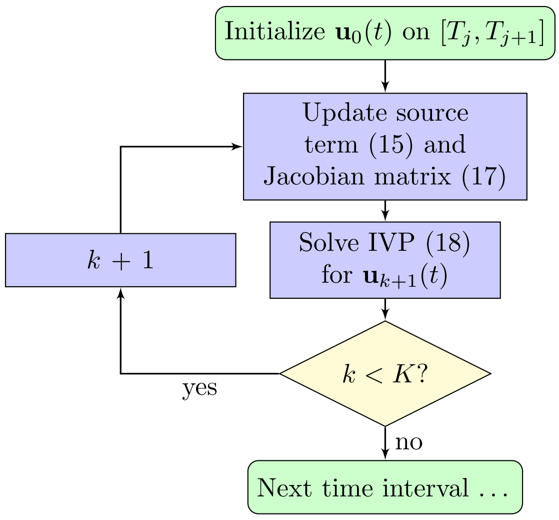
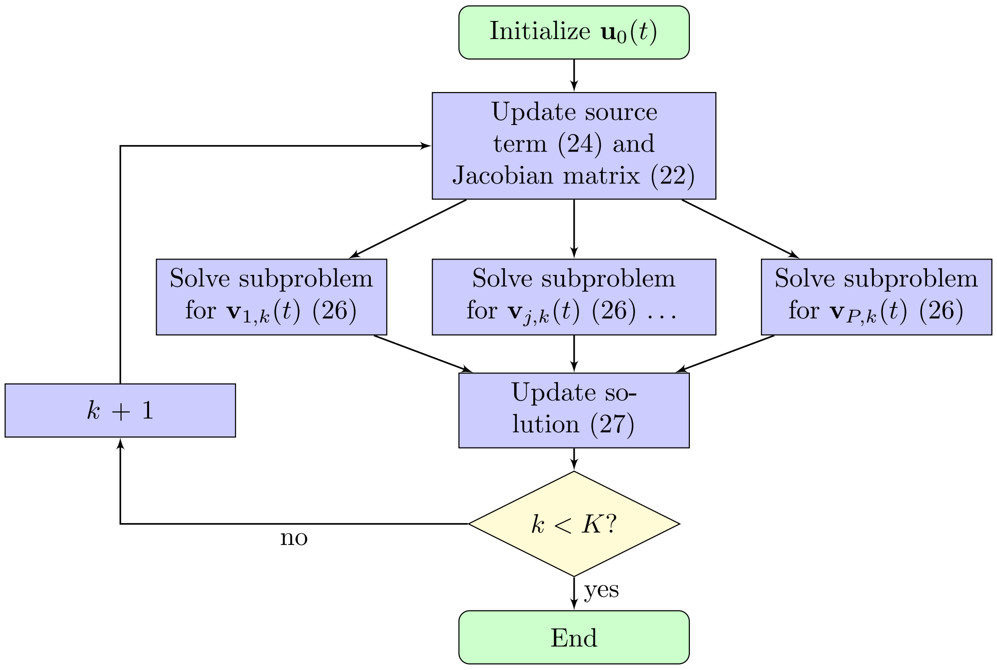

Initialize $\mathbf{u}_0(t)$ on $[T_j, T_{j+1}]$
Update source term (15) and Jacobian matrix (17)
$k + 1$
Solve IVP (18) for $\mathbf{u}_{k+1}(t)$
$k < K$?
yes
no
Next time interval ...

Figure 3: Flow diagram of the serial EBK method for nonlinear PDEs. The algorithm stops after K iterations, after which the solution is assumed to be converged.



Initialize $\mathbf{u}_0(t)$
Update source term (24) and Jacobian matrix (22)
Solve subproblem for $\mathbf{v}_{1,k}(t)$ (26)
Solve subproblem for $\mathbf{v}_{j,k}(t)$ (26) ...
Solve subproblem for $\mathbf{v}_{P,k}(t)$ (26)
Update solution (27)
$k + 1$
$k < K$?
no
yes
End

Figure 4: Flow diagram of the Paraexp-EBK method for nonlinear PDEs. The algorithm stops after K iterations, after which the solution is assumed to be converged.

We consider the parallel efficiency in an idealized setting, and estimate a theoretical upper bound here, assuming that communication among the processors can be carried out very efficiently. The computational cost can then be simply estimated by the number of iterations required for the numerical solution to converge. Suppose the computation time of a single EBK iteration is τ_0. The maximum parallel speedup is then,

$$\text{speedup} = \frac{K_1 \tau_0}{K_P \tau_0 / P} = \frac{K_1 P}{K_P}$$

where K_1 is number of iterations for serial time integration, and K_P for parallel time integration. The theoretical upper bound of the parallel efficiency is then,

$$\text{efficiency} = \frac{\text{speedup}}{P} = \frac{K_1}{K_P}. \tag{28}$$

High parallel efficiency can be achieved if the parallelization does not slow down the convergence of the numerical solution. As will be demonstrated in Section 4, we typically observe that K_P is not significantly larger than K_1, and a near-optimal efficiency is achieved in various relevant cases. For comparison, the parallel efficiency of the Parareal algorithm is formally bounded by $1/K_P$ [40]. In our case, this upper bound is improved by a factor K_1. Parareal is an iterative method for the parallelization of sequential time integration methods, whereas the EBK method, for nonlinear problems, is an iterative method to start with, and its parallelization does not necessarily increase the total number of iterations. Note the PFASST method [17] has a similar estimate of parallel efficiency as the PEBK method.

3. Advection-diffusion equation

In this section, we present results of numerical tests where the space-discretized advection-diffusion equation is solved with the EBK method. We demonstrate the consistency and stability of the EBK method for the linear advection-diffusion equation, which is a PDE describing a large variety of transport phenomena [32]. The spatial discretization, using central finite differencing for sake of illustration, of the PDE yields

```
ALGORITHM PARALLEL TIME INTEGRATION.
Given: $A$, $\mathbf{u}_0$, $\mathbf{g}(\mathbf{t}, \mathbf{u}(\mathbf{t}))$, ...
Solve: $\mathbf{u}'(t) = A\mathbf{u}(t) + \mathbf{g}(t, \mathbf{u}(t)), \quad \mathbf{u}(0) = \mathbf{u}_0$.
for $k = 1, \ldots, K$
  for $j = 1, \ldots, P$ (in parallel)
    Calculate time-averaged Jacobian matrix $J_k(t)$, see (22).
    Solve nonhomogeneous part linearized ODE (26), for $t \in [T_{j-1}, T_j]$.
    Solve homogeneous part linearized ODE (26), for $t \in [T_j, T_P]$.
  end for
  Update solution $\mathbf{u}_k(t)$ following (27).
end for
```

Figure 5: The algorithm for solving nonlinear problems with the Paraexp exponential block Krylov (PEBK) method, see also Fig. 4.

a linear system of ODEs. The time integration can be parallelized as described in Section 2.2. We illustrate the principle of parallelization for different physical parameters of the advection-diffusion equation, before we move on to nonlinear PDEs in Section 4. Convergence of the numerical solution is observed for different values of the physical parameters.

Here, we denote the relative tolerance of the PEBK solver by `tol`. In our tests, the block Krylov subspace is restarted every 20 iterations. For the truncated SVD approximation (4), $\mathbf{p}(t)$ are chosen to be piecewise cubic polynomials, although other types of approximations are possible as well, and are not crucial for the performance of the PEBK method. The sample points per subinterval are Chebyshev nodes. This is not crucial but gives a slightly better approximation than with uniform sample points.

3.1. Homogeneous PDE

We consider the advection-diffusion equation with a short pulse initial condition, to clearly distinguish the seperate effects of advection and diffusion. The PDE and the periodic boundary conditions are as follows

$$\begin{aligned} u_t + a u_x &= \nu u_{xx}, \quad x \in [0,1],\ t \in [0,1], \\ u(x,0) &= \sin^{20}(\pi x), \\ u(0,t) &= u(1,t), \\ u_x(0,t) &= u_x(1,t), \end{aligned} \tag{29}$$

where $a \in \mathbb{R}$ is the advection velocity, and $\nu \in \mathbb{R}$ the diffusivity coefficient. Both parameters are constant in space and time. The PDE is first discretized in space with a second-order central finite difference scheme [42]. The corresponding semi-discrete system of ODEs is then

$$\begin{aligned} \mathbf{v}'(t) &= A\mathbf{v}(t) + A\mathbf{u}_0, \\ \mathbf{v}(0) &= \mathbf{0}, \end{aligned} \tag{30}$$

where the matrix A results from the discretization of the spatial derivatives, and represents both the diffusive and the advective term. In (30), the substitution $\mathbf{u} = \mathbf{v} + \mathbf{u}_0$ has been applied, with $\mathbf{u}(t)$ being the vector function containing the values of the numerical solution on the mesh at time t, with $\mathbf{u}_0 = \mathbf{u}(0)$. The substitution leads to homogeneous initial conditions. In this case, the source term is constant in time, and its SVD polynomial approximation (4) is exact with $m = 1$. This allows us to focus on the two remaining parameters: the grid resolution and the tolerance of the EBK solver.

The linear IVP (30) is decoupled into independent subproblems by partitioning of the source term ($A\mathbf{u}_0$) on the time interval. The superposition of the subsolutions is illustrated in Fig. 6, in which the time interval of interest, $[0, 1]$, has been partitioned into four equal subintervals, to be integrated on four processors. The sum of the initial condition and the subsolutions gives the final solution on the entire time interval, see Section 2.2.

In the following numerical experiments, $P = 8$ subintervals are used. The advection-diffusion equation is solved for three different combinations of a and ν, see Fig. 7. The solutions are computed for different `tol` and Δx. The discretization error is controlled by the time integration with `tol`, and by the spatial discretization with Δx. The error of the numerical solution is measured in the relative ℓ^2-norm,

$$\|\mathbf{u}_h(T) - \mathbf{u}(T)\| / \|\mathbf{u}(T)\|, \tag{31}$$

where the exact solution $\mathbf{u}(T)$ is on the mesh, i.e., it has the entries $u_j(T) = u(x_j, T)$. The exact solution of Eq. (29) is

$$u(x,t) = \sum_{n=0}^{20} a_n \cos\left(n\pi(x - at)\right) \exp\left(-(n\pi)^2 \nu t\right), \tag{32}$$

with coefficients

$$a_0 = \frac{1}{2}\int_{-1}^{1} \sin^{20}(\pi x)\, dx, \tag{33}$$

$$a_n = \int_{-1}^{1} \sin^{20}(\pi x)\cos(n\pi x)\, dx, \tag{34}$$

which can be calculated either analytically or numerically. We note that the coefficients in Eqs. (33) and (34) are the standard Fourier coefficients for the interval $[0, 1]$, taking into account that $\sin^{20}(\pi x)$ is even. The error in (31) shows second-order convergence with Δx, as expected from the truncation error of the finite difference scheme. The convergence plots show that we are able to control the error of the parallel time integration method. In this case, the final error can be made to depend only on the spatial resolution and the tolerance of the EBK method. Also, the EBK method has no principal restrictions on the timestep size, as it directly approximates the exact solution of Eq. (30) by Krylov subspace projections.

3.2. Parallel efficiency

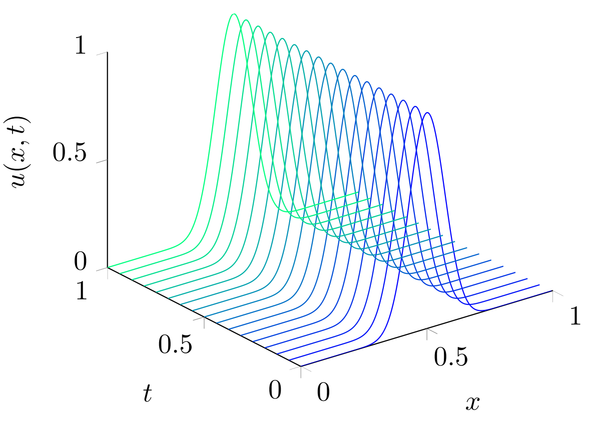
In the previous examples, we solved homogeneous IVPs. Nonhomogeneous problems are generally more expensive to solve, because an accurate SVD approximation of the source term (4) requires more singular values, which increases the block width of the block Krylov subspace (7). Parallel speedup can then be expected by splitting the nonhomogeneous problem into subproblems (12), which require less individual effort to solve by the PEBK method. To measure the parallel efficiency of our algorithm for nonhomogeneous IVPs, we introduce a source term $f(x, t)$ in the advection-diffusion equation

$$u_t + a u_x = \nu u_{xx} + f(x, t), \tag{35}$$

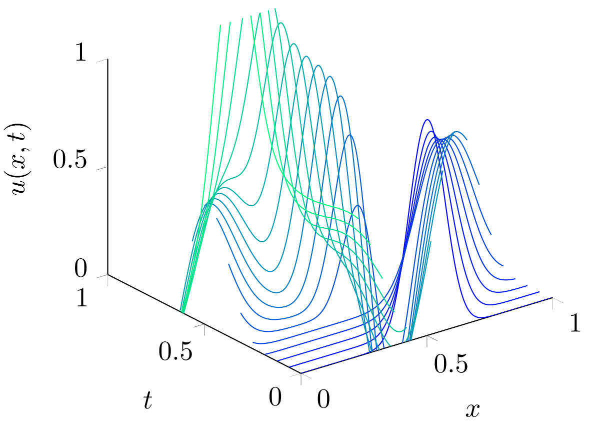
with $\nu = 10^{-2}$, and $a = 1$. The source term is chosen such that the solution is a series of five travelling pulses

$$u(x, t) = \tfrac{1}{2} - \tfrac{1}{2}\cos\left(10\pi(x - t)\right). \tag{36}$$

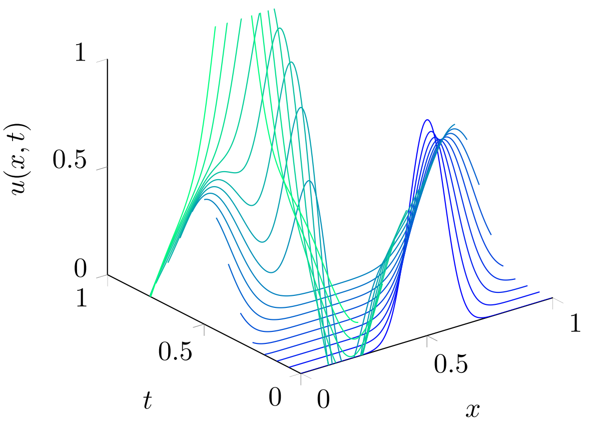
The mesh width of the spatial discretization is $\Delta x = 10^{-3}$, such that the error due to the spatial discretization is small compared to the time integration error. The mesh width gives a semi-discrete system with a 1000×1000 matrix, which is a suitable problem size for testing the EBK method. The tolerance of the EBK method is set to 10^{-4}. The SVD polynomial approximation is constructed from 100 time samples per subinterval. The singular values are plotted in Fig. 9, which reveals that the first two singular values are several orders larger than the rest. Therefore, we retain only the first two singular values in the truncated SVD. The decay of singular values is guaranteed by the upper bound from Theorem 1. We have verified that the SVD approximation is sufficiently accurate, i.e., the approximation error, measured in the L^2-norm, is less than the tolerance of the EBK method.



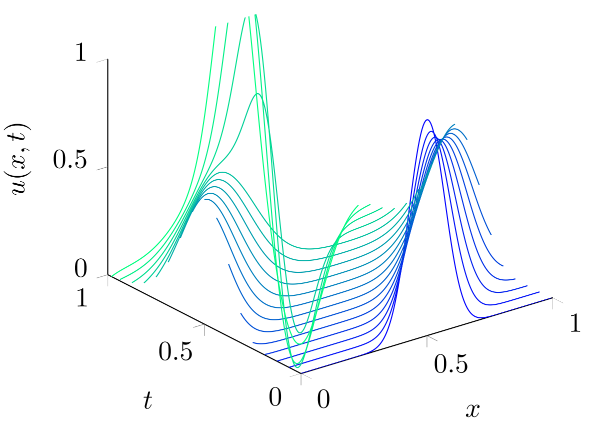
(a) $\mathbf{u}_0$



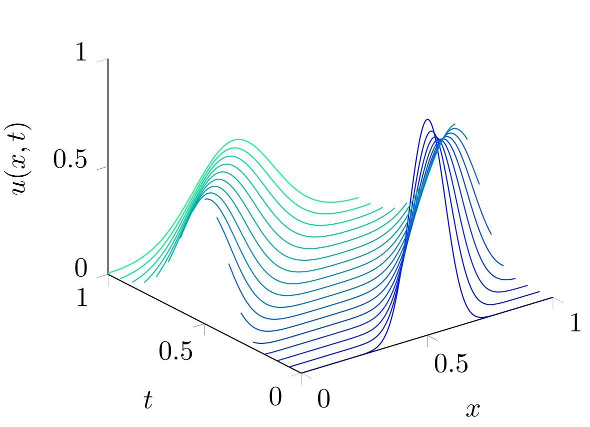
(b) $\mathbf{u}_0 + \mathbf{v}_1(t)$



(c) $\mathbf{u}_0 + \mathbf{v}_1(t) + \mathbf{v}_2(t)$

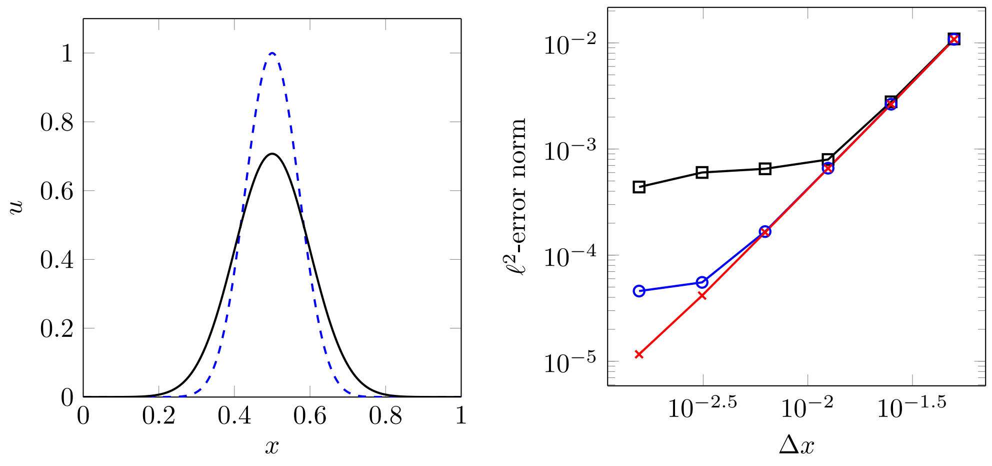


(d) $\mathbf{u}_0 + \mathbf{v}_1(t) + \mathbf{v}_2(t) + \mathbf{v}_3(t)$

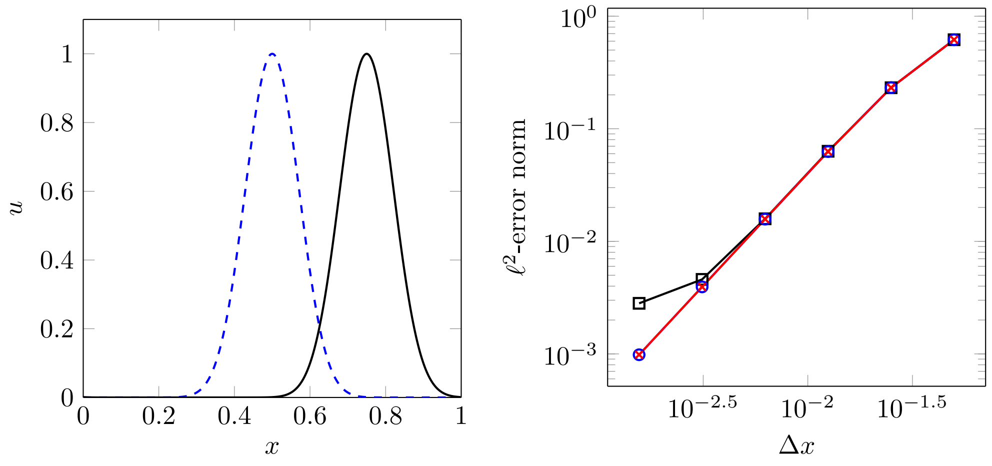


(e) $\mathbf{u}(t) = \mathbf{u}_0 + \sum_{j=1}^{4} \mathbf{v}_j(t)$

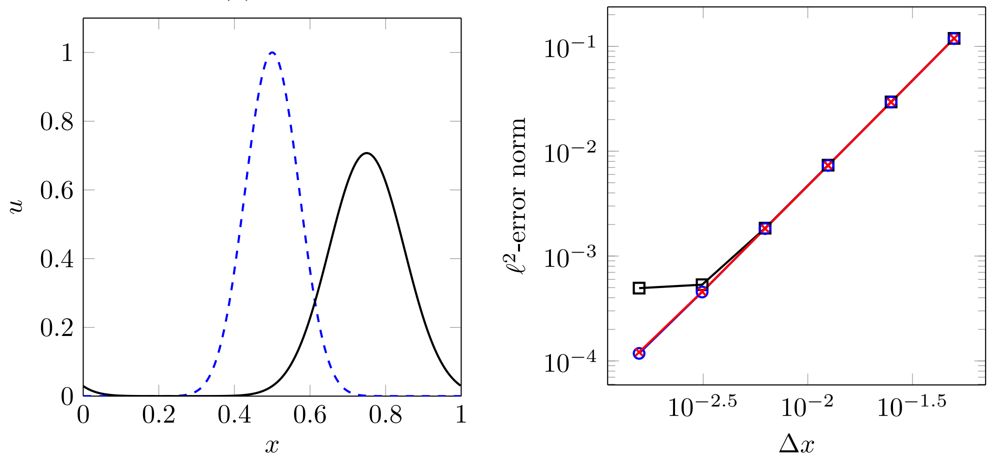
Figure 6: Superposition of subsolutions to the advection-diffusion equation, with partitioning $\mathbf{T} = \{0, 0.25, 0.5, 0.75, 1\}$, $\Delta x = 5 \cdot 10^{-3}$, $a = 1$, and $\nu = 10^{-2}$. The color depends on the time coordinate: blue corresponds to $t = 0$ and green to $t = 1$.



(a) Velocity $a = 0$ and diffusivity coefficient $\nu = 10^{-2}$.



(b) Velocity $a = 1$ and diffusivity coefficient $\nu = 0$.



(c) Velocity $a = 1$ and diffusivity coefficient $\nu = 10^{-2}$.

Figure 7: Left: numerical solution on the finest mesh at $t = 0$ (dashed) and $t = 0.25$ (solid). Note that the solution for $a = 1$ and $\nu = 0$ does not show odd-even decoupling because of the high spatial resolution used. Right: convergence of the numerical solution (at final time $t = 1$), for `tol`: □ 10^{-2}, ∘ 10^{-4}, × 10^{-6}.

We compare the parallel efficiency of the Paraexp-EBK implementation with a convential implementation of the Paraexp algorithm [21], where the nonhomogeneous problem is integrated with the Crank–Nicolson (CN) scheme. The linear system is solved directly using MATLAB. The matrix exponential propagator, for the homogeneous problem in the Paraexp method, is realized with an Arnoldi method using the shift-and-invert technique [52].

In order to have a Courant number of one, we take $\Delta t = 10^{-3}$ (for $\Delta x = 10^{-3}$ and $a = 1$). According to the Paraexp method, the time step size needs to be decreased in parallel computation by a factor $P^{1/2q}$ [21], where q is the order of the time integration method, in order to control the error. In case of the Crank–Nicolson scheme, we have $q = 2$.

As discussed in Section 2.2, there is no communication required between processors, except for the superposition of the solutions to the subproblems at the end. We can estimate the possible parallel performance of the algorithm on a serial computer by measuring the computation time of each independent subproblem, as done for example in [21]. Such an estimate provides a first indication for the parallel speedup. The computation time of the serial time integration is denoted τ_0. For the parallel time integration, we measure the computation times of the nonhomogeneous and the homogeneous part of the subproblems separately, denoted by τ_1 and τ_2 respectively. The parallel speedup can then be estimated as

$$\text{speedup} = \frac{\tau_0}{\max(\tau_1) + \max(\tau_2)}, \tag{37}$$

where we take the maximum value of τ_1 and τ_2 over all parallel processes. The timings of the EBK method and Paraexp are listed in Table 1.

The parallel efficiency of both methods is illustrated in Figure 8. Note that the parallel efficiency of the standard Paraexp method steadily decreases, whereas the PEBK method maintains a constant efficiency level around 90%. The decrease in efficiency of the standard Paraexp method is due to the reduced time step size in the Crank–Nicolson scheme, with respect to its serial implementation. The nonhomogeneous part of the subproblems requires more computation time as the number of processors increases.

The numerical test confirms the initial assumption that parallel speedup with the EBK method can be achieved by decomposing the original problem into simpler independent subproblems (12). Furthermore, the Paraexp implementation using the EBK method, appears to be more efficient than one using a traditional time-stepping method.

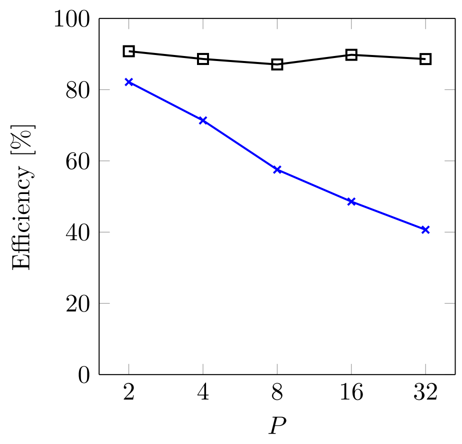


Figure 8: Parallel efficiency for solving the advection-diffusion equation. □ PEBK; × Paraexp/CN.

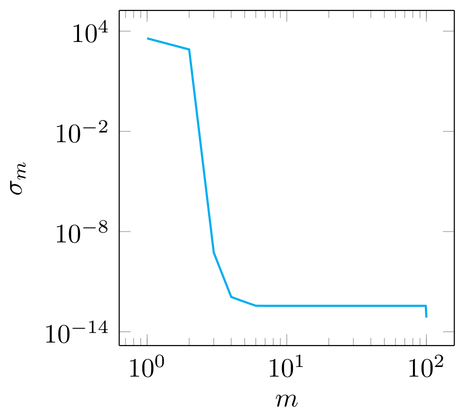


Figure 9: Singular values of the matrix composed of the source term samples.

Table 1: Parallel effiency of the Paraxp-EBK method and the Paraexp/Crank–Nicolson method for the advection-diffusion equation, with number of processors P. Timing τ_0 corresponds to the serial algorithm. For the parallel algorithm, τ_1 and τ_2 are timings of the nonhomogeneous and homogeneous problem respectively.

		Serial		Parallel			
	P	τ_0	Error	$\max(\tau_1)$	$\max(\tau_2)$	Error	*Efficiency*
PEBK	2	1.43e+00	3.05e-04	7.22e-01	6.44e-02	2.70e-04	91 %
	4	2.81e+00	3.05e-04	7.23e-01	7.06e-02	2.68e-04	89 %
	8	5.59e+00	3.05e-04	7.40e-01	6.21e-02	2.88e-04	87 %
	16	1.14e+01	3.05e-04	7.30e-01	6.28e-02	3.22e-04	90 %
	32	2.27e+01	3.05e-04	7.33e-01	6.63e-02	3.65e-04	89 %
Paraexp	2	2.15e+00	4.56e-04	1.29e+00	1.66e-02	4.10e-04	82 %
	4	4.37e+00	4.56e-04	1.52e+00	1.28e-02	3.80e-04	71 %
	8	8.56e+00	4.56e-04	1.85e+00	1.24e-02	3.59e-04	58 %
	16	1.70e+01	4.56e-04	2.18e+00	1.31e-02	3.43e-04	49 %
	32	3.43e+01	4.56e-04	2.62e+00	1.16e-02	3.32e-04	41 %

4. Burgers equation

In the previous section, the PEBK method was applied to a linear PDE. In this section, the performance of the PEBK method is tested on a nonlinear PDE, the viscous Burgers equation. In all tests $K = 10$ nonlinear iterations are carried out.

4.1. Travelling wave

Consider the viscous Burgers equation,

$$u_t + uu_x = \nu u_{xx} + f(x,t), \quad x \in [0,1], t \in [0,1], \tag{38}$$

where $\nu \in \mathbb{R}$ denotes the diffusivity (or viscosity) coefficient. In the following experiments, we take $\nu = 10^{-2}$, such that the problem is dominated by the nonlinear convective term. As in the previous example, the boundary conditions are periodic. Exact solutions to the Burgers equation can be found by the Cole–Hopf transformation [27]. In this test case, we construct a desired solution by introducing an appropriate source term, as shown for the advection-diffusion equation in Section 3.2. The the source term balances the dissipation of energy due to diffusion, and prevents the solution of vanishing in the limit $t \to \infty$.

The Burgers equation is an important equation in fluid dynamics. It can be seen as a simplification of the Navier–Stokes equation, which retains the nonlinear convective term, uu_x. In the limit $\nu \to 0$, the nonlinearity may produce discontinuous solutions, or shocks, so that a typical solution may resemble a sawtooth wave. A sawtooth wave can be represented by an infinite Fourier series. A smooth version of the sawtooth wave can be obtained by the modified Fourier series,

$$u(\xi) = \frac{1}{2} - \sum_{k=1}^{k_{\max}} \frac{1}{\pi k} \sin(2\pi k \xi)\Phi(k,\epsilon), \quad \xi \in \mathbb{R}, \tag{39}$$

where k is the wavenumber, $k_{\max}$ a cutoff wavenumber, and $\Phi(k,\epsilon)$ is a smoothing function, which supresses the amplitudes associated to high wavenumbers:

$$\Phi(k,\epsilon) = \left[\frac{\pi k \epsilon/2}{\sinh(\pi k \epsilon/2)}\right], \tag{40}$$

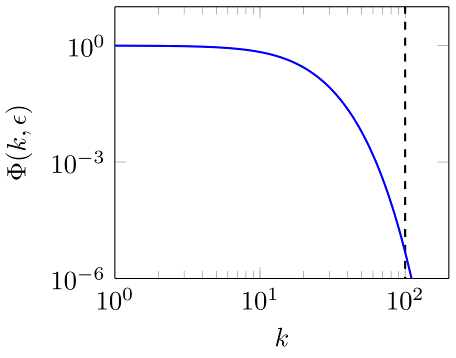
Here, ϵ is the smoothing parameter. The smoothing function $\Phi(k,\epsilon)$ is motivated by the viscosity-dependent inertial spectrum of the Burgers equation found by Chorin & Hald [11]. The smoothing function for $\epsilon = 0.1$

is shown in Figure 10a. As the smoothing function rapidly decreases with wavenumber, we choose a cutoff wavenumber of $k_{\max} = 100$. The value $\epsilon = 0.1$ is found to produce a smooth version of a sawtooth wave, as shown in Figure 10b.

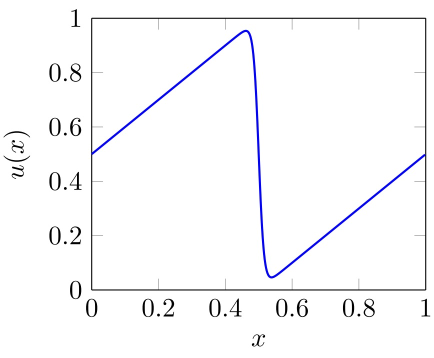
We consider a wave travelling in the positive x-direction by introducing the parametrization $\xi = x - \frac{1}{2}t + \frac{1}{2}$ in (38). The source term is then readily found by substituting the chosen solution (39) into the Burgers equation (38),

$$f(x,t) = u_t + uu_x - \nu u_{xx}. \tag{41}$$

The space derivatives in the Burgers equation are discretized with a second-order central finite difference schemes, using a uniform mesh width $\Delta x > 0$. The error of the numerical solution is again measured in the relative ℓ^2-norm, cf. (31).



(a) Smoothing function (solid line) and cutoff wavenumber (dashed line).



(b) Solution for $\epsilon = 0.1$ and $k_{max} = 100$.

Figure 10: Manufactured solution (39) of the Burgers equation.

The tolerance of the PEBK method is set to 10^{-4}. The SVD approximation of the source term, which includes the nonlinear term, is constructed from $s = 50$ samples per subinterval, and reveals that $m = 12$ singular values are sufficient in the truncated SVD, so that the error of the SVD approximation is less than the tolerance of the PEBK method.

The nonlinear system of ODEs is solved iteratively, as outlined in Section 2.3. Figure 11 shows the error history at different grid resolutions. Here, the error converges to a value that depends on the mesh width. In other words, the final error of the time integration method is much smaller than the error due to the spatial discretization.

The PEBK method is parallelized as discussed in Section 2.4. The time interval $[0, T]$ can be partitioned into P subintervals with a uniform subinterval size ΔT. In the following experiment, the complete time interval is extended with each subinterval added, $T = P\Delta T$. The mesh width is $\Delta x = 2.5 \cdot 10^{-3}$. Figure 12a shows that increasing the number of processors and hence the simulated time, generally increases the number of iterations required to achieve the same level of accuracy. The increased number of required iterations implies a decrease in theoretical parallel efficiency, which can be estimated by the ratio K_1/K_P, see (28).

Next, the final time is kept constant, and the subinterval size reduces with an increase in the number of processors, $\Delta T = T/P$. Figure 12b shows that the number of iterations is roughly independent of the number of processors, i.e., in this case, the parallel efficiency, K_1/K_P, does not decrease with P. Parallel speedup might also be improved by the fact that the SVD approximation converges faster on smaller subintervals, see Theorem 1. That is, fewer singular values have to be retained in the truncated SVD in order to achieve a certain accuracy. Also, the number of samples of the source term could be decreased on smaller subintervals.

4.2. Parallel efficiency

In this section, we test the parallel efficiency of the PEBK method for the viscous Burgers equation. In the following experiments, the source term is chosen such that the solution of the spatially discretized

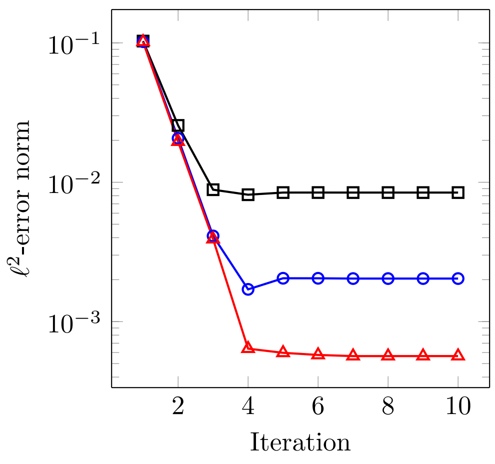
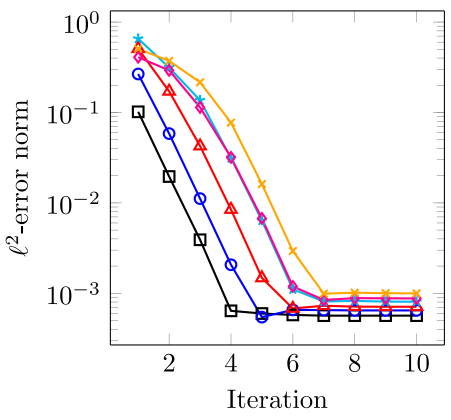
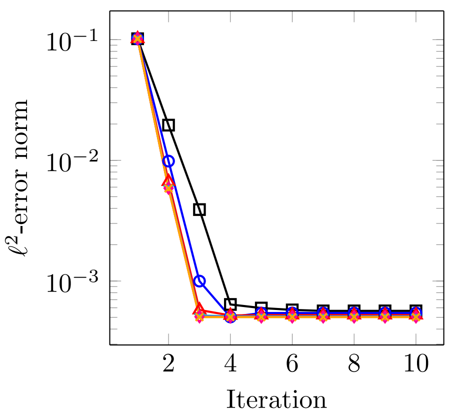


Figure 11: Error history with $\Delta T = 0.1$. $\square$, $\Delta x = 10^{-2}$; $\circ$, $\Delta x = 5 \cdot 10^{-3}$; $\triangle$, $\Delta x = 2.5 \cdot 10^{-3}$.



(a) $\Delta T = 0.1$, $T = P\Delta T$.



(b) $\Delta T = T/P$, $T = 0.1$.

Figure 12: Error history for the Paraxp-EBK method for a fixed subinterval size and a fixed final time. $\square$, $P = 1$; $\circ$, $P = 2$; $\triangle$, $P = 4$; $\star$, $P = 8$; $\diamond$, $P = 16$; $\times$, $P = 32$.

system is equal to the exact solution described in Section 4.1. In other words, the source term accounts for the error by the spatial discretization, and we measure only the error due to the time integration. The spatial discretization is here performed with a mesh width of $\Delta x = 2.5 \cdot 10^{-3}$.

If the subinterval size, ΔT, is fixed, the number of iterations generally increases in case the number of processors P increases, see Fig. 12. Increasing the number of iterations would reduce the parallel efficiency, according to (28). We therefore fix the final time T, such that the subinterval size decreases with increasing P. Also, the total number of samples over $[0, T]$ is fixed, such that the local number of samples decreases with increasing P. To keep the global distribution of samples constant in the parallel computations, the local sample points are uniformly distributed instead of Chebyshev points as used in previous experiments. In our experiments, we use $\Delta T = 0.2/P$ and $s = 128/P$. Furthermore, the number of (retained) singular values is $m = 12$, and the tolerance of the EBK method is set to 10^{-4}. The error history for different P is shown in Fig. 13. The results show that the convergence rate of the PEBK method can improve by increasing P. The nonlinear corrections, see (18), appear to be more effective on smaller subintervals.

As opposed to the linear problem in Section 3.2, communication between the parallel processes is here required because of the waveform relaxation method. To get a rough estimate of the best possible speedup, we measure the CPU time which would be required by different processors, on a serial computer. Per iteration of the PEBK method, the computation time of each individual process is measured, after which the maximum is stored, i.e., the computation time of the slowest process. The total computation time of the PEBK method is then taken as the sum, over the total number of iterations performed, of the maxima. In this experiment, we have measured a total of ten PEBK iterations.

The computation times for $\nu = 10^{-1}$ and $\nu = 10^{-2}$ are shown in Fig. 14a, which clearly demonstrate a considerable parallel speedup. The slightly higher timings at $\nu = 10^{-1}$ could be attributed to the increased stiffness of the problem. Figure 14b shows no significant influence of viscosity on the parallel efficiency of the PEBK method. However, at given Δx and ΔT, PEBK does not necessarily converge also at lower ν. For example at $\nu = 10^{-3}$ and $\nu = 10^{-4}$, we only reach convergence when P is high enough ($P \geq 8$ for $\nu = 10^{-4}$). The convergence generally improves with increasing P, because the subintervals become smaller and the nonlinear term is treated more accurately. We also include timings for $\nu = 10^{-3}$ and $\nu = 10^{-4}$ for a range of P values in Fig. 14a. These results also indicate that a similar parallel speedup can be achieved for $\nu = 10^{-3}$ and $\nu = 10^{-4}$. The parallel efficiency is not reported however because a reference point for $P = 1$ is not available as convergence is lacking.

The convergence of PEBK for lower values of ν can also be improved by decreasing the final time T. Figure 15a shows the timings for $\nu = 10^{-3}$ and $\nu = 10^{-4}$ using $\Delta T = 0.05/P$. In this case, the spatial resolution was $\Delta x = 10^{-4}$ in order to reduce the numerical dissipation with respect to the physical dissipation. Figure 15b indicates that a parallel speedup can be also achieved for $\nu = 10^{-3}$ and $\nu = 10^{-4}$. The parallel efficiency of the Parareal algorithm is known to deteriorate for similarly small viscosity coefficients [50]. Furthermore, we note that the actual parallel efficiency of the PEBK method may even be better in practice, since the required number of PEBK iterations could decrease with higher P. Also, the number of singular values, m, could possibly be reduced on smaller subintervals, based on (1), which would further enhance the potential parallel speedup.

The PFASST algorithm [17] shows a good parallel efficiency for the viscous Burgers equation as well. It is unknown whether changing the viscosity coefficient has an impact on the performance of PFASST. Our observations point in the direction of a good parallel efficiency of the PEBK method for simulations of the Navier–Stokes equation at high Reynolds numbers. For these problems there is a high demand for highly efficient parallel solvers [54]. The Parareal algorithm was for example reported to perform poorly in advection-dominated problems [18, 50].

4.3. Multiscale example

Turbulent flow is a multiscale phenomenon that is characterized by a wide range of length and time scales. In case of the Burgers equation, we can emulate such a solution by combining two functions that are

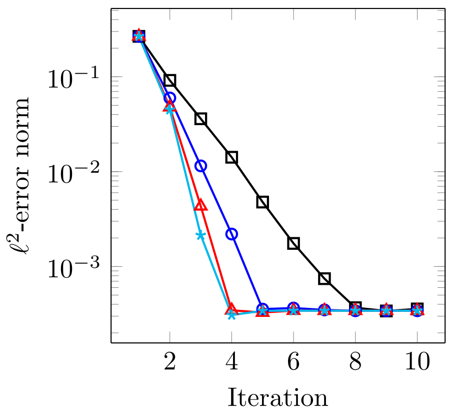
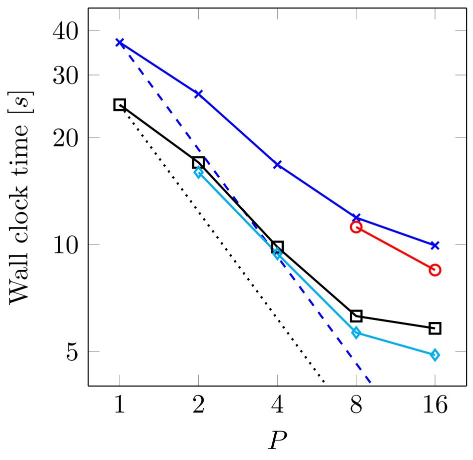
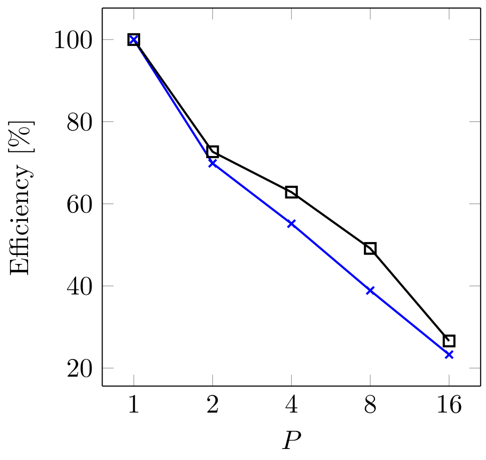


Figure 13: Error history for the Paraxp-EBK method with $\Delta T = 0.2/P$. $\square$, $P = 1$; $\circ$, $P = 2$; $\triangle$, $P = 4$; $\star$, $P = 8$.

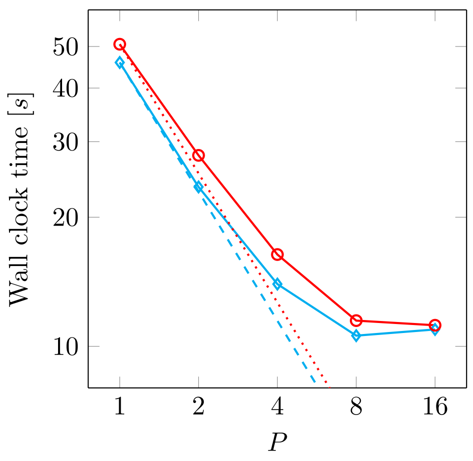


(a) $\times$ $\nu = 10^{-1}$; $\square$ $\nu = 10^{-2}$; $\diamond$ $\nu = 10^{-3}$; $\circ$ $\nu = 10^{-4}$; dotted line, $\nu = 10^{-1}$ (ideal); dashed line, $\nu = 10^{-2}$ (ideal).

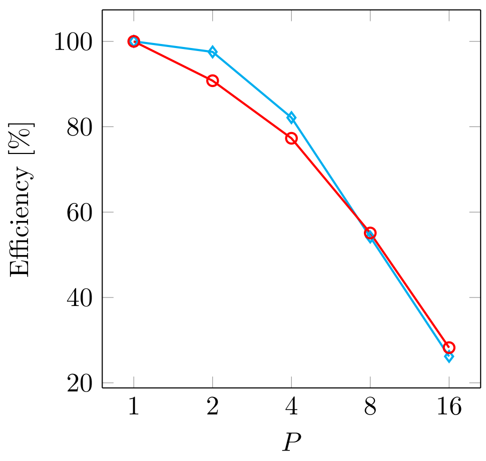


(b) $\times$: $\nu = 10^{-1}$; $\square$: $\nu = 10^{-2}$.

Figure 14: Total computation time (left) and parallel efficiency (right) of ten PEBK iterations with $\Delta T = 0.2/P$ and $\Delta x = 2.5 \cdot 10^{-3}$.



(a) ◇ $\nu = 10^{-3}$; ○ $\nu = 10^{-4}$; dotted line, $\nu = 10^{-4}$ (ideal); dashed line, $\nu = 10^{-3}$ (ideal).



(b) ◇ $\nu = 10^{-3}$; ○ $\nu = 10^{-4}$.

Figure 15: Total computation time (left) and parallel efficiency (right) of ten PEBK iterations with $\Delta T = 0.05/P$ and $\Delta x = 2.5 \cdot 10^{-4}$.

periodic in both space and time, e.g.,

$$u(x,t) = \sin(2\pi x)\sin(2\pi t) + \frac{1}{k_0}\sin(2k_0\pi x)\sin(2k_0\pi t), \quad k_0 > 1. \tag{42}$$

The solution features a large scale mode with wavenumber one, and a smaller scale mode with wavenumber $k_0 > 1$. This combination allows the construction of an arbitrarily wide dynamic range. The factor $1/k_0$ is included in compliance with an assumed energy distribution of $[\hat{u}(k)]^2 \propto k^{-2}$ [11], where $\hat{u}(k)$ is the Fourier transform,

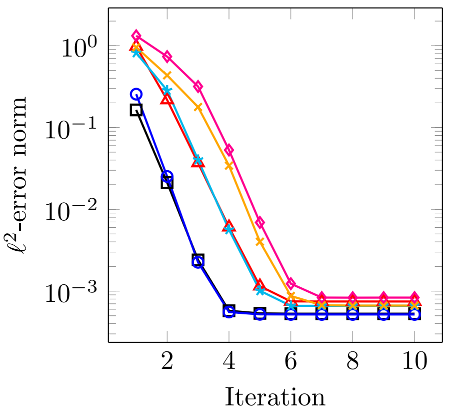
$$\hat{u}(k) = \int_{-\infty}^{\infty} u(x,t)\exp(-2\pi ik(x+t))\,dx\,dt, \tag{43}$$

where we have used the fact that the solution is periodic in space and time. The previous experiment, see Fig. 12a, is repeated for the manufactured solution (42) with different values of k_0. Figure 16 illustrates that widening the spectrum does not significantly affect the convergence of the PEBK iterations. Remarkably, the curves appear to form pairs based on P.

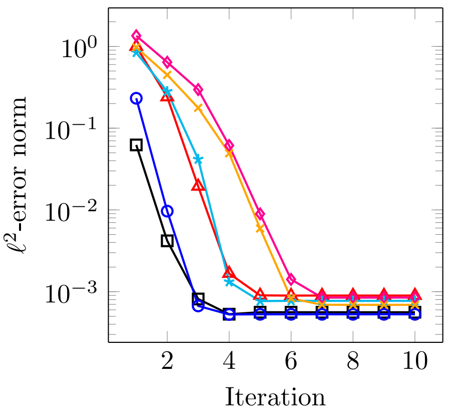
5. Conclusions

We propose an implementation of the Paraexp method with enhanced parallelism based on the exponential block Krylov (EBK) method. Furthermore, the method, Paraexp-EBK (PEBK), is extended to solve nonlinear PDEs iteratively by a waveform relaxation method. The nonlinear terms are represented by a source term in a nonhomogeneous linear system of ODEs. Each iteration the source term is updated with the latest solution. The convergence of the iterative process can be accelerated by adding a correction term based on the Jacobian matrix of the nonlinear term. Each iteration the initial value problem can then be decoupled into independent subproblems, which can be solved parallel in time. Essentially, we implement the Paraexp method within a waveform relaxation approach in order to integrate nonlinear PDEs. Also, the Paraexp-EBK (PEBK) method is used to integrate both the homogeneous and the nonhomogeneous parts of the subproblems. This is in contrast to the original Paraexp method, which assumes a convential time integration method for the nonhomogeneous parts.

The PEBK method is tested on the advection-diffusion equation for which we demonstrate the parallelization concept for linear PDEs. The parallelization also works in cases without diffusion present, in which



(a) $k_0 = 4$



(b) $k_0 = 16$

Figure 16: Error history with $\Delta T = 0.1$ and $T = P\Delta T$. □, $P = 1$; ○, $P = 2$; △, $P = 4$; ⋆, $P = 8$; ◇, $P = 16$; ×, $P = 32$.

the PDE is purely hyperbolic. The parallel efficiency is compared with a Crank–Nicolson (CN) scheme parallelized with the Paraexp algorithm. The parallel efficiency of the PEBK method remains roughly constant around 90%. On the other hand, the parallel efficiency of the CN/Paraexp combination steadily decreases with the number of processors.

As a model nonlinear PDE, the viscous Burgers equation is solved. The number of waveform relaxation iterations required for a certain error tolerance increases, when the relative importance of nonlinearity grows by decreasing the viscosity coefficient. Good parallel efficiency of the EBK method was observed for different values of the viscosity coefficient.

We present numerical experiments using one-dimensional equations, but the PEBK method is also applicable to equations in two or three dimensions. The parallel performance for higher-dimensional problems is subject of future study. Since the nonlinear convective term in the Burgers equation is shared by the Navier–Stokes equation, the presented results give a hint of the potential of the PEBK method as an efficient parallel solver in turbulent fluid dynamics, where nonlinearity plays a key role. The question remains how to treat the pressure in the incompressible Navier–Stokes equation, which enforces the incompressibility constraint on the velocity field (see for possible approaches in [15, 44]). This will be explored in future work.

Acknowledgements

Part of the contribution of the second author, in particular in Section 2, was supported via the Russian Science Foundation Grant 14-11-00659.

References

[1] M. Afanasjew, M. Eiermann, O. G. Ernst, and S. Güttel. Implementation of a restarted Krylov subspace method for the evaluation of matrix functions. *Linear Algebra Appl.*, 429:2293–2314, 2008.

[2] M. Botchev. A block Krylov subspace time-exact solution method for linear ordinary differential equation systems. *Numerical linear algebra with applications*, 20(4):557–574, 2013.

[3] M. Botchev, G. Sleijpen, and A. Sopaheluwakan. An SVD-approach to Jacobi–Davidson solution of nonlinear Helmholtz eigenvalue problems. *Linear Algebra and its Applications*, 431(3):427–440, 2009.

[4] M. A. Botchev, I. Faragó, and A. Havasi. Testing weighted splitting schemes on a one-column transport-chemistry model. *Int. J. Environment and Pollution*, 22(1/2):3–16, 2004.

[5] M. A. Botchev, V. Grimm, and M. Hochbruck. Residual, restarting, and Richardson iteration for the matrix exponential. *SIAM Journal on Scientific Computing*, 35(3):A1376–A1397, 2013.

[6] M. A. Botchev, I. V. Oseledets, and E. E. Tyrtyshnikov. Iterative across-time solution of linear differential equations: Krylov subspace versus waveform relaxation. *Computers & Mathematics with Applications*, 67(12):2088–2098, 2014. http://dx.doi.org/10.1016/j.camwa.2014.03.002.

[7] M. A. Botchev and H. A. van der Vorst. A parallel nearly implicit scheme. *Journal of Computational and Applied Mathematics*, 137:229–243, 2001.

[8] K. Burrage. *Parallel and sequential methods for ordinary differential equations.* The Clarendon Press Oxford University Press, New York, 1995. Oxford Science Publications.

[9] A. Cella and M. Lucchesi. Space-time finite elements for the wave propagation problem. *Meccanica*, 10(3):168–170, 1975. http://dx.doi.org/10.1007/BF02149028.

[10] E. Celledoni and I. Moret. A Krylov projection method for systems of ODEs. *Applied numerical mathematics*, 24(2):365–378, 1997.

[11] A. J. Chorin and O. H. Hald. Viscosity-dependent inertial spectra of the Burgers and Korteweg–deVries–Burgers equations. *Proceedings of the National Academy of Sciences of the United States of America*, 102(11):3921–3923, 2005.

[12] J. J. B. de Swart. *Parallel Software for Implicit Differential Equations.* PhD thesis, University of Amsterdam, Nov. 1997. ISBN 90-74795-77-3.

[13] S. V. Dolgov, B. N. Khoromskij, and I. V. Oseledets. Fast solution of parabolic problems in the tensor train/quantized tensor train format with initial application to the fokker–planck equation. *SIAM Journal on Scientific Computing*, 34(6):A3016–A3038, 2012. http://dx.doi.org/10.1137/120864210.

[14] J. Dongarra et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, page 1094342010391989, 2011.

[15] W. Edwards, L. Tuckerman, R. Friesner, and D. Sorensen. Krylov methods for the incompressible Navier–Stokes equations. *Journal of Computational Physics*, 110(1):82 – 102, 1994.

[16] M. Eiermann, O. G. Ernst, and S. Güttel. Deflated restarting for matrix functions. *SIAM J. Matrix Anal. Appl.*, 32(2):621–641, 2011.

[17] M. Emmett and M. Minion. Toward an efficient parallel in time method for partial differential equations. *Communications in Applied Mathematics and Computational Science*, 7(1):105–132, 2012.

[18] P. F. Fischer, F. Hecht, and Y. Maday. A Parareal in time semi-implicit approximation of the Navier–Stokes equations. In *Domain decomposition methods in science and engineering*, pages 433–440. Springer, 2005.

[19] M. Gander and S. Vandewalle. Analysis of the Parareal Time-Parallel Time-Integration Method. *SIAM Journal on Scientific Computing*, 29(2):556–578, 2007.

[20] M. J. Gander. Analysis of the parareal algorithm applied to hyperbolic problems using characteristics. *Boletin de la Sociedad Espanola de Matemática Aplicada*, 42:21–35, 2008.

[21] M. J. Gander and S. Güttel. Paraexp: A parallel integrator for linear initial-value problems. *SIAM Journal on Scientific Computing*, 35(2):C123–C142, 2013.

[22] T. Göckler and V. Grimm. Uniform approximation of φ-functions in exponential integrators by a rational Krylov subspace method with simple poles. *SIAM Journal on Matrix Analysis and Applications*, 35(4):1467–1489, 2014. http://dx.doi.org/10.1137/140964655.

[23] G. H. Golub and C. F. Van Loan. Matrix computations. *Johns Hopkins University, Press, Baltimore, MD, USA*, pages 374–426, 1996.

[24] W. Hackbusch. Parabolic multigrid methods. In *Computing methods in applied sciences and engineering, VI (Versailles, 1983)*, pages 189–197. North-Holland, Amsterdam, 1984.

[25] M. Hochbruck and J. Niehoff. Approximation of matrix operators applied to multiple vectors. *Mathematics and Computers in Simulation*, 79(4):1270–1283, 2008.

[26] M. Hochbruck and A. Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[27] E. Hopf. The partial differential equation $u_t + uu_x = \mu u_{xx}$. *Communications on Pure and Applied mathematics*, 3(3):201–230, 1950.

[28] R. A. Horn and C. R. Johnson. Matrix Analysis. *Cambridge University Press*, 1986.

[29] R. A. Horn and C. R. Johnson. Topics in Matrix Analysis. *Cambridge University Press*, 1991.

[30] G. Horton and S. Vandewalle. A space-time multigrid method for parabolic partial differential equations. *SIAM J. Sci. Comput.*, 16(4):848–864, 1995. http://dx.doi.org/10.1137/0916050.

[31] T. J. R. Hughes and G. M. Hulbert. Space-time finite element methods for elastodynamics: formulations and error estimates. *Comput. Methods Appl. Mech. Engrg.*, 66(3):339–363, 1988. http://dx.doi.org/10.1016/0045-7825(88)90006-0.

[32] W. Hundsdorfer and J. G. Verwer. *Numerical solution of time-dependent advection-diffusion-reaction equations*, volume 33. Springer Science & Business Media, 2013.

[33] O. Koch and C. Lubich. Dynamical low-rank approximation. *SIAM Journal on Matrix Analysis and Applications*, 29(2):434–454, 2007.

[34] E. Lelarasmee, A. E. Ruehli, and A. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 1(3):131–145, July 1982.

[35] G. Li. A block variant of the GMRES method on massively parallel processors. *Parallel Computing*, 23(8):1005–1019, 1997.

[36] J.-L. Lions. *Équations différentielles opérationnelles et problèmes aux limites.* Die Grundlehren der mathematischen Wissenschaften, Bd. 111. Springer-Verlag, Berlin-Göttingen-Heidelberg, 1961.

[37] J.-L. Lions, Y. Maday, and G. Turinici. Résolution d'edp par un schéma en temps pararéel . *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 332(7):661 – 668, 2001.

[38] C. Lubich and A. Ostermann. Multi-grid dynamic iteration for parabolic equations. *BIT Numerical Mathematics*, 27:216–234, 1987. 10.1007/BF01934186.

[39] U. Miekkala and O. Nevanlinna. Convergence of dynamic iteration methods for initial value problems. *SIAM Journal on*

Scientific and Statistical Computing, 8(4):459–482, 1987.

[40] M. Minion. A hybrid parareal spectral deferred corrections method. *Communications in Applied Mathematics and Computational Science*, 5(2):265–301, 2011.

[41] I. Moret and P. Novati. RD rational approximations of the matrix exponential. *BIT*, 44:595–615, 2004.

[42] K. W. Morton and D. F. Mayers. *Numerical solution of partial differential equations: an introduction.* Cambridge university press, 2005.

[43] O. Nevanlinna. Remarks on Picard-Lindelöf iteration. *BIT Numerical Mathematics*, 29(2):328–346, 1989.

[44] C. K. Newman. *Exponential Integrators for the Incompressible Navier–Stokes Equations.* PhD thesis, Virginia Polytechnic Institute and State University, 2004.

[45] A. R. Newton and A. L. Sangiovanni-Vincentelli. Relaxation-based electrical simulation. *IEEE Transactions on Electron Devices*, 30(9):1184–1207, 1983.

[46] J. Niehoff. *Projektionsverfahren zur Approximation von Matrixfunktionen mit Anwendungen auf die Implementierung exponentieller Integratoren.* PhD thesis, Mathematisch-Naturwissenschaftlichen Fakultät der Heinrich-Heine-Universität Düsseldorf, December 2006.

[47] A. Nikishin and A. Y. Yeremin. Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, I: General iterative scheme. *SIAM Journal on Matrix Analysis and Applications*, 16(4):1135–1153, 1995.

[48] D. Ruprecht and R. Krause. Explicit parallel-in-time integration of a linear acoustic-advection system. *Computers & Fluids*, 59:72–83, 2012.

[49] G. A. Staff and E. M. Rønquist. Stability of the parareal algorithm. In *Domain decomposition methods in science and engineering*, pages 449–456. Springer, 2005.

[50] J. Steiner, D. Ruprecht, R. Speck, and R. Krause. Convergence of Parareal for the Navier–Stokes equations depending on the Reynolds number. In *Numerical Mathematics and Advanced Applications-ENUMATH 2013*, pages 195–202. Springer, 2015.

[51] H. Tal-Ezer. On restart and error estimation for Krylov approximation of $w = f(A)v$. *SIAM J. Sci. Comput.*, 29(6):2426–2441, 2007.

[52] J. van den Eshof and M. Hochbruck. Preconditioning Lanczos approximations to the matrix exponential. *SIAM J. Sci. Comput.*, 27(4):1438–1457, 2006.

[53] P. J. van der Houwen and B. P. Sommeijer. CWI contributions to the development of parallel Runge-Kutta methods. *CWI Quarterly*, 11(1):33–53, 1998. Solving differential equations on parallel computers.

[54] E. P. van der Poel, R. Ostilla-Mónico, J. Donners, and R. Verzicco. A pencil distributed finite difference code for strongly turbulent wall-bounded flows. *Computers & Fluids*, 116(0):10–16, 2015.

[55] J. J. W. van der Vegt and H. van der Ven. Space-time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows. I. General formulation. *J. Comput. Phys.*, 182(2):546–585, 2002.

[56] J. G. Verwer and J. M. Sanz-Serna. Convergence of method of lines approximations to partial differential equations. *Computing*, 33(3-4):297–313, 1984.

[57] J. White, F. Odeh, A. L. Sangiovanni-Vincentelli, and A. Ruehli. Waveform relaxation: Theory and practice. Technical Report UCB/ERL M85/65, EECS Department, University of California, Berkeley, 1985.

[58] J. Zhu. A new parallel algorithm for the numerical solutions of time dependent partial differential equations (unpublished note). Engineering Research Center, Mississippi State University, 2000.