COMM. APP. MATH. AND COMP. SCI.
Vol. 5, No. 2, 2010

msp

A HYBRID PARAREAL SPECTRAL DEFERRED CORRECTIONS METHOD

MICHAEL L. MINION

The parareal algorithm introduced in 2001 by Lions, Maday, and Turinici is an iterative method for the parallelization of the numerical solution of ordinary differential equations or partial differential equations discretized in the temporal direction. The temporal interval of interest is partitioned into successive domains which are assigned to separate processor units. Each iteration of the parareal algorithm consists of a high accuracy solution procedure performed in parallel on each domain using approximate initial conditions and a serial step which propagates a correction to the initial conditions through the entire time interval. The original method is designed to use classical single-step numerical methods for both of these steps. This paper investigates a variant of the parareal algorithm first outlined by Minion and Williams in 2008 that utilizes a deferred correction strategy within the parareal iterations. Here, the connections between parareal, parallel deferred corrections, and a hybrid parareal-spectral deferred correction method are further explored. The parallel speedup and efficiency of the hybrid methods are analyzed, and numerical results for ODEs and discretized PDEs are presented to demonstrate the performance of the hybrid approach.

1. Introduction

The prospect of parallelizing the numerical solution of ordinary differential equations (ODEs) in the temporal direction has been the topic of research dating back at least to the early work of Nievergelt [55] and Miranker and Liniger [54]. These early papers as well as the methods described here employ multiple processing units to compute the solution over multiple time intervals in parallel. Hence, in the classification used in [13], for example, these methods are categorized as employing parallelization *across the steps* as opposed to *across the method* or *across the problem*.

Examples of approaches to parallelization across the method include the computation of intermediate or stage values in Runge–Kutta and general linear methods simultaneously on multiple processors [38; 14]. These attempts to parallelize can

MSC2000: 65L99.

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

only be efficient when the number of processors being used is no larger than the number of stage values and hence typically yield modest parallel speedup.

Methods that utilize parallelization *across the problem* rely on a splitting of the problem into subproblems that can be computed in parallel and an iterative procedure for coupling the subproblems so that the overall method converges to the solution of the full problem. A well known class of methods in this style is the parallel waveform-relaxation schemes [25; 61; 18]. Despite efforts to accelerate the rate of convergence of wave-form relaxation methods [29; 34], convergence can still be slow, especially for stiff problems.

In addition to the methods in [55; 54], methods based on multiple shooting [40] also employ parallelization across the steps. In 2001, a new approach similar in spirit to multiple shooting was introduced in [45]. The so called *parareal* method is appropriate for larger numbers of processors and has sparked many new papers devoted to the subject of time parallelization. The parareal method has been further analyzed and refined [47; 22; 7; 60; 49; 30; 28; 62; 48; 32; 27; 9] and implemented for different types of applications [8; 23; 46]. This paper details a new variant of the parareal method first outlined in [53] that utilizes an iterative ODE method based on deferred corrections within the parareal iteration. As outlined below, this approach can either be thought of as a way to parallelize a deferred correction approach to solving ODEs or as a way to increase the efficiency of the parareal algorithm by removing the requirement to solve the subproblems on each processor during each iteration with a full accuracy solver.

One of the justifications of the use of parareal methods is the scenario where a specific computation must be completed in a fixed amount of time and sufficient computational resources are available. In the context of the numerical solution of time dependent PDEs, although parallelization of methods in the spatial dimensions has seen a tremendous amount of successful research, for a fixed problem size, spatial parallel speedup will eventually saturate as more processors are employed. If additional processors are available, then additional parallelization in the temporal direction could reduce the overall parallel computational cost. The name *parareal* in fact is derived from *parallel* and *real time* and encapsulates the desire to complete a computation of a specific size faster in real time. This is in contrast to the common practice of reporting spatial parallel efficiency or speedup in the context of increasing problem size as the number of processors are increased. The current work is motivated by the desire to develop efficient methods for time-space parallelization for PDEs.

As discussed in detail in Section 3, it has become standard to describe the parareal algorithm in terms of two computational methods to approximate the temporal evolution of the equation over a fixed time interval. A fine, or accurate, method (denoted here by $\mathcal{F}$) computes an accurate (and hence more computationally

expensive) approximation to the solution. A coarse or less accurate method (denoted here by $\mathcal{G}$) is also defined, and is used in a serial fashion to propagate a correction through the time domain. We will refer to $\mathcal{G}$ and $\mathcal{F}$ as the coarse and fine propagators respectively. The parareal method alternates between the parallel application of $\mathcal{F}$ on multiple time domains using approximate initial conditions and a serial sweep using $\mathcal{G}$ that propagates a correction to the initial conditions throughout the time interval. Upon convergence, the accuracy of the parareal method is limited by what one would obtain if the $\mathcal{F}$ method was used in serial on each subdomain. Hence, as explained in detail in Section 5, in order for the parareal method to achieve parallel efficiency, $\mathcal{G}$ must be less expensive than $\mathcal{F}$. The reduced computational cost of $\mathcal{G}$ can be achieved by using a coarser time step for $\mathcal{G}$ than for $\mathcal{F}$, or a less expensive numerical method, or both. As has been pointed out [8; 7; 23; 26], for PDEs, it is also possible to use a coarser spatial discretization for $\mathcal{G}$. The parareal algorithm can, in principle, use any self-starting ODE method for $\mathcal{F}$ and $\mathcal{G}$ and hence can be used in a "black box" fashion.

As detailed in Section 5, the parallel efficiency of parareal is limited by the fact that during each iteration, the parallel application of $\mathcal{F}$ has the same computational cost as the serial algorithm applied to the subdomain. Hence, a significant parallel speed up (the ratio of serial to parallel cost) can only be achieved if the number of iterations required to converge to the serial solution to a given tolerance is significantly smaller than the number of subdomains. Similarly, the parallel efficiency (speedup divided by the number of processors), is bounded above by the reciprocal of the number of iterations regardless of the cost of $\mathcal{G}$. In most instances, the total computational cost of the parareal algorithm is dominated by the cost of the $\mathcal{F}$.

In [53], a new variant of the parareal method is presented that uses an iterative method for solving ODEs for the coarse and fine propagators rather than traditional methods like Runge–Kutta (RK) that are typically used in the literature. The key observation in [53] is that the $\mathcal{F}$ propagator in traditional parareal approaches makes no use of the previously computed solution on the same interval (a recent alternative approach to reusing information appears in [28]). It is shown how the use of an iterative method can be combined with parareal to improve the solution from the previous parareal iteration rather than computing a solution from scratch. The result is that the $\mathcal{F}$ propagator becomes much cheaper than a full accuracy solution on the interval, and in fact the dominant cost in the numerical tests in [53] becomes the $\mathcal{G}$ propagator.

The numerical method in [53] is based on the method of spectral deferred corrections (SDC) [21]. Since SDC methods converge to the solution of the Gaussian collocation formula, very accurate solutions can be obtained using a modest number of SDC substeps per time step. This accuracy is offset by the relatively high computational cost of SDC methods *per time step*. However, when one compares

the computational cost for a given (sufficiently small) error tolerance, SDC methods have been shown to compare favorably to RK schemes. This is especially true in the case of problems for which semi-implicit or implicit-explicit (IMEX) methods are appropriate since very higher-order IMEX RK methods have not been developed whereas IMEX SDC methods with arbitrarily high formal order of accuracy are easily constructed [51]. One main point of this paper is that the relatively high cost per time step of SDC methods can be effectively amortized when SDC methods are combined with the parareal algorithm since only one (or a few) SDC iterations are done during each parareal iteration. This means that the cost of the $\mathcal{F}$ propagator is similar to that of a modest number of steps of a low-order method rather than many steps of a higher-order method. Differences between the hybrid parareal/SDC approach and recent parallel deferred correction methods [33; 16] are discussed in Section 4.1.

The preliminary numerical results included in Section 6 suggest that the use of a single SDC iteration in lieu of a full-accuracy $\mathcal{F}$ propagator does not significantly affect the convergence behavior of the parareal iteration. Rather, the accuracy of $\mathcal{G}$ determines the rate of convergence (as was proven for the parareal method in [27]). Since using a single SDC iteration is markedly less expensive than a higher-order RK method with finer time steps, the dominant cost of the parareal/SDC hybrid method when many processors are used becomes the serial procedure for initializing the solution on each processor (typically done with $\mathcal{G}$). Hence for PDEs, the possibility of reducing the cost of $\mathcal{G}$ by using a coarser spatial discretization (already proposed in [8; 7; 23; 26]) is very attractive. This idea will be pursued in a sequel to this paper.

2. Spectral deferred corrections

The spectral deferred correction method (SDC) is a variant of the traditional deferred and defect correction methods for ODEs introduced in the 1960s [63; 57; 58; 19]. The original methods never gained the popularity of Runge–Kutta or linear multistep methods, however, a series of papers beginning in 2000 has rekindled interest in using such methods for large scale physical simulations. The SDC method introduced in [21] couples a Picard integral formulation of the correction equation with spectral integration rules to achieve stable explicit and implicit methods with arbitrarily high formal order of accuracy.

SDC methods possess two characteristics that make them an attractive option for the temporal integration of complex physical applications. First, SDC methods with an arbitrarily high formal order of accuracy and good stability properties can easily be constructed. Second, the SDC framework provides the flexibility to apply different time-stepping procedures to different terms in an equation (as in

operator splitting methods) while maintaining the high formal order of accuracy. This second property has led to the development of semi- and multiimplicit methods for equations with disparate time scales [50; 52; 10; 42]. Since SDC methods are nonstandard, the original SDC method will be reviewed in the next section followed by a brief discussion in Section 2.2 of the semi-implicit variants that are used in the second numerical example in Section 6.2.

2.1. ***Original spectral deferred corrections.*** Consider the ODE initial value problem

$$\begin{aligned} u'(t) &= f(t, u(t)), \quad t \in [0, T], \\ u(0) &= u_0, \end{aligned}$$

where $u_0, u(t) \in \mathbb{C}^N$ and $f : \mathbb{R} \times \mathbb{C}^N \to \mathbb{C}^N$. This equation is equivalent to the Picard integral equation

$$u(t) = u_0 + \int_0^t f(\tau, u(\tau))\, d\tau, \tag{1}$$

and this latter form is used extensively in the discussion that follows.

As with traditional deferred correction methods, a single time step $[t_n, t_{n+1}]$ of size $\Delta t = t_{n+1} - t_n$ is divided into a set of intermediate substeps by defining $\boldsymbol{t}_n = [t_1, \ldots, t_J]$ with $t_n \le t_1 < \cdots < t_J \le t_{n+1}$; however, for SDC methods, $\boldsymbol{t}_n$ corresponds to Gaussian quadrature nodes. The intermediate times in $\boldsymbol{t}_n$ will be denoted by the subscript j with $j = 1 \ldots J$, and numerical approximations of quantities at time t_j will likewise carry the subscript j. Beginning with the initial condition for the time step $U_{n,1} \approx u(t_n)$, a provisional approximation $\boldsymbol{U}_n^0 = [U_{n,1}^0, \ldots, U_{n,J}^0]$ is computed at the intermediate points using a standard numerical method. The superscripts on numerical values (for example $\boldsymbol{U}_n^0$) denote here the iteration number in the SDC procedure. The continuous counterpart of $\boldsymbol{U}_n^0$ can be constructed by standard interpolation theory and is represented as $U_n^0(t)$. Using $U_n^0(t)$, an integral equation similar to (1) for the error $\delta(t) = u(t) - U_n^0(t)$ is then derived

$$\delta(t) = \int_{t_n}^t \left[f(\tau, U_n^0(\tau) + \delta(\tau)) - f(\tau, U_n^0(\tau)) \right] d\tau + \epsilon(t), \tag{2}$$

where

$$\epsilon(t) = U_n + \int_{t_n}^t f(\tau, U_n^0(\tau))\, d\tau - U_n^0(t). \tag{3}$$

Note that $\epsilon(t_j)$ can be accurately and stably approximated using spectral integration [31], since the provisional solution $U^0(t)$ is known at the Gaussian quadrature

nodes. An update form of (2) is

$$\delta(t_{j+1}) = \delta(t_j) + \int_{t_j}^{t_{j+1}} \left[f(\tau, U_n^0(\tau) + \delta(\tau)) - f(\tau, U_n^0(\tau)) \right] d\tau + \epsilon(t_{j+1}) - \epsilon(t_j). \quad (4)$$

In order to discretize (4), an approximation to the integral term in

$$\epsilon(t_{j+1}) - \epsilon(t_j) = \int_{t_j}^{t_{j+1}} f(\tau, U_n^0(\tau)) d\tau - U_n^0(t_{j+1}) + U_n^0(t_j). \quad (5)$$

must be constructed. This is done using spectral integration representing quadrature at the nodes $\boldsymbol{t}_n$. The spectral quadrature approximation is denoted by

$$S_j^{j+1} f(\boldsymbol{t}_n, \boldsymbol{U}_n^0) \approx \int_{t_j}^{t_{j+1}} f(\tau, U_n^0(\tau)) d\tau, \quad (6)$$

and the computation of the values $S_j^{j+1} f(\boldsymbol{t}_n, \boldsymbol{U}_n^0)$ is a matrix-vector multiplication using a precomputed integration matrix (see [35] for details).

A low-order method is then applied to approximate (2) at the points $\boldsymbol{t}_n$ resulting in a correction to the provisional solution. For example, an explicit time-stepping scheme similar to the forward Euler method is

$$\delta_{j+1}^0 = \delta_j^0 + \Delta t_j [f(t_j, U_{n,j}^0 + \delta_j^0) - f(t_j, U_{n,j}^0)] + S_j^{j+1} f(\boldsymbol{t}_n, \boldsymbol{U}_n^0) - U_{n,j+1}^0 + U_{n,j}^0, \quad (7)$$

where $\Delta t_j = t_{j+1} - t_j$ and again subscripts on numerical values denote approximations corresponding the t_j. Similarly, an implicit method similar to the backward Euler method is

$$\delta_{j+1}^0 = \delta_j^0 + \Delta t_j [f(t_{j+1}, U_{n,j+1}^0 + \delta_{j+1}^0) - f(t_{j+1}, U_{n,j+1}^0)] + S_j^{j+1} f(\boldsymbol{t}_n, \boldsymbol{U}_n^0) - U_{n,j+1}^0 + U_{n,j}^0. \quad (8)$$

The correction (2) can also be approximated by higher-order methods [41; 17].

The provisional numerical solution is then updated by adding to it the approximation of the correction, that is, $U_{n,j}^1 = U_{n,j}^0 + \delta_{n,j}^0$. The SDC method then proceeds iteratively, by recomputing the residuals, approximating a new correction, and setting $U_{n,j}^{k+1} = U_{n,j}^k + \delta_{n,j}^k$. Each SDC iteration raises the formal order of accuracy of the numerical solution by the order of the approximation to (4) provided the quadrature rule in (6) is sufficiently accurate. In the methods used in the numerical experiments presented here, (2) is approximated with a first-order method so that M total SDC sweeps (including the predictor) are needed for M-th order accuracy.

An alternative form of (8) for general k can be derived using $U_{n,j}^{k+1} = U_{n,j}^k + \delta_{n,j}^k$:

$$U_{n,j+1}^{k+1} = U_{n,j}^{k+1} + \Delta t_j \left(f(t_{j+1}, U_{n,j+1}^{k+1}) - f(t_{j+1}, U_{n,j+1}^k) \right) + S_j^{j+1} f(\boldsymbol{t}_n, \boldsymbol{U}_n^k), \quad (9)$$

This form of the update equation is compared below to the serial step in the parareal algorithm.

2.2. ***Semiimplicit methods.*** SDC methods are particularly well-suited for the temporal integration of ODEs which can be split into stiff and nonstiff components. When both stiff and nonstiff terms appear in the equation, it is often significantly more efficient to use methods that treat only the stiff terms implicitly and treat nonstiff terms explicitly. Such methods are usually referred to as semi-implicit or IMEX (implicit-explicit) methods. IMEX linear multistep methods that build on a backward difference formula treatment of the stiff term have been developed [5; 24; 1; 2; 3; 37], but like backward difference formula methods themselves, the stability of these methods deteriorates as the order increases, and methods above sixth-order are unstable (see the discussion in [44]). IMEX or additive Runge–Kutta methods have also been proposed [59; 64; 4; 15; 39; 56], but methods with order higher than five have not yet appeared.

To derive IMEX SDC methods, consider the ODE

$$u'(t) = f(t, u(t)) = f_E(t, u(t)) + f_I(t, u(t)), \quad t \in [0, T], \tag{10}$$

$$u(0) = u_0. \tag{11}$$

Here the right hand side of the equation is split into two terms, the first of which is assumed to be nonstiff (and hence treated explicitly), and the second of which is assumed to be stiff (and treated implicitly). A first-order semi-implicit method for computing an initial solution is simply

$$U^0_{n,j+1} = U^0_{n,j} + \Delta t_j \big(f_E(t_j, U^0_{n,j}) + f_I(t_{j+1}, U^0_{n,j+1})\big). \tag{12}$$

Following the same logic used to derive (2), one arrives at the correction equation

$$\delta(t) = \int_0^t \big[f_E(\tau, U^0(\tau)+\delta(\tau)) - f_E(\tau, U^0_n(\tau)) + f_I(\tau, U^0_n(\tau)+\delta(\tau)) - f_I(\tau, U^0_n(\tau))\big]\, d\tau + \epsilon(t), \tag{13}$$

where

$$\epsilon(t) = U_0 + \int_0^t f_E(\tau, U^0_n(\tau)) + f_I(\tau, U^0_n(\tau))d\tau - U^0_n(t). \tag{14}$$

A simple semi-implicit time-stepping method analogous to (9) is then

$$U^{k+1}_{n,j+1} = U^{k+1}_{n,j} + \Delta t_j \big(f_E(t_j, U^{k+1}_{n,j}) - f_E(t_j, U^k_{n,j}) + f_I(t_{j+1}, U^{k+1}_{n,j+1}) - f_I(t_{j+1}, U^k_{n,j+1})\big) + S^{j+1}_j f(\boldsymbol{t}_n, \boldsymbol{U}^k_n). \tag{15}$$

In [50; 52], such a semi-implicit version of SDC is used in combination with an auxiliary variable projection method approach for the Navier–Stokes equations that treats the viscous terms implicitly and the nonlinear advective terms explicitly.

These methods have been subsequently examined in more detail in [50; 44; 42; 41], and the second numerical example in Section 6.2 is based on this semi-implicit approach. Semi-implicit SDC methods have been extended to treat three or more terms (explicit or implicit) in (10), including modifications that allow different terms in the equation to be treated with different time steps [10; 42; 11].

2.3. ***Computational cost and storage.*** Like any numerical method, SDC has disadvantages as well as advantages. The price one pays to achieve the flexibility and high order of accuracy for SDC methods is primarily in the large computational cost *per time step*. SDC methods that use a first-order numerical method in the correction iterates require M total iterations per time step (provisional and correction) to achieve formal M-th order accuracy. Since each iteration sweep requires that the solution be computed at a number of substeps that is proportional to the order, the number of function evaluations per time step grows quadratically with the order. This makes the cost per time step appear quite high compared to linear multistep or Runge–Kutta methods.

However, relying on the number of function evaluations per time step as a measure of efficiency is very misleading. First, the stability region of SDC methods also increases roughly linearly with the order, so that larger substeps can be taken as the order increases [21; 43]. In addition, a more relevant measure of cost is in terms of computational effort versus error, and as is demonstrated in Section 6, SDC methods compare well with higher-order RK methods in this measure (see also comparisons in [51; 42]). For equations with both stiff and nonstiff terms, there are no semi- or multiimplicit methods based on RK or linear multistep methods with order of accuracy greater than six, so particularly when a small error is required, higher-order SDC method are very attractive (see Section 6.2). Additionally, techniques to reduce the computational cost of SDC methods by accelerating the convergence have also appeared [35].

In the current context, however, it is the parallel cost of the time integration method that is of interest. One of the main results of this paper is that, when combined with the parareal strategy, the high cost per time step of SDC methods due to the need to iterate the correction equation is amortized over the iterations that must be performed during the parareal methods. Hence the cost of SDC methods per parareal iteration is much smaller than for a noniterative method like RK.

As previously mentioned, SDC methods require that function values be stored at a set of substeps within a given time step, which correspond to quadrature nodes of the Picard integral discretization. If a semi- or multiimplicit operator splitting treatment is used, each split piece of the function must be stored separately. Since the number of substeps grows linearly with the order of the method, the storage costs are comparable to higher-order Runge–Kutta or linear multistep methods.

3. The parareal method

The parareal method was introduced in 2001 by Lions, Maday and Turinici [45] and has sparked renewed interest in the construction of time parallel methods. In this section, a short review of the parareal method is provided, and then comparisons between parareal and spectral deferred corrections will be explored.

3.1. ***Notation and method.*** The general strategy for the parareal method is to divide the time interval of interest $[0, T]$ into N intervals with each interval being assigned to a different processor. To simplify the discussion, assume that there are N processors $\boldsymbol{P}_0$ through $\boldsymbol{P}_{N-1}$, and that the time intervals are of uniform size $\Delta T = T/N$ so that the n-th processor computes the solution on the interval $[t_n, t_{n+1}]$ where $t_n = n\Delta T$. On each interval, the parareal method iteratively computes a succession of approximations $U_{n+1}^k \approx u(t_{n+1})$, where k denotes the iteration number.

It is becoming standard to describe the parareal algorithm in terms of two numerical approximation methods denoted here 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 (1) from t_n to t_{n+1}. For example, if $\mathcal{G}$ is defined by the forward Euler method applied to (1) , then

$$\mathcal{G}(t_{n+1}, t_n, U_n) = U_n + (t_{n+1} - t_n) f(t_n, U_n). \tag{16}$$

As discussed below, 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; hence, in practice, $\mathcal{G}$ is usually a low-order method. Note that $\mathcal{G}$ or $\mathcal{F}$ could be defined to be more than one step of a particular numerical method on the interval $[t_n, t_{n+1}]$. 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_n^0 for $n = 1 \ldots N$ often performed with the coarse propagator $\mathcal{G}$, that is,

$$U_{n+1}^0 = \mathcal{G}(t_{n+1}, t_n, U_n^0) \tag{17}$$

with $U_0^0 = u(0)$. Alternatively, one could use the parareal method with a coarser time step to compute the initial approximation [8; 7]. Once each processor $\boldsymbol{P}_n$ has a value U_n^0, the processors can in parallel compute the approximation $\mathcal{F}(t_{n+1}, t_n, U_n^0)$. This step is in spirit an accurate solution of the ODE on the interval $[t_n, t_{n+1}]$ using the approximate starting value U_n^0. Lastly, the parareal algorithm computes the serial correction step

$$U_{n+1}^{k+1} = \mathcal{G}(t_{n+1}, t_n, U_n^{k+1}) + \mathcal{F}(t_{n+1}, t_n, U_n^k) - \mathcal{G}(t_{n+1}, t_n, U_n^k), \tag{18}$$

for $n = 0 \ldots N-1$. The parareal method proceeds iteratively alternating between the parallel computation of $\mathcal{F}(t_{n+1}, t_n, U_n^k)$ and the serial computation of (18), which requires computing the $\mathcal{G}$ propagator. The calculation in (18) will be referred to as the $\mathcal{G}$ correction sweep.

Parareal is an iterative method and hence requires a stopping criteria. Note that after k iterations of the parareal method, the solution U_m^k 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. Since each iteration of the parareal method requires the application of both $\mathcal{F}$ in parallel and $\mathcal{G}$ in serial (plus the cost of communication between processors), the parareal method can only provide parallel speedup compared to the serial $\mathcal{F}$ scheme if the number of iterations required to converge to the specified criteria (denoted here by K) is significantly less than N (see discussion in Section 5).

3.2. ***An examination of the parareal correction equation.*** Here we review the connection between deferred corrections and the parareal step defined by (18) first outlined in [53]. Both $\mathcal{F}$ and $\mathcal{G}$ are approximations to the exact update given by the Picard equation

$$u(t_{n+1}) = u(t_n) + \int_{t_n}^{t_{n+1}} f(\tau, u(\tau)) d\tau. \tag{19}$$

To highlight the approximation of $\mathcal{F}$ and $\mathcal{G}$ to the Picard Equation (19), we define

$$\mathcal{J}(t_{n+1}, t_n, U_n^k) = \mathcal{F}(t_{n+1}, t_n, U_n^k) - U_n^k, \tag{20}$$

$$\mathcal{Q}(t_{n+1}, t_n, U_n^k) = \mathcal{G}(t_{n+1}, t_n, U_n^k) - U_n^k, \tag{21}$$

so that

$$\mathcal{F}(t_{n+1}, t_n, U_n^k) = U_n^k + \mathcal{J}(t_{n+1}, t_n, U_n^k), \tag{22}$$

$$\mathcal{G}(t_{n+1}, t_n, U_n^k) = U_n^k + \mathcal{Q}(t_{n+1}, t_n, U_n^k). \tag{23}$$

Using these definitions, (18) can be rewritten

$$U_{n+1}^{k+1} = U_n^{k+1} + \mathcal{Q}(t_{n+1}, t_n, U_n^{k+1}) - \mathcal{Q}(t_{n+1}, t_n, U_n^k) + \mathcal{J}(t_{n+1}, t_n, U_n^k). \tag{24}$$

In the discussion leading to (9), (4) is discretized using a backward Euler type method to give a concrete example of a time stepping scheme. If $\mathcal{G}$ is similarly defined as a single step of backward Euler, then $\mathcal{Q}(t_{n+1}, t_n, U_n^k) = \Delta t f(t_{n+1}, U_{n+1}^k)$, and (24) becomes

$$U_{n+1}^{k+1} = U_n^{k+1} + \Delta t \big(f(t_{n+1}, U_{n+1}^{k+1}) - f(t_{n+1}, U_{n+1}^k) \big) + \mathcal{J}(t_{n+1}, t_n, U_n^k). \tag{25}$$

Note the similarities between this particular first-order incarnation of the parareal update and the first-order SDC substep given in (9). The two differences between

these equations are how the previous fine solution is used in the update and the fact that in the parareal update, only the solution at the end of the time interval is updated while the SDC sweep is done for each intermediate substep in the time interval. A hybrid parareal/SDC approach using deferred corrections for both the $\mathcal{F}$ propagator and the $\mathcal{G}$ correction sweep is described next.

4. Parallel and parareal SDC

4.1. ***Parallel SDC.*** In the description of SDC above, note that once the provisional solution has been computed at all the intermediate points t_j in a given time step, a provisional starting value for the next time step is available. This observation naturally leads to a time parallel version of SDC in which multiple processors are computing SDC iterations on a contiguous block of time steps simultaneously. Variations of this approach have been employed in [33; 16] to create parallel versions of deferred correction schemes. In practice, such an approach is only efficient when the number of processors is approximately the number of iterations of the serial deferred correction scheme. This then allows the first processor to finish the calculation of the first time step as the last processor is finishing the provisional solution. Then the first processor can receive a starting value from the last processor and continue forward in time. Hence, the parallel speedup in this type of parallel SDC method does not result from using an iterative strategy over the full time domain, but rather computing the necessary iterations of the SDC method on multiple processors simultaneously. In spirit, the hybrid parareal/SDC methods discussed below combine the parallel speedup obtained with parallel deferred corrections with that obtained with parareal.

4.2. ***Parareal using SDC.*** Since the parareal algorithm can in principle use any single-step ODE method for the $\mathcal{F}$ and $\mathcal{G}$ propagators, it would be straightforward to incorporate an SDC method into the parareal framework. However, in the standard parareal scheme, if the $\mathcal{F}$ propagator were an SDC method requiring M iterations in serial, then in the k-th parareal iteration, processor $\boldsymbol{P}_n$ would compute $\mathcal{F}$ using the initial value U_n^k by performing M total SDC iterations. When $k > 1$, however, it would be foolish to ignore the results of the $\mathcal{F}$ propagator from iteration $k-1$ when using SDC in iteration k.

Instead, the $\mathcal{F}$ propagator could perform one or several SDC sweeps on the solution from the previous parareal iteration (incorporating the initial condition in the first correction substep). In this approach, the fine solution is computed on each processor on the J Gaussian quadrature nodes within the time slice assigned to the processor. These values are stored from one parareal iteration to the next, and the $\mathcal{F}$ propagator is replaced by L SDC sweeps (e.g., the method described in (9) or a higher-order version thereof) over the J nodes.

As the parareal iterations converge, the fine solution on each processor converges to the high-accuracy SDC solution (in fact the spectral collocation solution [35]), but the cost of applying the $\mathcal{F}$ propagator during each iteration is that of a low-order method and less than a full step of the SDC method by a factor of M/L. Numerical experiments presented here suggest that $L = 1$ is sufficient for an efficient method. Hence, the cost of $\mathcal{F}$ is similar to J steps of a low-order method. In the numerical examples presented in Section 6, Gauss–Lobatto nodes are used for the fine solution with $J = 5$, 7 and 9.

In addition, following the argument in Section 3.2, the $\mathcal{G}$ corrector sweep can also be cast as a deferred correction sweep which both updates the initial condition for the following time step and provides a correction to the solution in the interval. However, since it is desirable to have $\mathcal{G}$ be as inexpensive as possible, the correction to the solution generated in the $\mathcal{G}$ correction sweep will generally not be available at all the fine SDC nodes. In [53] the $\mathcal{G}$ correction sweep is computed using one step of a third-order RK method applied to the correction equation, and the correction is interpolated from the 3 stage values to the SDC nodes of the fine solution. In general, $\mathcal{G}$ could be computed on a subset of the fine SDC nodes and the correction interpolated, and this approach is used in the examples presented here.

4.3. ***Parareal/SDC specifics.*** A detailed description of the parareal/SDC algorithm used in the numerical results is now presented. The algorithm in pseudocode appears in the Appendix. As in parareal, assume the time interval of interest $[0, T]$ is divided into N uniform intervals $[t_n, t_{n+1}]$ where $t_n = n\Delta T$ is assigned to $\boldsymbol{P}_n$. On each interval $[t_n, t_{n+1}]$ choose the J fine SDC nodes $\boldsymbol{t}_n$ corresponding to the Gauss–Lobatto nodes with $t_n = t_{n,1} < \cdots < t_{n,J} = t_{n+1}$. Likewise choose some subset $\tilde{\boldsymbol{t}}_n$ of size $\tilde{J}$ of $\boldsymbol{t}_n$ corresponding to the substeps for the coarse propagator $\mathcal{G}$. In the first numerical example $\tilde{J} = 3$, that is, $\tilde{\boldsymbol{t}}_n$ is the midpoint and endpoints of $[t_n, t_{n+1}]$. This situation is shown in Figure 1, where $J = 7$ and $\tilde{J} = 3$. The solution at the coarse nodes $\tilde{\boldsymbol{t}}_n$ on processor $\boldsymbol{P}_n$ during iteration k is denoted $\tilde{U}^k_{n,\tilde{j}}$, while the solution at the fine nodes is denoted $U^k_{n,j}$. One last piece of notational convention is that $f(\boldsymbol{t}_n, \boldsymbol{U}^k_n)$ refers to the set of function values $f(t_j, U^k_{n,j})$ at the nodes $\boldsymbol{t}_n$, with analogous notation using $\tilde{\boldsymbol{t}}_n$.

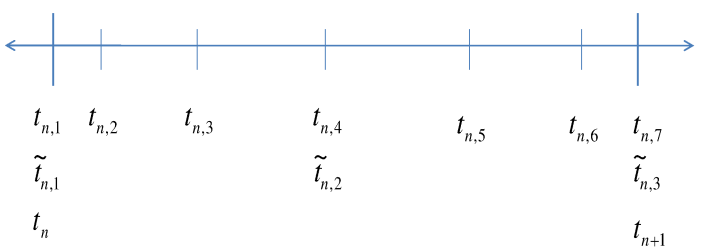

Figure 1. Notation for SDC substeps for the $\mathcal{F}$ and $\mathcal{G}$ propagators.

Predictor step. Starting with the initial data $\tilde{U}^0_{0,1} = u(0)$ on processor $\boldsymbol{P}_0$, compute the initial approximation $\tilde{U}^0_{n,1}$ on each processor in serial. Each processor (except $\boldsymbol{P}_0$) must wait to receive the value $\tilde{U}^0_{n,1} = \tilde{U}^0_{n-1,\tilde{J}}$ from processor $\boldsymbol{P}_{n-1}$. Then the values $\tilde{U}^0_{n,\tilde{j}}$ for $\tilde{j} = 1 \ldots \tilde{J}$ are computed using the numerical method of $\mathcal{G}$. The value $\tilde{U}^0_{n,\tilde{J}}$ is then passed to the processor $\boldsymbol{P}_{n+1}$ (if $n < N-1$), where it is received as $\tilde{U}^0_{n+1,1}$.

First parallel iteration $(k = 1)$**.** As soon as processor $\boldsymbol{P}_n$ is finished computing the last predictor value $\tilde{U}^0_{n,\tilde{J}}$ and sending it to $\boldsymbol{P}_{n+1}$ (if $n < N-1$), the following steps are taken in the first parallel iteration. Note that each of these steps can be done in parallel if the method is pipelined (see Section 5.2).

(1) Interpolate the values $\tilde{U}^0_{n,\tilde{j}}$ to all of the fine nodes $\boldsymbol{t}$ to form $U^0_{n,j}$ and compute $f(t_j, U^0_{n,j})$ for $j = 1 \ldots J$.

(2) Compute the values $S^{j+1}_j f(\boldsymbol{t}_n, \boldsymbol{U}^0_n)$ for $j = 1 \ldots J-1$ using the spectral quadrature rule as in (6).

(3) Perform one sweep ($J-1$ substeps) of the numerical method for the $\mathcal{F}$ propagator applied to the correction (2) using the values $f(t_j, U^0_{n,j})$ and $S^{j+1}_j f(\boldsymbol{t}_n, \boldsymbol{U}^0_n)$ just computed. This will yield updated values $U^1_{n,j}$.

(4) Compute the values $S^{\tilde{j}+1}_{\tilde{j}} f(\tilde{\boldsymbol{t}}_n, \boldsymbol{U}^1_n)$ for $\tilde{j} = 1 \ldots \tilde{J}-1$. Since the integral over a coarse time step is the sum of the integrals over the corresponding fine time steps, this is done by summing the corresponding values of the spectral integration rule $S^{j+1}_j f(\boldsymbol{t}_n, \boldsymbol{U}^1_n)$.

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

(6) Perform one sweep ($\tilde{J}-1$ substeps) of the numerical method for the $\mathcal{G}$ propagator applied to the correction (2) using the values $S^{\tilde{j}+1}_{\tilde{j}} f(\tilde{\boldsymbol{t}}_n, \boldsymbol{U}^0_n)$ just computed. This will yield updated values $\tilde{U}^1_{n,\tilde{j}}$ for $\tilde{j} = 1 \ldots \tilde{J}$.

(7) Pass the value $\tilde{U}^1_{n,\tilde{J}}$ to processor $\boldsymbol{P}_{n+1}$ (if $n < N-1$) which will be received as $\tilde{U}^2_{n+1,1}$

Each subsequent iteration. After the first iteration, the parareal/SDC algorithm proceeds in nearly the same way as in the first iteration except in the first two steps above. In the first iteration, the coarse values from the initial predictor must be interpolated to the fine nodes in step 1 above. In subsequent iterations, there are two alternatives regarding how the results from applying $\mathcal{G}$ in the previous parareal iteration are used. First, one could use no information from the values $U^{k-1}_{n,\tilde{j}}$ computed in the previous $\mathcal{G}$ step. This alternative is in the spirit of parareal where the $\mathcal{G}$ correction sweep only provides a new initial value for the $\mathcal{F}$ step on

the next processor. In this case step 1 above would be skipped and instead of using a quadrature rule in step 2, set $S_j^{j+1} f(\boldsymbol{t}, U_{n,j}^k) = S_j^{j+1} f(\boldsymbol{t}, \tilde{U}_{n,j}^{k-1})$.

Alternatively, the coarse values $U_{n,\tilde{j}}^{k-1}$ could also be used to improve the solution at the fine nodes as well. In the numerical tests included here, this is done simply by forming the difference $U_{n,\tilde{j}}^{k-1} - \tilde{U}_{n,\tilde{j}}^{k-1}$ at the coarse nodes, and then interpolating this difference to the points $\boldsymbol{t}_n$ to form $U_{n,j}^k$ in step 1. These values are then used to compute $S_j^{j+1} f(\boldsymbol{t}, \boldsymbol{U}_n^k)$ in step 2.

Note that in each iteration, two SDC sweeps are performed, one on the coarse nodes $\tilde{\boldsymbol{t}}_n$ corresponding to $\mathcal{G}$ and one on the fine nodes $\boldsymbol{t}_n$ corresponding to $\mathcal{F}$. However, data is only passed between processors once per iteration after $\mathcal{G}$ is completed. The use of the coarse nodes for $\mathcal{G}$ reduces the serial cost of computing the first predictor in contrast to the pipelined SDC methods from [33; 16] described in Section 4.1.

5. Parallel speedup and efficiency

In this section, an analysis of the theoretical parallel speedup and efficiency of the parareal and parareal/SDC methods is presented. First, the standard analysis of the parareal method is reviewed which shows that the parallel efficiency cannot exceed $1/K$, for K iterations of the method. Next, it is demonstrated that the parallel efficiency of the parareal/SDC method can be much closer to 1.

The application of the $\mathcal{G}$ corrector sweep described by (18) in the parareal method is generally regarded as a serial operation since $\boldsymbol{P}_n$ cannot apply the correction step described by (18) until the value U_n^{k+1} is computed on $\boldsymbol{P}_{n-1}$ and sent to $\boldsymbol{P}_n$. The $\mathcal{G}$ propagator is often assumed to be used for the initial prediction phase as well which is clearly a serial operation, hence parareal is often described as iteratively applying the $\mathcal{G}$ propagator in serial and the $\mathcal{F}$ propagator in parallel. The theoretical parallel speedup and efficiency of the parareal method from this point of view has been studied previously by [6; 7; 22] and this analysis is first reviewed below. Then, an analysis for a pipelined version of parareal will be discussed followed by a similar analysis for a parareal/SDC method.

5.1. ***Serial-parallel parareal.*** To begin, assume that each processor is identical and that the communication time between processors is negligible. Denote the time for a processor to compute one step of the numerical method used in the $\mathcal{G}$ propagator by τ_G. Likewise let τ_F denote the time for one processor to compute one step of the numerical method used as the $\mathcal{F}$ propagator. Since multiple steps of a numerical method can be used for either $\mathcal{G}$ or $\mathcal{F}$, denote the number of steps as N_G and N_F respectively, and the size of the steps by Δt and δt. Hence the total cost of $\mathcal{F}$ is $N_F \tau_F$, which is denoted Υ_F. We assume that the cost of applying the $\mathcal{G}$ correction in (18) is equal to $N_G \tau_G = \Upsilon_G$, that is, the cost of forming the difference

N	Number of processors
K	Number of parareal iterations
τ_G	Cost of the numerical method in $\mathcal{G}$
τ_F	Cost of the numerical method in $\mathcal{F}$
T	Length of time integration
ΔT	Time increment per processor (T/N)
Δt	Time increment for $\mathcal{G}$ propagator
δt	Time increment for $\mathcal{F}$ propagator
N_G	Number of $\mathcal{G}$ steps per processor $(\Delta T/\Delta t)$
N_F	Number of $\mathcal{F}$ steps per processor $(\Delta T/\delta t)$
Υ_G	Total cost $\mathcal{G}$ propagator $(N_G\tau_G)$
Υ_F	Total cost $\mathcal{F}$ propagator $(N_F\tau_F)$

Table 1. Notation for the parareal algorithm.

in (18) is negligible. Let N denote the number of processors used, and $\Delta T = T/N$ the time increment per processor. Table 1 summarizes these definitions.

Assume that the prediction step is done with the $\mathcal{G}$ propagator, then for N processors the cost of computing the predictor is $NN_G\tau_G = N\Upsilon_G$. Likewise the cost of each iteration of the parareal method is $NN_G\tau_G + N_F\tau_F = N\Upsilon_G + \Upsilon_F$ (assuming that the method ends with a correction step). A graphical description of the cost is shown in Figure 2. The dots in the figure indicate communication between two processors.

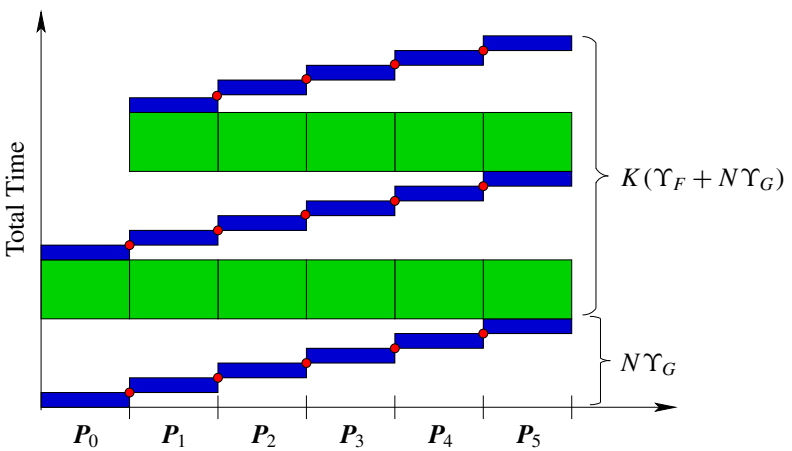

Figure 2. Cost of the serial-parallel version of the parareal method for $K = 2$ iterations and $N = 6$ processors. The dots indicate communication between two processors.

The total cost for the predictor and K parareal iterations as implemented in Figure 2 is hence

$$N N_G \tau_G + K(N N_G \tau_G + N_F \tau_F) = N\Upsilon_G + K(N\Upsilon_G + \Upsilon_F). \tag{26}$$

The cost of applying $\mathcal{F}$ serially is $N N_F \tau_F = N\Upsilon_F$; hence the speedup for parareal is

$$S = \frac{N\Upsilon_F}{N\Upsilon_G + K(N\Upsilon_G + \Upsilon_F)} = \frac{1}{\frac{\Upsilon_G}{\Upsilon_F} + K\left(\frac{\Upsilon_G}{\Upsilon_F} + \frac{1}{N}\right)}. \tag{27}$$

Denoting the ratio Υ_G/Υ_F by α we have

$$S = \frac{1}{\alpha + K(\alpha + 1/N)}. \tag{28}$$

In [6], some further assumptions are made to simplify the computation of a maximum possible speedup. Since we would like to compare the parareal solution with a serial fine solution with some fixed time step δt, N and N_F are related by

$$N_F = \Delta T/\delta t = \frac{T}{N\delta t}. \tag{29}$$

Assume further that the same method is used for both the fine and coarse propagator, and only one step is used for the coarse propagator; that is, $N_G = 1$ and $\tau_G = \tau_F$. Then

$$\alpha = \frac{1}{N_F} = \frac{N\delta t}{T}. \tag{30}$$

Under these additional assumptions, the speedup becomes

$$S = \frac{1}{(K+1)\frac{N\delta t}{T} + \frac{K}{N}}. \tag{31}$$

The maximum value of S in terms of N can hence be easily seen to occur when $N = N^*$, where

$$N^* = \sqrt{\frac{KT}{(K+1)\delta t}}. \tag{32}$$

This value of N^* gives a maximum speedup of S^* where

$$S^* = \frac{1}{2}\sqrt{\frac{T}{\delta t K(K+1)}}. \tag{33}$$

If one considers the parallel efficiency $E = S/N$, using (28) gives

$$E = \frac{1}{N\alpha(K+1) + K}. \tag{34}$$

Clearly the parallel efficiency is bounded by $E < 1/K$, which must be true since each parareal iteration is more expensive than computing the serial solution over the interval represented on each processor (i.e., ΔT). Since K is usually at least 2, it is often stated that the parallel efficiency of parareal is bounded by $\frac{1}{2}$. Using the value of N^* above, we see that the parallel efficiency at maximum speedup is $1/(2K)$.

5.2. ***Pipelined parareal.*** The version of the parareal algorithm just discussed is not the most efficient implementation possible on most current parallel architectures in which processors are able to perform different tasks at different times. Although the predictor step in parareal is certainly a serial operation, after $\boldsymbol{P}_n$ has computed the value U_n^0, it is free to immediately apply the $\mathcal{F}$ propagator rather than wait for the predictor to be computed on all processors. We refer to such an implementation where a processor computes a step in the algorithm as soon as possible as "pipelined". Likewise, if each processor begins computing the $\mathcal{G}$ propagator as soon as the initial condition is available, the parareal iterations can be pipelined so that the cost is of each parareal iteration is $N_F\tau_F + N_G\tau_G = \Upsilon_F + \Upsilon_G$ instead of $\Upsilon_F + N\Upsilon_G$. Figure 3 demonstrates graphically the cost of a pipelined parareal implementation. Note that this form of pipelining must be modified if the computation of the predictor is less expensive than applying the $\mathcal{G}$ corrector sweep.

The parallel speedup for pipelined parareal is

$$S = \frac{NN_F\tau_F}{NN_G\tau_G + K(N_G\tau_G + N_F\tau_F)} = \frac{N\Upsilon_F}{N\Upsilon_G + K(\Upsilon_G + \Upsilon_F)}. \tag{35}$$

Introducing $\alpha = \Upsilon_G/\Upsilon_F$ as above gives a parallel speedup of

$$S = \frac{1}{\alpha + K\left(\frac{\alpha}{N} + \frac{1}{N}\right)}. \tag{36}$$

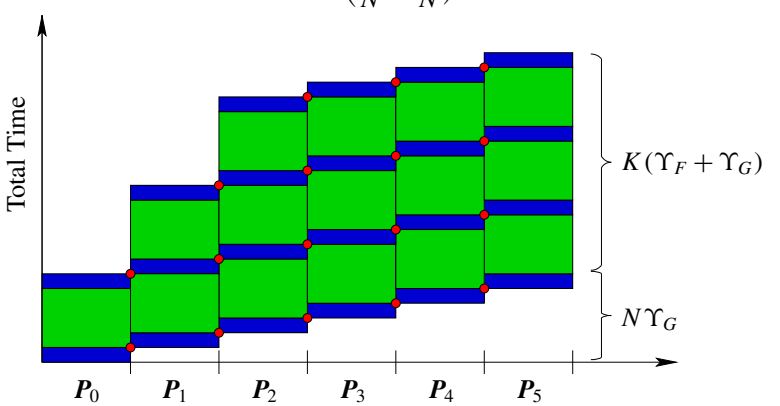

Figure 3. Cost of the pipelined version of the parareal method for $K = 3$ iterations and $N = 6$ processors. The dots indicate communication between two processors.

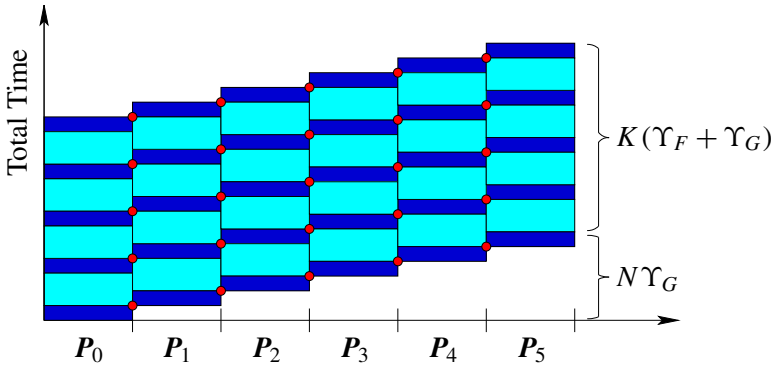

Figure 4. Cost of the hybrid parareal/SDC method for $K = 4$. The dots indicate communication between two processors.

Again making the simplifying assumptions from [6] of $N_G = 1$ and $\tau_G = \tau_F$, and hence $\alpha = N\delta t/T$, gives

$$S = \frac{1}{\frac{N\delta t}{T} + K\left(\frac{\delta t}{T} + \frac{1}{N}\right)}. \tag{37}$$

which is maximized by

$$N^* = \sqrt{KT/\delta t}. \tag{38}$$

This value is a factor of $\sqrt{K+1}$ larger than the value in (32), meaning that more processors can be used before the speedup saturates. This value of N^* gives a maximum speedup of

$$S^* = \frac{1}{2}\sqrt{\frac{T}{\delta t K}}\left(\frac{1}{1+\sqrt{K\delta t/T}}\right). \tag{39}$$

Comparing (33) and (39) shows that pipelining increases the maximum speedup by approximately a factor of $\sqrt{K+1}$ when $\delta t/T$ is small.

The parallel efficiency for pipelined parareal is

$$E = \frac{1}{N\alpha + K(\alpha+1)} = \frac{1}{\alpha(N+K)+K}. \tag{40}$$

Hence despite the lower cost of each parareal iteration, $E < 1/K$ since the cost of each iteration is still greater than the serial cost of computing the fine solution over the interval of length ΔT. In the first numerical tests presented in Section 6, for the tolerances used, K is larger than 10, which unfortunately gives a low parallel efficiency. Note that the bound $E < 1/K$ holds regardless of the assumptions made about the relative cost of τ_G because in the parareal iteration, $\mathcal{F}$ is computed during every iteration.

5.3. ***Parareal with SDC.*** In order to examine the speedup and efficiency of the parareal/SDC hybrid method, the analysis and notation above must be modified slightly. First we assume again that parareal/SDC has been implemented in a pipelined fashion as in Figure 3. We assume that the cost of computing one substep of SDC with $\mathcal{F}$ or $\mathcal{G}$ is the same as one step of $\mathcal{F}$ or $\mathcal{G}$ for an ODE (i.e., the cost of additional residual term in the correction equation is negligible). In the implementation used for the numerical results presented here, the parareal/SDC method will converge to an SDC method that uses one time step per processor. Of relevance to the cost is the number of substeps used in the fine SDC method, and this is denoted as in Section 4.3 as J (which is the number of SDC nodes used minus one). Let τ_F denote the cost of the method used for each substep in the correction step of SDC on the fine nodes. Likewise, let τ_G and $\tilde{J}$ be the corresponding constants for the $\mathcal{G}$ propagator. The total cost of the computing $\mathcal{F}$ is $J\tau_F$, which is again denoted Υ_F. The corresponding cost for $\mathcal{G}$ is $\Upsilon_F = \tilde{J}\tau_G$. Table 2 summarizes this notation.

N	Number of processors
K	Number of parareal iterations
M	Number of SDC iterations for a serial method
τ_G	Cost of the numerical method in $\mathcal{G}$
τ_F	Cost of the numerical method in $\mathcal{F}$
$\tilde{J}$	Number of substeps for coarse SDC sweep in $\mathcal{G}$
J	Number of substeps for fine SDC sweep in $\mathcal{F}$
Υ_G	Total cost $\mathcal{G}$ propagator ($\tilde{J}\tau_G$)
Υ_F	Total cost $\mathcal{F}$ propagator ($J\tau_F$)

Table 2. Notation for the parareal/SDC algorithm.

Assuming the method is pipelined, each parareal/SDC iteration will have a cost $J\tau_F + \tilde{J}\tau_G = \Upsilon_F + \Upsilon_G$. If the method used for the predictor also has cost per time step Υ_G, then the total cost for K iterations of parareal/SDC is $N\Upsilon_G + K(\Upsilon_F + \Upsilon_G)$.

Let M denote the number of SDC iterations needed to compute the solution to the desired accuracy on a single processor using the fine SDC nodes. Then the cost of the serial SDC method will be approximately $NM\Upsilon_F$. The parallel speedup S is hence

$$S = \frac{NM\Upsilon_F}{N\Upsilon_G + K(\Upsilon_G + \Upsilon_F)}. \tag{41}$$

Note that this is exactly M times greater than the value for the speedup for the pipelined parareal method in (35). Proceeding as in the pipelined parareal analysis above would lead to the same value for N^* and a value of S^* which is again M times bigger than before.

Setting $\alpha = \Upsilon_G/\Upsilon_F$ in (41) gives

$$S = \frac{M}{\alpha + (K/N)(\alpha+1)}, \tag{42}$$

and efficiency

$$E = \frac{M}{N\alpha + K(\alpha+1)} = \frac{M}{(N+K)\alpha + K}. \tag{43}$$

Again, the efficiency is exactly M times greater than the value for the pipelined parareal method in (40). This is due to the fact that the cost of doing M iterations of the SDC method has been amortized over the cost of the parareal iterations. In contrast to the standard parareal method, the efficiency is not automatically bounded above by $E < 1/K$. However, since M is fixed, for the efficiency to approach M/K, the product $(N+K)\alpha$ should be as small as possible. Unfortunately, for ODEs decreasing α is equivalent to decreasing the accuracy of $\mathcal{G}$, and this leads in general to an increase in K as is demonstrated in Section 6.

However for PDEs, if $\mathcal{G}$ could be computed on a coarser spatial mesh, the cost τ_G could be reduced significantly. This idea has in fact been suggested for the parareal method as well [8; 7; 23; 26]. In the parareal method however, reducing the cost of τ_G cannot increase the parallel efficiency beyond the bound $E < 1/K$ since $\mathcal{F}$ is still computed in every iteration.

5.4. ***Further discussion.*** Several comments can be made about the parallel cost and efficiency of parareal/SDC. First, it should be noted that achieving a large parallel speedup is only significant if the serial method to which one is comparing is efficient. Serial SDC methods have already been shown to be competitive in terms of cost per accuracy with higher-order Runge–Kutta and linear multistep methods [51; 43]. A comparison for explicit SDC methods and Runge–Kutta is also included in Section 6.

Secondly, in order for parareal/SDC to converge to the accuracy of the serial SDC method, K must be greater than $M/2$ since only $2K$ correction sweeps are done for K parareal iterations. In practice, since it is desirable to make α small (i.e., $\mathcal{G}$ much less expensive than $\mathcal{F}$), then K must be closer to M to achieve the accuracy of M serial iterations of SDC. If one examines the efficiency when $M = K$, then

$$E = \frac{1}{\left(\frac{N}{M}+1\right)\alpha + 1}. \tag{44}$$

Although this still seems to scale poorly as N grows larger, it should be noted that for the solution of PDEs (which motivates this work), temporal parallelization would be combined with spatial parallelization and hence N would be chosen so that the gain in efficiency from temporal parallelization exceeds the (presumably

saturated) spatial parallelization. Also, the processors in time parallelization can be used in a cyclical nature. Rather than dividing the entire time domain by the number of processors available, the processors are employed on the first N time-steps, and as the first processor finishes, it is employed to begin iterating on the $N+1$ time step as in the parallel deferred correction methods [33; 16].

As previously mentioned, a very promising approach is to use a coarser spatial grid for $\mathcal{G}$ as is discussed in [8; 7; 23; 26]. For three-dimensional calculations, even a factor of two reduction in grid spacing results in a factor of eight reduction in the number of spatial unknowns. In this scenario, the quantity α could be quite small, and the efficiency could theoretically exceed 1/2 in contrast to parareal where the efficiency is bounded by $1/K$ regardless of the cost of $\mathcal{G}$. Investigation of parareal/SDC using spatial coarsening for PDEs will be reported in a subsequent paper.

Finally, the above discussion ignores the reality that current massively parallel computers typically exhibit highly inhomogeneous communication costs between processors, and in the case of large scale grid computing, processors are not necessarily of comparable computational power. Even in a serial computation, when iterative methods are employed to solve the implicit equations associated with methods for stiff equations, the amount of work per time step can vary greatly. Furthermore, the issue of time-step adaptivity, which is critical for the efficient solution of many problems, causes further difficulties in analyzing the parallel efficiency of the time-parallel methods.

6. Numerical examples

In this section, preliminary numerical results are presented to compare the efficiency of the parareal/SDC method to standard implementations of parareal using Runge–Kutta methods for both the $\mathcal{F}$ and $\mathcal{G}$ propagators. The problems considered are taken from recent papers on parareal and focus on the effect of using SDC sweeps for the $\mathcal{F}$ and $\mathcal{G}$ propagator on the convergence of the parareal iterations.

6.1. ***The Lorenz oscillator.*** Here the effectiveness of the hybrid parareal/SDC method is explored using the Lorenz equation test problem from [27]. Specifically, we consider

$$x' = \sigma(y - x), \quad y' = x(\rho - z) - y, \quad z' = xy - \beta z. \tag{45}$$

in $t \in [0, 10]$ with the usual choice of parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$ and initial conditions $(x, y, z)(0) = (5, -5, 20)$. The resulting solution with these parameters is very sensitive to numerical error.

The particular implementation of parareal considered in [27] uses use $N = 180$ processors and the standard explicit fourth-order RK method for both $\mathcal{G}$ and $\mathcal{F}$, where

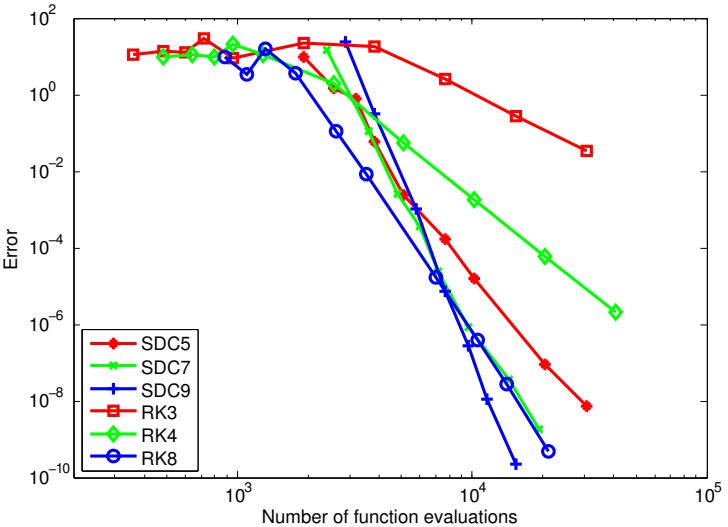

Figure 5. Error versus number of function evaluations: serial Runge–Kutta and SDC methods for the Lorenz equation.

$\mathcal{G}$ is implemented as a single step and $\mathcal{F}$ 80 steps. Here, additional implementations are considered using the standard explicit third-order RK method for $\mathcal{G}$ and explicit third-, fourth-, and eighth-order [20] RK methods (denoted here RK3, RK4, and RK8 respectively) as the $\mathcal{F}$ propagators. The convergence and efficiency of different combinations of RK methods will be compared to parareal/SDC methods using a single explicit correction sweep of SDC as the $\mathcal{F}$ propagator with five, seven, and nine Gauss–Lobatto nodes for each processor.

Before studying the parallel methods, the error for serial methods is first examined to understand the accuracy of different choices. Figure 5 compares the L_2 error of the solution at the final time $T = 10$ versus the number of function evaluations using serial Runge–Kutta and SDC methods. In the SDC methods, the RK4 method is used to compute the provisional solution at each node, and a second-order RK method is applied to the correction (2) during the deferred correction sweeps. Two, three, and four SDC sweeps respectively, are applied for the SDC methods using five, seven, and nine Lobatto nodes.

Note that the formal order of accuracy of the three SDC methods if the SDC iterations are fully converged to the collocation method would be 8, 12 and 16, but in this example, the number of SDC iterations used limits the formal order of accuracy to 8, 10, and 12. Here the computational cost of the SDC method ignores the cost of computing the numerical quadrature for the correction equations (which is done using a simple matrix-vector multiply). Figure 5 demonstrates that the numerical approximations for the SDC methods are more efficient than the third-

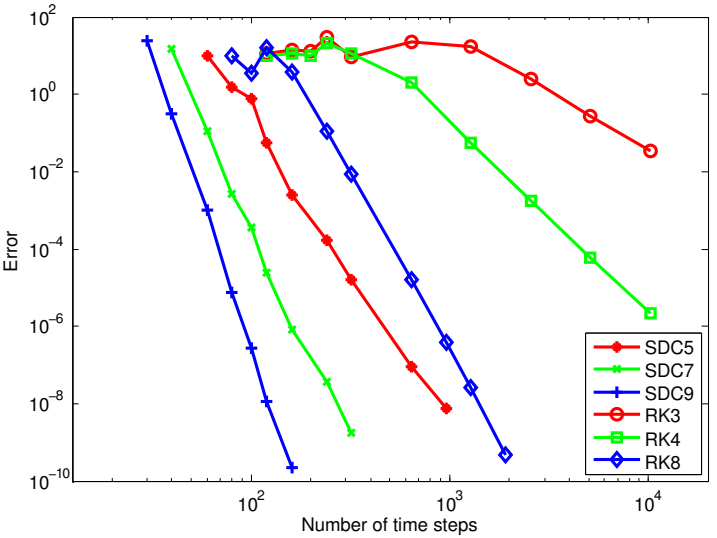

Figure 6. Error versus time step: serial Runge–Kutta and SDC methods for the Lorenz equation.

and fourth-order RK methods for a very modest error tolerance. For more stringent error tolerances, the efficiency of the SDC methods using seven and nine Lobatto nodes are similar to the eighth-order RK method.

In Figure 6 the error versus time step of the methods is compared. In this figure, the cost per time step of the methods is not relevant, and it is evident that the SDC methods are able to obtain the same level of accuracy as the RK methods with a much larger time step. This fact will allow a substantial increase in the efficiency of the parareal/SDC methods considered below since the increased cost per time step is amortized over parareal iterations.

Next the behavior of various implementations of the parareal and parareal/SDC methods in terms of the convergence of the parareal iterations is examined. First, the traditional parareal method using RK is examined. In Figure 7, the error at the final time is plotted versus parareal iteration for six different combinations of the $\mathcal{G}$ and $\mathcal{F}$ propagators. Two choices for $\mathcal{G}$ are used (one step of RK3 or RK4), while three choices for $\mathcal{F}$ are used (100 steps of RK3, 80 steps of RK4, or 8 steps of RK8). In each case the $\mathcal{G}$ propagator is used for the initial serial prediction step. Note that the convergence behavior for methods using the same $\mathcal{G}$ propagator are very similar. On the other hand, the overall error after convergence of parareal depends on the accuracy of the $\mathcal{F}$ propagator. Note that the absolute accuracy of the numerical method is used in the plots, rather than the difference between the parareal iterates and the result obtained from using $\mathcal{F}$ in serial. Hence the leveling off of the error in the convergence plots gives an indication of the accuracy of the serial $\mathcal{F}$ method.

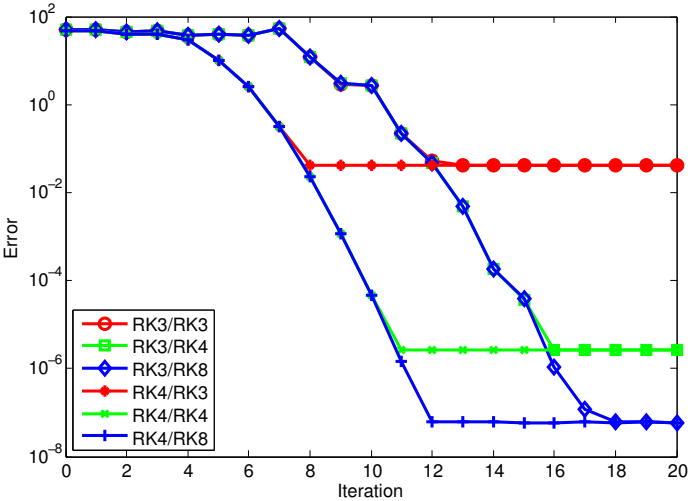

Figure 7. Convergence of parareal iterations for the Lorenz equation using different Runge–Kutta combinations for $\mathcal{G}$ and $\mathcal{F}$. In the legend, the combinations are listed as $\mathcal{G}/\mathcal{F}$.

The convergence results for parareal/SDC are presented in Figure 8. Again, six variations are presented. As before $\mathcal{G}$ is either RK3 or RK4, although these methods

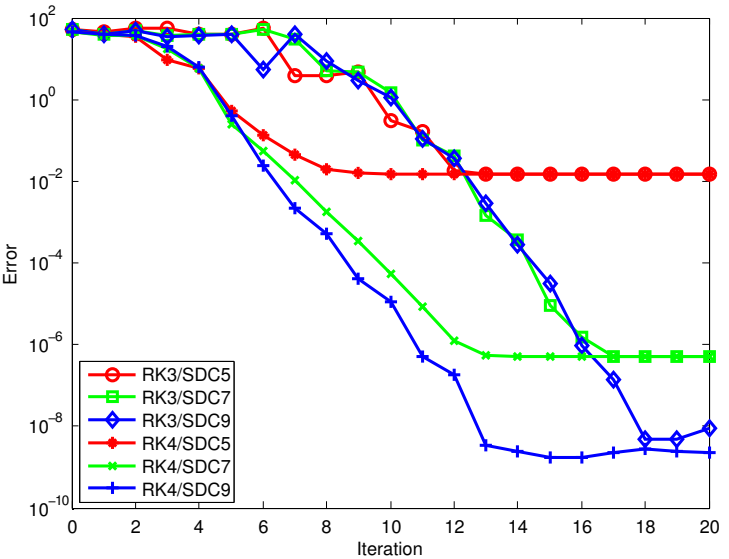

Figure 8. Convergence of parareal/SDC iterations for the Lorenz equation using different combinations of Runge–Kutta for $\mathcal{G}$ and a second-order SDC sweep for $\mathcal{F}$ with different number of nodes. In the legend, the combinations are listed as $\mathcal{G}/\mathcal{F}$.

are now applied to the correction (2). The $\mathcal{F}$ propagator is done using a single sweep of (2) using the first order method in (7) for each node. This means that $\mathcal{F}$ requires only 4, 6, and 8 function evaluations for the methods using 5, 7, and 9 Lobatto nodes respectively. As in Figure 7, the data in Figure 8 show that the number of parareal iterations required to converge depends on $\mathcal{G}$ while the overall accuracy depends on $\mathcal{F}$ (here the number of Lobatto nodes).

Next the convergence behavior of traditional parareal and parareal/SDC methods is compared in Figure 9. Of interest is whether the use of a low-order corrector for $\mathcal{F}$ in the parareal/SDC method adversely affects the number of iterations required for convergence. The left panel in Figure 9 shows the convergence for methods using RK3 in $\mathcal{G}$ and different choices of RK and SDC for $\mathcal{F}$. The right panel shows the corresponding data using RK4 for $\mathcal{G}$. The data demonstrate that replacing the full accuracy RK solve in $\mathcal{F}$ with a single SDC sweep does not significantly affect the convergence of the parareal iterates for this example. As observed above, the convergence behavior depends mainly on the accuracy of $\mathcal{G}$ for both methods and the number of iterations needed to converge to a given tolerance is very similar parareal and parareal/SDC methods.

Of greater interest here however is the total parallel computational cost of the parareal and parareal/SDC methods. Hence Figure 10 shows the convergence of parareal iterations versus total parallel cost for traditional parareal methods using

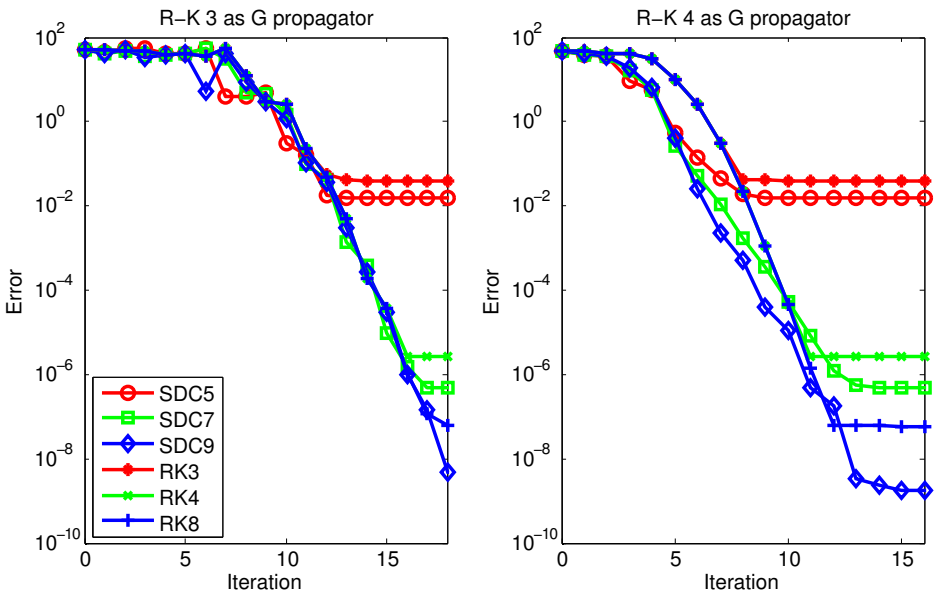

Figure 9. Convergence of parareal and parareal/SDC methods for the Lorenz equation. The methods in the left panel use explicit RK3 for $\mathcal{G}$, while those in the right panel use explicit RK4.

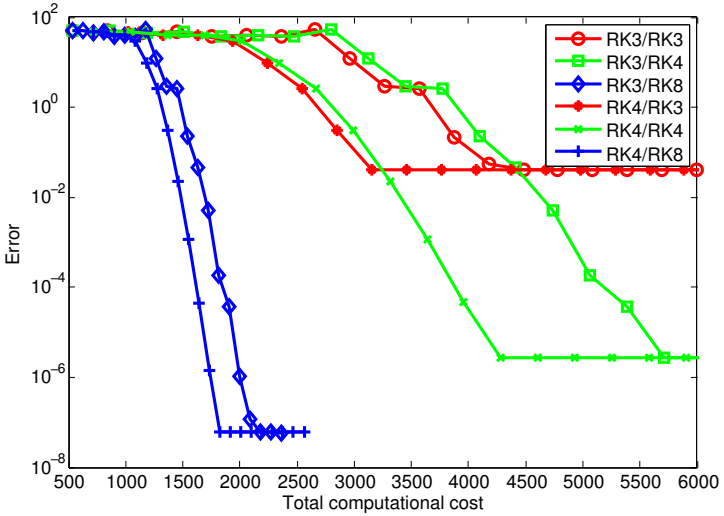

Figure 10. Error versus parallel computational cost of the parareal method using different combinations of Runge–Kutta for $\mathcal{G}$ and $\mathcal{F}$ (listed as $\mathcal{G}/\mathcal{F}$ in the legend) for the Lorenz equation.

RK methods for $\mathcal{G}$ and $\mathcal{F}$. The cost is computed using the assumption of a pipelined implementation as explained in Section 5.2 and is based on the number of explicit function evaluations: communication cost is ignored. Figure 10 shows that the total parallel cost is dominated by the cost of $\mathcal{F}$. Although using RK4 for $\mathcal{G}$ rather than RK3 reduces the number of iterations required somewhat, this is offset by the increased cost of RK4. Since the use of higher-order methods for $\mathcal{F}$ is more efficient in terms of accuracy per functions evaluations (see Figure 5), the total parallel cost for methods using RK8 for $\mathcal{F}$ is substantially less than those using RK3 and RK4.

Next consider the parallel cost for parareal/SDC shown in Figure 11. Again a pipelined implementation is assumed, and since only function evaluations are counted, the cost of the computing the numerical quadrature and interpolating the correction computed by $\mathcal{G}$ (both of which are simple matrix multiplications) is not included. The most notable difference in the data is that the total computational cost of the parareal/SDC method is dominated by the cost of the initial serial $\mathcal{G}$ sweep. As noted before, this fact suggests that using spatial coarsening for $\mathcal{G}$ for PDEs could increase the efficiency of parareal/SDC methods significantly.

Finally, the total parallel cost for both traditional parareal using RK and parareal/SDC is compared in Figure 12. Since the methods use the same predictor, the cost is identical at the end of the serial predictor step, but it is evident that the much reduced cost of using SDC for $\mathcal{F}$ greatly reduces the total computational cost. Specifically, the cost of $\mathcal{F}$ for the parareal/SDC method is either 4, 6, or 8 function

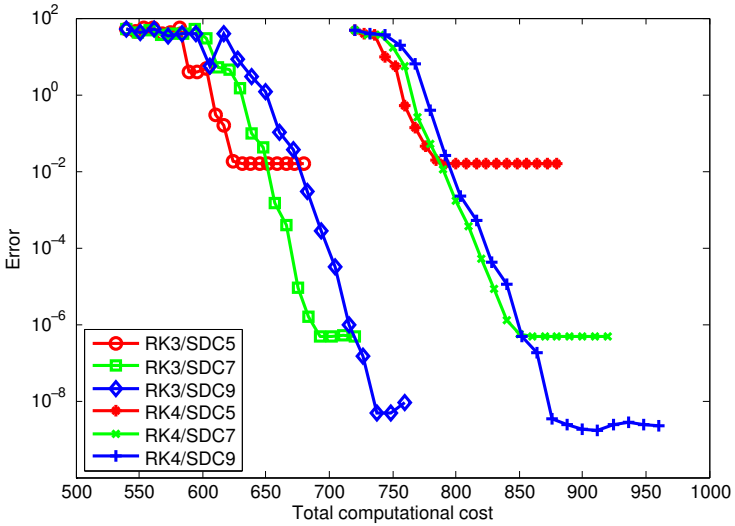

Figure 11. Error versus parallel computational cost of the parareal/SDC method using different combinations of $\mathcal{G}$ and $\mathcal{F}$ (listed as $\mathcal{G}/\mathcal{F}$ in the legend) for the Lorenz equation.

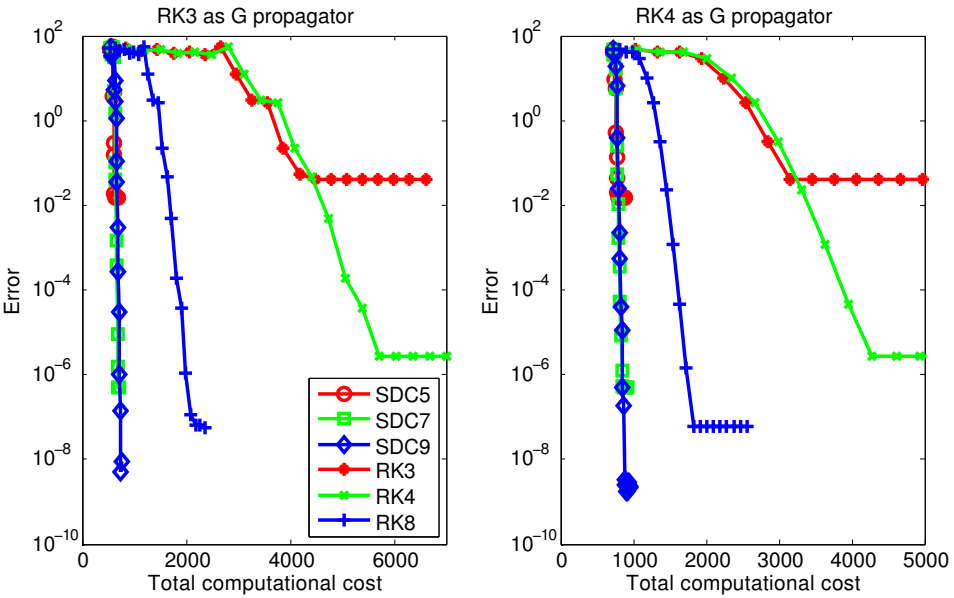

Figure 12. Comparison of total parallel cost for parareal and parareal/SDC methods for the Lorenz equation.

evaluations while for the RK based parareal method, it is 300, 320, or 88 function evaluations.

6.2. ***A PDE example.*** Next, the performance of the parareal/SDC method is demonstrated for a discretized partial differential equation, namely the viscous Burgers equation

$$u_t + uu_x = \nu u_{xx}, \quad u(x,0) = g(x).$$

The spatial domain for all experiment is the periodic interval [0, 1], with initial data given by $g(x) = \sin(2\pi x)$, and $\nu = 1/50$. This example is also considered in [27], although there a second-order finite difference spatial discretization with first-order backward Euler in time is used. Here, a semi-implicit or IMEX temporal discretization is used, and the finite difference discretization is replaced by a pseudo-spectral approach (with no de-aliasing). The semi-implicit time stepping requires only the diffusive piece to be treated implicitly; hence the implicit problem is linear (and here solved in spectral space).

For the parareal/SDC method, 7 Gauss–Lobatto nodes are used in the $\mathcal{F}$ propagator with a first-order semi-implicit corrector. For $\mathcal{G}$, either 1 or 2 steps of a first-order semi-implicit corrector is used. In each of the tests below, the error reported is the maximum of the error at the final time as compared to a highly resolved reference solution computed with Runge–Kutta and not the solution generated by using $\mathcal{F}$ in serial. Note that the reference solution is computing using the same number of spatial unknowns as the problem being run, so the solution to the PDE is not necessarily fully resolved in space.

In the first set of tests, 64 spatial grid points are used, and the convergence behavior for runs of different length of integration is compared. Specifically, the simulations are run to final time $T = 0.1$, $T = 0.5$, and $T = 1.0$. In all cases, the time interval for each processor is 1/100; hence 10, 50, and 100 processors are used. In Figure 13, the results are displayed in the left panel using one step of forward/backward Euler for $\mathcal{G}$ and in the right panel for two steps of forward/backward Euler. In these figures, one does not observe the exponential convergence of the error in the parareal iterations as is evident in the examples for the Lorenz equation. This is due to the fact that the convergence of the SDC iteration is slower in this example than the convergence of the parareal iterations for short time (Note in particular the nearly constant convergence rate for the $T = 0.1$ example). Note also in the right panel that the increase in the accuracy of $\mathcal{G}$ from using an additional step of forward/backward Euler increases the rate of convergence of the parareal/SDC iterations.

In the next set of tests, the integration time is fixed at $T = 1$ with 100 processors, but the number points used in the spatial discretization is varied. Figure 14 shows the convergence behavior for 64, 128, and 256 spatial points. The data show that the convergence behavior is completely unaffected by the number of spatial variables used.

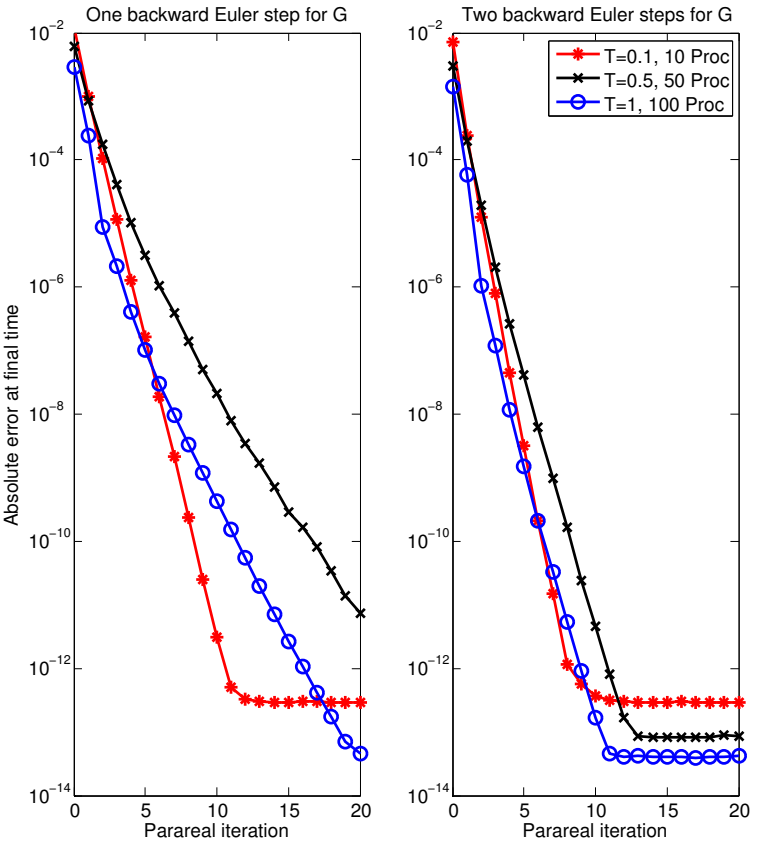

Figure 13. Convergence of the parareal/SDC method for Burgers equation using one step (left) and two steps (right) of forward/backward Euler for $\mathcal{G}$.

In the final set of tests, the integration time is fixed at $T = 1$ with a grid size of 64 spatial points, but the number of processors is varied. Figure 15 shows the convergence behavior for $N = 20$, 40 and 100 for 1 step of forward/backward Euler for $\mathcal{G}$ in the left panel, and 2 steps in the right panel. The fact that the parareal iterates are converging faster for a larger number of processors is again due to the increased accuracy of $\mathcal{G}$. As the number of processors is increased, the coarse time step gets smaller and hence $\mathcal{G}$ becomes more accurate. Hence the number of iterations needed to converge decreases.

A few words should be said regarding any comparison in cost to the method in [27]. First, the $\mathcal{F}$ propagator in [27] is based on 10 steps of backward Euler for the first set of test cases. Here, the $\mathcal{F}$ method is 6 substeps of a semi-implicit method applied in the SDC sweep. Depending on the efficiency of solving the nonlinear equation for the fully implicit method, the semi-implicit approach could

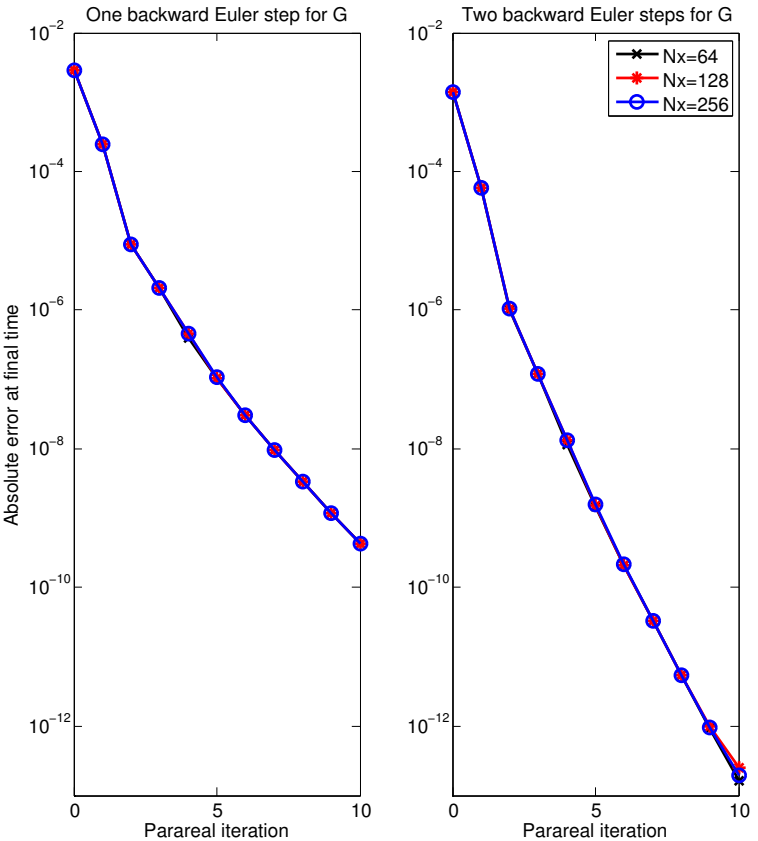

Figure 14. Parareal/SDC convergence for different spatial resolutions for Burgers equation. For all runs, the final time is $T = 1$, and 100 processors are used.

be substantially more efficient. The biggest difference between the two approaches is in the overall accuracy, which for the high-order SDC-based method used here is over ten orders of magnitude smaller than that achieved with the first-order temporal method used in [27].

7. Conclusions

A new strategy for combining deferred corrections and the parareal method for the temporal parallelization of ordinary differential equations first presented in [53] is further developed and evaluated. One can regard the parareal/SDC strategy as either a way to parallelize SDC methods in time or as a way to increase the efficiency of parareal methods by reducing the computational cost of the $\mathcal{F}$ propagator.

The motivation of this research is to develop parallel-in-time strategies to be combined with spatial parallelization for massively parallel computations of PDEs,

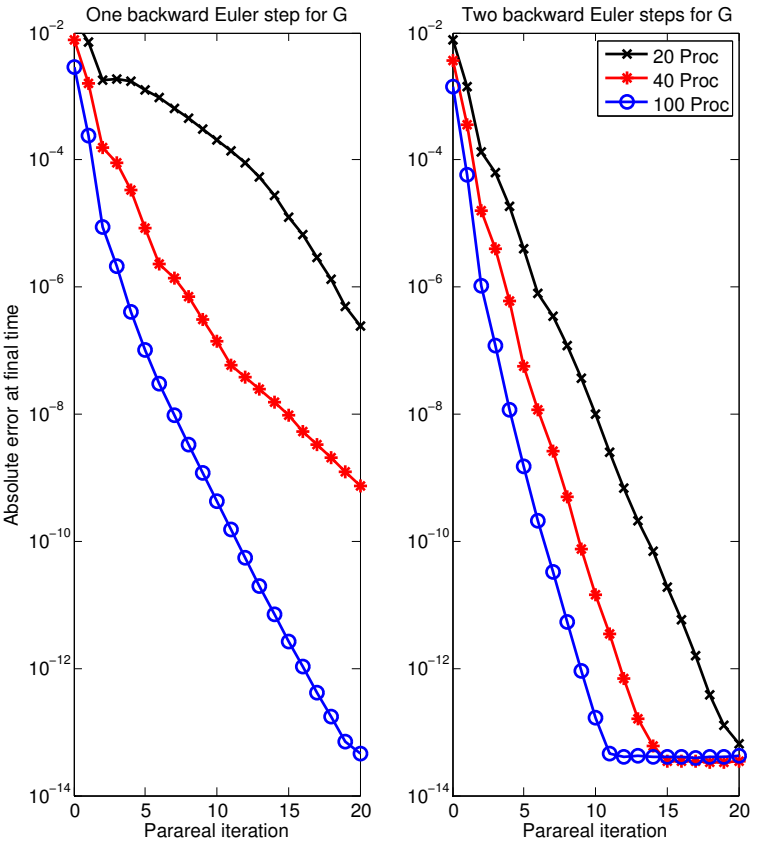

Figure 15. Parareal/SDC convergence for different numbers of processors. Each run is to the final time $T = 1$ with 64 spatial grid points.

particularly in computational fluid dynamics. One of the significant features of the parareal/SDC approach is that the greatly reduced cost of computing $\mathcal{F}$ means that reducing the cost of $\mathcal{G}$ by using a coarser spatial grid could increase the overall parallel efficiency significantly. Results in this direction will be reported in the future.

Other possibilities for further increasing the efficiency of the parareal/SDC approach include the use of Krylov methods to accelerate the convergence of the parareal/SDC iterations. Krylov acceleration methods have already been studied for serial SDC methods for both ODEs and DAEs [35; 36; 12], as well as for the traditional parareal method [28], although the effectiveness of these methods for large scale PDEs has not yet been demonstrated. Another possibility concerns the use of iterative solvers within implicit or semi-implicit temporal methods for PDEs. It seems reasonable that the error tolerance within implicit solvers could

be dynamically decreased as the parareal iterations progress, but this may affect the convergence of the parareal iterations. In the parareal/SDC approach, a very good initial guess for these implicit solves is available from the previous parareal iteration. In conclusion, despite the promising initial results reported here, avenues for further improvement need to be pursued.

Appendix: Pseudo-code of the parareal/SDC method

The following is the pseudocode for a semi-implicit parareal/SDC implementation using the first-order time-stepping method in (12) and (15).

Serial initialization:

FOR $n = 0 \ldots N-1$
  COMMENT: Get initial data
  IF $n = 0$
    $\tilde{U}^0_{1,1} = u(0)$
  ELSE
    Receive $\tilde{U}^0_{n-1,\tilde{J}}$ from $\boldsymbol{P}_{n-1}$
    Set $\tilde{U}^0_{n,1} = \tilde{U}^0_{n-1,\tilde{J}}$
  END IF
  COMMENT: Compute solution at coarse time nodes
  FOR $\tilde{j} = 0 \ldots \tilde{J}-1$
    $\tilde{U}^0_{n,\tilde{j}+1} = \tilde{U}^0_{n,\tilde{j}} + \Delta t_{\tilde{j}} \big(F_E(t_{\tilde{j}}, \tilde{U}^0_{n,\tilde{j}}) + F_I(t_{\tilde{j}+1}, \tilde{U}^0_{n,\tilde{j}+1}) \big)$
  END FOR
  COMMENT: Send data forward
  IF $n < N-1$
    Send $\tilde{U}^0_{n,\tilde{J}}$ to $\boldsymbol{P}_{n+1}$
  END IF
END FOR

Parallel iteration:

FOR $k = 1 \ldots K$
  DO in parallel on $\boldsymbol{P}_n$, $n = 0 \ldots N-1$
    COMMENT: Update values at fine time steps
    IF $k = 1$
      INTERPOLATE $\tilde{U}^k_{n,\tilde{j}}$ at coarse times to form $U^k_{n,j}$ at fine points
      COMPUTE $f(t_j, U^k_{n,j})$ for $j = 1 \ldots J$.
    ELSE
      INTERPOLATE $\tilde{U}^k_{n,\tilde{j}} - U^{k-1}_{n,\tilde{j}}$ at coarse times to form $U^k_{n,j}$ at fine points
    END IF

COMMENT: Compute time integral of fine solution at fine nodes
COMPUTE $S_j^{j+1} f(\boldsymbol{t}_n, \mathbf{U}_n^k)$ for $j = 0 \ldots J-1$ using spectral integration

COMMENT: Do a fine SDC sweep
FOR $j = 0 \ldots J-1$
$U_{n,j+1}^k = U_{n,j}^k + \Delta t_j \big(F_E(t_j, U_{n,j}^k) + F_I(t_{j+1}, U_{n,j+1}^k)\big) + S_j^{j+1} f(\boldsymbol{t}_n, \mathbf{U}_n^k)$
END FOR

COMMENT: Compute time integral of fine solution at coarse nodes
COMPUTE $S_{\tilde{j}}^{\tilde{j}+1} f(\tilde{\boldsymbol{t}}_n, \mathbf{U}_n^k)$ $\tilde{j} = 0 \ldots \tilde{J}-1$ by summing $S_j^{j+1} f(\boldsymbol{t}_n, \mathbf{U}_n^k)$

COMMENT: Get new initial data
IF $n = 0$
SET $U_{0,1}^k = \tilde{U}_{0,1}^{k-1}$,
SET $f(t_n, U_{0,1}^k) = f(t_n, \tilde{U}_{0,1}^{k-1})$
ELSE
RECEIVE $\tilde{U}_{n-1,\tilde{J}}^k$ from $\boldsymbol{P}_{n-1}$
SET $U_{n,1}^0 = \tilde{U}_{n-1,\tilde{J}}^k$
COMPUTE $f(t_n, U_{n,1}^k) = f(t_n, U_{n,1}^k)$
END IF

COMMENT: Do a coarse SDC sweep
FOR $\tilde{j} = 0 \ldots \tilde{J}-1$
$\tilde{U}_{n,\tilde{j}+1}^k = \tilde{U}_{n,\tilde{j}}^k + \Delta t_{\tilde{j}} \big(F_E(t_{\tilde{j}}, \tilde{U}_{n,\tilde{j}}^k) + F_I(t_{\tilde{j}+1}, \tilde{U}_{n,\tilde{j}+1}^k)\big) + S_{\tilde{j}}^{\tilde{j}+1} f(\tilde{\boldsymbol{t}}_n, \tilde{U}_n^k)$
END FOR

COMMENT: Send data forward
IF $n < P-1$
Send $U_{n,\tilde{J}}^k$ to $\boldsymbol{P}_{n+1}$
END IF
END DO
END FOR

References

[1] G. Akrivis, M. Crouzeix, and C. Makridakis, *Implicit-explicit multistep finite element methods for nonlinear parabolic problems*, Math. Comp. **67** (1998), no. 222, 457–477. MR 98g:65088 Zbl 0896.65066

[2] ———, *Implicit-explicit multistep methods for quasilinear parabolic equations*, Numer. Math. **82** (1999), no. 4, 521–541. MR 2000e:65075 Zbl 0936.65118

[3] G. Akrivis and Y.-S. Smyrlis, *Implicit-explicit BDF methods for the Kuramoto–Sivashinsky equation*, Appl. Numer. Math. **51** (2004), no. 2-3, 151–169. MR 2005h:65126 Zbl 1057.65069

[4] 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

[5] U. M. Ascher, S. J. Ruuth, and B. T. R. Wetton, *Implicit-explicit methods for time-dependent partial differential equations*, SIAM J. Numer. Anal. **32** (1995), no. 3, 797–823. MR 96j:65076 Zbl 0841.65081

[6] G. Bal, *Parallelization in time of (stochastic) ordinary differential equations*, preprint, 2003, Available at http://www.columbia.edu/∼gb2030/PAPERS/paralleltime.pdf.

[7] 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

[8] 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. Pavarino and A. Toselli, eds.), Lect. Notes Comput. Sci. Eng., no. 23, Springer, Berlin, 2002, pp. 189–202. MR 1962689

[9] G. Bal and Q. Wu, *Symplectic parareal*, Domain decomposition methods in science and engineering XVII (U. Langer et al., eds.), Lect. Notes Comput. Sci. Eng., no. 60, Springer, Berlin, 2008, pp. 401–408. MR 2436107 Zbl 1140.65372

[10] 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

[11] E. L. Bouzarth, *Regularized singularities and spectral deferred correction methods: A mathematical study of numerically modeling Stokes fluid flow*, Ph.D. thesis, The University of North Carolina at Chapel Hill, 2008. MR 2711923

[12] S. Bu, J. Huang, and M. L. Minion, *Semi-implicit Krylov deferred correction methods for ordinary differential equations*, Proceedings of the 15th American Conference on Applied Mathematics, WSEAS, 2009, pp. 95–100.

[13] K. Burrage, *Parallel methods for ODEs*, Adv. Comput. Math. **7** (1997), no. 1-2, 1–31.

[14] J. C. Butcher, *Order and stability of parallel methods for stiff problems: Parallel methods for odes*, Adv. Comput. Math. **7** (1997), no. 1-2, 79–96. MR 98c:65105 Zbl 0884.65090

[15] M. P. Calvo, J. de Frutos, and J. Novo, *Linearly implicit Runge–Kutta methods for advection-reaction-diffusion equations*, Appl. Numer. Math. **37** (2001), no. 4, 535–549. MR 2002e:65114 Zbl 0983.65106

[16] A. Christlieb, C. Macdonald, and B. Ong, *Parallel high-order integrators*, SIAM J. Sci. Comput. **32** (2010), no. 2, 818–835.

[17] A. Christlieb, B. Ong, and J.-M. Qiu, *Integral deferred correction methods constructed with high order Runge–Kutta integrators*, Math. Comp. **79** (2010), no. 270, 761–783. MR 2600542 Zbl 05776244

[18] M. L. Crow and M. Ilic, *The parallel implementation of the waveform relaxation method for transient stability simulations*, IEEE Trans. Power Syst. **5** (1990), no. 3, 922–932.

[19] 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

[20] J. R. Dormand and P. J. Prince, *A family of embedded Runge–Kutta formulae*, J. Comput. Appl. Math. **6** (1980), no. 1, 19–26. MR 81g:65098 Zbl 0448.65045

[21] 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

[22] 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

[23] 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 (T. J. Barth et al., eds.), Lect. Notes Comput. Sci. Eng., no. 40, Springer, Berlin, 2005, pp. 433–440. MR 2235770 Zbl 02143574

[24] J. Frank, W. Hundsdorfer, and J. G. Verwer, *On the stability of implicit-explicit linear multistep methods*, Appl. Numer. Math. **25** (1997), no. 2-3, 193–205. MR 98m:65126 Zbl 0887.65094

[25] M. J. Gander, *A waveform relaxation algorithm with overlapping splitting for reaction diffusion equations*, Numer. Linear Algebra Appl. **6** (1999), no. 2, 125–145. MR 2000m:65110 Zbl 0983.65107

[26] ———, *Analysis of the parareal algorithm applied to hyperbolic problems using characteristics*, Bol. Soc. Esp. Mat. Apl. SẽMA **42** (2008), 21–35. MR 2009b:65268

[27] M. J. Gander and E. Hairer, *Nonlinear convergence analysis for the parareal algorithm*, Domain decomposition methods in science and engineering XVII (U. Langer et al., eds.), Lect. Notes Comput. Sci. Eng., no. 60, Springer, Berlin, 2008, pp. 45–56. MR 2009j:65165 Zbl 1140.65336

[28] M. J. Gander and M. Petcu, *Analysis of a Krylov subspace enhanced parareal algorithm for linear problems*, Paris-Sud Working Group on Modelling and Scientific Computing 2007–2008 (E. Cancès et al., eds.), ESAIM Proc., no. 25, EDP Sci., Les Ulis, 2008, pp. 114–129. MR 2010i:65119 Zbl 1156.65322

[29] M. J. Gander and A. E. Ruehli, *Optimized waveform relaxation methods for RC type circuits*, IEEE Trans. Circuits Syst. I Regul. Pap. **51** (2004), no. 4, 755–768. MR 2005d:94228

[30] M. J. Gander and S. Vandewalle, *Analysis of the parareal time-parallel time-integration method*, SIAM J. Sci. Comput. **29** (2007), no. 2, 556–578. MR 2008c:65386 Zbl 1141.65064

[31] L. Greengard, *Spectral integration and two-point boundary value problems*, SIAM J. Numer. Anal. **28** (1991), no. 4, 1071–1080. MR 92h:65033 Zbl 0731.65064

[32] D. Guibert and D. Tromeur-Dervout, *Adaptive parareal for systems of ODEs*, Domain decomposition methods in science and engineering XVI (O. Widlund and D. Keyes, eds.), Lect. Notes Comput. Sci. Eng., no. 55, Springer, Berlin, 2007, pp. 587–594. MR 2334151

[33] ———, *Parallel deferred correction method for CFD problems*, Parallel computational fluid dynamics: parallel computing and its applications (J. H. Kwon, A. Ecer, N. Satofuka, J. Periaux, and P. Fox, eds.), Elsevier, Amsterdam, 2007, pp. 131–138.

[34] M. Hu, K. Jackson, J. Janssen, and S. Vandewalle, *Remarks on the optimal convolution kernel for CSOR waveform relaxation: Parallel methods for odes*, Adv. Comput. Math. **7** (1997), no. 1-2, 135–156. MR 98c:65107 Zbl 0889.65069

[35] 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

[36] ———, *Arbitrary order Krylov deferred correction methods for differential algebraic equations*, J. Comput. Phys. **221** (2007), no. 2, 739–760. MR 2008a:65134 Zbl 1110.65076

[37] K. J. in 't Hout, *On the contractivity of implicit-explicit linear multistep methods*, Appl. Numer. Math. **42** (2002), no. 1-3, 201–212. MR 2003j:65065 Zbl 1001.65090

[38] A. Iserles and S. P. Nørsett, *On the theory of parallel Runge–Kutta methods*, IMA J. Numer. Anal. **10** (1990), no. 4, 463–488. MR 91i:65127

[39] 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

[40] M. Kiehl, *Parallel multiple shooting for the solution of initial value problems*, Parallel Comput. **20** (1994), no. 3, 275–295. MR 95c:65099 Zbl 0798.65079

[41] 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

[42] 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

[43] ———, *Implications of the choice of quadrature nodes for Picard integral deferred corrections methods for ordinary differential equations*, BIT **45** (2005), no. 2, 341–373. MR 2006h:65087 Zbl 1078.65552

[44] ———, *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

[45] 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

[46] Y. Liu and J. Hu, *Modified propagators of parareal in time algorithm and application to Princeton ocean model*, Internat. J. Numer. Methods Fluids **57** (2008), no. 12, 1793–1804. MR 2009g:86009 Zbl 05312577

[47] Y. Maday and G. Turinici, *Parallel in time algorithms for quantum control: Parareal time discretization scheme*, Int. J. Quantum Chem. **93** (2003), 223–228.

[48] Y. Maday, J. Salomon, and G. Turinici, *Monotonic parareal control for quantum systems*, SIAM J. Numer. Anal. **45** (2007), no. 6, 2468–2482. MR 2008k:81003 Zbl 1153.49005

[49] Y. Maday and G. Turinici, *The parareal in time iterative solver: a further direction to parallel implementation*, Domain decomposition methods in science and engineering (R. Kornhuber et al., eds.), Lect. Notes Comput. Sci. Eng., no. 40, Springer, Berlin, 2005, pp. 441–448. MR 2235771 Zbl 1067.65102

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

[51] ———, *Semi-implicit spectral deferred correction methods for ordinary differential equations*, Commun. Math. Sci. **1** (2003), no. 3, 471–500. MR 2005f:65085 Zbl 1088.65556

[52] ———, *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

[53] M. L. Minion and S. A. Williams, *Parareal and spectral deferred corrections*, Numerical analysis and applied mathematics (T. E. Simos, ed.), AIP Conference Proceedings, no. 1048, AIP, 2008, pp. 388–391.

[54] W. L. Miranker and W. Liniger, *Parallel methods for the numerical integration of ordinary differential equations*, Math. Comp. **21** (1967), 303–320. MR 36 #6155 Zbl 0155.47204

[55] J. Nievergelt, *Parallel methods for integrating ordinary differential equations*, Comm. ACM **7** (1964), 731–733. MR 31 #889 Zbl 0134.32804

[56] 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 Sci. Publ., Huntington, NY, 2001, pp. 269–288. MR 2005a:65065 Zbl 1018.65093

[57] 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

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

[59] 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, 1996, pp. 17–20.

[60] G. A. Staff and E. M. Rønquist, *Stability of the parareal algorithm*, Domain decomposition methods in science and engineering (R. Kornhuber et al., eds.), Lect. Notes Comput. Sci. Eng., no. 40, Springer, Berlin, 2005, pp. 449–456. MR 2235772

[61] S. Vandewalle and D. Roose, *The parallel waveform relaxation multigrid method*, Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing, Soc. Indust. Appl. Math., 1989, pp. 152–156.

[62] S. L. Wu, B. C. Shi, and C. M. Huang, *Parareal-Richardson algorithm for solving nonlinear ODEs and PDEs*, Commun. Comput. Phys. **6** (2009), no. 4, 883–902.

[63] P. E. 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 (G. Contopoulos, ed.), Proc. Int. Astron. Union Symp., no. 25, Academic Press, London, 1964.

[64] X. Zhong, *Additive semi-implicit Runge–Kutta methods for computing high-speed nonequilibrium reactive flows*, J. Comput. Phys. **128** (1996), no. 1, 19–31. MR 97e:80019 Zbl 0861.76057

Received January 18, 2010. Revised October 16, 2010.

MICHAEL L. MINION: minion@email.unc.edu
Department of Mathematics, University of North Carolina – Chapel Hill, Campus Box 3250, Chapel Hill, NC 27514-3250, United States
amath.unc.edu/Minion