SIAM J. SCI. COMPUT. © 2015 Society for Industrial and Applied Mathematics
Vol. 37, No. 5, pp. S244–S263

INTERWEAVING PFASST AND PARALLEL MULTIGRID[*]

M. L. MINION[†], R. SPECK[‡], M. BOLTEN[§], M. EMMETT[†], AND D. RUPRECHT[¶]

Abstract. The parallel full approximation scheme in space and time (PFASST) introduced by Emmett and Minion in 2012 is an iterative strategy for the temporal parallelization of ODEs and discretized PDEs. As the name suggests, PFASST is similar in spirit to a space-time full approximation scheme multigrid method performed over multiple time steps in parallel. However, since the original focus of PFASST was on the performance of the method in terms of time parallelism, the solution of any spatial system arising from the use of implicit or semi-implicit temporal methods within PFASST have simply been assumed to be solved to some desired accuracy completely at each substep and each iteration by some unspecified procedure. It hence is natural to investigate how iterative solvers in the spatial dimensions can be interwoven with the PFASST iterations and whether this strategy leads to a more efficient overall approach. This paper presents an initial investigation on the relative performance of different strategies for coupling PFASST iterations with multigrid methods for the implicit treatment of diffusion terms in PDEs. In particular, we compare full accuracy multigrid solves at each substep with a small fixed number of multigrid V-cycles. This reduces the cost of each PFASST iteration at the possible expense of a corresponding increase in the number of PFASST iterations needed for convergence. Parallel efficiency of the resulting methods is explored through numerical examples.

Key words. parallel in time, PFASST, multigrid

AMS subject classifications. 65N55, 65N40, 65N06, 65Y05

DOI. 10.1137/14097536X

1. Introduction.

The past decade has seen a growing interest in the development of parallel methods for temporal integration of ordinary differential equations (ODEs), particularly in the context of temporal strategies for partial differential equations (PDEs). One factor fueling this interest is related to the evolution of supercomputers during this time. Since the end of the exponential increase in individual processors speeds, increases in supercomputer speeds have been mostly due to increases in the number of computational cores, and current projections suggest that the first exaflop computer will contain on the order of a billion cores [15]. The implication of this trend is that increasing concurrency in algorithms is essential, and in the case of time-dependent PDE simulations, the use of space-time parallelism is an attractive option.

[*]Received by the editors July 1, 2014; accepted for publication (in revised form) March 23, 2015; published electronically October 29, 2015.
http://www.siam.org/journals/sisc/37-5/97536.html

[†]Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (mlminion@lbl.gov, mwemmett@lbl.gov). These authors' work was supported by the Applied Mathematics Program of the U.S. Department of Energy Office of Advanced Scientific Computing Research under contract DE-AC02-05CH11231.

[‡]Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany, and Institute of Computational Science, Università della Svizzera italiana, Lugano, Switzerland (r.speck@fz-juelich.de).

[§]Department of Mathematics, Bergische Universität, Wuppertal, Germany (bolten@math.uni-wuppertal.de). Computing time on JUQUEEN at the Jülich Supercomputing Centre (JSC) was provided by project HWU12.

[¶]Institute of Computational Science, Università della Svizzera italiana, Lugano, Switzerland (daniel.ruprecht@usi.ch). This author's work was supported by the Swiss National Science Foundation (SNSF) under the lead agency agreement through the project "ExaSolvers" (SNF-145271) within the Priority Programme 1648 "Software for Exascale Computing" (SPPEXA) of the Deutsche Forschungsgemeinschaft.

Time parallel methods have a long history dating back at least to the work of Nievergelt [27]. In the context of the space-time multigrid, Hackbusch noted in 1984 that relaxation operators in a parabolic multigrid can be employed on multiple time steps simultaneously [16]. The 1997 review article by Burrage [7] provides a summary of early work on the subject. More recently, the parareal method proposed in 2001 [23] has renewed interest in temporal parallelization methods. In 2012 the parallel full approximation scheme in space and time (PFASST) was introduced by Emmett and Minion [10], and performance results for PFASST using space-time parallelization with hundreds of thousands of cores can be found in [32, 29].

The PFASST algorithm is based on a type of deferred corrections strategy for ODEs [9], with corrections being applied on multiple time steps in parallel. As such, there are similarities between parareal and PFASST (see [26, 25]). On the other hand, the parallel efficiency of PFASST depends on the construction of a hierarchy of space-time discretizations; hence there are also similarities between PFASST and space-time multigrid methods. However, in the original papers on PFASST, the solution of any spatial systems due to implicit time-stepping was assumed to be found to full precision since the interest was in the temporal accuracy and efficiency of the methods. From this point of view, PFASST is an iterative solver in the time direction but not in the spatial dimensions. This is, in a sense, orthogonal to the traditional use of multigrid solvers within PDE methods, where multigrid is used to iteratively solve spatial equations and the time direction is not iterative. One of the main goals of this paper is to investigate the use of both spatial and temporal iterative methods utilizing PFASST and multigrid.

To be more specific, the iterative strategy within the PFASST algorithm is derived from the method of spectral deferred corrections (SDCs) [9], which is a variant of the defect and deferred correction methods developed in the 1960s [36, 28, 33, 2]. One advantage of SDC methods is that it is straightforward to construct methods of very high order of accuracy by iteratively applying low-order methods to a series of correction equations. This flexibility has been exploited to construct higher-order semi-implicit (IMEX) and multi-implicit SDC methods [24, 4, 21], as well as multirate SDC methods [5]. Such methods are very difficult to construct using traditional Runge–Kutta or linear-multistep approaches.

The main disadvantage of SDC methods is that they have a high cost per time step in the sense of the number of function evaluations (explicit or implicit) required per time step. When high accuracy is desired, this cost is generally offset by the use of relatively large time steps compared to lower-order methods for the same given accuracy. Nevertheless, approaches to reducing the cost of each SDC iteration have been proposed, including those generally referred to as *multi-level* SDC methods (MLSDC). The main idea in MLSDC is to perform some SDC iterations on coarser discretizations of the problem, and methods that coarsen the temporal discretization, the spatial discretization, and the order of the spatial approximation have recently been investigated [31]. SDC iterations are then performed on the hierarchy of levels in much the same way as V-cycles in traditional multigrid. The PFASST algorithm can be considered a time parallel version of MLSDC.

Using implicit SDC methods (as well as popular methods like backward Euler, trapezoid rule, diagonally implicit Runge–Kutta, or backward difference formula) for grid-based PDE simulations requires the solution of implicit equations in the spatial domain. If an iterative solver is used for these equations in the context of SDC, one advantageous result is that the initial guess for each solve becomes better as the SDC iterations converge. This raises the possibility of reducing the cost of SDC or MLSDC iterations further by limiting the number of spatial iterations used for each implicit solve rather than requiring each to be done to some specified tolerance. A recent paper explores this possibility in the context of SDC methods [8]. We refer to variants in which spatial solves are done only incompletely with a prepended capital "I" for "inexact" (ISDC, IMLSDC, IPFASST), resulting in the interweaving of spatial and temporal iterations. This paper adopts the ISDC strategy for MLSDC and PFASST and explores the performance of both IMLSDC and IPFASST.

As a starting point, we focus here on the linear test problem resulting from a finite-difference approximation of the heat equation. Variants of this example are considered in early papers on space-time multigrid [18, 19], waveform relaxation [34, 14], and block-parallel methods [1], as well as more recent papers on parallel space-time multigrid [13, 12]. In all the cases above, a second-order finite-difference approximation is employed in space and first- or second-order methods are used in time. Although this is not the optimal setting for the PFASST algorithm, we present results using second-order space-time discretization to compare with other published results. We also show how, for a given accuracy, using fourth-order methods in space and/or time results in significant computational savings compared to second-order methods.

The remainder of this paper is organized as follows. We first present the SDC algorithm for linear problems in a compact notation that can be interpreted as matrix iterations and highlight the similarities between SDC iterations and classical relaxation schemes. We then discuss how IMLSDC and IPFASST are constructed. In section 3, we provide the relevant analysis of SDC methods as a relaxation operator for parabolic problems. In section 4 we examine the scaling of the IPFASST method in terms of problem size and number of parallel time steps, consider the effect of limiting the number of multigrid V-cycles per implicit solve, and provide numerical examples comparing PFASST with IPFASST. Finally, in section 4.2 we provide timing comparisons for three-dimensional examples scaling to hundreds of thousands of cores.

2. Collocation and SDC.

This section briefly describes the methods used later. In section 2.1, the collocation formulation for initial value problems is first reviewed. SDC as an iterative solver for a collocation solution is discussed briefly in section 2.2. A compact notation of SDC for linear problems is given, and its interpretation as relaxation is discussed. The extension of SDC to MLSDC is outlined in section 2.3, including a number of strategies to coarsen the representation of the problem in order to reduce the overall cost. The possibility of solving linear systems approximately on all levels leads to ISDCs and their multi-level counterparts IMLSDC and IPFASST.

2.1. The collocation formulation.

This paper concerns methods for the solution of initial value ODEs, particularly those arising from the spatial discretization of PDEs through the method of lines technique. To set notation, consider the scalar ODE of the form

$$y'(t) = f(t, y(t)), \tag{2.1}$$

$$y(t_0) = y_0, \tag{2.2}$$

where $y(t), y_0 \in \mathbf{C}$ and $f : \mathbb{R} \times \mathbf{C} \to \mathbf{C}$. Similarly for systems of equations

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)), \tag{2.3}$$

$$\mathbf{y}(t_0) = \mathbf{y}_0, \tag{2.4}$$

where $\mathbf{y}(t), \mathbf{y}_0 \in \mathbf{C}^D$ and $\mathbf{f} : \mathbb{R} \times \mathbf{C}^D \to \mathbf{C}^D$. An equivalent form is given by the Picard equation, which for (2.3) is

$$\mathbf{y}(t) = \mathbf{y}_0 + \int_{t_0}^{t} \mathbf{f}(\tau, \mathbf{y}(\tau))d\tau. \tag{2.5}$$

Since the goal here is to describe numerical methods for a given time step, consider the time interval $t \in [t_n, t_{n+1}]$ with $\Delta t = t_{n+1} - t_n$. Define the set of points t_m for $m = 0, \ldots, M$ to be quadrature nodes scaled to $[t_n, t_{n+1}]$, so that $t_n = t_0 < t_1 < t_2 < \cdots < t_M = t_{n+1}$. Given a scalar function $g(t)$, define the approximations to the integral of $g(t)$ over the intervals $[t_0, t_m]$ by choosing the coefficients $q_{m,i}$ such that

$$\int_{t_0}^{t_m} g(\tau)d\tau \approx \Delta t \sum_{i=0}^{M} q_{m,i} g(t_i).$$

The coefficients $q_{m,i}$ that give the highest order of accuracy given the points t_m can be derived using standard techniques. The quadrature rules used here have the property that $q_{m,0} \equiv 0$ (see [22] for a discussion of different choices of quadrature rules for SDC methods). Let the matrix Q of size $(M+1) \times (M+1)$ be composed of the coefficients $q_{m,i}$. The first row of Q contains only zeros. Q will be referred to here as the integration matrix.

Considering for the time being scalar equations, the integration matrix Q can be used to discretize (2.5) directly over the interval $[t_n, t_{n+1}]$. Let $y_0 = y_n \approx y(t_n)$ be the initial condition, and define $y_m \approx y(t_m)$ by

$$y_m = y_n + \Delta t \sum_{i=0}^{M} q_{m,i} f(t_i, y_i) \qquad \text{for } m = 0, \ldots, M. \tag{2.6}$$

We can write this equation in a compact form by denoting the vector of unknowns by $Y = [y_0 \ldots y_M]^T$, and the vector of function values $F(Y) = [f(y_0, t_0) \ldots f(y_M, t_M)]^T$. Furthermore, let Y_0 be the $(M+1) \times 1$ column vector with each entry equal to y_0. Then (2.6) is equivalent to

$$Y = Y_0 + \Delta t Q F(Y). \tag{2.7}$$

This coupled (generally nonlinear) equation can be solved directly for the values Y, resulting in a collocation scheme for the ODE.

It should also be noted that the collocation form (2.7) corresponds to a fully implicit Runge–Kutta (IRK) method given by the Butcher tableau

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b} \end{array}, \tag{2.8}$$

where $\mathbf{c}$ are the nodes t_m scaled to the unit interval, $\mathbf{A} = Q$, and the vector $\mathbf{b}$ corresponds to the last row of Q.

For the simplest linear scalar equation where $f(y, t) = \lambda y$ in (2.1), the collocation formulation given in (2.7) becomes

$$Y = Y_0 + \lambda \Delta t Q Y, \tag{2.9}$$

or in more standard matrix form

$$(I_{M+1} - \lambda \Delta t Q)Y = Y_0, \tag{2.10}$$

with I_{M+1} being the $(M+1)\times(M+1)$ identity matrix. In the case of a linear system of equations $\mathbf{f}(t,\mathbf{y}) = A\mathbf{y}$, where A is a $D \times D$ matrix, the Picard equation becomes

$$\mathbf{y}(t) = \mathbf{y}_n + \int_{t_n}^{t} A\mathbf{y}(\tau)d\tau, \tag{2.11}$$

where the integration is done term by term. To discretize (2.11), the solution is the vector

$$\mathbf{Y} = [y_0^1 \dots y_0^D \dots y_M^1 \dots y_M^D]^T \tag{2.12}$$

of length $D(M+1)$. Here the subscripts m correspond to the quadrature nodes in time and superscripts j correspond to the component of the solution vector.

To apply the numerical quadrature matrices to $\mathbf{Y}$, let $\mathbf{Q} = Q \otimes I_D$, where I_D is the $D \times D$ identity matrix. Likewise let $\mathbf{A} = I_{M+1} \otimes A$, and let $\mathbf{Y}_0$ be the vector of length $(M+1)D$ consisting of M copies of the $\mathbf{y}_0$. Then the analogous form of (2.7) is

$$\mathbf{Y} = \mathbf{Y}_0 + \Delta t \mathbf{QAY} \tag{2.13}$$

or

$$(\mathbf{I} - \Delta t \mathbf{QA})\mathbf{Y} = \mathbf{Y}_0, \tag{2.14}$$

where $\mathbf{I} = I_{M+1} \otimes I_D$.

2.2. Single level SDC. SDC can be understood as a preconditioned fixed point iteration to solve (2.14), which avoids treating the full system directly by computing a series of corrections node by node (see, e.g., [20, 35]). Note that originally SDC was introduced in a different way, as a variant of earlier deferred and defect correction schemes [36, 28] designed to achieve a fixed order of accuracy for a fixed number of correction sweeps [9]. The MLSDC methods (and PFASST) described below move away from the idea of a fixed number of iterations in favor of the convergence toward the collocation (or IRK) solution.

Here we provide a short review of the SDC method. For more details see [9, 24]. Let the superscript k denote the numerical approximation at the kth SDC iteration. Using backward Euler as the base method, one iteration of SDC can be written as a sweep through the quadrature nodes, successively updating the solutions. In the case of scalar equations, this takes the form

$$y_{m+1}^{k+1} = y_m^{k+1} + \Delta t_m \left(f(t_{m+1}, y_{m+1}^{k+1}) - f(t_{m+1}, y_{m+1}^{k}) \right) + Q_m^{m+1}(Y^k) \tag{2.15}$$

for $m = 0, \dots, M-1$, where

$$Q_m^{m+1}(Y^k) \approx \int_{t_m}^{t_{m+1}} f(\tau, y^k(\tau))d\tau. \tag{2.16}$$

The values of $Q_m^{m+1}(Y^k)$ can easily be constructed using the integration matrix $\mathbf{Q}$,

$$Q_m^{m+1}(Y^k) = \Delta t \sum_{i=0}^{M} (q_{m+1,i} - q_{m,i}) f(t_i, y_i^k). \tag{2.17}$$

Here we consider only implicit SDC methods, even though one particularly attractive feature of SDC is the flexibility to use different base methods in order to create, e.g., high-order implicit-explicit or multirate methods [24, 4, 17, 5].

2.2.1. Compact notation. We will now present a compact notation of the SDC iterations. Given the points $t_m \in [t_n, t_{n+1}]$ as discussed above, let $\Delta t_m = t_m - t_{m-1}$, and let $\gamma_m = \Delta t_m / \Delta t$, where again $\Delta t = t_{n+1} - t_n$. We begin by defining the lower-order integration matrix that has the same dimensions as Q:

$$Q_I = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & \gamma_1 & \cdots & 0 & 0 \\ \cdot & \cdot & \cdots & 0 & 0 \\ 0 & \gamma_1 & \cdots & \gamma_{M-1} & 0 \\ 0 & \gamma_1 & \cdots & \gamma_{M-1} & \gamma_M \end{bmatrix}. \tag{2.18}$$

Using the same notation as in section 2.1, M steps of backward Euler for (2.1) can be written as

$$Y = Y_0 + \Delta t Q_I F(Y). \tag{2.19}$$

This differs from the collocation formulation (2.7) only in that Q_I replaces Q.

Since Q_I is lower triangular with nonzero diagonal entries, (2.19) can be solved sequentially for the values y_m, where each value requires an implicit equation to be solved of the form

$$y_{m+1} = y_m + \Delta t_m f(t_{m+1}, y_{m+1}).$$

Using the matrix-vector notation above, one SDC sweep for (2.3) can be compactly written as

$$\mathbf{Y}^{k+1} = \mathbf{Y}_0 + \Delta t \mathbf{Q_I}(\mathbf{F}(\mathbf{Y}^{k+1}) - \mathbf{F}(\mathbf{Y}^k)) + \Delta t \mathbf{Q}\mathbf{F}(\mathbf{Y}^k) \tag{2.20}$$

or

$$\mathbf{Y}^{k+1} = \mathbf{Y}_0 + \Delta t \mathbf{Q_I}\mathbf{F}(\mathbf{Y}^{k+1}) + \Delta t(\mathbf{Q} - \mathbf{Q_I})\mathbf{F}(\mathbf{Y}^k). \tag{2.21}$$

2.2.2. SDC for linear problems. The compact formulas derived above can be recast as matrix-vector operations when the governing equation is linear. In the case of the linear system, $\mathbf{y}' = A\mathbf{y}$, (2.20) simplifies to

$$\mathbf{Y}^{k+1} = \mathbf{Y}_0 + \Delta t \mathbf{Q_I}(\mathbf{A}\mathbf{Y}^{k+1} - \mathbf{A}\mathbf{Y}^k) + \mathbf{Q}\mathbf{A}\mathbf{Y}^k \tag{2.22}$$

or

$$(\mathbf{I} - \mathbf{Q_I}\mathbf{A})\,\mathbf{Y}^{k+1} = \mathbf{Y}_0 + \Delta t\,(\mathbf{Q} - \mathbf{Q_I})\,\mathbf{A}\mathbf{Y}^k. \tag{2.23}$$

As in the case of backward Euler, this system of equations can be solved by substepping, requiring the solution of

$$(I - \Delta t_m A)\mathbf{y}_{m+1}^{k+1} = \mathbf{y}_0 - \Delta t_m A \mathbf{y}_{m+1}^k + \mathbf{Q}_m^{m+1}(\mathbf{Y}^k). \tag{2.24}$$

This equation involves the inversion of the same operator that arises from a backward Euler method with a modified right-hand side. Note that as the SDC iterations converge in k, an increasingly good approximation to this solution is provided by $\mathbf{y}_{m+1}^k$. This fact has two relevant implications when iterative methods are employed to solve the system. First, when considering the total cost of one time-step of SDC, the reduced cost of the implicit solves as k increases should be taken into consideration.

Second, and more relevant for this paper, is that an iterative method need not solve (2.24) to full precision at each iteration. Instead, a fixed number of iterations could be performed during each SDC iteration, or each could be done so that the residual is reduced by some set tolerance. In the numerical studies presented in section 4, multigrid methods are employed for solving (2.24) and we investigate limiting the number of multigrid V-cycles instead of solving to full precision.

Finally, note that in the SDC iterations, it is straightforward to compute the residual in terms of the solution to (2.13), specifically

$$\mathbf{r}^k := \mathbf{Y}_0 - (\mathbf{I} - \Delta t \mathbf{Q}\mathbf{A})\mathbf{Y}^k. \tag{2.25}$$

In section 4, the norm of $\mathbf{r}^k$ is used to monitor convergence of the different choices of methods.

2.2.3. SDC as a relaxation. For a linear problem, it is possible to write an SDC sweep described above as a relaxation operator applied to the linear collocation equation

$$(\mathbf{I} - \Delta t \mathbf{Q}\mathbf{A})\mathbf{Y} = \mathbf{Y}_0. \tag{2.26}$$

As in classical iteration methods for linear systems, we look for a decomposition of the matrix $(\mathbf{I} - \Delta t \mathbf{Q}\mathbf{A}) = \mathbf{M} - \mathbf{K}$ such that $\mathbf{M}$ is relatively less expensive to invert than $(\mathbf{I} - \Delta t \mathbf{Q}\mathbf{A})$. Then a classical relaxation scheme based on this splitting would be

$$\mathbf{Y}^{k+1} = \mathbf{M}^{-1}\mathbf{K}\mathbf{Y}^k + \mathbf{M}^{-1}\mathbf{Y}_0. \tag{2.27}$$

Choosing

$$\mathbf{M} = \mathbf{I} - \Delta t \mathbf{Q}_\mathbf{I}\mathbf{A} \quad \text{and} \quad \mathbf{K} = \Delta t(\mathbf{Q} - \mathbf{Q}_\mathbf{I})\mathbf{A} \tag{2.28}$$

produces the SDC sweep as given by (2.23); hence inverting $\mathbf{M}$ can be done by substepping on the SDC nodes solving the appropriate version of (2.24).

In the next section we carry this analogy further by introducing variants of SDC that utilize multiple levels of resolution as in classical multigrid methods.

2.3. MLSDC. The goal of MLSDC methods is that by introducing a hierarchy of levels from fine to coarse, some of the expensive fine level correction sweeps can be replaced with sweeps on coarser levels, so that the runtime required for convergence of SDC iterations is reduced (see recent results in [31]). A space-time full approximation scheme (FAS) term is employed when forming coarser level equations, which is the difference between the coarsening of the latest fine function values and the coarse level function applied to a coarsening of the latest fine solution. The result of including the FAS term is that the accuracy on the coarse level approaches that of the fine level as the MLSDC iterations converge (see, for example, [6]). Different strategies for reducing the cost of SDC sweeps on the coarser levels are explored in [31], including using a lower-order spatial discretization, using a spatial mesh with fewer points, or performing only incomplete implicit solves. The structure of an MLSDC level hierarchy is sketched in Figure 1. For a detailed explanation of the method including performance results we refer to [31]. MLSDC is the basic building block for the time parallel PFASST method summarized in section 2.4.

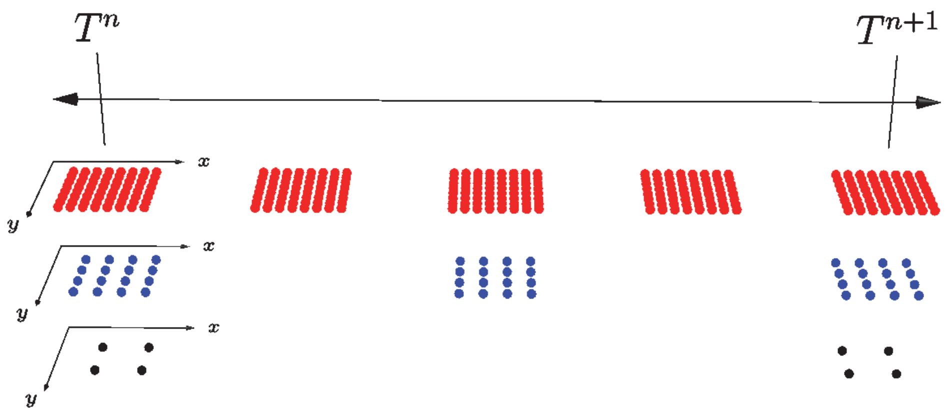

FIG. 1. *An MLSDC level hierarchy over a time step* $[T^n, T^{n+1}]$. *The upper fine level (red) uses five quadrature nodes, the medium level (blue) three, and the bottom coarsest level (black) two. Also sketched is the possibility of using a spatial discretization with fewer degrees of freedom on the coarser levels: Depicted is an* 8×8 *point mesh on the fine level, a* 4×4 *point mesh in the middle, and a* 2×2 *point mesh on the coarse level. A detailed description of MLSDC can be found in* [31]. *Figure reprinted from* [29].

2.3.1. Inexact MLSDC. In [8], another approach to reducing the cost of SDC methods is discussed, ISDC. Here, implicit solves in SDC sweeps (2.15) are computed only approximately, e.g., by a fixed small number of V-cycles of multigrid. While this strategy makes each SDC sweep less expensive, it can increase the number of SDC iterations required to reduce the residual to a given threshold. If the increase in iterations is not too large, ISDC can in total be faster than SDC. In [8] it is demonstrated that ISDC can save up to 50% of the V-cycles required by regular SDC while converging to the same tolerance. It is also shown that the reason this strategy works is the increasingly accurate initial guesses provided for the iterative solver by SDC (see the discussion toward the end of section 2.2.2).

In regular MLSDC, doing incomplete solves on the coarse levels is discussed and explored in [31] as a means to reduce the cost of coarse level sweeps. However, the ISDC strategy can also be incorporated into MLSDC, resulting in an IMLSDC method. Here, all implicit solves are done only approximately, including the finest level. Ideally, IMLSDC would save on runtime compared to ISDC, just as MLSDC does compared to SDC. However, the results presented in section 4 suggest that this is not necessarily the case. However, MLSDC serves as the basis of the parallel-in-time method PFASST, and hence IMLSDC could be used in parallel as well. In section 4 we demonstrate that IPFASST (see section 2.4.1) can provide a significant reduction of runtime by exploiting temporal concurrency. A detailed description of MLSDC including pseudo code can be found in [31], and we refer the reader there for details.

2.4. The parallel full approximation scheme in space and time. PFASST, introduced in [25, 10], is an iterative time parallel method for PDEs that has similarities to both the parareal method and space-time multigrid methods. In PFASST each time step is assigned its own processor or, if combined with spatial parallelization, its own communicator (for a more detailed explanation of the latter case, see [32]). PFASST can be considered a time parallel extension of MLSDC, where after an initialization procedure, MLSDC iterations are performed on multiple time steps in parallel with updates to initial conditions being passed between processors as each SDC sweep is completed.

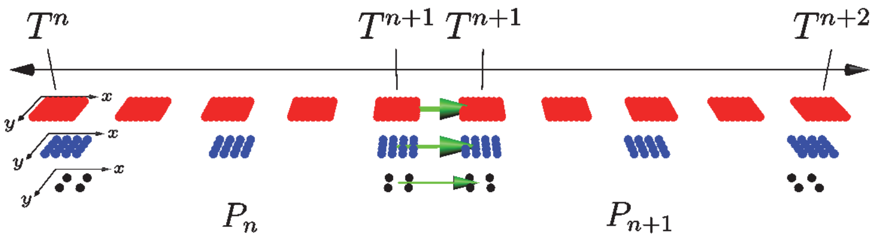

FIG. 2. *PFASST performs MLSDC iterations concurrently on multiple time steps. For simplicity, only two time steps* $[T^n, T^{n+1}]$ *and* $[T^{n+1}, T^{n+2}]$ *are sketched here. After each sweep, an updated initial value on the corresponding level is sent forward to the processor handling the next time step. In the setup above, processor* P_n *handles time step* $[T^n, T^{n+1}]$ *and sends forward updates to processor* P_{n+1}*, which handles* $[T^{n+1}, T^{n+2}]$*. Figure reprinted from* [29].

PFASST typically starts by distributing the initial value over all time-ranks as an initial guess, which is then refined by a number of sweeps on the coarse level, where processors handling time steps later in time do more sweeps (this is usually referred to as the predictor phase). After completing the predictor phase, each processor starts with its own MLSDC iterations while, after each sweep, sending forward an updated initial value for the current level to the processor handling the next time step; cf. Figure 2. Blocking communication is required on the coarsest level only, so that minimal synchronicity between the different MLSDC iterations is required; see [11]. Benchmarks for PFASST in large-scale parallel simulations illustrate how PFASST can extend strong scaling limits [32] or improve utilization of large parallel installations in comparison to codes utilizing only spatial parallelism [29].

A detailed description of PFASST including a sketch of the algorithm in pseudo code can be found in [10], and we refer the reader there for more details.

2.4.1. IPFASST. Just as incomplete implicit solves can be used in MLSDC, they can also be used in PFASST. Essentially, each processor now performs IMLSDC iterations instead of iterations with full solves on the fine level. Apart from that, IPFASST proceeds the same as PFASST. In section 4, performance of IPFASST will be studied through numerical examples.

3. Analysis of SDC on the linear model problem.

In section 2.2.3, the analogy between a single SDC sweep and a classical relaxation scheme derived from a splitting of the linear system is presented. In this section we examine the effect of a single SDC sweep on the scalar linear model problem

$$y' = \lambda y, \tag{3.1}$$

$$y(0) = 1. \tag{3.2}$$

In the context of parabolic problems, the relevant set of λ is the negative real axis, and we are interested in how one sweep of SDC reduces the error in the discrete approximation.

In this case, the compact form of the SDC sweep from (2.23) becomes

$$(I - \lambda\Delta t Q_I)Y^{k+1} = Y_0 + \lambda\Delta t(Q - Q_I)Y^k. \tag{3.3}$$

The corresponding solution to the collocation equation satisfies

$$(I - \lambda\Delta t Q)Y = Y_0, \tag{3.4}$$

which implies

$$(I - \lambda\Delta t Q_I)Y = Y_0 + \lambda\Delta t(Q - Q_I)Y. \tag{3.5}$$

That is, the SDC sweep acts as the identity operator for the collocation solution Y.

Subtracting (3.5) from (3.3) gives

$$(I - \lambda\Delta t Q_I)(Y^{k+1} - Y) = \lambda\Delta t(Q - Q_I)(Y^k - Y) \tag{3.6}$$

or

$$Y^{k+1} - Y = (I - \lambda\Delta t Q_I)^{-1}\lambda\Delta t(Q - Q_I)(Y^k - Y). \tag{3.7}$$

Hence we can study the converge properties of SDC methods for the scalar model problem by examining the largest eigenvalue of the matrix,

$$(I - \lambda\Delta t Q_I)^{-1}\lambda\Delta t(Q - Q_I). \tag{3.8}$$

This matrix depends in general on the form of the substepping encapsulated in the approximate quadrature matrix (here backward Euler in Q_I), the number and type of quadrature nodes, and the product $\lambda\Delta t$. Here we examine the cases corresponding to the second- and fourth-order methods used in the numerical results in section 4. These correspond to uniform quadrature nodes and a quadrature rule which does not use the left-hand endpoint in the quadrature rule (see [22] for a discussion of different quadrature rules). These methods are used together in the fourth-order PFASST example as well, where the second-order method is the time coarsened version of the fourth-order method. This choice requires two implicit substeps for second-order and four for fourth-order and hence is not "spectral" in the sense of using Gaussian quadrature rules. This choice does, however, provide good damping factors for low-order methods.[1]

Figure 3 shows the maximum magnitude of the eigenvalues for both second- and fourth-order cases as a function of $\lambda\Delta t$. Clearly in both limits $\lambda\Delta t \to 0$ and $\lambda\Delta t \to -\infty$, the damping factor goes to zero. This behavior differs from the traditional analysis of point relaxation for elliptic equations like the Laplace equation, where high-frequency eigen-components are damped more rapidly than low-frequency components for the classical relaxation schemes. This feature is the reason multigrid methods provide a tremendous speedup compared to relaxation alone. In the context of SDC sweeps, Figure 3 demonstrates that although MLSDC with temporal coarsening may provide some increase in efficiency compared to higher-order SDC, SDC alone will still converge rapidly in the stiff limit for this problem.

For any linear problem, one can represent one iteration of PFASST or IPFASST as a matrix vector multiplication on a vector composed of the solution at each SDC node within each parallel time step. Examining the eigenvalues of this matrix would then give an indication of how PFASST iterations would converge for a given choice of parameters. One could then examine, for a given linear PDE, different numbers and types of nodes; types of sweeps and quadrature rules; number of levels, refinement factors in space and time, and type of interpolation and restriction between levels; and the size and number of parallel time-steps. In the case of IPFASST, the type of relaxation, the order of interpolation and restriction, and the number of V-cycles

[1]A recent paper by Weiser [35] studies the optimization of more general substepping rules for SDC methods. These ideas are not pursued here despite their apparent promise.

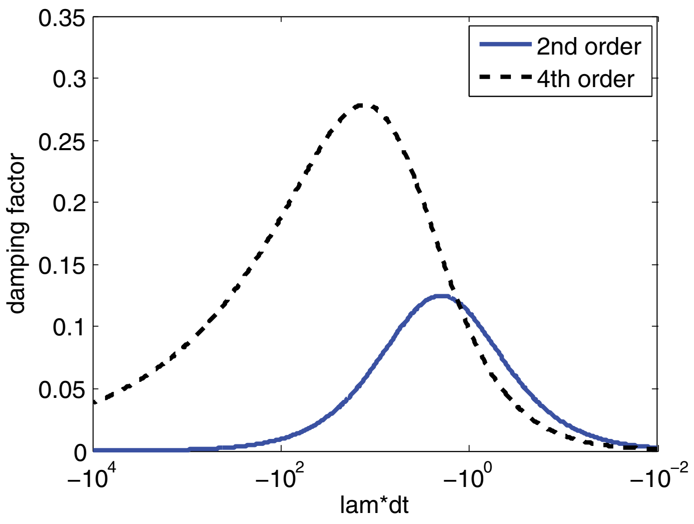

FIG. 3. *Damping factors for the second- and fourth-order SDC methods used in section* 4.

per implicit solve could also be varied.[2] A paper presenting a systematic study of the linear convergence of PFASST and IPFASST is in preparation.

4. Numerical results.

In this section, numerical results illustrating some features of the IPFASST method are presented on the model problem given by the linear heat equation. First, results analyzing the convergence behavior of IPFASST are reported. Strong scaling timing results for three-dimensional problems using space-time parallelism are then presented in section 4.2.

Despite the popularity of the heat equation as a test case for space-time parallel methods (e.g., [18, 19, 34, 14, 1, 13]), there are several reasons why it is not a particularly good test case for time parallelization. First, when considering time parallel methods, one would like to show how the method scales as the number of parallel time steps grows. Hence for many time steps, one must either choose a relatively long interval of integration or a relatively small time step. If a long time interval is chosen, the solution will decay to a value close to zero, which complicates any discussion of accuracy and convergence. If, on the other hand, very small time steps are chosen, then the temporal accuracy of the method will likely far exceed the spatial accuracy, which brings into question why such a small time step is being considered. Finally, it is important to note that the performance of parallel-in-time methods applied to the linear heat equation may not be indicative of performance on other problems that are not strictly parabolic, have dynamic spatial features, or have nontrivial boundary conditions. Despite these drawbacks, we will investigate the performance of IPFASST on this test case. The main motivation for doing so is to provide a comparison of the PFASST method with other published space-time parallel methods using this example.

4.1. One-dimensional convergence studies.

In this section we consider the accuracy and convergence of IPFASST, including the dependence on the number of V-cycles. For reasons addressed above, we use here the simple model problem consisting of the one-dimensional heat equation with homogeneous Dirichlet boundary

[2]It is also worth noting that the use of nonuniform substeps in SDC removes the equivalence between Fourier and eigenmode decomposition in the time direction.

conditions. The equation in a general form prescribed on the space-time domain $[0, L] \times [0, T]$ is given by

$$u_t = \nu u_{xx},$$
$$u(x, 0) = u_0(x),$$
$$u(0, t) = u(L, t) = 0.$$

Choosing the initial conditions

$$u_0(x) = \sin(k\pi x/L)$$

gives the exact solution

$$u(x, t) = e^{-\nu(k\pi/L)^2 t} u_0(x).$$

Using the method of lines and a second-order finite-difference approximation of the spatial derivatives gives the linear system of ODEs

$$u_i'(t) = \nu \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2}, \tag{4.1}$$

where $u_i(t) \approx u(i\Delta x, t)$ for $i = 1 \ldots N - 1$, $\Delta x = L/N$, and $u_0 = u_N = 0$. The exact solution of the systems of ODEs given initial conditions

$$u_i(0) = \sin(k\pi x_i/L)$$

is

$$u_i(t) = e^{-\nu d(k)t} u_0(x_i),$$

where

$$d(k) = \frac{-2 + 2\cos(k\pi\Delta x/L)}{\Delta x^2}.$$

Note that for k even moderately large, the solution decays very rapidly to zero. In the following one-dimensional examples, we choose $k = L = \nu = T = 1$, which means the solution decays to approximately 5×10^{-5} during the time interval.

One of the convenient features of SDC methods is that it is simple to construct higher-order methods simply by increasing the number of quadrature nodes being used. Even restricting the discussion to second-order spatial discretizations, using a first-order or second-order temporal integration method is very inefficient. As a simple demonstration of this, consider the following numerical experiment. We apply SDC methods of different orders to (4.1) with $k = \nu = L = T = 1$. Specifically we consider SDC methods of order 1, 2, 4, and 8, where the number of SDC nodes per time step corresponds to the formal order. We choose $\Delta x = 1/128$ and compute the L_∞ error of the solution at the final time $T = 1$ compared to the exact solution of the PDE and the exact solution of the discrete ODE for various values of $\Delta t = 1/N_t$ for $N_t = 2^p$ with p ranging from 2 to 12. In addition, we compute the residual at the final time step as in (2.25).

The results are displayed in Figure 4. Several comments can be made. Most obviously, as the order of the time integration increases, the number of time steps required to reduce the PDE error to the minimum possible given the spatial resolution

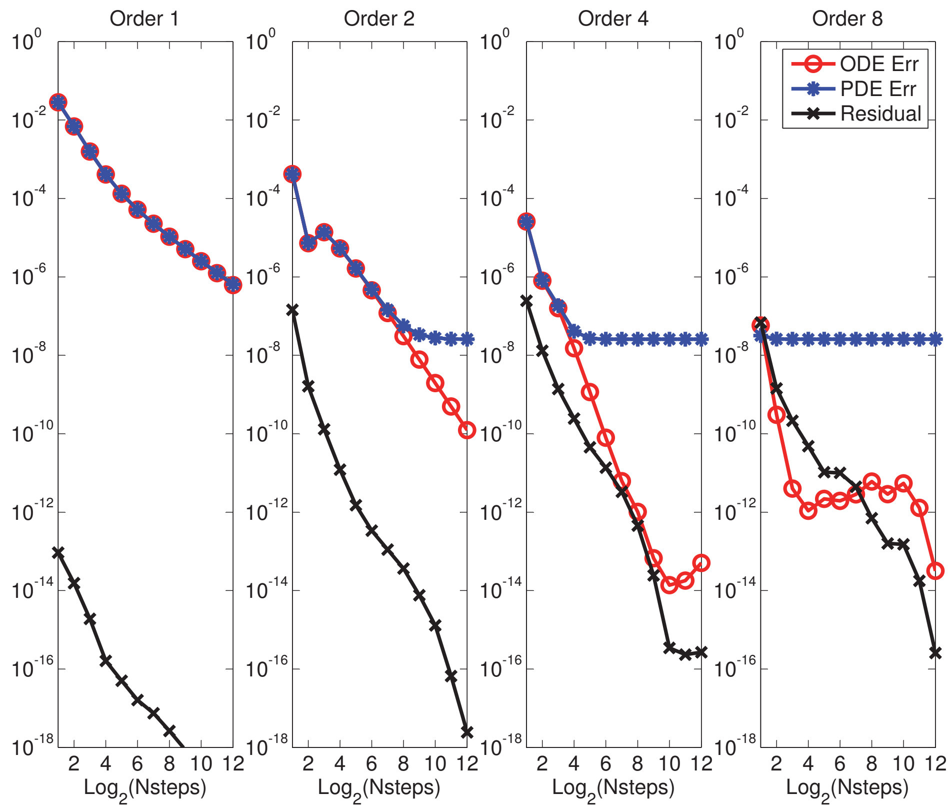

FIG. 4. *Convergence in error and residual for serial SDC methods of different orders for the one-dimensional heat equation.*

goes down dramatically. For the eighth-order method, two time steps of size 0.5 yields a smaller error than 128 time steps of the second-order method. Of course, higher-order methods require more work per time step due to the increase in substeps on the SDC nodes, but as shown in section 4.2 higher-order discretizations in space and time lead to much reduced computational times for problems in three dimensions. Note also that the residual and error in the discretized ODE continue to converge to zero well past the minimum error due to the spatial discretization. This implies that a judicious choice of residual tolerance is needed for iterative approaches like IPFASST to avoid wasted iterations.

In the PFASST algorithm, parallel efficiency depends on the availability of a hierarchy of space-time discretizations as in MLSDC, where coarsening in the temporal direction is achieved by reducing the number of nodes used in the underlying SDC method on each processor. Obviously for a second-order temporal discretization, only one coarsening step in time is possible leading to a coarse level based on the backward Euler method. In the spatial directions, one way to reduce the work on PFASST levels is to reduce the order of the spatial approximation on coarser levels; however, this is possible only when higher-order spatial approximation is being used on the finest level. Hence the options for coarsening in PFASST when restricted to second-order discretizations are more limited than for higher-order methods.

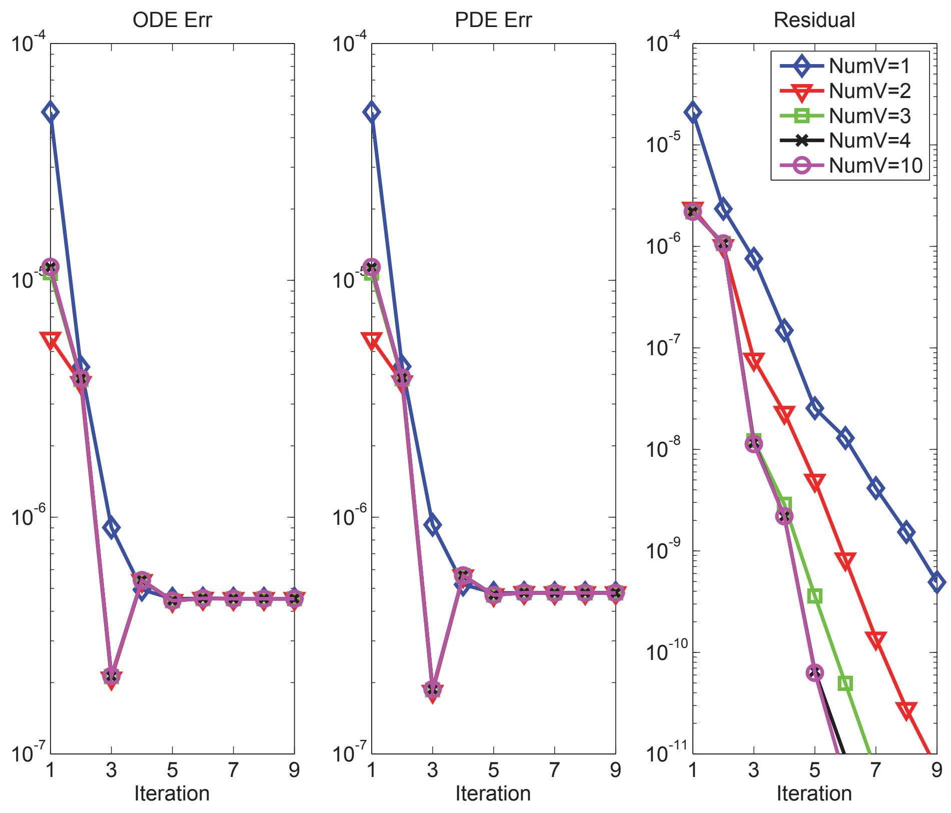

FIG. 5. *Comparison of the convergence of IPFASST using different numbers of V-cycles per solve.*

4.1.1. Comparison of IPFASST with different numbers of V-cycles. In this numerical experiment, we compare the convergence of IPFASST with different numbers of prescribed V-cycles for each approximate linear solve at each substep. A standard geometric multigrid method is used for these tests with linear interpolation, full-weighting restriction, and two pre- and two postsmoothing sweeps using a Jacobi smoother. IPFASST will only be more efficient than PFASST if the reduced cost of fewer V-cycles is not offset by an increase in the number of iterations required for convergence.

We use here $N_x = N_t = 128$ with 128 parallel time steps and the same parameters as in the previous examples, namely, $k = \nu = L = T = 1$. Figure 5 shows the convergence of IPFASST in terms of ODE error, PDE error, and residual versus iterations for different numbers of V-cycles used for the approximate implicit solve in every substep. The residual decays faster as the number of V-cycles is increased from one to three, but using more than three V-cycles does not yield further improvement. In contrast, both the ODE and the PDE error are less affected by the number of V-cycles. Using two V-cycles per solve gives virtually the same behavior as using 10. We stress again that these results for linear heat equation may not hold for PDEs of different mathematical type (see [8] for some preliminary results on advection diffusion equations).

4.1.2. Convergence of IPFASST with weak scaling. An important motivation for the study of time parallel methods is the "trap of weak scaling" given current supercomputer evolution where cores counts are increasing, but processor speed is not. For a given application, if the spatial resolution is increased as core counts increase, the cost of spatial operators will remain close to constant assuming good weak scaling of the spatial operations. However, the time step size will eventually need to also be reduced, either because of stability constraints or to match the increased spatial accuracy (depending on whether explicit or implicit methods are used). This means that more time steps are necessary for the same simulation time, and therefore, the total runtime will increase even with perfect spatial scaling unless some parallelism in the time direction is employed. For time parallel methods to be effective, it is necessary that the performance does not deteriorate as resolution is refined. Therefore, in this test we consider how the convergence of the IPFASST iterations scales with the problem size and number of parallel time steps.

Three runs are performed with $N_x = N_t = 32$, 64, and 128 again choosing $k = \nu = L = T = 1$. The number of parallel time steps in IPFASST is equal to the number of time steps N_t. We use three levels with coarsening by a factor two in space and second-order finite-difference approximations on all levels. On the coarsest level, the collocation rule corresponds to first-order backward Euler. On the middle and finest levels, we use three uniform nodes corresponding to a second-order collocation rule. Based on the previous experiments, implicit solves are approximated by two V-cycles with two pre- and postsmoothing steps with a Gauss–Seidel smoother.

Figure 6 shows how IPFASST converges for the different resolutions in terms of the ODE error (left), the PDE error (middle), and the residual (right). Errors are reported for the last time step. As the resolution increases in space or time, the error level decreases up to some minimum level. Depending on which discretization error is dominant, this level is either the discretization error of the collocation rule or of the spatial discretization; cf. the discussion in [31, section 3.2]. This level is reached at iteration 5 for $N = 32$ and at iteration 3 for $N = 64$ and $N = 128$. Therefore, increasing the resolution and with it the number of parallel time steps does not increase the number of iterations required by IPFASST. Note that this does not cover the case where the number of concurrently computed steps is increased while the time step size is kept constant in order to compute over a longer time interval.

4.2. Three-dimensional strong scaling studies. To compare the computational cost of IPFASST with serial methods we consider the three-dimensional heat equation

$$u_t(\mathbf{x}, t) = \nu \Delta u(\mathbf{x}, t), \quad \mathbf{x} \in \Omega, \quad 0 \le t \le T, \tag{4.2}$$

on the unit cube $\Omega = [0, 1]^3$ with $T = 1.0$, initial conditions

$$u(\mathbf{x}, 0) = \sin(\pi x) \sin(\pi y) \sin(\pi z), \tag{4.3}$$

and homogeneous Dirichlet boundary conditions. We choose $\nu = \frac{1}{3}$ so that the solution decays at the same rate as the previous one-dimensional tests. The Laplacian is discretized with either a second- or a fourth-order finite-difference stencil, and either ISDC or IPFASST is used to solve the resulting initial value problem. In all cases an implicit Euler substepping is used for the SDC sweeps, and PMG [3] is used for parallel multigrid V-cycles. Simulations are run on the IBM BlueGene/Q installation JUQUEEN at Jülich Supercomputing Centre. Timing results are displayed in Figure 7, and the specifications of the different runs are provided below.

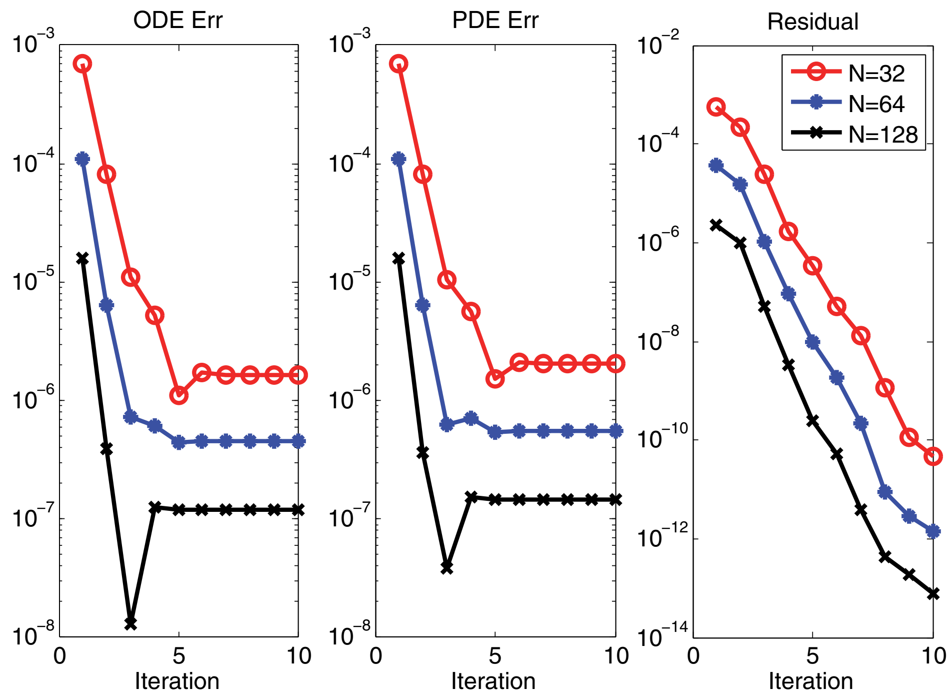

FIG. 6. *Convergence of IPFASST in terms of ODE error, PDE error, and residual in the final time step for increasing resolution in both space and time.*

4.2.1. Problem and method specifications. ISDC, IMLSDC, and IPFASST are run in three different configurations, which all give approximately identical errors measured in the infinity norm against the analytical solution of (4.2). The tolerance for the residual of ISDC and IPFASST is set to 10^{-9}, which in all three configurations results in an error of about 1.5×10^{-7}. PMG uses a tolerance of 10^{-12} whenever a full solve is performed and a stalling criterion that stops the iteration if the new residual is not smaller than 75% of the previous one. Two levels are used in IMLSDC and IPFASST, with fourth-order spatial interpolation and pointwise restriction in both space and time.

In all runs, approximate implicit solves consisting of two PMG V-cycles are used in the SDC sweeps. PMG V-cycles use two pre- and two postsmoothing sweeps consisting of JOR red-black smoothers. Linear interpolation and full-weighting restriction are used in the V-cycles.

Second-order spatial and second-order temporal discretization. For these runs, marked by circles in Figure 7, ISDC and the fine level of IMLSDC/IPFASST use two quadrature nodes corresponding to the midpoint of the time step and the right-hand value t_{n+1}. Therefore the method converges to a second-order collocation scheme. The coarse levels in IMLSDC and IPFASST correspond to backward Euler. The spatial mesh has $N = 128^3$ points and 128 time steps are performed. The coarse level uses a 64^3 point spatial mesh. For ISDC, runs are performed with the number of cores used by PMG varying between 16 and 32,768. For IMLSDC and IPFASST, the number of cores for PMG is fixed to 4,096 and the number of parallel time ranks varied between 1 (for IMLSDC) and 32, for a total of $4,096 \times 32 = 131{,}072$ cores. On average, ISDC requires about 3.5 iterations and IMLSDC about 3.7 for convergence. The last time-

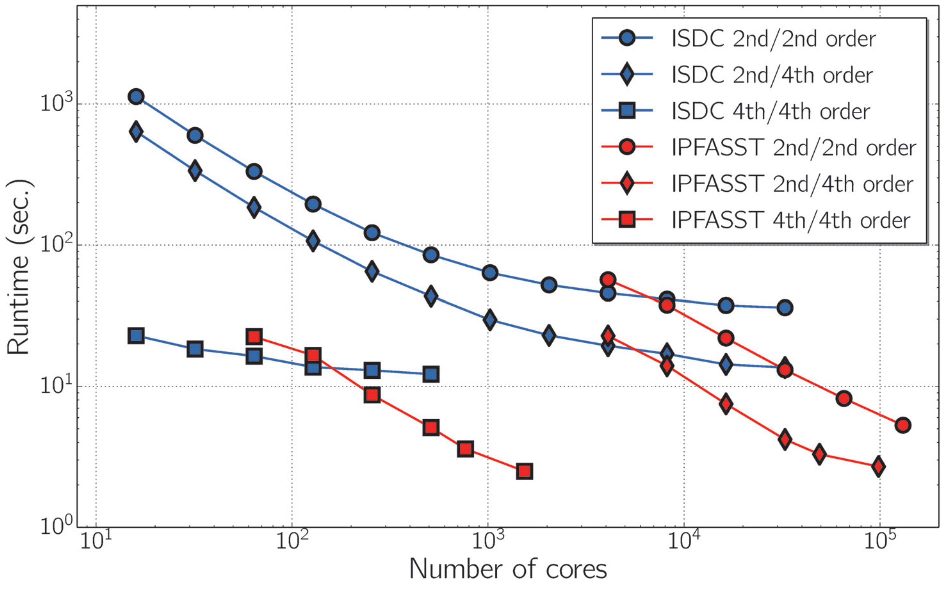

FIG. 7. *Runtimes for the three-dimensional heat equation depending on the total number of cores used for different configurations of ISDC and IPFASST. The first entry for each configuration of IPFASST corresponds to a run with only a single time-rank, that is, IMLSDC. All setups result in an error of about* 1.5×10^{-7}*, making a comparison of their runtimes meaningful.*

rank in IPFASST requires between about 3.4 (for two parallel time-ranks) and 4.0 iterations (for 32 time-ranks).

Second-order spatial and fourth-order temporal discretization. Marked by diamonds in Figure 7, ISDC and the fine level of IPFASST use four quadrature nodes, corresponding to fourth-order collocation. The coarse level of IPFASST is the same as the second-order runs above. Only 24 time steps are required to achieve the same overall error because of the higher-order temporal discretization. The spatial mesh remains as described above. As before, scaling of ISDC is measured using 16 up to 32,768 cores for PMG, while the number of cores for PMG in IMLSDC and IPFASST is fixed at 4,096. Because here only 24 time steps have to be performed, the number of time-ranks is varied only from 1 up to 24, for a total maximum number of cores of 24 × 4,096 = 98,304 cores. IMLSDC needs an average of 3.8 iterations, and the last time-rank in IPFASST between 4.0 (for two time-ranks) up to 6.0 (for 24 time-ranks).

Fourth-order spatial and fourth-order temporal discretization. As above, 24 time-steps are used with a fourth-order temporal discretization, marked by squares in Figure 7. $N = 32^3$ spatial points suffice to maintain the same error as above. On the coarse level, IPFASST uses only $N = 16^3$ points and a second-order finite-difference stencil. Here, because many fewer spatial degrees of freedom are needed, scaling of ISDC is measured only using 16 to 512 cores in PMG. IMLSDC and IPFASST use 64 cores for PMG and, as before, up to 24 time-ranks. The maximum total number of cores here is therefore only $24 \times 64 = 1,536$.

4.2.2. Results. Figure 7 displays the runtimes for the different configurations of ISDC and IPFASST laid out above. In addition, Table 1 lists parallel speedup and efficiency (speedup divided by number of processors) provided by IPFASST compared to the ISDC counterpart with the same number of spatial processors for PMG. Note

Table 1
Additional speedup provided by IPFASST compared to the ISDC run with the same number of spatial processors for PMG. N_p indicates the number of parallel time steps.

2nd-/2nd-order			2nd-/4th-order			4th-/4th-order		
N_p	Speedup	Efficiency	N_p	Speedup	Efficiency	N_p	Speedup	Efficiency
1	0.8	–	1	0.9	–	1	0.7	–
2	1.2	60.0 %	2	1.4	70.0 %	2	1.0	–
4	2.1	53.5 %	4	2.6	65.0 %	4	1.9	47.5 %
8	3.5	43.5 %	8	4.6	57.5 %	8	3.2	40.0 %
16	5.6	35.0 %	12	5.9	49.2 %	12	4.6	38.3 %
32	8.6	26.9 %	24	7.2	30.0 %	24	6.6	27.5 %

that the first marker for each IPFASST line corresponds to a run with only a single time-rank, which is IMLSDC. For each of the three setups, IMLSDC by itself is somewhat slower than ISDC using the same number of cores for PMG due to a slight increase in the number of iterations required by IMLSDC and the overhead of coarsening and interpolation between SDC levels. This is in contrast to the SDC/MLSDC method using full solves on the fine level as studied in [31], where MLSDC could significantly reduce runtimes compared to SDC. This means that the most efficient parallel variant for this test case (namely, IPFASST) is not a direct parallelization of the most efficient serial variant (namely, ISDC). Hence while the use of incomplete solves reduces the computational cost for both ISDC and IPFASST, it actually decreases the parallel efficiency of IPFASST because we have to compare to ISDC. We should also note that ISDC is not necessarily the most efficient serial method for this problem, but we use it as a comparison to the parallel performance of IPFASST based on ISDC sweeps.

As already observed in [30] for the PFASST method, it is more efficient to begin space-time parallelism with IPFASST using fewer spatial cores for PMG as for when the parallel speedup saturates. At the limit of spatial scaling, the spatial coarsening within PFASST does not make the coarse levels sweeps much cheaper than the fine level and therefore reduces the parallel efficiency of PFASST or IPFASST.

While the second-/second-order and the second-/fourth-order versions of ISDC and IPFASST scale approximately equally well, the higher-order methods result in shorter runtimes. For fully fourth-order ISDC, spatial scaling is of course significantly worse than for the second-order spatial methods because the size of the problem is drastically smaller. However, no matter how many processors are used for the second-order version, the fourth-order ISDC is always significantly faster. The same is true for IPFASST. While all three configurations of IPFASST scale approximately equally well, the fourth-order version requires significantly fewer cores to achieve the same runtime as the second-order version. The smallest time-to-solution is about 2.5 seconds, provided by fourth-order IPFASST using a total of 1,536 cores. While the mixed second-/fourth-order version is only slightly slower at 2.7 seconds, it requires 98,304 cores to achieve this runtime. We stress again that these results may not translate to other problems, particularly where higher-order methods are not appropriate due to a lack of smoothness in the solutions.

5. Discussion and outlook.

The two main goals for this paper are (1) to investigate combining spatial and temporal iterative strategies for space-time parallelization and (2) to provide some data on the performance of the resulting IPFASST method applied to the linear heat equation for comparison with other published results. The IPFASST algorithm introduced here interweaves the iterations of a spatial multigrid solver (PMG) with the temporal iterations in the SDC methods to achieve space-time parallelism, where the resulting hybrid space-time iterations are constructed by considering the spatial and temporal dimensions independently. Numerical results suggest that reducing the number of V-cycles for implicit spatial solves in IPFASST can be done without a significant impact on the convergence of the time parallel iterations (see, e.g., Figure 5). The scaling results shown in Figure 7 demonstrate that incorporating temporal parallelization as in IPFASST can extend strong scaling and further reduce the time-to-solution when spatial parallelism is close to saturation, assuming more resources are available.

One interesting result suggested by the numerical experiments is that although MLSDC might be more efficient than SDC for the heat equation, it seems that ISDC is more efficient than IMLSDC. The difference between ISDC and IMLSDC is essentially only the order in which SDC relaxation sweeps and multigrid V-cycles are performed on various levels, and hence it is likely that even more efficient variations using a more general ordering of space-time relaxations could be found. A careful analysis of different variations of the methods presented here, and the extension of the linear analysis in section 3 to PFASST and IPFASST is currently underway. As mentioned, given the number of different options already possible, finding optimal configurations may be difficult, and more importantly these optimal configurations are probably problem-dependent.

REFERENCES

[1] P. Amodio and L. Brugnano, *Parallel solution in time of ODEs: Some achievements and perspectives*, Appl. Numer. Math., 59 (2009), pp. 424–435.

[2] K. Böhmer, P. Hemker, and H. J. Stetter, *The defect correction approach*, in Defect Correction Methods. Theory and Applications, K. Böhmer and H. J. Stetter, eds., Springer-Verlag, Berlin, 1984, pp. 1–32.

[3] M. Bolten, *Evaluation of a multigrid solver for* 3*-level Toeplitz and circulant matrices on Blue Gene/Q*, in Proceedings of NIC Symposium 2014, K. Binder, G. Münster, and M. Kremer, eds., John von Neumann Institute for Computing, 2014, pp. 345–352.

[4] 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.

[5] E. L. Bouzarth and M. L. Minion, *A multirate time integrator for regularized Stokeslets*, J. Comput. Phys., 229 (2010), pp. 4208–4224.

[6] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, SIAM, Philadelphia, 2000.

[7] K. Burrage, *Parallel methods for ODEs*, Adv. Comput. Math., 7 (1997), pp. 1–3.

[8] Th. Dickopf, M. J. Gander, L. Halpern, R. Krause, and L. F. Pavarino, eds., *Domain Decomposition Methods in Science and Engineering* XXII, Lect. Notes Comput. Sci. Eng. 104, Springer International Publishing, Switzerland, 2015.

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

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

[11] 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, Lect. Notes Comput. Sci. Eng. 98, Springer, Berlin, 2014, pp. 359–366.

[12] R. D. Falgout, S. Friedhoff, Tz. V. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel time integration with multigrid*, SIAM J. Sci. Comput., 36 (2014), pp. C635–C661.

[13] S. Friedhoff, R. D. Falgout, T. V. Kolev, S. MacLachlan, and J. B. Schroder, *A multigrid-in-time algorithm for solving evolution equations in parallel*, presented at 16th Copper Mountain Conference on Multigrid Methods, Copper Mountain, CO, 2013.

[14] M. J. Gander and A. M. Stuart, *Space-time continuous analysis of waveform relaxation for the heat equation*, SIAM J. Sci. Comput., 19 (1998), pp. 2014–2031.

[15] A. Geist, *Paving the roadmap to exascale*, SciDAC Rev., 16 (2010), pp. 53–59.

[16] W. Hackbusch, *Parabolic multi-grid methods*, in Proceedings of Computing Methods in Applied Sciences and Engineering VI, 1984, pp. 189–197.

[17] T. Hagstrom and R. Zhou, *On the spectral deferred correction of splitting methods for initial value problems*, Commun. Appl. Math. Comput. Sci., 1 (2006), pp. 169–205.

[18] G. Horton and R. Knirsch, *A time-parallel multigrid-extrapolation method for parabolic partial differential equations*, Parallel Computing, 18 (1992), pp. 21–29.

[19] G. Horton and S. Vandewalle, *A space-time multigrid method for parabolic partial differential equations*, SIAM J. Sci. Comput., 16 (1995), pp. 848–864.

[20] J. Huang, J. Jia, and M. Minion, *Accelerating the convergence of spectral deferred correction methods*, J. Comput. Phys., 214 (2006), pp. 633–656.

[21] A. T. Layton and M. L. Minion, *Conservative multi-implicit spectral deferred correction methods for reacting gas dynamics*, J. Comput. Phys., 194 (2004), pp. 697–715.

[22] A. T. Layton and M. L. Minion, *Implications of the choice of quadrature nodes for Picard integral deferred corrections methods for ordinary differential equations*, BIT, 45 (2005), pp. 341–373.

[23] J.-L. Lions, Y. Maday, and G. Turinici, *A "parareal" in time discretization of PDE's*, C. R. Acad. Sci. Ser. I Math., 332 (2001), pp. 661–668.

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

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

[26] M. L. Minion and S. A. Williams, *Parareal and spectral deferred corrections*, in AIP Conf. Proc. 1048, American Institute of Physics, College Park, MD, 2008.

[27] J. Nievergelt, *Parallel methods for integrating ordinary differential equations*, Commun. ACM, 7 (1964), pp. 731–733.

[28] V. Pereyra, *On improving an approximate solution of a functional equation by deferred corrections*, Numer. Math., 8 (1966), pp. 376–391.

[29] D. Ruprecht, R. Speck, M. Emmett, M. Bolten, and R. Krause, *Poster: Extreme-scale space-time parallelism*, in Proceedings of the 2013 Conference on High Performance Computing Networking, Storage and Analysis Companion, 2013.

[30] R. Speck, D. Ruprecht, M. Emmett, M. Bolten, and R. Krause, *A space-time parallel solver for the three-dimensional heat equation*, in Parallel Computing: Accelerating Computational Science and Engineering (CSE), Adv. Parallel Comput. 25, IOS Press, Amsterdam, 2014, pp. 263–272.

[31] R. Speck, D. Ruprecht, M. Emmett, M. Minion, M. Bolten, and R. Krause, *A multi-level spectral deferred correction method*, BIT (2014), pp. 1–25.

[32] R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, and P. Gibbon, *A massively space-time parallel N-body solver*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Los Alamitos, CA, IEEE Computer Society Press, 2012, pp. 92:1–92:11.

[33] H. J. Stetter, *Economical global error estimation*, in Stiff Differential Systems, R. A. Willoughby, ed., Plenum Press, New York, 1974, pp. 245–258.

[34] S. Vandewalle and G. Horton, *Fourier mode analysis of the multigrid waveform relaxation and time-parallel multigrid methods*, Computing, 54 (1995), pp. 317–330.

[35] M. Weiser, *Faster SDC Convergence on Non-Equidistant Grids with DIRK Sweeps.* ZIB Report 13–30, 2013.

[36] P. E. Zadunaisky, *A method for the estimation of errors propagated in the numerical solution of a system of ordinary differential equations*, in The Theory of Orbits in the Solar System and in Stellar Systems, Vol. 25, Academic Press, London, 1966.