SIAM J. SCI. COMPUT.
Vol. 41, No. 6, pp. C603–C626

AN EFFICIENT PARALLEL-IN-TIME METHOD FOR OPTIMIZATION WITH PARABOLIC PDEs[*]

SEBASTIAN GÖTSCHEL[†] AND MICHAEL L. MINION[‡]

Abstract. To solve optimization problems with parabolic PDE constraints, often methods working on the reduced objective functional are used. They are computationally expensive due to the necessity of solving both the state equation and a backward-in-time adjoint equation to evaluate the reduced gradient in each iteration of the optimization method. In this study, we investigate the use of the parallel-in-time method PFASST in the setting of PDE-constrained optimization. In order to develop an efficient fully time-parallel algorithm, we discuss different options for applying PFASST to adjoint gradient computation, including the possibility of doing PFASST iterations on both the state and the adjoint equations simultaneously. We also explore the additional gains in efficiency from reusing information from previous optimization iterations when solving each equation. Numerical results for both a linear and a nonlinear reaction-diffusion optimal control problem demonstrate the parallel speedup and efficiency of different approaches.

Key words. PDE-constrained optimization, parallel-in-time methods, PFASST

AMS subject classifications. 65K10, 65M55, 65M70, 65Y05

DOI. 10.1137/19M1239313

1. Introduction.

Large-scale PDE-constrained optimization problems occur in a multitude of applications, for example in solving inverse problems for nondestructive testing of materials and structures [20] or in individualized medicine [15]. More recently, the training of certain deep neural networks in machine learning, e.g, for image recognition or natural language processing, has been formulated as a dynamic optimal control problem [27, 41]. Algorithms for the solution of PDE-constrained optimization problems are computationally extremely demanding, as they require one to numerically solve multiple PDEs during the iterative optimization process. This is especially challenging for transient problems, where the solution of the associated optimality system requires information about the discretized variables on the whole space-time domain. For the solution of such optimization problems, methods working on the reduced objective functional are often employed to avoid a full spatio-temporal discretization of the problem. The evaluation of the reduced gradient then requires one solve of the state equation forward in time and one backward-in-time solve of the adjoint equation. In order to tackle realistic applications, it is not only essential to devise efficient discretization schemes for optimization, but also to use advanced

[*] Submitted to the journal's Software and High-Performance Computing section January 17, 2019; accepted for publication (in revised form) September 18, 2019; published electronically December 12, 2019. The U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. Copyright is owned by SIAM to the extent not limited by these rights.
https://doi.org/10.1137/19M1239313

Funding: The work of the first author was partially supported by the Deutsche Forschungsgemeinschaft (DFG), project WE 2937/6-1. The work of the second author was supported by the Applied Mathematics Program of the DOE Office of Advanced Scientific Computing Research under U.S. Department of Energy contract DE-AC02-05CH11231 and by the Alexander von Humboldt Foundation.

[†] Numerical Mathematics, Zuse Institute Berlin, Berlin, 14195, Germany (goetschel@zib.de, http://www.zib.de/goetschel).

[‡] Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (mlminion@lbl.gov, https://crd.lbl.gov/michael-minion).

techniques to exploit computer architectures and decrease the time-to-solution, which otherwise is prohibitively long. One approach is to utilize the number of CPU cores of current and future many-core high performance computing systems by parallelizing the PDE solution method. In addition to well-established methods for parallelization in the spatial degrees of freedom, parallel-in-time methods have seen a growing interest in the last 15 years. Research into time parallelism dates back at least to the 1960s, and in 2001 the introduction of parareal by Lions, Maday, and Turinici [34] sparked new research into time-parallel methods. The methods in this paper are based on the parallel full approximation scheme in space and time (PFASST) introduced by Emmett and Minion [13]. We will not attempt a thorough review of the field here, and the interested reader is encouraged to consult the survey article [17] for an overview of competing approaches.

More recently, the application of space-time parallel methods to the solution of optimization problems governed by PDEs has become an active research area, with approaches including multiple shooting (see, e.g., [29] and the references therein), Schwarz methods [5, 19], and the application of parareal preconditioners [35, 44]. A time-parallel gradient-type method is presented in [10]. There the time interval of interest is subdivided into time steps, which are solved in parallel using quantities from the previous optimization iteration as input. This leads to jumps in the solutions of state and adjoint equations such that these equations are not satisfied during optimization. While they report excellent speedups and linear scaling up to 50 processors and show convergence if sufficiently small step sizes for updating the control are used, it is unclear how to automatically select such a step size. Alternatively, space-time parallel multigrid methods are applied to adjoint gradient computation and simultaneous optimization [23, 24] within the XBraid software library [4]. XBraid provides a nonintrusive framework adding time parallelism to existing serial time stepping codes, and using simultaneous instead of reduced space optimization, a speedup of 19 using 256 time processors has been reported. The same method is also applied to perform layer-parallel training of neural networks [25].

In this paper, we employ PFASST to provide a fully time-parallel reduced-space gradient- or nonlinear conjugate gradient method that allows using the usual line search criteria, e.g., the strong Wolfe conditions, for step size selection to guarantee convergence, and is thus nonintrusive with respect to the optimization algorithm. On the other hand, this is an intrusive approach concerning the PDE solvers, as it requires using multilevel spectral deferred correction methods as time steppers. The implementation effort is mitigated by the availability of libraries like LibPFASST [2] or dune-PFASST [1]. While the basic ideas of the approach are outlined in the short paper [21], here we provide more details and develop additional approaches to increase speedup and efficiency. We demonstrate that using PFASST brings additional benefits, like flexibility in treating nonlinearities and, more importantly, the possibility of warm starting the PDE solutions required during the optimization. The remainder of the paper is organized as follows. In section 2, we present the optimization problem and review optimality conditions as well as adjoint gradient computation. The PFASST method is introduced in section 3; it is used to derive parallel-in-time methods for solving optimization problems with parabolic PDEs in section 4. Finally, in section 5 we present numerical examples, followed by a discussion of results and future improvements in section 6.

2. Adjoint gradient computation for optimization with parabolic PDEs.

Here we briefly summarize the mathematical approach to parabolic PDE-constrained optimization problems. For more details and generalizations, we refer the reader to, e.g., [31].

We consider optimization problems of the form

$$\min_{y\in Y, u\in U} J(y,u) \text{ subject to } c(y,u)=0, \tag{2.1}$$

with the equality constraint $c: Y \times U \to Z^\star$ being a parabolic PDE on Hilbert spaces Y, U, Z. $Z^\star, U^\star$ denote the dual spaces of Z and U, respectively; the dual pairing between a space X and its dual is denoted by $\langle \cdot, \cdot \rangle_{X^\star, X}$, and $(\cdot,\cdot)_X$ denotes the scalar product on X. We drop the subscripts like $\cdot_X$ if the involved spaces are clear from the context. In the present setting, the constraint $c(y,u)=0$ means that the *state* y satisfies a PDE where the *control* u is a specific forcing term, occurring, e.g., as a source term, in the boundary conditions, or as some other parameter.

To derive optimality conditions, we assume that there exists a unique solution $y = y(u) \in Y$ of the state equation $c(y,u)=0$ for each control $u \in U$. We additionally assume that $c_y(y,u) : Y \to Z^\star$ is continuously invertible. Then, by the implicit function theorem (see, e.g., [46, section 4.7]), the control-to-state mapping is continuously differentiable, and the derivative $y'(u)$ is given by the solution of

$$c_y(y,u)y'(u) + c_u(y,u) = 0. \tag{2.2}$$

Subscripts like $c_u()$ denote the partial derivatives with respect to the indicated variable. By inserting $y(u)$ into the optimization problem (2.1) we arrive at the reduced problem

$$\min_{u\in U} j(u) := J(y(u),u). \tag{2.3}$$

In this unconstrained setting, the following simple first-order necessary optimality condition holds. If $u^\star \in U$ is a local solution of the reduced problem (2.3), it is a zero of the reduced derivative, $j'(u^\star) = 0$. If the reduced functional j is convex, this condition is also sufficient. If we allow control constraints, i.e., demand $u \in U_{\text{ad}}$ with $U_{\text{ad}} \subset U$ nonempty, convex, and closed, the optimality condition changes to the variational inequality for the local minimizer $u^\star \in U_{\text{ad}}$:

$$\langle j'(u^\star), u - u^\star \rangle_{U^\star, U} \geq 0 \quad \forall u \in U_{\text{ad}}. \tag{2.4}$$

To formally derive a representation for the reduced gradient, we define the Lagrange functional $\mathcal{L} : Y \times U \times Z \to \mathbb{R}$,

$$\mathcal{L}(y,u,p) = J(y,u) + \langle p, c(y,u) \rangle_{Z, Z^\star}, \tag{2.5}$$

where in the present context, the Lagrange multiplier $p \in Z$ is referred to as the *adjoint*. Clearly, inserting $y = y(u)$ into (2.5), we get $j(u) = \mathcal{L}(y(u), u, p)$ for arbitrary $p \in Z$. Differentiation in direction $\delta u \in U$ yields

$$\langle j'(u), \delta u \rangle_{U^\star, U} = \langle \mathcal{L}_y(y(u),u,p), y'(u)\delta u \rangle_{Y^\star, Y} + \langle \mathcal{L}_u(y(u),u,p), \delta u \rangle_{U^\star, U}. \tag{2.6}$$

Choosing $p = p(u)$ such that the adjoint equation

$$c_y(y(u),u)^\star p(u) = -J_y(y(u),u) \tag{2.7}$$

is fulfilled gives

$$\mathcal{L}_y(y(u),u,p(u)) = J_y(y(u),u) + c_y(y(u),u)^\star p(u) = 0. \tag{2.8}$$

Inserting this into (2.6), the first term on the right-hand side vanishes, and we get the reduced derivative $j'(u) \in U^\star$ as

$$\langle j'(u), \delta u\rangle_{U^\star,U} = \langle \mathcal{L}_u(y(u), u, p), \delta u\rangle_{U^\star,U},$$

i.e.,

$$j'(u) = \mathcal{L}_u(y(u), u, p(u)) = J_u(y(u), u) + c_u(y(u), u)^\star p(u). \tag{2.9}$$

In the Hilbert space setting used here, for a given $u \in U$ the reduced gradient $\nabla j(u) \in U$ is then given as the Riesz representative of the reduced derivative $j'(u) \in U^\star$, i.e., via

$$\big(\delta u, \nabla j(u)\big)_U = j'(u)\delta u \quad \forall \delta u \in U.$$

To be more concrete, we consider a distributed tracking-type objective functional $J(y, u)$ with an additional term penalizing the control cost for $\lambda \geq 0$,

$$J(y, u) = \underbrace{\frac{1}{2}\int_0^T \|y - y_d\|^2_{L^2(\Omega)}\; dt}_{=J^\Omega(y,u)} + \underbrace{\frac{\lambda}{2}\int_0^T \|u\|_{L^2(\Omega)}\; dt}_{=J^u(y,u)}, \tag{2.10}$$

with y_d representing a desired solution. The state y is subject to a linear or semilinear parabolic PDE with distributed control

$$y_t - \kappa\Delta y + f(y) - u = 0 \quad \text{in } \Omega \times [0, T], \tag{2.11}$$

$$y(0) - y_0 = 0 \quad \text{in } \Omega \tag{2.12}$$

and suitable boundary conditions. For simplicity, we consider a constant scalar diffusivity κ; Δy denotes the Laplacian of y. Thus, for sufficiently smooth f, we choose $Y = U = Z = L^2(0, T; H)$, with $H = L^2(\Omega)$ in (2.1). Here, $\Omega \subset \mathbb{R}^n$, and $T > 0$ denotes a given final time. The adjoint equation (2.7) for a given state solution $\bar{y}$ becomes

$$-p_t - \kappa\Delta p + f_y(\bar{y})p = -J^\Omega_y(\bar{y}, u) = -(\bar{y} - y_d) \quad \text{in } \Omega \times [0, T], \tag{2.13}$$

$$p(T) = 0 \quad \text{in } \Omega, \tag{2.14}$$

with homogeneous boundary conditions of the same type as in the state equation. Generalizations to controls or observations on the boundary are straightforward. Having a terminal cost contribution in the objective, e.g.,

$$J^T(y, u) = \frac{\sigma}{2}\left\|y(T) - y^T_d\right\|^2_{L^2(\Omega)},$$

leads to the modified terminal condition

$$p(T) = -J^T_y(\bar{y}, u) = -(y(T) - y^T_d). \tag{2.15}$$

Since the adjoint equation (2.7) is backward in time, due to the occurrence of $-J_y(y(u), u)$ as a source term and—in the nonlinear case—the dependence of $c_y(y(u), u)$ on the state solution $y(u)$, adjoint gradient computation consists of three steps (see also Figure 2.1):

1. Solve the PDE $c(y, u) = 0$ for a given control u, and store the solution trajectory $y \in Y$.

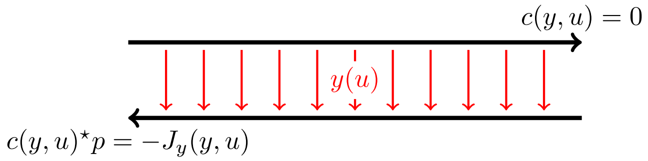

FIG. 2.1. *Adjoint gradient computation. The full solution of the forward state equation is required to solve the adjoint equation backward in time.*

2. Solve the adjoint PDE $c_y(y,u)^\star p = -J_y(y,u)$ for $p \in Z$.
3. Set $j'(u) = J_u(y,u) + c_u(y,u)^\star p$, and compute the Riesz representation $\nabla j(u)$.

Thus, the computation of the reduced gradient requires the solution of two parabolic PDEs. For solving the optimization problem, in this work we consider nonlinear conjugate gradient (ncg)- and steepest descent (sd) methods, as they require only gradient information. Hence gradient-based methods for the optimization problems that iterate over these three steps require many forward and backward PDE solves. The goal of this paper is to exploit properties of the PFASST parallel-in-time algorithm to reduce the computational complexity of the optimization procedure.

Remark 2.1. State and adjoint equations can be solved as a coupled system, as is done, e.g., in [26]. In view of parallelizing in time, this becomes more difficult; see the brief discussion in section 4.2 and Figure 4.2; our main focus thus is on gradient computation and using algorithms like sd or ncg to iteratively solve the optimization problem. This can easily be applied to, e.g., control constrained problems, and be extended to Newton-CG and second-order methods like (semismooth) Newton.

For the gradient-based methods considered here, the optimization iteration proceeds as

$$u_{k+1} = u_k + \alpha_k d_k, \tag{2.16}$$
$$d_{k+1} = -\nabla j(u_{k+1}) + \beta_k d_k, \tag{2.17}$$

where $d_0 = \nabla j(u_0)$, and the choice of β_k defines the actual method. Abbreviating $\nabla j(u_k)$ by g_k, specific variants of ncg include, e.g.,

$$\beta_k^{\mathrm{FR}} = \frac{(g_{k+1}, g_{k+1})}{(g_k, g_k)} \qquad \text{Fletcher–Reeves [16],}$$
$$\beta_k^{\mathrm{PRP}} = \frac{(g_{k+1}, g_{k+1} - g_k)}{(g_k, g_k)} \qquad \text{Polak–Ribiere–Polyak [39, 40],}$$
$$\beta_k^{\mathrm{DY}} = \frac{(g_{k+1}, g_{k+1})}{(d_k, g_{k+1} - g_k)} \qquad \text{Dai–Yuan [9].}$$

Using $\beta_k = 0$ yields the usual steepest descent method.

To guarantee convergence under the usual assumptions, the step size α_k is chosen to satisfy the strong Wolfe conditions for ncg,

$$j(u_k + \alpha_k d_k) \le j(u_k) + c_1 \alpha_k \langle g_k, d_k \rangle, \tag{2.18}$$
$$|\langle j'(u_k + \alpha_k d_k), d_k \rangle| \le c_2 |(g_k, d_k)|, \tag{2.19}$$

$0 < c_1 < c_2 < 1$, and just the Armijo condition (2.18) for sd [38].

Remark 2.2. In this work, we follow the *first optimize, then discretize* approach. To implement the optimization methods based on the optimality conditions of the previous section, we need to discretize the arising parabolic PDEs in time and space. To facilitate using PFASST for time parallelism, this is done using the method of lines approach, so we discretize space first. In the examples in section 5 we use a pseudospectral method, but other techniques like finite element, finite difference, or finite volume methods are possible as well. The resulting system of ODEs is then solved using PFASST.

3. SDC, MLSDC, and PFASST.

In this section, we give an overview of the parallel-in-time strategy used to solve the state and adjoint equations. The strategy is mainly based on the PFASST algorithm [13] with some modifications specific to PDE-constrained optimization problems. PFASST can be thought of as a time parallel variant of the multilevel spectral deferred correction (MLSDC) method [42], which in turn is constructed from the spectral deferred correction (SDC) method [12]. Since all three of these methods are well established, we give only a brief overview here skewed towards the numerical methods tested in section 5.

3.1. SDC methods.

Consider the generic ODE over the time interval $[t_j, t_{j+1}]$ representing one time step

$$y'(t) = F(t, y), \tag{3.1}$$

with initial condition $y(t_j) = y_j$. Divide the interval $[t_j, t_{j+1}]$ into M smaller subintervals by choosing points $t_m,\ m = 0, \dots, M$, corresponding to the Gauss–Lobatto quadrature nodes.

The exact solution of the ODE at each point t_m is given by

$$y(t_m) = y_j + \int_{t_j}^{t_m} F(\tau, y(\tau)) d\tau. \tag{3.2}$$

The collocation method is defined by the solution of the system of equations derived by applying quadrature rules to (3.2),

$$y_m \approx y_j + \Delta t \sum_{i=0}^{M} w_{m,i} F(t_i, y_i), \tag{3.3}$$

where $\Delta t = t_{j+1} - t_j$ and $w_{m,i}$ are the quadrature weights,

$$w_{m,i} = \frac{1}{\Delta t} \int_{t_j}^{t_m} \ell_i(s) ds, \quad m = 0, \dots, M, \quad i = 0, \dots, M,$$

with Lagrange polynomials ℓ_i defined by the quadrature nodes $(t_m)_{m=0,\dots,M}$ (here Lobatto IIIA). Equation (3.3) is a (typically nonlinear) system that is M times larger than that of a single-step implicit method like backward Euler. It is also equivalent to a fully implicit Runge–Kutta method with the values $w_{m,i}$ corresponding to the matrix in the Butcher tableaux. Such methods have good stability properties and have formal order of accuracy $2M$ for $M + 1$ Lobatto quadrature nodes. (See, e.g., [28] for a more detailed discussion of collocation and implicit Runge–Kutta methods.)

SDC methods can be considered as a fixed point iteration to solve the collocation formulation (3.3). Each SDC iteration or *sweep* consists of stepping through the nodes

t_m and updating the solution at that node $y_m^{[k]}$. A typical version of SDC using implicit substepping takes the form (with $[k]$ denoting iteration)

$$y_m^{[k+1]} = y_j + \Delta t \sum_{i=0}^{m} \tilde{w}_{m,i} F(t_i, y_i^{[k+1]}) + \Delta t \sum_{i=0}^{M} (w_{m,i} - \tilde{w}_{m,i}) F(t_i, y_i^{[k]}). \tag{3.4}$$

Here the values $\tilde{w}_{m,i} = 0$ for $i > m$, and hence each of the substeps has the same computational complexity of a single backward Euler step.

One attractive aspect of SDC methods is that they can easily be extended to cases in which (3.1) can be split into stiff and nonstiff components. Semi-implicit or implicit-explicit (IMEX) methods that split the equation into stiff and nonstiff terms first appeared in [37]. So-called multi-implicit or MISDC methods that treat two implicit terms in an operator splitting approach are introduced in [7]. Such splittings are explored in the numerical tests in section 5.2 by using MISDC variants to treat the nonlinearity of the state equation. In the numerical examples, the values of $\tilde{w}_{m,i}$ for implicit terms are chosen following the LU factorization method of Weiser [45], while those for explicit terms correspond to the usual forward Euler substepping.

3.2. MLSDC methods. Higher-order SDC methods require a relatively large number of function evaluations (explicit and/or implicit) per time step. One method to reduce the cost of these iterations is to employ a multilevel formulation of iterations where SDC sweeps are done on a hierarchy of discretization levels. In [42], so-called multilevel SDC (MLSDC) methods are studied where the levels are differentiated by the spatial and/or temporal order and resolution as well as the tolerance of implicit solvers. SDC sweeps are scheduled like V-cycles in multigrid, and coarse level problems are modified with a term analogous to a space-time full approximation scheme (FAS) correction term. In the current study, only the number of spatial degrees of freedom and the number of SDC quadrature nodes are varied on MLSDC or PFASST levels. MLSDC is the basic building block for the parallel-in-time method PFASST discussed next.

3.3. PFASST. The parallel full approximation scheme in space and time (PFASST) [13] is, as the name suggests, a method for exploiting both spatial and temporal parallelism in a manner similar to FAS multigrid methods for nonlinear problems (see [6] for a multigrid perspective of PFASST). As mentioned above, PFASST can be considered to be a pipelined version of the MLSDC method, with each time step being assigned to a separate processor (or groups of processors), such that MLSDC sweeps are performed on multiple time steps in parallel. "Pipelined" here refers to the idea that each processor begins a step of the algorithm as soon as new initial conditions are passed forward in time from the previous processor [36].

As all processors handling time steps $[t_j, t_{j+1}]$ need an initial condition y_j, which for $j > 0$ is not known, PFASST starts with a *predictor* phase. Usually, this is done by integrating the equations at the coarsest level in serial (with or without the FAS correction term) with a low-order method. Alternatively, this step can be conducted without coarse level communication since after distributing the initial value y_0 to all processors, the jth processor can compute j time steps sequentially rather than wait for the first $j-1$ time steps to be completed before beginning (*burn-in*). Note, however, that in the context of PDE-constrained optimization, the latter is not possible, as each processor only has data (control, desired state) for the time steps it actually computes, and not for the other time intervals. Instead, except for the first iteration of optimization method, one can initialize all processors using the

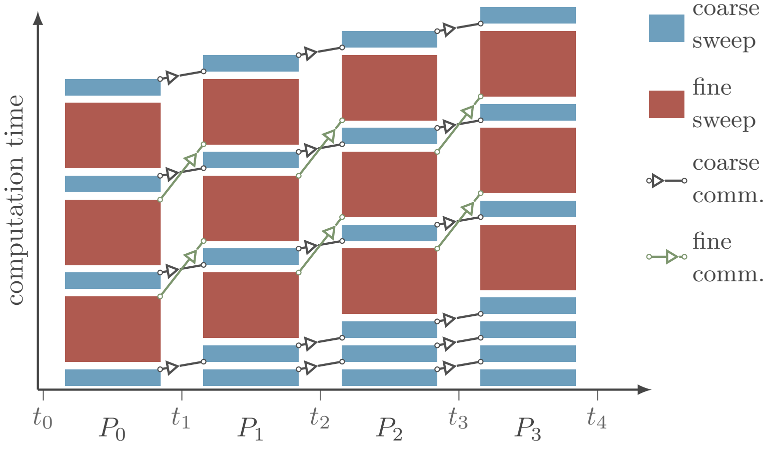

FIG. 3.1. *Generic two-level PFASST scheduling for the forward solution of an ODE* (3.1). *The time steps* $[t_j, t_{j+1}]$ *are distributed among processors* P_j*, which perform coarse SDC sweeps sequentially, and sweeps on the fine level in parallel. The coarse sweeps at the bottom illustrate the predictor, which for PDE-constrained optimization requires communication. Picture created using pfasst-tikz (https://github.com/f-koehler/pfasst-tikz).*

solution from the previous optimization iteration. We refer to this procedure as *warm starting* PFASST iterations. In this case, each processor will start doing SDC sweeps with communication at the coarse level, propagating the solution forward in time to receive an updated initial value. The predictor step is represented at the bottom of Figure 3.1, with warm starting discussed further in section 4.3.

Having finished the predictor phase, each processor performs MLSDC sweeps, where after each SDC sweep on each level of the MLSDC hierarchy, the solution is communicated forward in time to the next processor. This communication overlaps with computation, except at the coarsest level (see Figure 3.1 and [14]). A more detailed description can be found in [13].

4. Time-parallel PDE-constrained optimization.

In this section, we discuss three components to produce an efficient fully time-parallel method for PDE-constrained optimization. First, we briefly describe parallelizing the outer optimization loop, before coming to the more involved parallel-in-time computation of the reduced gradient. Finally, as the time integrators for solving state and adjoint equations are iterative, we discuss using previous solutions to *warm start* the time integration.

For parallelization in time, the time domain $[0, T]$ is subdivided into N time steps $0 = t_0 < \cdots < t_N = T$, which are distributed on R processors. For examples with relatively fewer time steps, the shortest computation times may result from choosing $R = N$. On the other hand, when the number of processors used is smaller than the total number of time steps (e.g., in the strong scaling studies included below), the PFASST algorithm is applied sequentially to blocks of time steps where N is an integer multiple of R. Note that here we consider only temporal parallelism. Usually, this is used in combination with spatial parallelism and *multiplies* the speedup gained from spatial parallelism. This means that the R processors used for parallelizing in time can be considered as groups of processors with additional spatial parallelism. The best strategy for distributing processors between time and space parallelism is in general problem- and machine-dependent and not considered here.

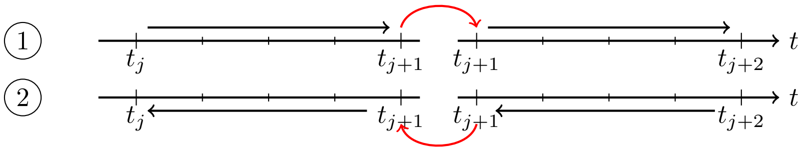

Fig. 4.1. *Two time steps of the first state, then adjoint variant (section* 4.2.1*). In step* 1*, the state equation is fully solved, including forward communication; afterwards, the adjoint equation is solved, including backward communication.*

4.1. Parallelization of the optimization loop. Time parallelization of the optimization loop (2.16), (2.17) requires parallel-in-time computation of the reduced gradient, as well as evaluation of inner products. In the present optimal control setting for the prototype problem (2.10)–(2.14) with the usual L^2-regularity in time, $(\cdot,\cdot)$ denotes the inner product in $L^2(0,T;H)$, so

$$(v,w) = \int_0^T (v(t),w(t))_H \, dt. \tag{4.1}$$

The evaluation of these inner products, and thus the computation of β_k as well as the evaluation of the objective function $j(u_k), j(u_k+\alpha_k d_k)$, can easily be done time-parallel, as

$$(v,w) = \sum_{i=0}^{N} \int_{t_i}^{t_{i+1}} (v(t),w(t))_H. \tag{4.2}$$

After discretization, each processor evaluates the discrete spatial inner product and integrates in time only for its own time steps. In the numerical examples below, evaluation of the time integral is done using a simple trapezoidal rule; of course using other quadrature rules, like reusing the spectral quadrature matrices provided by PFASST, is possible as well. In terms of communication, for each inner product only one scalar value per processor has to be transmitted to one master processor collecting the results. The essential ingredient for a fully parallel optimization method is the computation of the reduced gradient $\nabla j(u_k)$, which is discussed next.

4.2. Time-parallel adjoint gradient computation. Here we discuss three different ways in which PFASST can be used to enable a time-parallel computation of the reduced gradient $\nabla j(u_k)$. In the simplest case, PFASST is used to first solve the state equation and then to solve the adjoint equation afterwards (section 4.2.1). Alternatively, if $R = N$, the adjoint can be solved simultaneously with the state equation (section 4.2.2). A third variant is inspired by the paraexp method [18]. In this variant, we make use of the linearity of the adjoint equation to split the adjoint solve into an inhomogeneous equation with homogeneous terminal conditions on each time step (without communication) and a subsequent propagation of the correct terminal conditions backward in time (section 4.2.3).

4.2.1. First state, then adjoint. In this straightforward approach, PFASST is used first to solve the state equation (2.11), (2.12). The state solution is recorded at the quadrature nodes and then used in a subsequent PFASST run to solve the adjoint equation (2.13), (2.14); see Figure 4.1 for a sketch. This has already been used in the preliminary study [21].

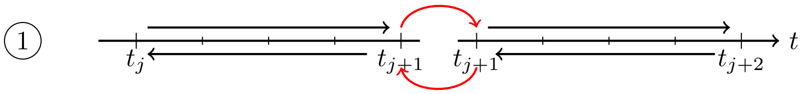

FIG. 4.2. *Simultaneous solve of state and adjoint (section* 4.2.2*), requiring forward and backward communication between two adjacent time steps.*

Compared to sequential time stepping, the storage overhead is larger here, as the discretized-in-space state solution has to be stored on quadrature nodes, not only on the time steps. This is somewhat mitigated by the fact that the time intervals are distributed across several processors, typically giving access to more memory. Using PFASST gives some flexibility in reducing memory requirements. Besides compressed storage (see, e.g., [22]) for adaptive lossy compression of finite element solutions, options include storing the state on the coarse level only and reducing the compression error by additional state sweeps without communication for the adjoint solve. A detailed analysis of storage requirements and storage reduction techniques will be reported elsewhere.

4.2.2. Simultaneous approach. Alternatively, if $R = N$, the adjoint can be solved simultaneously with the state equation (Figure 4.2), requiring communication of updated initial values for the state equation forward in time as well as updated terminal conditions for the adjoint backward in time. This cross-communication makes an efficient implementation with overlapping communication and computation difficult (see Figure 4.3). As our numerical experiments show that this induces severe wait times, thus rendering the method inefficient, we do not consider this approach further.

4.2.3. Mixed approach. Making use of the linearity of the adjoint equation, the adjoint p described in (2.13), (2.15) can split into two terms $p = \tilde{p} + \delta$, where $\tilde{p}$ satisfies the same equation as p but with homogeneous terminal conditions, and the defect δ which corrects the terminal conditions. Specifically, for a given state solution $\bar{y}$, let

$$-\tilde{p}_t - \kappa\Delta\tilde{p} + f_y(\bar{y})\tilde{p} = -J^\Omega(\bar{y}, u) \quad \text{in } \Omega \times [0, T], \tag{4.3}$$

$$\tilde{p}(T) = 0 \quad \text{in } \Omega \tag{4.4}$$

and

$$-\delta_t - \kappa\Delta\delta + f_y(\bar{y})\delta = 0 \quad \text{in } \Omega \times [0, T], \tag{4.5}$$

$$\delta(T) = -J^T(\bar{y}, u) \quad \text{in } \Omega. \tag{4.6}$$

To allow for an efficient time-parallel solution, we have to further modify the equations on the time intervals. Denote by the superscript j, e.g., $\tilde{p}^j, \delta^j$, the respective solution on the jth time interval $[t_j, t_{j+1}]$. With this notation, let $\tilde{p}^j$ solve

$$-\tilde{p}^j_t - \kappa\Delta\tilde{p}^j + f_y(\bar{y})\tilde{p}^j = -J^\Omega(\bar{y}, u) \quad \text{in } \Omega \times [t_j, t_{j+1}], \tag{4.7}$$

$$\tilde{p}^j(t_{j+1}) = 0 \quad \text{in } \Omega. \tag{4.8}$$

Note the homogeneous boundary condition *on the interval*. For the defect δ^j, the respective equation then is

$$-\delta^j_t - \kappa\Delta\delta^j + f_y(\bar{y})\delta^j = 0 \quad \text{in } \Omega \times [t_j, t_{j+1}] \tag{4.9}$$

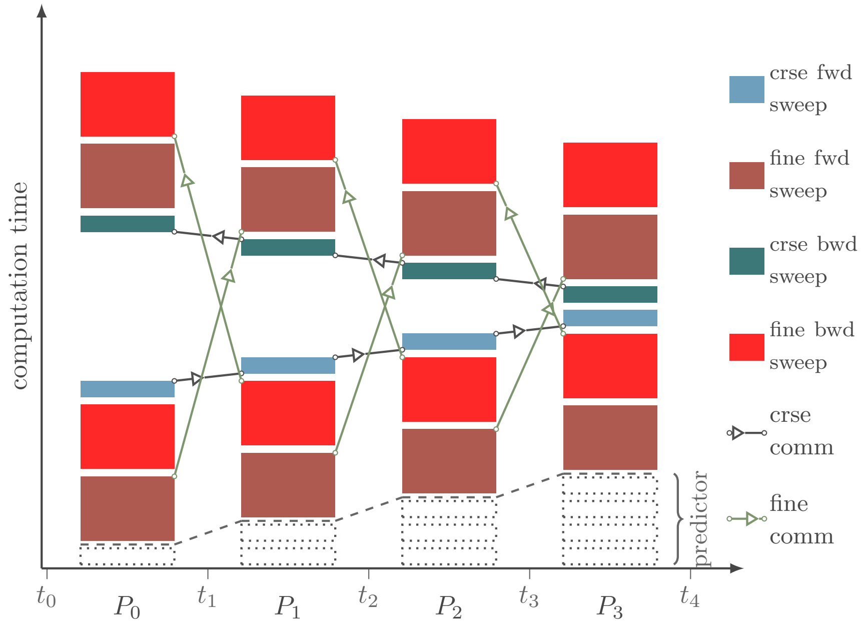

FIG. 4.3. *Sweeps and communication pattern for simultaneously solving state and adjoint equations. The forward-backward communication makes an efficient implementation without severe wait times difficult. Picture created using a modified version of pfasst-tikz (https://github.com/f-koehler/pfasst-tikz).*

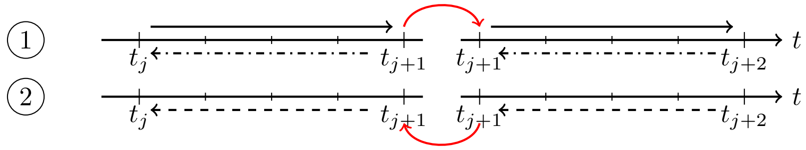

FIG. 4.4. *Mixed approach (section 4.2.3). In step 1, the full state equation is solved forward in time, including forward communication. Simultaneously, a modified adjoint equation is solved without backward communication. In step 2, a correction for the adjoint is computed, propagating the correct terminal values.*

with terminal conditions

$$\delta^j(t_{j+1}) = \begin{cases} \tilde{p}^{j+1}(t_{j+1}) + \delta^{j+1}(t_{j+1}), & j = 0, \dots, N-1, \\ -J^T(\bar{y}, u), & j = N. \end{cases} \tag{4.10}$$

With this, (4.7), (4.8) can be solved in parallel together with the state equation on the interval without requiring communication backward in time (in the first step in Figure 4.4, the dash-dotted line denotes the modified adjoint equation). In the second step, the defect equation is solved (dashed line in Figure 4.4). Note that on each time interval, the adjoint solvers only need to access the state solution computed on the same interval (even if each processor computes multiple time steps), thus requiring no additional communication of state values.

Remark 4.1. In case of no terminal cost in the objective function, the splitting of the adjoint is still required, since the homogeneous terminal condition only holds at time $t = T$. After time discretization, the terminal condition *of the time step*, i.e., $\tilde{p}^j(t_{j+1})$, is in general nonzero. To avoid backward-in-time communication, these terminal conditions on the respective intervals are treated by the defect equation (4.9), (4.10).

For the solution of (4.9), (4.10), we note that formally, defining the operator $L = -\kappa\Delta + f_y(\bar{y})$, the solution is given by

$$\delta^j(t) = \exp\big((t - t_{j+1})L\big)\delta^{j+1}(t_{j+1}), \quad t \in [t_j, t_{j+1}]. \tag{4.11}$$

For linear state equations, we have $f_y = 0$, and (4.11) can be solved efficiently using suitable approximations of the matrix exponential; see the discussion in [18]. Thus, for the numerical experiments we restrict ourselves to the linear case. Besides having a certain relevance on their own, such equations occur, e.g., as subproblems in an SQP method, or in the evaluation of Hessian-times-vector products for Newton-CG methods, where an additional linearized state equation and a corresponding second adjoint equation need to be solved. For nonlinear state equations, the differential operator and/or coefficients of the adjoint are time-dependent (as they depend on the state solution). The extension of the mixed approach to this case (e.g., facilitating the Magnus expansion along the lines of [33], where temporal parallelism of Magnus integrators is explored) is left for future work.

Remark 4.2. The mixed approach is to some extent related to the paraexp algorithm [18] for linear initial value problems. While the treatment of the source term on time intervals $[t_j, t_{j+1}]$ is the same, the paraexp method uses a near-optimal exponential integrator to compute in parallel solutions with the correct initial value on intervals $[t_j, T]$, $j = 0, \dots, N-1$, i.e., up to the final time T, and then sums up the individual solutions, requiring communication between several processors for computing the superposition at the required time points. In contrast, we solve the defect equation to propagate the correct terminal values on the time interval $[t_j, t_{j+1}]$, $j = 0, \dots, N-1$, only, such that only solution values at t_j, $j = N, \dots, 1$, have to be communicated to the adjacent processor.

4.3. Warm starts. An important benefit of using SDC sweeps is the option to reuse information from the previous optimization iteration to initialize the subsequent PDE solves. For this, the solution of state and/or adjoint at the quadrature nodes is stored in one optimization iteration and used as an initial guess instead of the predictor step during the following state/adjoint solve with an updated control. As the optimization is converging, $\|u_{k+1} - u_k\|$ gets smaller, so that the previous solution of the state and adjoint equations is a suitable initialization, and the PFASST algorithm will take fewer iterations to converge. This incurs an overhead in storage that can be reduced by either using compression or, as it is only used as initialization, having values stored only on the coarsest level and interpolated.

For the numerical examples below, to sweep on state or adjoint in optimization iteration $k+1$ we load the stored fine-level solution of the previous iteration k to initialize at the fine quadrature nodes. Instead of invoking the usual PFASST predictor, we restrict the solution values and function evaluations to the coarser levels and perform sweeps on the coarse level with communication as described in section 3.3.

5. Numerical results. In this section, we test the developed methods on two optimal control problems. First, we consider optimal control for a linear heat equation in section 5.1 to compare the first state, then adjoint approach and the mixed approach for cold and warm starts. As a second example, an optimal control problem governed by the nonlinear Nagumo reaction-diffusion equation is considered (section 5.2). There we compare different methods of treating the nonlinearity and again consider cold and warm starting of the time integration, using the first state, then adjoint approach. All examples are implemented using the LibPFASST library [2] and are timed on Intel Xeon E5-4640 CPUs clocked at 2.4GHz.

Before discussing the results in detail, two remarks are in order. First, we report speedup and parallel efficiency with respect to the sequential versions of the described methods, i.e., MLSDC for state and adjoint solutions. In particular, the sequential integration method has the same temporal order as the parallel version. We do not perform a thorough comparison of our methods with other commonly used sequential temporal integrators, as it is not clear what the "best" sequential method is for the respective problems. However, in section 5.2, we briefly report results using IMEX-Euler as well as a fourth-order additive Runge–Kutta method as sequential references. Second, scaling experiments are performed for a relatively low number of processors (20 and 32 in the two examples). Usually, parallel-in-time methods are combined with parallelization in the spatial domain and multiply the speedup (and the required number of processors). As a rule of thumb, time-parallel methods come into play when spatial parallelism saturates; i.e., most of the available processors will be used for spatial parallelism.

5.1. Linear problem: Heat equation. For the first numerical test, we consider the linear-quadratic optimal control problem to minimize $J(y,u)$ given by (2.10) subject to

$$\begin{aligned} y_t - \Delta y &= u \quad \text{in } \Omega \times [0,T], \\ y(0) &= y_0 \quad \text{in } \Omega, \end{aligned}$$

with periodic boundary conditions. The spatial domain $\Omega = [0,1]^3$, with initial conditions

$$y_0(x) = \frac{1}{12\pi^2\lambda}(1-T)\prod_{i=1}^{3}\sin(2\pi x_i), \quad x \in \Omega,$$

and target solution

$$y_d(x,t) = \left[\left(12\pi^2 + \frac{1}{12\pi^2\lambda}\right)(t-T) - \left(1 + \frac{1}{(12\pi^2)^2\lambda}\right)\right]\prod_{i=1}^{3}\sin(2\pi x_i), \qquad x \in \Omega, \quad t \in [0,T].$$

This implies the exact solution

$$\begin{aligned} p^\star &= (T-t)\prod_{i=1}^{3}\sin(2\pi x_i), \\ u^\star &= -\frac{1}{\lambda}p^\star, \\ y^\star &= \left(-\frac{1}{(12\pi^2)^2\alpha}\,t - \frac{1}{12\pi^2\alpha}\,T + \frac{1}{12\pi^2\lambda}\right)\prod_{i=1}^{3}\sin(2\pi x_i). \end{aligned}$$

For strong scaling results, we solve the above problem, with control cost parameter $\lambda = 0.05$ in the objective (2.10). We use a three-level PFASST scheme and a pseudospectral discretization in space with $(16/32/64)^3$ degrees of freedom on the

Table 5.1

Strong scaling results for the heat example: first state, then adjoint approach with cold start. Speedup and parallel efficiency are compared to the sequential run with one processor.

#CPUs	Time [s]	Speedup	Efficiency [%]
1	7,808.9	–	0.0
2	6,063.5	1.3	65.0
5	3,797.3	2.1	42.0
10	2,677.8	2.9	29.0
20	1,679.9	4.6	23.0

respective levels. The implicit linear solves are done via the FFT. In time, we use $T = 2$ and 20 time steps with 2/3/5 Lobatto IIIA quadrature rules, yielding a temporal order 8. To solve the optimization problem, we apply gradient descent with the Armijo step size rule and stop the optimization after 50 iterations to compare the optimization progress, computed controls, and timings of the different algorithmic variants (first state, then adjoint vs. mixed, cold vs. warm start). PFASST iterations are stopped when the absolute or relative residual drops below 10^{-10}. We note that this residual tolerance is too strict for the initial optimization iterations, where gradients with lower accuracy would be sufficient. Future research will be concerned with a thorough analysis of accuracy requirements and the influence of inexactness on the convergence of the optimization methods along the lines of [22]. A brief comparison of different residual tolerances is shown in Table 5.5. We note that while naturally the wallclock time decreases, the influence of increased residual tolerances on parallel efficiency is small, as both sequential and parallel versions are accelerated similarly.

Results are obtained using the first state, then adjoint approach as well as the mixed approach. For each variant, we compare the usual PFASST predictor (cold start) to initialize the MLSDC sweepers on each time step and warm starts, using the previously computed solution on the fine level to initialize the sweeper. Tables 5.1–5.4 show speedup and efficiency for strong scaling using up to 20 processors for the different approaches. In Tables 5.1 and 5.2, the usual PFASST predictor is used, spreading the given initial value on each parallel time step to the quadrature nodes. While speedup in Table 5.1 shows the gain in computation time due to using PFASST instead of MLSDC, the speedup in Table 5.2 is computed with respect to the sequential first state, then adjoint approach, i.e., contains speedup due to parallelism as well as splitting the adjoint solve. Tables 5.3 and 5.4 show results for warm starting the respective methods. Total speedup is computed with respect to the sequential first state, then adjoint–based method. Speedup to cold denotes the speedup compared to the same method and same number of processors, but using the standard PFASST predictor (burn-in instead of warm start; see section 3.3). In this example, warm starts increase the speedup by 10–20%. For the first state, then adjoint variant, the 10% gain due to warm start in the 20 processor setting corresponds to a reduction in MLSDC sweeps of 14% (state) and 34% (adjoint). Thus, for problems where the sweeps are more expensive, we expect warm starting to be more effective (compare also section 5.2).

Figure 5.1 shows the progress of the optimization, which is similar for all approaches. The individual wallclock times are compared again in Figure 5.2. For cold starting, the mixed approach performs better than the first state, then adjoint approach for most settings, but the advantage is rather small. Here, the mixed approach mainly saves computation time due to reduced communication, as both the propaga-

Table 5.2

Strong scaling results for the heat example: mixed approach with cold start. Speedup computed with respect to the first state, then adjoint approach with one CPU.

#CPUs	Time [s]	Speedup	Efficiency [%]
1	8,888.8	–	0.0
2	6,116.0	1.3	65.0
5	3,518.1	2.2	44.0
10	2,508.1	3.1	31.0
20	1,606.0	4.9	24.5

Table 5.3

Strong scaling results for the heat example: first state, then adjoint with warm start. Total speedup computed with respect to first state, then adjoint approach with one processor and cold start. Speedup to cold denotes the speedup only due to warm start, i.e., compared to the time for cold start with the same number of CPUs. Efficiency is computed using the total speedup.

#CPUs	Time [s]	Total speedup	Speedup to cold	Efficiency [%]
1	6,474.1	1.2	1.2	–
2	5,087.1	1.5	1.2	75.0
5	3,172.1	2.5	1.2	50.0
10	2,034.9	3.8	1.3	38.0
20	1,347.5	5.8	1.2	29.0

Table 5.4

Strong scaling results for the heat example: mixed approach with warm start. Total speedup computed with respect to the first state, then adjoint approach with one CPU and cold start. Speedup to cold denotes the speedup only due to warm start, i.e., compared to the time for the mixed approach using cold start with the same number of CPUs. Efficiency is computed using the total speedup.

#CPUs	Time [s]	Total speedup	Speedup to cold	Efficiency [%]
1	7,187.6	1.1	1.2	–
2	5,537.4	1.4	1.1	70.0
5	3,424.8	2.3	1	46.0
10	2,276.3	3.4	1.1	34.0
20	1,494.6	5.2	1.1	26.0

Table 5.5

Influence of residual tolerance on parallel speedup for the first state, then adjoint approach with cold start. Note that the sequential reference is computed using the same tolerances; otherwise, total speedups are computed as in Tables 5.1–5.4.

	Speedup (20 CPUs)			
Residual tolerance	First state, then adjoint		Mixed	
	cold	warm	cold	warm
10^{-10}	4.9	5.8	4.9	5.2
10^{-8}	4.9	5.9	4.6	5.3
10^{-6}	4.5	5.5	4.3	4.9
10^{-4}	4.1	5.0	3.9	4.4

tion of the terminal condition in the mixed approach and the SDC sweeps in the plain first state, then adjoint approach are computed using the FFT and have a similar computational burden. Thus, for other space discretizations, e.g., finite differences or

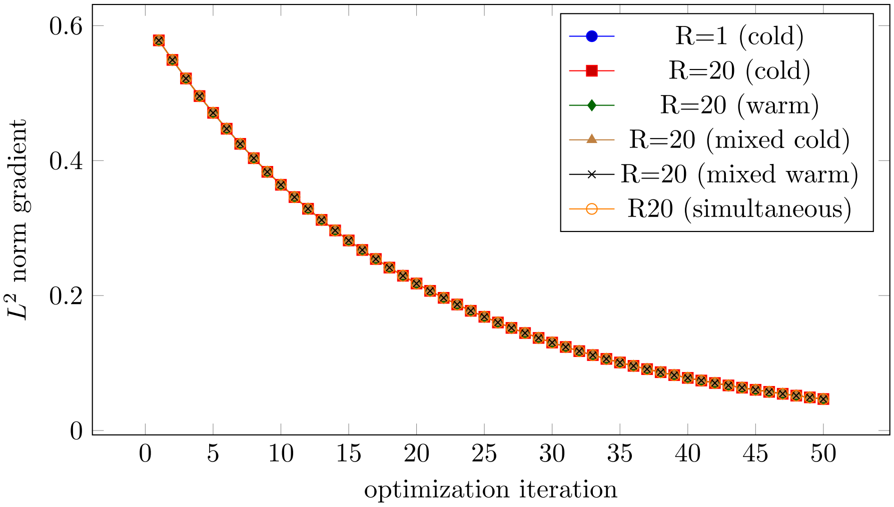

FIG. 5.1. *Heat example: Optimization progress is the same for the sequential first state, then adjoint approach (plain), its parallel variant (cold and warm starts), and the parallel mixed approach (cold and warm start), as well as the simultaneous approach.*

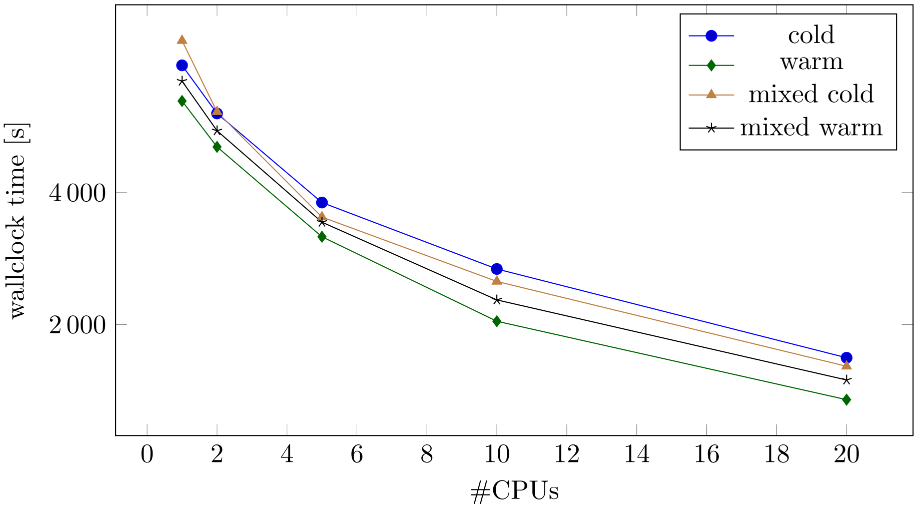

FIG. 5.2. *Strong scaling for the heat example: comparison of wallclock times of mixed and first state, then adjoint approaches with cold and warm starts.*

finite elements, we expect the mixed approach to offer larger gains in computation times. Warm starting offers a moderate gain in speedup and efficiency for both variants (first state, then adjoint, and mixed). As the gain is slightly higher in the first state, then adjoint approach, this variant provides the best performance in this test, achieving a parallel efficiency of 29% on 20 processors.

For completeness, the third variant, solving state and adjoint simultaneously (section 4.2.2), leads to the same optimization progress (shown in Figure 5.1), but gives a speedup of merely 1.8 on 20 processors, which is significantly worse than other variants.

5.2. Nonlinear problem: Nagumo equation. Next we consider the following optimal control problem (see also [3, 8, 30]):

$$\min_{y,u} \frac{1}{2}\int_0^T \int_\Omega (y-y_d)^2 \; dx \; dt + \frac{\lambda}{2}\int_0^T \int_\Omega u^2 \; dx \; dt;$$

i.e., as in the linear case, minimize (2.10), subject to

$$\begin{aligned} \frac{\partial}{\partial t}y(x,t) - \frac{\partial^2}{\partial x^2}y(x,t) + \left(\frac{\gamma}{3}y^3(x,t) - y(x,t)\right) &= u(x,t) && \text{in } \Omega \times (0,T), \\ \frac{\partial}{\partial x}y(0,t) = \frac{\partial}{\partial x}y(L,t) &= 0 && \text{in } (0,T), \\ y(x,0) &= y_0(x) && \text{in } \Omega. \end{aligned} \tag{5.1}$$

The parameter γ steers the influence of the nonlinear reaction terms, and thus the stiffness due to the reaction term, and will be used below to compare IMEX and MISDC (multi-implicit SDC) formulations for solving the state equation. The spatial domain $\Omega = (0,L)$, with $L = 20$, and the equation is solved up to final time $T = 5$. The initial condition is

$$y_0(x) = \begin{cases} 1.2\sqrt{3}, & x \in [9,11], \\ 0 & \text{elsewhere} \end{cases}$$

and the target solution

$$y_d(x,t) = \begin{cases} y_{\text{nat}}(x,t), & t \in [0,2.5], \\ y_{\text{nat}}(x,2.5), & t \in (2.5,T], \end{cases}$$

where y_{nat} denotes the solution to the PDE (5.1) for $u \equiv 0$. Here, for $\lambda = 0$, an exact optimal control is known,

$$u_{\text{exact}} = \begin{cases} 0, & t \le 2.5, \\ (\frac{\gamma}{3}y^3_{\text{nat}}(x,2.5) - y_{\text{nat}}(x,2.5)) - \frac{\partial^2}{\partial x^2}y_{\text{nat}}(x,2.5), & t > 2.5, \end{cases}$$

which will be used as a comparison for the computed optimal controls. For this example, the adjoint equation is

$$\begin{aligned} -\frac{\partial}{\partial t}p - \frac{\partial^2}{\partial x^2}p + (\gamma y^2 - 1)p &= -(y-y_d) && \text{in } \Omega \times (0,T), \\ \frac{\partial}{\partial x}p(0,\cdot) = \frac{\partial}{\partial x}p(L,\cdot) &= 0 && \text{in } (0,T), \\ p(\cdot,T) &= 0 && \text{in } \Omega, \end{aligned} \tag{5.2}$$

and the reduced gradient is given as $\nabla j(u) = \lambda u - p$. For solving state and adjoint equations, we do not consider the mixed method here, but always use the first state, then adjoint approach, since the adjoint equation contains time-dependent terms in the differential operator (see the discussion in section 4.2.3).

State equation and adjoint solve: MISDC vs. IMEX. Before coming to the actual optimization, let us briefly compare MISDC and IMEX-MLSDC for solving the state and adjoint equations (5.1), (5.2) with zero control and varying γ. This not only demonstrates the flexibility of PFASST, but it allows us to choose a suitable method in the scaling study for the optimization problem.

The IMEX and MISDC approaches are explained by examining a single substep of an SDC sweep. With k denoting the SDC iteration, m the substep index, and D^2 the discretization of the second derivative term, the fully implicit SDC version of an SDC substep (3.4) for (5.1) takes the form

$$y_m^{[k+1]} = y_j + \tilde{w}_{m,m}\Delta t \left(D^2 y_m^{[k+1]} - y_m^{[k+1]} \left(\frac{\gamma}{3}(y_m^{[k+1]})^2 - 1 \right) \right) + S_m^{[k]}, \tag{5.3}$$

where the term $S_m^{[k]}$ contains terms that either depend on the previous iteration $[k]$ or values at iteration $[k+1]$ already computed at substep $i < m$, including the control terms arising from the discretization of $u(x,t)$. The implicit equation couples nonlinear reaction and diffusion terms and hence requires a global nonlinear solver in each substep. For problems in which the reaction terms are nonstiff and can be treated explicitly, the reaction terms at node m do not appear in the implicit equation, giving the form

$$y_m^{[k+1]} = y_j + \tilde{w}_{m,m}\Delta t (D^2 y_m^{[k+1]}) + S_m^{[k]}. \tag{5.4}$$

Each IMEX substep now requires only the solution of a linear implicit equation, and thus is computationally cheaper than the fully implicit approach, provided that the explicit treatment of the reaction term does not impose an additional time step restriction. When the reaction term is stiff, and hence it is advantageous to treat it implicitly, a standard MISDC approach applies an operator splitting between diffusion and reaction in the correction equation. For example,

$$y^* = y_j + \tilde{w}_{m,m}\Delta t D^2 y^* + S_m^{*,[k]}, \tag{5.5}$$

$$y_m^{[k+1]} = y_j + \tilde{w}_{m,m}\Delta t \left(D^2 y^* - y_m^{[k+1]} \left(\frac{\gamma}{3}(y_m^{[k+1]})^2 - 1 \right) \right) + S_m^{[k]}. \tag{5.6}$$

We use Newton's method with damping and the natural monotonicity test according to [11] to solve the nonlinear equation (5.6) in each sweep. To reduce the computational effort, the MISDC approach is further modified so that the nonlinear solve for reaction in (5.6) is made linear by lagging terms in the splitting:

$$y_m^{[k+1]} = y_j + \tilde{w}_{m,m}\Delta t \left(D^2 y^* - y_m^{[k+1]} \left(\frac{\gamma}{3}(y^*)^2 - 1 \right) \right) + S_m^{[k]}. \tag{5.7}$$

This form creates an implicit solve with roughly the same cost as treating reaction explicitly but is more stable.

In all cases, a three-level PFASST scheme is applied to a method of lines discretization. We use a pseudospectral discretization with 64/128/256 degrees of freedom in space, with spatial derivatives computed spectrally with the FFT. In time, 32 uniform time steps are used, with 3/5/9 Lobatto IIIA collocation rules. Thus, all variants have the same order, and converge to the same collocation solution, if they converge. Sweeps are stopped when the relative or absolute residual drops below 10^{-11}. Table 5.6 shows the required number of sweeps for convergence for several values of γ, time steps, and number of processors. While both MISDC variants stably converge for all values of γ and time steps in sequential and parallel, IMEX requires finer time discretizations to converge, especially in parallel. In the sequential runs, the stability of the IMEX scheme requires a time step small enough so that the IMEX SDC iterations converge despite the explicit treatment of the reaction term. On the other hand, in PFASST, the sequential coarse grid IMEX sweeps are low-order and have a more restrictive stability constraint than the IMEX serial schemes. When the solvers

Table 5.6

IMEX vs. MISDC for the Nagumo example: required number of PFASST iterations (average per time step, rounded). For the nonlinear solver, Newton's method is stopped when the discrete ℓ_2-norm of the correction is smaller than 10^{-12}.

γ	$T/\Delta t$	#CPUs	IMEX		MISDC lagged		MISDC nonlinear	
			state	adjoint	state	adjoint	state	adjoint
1	32	1	-	-	14	9	14	9
		32	-	-	53	23	55	23
	64	1	7	6	7	6	7	6
		32	-	-	31	19	32	19
	128	1	5	4	5	4	5	4
		32	20	18	20	18	20	18
3	32	1	-	-	14	9	14	9
		32	-	-	53	24	54	24
	64	1	-	-	7	6	7	6
		32	-	-	31	19	32	19
	128	1	5	4	5	4	5	4
		32	20	18	21	18	21	18
5	32	1	-	-	14	9	14	9
		32	-	-	53	24	56	24
	64	1	-	-	7	6	7	6
		32	-	-	31	19	32	19
	128	1	5	4	5	4	5	4
		32	-	-	21	18	21	18
	256	1	4	4	4	4	4	4
		32	18	16	18	18	18	18

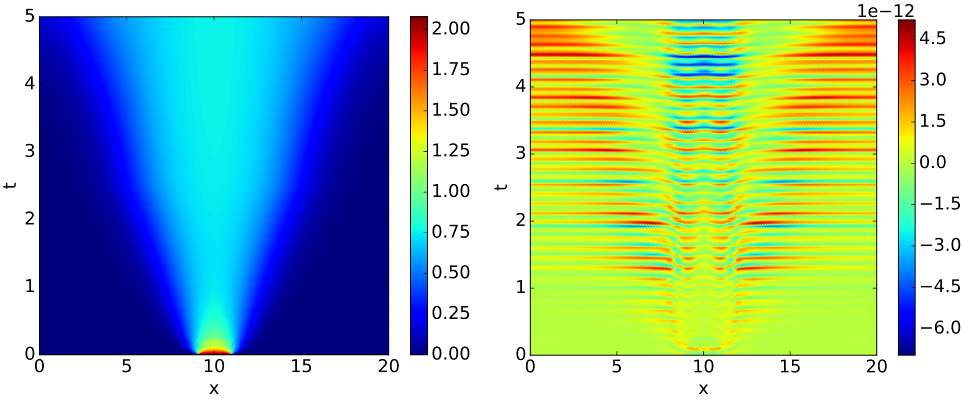

Fig. 5.3. *Nagumo example: state solution for $\gamma = 5$, 32 parallel time steps using MISDC with lagging (left) and difference to the state solution obtained with nonlinear MISDC (right).*

converge, all variants require approximately the same number of sweeps on average, and hence the IMEX scheme would be expected to need the least computational time. Figure 5.3 shows the solution for $\gamma = 5$ and 32 parallel time steps, computed with MISDC with lagging, and the difference to the solution obtained using MISDC with a nonlinear solver. While in most settings nonlinear and lagged versions need the same number of iterations, here the nonlinear variant requires on average three iterations more than the lagged version, but the solutions agree to the convergence tolerance.

The conclusions of this comparison are two-fold. First, the benefits of using an

Table 5.7
Nagumo example: Strong scaling results for IMEX (first state, then adjoint; cold start). Speedup and parallel efficiency are compared to the sequential IMEX version with one CPU.

#CPUs	Time [s]	Speedup	Efficiency [%]	Rel. err. to u_{exact}
1	169.8	–	0.0	$1.1 \cdot 10^{-1}$
2	109.0	1.6	80.0	$1.3 \cdot 10^{-1}$
4	73.4	2.3	57.5	$1.3 \cdot 10^{-1}$
8	51.0	3.3	41.2	$1.1 \cdot 10^{-1}$
16	37.7	4.5	28.1	$1.1 \cdot 10^{-1}$
32	32.0	5.3	16.6	$1.2 \cdot 10^{-1}$

Table 5.8
Nagumo example: Strong scaling results for IMEX (first state, then adjoint; warm start). Total speedup is compared to IMEX with one CPU and cold start, speedup to cold denotes the speedup only due to warm start, i.e., compared to the time for cold start with the same number of CPUs. Efficiency is computed using the total speedup.

#CPUs	Time [s]	Total speedup	Speedup to cold	Efficiency [%]	Rel. err. to u_{exact}
1	132.7	1.3	1.3	130.0	$1.3 \cdot 10^{-1}$
2	84.6	2	1.3	100.0	$1.3 \cdot 10^{-1}$
4	58.0	2.9	1.3	72.5	$1.2 \cdot 10^{-1}$
8	39.6	4.3	1.3	53.7	$1.2 \cdot 10^{-1}$
16	30.4	5.6	1.2	35.0	$1.2 \cdot 10^{-1}$
32	22.1	7.7	1.4	24.1	$1.2 \cdot 10^{-1}$

MISDC scheme versus an IMEX scheme will depend on the relative stiffness of the two terms and the relative cost of the nonlinear versus linear implicit equations. Second, at least for the problems considered here, the MISDC with lagging method performs as well as or better than an IMEX splitting in terms of iteration count with only a small computational overhead relative to the IMEX variant.

Optimization. To evaluate the performance of the parallel-in-time optimization method, we use again three-level PFASST with an IMEX sweeper for solving state and adjoint equations up to a residual tolerance of 10^{-11}. We set $\gamma = 1$ in the state equation and use 32/64/128 degrees of freedom in space as well as 32 time steps. In contrast to the experiments above, IMEX is converging in this setting due to the coarser space discretization and has the smallest computation time. In the objective function, the regularization parameter $\lambda = 10^{-6}$ is used. Note that this leads to a difference between the computed optimal control and u_{exact}, as the latter is valid only for $\lambda = 0$. The optimization is done using the ncg method of Dai and Yuan [9] in combination with linesearch to satisfy the strong Wolfe conditions. As reported in [8], the optimization progress with several ncg variants is slow, so we stop the optimization after 200 iterations. We again consider strong scaling with up to 32 CPUs. Due to the less severe nonlinearity in this test case and in view of the comparison above, only results using the IMEX variant are shown here.

Tables 5.7 and 5.8 show timings, speedup, and parallel efficiency for IMEX with cold and warm starts. For cold starts, a speedup of more than 5 for 32 processors is achieved, corresponding to a parallel efficiency of 16.5%.

For the case with 32 CPUs, warm starting reduces the number of required state sweeps by 65%, while the reduction in adjoint sweeps is less pronounced. Overall, this translates to a decrease in computation time by nearly 40%, leading to a parallel

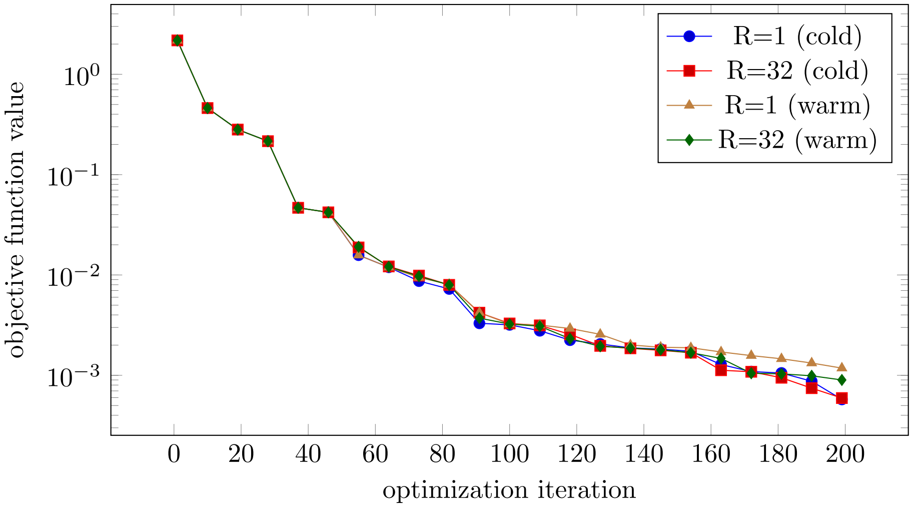

Fig. 5.4. *Nagumo example: optimization progress for IMEX, using the first state, then adjoint approach, sequential and in parallel (32 CPUs), both with cold and warm starts.*

efficiency of around 24%. The gain here is significantly larger than in the linear heat example (section 5.1). This is true despite the tests being in one spatial dimension: the PFASST method typically produces better parallel efficiencies for problems in higher dimensions, since spatial coarsening then produces a greater relative reduction in computational cost on coarse levels.

Similar results (not reported here) are achieved using MISDC with lagging, albeit with overall higher computation times.

Figure 5.4 shows the objective function value over the optimization process with 1 and 32 processors as well as using cold and warm starts. We note that there are slight variances in the optimization progress and the computed controls, as also seen in the final error of the control in Table 5.7. The difference is that the sequential version does SDC sweeps on the later time steps with the correct initial value, while PFASST iteratively corrects the incorrect initial values. For warm starts, in addition the initial function evaluations as well as coarse grid corrections are inexact. Thus, the iterative solution of the PDEs progresses differently, leading to slightly different final solutions and thus different gradients and step sizes, which accumulate during the optimization.

Comparison with classical time steppers. The results reported above use MLSDC on one processor as the sequential reference for the computation of speedups. This allows the use of the same time discretization for all setups in the strong scaling study. However, due to the relatively high computational cost, MLSDC may not be the best sequential method. For comparison, we consider a standard IMEX-Euler method as well as a fourth-order additive Runge–Kutta (ARK-4) method of Kennedy and Carpenter [32]. The IMEX-Euler method treats the reaction part explicitly and the diffusion part implicitly; ARK-4 combines an explicit Runge–Kutta method for the reaction part with an ESDIRK method for the diffusion. To obtain a reference solution for state and adjoint equations, we fix the spatial discretization to 128 degrees of freedom as before and use SDC with nine Lobatto IIIA collocation nodes on a very fine temporal mesh. Time step sizes Δt for the different methods are chosen such

Table 5.9
Runtimes of different sequential methods yielding similar optimization results.

Method	Δt	Time [s]	Rel. err. to u_{exact}
IMEX-Euler	10^{-3}	102.5	$1.2 \cdot 10^{-1}$
ARK-4	$2.5 \cdot 10^{-3}$	183.0	$1.4 \cdot 10^{-1}$
MLSDC cold	$1.5625 \cdot 10^{-1}$	169.8	$1.1 \cdot 10^{-1}$
MLSDC warm	$1.5625 \cdot 10^{-1}$	132.7	$1.3 \cdot 10^{-1}$

that the time discretization error compared to the reference solution is approximately the same for each case, which would result in similar progress of the full optimization process. All other parameters are chosen as before.

One complication in this comparison is that computing the stage values in the Runge–Kutta method requires evaluating the control u and desired state y_d at intermediate times (the MLSDC method requires this data at the collocation nodes too). For MLSDC, we chose symmetric collocation nodes, such that forward-in-time and backward-in-time use the same nodes. Thus, u, y_d have distinct values at the collocation nodes. However, in the ARK-4 method, the stage values are not symmetric, and interpolation or dense output would be required to compute these values. For simplicity, we therefore (artificially) set u and y_d constant on the time steps for this example. Results are reported in Table 5.9. Despite the small time step size, IMEX-Euler is the fastest of the sequential methods. However, using PFASST with warm starting, speedup is achieved already by adding the second processor (speedup 1.2). Compared to IMEX-Euler, on 32 processors, PFASST with cold starting achieved a speedup/parallel efficiency of 3.2/10%; with warm starts, this increases to 4.6/14%. As noted before, the benefit of MLSDC compared to SDC is greater for problems in more than one dimension, and hence these results should not be considered as a general comparison.

6. Conclusions and outlook. In this paper, we introduce an efficient fully time-parallel strategy for gradient-based optimization with parabolic PDEs. In the most critical component, the computation of the reduced gradient, we discuss and compare three competing approaches for applying the PFASST algorithm to the solution of the state and adjoint equations. While simultaneously solving state and adjoint induces severe communication and wait times in the implementation, both the first state, then adjoint approach and the mixed approach yield comparatively good speedups and parallel efficiency of up to 29%. The warm starting capability of MLSDC/PFASST is an important feature and makes the method competitive with standard time stepping methods even in a sequential run. It is also noteworthy that speedup is obtained already with two processors.

Although the presented results are promising, there are several additional avenues to further increase the parallel speedup or efficiency of the approach. For example, the test cases studied here were set in simple periodic domains using the FFT for implicit solves. In many cases, implicit solves are done iteratively, and hence solver tolerances can be adjusted dynamically to do less work when less accuracy is needed [43]. The same is true of the tolerances used to terminate PFASST iterations. Likewise, the spatial or temporal order of the PDE solver could be changed dynamically during the optimization process. Hence, a dynamic, adaptive strategy for controlling these tolerances and accuracy in the optimization loop is a promising direction for future study. Similarly, analyzing the impact of inexact storage of solution values for warm starting will be necessary for larger scale applications. Finding suitable extensions of the mixed approach to nonlinear state equations is also of interest. For this approach, it might also be beneficial to solve state and modified adjoint equations in the first step *concurrently*, making use of additional processors during SDC sweeps.

REFERENCES

[1] *dune-PFASST: A Time-Parallel Solver for Partial Differential Equations Using the Finite Element Method for Spatial Discretisation*, https://github.com/Parallel-in-Time/dune-PFASST.

[2] *LibPFASST: A Lightweight Implementation of the PFASST Algorithm*, https://github.com/libpfasst/LibPFASST.

[3] *OPTPDE — A Collection of Problems in PDE-Constrained Optimization*, http://www.optpde.net.

[4] *XBraid: Parallel Multigrid in Time*, http://llnl.gov/casc/xbraid.

[5] A. T. Barker and M. Stoll, *Domain decomposition in time for PDE-constrained optimization*, Comput. Phys. Commun., 197 (2015), pp. 136–143.

[6] M. Bolten, D. Moser, and R. Speck, *A multigrid perspective on the parallel full approximation scheme in space and time*, Numer. Linear Algebra Appl., 24 (2017), e2110.

[7] A. Bourlioux, A. T. Layton, and M. L. Minion, *High-order multi-implicit spectral deferred correction methods for problems of reactive flow*, J. Comput. Phys., 189 (2003), pp. 651–675.

[8] R. Buchholz, H. Engel, E. Kammann, and F. Tröltzsch, *On the optimal control of the Schlögl-model*, Comput. Optim. Appl., 56 (2013), pp. 153–185.

[9] Y. H. Dai and Y. Yuan, *A nonlinear conjugate gradient method with a strong global convergence property*, SIAM J. Optim., 10 (1999), pp. 177–182, https://doi.org/10.1137/S1052623497318992.

[10] X. Deng and M. Heinkenschloss, *A Parallel-in-Time Gradient-Type Method for Discrete Time Optimal Control Problems*, preprint, Department of Computational and Applied Mathematics, Rice University, Houston, TX, 2016; available online from http://www.caam.rice.edu/~heinken.

[11] P. Deuflhard, *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*, 2nd ed., Springer, Heidelberg, 2006.

[12] A. Dutt, L. Greengard, and V. Rokhlin, *Spectral deferred correction methods for ordinary differential equations*, BIT, 40 (2000), pp. 241–266.

[13] M. Emmett and M. L. Minion, *Toward an efficient parallel in time method for partial differential equations*, Commun. Appl. Math. Comput. Sci., 7 (2012), pp. 105–132.

[14] M. Emmett and M. L. Minion, *Efficient implementation of a multi-level parallel in time algorithm*, in Domain Decomposition Methods in Science and Engineering XXI, J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. B. Widlund, eds., Springer, Cham, 2014, pp. 359–366.

[15] H. Finsberg, C. Xi, J. L. Tan, L. Zhong, M. Genet, J. Sundnes, L. C. Lee, and S. T. Wall, *Efficient estimation of personalized biventricular mechanical function employing gradient-based optimization*, Int. J. Numer. Methods Biomed. Eng., 34 (2018), e2982, https://doi.org/10.1002/cnm.2982.

[16] R. Fletcher and C. M. Reeves, *Function minimization by conjugate gradients*, Comput. J., 7 (1964), pp. 149–154.

[17] M. J. Gander, 50 *years of time parallel time integration*, in Multiple Shooting and Time Domain Decomposition Methods, T. Carraro et al., eds., Springer, Cham, 2015, pp. 69–113.

[18] M. J. Gander and S. Güttel, *PARAEXP: A parallel integrator for linear initial-value problems*, SIAM J. Sci. Comput., 35 (2013), pp. C123–C142, https://doi.org/10.1137/110856137.

[19] M. J. Gander and F. Kwok, *Schwarz methods for the time-parallel solution of parabolic control problems*, in Domain Decomposition Methods in Science and Engineering XXII, T. Dickopf et al., eds., Springer, Cham, 2016, pp. 207–216.

[20] S. Götschel, C. Maierhofer, J. P. Müller, N. Rothbart, and M. Weiser, *Quantitative defect reconstruction in active thermography for fiber-reinforced composites*, in Proceedings of the 19th World Conference on Non-Destructive Testing, 2016.

[21] S. Götschel and M. L. Minion, *Parallel-in-time for parabolic optimal control problems using PFASST*, in Domain Decomposition Methods in Science and Engineering XXIV, P. E.

Bjørstad, S. C. Brenner, L. Halpern, H. H. Kim, R. Kornhuber, T. Rahman, and O. B. Widlund, eds., Springer, Cham, 2018, pp. 363–371.

[22] S. Götschel and M. Weiser, *Lossy compression for PDE-constrained optimization: Adaptive error control*, Comput. Optim. Appl., 62 (2015), pp. 131–155.

[23] S. Günther, N. R. Gauger, and J. B. Schroder, *A non-intrusive parallel-in-time adjoint solver with the XBraid library*, Comput. Vis. Sci., 19 (2018), pp. 85–95.

[24] S. Günther, N. R. Gauger, and J. B. Schroder, *A non-intrusive parallel-in-time approach for simultaneous optimization with unsteady PDEs*, Optim. Methods Softw., 34 (2019), pp. 1306–1321.

[25] S. Günther, L. Ruthotto, J. Schroder, E. Cyr, and N. Gauger, *Layer-Parallel Training of Deep Residual Neural Networks*, preprint, https://arxiv.org/abs/1812.04352, 2018.

[26] S. Güttel and J. W. Pearson, *A rational deferred correction approach to parabolic optimal control problems*, IMA J. Numer. Anal., 38 (2018), pp. 1861–1892.

[27] E. Haber and L. Ruthotto, *Stable architectures for deep neural networks*, Inverse Problems, 34 (2017), 014004.

[28] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations* II: *Stiff and Differential-Algebraic Problems*, Springer, Berlin, Heidelberg, 1991.

[29] M. Heinkenschloss, *A time-domain decomposition iterative method for the solution of distributed linear quadratic optimal control problems*, J. Comput. Appl. Math., 173 (2005), pp. 169–198.

[30] R. Herzog, A. Rösch, S. Ulbrich, and W. Wollner, *OPTPDE: A collection of problems in PDE-constrained optimization*, in Trends in PDE Constrained Optimization, Internat. Ser. Numer. Math. 165, G. Leugering, P. Benner, S. Engell, A. Griewank, H. Harbrecht, M. Hinze, R. Rannacher, and S. Ulbrich, eds., Springer, Cham, 2014, pp. 539–543, https://doi.org/10.1007/978-3-319-05083-6_34.

[31] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, *Optimization with PDE Constraints*, Springer, Berlin, 2009.

[32] C. A. Kennedy and M. H. Carpenter, *Additive Runge–Kutta schemes for convection–diffusion–reaction equations*, Appl. Numer. Math., 44 (2003), pp. 139–181, https://doi.org/10.1016/S0168-9274(02)00138-1.

[33] B. T. Krull and M. L. Minion, *Parallel-in-time Magnus integrators*, SIAM J. Sci. Comput., 41 (2019), pp. A2999–A3020, https://doi.org/10.1137/18M1174854.

[34] J.-L. Lions, Y. Maday, and G. Turinici, *A "parareal"in time discretization of PDEs*, C. R. Acad. Sci. Paris Sér. I Math., 332 (2001), pp. 661–668.

[35] T. P. Mathew, M. Sarkis, and C. E. Schaerer, *Analysis of block parareal preconditioners for parabolic optimal control problems*, SIAM J. Sci. Comput., 32 (2010), pp. 1180–1200, https://doi.org/10.1137/080717481.

[36] M. Minion, *A hybrid parareal spectral deferred corrections method*, Commun. Appl. Math. Comput. Sci., 5 (2010), pp. 265–301.

[37] M. L. Minion, *Semi-implicit spectral deferred correction methods for ordinary differential equations*, Commun. Math. Sci., 1 (2003), pp. 471–500.

[38] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, 2006.

[39] E. Polak and G. Ribiere, *Note sur la convergence de méthodes de directions conjuguées*, Rev. Française Informat. Recherche Opérationnelle, 3 (1969), pp. 35–43.

[40] B. T. Polyak, *The conjugate gradient method in extremal problems*, USSR Comput. Math. Math. Phys., 9 (1969), pp. 94–112.

[41] L. Ruthotto and E. Haber, *Deep neural networks motivated by partial differential equations*, J. Math. Imaging Vision, (2019), https://doi.org/10.1007/s10851-019-00903-1.

[42] R. Speck, D. Ruprecht, M. Emmett, M. L. Minion, M. Bolten, and R. Krause, *A multi-level spectral deferred correction method*, BIT, 55 (2015), pp. 843–867.

[43] R. Speck, D. Ruprecht, M. Minion, M. Emmett, and R. Krause, *Inexact spectral deferred corrections*, in Domain Decomposition Methods in Science and Engineering XXII, T. Dickopf et al., eds., Springer, Cham, 2016, pp. 389–396.

[44] S. Ulbrich, *Preconditioners based on "parareal" time-domain decomposition for time-dependent PDE-constrained optimization*, in Multiple Shooting and Time Domain Decomposition Methods, T. Carraro et al., eds., Springer, Cham, 2015, pp. 203–232.

[45] M. Weiser, *Faster SDC convergence on non-equidistant grids by DIRK sweeps*, BIT, 55 (2013), pp. 1219–1241.

[46] E. Zeidler, *Nonlinear Functional Analysis and Its Applications* I: *Fixed-Point Theorems*, Springer, New York, 1986.