COMM. APP. MATH. AND COMP. SCI.
Vol. 7, No. 1, 2012

TOWARD AN EFFICIENT PARALLEL IN TIME METHOD FOR PARTIAL DIFFERENTIAL EQUATIONS

MATTHEW EMMETT AND MICHAEL L. MINION

A new method for the parallelization of numerical methods for partial differential equations (PDEs) in the temporal direction is presented. The method is iterative with each iteration consisting of deferred correction sweeps performed alternately on fine and coarse space-time discretizations. The coarse grid problems are formulated using a space-time analog of the full approximation scheme popular in multigrid methods for nonlinear equations. The current approach is intended to provide an additional avenue for parallelization for PDE simulations that are already saturated in the spatial dimensions. Numerical results and timings on PDEs in one, two, and three space dimensions demonstrate the potential for the approach to provide efficient parallelization in the temporal direction.

1. Introduction

The last decade has seen an increase in research into the parallelization of numerical methods for ordinary and partial differential equations in the temporal direction beginning with the introduction of the parareal algorithm in 2001 [20] and the related PITA scheme in 2003 [11]. Both parareal and PITA are iterative methods where, in each iteration, each processor (corresponding to distinct time steps) uses both an accurate (or fine) method and a less computationally expensive (or coarse) method to propagate an improved solution through the time domain. Parallel speedup can be achieved because the fine solutions can be computed in parallel.

The main drawback of the parareal algorithm is that it has low parallel efficiency. Specifically, the parallel efficiency is formally bounded above by $1/K$, where K

Emmett was supported by the Director, DOE Office of Science, Office of Advanced Scientific Computing Research, Office of Mathematics, Information, and Computational Sciences, Applied Mathematical Sciences Program, under contract DE-SC0004011. Minion's work was supported by the Alexander von Humboldt Foundation and the Director, DOE Office of Science, Office of Advanced Scientific Computing Research, Office of Mathematics, Information, and Computational Sciences, Applied Mathematical Sciences Program, under contract DE-SC0004011 and by the National Science Foundation under contract DMS-0854961.

MSC2010: 65M99.

Keywords: parallel computing, time parallel, ordinary differential equations, partial differential equations, deferred corrections, parareal.

is the number of iterations needed to converge to the desired accuracy. Since K must be at least 2 for any meaningful stopping criteria, the efficiency of parareal is always less than $\frac{1}{2}$, and in practice can be much worse, particularly if many processors are used, or high temporal accuracy is desired.

For the parallel solution of partial differential equations (PDEs), parallelization in the spatial dimensions is well established, and hence temporal parallelization is only attractive if the temporal parallel efficiency exceeds that of (additional) spatial parallelization. In [21; 24] a method for the parallelization of ordinary differential equations (ODEs) is presented, similar in structure to the parareal method but utilizing a defect or deferred correction strategy in place of standard methods for ODEs as in parareal. Both the fine and coarse propagators in parareal are cast as spectral deferred correction (SDC) [10] sweeps using different temporal resolutions to improve the solution on each time step. Hence the method described in [21; 24] can be heuristically thought of in two different ways:

(1) A modification of the parareal algorithm that replaces direct solves in each iteration with a deferred correction procedure applied to solutions generated at previous iterations to reduce the cost of each parareal iteration.

(2) A time parallel version of the SDC method that incorporates a coarse and fine temporal discretization to achieve better parallel efficiency.

In this paper, the ideas introduced in [21; 24] are extended to the temporal parallelization of partial differential equations (PDEs). The key difference between the ODE method and the PDE method is that coarsening can be done in both space and time in the coarse discretization. This observation has been made previously for both the parareal and hybrid parareal/SDC methods [3; 2; 12; 13; 24], although details of how best to translate information from the coarse and fine discretizations have not been extensively explored. Furthermore, for the parareal method, reducing the cost of the coarse propagator does not alter the fact that the parallel efficiency is bounded by $1/K$. Here we present a procedure for using coarse grid information based on the full approximation scheme (FAS) technique developed for the solution of nonlinear equations by multigrid methods [8] that increases the accuracy of the coarse grid SDC sweeps. Hence the method introduced here can be considered a *parallel full approximation scheme in space and time* (or PFASST for short).

The numerical techniques used in the construction of the PFASST method are reviewed in Section 2, and the details of how these techniques are synthesized to construct the PFASST algorithm are outlined in Section 3. The computational cost and theoretical parallel efficiency and speedup of the PFASST algorithm is discussed in Section 4, followed by numerical results confirming the convergence properties and efficiency in Section 5. Finally, a short discussion of the current results and future research directions can be found in Section 6.

2. Method components

In this section, the components used to construct the PFASST algorithm are reviewed. First, a short description of spectral deferred corrections is presented and the method is cast in a concise notational formulation used later on. Then a description of the parareal and hybrid parareal/SDC method from [21; 24] is provided. Finally a short review of the full approximation scheme is included.

2.1. ***Spectral deferred corrections.*** Spectral deferred correction (SDC) methods are variants of the traditional defect correction methods (or the closely related deferred correction methods) for ODEs introduced in the 1960s [30; 26; 27; 9; 29; 4]. SDC is introduced in [10], and the method has been modified and analyzed extensively since (see [22; 23; 19; 18; 17], for instance).

For the following description, consider the ODE initial value problem

$$u'(t) = f(t, u(t)), \qquad u(0) = u_0, \tag{1}$$

where $t \in [0, T]$; $u_0,\ u(t) \in \mathbb{C}^N$; and $f : \mathbb{R} \times \mathbb{C}^N \to \mathbb{C}^N$. In the numerical examples presented in Section 5, a method of lines discretization based on a pseudospectral approach is used to reduce the PDE in question to a large system of ODEs. To describe SDC, it is convenient to use the equivalent Picard integral form of (1):

$$u(t) = u_0 + \int_0^t f\big(\tau, u(\tau)\big)\, d\tau. \tag{2}$$

As with traditional deferred correction methods, a single time step $[t_n, t_{n+1}]$ is divided into a set of intermediate substeps by defining intermediate points $t_m \in [t_n, t_{n+1}]$. In SDC, the intermediate points t_m correspond to the nodes in Gaussian quadrature rules. Here Gauss–Lobatto rules are used so that $\boldsymbol{t} = [t_0, \dots, t_M]$ (with $t_n = t_0 < \cdots < t_M = t_{n+1}$) corresponds to the Gauss–Lobatto quadrature rule with $M + 1$ nodes (which has order $2M$).

SDC constructs higher-order accurate solutions within one full time step by iteratively approximating a series of correction equations at the intermediate nodes using lower-order methods. One attractive feature of SDC methods is that, since only lower-order methods are required, one can construct methods that employ operator splitting and/or multirate time-stepping and still achieve higher-order accuracy. (See [6; 22; 18; 7], for instance.)

The SDC method begins by computing a provisional solution $\boldsymbol{U}^0 = [U_1^0, \dots, U_M^0]$, at each of the intermediate nodes with $U_m^0 \approx u(t_m)$. As described below, this initial approximation can be simply the solution at the beginning of the time step replicated at each node. The method then proceeds iteratively. Let $\boldsymbol{U}^k = [U_1^k, \dots, U_M^k]$ denote the vector of solution values at each point $t_1 \dots t_M$ and SDC iteration k, and $\boldsymbol{F}^k = [f(t_0, U_0^k), \dots, f(t_M, U_M^k)]$ the vector of function values at each point

$t_0 \dots t_M$ and SDC iteration k. Note that $\boldsymbol{U}^k$ has M entries but $\boldsymbol{F}^k$ has $M+1$. To compute the approximations $\boldsymbol{U}^{k+1}$, one first computes the approximate integrals

$$\boldsymbol{S}_m^{m+1}\boldsymbol{F}^k = \sum_{j=0}^{M} q_{m,j} f(t_j, U_j^k) \approx \int_{t_m}^{t_{m+1}} f\big(\tau, U^k(\tau)\big)\, d\tau \tag{3}$$

for $m = 0 \dots M-1$. These approximations can be calculated by a matrix-vector multiplication by the $M \times M+1$ spectral integration matrix $\boldsymbol{S}$ with entries $q_{m,j}$. For a system of ODEs of size N, the integration matrix is applied component-wise, and hence $\boldsymbol{S}$ must be defined as the Kronecker product of the scalar integration matrix with the $N \times N$ identity matrix (see discussion in [14]).

Using these values, a first-order implicit time-stepping method similar to backward Euler for computing U^{k+1} at each substep can be written

$$U_{m+1}^{k+1} = U_m^{k+1} + \Delta t_m \big[f(t_{m+1}, U_{m+1}^{k+1}) - f(t_{m+1}, U_{m+1}^k)\big] + \boldsymbol{S}_m^{m+1}\boldsymbol{F}^k, \tag{4}$$

where $\Delta t_m = t_{m+1} - t_m$. The computational cost of each substep is essentially that of backward Euler (although if an iterative method is used to solve the implicit system, a very good initial guess is provided by the solution at iteration k). The process of solving (4) at each node t_m is referred to here as an *SDC sweep*.

The accuracy of the solution generated after k SDC sweeps done with such a first-order method is formally $O(\Delta t^k)$ as long as the spectral integration rule (here Gauss–Lobatto) is at least order k. In fact, if SDC converges, it converges to the solution of the spectral collocation or implicit Runge–Kutta method

$$\boldsymbol{U} = \boldsymbol{U}_0 + \Delta t\, \boldsymbol{S}\boldsymbol{F}, \tag{5}$$

where $\boldsymbol{U}_0 = [U_0, \dots, U_0]$. Hence SDC can be considered as an iterative method for solving the spectral collocation formulation. (See [14] for a technique to accelerate this convergence.)

For equations which can be split into stiff and nonstiff pieces, the above method is easily modified to create semi-implicit or IMEX schemes. Consider the ODE

$$u'(t) = f\big(t, u(t)\big) = f_E\big(t, u(t)\big) + f_I\big(t, u(t)\big), \qquad u(0) = u_0. \tag{6}$$

The first term on the right-hand side is assumed to be nonstiff (and hence treated explicitly), and the second is assumed to be stiff (and hence treated implicitly). Equation (4) can be easily modified to give a semi-implicit scheme:

$$\begin{aligned} U_{m+1}^{k+1} = U_m^{k+1} &+ \Delta t_m \big[f_I(t_{m+1}, U_{m+1}^{k+1}) - f_I(t_{m+1}, U_{m+1}^k)\big] \\ &+ \Delta t_m \big[f_E(t_m, U_m^{k+1}) - f_E(t_m, U_m^k)\big] + \boldsymbol{S}_m^{m+1}\boldsymbol{F}^k. \end{aligned} \tag{7}$$

Note that unlike semi-implicit or IMEX schemes based on Runge–Kutta (see [28; 1; 25; 15; 5], for instance), it is straightforward to construct semi-implicit

SDC schemes with very high formal order of accuracy. This semi-implicit form is used for all of the numerical examples presented in Section 5.

Following the ideas introduced in [14], we can write an SDC sweep more compactly by using matrix notation. Specifically, one can write all M steps of the forward Euler time-stepping scheme as

$$\boldsymbol{U} = \boldsymbol{U}_0 + \Delta t \boldsymbol{S}_E \boldsymbol{F}, \tag{8}$$

where the $M \times M + 1$ matrix $\boldsymbol{S}_E$ has entries

$$\boldsymbol{S}_E(i, j) = \begin{cases} \Delta t_j / \Delta t & \text{if } j \le i, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Likewise, all M steps of the backward Euler scheme can be written

$$\boldsymbol{U} = \boldsymbol{U}_0 + \Delta t \boldsymbol{S}_I \boldsymbol{F}, \tag{10}$$

where the $M \times M + 1$ matrix $\boldsymbol{S}_I$ has entries

$$\boldsymbol{S}_I(i, j) = \begin{cases} \Delta t_{j-1} / \Delta t & \text{if } 1 < j \le i + 1, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

The matrices $\boldsymbol{S}_E$ and $\boldsymbol{S}_I$ are also first order approximations to the integration matrix $\boldsymbol{S}$ that arise from using either the left-hand or right-hand rectangle rule approximation to the integrals S_m^{m+1} defined in (3).

Furthermore, let $\tilde{\boldsymbol{S}}_E = \boldsymbol{S} - \boldsymbol{S}_E$ and $\tilde{\boldsymbol{S}}_I = \boldsymbol{S} - \boldsymbol{S}_I$. Then, one SDC sweep using the semi-implicit time-stepping scheme in (7) can be compactly expressed as

$$\boldsymbol{U}^{k+1} = \boldsymbol{U}_0 + \Delta t \boldsymbol{S}_E \boldsymbol{F}^{k+1} + \Delta t \boldsymbol{S}_I \boldsymbol{F}^{k+1} + \Delta t (\tilde{\boldsymbol{S}}_E + \tilde{\boldsymbol{S}}_I) \boldsymbol{F}^k. \tag{12}$$

2.2. ***Parareal and SDC.*** The parareal method for the temporal parallelization of ODEs and PDEs was introduced in 2001 by Lions, Maday, and Turinici [20] and has sparked renewed interest in the construction of time parallel methods. In the parareal method, the time interval of interest $[0, T]$ is divided into N intervals with each interval being assigned to a different processor. On each interval $[t_n, t_{n+1}]$ for $n = 0 \dots N - 1$, the parareal method iteratively computes a succession of approximations $U_{n+1}^k \approx u(t_{n+1})$, where k denotes the iteration number.

The parareal algorithm can be described in terms of two numerical approximation methods typically denoted by $\mathcal{G}$ and $\mathcal{F}$. Both $\mathcal{G}$ and $\mathcal{F}$ propagate an initial value $U_n \approx u(t_n)$ by approximating the solution to (2) from t_n to t_{n+1} and can in principle be any self-starting ODE method. However, in order for the parareal method to be efficient, it must be the case that the $\mathcal{G}$ propagator is computationally less expensive than the $\mathcal{F}$ propagator, and hence $\mathcal{G}$ is typically a low-order method. Since the overall accuracy of parareal is limited by the accuracy of the $\mathcal{F}$ propagator,

$\mathcal{F}$ is typically higher-order and in addition may use a smaller time step than $\mathcal{G}$. For these reasons, $\mathcal{G}$ is referred to as the coarse propagator and $\mathcal{F}$ the fine propagator.

The parareal method begins by computing a first approximation in serial, U^0_{n+1} for $n = 0 \dots N-1$, often performed with the coarse propagator $\mathcal{G}$, i.e.,

$$U^0_{n+1} = \mathcal{G}(t_{n+1}, t_n, U^0_n) \tag{13}$$

with $U^0_0 = u(0)$. Alternatively, one could use the parareal method with a coarser time discretization to compute the initial approximation [2]. The parareal method proceeds iteratively, alternating between the parallel computation of $\mathcal{F}(t_{n+1}, t_n, U^k_n)$ and an update of the initial conditions at each processor of the form

$$U^{k+1}_{n+1} = \mathcal{G}(t_{n+1}, t_n, U^{k+1}_n) + \mathcal{F}(t_{n+1}, t_n, U^k_n) - \mathcal{G}(t_{n+1}, t_n, U^k_n) \tag{14}$$

for $n = 0 \dots N-1$. Although this step has serial dependencies, the computations on independent processors can be scheduled so that each processor can begin the computation of the new $\mathcal{G}$ value $\mathcal{G}(t_{n+1}, t_n, U^{k+1}_n)$ as soon as $\mathcal{F}(t_{n+1}, t_n, U^k_n)$ has been computed and the new starting value U^{k+1}_n has been received from processor $n-1$ [24]. The calculation in (14), which requires computing the $\mathcal{G}$ propagator, is referred to as here the $\mathcal{G}$ correction sweep.

Note that after k iterations of the parareal method, the solution U^k_m for $m \leq k$ is exactly equal to the numerical solution given by using the $\mathcal{F}$ propagator in a serial manner. Hence after N iterations the parareal solution is exactly equal to applying $\mathcal{F}$ in serial, but in practice the iterations converge more quickly for large N. Since each iteration of the parareal method requires the application of both $\mathcal{F}$ and $\mathcal{G}$ (plus the cost of communication between processors), the parareal method can only provide parallel speedup compared to the using $\mathcal{F}$ in serial if the number of iterations required to converge to the specified criteria (denoted here by K) is significantly less than N.

The dominant cost of the parareal method is the computation of $\mathcal{F}$ in each iteration. It has been well documented that the parallel efficiency of parareal is bounded by $1/K$ where K is the number of iterations needed to converge. In [21; 24] a hybrid parareal/SDC method is introduced that replaces the $\mathcal{F}$ propagator in parareal with a deferred correction sweep using the solution from the last iteration (and the new initial value U^{k+1}_n). This reduces the cost of the $\mathcal{F}$ propagator in two ways. First, instead of using many steps of a standard method like Runge–Kutta, the same level of accuracy can be achieved by one step of SDC due to the spectral accuracy of SDC methods. Second, the SDC sweep has a computational cost of approximately M steps of a first-order method, rather than many steps of a higher-order method. In effect, the high-cost of SDC *per time step* is amortized over the parallel iterations.

Furthermore, a connection between the parareal correction sweep defined by (14) and an SDC sweep is explained in [21; 24]. Hence the hybrid parareal/SDC method also casts the $\mathscr{G}$ correction sweep in (14) as an SDC sweep, which not only provides an updated starting value for the next processor, but can also be used to further improve the solution in the interval $[t_n, t_{n+1}]$. The numerical experiments in [24] suggest that the hybrid parareal/SDC strategy has similar convergence behavior as standard parareal, but with a reduced parallel cost and higher parallel efficiency. Specifically, the parallel efficiency (as compared to the serial SDC method) is bounded not by $1/K$, but K_s/K_p where K_s is the number of iterations required of the serial SDC method to converge to a given tolerance and K_p is the number of iterations for the parallel iterations to converge.

Note there are some disadvantages to the parareal/SDC hybrid approach. Because $\mathscr{F}$ and $\mathscr{G}$ in parareal are only used to provide solutions at the values t_n, parareal can be used in a "black-box" fashion with standard time integration methods. The gain in efficiency in the parareal/SDC approach requires that SDC be adopted as the time integration method, although there is still a great deal of flexibility in how the coarse and fine SDC sweeps are formulated (that is in fact one of the aforementioned advantages of SDC). Also, the parareal/SDC approach requires that both coarse and fine function values be stored at the intermediate nodes (however, the storage required is similar to that of higher-order Runge–Kutta methods).

2.3. ***Full approximation scheme.*** As mentioned above, when parallelizing PDEs in the temporal direction, an obvious way to reduce the cost of the coarse propagator in parareal or a hybrid parareal/SDC approach is to reduce both the temporal and spatial resolution. In the hybrid parareal/SDC method, it is desirable to use information from the coarse SDC sweep to not only improve the initial condition passed forward in time to the next processor, but also to improve the fine resolution solution on the same processor. To do so, the coarse resolution problem must be initialized including fine information, and the coarse resolution solution must be interpolated somehow (in both time and space) once computed. In the PFASST algorithm described in the next section, the coarse and fine resolution solutions are connected in the same manner as the full approximation scheme (FAS) method popular in multigrid methods for nonlinear problems (see [8], for instance). In fact, although only two resolutions are used here, the PFASST method could be extended to use a hierarchy of fine and coarse space-time grids, reminiscent of multigrid methods. Results along these lines will be reported in the future.

To review the FAS procedure, consider a nonlinear equation of the form

$$A(\boldsymbol{x}) = \boldsymbol{b}, \tag{15}$$

where the solution vector $\boldsymbol{x}$ and the right side $\boldsymbol{b}$ correspond to spatial discretizations of some function. Given an approximate solution $\tilde{\boldsymbol{x}}$, the corresponding residual equation is

$$A(\tilde{\boldsymbol{x}}+\boldsymbol{e}) = \boldsymbol{r} + A(\tilde{\boldsymbol{x}}), \tag{16}$$

where $\boldsymbol{e}$ is the error and $\boldsymbol{r} = \boldsymbol{b} - A(\tilde{\boldsymbol{x}})$ is the residual. In a multigrid approach, the residual equation (16) is solved on a coarser discretization level by introducing an operator T_F^G that restricts solutions at the fine resolution to the coarse. Then, assuming A^G is an appropriate approximation to A on the coarse level, the coarse residual equation becomes

$$A^G(\tilde{\boldsymbol{x}}^G + \boldsymbol{e}^G) = A^G(\tilde{\boldsymbol{x}}^G) + \boldsymbol{r}^G = A^G(\tilde{\boldsymbol{x}}^G) + T_F^G\big(\boldsymbol{b} - A(\tilde{\boldsymbol{x}})\big) \tag{17}$$

$$= \boldsymbol{b}^G + A^G(\tilde{\boldsymbol{x}}^G) - T_F^G A(\tilde{\boldsymbol{x}}), \tag{18}$$

where superscript G denotes the coarse level. With $\boldsymbol{y}^G = \tilde{\boldsymbol{x}}^G + \boldsymbol{e}^G$, the coarse FAS residual equation becomes

$$A^G(\boldsymbol{y}^G) = \boldsymbol{b}^G + \boldsymbol{\tau}, \tag{19}$$

with the FAS correction term

$$\boldsymbol{\tau} = A^G(\tilde{\boldsymbol{x}}^G) - T_F^G A(\tilde{\boldsymbol{x}}). \tag{20}$$

The addition of the FAS correction allows the coarse solution to attain a similar degree of accuracy as the fine solution, but at the resolution of the coarse level [8]. In particular, if the fine residual is zero (i.e., $\tilde{\boldsymbol{x}}$ is the fine solution), the FAS corrected coarse equation (19) becomes $A^G(\boldsymbol{y}^G) = A^G(\tilde{\boldsymbol{x}}^G)$, and the coarse solution $\boldsymbol{y}^G$ is the restriction of the fine solution.

Once the coarse solution $\boldsymbol{y}^G$ has been computed, the fine approximate solution is improved using an interpolation operator T_G^F

$$\tilde{\boldsymbol{x}} = T_G^F(\boldsymbol{y}^G - \tilde{\boldsymbol{x}}^G). \tag{21}$$

Returning to SDC methods, the FAS correction for coarse SDC iterations is determined by considering SDC as an iterative method for solving the collocation formulation given by (5), which can be written

$$\boldsymbol{U} - \Delta t\, \boldsymbol{S}\boldsymbol{F} = \boldsymbol{U}_0, \tag{22}$$

where $\boldsymbol{U}_0$, $\boldsymbol{S}$, and $\boldsymbol{F}$ are defined as in Section 2.1. Therefore, combining (20) and (22), the FAS correction for coarse SDC iterations is given by

$$\boldsymbol{\tau} = \Delta t\big(\boldsymbol{S}^G\boldsymbol{F}^G - T_F^G\boldsymbol{S}\boldsymbol{F}\big), \tag{23}$$

where $\boldsymbol{S}^G$ is the integration matrix defined by the coarse nodes, $\boldsymbol{F}^G$ is the vector of function values at the coarse level, and T_F^G is a space-time restriction operator. This allows the coarse SDC iterations to achieve the accuracy of the fine SDC iterations at the resolution of the coarse level, and ultimately allows the PFASST algorithm to achieve similar accuracy as a serial computation performed on the fine level. The numerical experiments in Section 5 confirm the benefit of using the FAS correction term in the coarse SDC sweep.

3. PFASST

At this point we have reviewed the main ingredients of the PFASST algorithm: SDC, time parallel iterations, and FAS. Now we describe how these ingredients are combined to form the PFASST algorithm. As in parareal, the time interval of interest $[0, T]$ is divided into N uniform intervals $[t_n, t_{n+1}]$ which are assigned to the processors $\boldsymbol{P}_n$ where $n = 0 \dots N-1$. Each interval is subdivided by defining $M+1$ fine SDC nodes $\boldsymbol{t}_n = [t_{n,0} \cdots t_{n,M}]$ such that $t_n = t_{n,0} < \cdots < t_{n,M} = t_{n+1}$; and $\tilde{M}+1$ coarse SDC nodes $\tilde{\boldsymbol{t}}_n$ such that $t_n = \tilde{t}_{n,0} < \cdots < \tilde{t}_{n,\tilde{M}} = t_{n+1}$. The coarse SDC nodes $\tilde{\boldsymbol{t}}_n$ are chosen to be a subset of the fine SDC nodes $\boldsymbol{t}_n$ to facilitate interpolation and restriction between the coarse and fine levels. The solution at the m-th fine node on processor $\boldsymbol{P}_n$ during iteration k is denoted $U^k_{n,m}$. Similarly, the solution at the $\tilde{m}$-th coarse node on processor $\boldsymbol{P}_n$ during iteration k is denoted $\tilde{U}^k_{n,\tilde{m}}$. For brevity let

$$\boldsymbol{U}^k_n = [U^k_{n,1}, \cdots, U^k_{n,M}] \quad \text{and} \quad \boldsymbol{F}^k_n = [f(t_{n,0}, U^k_{n,0}), \cdots, f(t_{n,M}, U^k_{n,M})],$$

with analogous notation for the coarse level (marked with a tilde). Note that the use of point injection as the coarsening procedure with Gaussian quadrature nodes means that the coarse nodes may not correspond to Gaussian nodes. This is further discussed at the beginning of Section 5.

3.1. ***Initialization.*** In the parareal method, the processors are typically initialized by using the coarse grid propagator in serial to yield a low-accuracy initial condition for each processor. This means in practice that all processors except the first are idle until passed an initial condition from the previous processor. Here we employ a different initialization scheme wherein each processor begins coarse SDC sweeps during this idle time. Hence the number of coarse iterations (SDC sweeps) done on processor $\boldsymbol{P}_n$ in the initialization is equal to n rather than 1. This has the same total computational cost of doing one SDC sweep per processor in serial, but the additional SDC sweeps can improve the accuracy of the solution significantly, as will be demonstrated in Section 5.

Specifically, the initial data $u(0)$ is spread to each processor $\boldsymbol{P}_n$ and stored in the fine level at each fine SDC node so that $U^0_{n,m} = u(0)$ for $n = 0 \dots N-1$ and $m = 0 \dots M$. Next, the fine values $\boldsymbol{U}^0_n$ are restricted to the coarse level and stored in $\tilde{\boldsymbol{U}}^{0,0}_n$, where the two superscripts denote PFASST iteration and initialization iteration, respectively. Before beginning the initialization iterations, the function values $\boldsymbol{F}^0_n$ and $\tilde{\boldsymbol{F}}^{0,0}_n$ are computed and used to form the first FAS correction $\boldsymbol{\tau}^0_n$.

The initialization iterations for $j = 1 \dots n$ on processor $\boldsymbol{P}_n$ are comprised of the following steps:

(1) Receive the new initial value $\tilde{U}_{n,0}^{0,j}$ from processor $\boldsymbol{P}_{n-1}$ if $n > 0$ and $j > 1$.

(2) Perform one or more coarse SDC sweeps using the values $\tilde{\boldsymbol{F}}_n^{0,j-1}$ computed previously and the FAS correction $\boldsymbol{\tau}_n^0$. This will yield updated values $\tilde{\boldsymbol{U}}_n^{0,j}$ and $\tilde{\boldsymbol{F}}_n^{0,j}$.

(3) Send $\tilde{U}_{n,\tilde{M}}^{0,j}$ to processor $\boldsymbol{P}_{n+1}$ (if $n < N-1$). This will be received as the new initial condition $\tilde{U}_{n+1,0}^{0,j+1}$ in the next iteration.

After processor $\boldsymbol{P}_n$ is finished computing the value $\tilde{U}_{n,\tilde{M}}^{0,n}$ and sending it to $\boldsymbol{P}_{n+1}$, the correction $\tilde{\boldsymbol{U}}_n^{0,n} - \tilde{\boldsymbol{U}}_n^{0,0}$ is interpolated to the fine grid to yield the initial value $\boldsymbol{U}_n^0$. The PFASST iterations on each processor are then begun immediately with this initial value.

3.2. ***PFASST iterations.*** The PFASST iterations for $k = 1 \dots K$ on each processor $\boldsymbol{P}_n$ proceed as follows. Assuming that the fine solution and function values $\boldsymbol{U}_n^{k-1}$ and $\boldsymbol{F}_n^{k-1}$ are available, the iterations are comprised of the following steps:

(1) Perform one fine SDC sweep using the values $\boldsymbol{F}_n^{k-1}$. This will yield provisional updated values $\boldsymbol{U}_n^{k'}$ and $\boldsymbol{F}_n^{k'}$.

(2) Restrict the fine values $\boldsymbol{U}_n^{k'}$ to the coarse nodes to form $\tilde{\boldsymbol{U}}_n^{k'}$ and compute $\tilde{\boldsymbol{F}}_n^{k'}$.

(3) Compute the FAS correction $\boldsymbol{\tau}_n^k$ using $\boldsymbol{F}_n^{k'}$ and $\tilde{\boldsymbol{F}}_n^{k'}$.

(4) Receive the new initial value $U_{n,0}^k$ from processor $\boldsymbol{P}_{n-1}$ if $n > 0$.

(5) Perform n_G coarse SDC sweeps beginning with the values $\tilde{\boldsymbol{F}}_n^{k'}$, the FAS correction $\boldsymbol{\tau}_n^k$, and the restriction of the new initial value $\tilde{U}_{n,0}^k$. This will yield new values $\tilde{\boldsymbol{U}}_n^k$ and $\tilde{\boldsymbol{F}}_n^k$.

(6) Interpolate the coarse correction $\tilde{U}_{n,\tilde{M}}^{k'} - \tilde{U}_{n,\tilde{M}}^{k}$ (in space only) and add to $U_{n,M}^{k'}$ to yield the updated value $U_{n,M}^k$.

(7) Send $U_{n,M}^k$ to processor $\boldsymbol{P}_{n+1}$ (if $n < N-1$). This will be received as the new initial condition $U_{n+1,0}^{k+1}$ in the next iteration.

(8) Interpolate the coarse grid correction $\tilde{\boldsymbol{U}}_n^{k'} - \tilde{\boldsymbol{U}}_n^k$ in space and time at the remaining fine time nodes ($0 < m < M$) and add to $\boldsymbol{U}_n^{k'}$ to yield $\boldsymbol{U}_n^k$. Recompute new values $\boldsymbol{F}_n^k$

The majority of the overhead associated with FAS is done in step 8 above, which is delayed until after the new initial condition is sent in step 7. This minimizes the amount of computation done between receiving a new initial condition from the previous processor (step 4) and sending the data forward (step 7). Pseudocode for the PFASST algorithm can be found in the Appendix.

4. Parallel speedup and efficiency

In this section, the theoretical parallel efficiency and speedup of the PFASST algorithm are examined. In the implementation used for the numerical results presented here, the PFASST method will converge to the collocation formulation (22) using one time step per processor. The number of substeps used in the fine SDC method is denoted again by M (which is the number of fine SDC nodes used minus one). Let η_F denote the cost of the method used for each substep of the fine SDC sweep. Likewise, let η_G and $\tilde{M}$ be the corresponding constants for the coarse SDC sweep. Further, define $\Upsilon_F = M\eta_F$ and $\Upsilon_G = \tilde{M}\eta_G$ to be the cost of one fine/coarse SDC sweep (assuming that the cost of computing the integration term used in the sweep is negligible). Let n_G denote the number of coarse SDC sweeps performed per PFASST iteration. To take into consideration the overhead of parallelization, we define γ_F and γ_G as the cost of sending a coarse and fine solution from one processor to the next. Finally, we define $\Upsilon_{\copyright}$ as the cost of the interpolation and restriction done in FAS.

If the PFASST iterations converge to the required accuracy in K_p iterations, the total cost on N processors is

$$C_p = Nn_G\Upsilon_G + (N-1)\gamma_G + K_p(\Upsilon_F + n_G\Upsilon_G + \Upsilon_{\copyright} + \gamma_F). \tag{24}$$

For simplicity, the communication costs γ_G and γ_F (which are relatively small in the numerical experiments in Section 5) are treated as overhead and are included in the $\Upsilon_{\copyright}$ term hereafter.

Let K_s denote the number of SDC iterations needed to compute the solution to the desired accuracy in serial using the fine SDC nodes. Then the cost of the serial SDC method will be approximately $C_s = NK_s\Upsilon_F$. Therefore, the parallel speedup S of the PFASST algorithm is

$$S = \frac{C_s}{C_p} = \frac{NK_s\Upsilon_F}{Nn_G\Upsilon_G + K_p(\Upsilon_F + n_G\Upsilon_G + \Upsilon_{\copyright})}. \tag{25}$$

Defining $\alpha = \Upsilon_G/\Upsilon_F$ and $\beta = \Upsilon_{\copyright}/\Upsilon_F$, (25) becomes

$$S = \frac{N}{\dfrac{Nn_G\alpha}{K_s} + \dfrac{K_p}{K_s}(1 + n_G\alpha + \beta)} \tag{26}$$

which gives a parallel efficiency of

$$E = \frac{1}{\dfrac{Nn_G\alpha}{K_s} + \dfrac{K_p}{K_s}(1 + n_G\alpha + \beta)}. \tag{27}$$

To achieve a parallel efficiency that is close to 1, the two quantities $Nn_G\alpha/K_s$ and K_p/K_s should be as small as possible. If the coarsening ratio between the coarse and fine grids is two in both time and space, and the implicit solves in the method have a cost proportional to the total number of grid points in the problem, then the cost ratio α between the coarse and fine SDC sweeps is approximately $1/2^{D+1}$, where D is the spatial dimension of the problem. This means that the $Nn_G\alpha/K_s$ term is well approximated by $Nn_G/(2^{D+1}K_s)$. Furthermore, since in each PFASST iteration both coarse and fine SDC sweeps are done, K_p/K_s can actually be less than one (as shown in Section 5). The numerical experiments also show that increasing the number of coarse SDC sweeps, n_G, reduces K_p, but at the cost of increasing all terms containing α. Finally, it should be noted that the cost of the overhead from the FAS procedure (and communication) signified by $\Upsilon_{\mathbb{C}}$ is not necessarily small. Since in the numerical tests presented here, both the evaluation of the function values and the interpolation in FAS are done with the FFT, β is in fact close to 1.

In contrast to the standard parareal method, note that the efficiency is not automatically bounded above by $E < 1/K_p$ but rather $E < K_s/K_p$. That is, by combining the SDC and parareal iterations into one hybrid parareal/SDC iteration, the bound on the parallel efficiency is relaxed by a factor of K_s when compared to the standard parareal method.

5. Numerical examples

In this section numerical results are presented to demonstrate the performance of the PFASST method for several PDEs of varying complexity. Since we are most interested in temporal errors, a pseudospectral discretization in space is used for all examples to minimize spatial errors. Also periodic boundary conditions are prescribed so that the discrete fast Fourier transform (FFT) can be used to evaluate the spectral derivatives and to interpolate in space. Temporal interpolation is done using standard polynomial interpolation from coarse nodes to fine. In both space and time, coarse grids are formed by taking every other fine point, so that the restriction operator is simply point-wise injection.

In the numerical tests that follow, the convergence of PFASST iterates is at times compared with the convergence of a serial SDC method. For the serial SDC method, the number of iterations reported refers to how many SDC sweeps are performed during each time step. This relates to the order of the method since each SDC sweep raises the formal order of accuracy by one, up to the accuracy of the underlying quadrature rule. In all cases, the errors reported are computed by comparing to a temporally resolved run on the fine grid (i.e., the solution of the discretized ODE and not the solution to the underlying PDE).

In all the examples below, the fine nodes in time correspond to either 9 or 5 Gauss–Lobatto nodes. Hence there are 5 or 3 coarse nodes respectively. When

9 nodes are used on the fine level, the 5 coarse nodes do not correspond to the Lobatto nodes, and hence the underlying quadrature rule is of order 6 instead of order 8. For 5 fine nodes, the 3 coarse nodes are the Lobatto nodes (Simpson's rule). When errors from serial SDC runs on coarse nodes are reported, the nodes used correspond to the coarse PFASST level and not the coarse Lobatto rule. One could instead interpolate the solutions in time to coarse Lobatto nodes or use Clenshaw–Curtis quadrature nodes at each level so that coarse and fine nodes correspond. Numerical experiments (not reported here) suggest the convergence of the PFASST iterations is not improved when using Clenshaw–Curtis nodes, and the accuracy of the fine solution is reduced. A more careful examination of the impact of different interpolation and restriction strategies in space and time is in preparation.

5.1. ***Viscous Burger's equation.*** In the first set of examples, the convergence of the PFASST iterates are examined on a simple one-dimensional equation, namely the viscous Burger's (VB) equation

$$u_t + uu_x = \nu u_{xx}, \tag{28}$$

where $\nu = 0.005$ is the diffusion constant. The VB equation (28) is split into explicit and implicit parts according to $f_E = -uu_x$ and $f_I = \nu u_{xx}$ in (6). That is, the nonlinear advection term is treated explicitly while the linear diffusion term is treated implicitly. The domain is the unit interval, and the initial conditions are

$$u_0(x) = e^{-(x-0.5)^2/\sigma}, \tag{29}$$

with $\sigma = 0.004$, so that the solution has a full spatial spectrum. The periodic images of the Gaussian are included in the initial condition to ensure it is spatially smooth. The spatial discretization is chosen so that the solution is resolved on the fine and coarse resolutions: 512 points are used on the fine grid and 256 on the coarse. The temporal discretizations are done with 5 Gauss–Lobatto SDC nodes on the fine level, and 3 Gauss–Lobatto SDC nodes on the coarse level. Analysis of the performance of PFASST when the coarse level is not well-resolved is ongoing and will be presented elsewhere.

For the VB test, 64 processors are used with the time step on each processor being $\Delta t = 0.08/64$. Note that the real part of the quantity $-\Delta t \nu (2\pi k_{\max})^2$, where $k_{\max}$ is the largest Fourier wave number, is approximately -15.16 on the fine grid. Hence the use of a semi-implicit method (as opposed to a fully explicit method) avoids a substantial time step restriction.

Figure 1 shows the convergence of the PFASST algorithm and the serial SDC method for the VB test. The error is computed for each SDC and PFASST iteration at the end of each time step, and therefore the horizontal axis corresponds to the time step, which in turn corresponds to the processor number in the PFASST case.

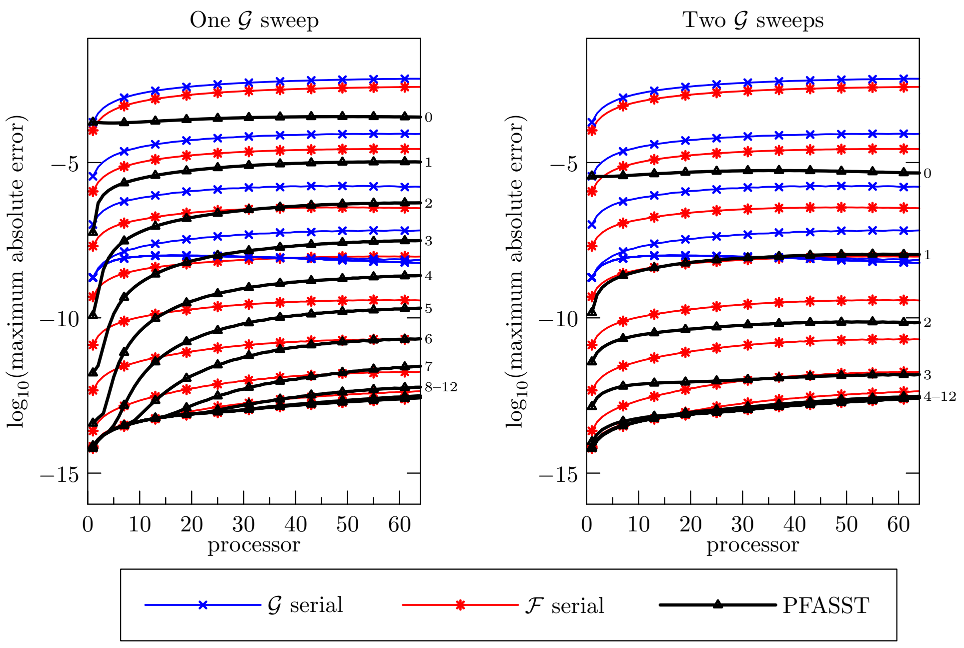

Figure 1. Maximum absolute error versus time (processor) for several PFASST iterations applied to the VB equation. The left and right panels correspond to one and two coarse SDC sweeps per PFASST iteration respectively. Each PFASST line represents the error at the end of the corresponding PFASST iteration, with 0 representing the solution after initialization. The $\mathscr{G}$ and $\mathscr{F}$ serial lines represent the error of serial SDC runs on the coarse and fine space-time discretizations with varying numbers of SDC sweeps (iterations) per time step.

Again, keep in mind that the number of iterations reported for the serial SDC runs refers to how many SDC sweeps are performed during each time step of the serial method. The PFASST algorithm is run using $n_G = 1$ (left panel) and $n_G = 2$ (right panel) coarse SDC sweeps per iteration. Several observations can be made from the data.

Concerning the convergence of the serial SDC method, note first that the minimum error is reached after 4 SDC iterations for the coarse grid, which is in agreement with the formal fourth-order accuracy expected with 3 Gauss–Lobatto nodes. Similarly, the fine serial SDC method very nearly attains minimum error after 8 iterations, which again is consistent with the formal eighth-order accuracy. Note that the convergence of SDC to the collocation solution can be slower for very stiff problems [14], as is the case for the examples in Section 5.2. Because of the spectral accuracy of the SDC method, the fine solution with 5 Gauss–Lobatto nodes is substantially more accurate than the coarse.

Turning to the convergence of the PFASST algorithm, the first thing to note is that the PFASST iterates do converge to the converged SDC solution on the fine grid. Next, note that using two coarse SDC sweeps per PFASST iteration in this example reduces the total number of iterations needed to obtain a highly accurate solution from 9 iterations to 4 iterations, albeit at a higher cost per iteration. This behavior is similar to the parareal algorithm in that the overall speed of convergence of the algorithm depends on the accuracy of the coarse propagator [24]. Although not shown here, it has been observed that using multiple fine SDC sweeps per iteration does not have a significant effect on the rate of convergence of the PFASST algorithm for this example.

Note the effect of the iterative initialization procedure described in Section 3.1. The error after initialization (labeled 0 in each panel) is significantly less than that produced using the corresponding number of serial coarse SDC sweeps (one in the left panel and two in the right) at later times. Of course for the first processor, the initialization is exactly equivalent to the first step of a serial SDC method.

Figure 2 compares the convergence of the PFASST algorithm and serial SDC runs by considering error versus iteration computed at the final time. Additional

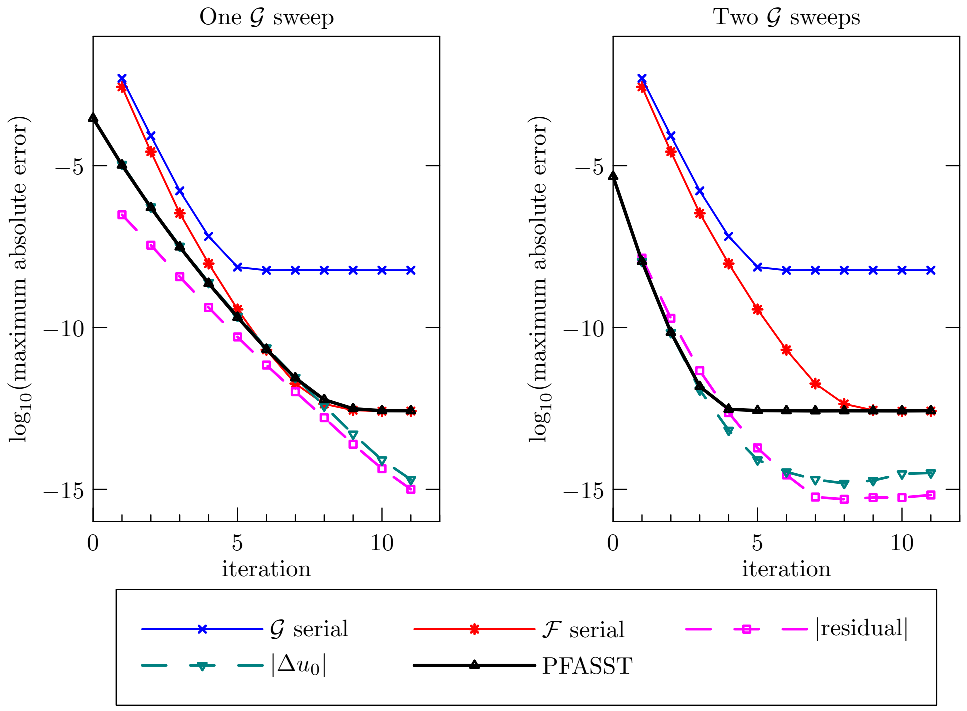

Figure 2. Maximum absolute error, residual, and change in fine initial condition computed at the final time interval for several PFASST iterations applied to the VB equation. The left and right panels correspond respectively to one and two coarse SDC sweeps per coarse PFASST iteration. The $\mathcal{G}$ and $\mathcal{F}$ serial lines represent the error of serial SDC runs on the coarse and fine space-time discretizations with varying numbers of SDC sweeps (iterations) per time step.

data corresponding to the residual and the maximum change in the initial condition at each processor is also included. It is again apparent that the PFASST algorithm achieves the accuracy of the fine serial run despite the relatively poor temporal accuracy of the coarse resolution. Also, the convergence of the PFASST algorithm is accelerated by using two coarse SDC sweeps per iteration. Since the problem is well-resolved at the finest level, the residual and change in initial condition are good indicators of the error at each iteration, until the accuracy of the time integration scheme is reached. Using the residual and change in initial condition to adaptively control the number of PFASST iterations performed will be explored elsewhere.

To demonstrate the importance of including the FAS correction term in the coarse sweeps, the above test was rerun omitting the FAS correction term defined in (23). Figure 3 shows the convergence of the algorithm without FAS corrections. It is clear that when FAS corrections are not included, the algorithm can only achieve accuracy comparable to the coarse level.

Finally, Figure 4 shows how the performance of the PFASST algorithm depends on the numbers of processors for the VB example. The left panel shows the convergence results for four different numbers of processors with the final time fixed (so

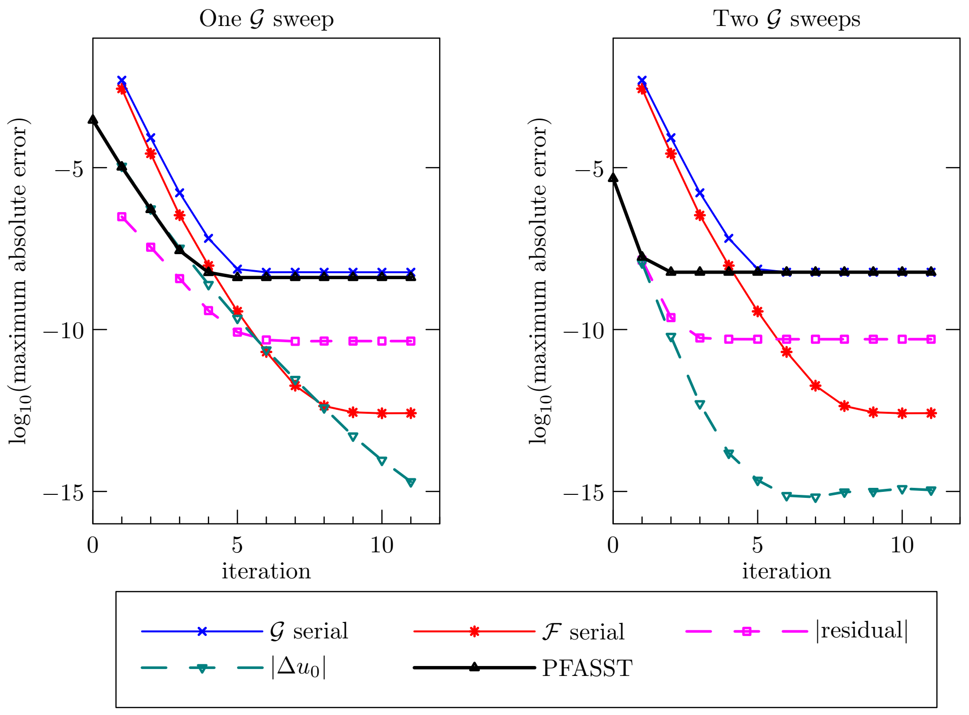

Figure 3. Maximum absolute error, residual, and change in fine initial condition computed at the final time for several PFASST iterations for the VB equation without FAS corrections. This figure should be compared to Figure 2.

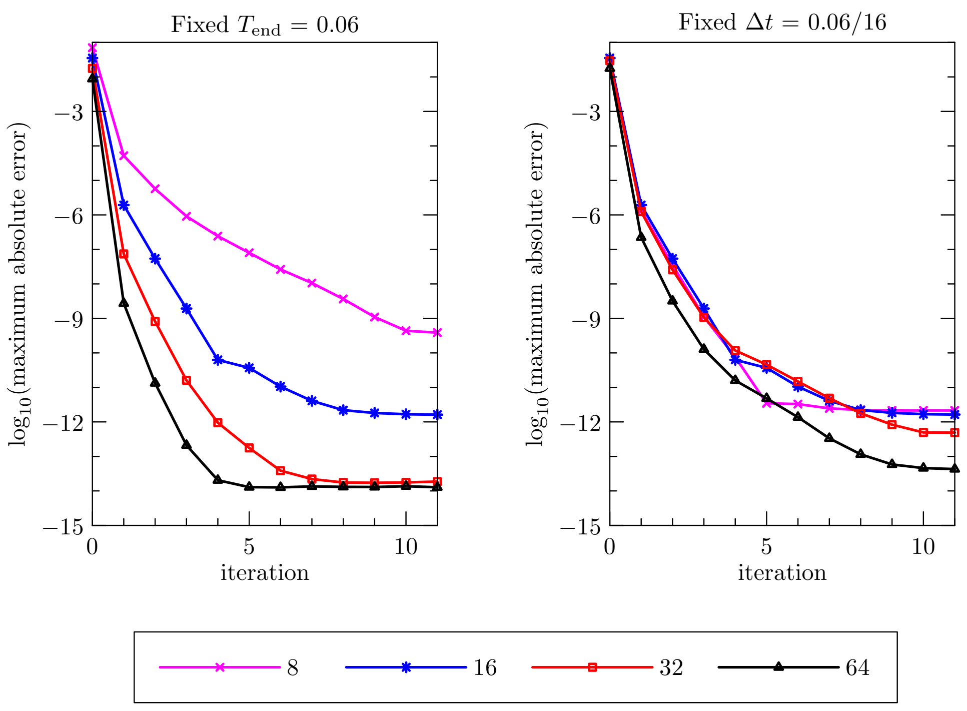

Figure 4. Maximum absolute error computed at the final time for several PFASST iterations and number of processors for the VB equation. The left panel shows PFASST errors at the final time for a fixed final time. The right panel shows PFASST errors at the final time for a fixed time step.

that the time step decreases as the number of processors increases). The right panel shows the convergence for the same numbers of processors with a fixed time step (so that the final integration time increases as the number of processors increases).

When the final time is fixed (left panel) the convergence of the PFASST algorithm improves as N increases in that the number of PFASST iterations required to achieve a given level of accuracy decreases. When the time step is fixed (right panel) the convergence is similar for each run despite the difference in simulation time. In summary, for this example the convergence depends largely on the time step used, not on the number of processors.

5.2. ***The Kuramoto–Silvashinsky equation.*** Next, the performance of the PFASST algorithm is explored for the Kuramoto–Silvashinsky equation

$$u_t + \tfrac{1}{2}|\nabla u|^2 + \nabla^2 u + \nabla^4 u = 0, \tag{30}$$

where the solution u is a function of two space variables and time. The KS equation arises as a model for interfacial instabilities in a variety of physical contexts and has been shown to exhibit nontrivial dynamical behavior, both spatially and temporally, including chaos [16]. It contains a nonlinear term and high-order derivatives which, from a numerical perspective, make it a challenging equation to solve as it is nonlinear, very stiff, and highly sensitive to changes in the initial conditions or numerical error.

The KS equation (30) is split into explicit and implicit parts according to $f_E = -\frac{1}{2}|\nabla u|^2$ and $f_I = -\nabla^2 u - \nabla^4 u$ in (6). That is, the nonlinear term is treated explicitly while the linear antidiffusion and hyper-diffusion terms are treated implicitly. As with the VB equation, periodic boundary conditions are used, all spatial operators are evaluated spectrally, and the computational grid is chosen so that the solution is fairly well resolved on the fine grid. The domain size used throughout is a two-dimensional square domain with sides of length $L = 100.0$, with 512 points in each dimension on the fine grid and 256 on the coarse. The initial condition used throughout is shown in Figure 5. This initial condition was obtained by running the KS equation from a simple initial condition with only three Fourier modes to a final time of approximately 97, and subsequently removing high frequency modes (magnitude of wave-number greater than 121). This produces an initial condition with a broad spectrum of Fourier modes but without any fast initial transients. The temporal discretization uses 9 Gauss–Lobatto SDC nodes on the fine level, and 5 nodes on the coarse level. Note that the coarse nodes do not correspond to the 5-point Gauss–Lobatto rule.

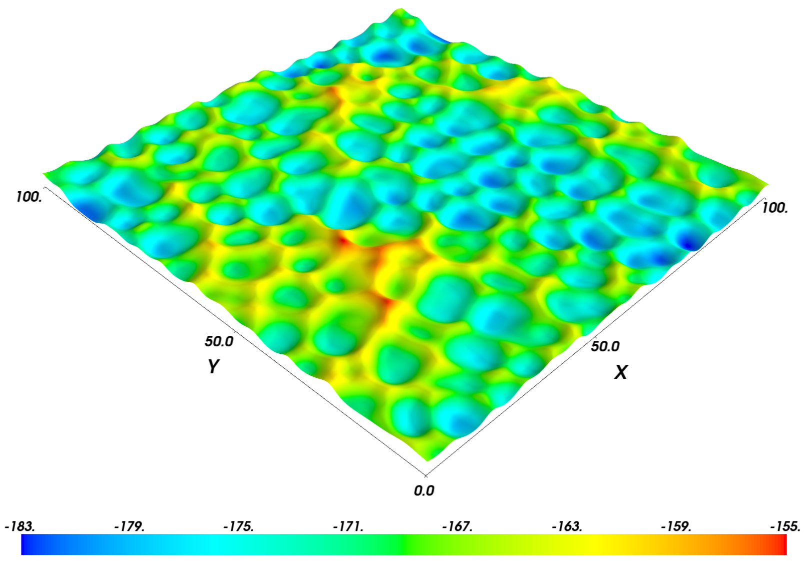

Figure 5. Initial condition for the KS and AD equation examples.

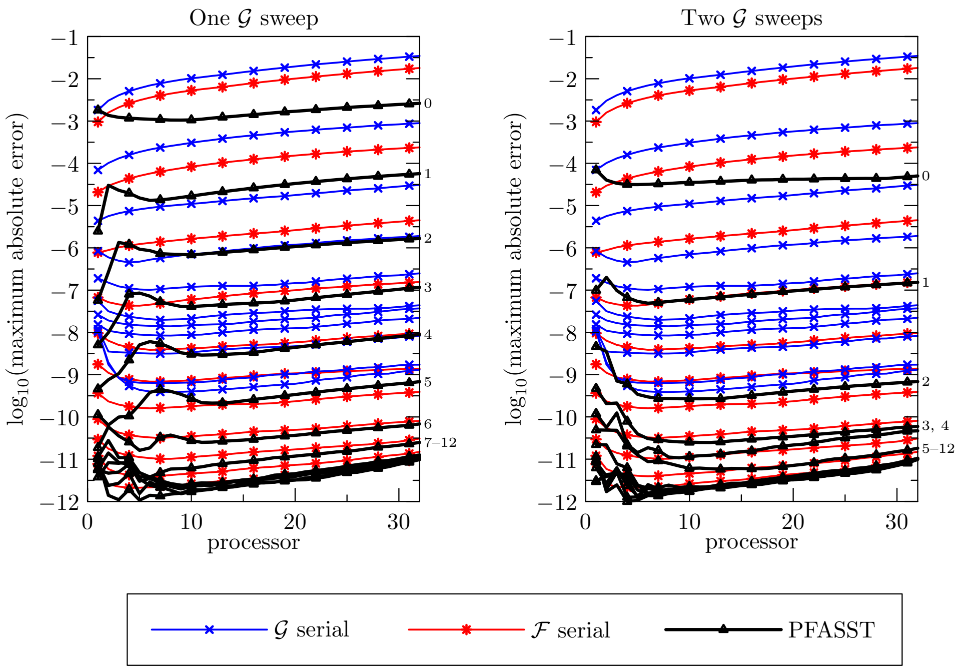

Figure 6. Maximum absolute error versus time (processor) for several PFASST iterations applied to the KS equation. The left and right panels show PFASST errors for one and two coarse SDC sweeps per PFASST iteration respectively. Each PFASST line represents the error at the end of the corresponding PFASST iteration, with 0 representing the solution after initialization. The $\mathcal{G}$ and $\mathcal{F}$ serial lines represent the error of serial SDC runs on the coarse and fine space-time discretizations with varying numbers of SDC sweeps (iterations) per time step.

For the KS test, 32 processors are used with the time step on each being $\Delta t = 1.0/32$. Figures 6 and 7 compare the convergence of the PFASST algorithm and serial SDC runs by considering the error versus processor (or simulation time) for each PFASST iteration, and the error versus iteration at the final time, respectively.

The results for KS shown in Figures 6 and 7 are qualitatively the same as those for VB shown in Figures 1 and 2. Again one can note the benefit of the iterative initialization procedure for the PFASST algorithm in that the error after initialization is considerably lower than a serial SDC method with the corresponding number of sweeps (except for at the first processor where they are identical). Note also that the PFASST method converges faster when using $n_G = 2$ coarse SDC sweeps per iteration instead of one, and this can improve the parallel efficiency (as discussed in Section 5.3).

For the KS equation, the minimum error achieved by both the serial and parallel methods is higher than in the VB case despite the use of 9 fine nodes for the KS equation instead of 5 for VB. To highlight the difficulty inherent in the parallel

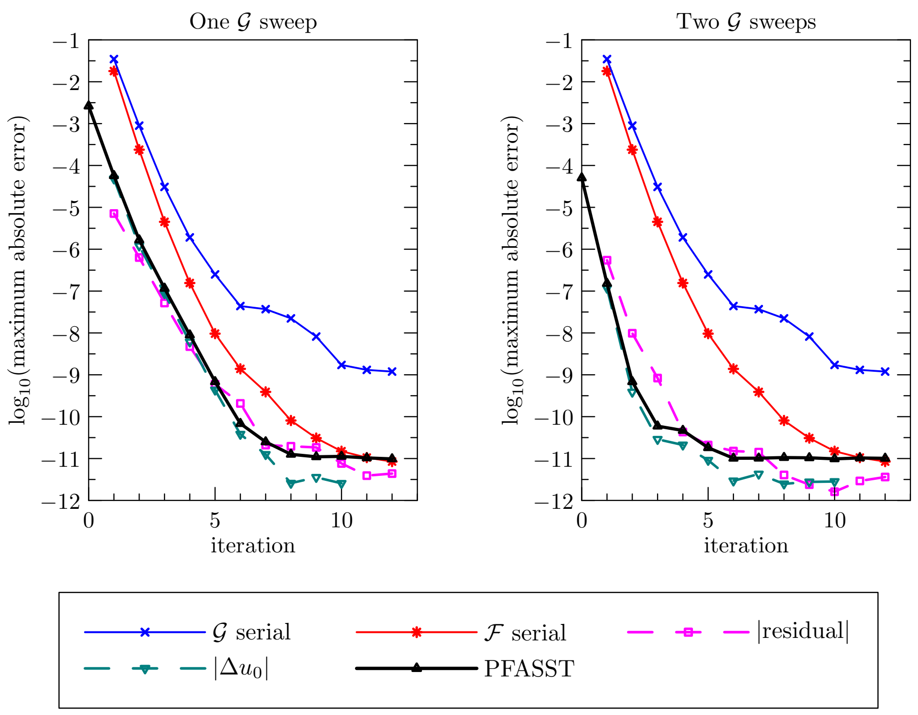

Figure 7. Maximum absolute error, residual, and change in fine initial condition computed at the final time for PFASST iterations applied to the KS equation. The left and right panels correspond to one and two coarse SDC sweeps per PFASST iteration respectively. The $\mathcal{G}$ and $\mathcal{F}$ serial lines represent the error of serial SDC runs on the coarse and fine space-time discretizations with varying numbers of SDC sweeps (iterations) per time step.

numerical approximations of the KS equation, a final numerical experiment is performed using the same initial conditions as the KS example (Figure 5), but with a simpler linear advection-diffusion equation

$$u_t + \nabla \cdot u = \nu \nabla^2 u \tag{31}$$

with $\nu = 0.02$. The domain size, spatial discretization, and temporal discretization are the same as in the KS example. Figure 8 shows the convergence of the PFASST algorithm for this linear advection/diffusion (AD) equation. It is clear that PFASST converges much faster than in the KS example, specifically in four and two iterations for one and two coarse SDC sweeps per iteration respectively.

5.3. ***Parallel timing results.*** The results in Section 5.2 show that the PFASST algorithm exhibits reasonable convergence behavior for a selection of PDEs in simple geometries. In this section, the parallel speedup and efficiency of the PFASST algorithm are explored. The PFASST algorithm has been implemented in F90 using MPI for communication between processors. The timing results correspond

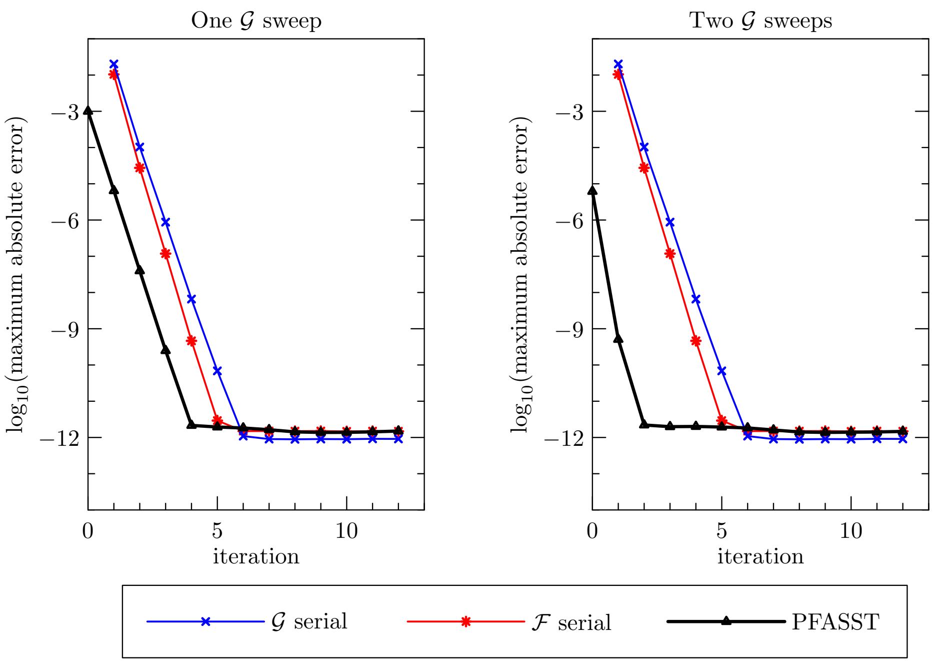

Figure 8. Maximum absolute error versus iteration at the final time for PFASST iterations applied to the AD equation. The left and right panels correspond to one and two coarse SDC sweeps per PFASST iteration respectively.

to numerical experiments performed using 8 and 16 processors on a multicore UNIX machine with 2×8 2GHz AMD Opteron cores, and run times reported were computed using the MPI `wtime` command. Preliminary timings have also been performed on a distributed memory cluster using up to 512 cores. Initial results suggest that communication costs across interconnects do not significantly impact the efficiency of the PFASST method. A more thorough analysis of these costs will be presented elsewhere.

For the following tests, the PFASST method is applied to a scalar VB equation (28) in three dimensions

$$u_t + u\nabla \cdot u = \nu \nabla^2 u, \tag{32}$$

hereafter referred to as the 3d NAD equation. The spatial resolution is 128 points in each dimension on the fine grid and 64 on the coarse. The initial condition used was a periodic image of

$$u_0(x) = (4\pi\nu)^{-3/2} e^{-(x-0.5)^2/(4\pi\nu)}. \tag{33}$$

The parameter values used were $\nu = 0.02$ and $T_{\text{end}} = 0.002$. The temporal discretizations are done with 5 Gauss–Lobatto SDC nodes on the fine level, and 3 Gauss–Lobatto SDC nodes on the coarse level.

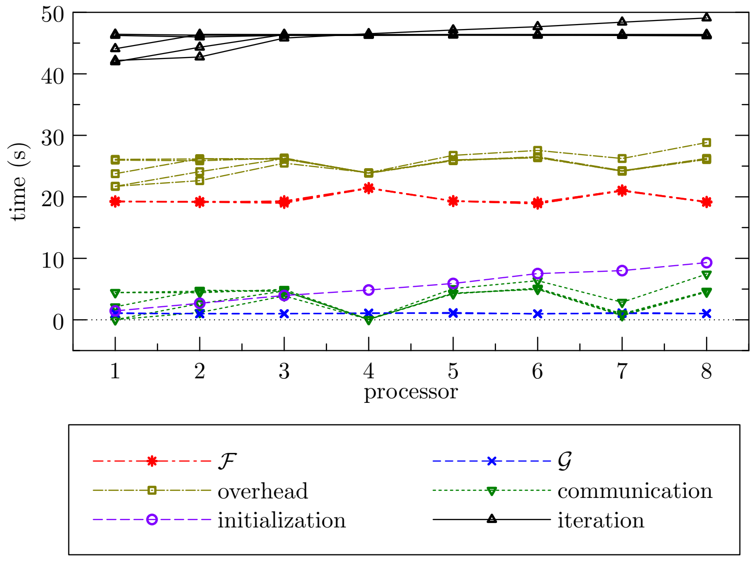

Figure 9. PFASST iteration timings for the third NAD equation on 8 processors. For all series except the predictor, each line corresponds to a different PFASST iteration from 1 to 8.

Figure 9 shows a timing breakdown of PFASST iterations for the 3d NAD equation using 8 processors. As expected, the predictor time grows linearly with the processor number. This "burn-in" time is unavoidable, but the slope could be reduced by introducing even coarser levels in the iterative initialization procedure. The ratio of the cost of fine to coarse SDC sweeps (denoted by α in Section 4) is approximately 16, which is consistent with a factor of two coarsening in both time and space. The communication cost associated with passing the fine solution forward in time from $\boldsymbol{P}_n$ to $\boldsymbol{P}_{n+1}$ is greater than the cost of a coarse SDC sweep, but less than that of a fine SDC sweep. Finally, the overhead of interpolation and restriction performed to compute the FAS correction is slightly more costly than a fine SDC sweep (i.e., $\beta > 1$). This is because the FFT is used for both spatial interpolation and the "explicit" computation of the nonlinear advective terms.

Table 1 shows the parallel speedup and efficiency of the PFASST algorithm for the 3d NAD equation. The number of iterations used for each method was chosen so that the accuracy of the solution at the final time was consistent between the runs and approximately equal to 10^{-13}. The number of PFASST iterations required is less than the number of serial SDC iterations required since, during each PFASST iteration, at least two SDC sweeps are performed (one on the fine level and one or more on the coarse level). That is, K_p/K_s is less than one (see (26)). The parallel speedup and efficiency achieved are well predicted by the theoretical formulas (26) and (27).

method	processors	iterations	time	speedup	efficiency
Serial SDC	$\Delta t = T_{\text{end}}/8$	7 SDC	1115.34s		
PFASST, one $\mathcal{G}$ sweep	8	5 PFASST	310.47s	3.59	0.45
PFASST, two $\mathcal{G}$ sweeps	8	3 PFASST	212.48s	5.25	0.66
Serial SDC	$\Delta t = T_{\text{end}}/16$	5 SDC	1606.69s		
PFASST, one $\mathcal{G}$ sweep	16	4 PFASST	216.20s	7.43	0.46
PFASST, two $\mathcal{G}$ sweeps	16	3 PFASST	202.13s	7.95	0.50

Table 1. Parallel speedup and efficiency of the PFASST algorithm for the 3d NAD equation.

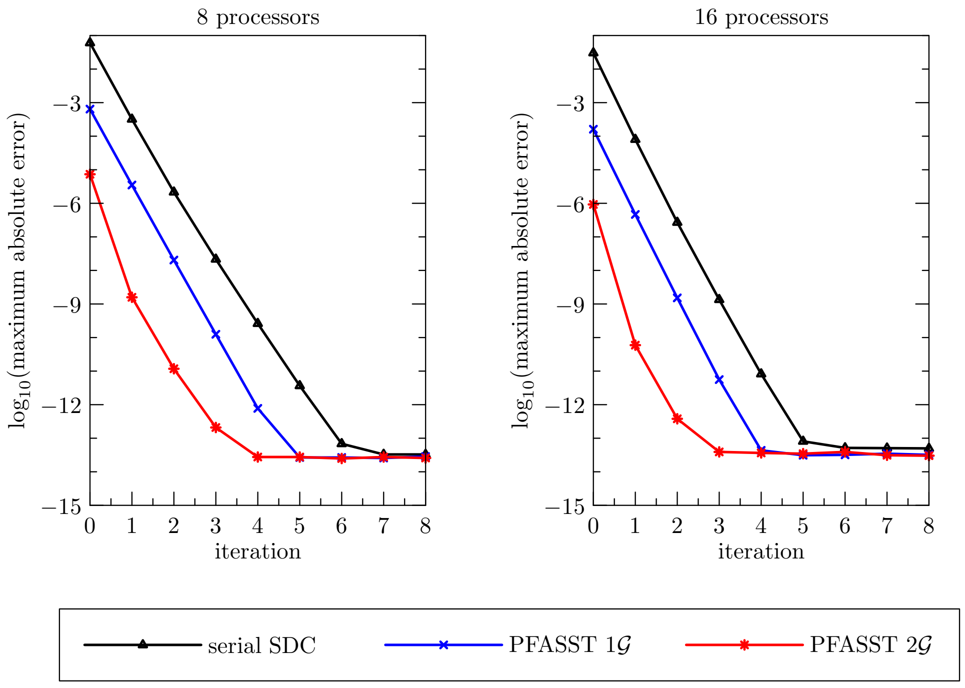

Figure 10. Maximum absolute error for the serial and PFASST iterations computed at the final time for the 3d NAD equation used for parallel timings.

Figure 10 compares the convergence of the PFASST runs and serial SDC runs by considering error versus iteration for the 3d NAD equation computed at the final time. All runs eventually achieve an accuracy of roughly 10^{-13}. This figure justifies the number of iterations used to perform the timings in Table 1.

6. Discussion

The preliminary results included in the previous section suggest that it is possible to achieve reasonable parallel efficiency in the temporal direction. Hence for PDE computations for which spatial parallelization has been saturated, parallelizing in time appears attractive. Efforts to implement the PFASST algorithm within an existing spatial parallelization infrastructure are underway and will be reported on in the future. For problems in which many more time steps are desired than processor groups available, an algorithm for terminating the iterations at one time step so that the processors can begin computation on a new time step will be required. There are also several other immediate research directions we are pursuing that may increase the efficiency of the PFASST approach.

The test problems examined in this paper use spectrally accurate derivative and interpolation operators so that the convergence of the temporal scheme is easy to identify. One drawback of this approach is that the relative cost of interpolation and recomputing explicit function values in the FAS procedure is high. Of obvious interest is the performance of the PFASST method on problems discretized with finite-differences, finite-volumes, or finite-elements. Also, the performance of the PFASST approach on hyperbolic problems or those dominated by dispersive waves has not yet been fully investigated.

For the pseudospectral discretization used throughout the test cases here, the implicit equation at each substep is solved directly using the FFT. In practice, such equations for PDEs are usually solved using an iterative method such as a Krylov subspace or multigrid method. When combined with the PFASST algorithm, another avenue for a further gain in efficiency for PDEs employing iterative solvers is to reduce or vary the number of iterations of the implicit solver in different parts of the algorithm. For example, it may not be necessary to solve implicit equations within SDC substeps to full precision at every iteration, although it is difficult to predict *a priori* how a reduction in spatial solver accuracy will affect the convergence of the time parallel iterations.

Finally, the two-level method used in the paper can be easily extended to multiple levels as in standard multigrid methods. Such a method then resembles a space-time multigrid method where the "relaxation" operator in the temporal direction is an SDC sweep. Analysis and numerical testing of such a multilevel approach is ongoing.

Appendix: Pseudocode for the PFASST algorithm

See Figure 11 for a scheduling diagram of the algorithm.

Initialization on processor P_n.

```
*Spread initial condition and compute FAS correction*
SET: $\boldsymbol{U}^{0,0} = u(0)$, and evaluate $\boldsymbol{F}^{0,0}$
RESTRICT: fine $\boldsymbol{U}^{0,0}$ to coarse $\tilde{\boldsymbol{U}}^{0,0}$, and evaluate $\tilde{\boldsymbol{F}}^{0,0}$
COMPUTE: FAS correction $\boldsymbol{\tau}^0$ between $\boldsymbol{F}^{0,0}$ and $\tilde{\boldsymbol{F}}^{0,0}$
FOR $j = 0, \dots, n-1$:
```

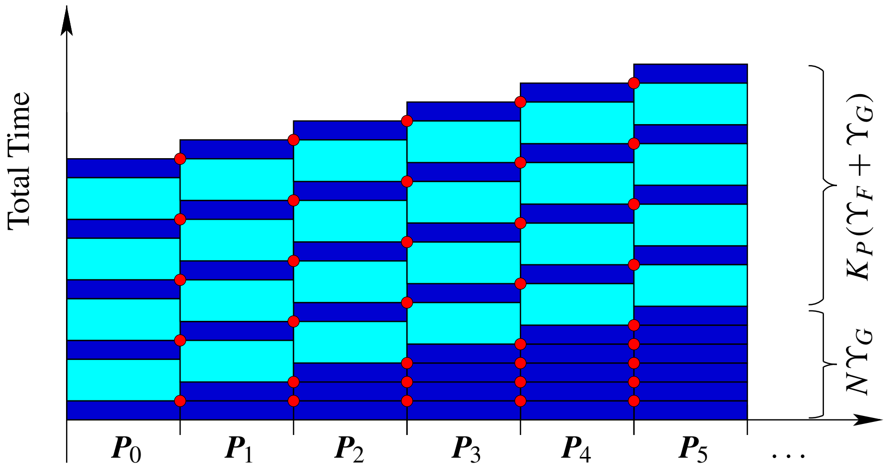

Figure 11. Cost diagram for the PFASST method with two space/time discretizations and the new iterative initialization procedure. Thinner darker rectangles correspond to coarse SDC sweeps, while finer correspond to fine SDC sweeps. Dots correspond to communication between processors.

```
  *Get new coarse initial value, sweep, and send forward*
  IF $j > 0$ and $n > 0$:
    RECEIVE: $\tilde{U}_0^{0,j} = \tilde{U}_{\tilde{M}}^{0,j}$ from $\boldsymbol{P}_{n-1}$
  SWEEP: perform one SDC sweep with $\tilde{\boldsymbol{U}}^{0,j}$ and $\tilde{\boldsymbol{F}}^{0,j}$
  UPDATE: during sweep, update $\tilde{\boldsymbol{U}}^{0,j+1}$ and evaluate $\tilde{\boldsymbol{F}}^{0,j+1}$ appropriately
  IF $n < N-1$:
    SEND: $\tilde{U}_{\tilde{M}}^{0,j+1}$ to $\boldsymbol{P}_{n+1}$
```

PFASST iterations on processor $\boldsymbol{P}_n$.

```
SET: $\tilde{\boldsymbol{U}}^0$ to last coarse initialization $\tilde{\boldsymbol{U}}^{0,n-1}$, and evaluate $\tilde{\boldsymbol{F}}^0$
INTERPOLATE: coarse correction $\tilde{\boldsymbol{U}}^0 - \tilde{\boldsymbol{U}}^{0,0}$ to fine $\boldsymbol{U}^0$, and evaluate $\boldsymbol{F}^0$
FOR $k = 1, \dots, K$
  *Sweep on fine level, restrict, and compute FAS correction*
  SWEEP: perform one SDC sweep with $\boldsymbol{U}^{k-1}$ and $\boldsymbol{F}^{k-1}$
  UPDATE: during sweep, update $\boldsymbol{U}^k$ and evaluate $\boldsymbol{F}^k$ appropriately
  RESTRICT: fine $\boldsymbol{U}^k$ to coarse $\tilde{\boldsymbol{U}}^k$, and evaluate $\tilde{\boldsymbol{F}}^k$
  COMPUTE: FAS correction $\boldsymbol{\tau}^k$ between $\boldsymbol{F}^k$ and $\tilde{\boldsymbol{F}}^k$
  *Receive new fine initial condition, restrict*
  IF $n > 0$:
    RECEIVE: $U_0^k = U_M^k$ from $\boldsymbol{P}_{n-1}$
  RESTRICT: fine $U_0^k$ to coarse $\tilde{U}_0^k$
  *Sweep on coarse level, interpolate final value, and send forward*
  SWEEP: perform one SDC sweep with $\tilde{\boldsymbol{U}}^k$, $\tilde{\boldsymbol{F}}^k$, and $\boldsymbol{\tau}^k$
```

```
UPDATE: during sweep, update $\tilde{\boldsymbol{U}}^k$ and evaluate $\tilde{\boldsymbol{F}}^k$ appropriately
INTERPOLATE: coarse correction $\tilde{U}^k_{\tilde{M}} - \tilde{U}^{k-1}_{\tilde{M}}$ to fine $U^k_M$
IF $n < N - 1$:
   SEND: $U^k_M$ to $\boldsymbol{P}_{n+1}$
*Interpolate coarse to fine and set initial condition*
INTERPOLATE: coarse correction $\tilde{\boldsymbol{U}}^k - \tilde{\boldsymbol{U}}^{k-1}$ to fine $\boldsymbol{U}^k$, and evaluate $\boldsymbol{F}^k$
```

References

[1] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, *Implicit-explicit Runge–Kutta methods for time-dependent partial differential equations*, Appl. Numer. Math. **25** (1997), no. 2-3, 151–167. MR 98i:65054 Zbl 0896.65061

[2] G. Bal, *On the convergence and the stability of the parareal algorithm to solve partial differential equations*, Domain decomposition methods in science and engineering (R. Kornhuber et al., eds.), Lect. Notes Comput. Sci. Eng., no. 40, Springer, Berlin, 2005, pp. 425–432. MR 2235769 Zbl 1066.65091

[3] G. Bal and Y. Maday, *A "parareal" time discretization for non-linear PDE's with application to the pricing of an American put*, Recent developments in domain decomposition methods (L. F. Pavarino and A. Toselli, eds.), Lect. Notes Comput. Sci. Eng., no. 23, Springer, Berlin, 2002, pp. 189–202. MR 1962689

[4] K. Böhmer, P. Hemker, and H. J. Stetter, *The defect correction approach*, Defect correction methods (K. Böhmer and H. J. Stetter, eds.), Comput. Suppl., no. 5, Springer, Vienna, 1984, pp. 1–32. MR 86i:65007 Zbl 0551.65034

[5] S. Boscarino, *Error analysis of IMEX Runge–Kutta methods derived from differential-algebraic systems*, SIAM J. Numer. Anal. **45** (2007), no. 4, 1600–1621. MR 2008g:65086 Zbl 1152.65088

[6] 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), no. 2, 651–675. MR 2004f:76084 Zbl 1061.76053

[7] E. L. Bouzarth and M. L. Minion, *A multirate time integrator for regularized Stokeslets*, J. Comput. Phys. **229** (2010), no. 11, 4208–4224. MR 2011b:65099 Zbl 05712958

[8] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, 2nd ed., Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. MR 2001h:65002 Zbl 0958.65128

[9] J. W. Daniel, V. Pereyra, and L. L. Schumaker, *Iterated deferred corrections for initial value problems*, Acta Ci. Venezolana **19** (1968), 128–135. MR 40 #8270

[10] A. Dutt, L. Greengard, and V. Rokhlin, *Spectral deferred correction methods for ordinary differential equations*, BIT **40** (2000), no. 2, 241–266. MR 2001e:65104 Zbl 0959.65084

[11] C. Farhat and M. Chandesris, *Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications*, Internat. J. Numer. Methods Engrg. **58** (2003), no. 9, 1397–1434. MR 2004h:65154 Zbl 1032.74701

[12] P. F. Fischer, F. Hecht, and Y. Maday, *A parareal in time semi-implicit approximation of the Navier–Stokes equations*, Domain decomposition methods in science and engineering (R. Kornhuber et al., eds.), Lect. Notes Comput. Sci. Eng., no. 40, Springer, Berlin, 2005, pp. 433–440. MR 2235770 Zbl 02143574

[13] M. J. Gander, *Analysis of the parareal algorithm applied to hyperbolic problems using characteristics*, Bol. Soc. Esp. Mat. Apl. S⃗eMA (2008), no. 42, 21–35. MR 2009b:65268

[14] J. Huang, J. Jia, and M. Minion, *Accelerating the convergence of spectral deferred correction methods*, J. Comput. Phys. **214** (2006), no. 2, 633–656. MR 2006k:65173 Zbl 1094.65066

[15] C. A. Kennedy and M. H. Carpenter, *Additive Runge–Kutta schemes for convection-diffusion-reaction equations*, Appl. Numer. Math. **44** (2003), no. 1-2, 139–181. MR 2003m:65111 Zbl 1013.65103

[16] I. G. Kevrekidis, B. Nicolaenko, and J. C. Scovel, *Back in the saddle again: a computer assisted study of the Kuramoto–Sivashinsky equation*, SIAM J. Appl. Math. **50** (1990), no. 3, 760–790. MR 91c:58095 Zbl 0722.35011

[17] A. T. Layton, *On the choice of correctors for semi-implicit Picard deferred correction methods*, Appl. Numer. Math. **58** (2008), no. 6, 845–858. MR 2009e:65116 Zbl 1143.65057

[18] A. T. Layton and M. L. Minion, *Conservative multi-implicit spectral deferred correction methods for reacting gas dynamics*, J. Comput. Phys. **194** (2004), no. 2, 697–715. MR 2004k:76089 Zbl 1100.76048

[19] ———, *Implications of the choice of predictors for semi-implicit Picard integral deferred correction methods*, Commun. Appl. Math. Comput. Sci. **2** (2007), 1–34. MR 2008e:65252 Zbl 1131.65059

[20] J.-L. Lions, Y. Maday, and G. Turinici, *Résolution d'EDP par un schéma en temps "pararéel"*, C. R. Acad. Sci. Paris Sér. I Math. **332** (2001), no. 7, 661–668. MR 2002c:65140

[21] M. L. Minion and S. A. Williams, *Parareal and spectral deferred corrections*, AIP Conference Proceedings, vol. 1048, 2008, pp. 388–391.

[22] M. L. Minion, *Higher-order semi-implicit projection methods*, Numerical simulations of incompressible flows (M. Hafez, ed.), World Sci. Publ., River Edge, NJ, 2003, pp. 126–140. MR 1984431 Zbl 1079.76056

[23] ———, *Semi-implicit projection methods for incompressible flow based on spectral deferred corrections*, Appl. Numer. Math. **48** (2004), no. 3-4, 369–387. MR 2056924 Zbl 1035.76040

[24] ———, *A hybrid parareal spectral deferred corrections method*, Commun. Appl. Math. Comput. Sci. **5** (2010), no. 2, 265–301. MR 2765386 Zbl 1208.65101

[25] L. Pareschi and G. Russo, *Implicit-explicit Runge–Kutta schemes for stiff systems of differential equations*, Recent trends in numerical analysis (D. Trigiante, ed.), Adv. Theory Comput. Math., no. 3, Nova Scientific, Huntington, NY, 2001, pp. 269–288. MR 2005a:65065 Zbl 1018.65093

[26] V. Pereyra, *On improving an approximate solution of a functional equation by deferred corrections*, Numer. Math. **8** (1966), 376–391. MR 34 #3814 Zbl 0173.18103

[27] ———, *Iterated deferred corrections for nonlinear operator equations*, Numer. Math. **10** (1967), 316–323. MR 36 #4812 Zbl 0258.65059

[28] J. W. Shen and X. Zhong, *Semi-implicit Runge–Kutta schemes for the non-autonomous differential equations in reactive flow computations*, Proceedings of the 27th AIAA Fluid Dynamics Conference, AIAA, June 1996, pp. 17–20.

[29] H. J. Stetter, *Economical global error estimation*, Stiff differential systems (R. A. Willoughby, ed.), Plenum, New York, 1974, pp. 245–258. MR 53 #9655

[30] P. Zadunaisky, *A method for the estimation of errors propagated in the numerical solution of a system of ordinary differential equations*, The Theory of Orbits in the Solar System and in Stellar Systems. Proceedings of International Astronomical Union, Symposium 25 (G. Contopoulos, ed.), 1964, pp. 281–287.