A MULTIGRID-IN-TIME ALGORITHM FOR SOLVING EVOLUTION EQUATIONS IN PARALLEL

S. FRIEDHOFF†, R. FALGOUT‡, T. KOLEV‡, S. MACLACHLAN†, AND J. SCHRODER‡

Abstract. We consider optimal-scaling multigrid solvers for the linear systems that arise from the discretization of problems with evolutionary behavior. Typically, solution algorithms for evolution equations are based on a time-marching approach, meaning solving for one time step after the other. These traditional time integration techniques lead to optimal-scaling but not parallelizable algorithms. However, current trends in computer architectures are leading towards more, but slower, processors and, therefore, driving the need for greater parallelism. One approach to achieve parallelism in time is with multigrid, but while classical multigrid methods rely on multiscale representations in space, that arise naturally from decomposing a function into a hierarchy of frequencies from global smooth modes to local oscillations, these approaches do not extend to evolutionary variables in a straightforward manner, because of the fundamentally local structure of the evolution. In this paper, we present an optimal and scalable multigrid-in-time algorithm for diffusion equations as simple examples of evolution equations. Our algorithm is based on interpreting the parareal time integration method [13] as a two-level reduction scheme, and developing a multilevel algorithm from this viewpoint. We demonstrate optimality of our algorithm for solving the one-dimensional diffusion equation in numerical experiments. Furthermore, by using parallel performance models, we show that we can expect speedup in comparison to sequential time-marching on modern architectures.

1. Introduction. One of the major challenges facing the computational science community with future architectures is illustrated in Figure 1.1: although transistor counts are still growing according to Moore's Law, clock speeds are no longer increasing and core counts are going up sharply.

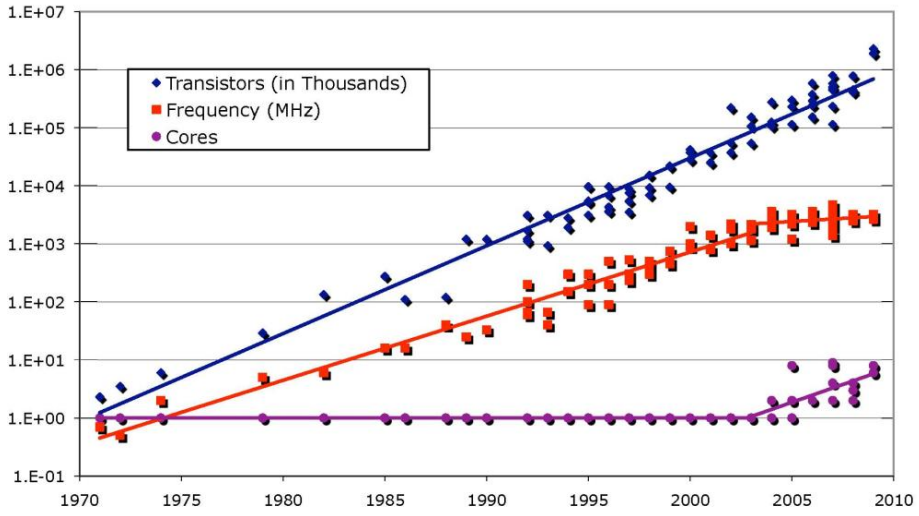

Fig. 1.1: From Kathy Yelick's talk titled "Ten Ways to Waste a Parallel Computer." Also published in [4], Figure 2.1. Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç.

As a consequence, traditional time marching is becoming a huge sequential bottleneck. Solving for multiple time steps in parallel and, therefore, increasing concurrency would remove this bottleneck. Because time is sequential by nature, the idea of simultaneously solving for multiple time steps is not intuitive. Yet it *is* possible, with work on this topic going back to as early as 1964 [17]. Other papers on this subject include [1, 6–11, 13–16, 22]. One approach to achieve paral-

†Department of Mathematics, Tufts University, 503 Boston Avenue, Medford, MA 02155. email: {stephanie.friedhoff, scott.maclachlan}@tufts.edu

‡Center for Applied Scientific Computation, Lawrence Livermore National Laboratory, P. O. Box 808, L-561, Livermore, CA 94551. email: {rfalgout, tzanio, schroder2}@llnl.gov. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-606952).

lelism in time are multigrid methods, but there are only a few known optimal multigrid algorithms such as [10, 21]. Furthermore, these methods are full space-time algorithms, whereas our algorithm employs a semi-coarsening strategy using coarsening only in the time dimension as discussed below.

While classical multigrid methods rely on multiscale representations in space, that arise naturally from decomposing a function into a hierarchy of frequencies from global smooth modes to local oscillations, these approaches do not extend to evolutionary variables in a straightforward manner. There are two approaches for extending classical multigrid methods to include the time dimension: multigrid only in time and space-time multigrid. In this paper, we present a multigrid-in-time algorithm that is based on interpreting the parareal time integration method [13] as a two-level reduction method. This non-traditional view of parareal allows us to develop an optimal-scaling multilevel algorithm, while exploiting the non-intrusiveness on existing codes from parareal; i. e., our multigrid-in-time algorithm simply calls an existing time-stepping routine. However, to achieve the full benefit of computing multiple time steps at once, space-time multigrid methods, where time is simply another dimension in the grid, have to be considered. This approach is more intrusive on existing codes and is a separate research topic not explored here.

This paper is organized as follows. First, in §2, we consider the connection of time integration methods to linear systems of equations. Based on this correspondence, we then describe the parareal algorithm. In §3, we introduce our multigrid-in-time algorithm. We start by showing how the parareal algorithm can be interpreted as a two-level reduction scheme, followed by a description of a multilevel algorithm developed from this viewpoint. Finally, in §4, we consider optimality and parallel performance of our multigrid-in-time algorithm. We demonstrate optimality of our algorithm for solving a parabolic model problem, the one-dimensional diffusion equation. Then, we use parallel performance models to show that we can expect speedup in comparison to sequential time-marching on future architectures, followed by a discussion in §5.

2. Time integration methods. Time integration methods for solving problems with evolutionary behavior are typically based on a time-marching approach. While these traditional techniques lead to optimal-scaling algorithms, they are not parallelizable. The parareal algorithm, introduced by Lions, Maday, and Turinici in [13], is a time integration method that does allow parallelism in the solution process. The name refers to the characteristic of the algorithm, namely using parallel real time computations to solve evolution problems that cannot be solved in real time using one processor only.

In §2.1, we consider the connection of time integration methods to the solution of linear systems of equations. This correspondence is the basis of our description of the parareal algorithm in §2.2. It is a non-traditional view of parareal but, as in [7], it allows us to show how the algorithm fits into the framework of multigrid-in-time methods.

2.1. Connection to linear systems. We consider a system of ordinary differential equations (ODEs) of the form

$$\mathbf{u}'(t) = \mathbf{f}(t, \mathbf{u}(t)), \quad \mathbf{u}(0) = \overline{\mathbf{u}}_0, \quad t \in [0, T], \tag{2.1}$$

such as in a method of lines approximation of a parabolic PDE. Let $t_i = i\ \delta t,\ i = 0, 1, \ldots, N_t$, be a temporal mesh with constant spacing δt, and, for $i = 1, \ldots, N_t$, let $\mathbf{u}_i$ be an approximation to $\mathbf{u}(t_i)$ and $\mathbf{u}_0 = \mathbf{u}(0)$. Then, a general one-step time discretization method for (2.1) can be written as

$$\begin{aligned} &\mathbf{u}_0 = \overline{\mathbf{u}}_0 \\ &\mathbf{u}_i = \Phi_i(\mathbf{u}_{i-1}) + \mathbf{g}_i, \quad i = 1, 2, \ldots, N_t. \end{aligned} \tag{2.2}$$

In the case of a linear problem, the function $\Phi_i(\cdot)$, corresponds to a matrix-vector product. For simplicity, we consider a time-independent discretization, thus, function $\Phi_i(\cdot)$ corresponds to a matrix-vector product with a fixed matrix which we denote by Φ, $\Phi_i(\mathbf{u}_{i-1}) = \Phi\mathbf{u}_{i-1}$; a specific example of Φ will be given in §4.1. Then, the time discretization method (2.2) is equivalent to the

linear system of equations

$$A\mathbf{u} \equiv \begin{bmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N_t} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{N_t} \end{bmatrix} \equiv \mathbf{g}, \tag{2.3}$$

where $\mathbf{g}_0 = \mathbf{u}(0)$. Note that traditional time marching corresponds to a block forward-solve of this system, which is not parallelizable. Considering the lower block bidiagonal structure, we could apply cyclic reduction. However, although cyclic reduction is optimal for scalar systems, for time discretizations, it requires products of spatial blocks that produce fill-in in the spatial dimension, yielding a method that is overall non-optimal. Nonetheless, the cyclic-reduction viewpoint can be useful in developing truly optimal and parallelizable methods. In fact, there are many spatial multigrid methods that have been designed from a similar reduction viewpoint. The idea is to replace interpolation and/or the Petrov-Galerkin coarse-grid operator with suitable approximations. Using this perspective for the time dimension allows us to design a multigrid-in-time algorithm. Before we pursue this approach, we first describe how parareal can be viewed as a method that uses a fine and a coarse temporal mesh, laying the foundation of our interpretation of parareal as a two-level reduction-based multigrid method, considered in §3.1. The connection to reduction-based multigrid methods is crucial to achieve optimality when extending the two-level algorithm to a full multilevel scheme.

2.2. Parareal. The idea of the parareal algorithm is, instead of solving the system (2.3) with a direct method, to solve it iteratively by introducing a preconditioner on a coarse temporal mesh. Therefore, let $T_j = j\ \Delta T,\ j = 0, 1, \ldots, N_t/m$, be a coarse temporal mesh with constant spacing $\Delta T = m\ \delta t$, where m is a positive integer (see Figure 2.1).

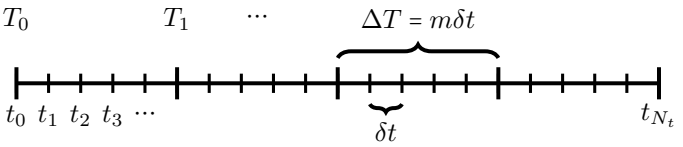

Fig. 2.1: Uniformly-spaced fine and coarse time discretization meshes.

It is easy to verify that the solution, $\mathbf{u}$, of (2.3) at mesh points $i = jm$ satisfies the coarse system of equations

$$A_\Delta \mathbf{u}_\Delta = \begin{bmatrix} I & & & \\ -\Phi^m & I & & \\ & \ddots & \ddots & \\ & & -\Phi^m & I \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\Delta,0} \\ \mathbf{u}_{\Delta,1} \\ \vdots \\ \mathbf{u}_{\Delta,N_t/m} \end{bmatrix} = R_\Phi \mathbf{g} \equiv \mathbf{g}_\Delta, \tag{2.4}$$

where $\mathbf{u}_{\Delta,j} = \mathbf{u}_{jm},\ j = 0, 1, \ldots, N_t/m$, and R_Φ is the rectangular restriction operator

$$R_\Phi = \begin{bmatrix} I & & & & & & & & \\ & \Phi^{m-1} & \cdots & \Phi & I & & & & \\ & & & & & \ddots & & & \\ & & & & & & \Phi^{m-1} & \cdots & \Phi & I \end{bmatrix}. \tag{2.5}$$

The parareal algorithm solves this coarse system iteratively, then computes the remaining fine values in parallel using (2.2) on each interval (t_{jm}, t_{jm+m-1}). To solve the coarse system (2.4), parareal uses the simple residual correction scheme

$$\mathbf{u}_\Delta^{k+1} = \mathbf{u}_\Delta^k + B_\Delta^{-1}(\mathbf{g}_\Delta - A_\Delta \mathbf{u}_\Delta^k), \tag{2.6}$$

where B_Δ is some coarse-scale time discretization of (2.1) (the analog of A on the coarse mesh),

$$B_\Delta = \begin{bmatrix} I & & & \\ -\Phi_\Delta & I & & \\ & \ddots & \ddots & \\ & & -\Phi_\Delta & I \end{bmatrix}.$$

The residual correction (2.6) is usually presented as the following equivalent update step in the parareal literature

$$\mathbf{u}_{\Delta,j+1}^{k+1} = \Phi_\Delta \mathbf{u}_{\Delta,j}^{k+1} + \Phi^m \mathbf{u}_{\Delta,j}^{k} - \Phi_\Delta \mathbf{u}_{\Delta,j}^{k} + \mathbf{g}_{\Delta,j}, \quad j = 1, 2, \ldots, \text{ with } \mathbf{u}_{\Delta,0}^{k} = \mathbf{g}_{\Delta,0}. \tag{2.7}$$

3. The multigrid-in-time algorithm. The key feature of parareal is the use of a coarse-scale time discretization that approximates the fine-scale evolution over the coarse-scale subspace. This idea is similar to the idea of multigrid reduction methods. Motivated by this observation, we interpret the parareal algorithm as a two-level reduction scheme. Gander and Vandewalle have demonstrated in [7, Section 3] that parareal coincides with a two-level multigrid-in-time method for a particular choice of smoother, restriction, and interpolation operators. Our interpretation is based on a different choice of operators, but it is straightforward to show that the resulting two-level methods are the same. The advantage of the operator choice of [7] is its simplicity, but the advantage of our operator choice is that multigrid components are more typical and, thus, allow us to extend the two-level method to a full multilevel algorithm.

In §3.1, we show how the parareal algorithm can be interpreted as an approximate two-level reduction method and, in §3.2, we describe how this viewpoint can be used to extend parareal to a multilevel algorithm.

3.1. Parareal as a two-level multigrid reduction method. We partition the temporal mesh into C-points, given by the set of coarse time-scale points, $\{i = jm\}$, and F-points. Reordering the fine-grid operator, A, by F-points first and using the subscript notation c and f to indicate the two sets of points, we consider the following well-known matrix decomposition, valid for any matrix as long as A_{ff} is nonsingular,

$$A = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix} = \begin{bmatrix} I_f & 0 \\ A_{cf}A_{ff}^{-1} & I_c \end{bmatrix} \begin{bmatrix} A_{ff} & 0 \\ 0 & S_{cc} \end{bmatrix} \begin{bmatrix} I_f & A_{ff}^{-1}A_{fc} \\ 0 & I_c \end{bmatrix}, \tag{3.1}$$

where $S_{cc} = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$ is the Schur complement and where I_c and I_f are identity operators. We define the operators R_Φ, P_Φ (known as "ideal" restriction and interpolation), and S by

$$R_\Phi = \begin{bmatrix} -A_{cf}A_{ff}^{-1} & I_c \end{bmatrix}, \quad P_\Phi = \begin{bmatrix} -A_{ff}^{-1}A_{fc} \\ I_c \end{bmatrix}, \quad S = \begin{bmatrix} I_f \\ 0 \end{bmatrix}. \tag{3.2}$$

Then, since $A_{ff} = S^T A S$ and $S_{cc} = R_\Phi A P_\Phi$, it is straightforward to see from (3.1) that

$$A^{-1} = S\left(S^T A S\right)^{-1} S^T + P_\Phi \left(R_\Phi A P_\Phi\right)^{-1} R_\Phi,$$

and, thus,

$$0 = \left(I - A^{-1}A\right) \tag{3.3}$$

$$= (I - P_\Phi (R_\Phi A P_\Phi)^{-1} R_\Phi A)(I - S(S^T A S)^{-1} S^T A). \tag{3.4}$$

Both (3.3) and (3.4) can be thought of as error propagators for exact methods that can be used to develop iterative methods by making various approximations. Equation (3.4) defines the error propagator of an exact two-level multigrid method, with the first term corresponding to the error propagator of coarse-grid correction using the Petrov-Galerkin coarse-grid operator, $R_\Phi A P_\Phi$, and the second term being the error propagator of F-relaxation. To produce an iterative multigrid method, the MGR method [3, 12, 18–20], for example, replaces the Petrov-Galerkin coarse-grid operator, $R_\Phi A P_\Phi$, with a suitable approximation and adds relaxation. The method then recurses on the coarse grid to achieve a full multilevel V-cycle.

The parareal algorithm does something similar to MGR. With the fine-grid operator, A, given by (2.3), the restriction operator, R_Φ, in (3.2) is the same as that in (2.5). Furthermore, the parareal coarse-grid operator, A_Δ, given by (2.4), satisfies the Petrov-Galerkin condition, $A_\Delta = R_\Phi A P_\Phi$. It is therefore easy to show that the error propagator for the parareal algorithm is basically given by (3.4), with the coarse-grid operator, A_Δ, replaced by the coarse-scale time discretization, B_Δ,

$$(I - P_\Phi B_\Delta^{-1} R_\Phi A)(I - S(S^T A S)^{-1} S^T A). \tag{3.5}$$

3.2. The optimal-scaling multilevel algorithm. Our multigrid-in-time algorithm is a full multilevel V-cycle scheme that results from extending the parareal method using techniques similar to those used in MGR. In particular, we replace F-relaxation in the two-level method (3.5) by FCF-relaxation, thus, we add one full FC relaxation sweep, and apply the resulting method recursively to solve the coarse system of equations defined by the coarse-scale time discretization, B_Δ. Furthermore, since $R_\Phi A P_\Phi = R_I A P_\Phi$, where $R_I = \begin{bmatrix} 0 & I_c \end{bmatrix}$ is injection, we can replace R_Φ with R_I in (3.5) to save some computational work.

To describe our method, we consider a hierarchy of time discretization meshes, Ω_l, $l = 1, \ldots, L = \log_m(N_t)$ with constant spacing δt on level 1, $m\delta t$ on level 2, etc., for a positive coarsening factor, m. Let $A_l \mathbf{u}_l = \mathbf{g}_l$ be the linear system of equations on level $l = 1, \ldots, L$, where A_l is the time discretization on the mesh Ω_l, characterized by the matrix Φ_l. For each level, l, we decompose the matrix A_l into F- and C-points and define the interpolation operator, P_Φ, as in (3.2). Then, our multigrid-in-time algorithm for solving (2.1) can be written as follows:

MGIT(l)
if l is the coarsest level, L
- Solve coarse-grid system $A_L \mathbf{u}_L = \mathbf{g}_L$.

else
- Relax on $A_l \mathbf{u}_l = \mathbf{g}_l$ using FCF-relaxation.
- Compute and restrict residual using injection, $\mathbf{g}_{l+1} = R_I(\mathbf{g}_l - A_l \mathbf{u}_l)$.
- Solve on next level using this algorithm: **MGIT**($l+1$).
- Correct using "ideal interpolation", $\mathbf{u}_l \leftarrow \mathbf{u}_l + P_\Phi \mathbf{u}_{l+1}$.

end

Fig. 3.1: Our multigrid-in-time algorithm.

Note that parareal and our algorithm solve for the exact solution in N_t/m iterations corresponding to the number of points on the first coarse level. This is an interesting property of the two algorithms, however, in practice, the fact that they converge to some error tolerance in $\mathcal{O}(1)$ iterations is more relevant.

4. Numerical results. In this section, we consider optimality and parallel performance of our multigrid-in-time algorithm. In §4.1, we describe our parabolic model problem, the one-dimensional diffusion equation, and a simple discretization that defines the matrix, Φ, of a constant coefficient one-step method and, thus, the entries on the lower diagonal of the matrices of the linear systems on each level. In §4.2, we then demonstrate optimality of our algorithm for solving the model problem. Section 4.3 is devoted to a brief review of a simple parallel performance model, followed by a comparison of our algorithm to sequential time-marching using this model.

4.1. The parabolic model problem and its discretization. We consider the one-dimensional diffusion equation,

$$\mathbf{u}_t = \kappa \mathbf{u}_{xx} + \mathbf{b}(x,t), \quad \kappa > 0, \quad x \in [0,\pi],\ t \in [0,T], \tag{4.1}$$

subject to an initial condition and zero Dirichlet boundary conditions,

$$\mathbf{u}(x,0) = \overline{\mathbf{u}}_0(x), \quad x \in [0,\pi] \tag{4.2}$$

$$\mathbf{u}(0,t) = \mathbf{u}(\pi,t) = 0, \qquad t \in [0,T]. \tag{4.3}$$

We use the method of lines to transform our model problem to a system of ODEs of the form (2.1). Let $x_j = j\ \Delta x$, $j = 0, 1, \ldots, N_x$, be a spatial mesh with constant spacing Δx and, for $j = 0, 1, \ldots, N_x$ and a given time, t, let $u_j(t)$ be an approximation to $u(x_j, t)$ with $u_0(t) = u_{N_x}(t) = 0$ using the boundary conditions, (4.3). Using central finite differences, the method of lines approximation of (4.1) is

$$\frac{\partial u_j}{\partial t} = \kappa \frac{u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)}{(\Delta x)^2} + b_j(t), \quad j = 1, 2, \ldots, N_x - 1. \tag{4.4}$$

With

$$M = \frac{\kappa}{(\Delta x)^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}, \tag{4.5}$$

the method of lines approximation in Equation (4.4) becomes

$$\frac{\partial \mathbf{u}(t)}{\partial t} = M\mathbf{u}(t) + \mathbf{b}(t) \equiv \mathbf{f}(t, \mathbf{u}(t)).$$

Now, let $t_i = i\ \delta t,\ i = 0, 1, \ldots, N_t$, be a temporal mesh with constant spacing δt, and, for $i = 0, 1, \ldots, N_t$, let $\mathbf{u}_i$ be an approximation to $\mathbf{u}(t_i)$ with $\mathbf{u}_0 = \overline{\mathbf{u}}_0$ using the initial condition, (4.2). If we use backward Euler, we obtain

$$(I - \delta t M)\mathbf{u}_i - \delta t \mathbf{b}_i = \mathbf{u}_{i-1}, \quad i = 1, 2, \ldots, N_t,$$

defining a one-step method of the form (2.2) with $\Phi = (I - \delta t M)^{-1}$ and $\mathbf{g}_i = (I - \delta t M)^{-1}\delta t \mathbf{b}_i$ for $i = 1, 2, \ldots, N_t$. If we use forward Euler, then $\Phi = I + \delta t M$ and $\mathbf{g}_i = \delta t \mathbf{b}_{i-1}$ for $i = 1, 2, \ldots, N_t$.

4.2. Optimality of the multigrid-in-time algorithm. We report on tests of using our multigrid-in-time algorithm to solve the model problem described in §4.1, with a zero right-hand side, $\kappa = 1$, and subject to the initial condition $\mathbf{u}(0, x) = \sin(x),\ 0 \le x \le \pi$, as well as zero Dirichlet boundary conditions. Choosing a zero right-hand side simplifies verifying that our algorithm computes a good approximation to the true solution. However, we want to emphasize that a non-zero right-hand side does not change our algorithm, since it only defines the right-hand side, $\mathbf{g}$, of the linear system (2.3). We consider both the two-level and full multilevel, $L = \log_m(N_t)$, variants of our multigrid-in-time algorithm. On the finest grid, we use the initial condition as the initial guess for $t = 0$, and a random initial guess for all other times. Choosing a random initial guess for all times $t > 0$ corresponds to not using any knowledge of the right-hand side that could affect convergence. For simplicity, we first consider factor-2 coarsening only; other coarsening factors are discussed later.

Tables 1 and 2 show the number of multigrid iterations for solving the model problem discretized using backward Euler in time and central finite differences in space with the two-level or full multilevel versions of our multigrid-in-time algorithm with factor-2 coarsening and FCF-relaxation on different space-time grids. We used the relative residual norm, measured in the L_2-norm, to be less than 10^{-9} as the stopping criterion for our algorithm. The time step on the finest grid is chosen to be $\delta t = (\Delta x)^2$, and $2^{l-1}\delta t$ for all other levels, $l > 1$. In Table 1, we consider the effect of increasing the number of time steps, N_t, and, thus, varying the length of the time interval, T, for a fixed number of spatial steps, N_x. Since we are using a semi-coarsening strategy with coarsening in time only, this test is important for answering the question whether convergence is independent of the problem size or not. Results show that iterations of the two-level and full multilevel variants of our multigrid-in-time algorithm are bounded independently of the problem size.

	$N_t = 2^5$	$N_t = 2^6$	$N_t = 2^7$	$N_t = 2^8$	$N_t = 2^9$	$N_t = 2^{10}$	$N_t = 2^{11}$
two-level	6	7	7	7	7	7	7
full multilevel	6	7	8	8	8	9	9

Table 1: Number of iterations for solving the model problem discretized using backward Euler in time and central finite differences in space with the two-level or full multilevel versions of our multigrid-in-time algorithm with factor-2 coarsening and FCF-relaxation on $16 \times N_t$ space-time grids.

Increasing only the number of time steps while fixing the number of spatial steps and the ratio $\delta t/(\Delta x)^2$ changes the domain of the problem. In Table 2, we consider a domain refinement corresponding to simultaneously scaling up the spatial and temporal resolutions. Fixing $\delta t = (\Delta x)^2$, we have to quadruple the number of points in time when doubling the number of points in space.

	$N_x \times N_t = 2^4 \times 2^5$	$N_x \times N_t = 2^5 \times 2^7$	$N_x \times N_t = 2^6 \times 2^9$	$N_x \times N_t = 2^7 \times 2^{11}$
two-level	6	7	7	6
full multilevel	6	8	8	9

Table 2: Number of iterations for solving the model problem discretized using backward Euler in time and central finite differences in space with the two-level or full multilevel versions of our multigrid-in-time algorithm with factor-2 coarsening and FCF-relaxation on $N_x \times N_t$ space-time grids.

Again, the number of iterations of the two-level and full multilevel variants of our multigrid-in-time algorithm appear to be bounded independently of the problem size.

Our multigrid-in-time algorithm results from extending the parareal algorithm using techniques similar to those used in MGR. In particular, we replaced F-relaxation by FCF-relaxation. Table 3 shows that the additional full FC relaxation sweep is necessary to achieve optimality in the full multilevel algorithm. Note that the two-level algorithm is the parareal method.

	$N_t = 2^5$	$N_t = 2^6$	$N_t = 2^7$	$N_t = 2^8$	$N_t = 2^9$	$N_t = 2^{10}$	$N_t = 2^{11}$	$N_t = 2^{12}$
two-level	9	9	9	9	9	9	9	9
full multilevel	10	13	16	20	23	23	25	31

Table 3: Number of iterations for solving the model problem discretized using backward Euler in time and central finite differences in space with the two-level or full multilevel versions of our multigrid-in-time algorithm with factor-2 coarsening and F-relaxation (no additional full FC relaxation) on $16 \times N_t$ space-time grids.

So far, we have only presented results for factor-2 coarsening. Table 4 shows results similar to those in Table 1 for factor-4 coarsening. Coarsening by a factor of 4 is particularly interesting for extending our algorithm to simultaneously coarsen in space and time. Results look promising for this approach since, again, the number of iterations of the two-level and full multilevel variants of our multigrid-in-time algorithm are bounded independently of the problem size. Furthermore, the numbers of iterations appear to be independent of the coarsening factor and, thus, factor-4 coarsening could produce a more efficient algorithm, since we have less work on the coarse grids.

	$N_t = 4^3$	$N_t = 4^4$	$N_t = 4^5$	$N_t = 4^6$	$N_t = 4^7$
two-level	7	7	7	7	7
full multilevel	7	7	9	9	9

Table 4: Number of iterations for solving the model problem discretized using backward Euler in time and central finite differences in space with the two-level or full multilevel versions of our multigrid-in-time algorithm with factor-4 coarsening and FCF-relaxation on $16 \times N_t$ space-time grids.

4.3. Parallel performance. Current trends in computer architectures are leading towards more, but slower, processors. In contrast to traditional time marching that only allows parallelization in space, using our multigrid-in-time algorithm, we can parallelize the solution process of space-time problems in both space and time and, thus, increase concurrency. In this section, we use a simple parallel performance model to compare the time to solve a given problem with our multigrid-in-time algorithm to the time to do sequential time marching. In particular, we are interested in answering three questions. First, is it beneficial to use our multigrid-in-time algorithm on modern architectures? Second, considering the time to solution with both methods as functions of the number of processors, where is the crossover point? And third, what speedup can we expect from using our algorithm?

We consider the d-dimensional diffusion equation. Using an implicit time discretization method of the form (2.2), the function $\Phi_i(\cdot)$, or $\Phi(\cdot)$ in the time-independent case, corresponds to a spatial solve. We assume that we solve these spatial problems in parallel using multigrid-in-space with coarsening by a factor of 2 in each dimension. The time to do sequential time marching, is therefore, given by the number of time steps multiplied by the time of one spatial solve, thus, the time of one parallel multigrid-in-space V-cycle multiplied by the number of iterations necessary to solve to a given accuracy. For the d-dimensional diffusion equation, we have to solve a system with a Laplacian

and, therefore, it is reasonable to assume eight multigrid-in-space iterations. Since our multigrid-in-time algorithm is a V-cycle scheme, the time to solve the d-dimensional diffusion equation is given by the time of one parallel multigrid-in-time V-cycle multiplied by the number of iterations necessary to solve to a given accuracy. Numerical experiments presented in §4.2 show that the number of iterations is bounded by nine for solving the one-dimensional diffusion equation. Since the spatial solve corresponds to a function call (of Φ) in our multigrid-in-time algorithm, we assume that the number of iterations does not change when we consider more space dimensions.

We estimate the time of multigrid-in-space and multigrid-in-time V-cycles by applying a simple performance model. We assume that the time to communicate n doubles is given by

$$T_{\text{comm}} = \alpha + n\beta, \tag{4.6}$$

where α represents the communication latency and β is the actual communication time per double. Furthermore, let the time to perform n floating point operations be of the form

$$T_{\text{comp}} = n\gamma. \tag{4.7}$$

We assume that the spatial domain consists of N_x^d points distributed across P_x^d processors. For our multigrid-in-time algorithm with factor-m coarsening, we furthermore assume that the time domain consists of $N_t = m^q N_x$ points, where q is a positive integer. We choose the temporal problem size to be a multiple of the spatial problem size since, for small problems, sequential time-stepping with multigrid-in-space is already efficient. The temporal domain is distributed across P_t processors such that we consider a perfect hypercube in space-time on each processor.

Using the above assumptions, expressions for the time to solve a given problem with our multigrid-in-time algorithm and for the time to do sequential time-stepping can be derived. Assuming the communication and computation models in Equations (4.6) and (4.7), the expressions depend on the machine parameters α, β, and γ. The numbers in [5, Table 2] can be used as the basis for choosing two parameter sets characterizing modern machines: a "computation dominant" set consisting of the parameters

$$\alpha = 1\ \mu\text{s}, \quad \beta = 10\ \text{ns/double}, \quad \gamma = 8\ \text{ns/flop}, \tag{4.8}$$

and a "communication dominant" set defined by

$$\alpha = 1\ \mu\text{s}, \quad \beta = 0.74\ \text{ns/double}, \quad \gamma = 0.15\ \text{ns/flop}. \tag{4.9}$$

The ratios α/β and α/γ are assumed to be "small" in the computation dominant set and "large" in the communication dominant set. To define the parameter sets (4.8) and (4.9), we have set $\alpha = 1\ \mu$s and chosen β and γ such that the ratios α/β and α/γ are equal to the minimum or maximum ratios from [5, Table 2], respectively.

Based on the two parameter sets (4.8) and (4.9), we compare the two time integration approaches. Figure 4.1 shows the time to solve a $1024^3 \times 16,384$ space-time problem using sequential time-stepping and the time to solution for applying our multigrid-in-time algorithm as functions of the number of processors used for the computations. The left plot shows the expected behavior based on the computation dominant parameters (4.8), and the right plot presents the expected behavior based on the communication dominant parameters (4.9). Considering a smaller number of processors, sequential time-stepping is both faster and uses less memory (for sequential time-stepping, one has to store data from one time step only, whereas for the multigrid-in-time approach, a whole space-time subdomain, i. e., data from several time steps, needs to be stored). On a larger number of processors, however, multigrid-in-time is faster. The choice of which algorithm to use, therefore, depends primarily on the available computational resources. More precisely, for the computation dominant model, the crossover point at which it becomes beneficial to use our multigrid-in-time algorithm is at about 2^{24}(~ 17 million) processors. Increasing the number of processors to 2^{28}(~ 268 million) results in an expected speedup of about six compared to sequential time-stepping. For the communication dominant model, the crossover point is at about one million processors at which point we already expect a speedup of about two. Using 2^{24} processors leads to an expected

speedup of about seven, while with 2^{28} processors it is about 14. With current trends in computer architectures leading towards more processors, our multigrid-in-time algorithm looks promising to speedup computations, especially for the communication dominant model. This result is attractive since on future architectures, we expect the parameters to be most likely in the more communication dominant regime.

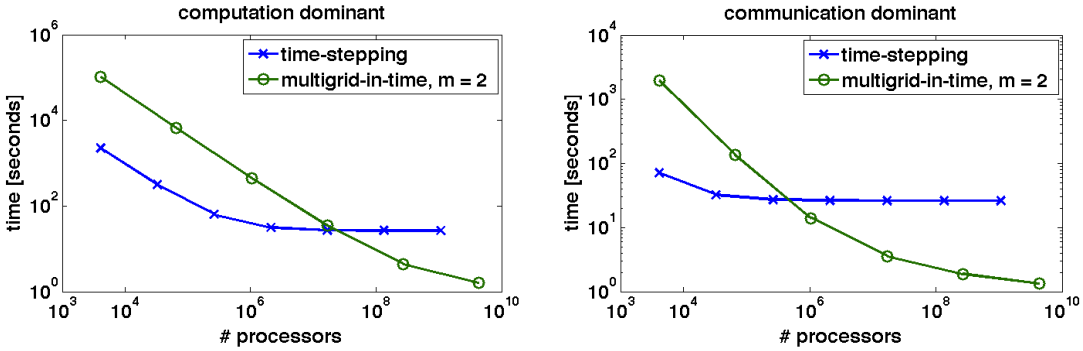

Fig. 4.1: Time to solve a $1024^3 \times 16,384$ space-time problem using sequential time-stepping or our multigrid-in-time algorithm. At left, expected behavior based on the computation dominant parameters (4.8) and at right, expected behavior based on the communication dominant parameters (4.9).

Benefits of our multigrid-in-time approach can also be realized at smaller scales. Comparing the two time integration approaches for other problem sizes, the time curves look similar to those in Figure 4.1, but the crossover point changes. For a $128^3 \times 2048$ space-time problem, for example, the crossover point is at about 100k processors when considering the computation dominant model. Increasing the number of processors to one million results in an expected speedup of about two compared to sequential time-stepping. For the communication dominant model, the crossover point is at about $2^{12}(= 4096)$ processors, and using $2^{16}(\sim 65\text{k})$ processors leads to an expected speedup of about two. In general, the larger the time dimension in comparison to the spatial dimension (the factor m^q in our experiments), the greater is the potential for speedup.

Increasing the number of processors decreases the local problem size and, consequently, increases the ratio of boundary to domain or, equivalently, decreases the computation/communication ratio. However, even though a small computation/communication ratio seems unreasonable at first, if the code runs faster by adding more processors, small local problems may be reasonable and beneficial provided that computational resources are available. Our multigrid-in-time algorithm allows one to exploit substantially more computational resources than standard sequential time-stepping.

5. Discussion. We have presented an optimal-scaling multigrid-in-time algorithm for solving evolution equations in parallel. With current trends in computer architectures leading towards more, but slower, processors, solving for multiple time steps in parallel is an important practical consideration. One approach to achieve parallelism in time are multigrid methods, but research is needed for extending classical (spatial) multigrid methods to include the time dimension. Although we are interested in full space-time multigrid schemes, in this work we pursued a multigrid method that only coarsens in time. The benefit of this approach is that the resulting method is fairly unintrusive on existing codes, meaning parallelization in time can be added with very few changes in an existing code. Our multigrid-in-time algorithm inherits this property from the parareal time integration method which served as the starting point of our research. Interpreting parareal as a two-level reduction method is non-traditional, but precisely this perspective allowed us to draw a connection to the MGR method needed to extend the two-level scheme to an optimal full multilevel algorithm. More precisely, the MGR viewpoint made it possible to see that replacing F-relaxation with FCF-relaxation would be needed for optimality.

If the problem (2.1) is nonlinear, our MGR ideas can be generalized to the full approximation storage (FAS) setting. FAS was originally proposed by Brandt [2] and can essentially be seen as "nonlinear multigrid". The main differences between two-level FAS and linear two-level multigrid

are that relaxation is nonlinear and that the nonlinear coarse system of equations is used to compute a coarse approximation to the fine-grid solution, not the fine-grid error. For linear systems, the two approaches are equivalent. The two-level interpretation of parareal in [7] was described as an FAS method for the full nonlinear setting of (2.2). It is similarly straightforward to generalize our MGR ideas to the nonlinear setting, though convergence questions become even harder to answer.

A preliminary parallel scaling study shows that our multigrid-in-time algorithm looks promising when an implicit time discretization method is used. Since, in general, performance models give a good qualitative picture of the relationship between compared algorithms, motivated by the results of our performance model, future work also includes writing parallel code to compare actual behavior to expectations.

For explicit time discretization schemes, stability has to be considered. Numerical experiments for solving the one-dimensional diffusion equation using forward Euler instead of backward Euler time discretization show that results are no longer scalable unless the CFL condition is satisfied on all levels. Simultaneously coarsening in space and time and, therefore, extending to a space-time multigrid approach could help with these stability issues.

REFERENCES

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

[2] A. Brandt, *Multi–level adaptive solutions to boundary–value problems*, Math. Comp., 31 (1977), pp. 333–390.

[3] H. Foerster, K. Stüben, and U. Trottenberg, *Nonstandard multigrid techniques using checkered relaxation and intermediate grids*, in Elliptic Problem Solvers, M. Schulz, ed., Academic, New York, 1981, pp. 285–300.

[4] S. H. Fuller and L. I. Millett, eds., *The Future of Computing Performance: Game Over or Next Level?*, The National Academies Press, 2011. Committee on Sustaining Growth in Computing Performance; National Research Council.

[5] H. Gahvari, A. Baker, M. Schulz, U. M. Yang, K. Jordan, and W. Gropp, *Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms*, in 25th ACM International Conference on Supercomputing, Tucson, AZ, 2011.

[6] M. J. Gander and E. Hairer, *Nonlinear convergence analysis for the parareal algorithm*, in Domain Decomposition Methods in Science and Engineering XVII, U. Langer, M. Discacciati, D. E. Keyes, O. B. Widlund, and W. Zulehner, eds., vol. 60 of Lecture Notes in Computational Science and Engineering, Springer Berlin Heidelberg, 2008, pp. 193–200.

[7] M. J. Gander and S. Vandewalle, *Analysis of the parareal time-parallel time-integration method*, SIAM J. Sci. Comput., 29 (2007), pp. 556–578.

[8] I. Garrido, B. Lee, G. E. Fladmark, and M. S. Espedal, *Convergent iterative schemes for time parallelization*, Math. Comp., 75 (2006), pp. 1403–1428.

[9] C. W. Gear, *Parallel methods for ordinary differential equations*, Calcolo, 25 (1988), pp. 1–20.

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

[11] K. R. Jackson, *A survey of parallel numerical methods for initial value problems for ordinary differential equations*, IEEE Trans. Magnetics, 27 (1991), pp. 3792–3797.

[12] D. Kamowitz and S. V. Parter, *On MGR[ν] multigrid methods*, SIAM J. Numer. Anal., 24 (1987), pp. 366–381.

[13] 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), pp. 661–668.

[14] Y. Maday, *The parareal in time algorithm*, June 2008.

[15] M. L. Minion, *A hybrid parareal spectral deferred corrections method*, Comm. App. Math. and Comp. Sci., 5 (2010), pp. 265–301.

[16] M. L. Minion and S. A. Williams, *Parareal and spectral deferred corrections*, in Numerical Analysis and Applied Mathematics, T. E. Simos, ed., no. 1048 in AIP Conference Proceedings, AIP, 2008, pp. 388–391.

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

[18] S. V. Parter, *On an estimate for the three-grid MGR multigrid method*, SIAM J. Numer. Anal., 24 (1987), pp. 1032–1045.

[19] M. Ries and U. Trottenberg, *MGR-ein blitzschneller elliptischer Löser*, Tech. Rep. Preprint 277 SFB 72, Universität Bonn, 1979.

[20] M. Ries, U. Trottenberg, and G. Winter, *A note on MGR methods*, Linear Algebra Appl., 49 (1983), pp. 1–26.

[21] H. D. Sterck, T. A. Manteuffel, S. F. McCormick, and L. Olson, *Least-squares finite element methods and algebraic multigrid solvers for linear hyperbolic PDEs*, SIAM J. Sci. Comput., 26 (2004), pp. 31–54.

[22] D. E. Womble, *A time-stepping algorithm for parallel computers*, SIAM J. Stat. Comput., 11 (1990), pp. 824–837.