A Performance Model for Allocating the Parallelism in a Multigrid-in-Time Solver

Hormozd Gahvari, Veselin A. Dobrev, Robert D. Falgout,
Tzanio V. Kolev, Jacob B. Schroder, Martin Schulz, Ulrike Meier Yang
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA USA
{gahvari1,dobrev,rfalgout,tzanio,schroder2,schulzm,umyang}@llnl.gov

***Abstract*—The traditional way to numerically solve time-dependent problems is to sequentially march through time, solving for one time step and then the next. The parallelism in this approach is limited to the spatial dimension, which is quickly exhausted, causing the gain from using more processors to solve a problem to diminish. One approach to overcome this barrier is to use methods that are parallel in time. These methods have the potential to achieve dramatically better performance compared to time-stepping approaches, but achieving this performance requires carefully choosing the amount of parallelism devoted to space versus the amount devoted to time. Here, we present a performance model that, for a multigrid-in-time solver, makes the decision on when to switch to parallel-in-time and on how much parallelism to devote to space vs. time. In our experiments, the model selects the best parallel configuration in most of our test cases and a configuration close to the best one in all other cases.**

I. Introduction

On our path to exascale and beyond we continue to see a steep increase in the amount of parallelism provided by leading HPC systems. Already today, core counts have reached the millions, and they are expected to grow even more in the future. Algorithms must be capable of exploiting this available parallelism efficiently in order to make use of these machines. However, this is increasingly becoming problematic, especially for strong-scaling problems, since the increase in the level of parallelism requires a finer and finer decomposition of the domain and data. If this decomposition gets too fine, we are faced with higher data transfer overheads, or, in the extreme case, even idling resources. We therefore need to find new sources to extract parallelism.

Time-dependent simulations are traditionally time-stepped, i.e., a data domain (often a grid or mesh) is decomposed in 2D or 3D physical space and evolved over time through a main loop using a series of sequential time steps. Each time step relies on the result of the previous one and advances the simulation by a time-step size δt. In this approach, parallelism is achieved by distributing the decomposed 2D or 3D space among the compute resources, which can work on their parts concurrently. Consequently, the amount of parallelism is constrained by the granularity of the spatial domain decomposition, which can be limited.

One way to overcome this limitation is to use a method that is parallel in time. Such methods reformulate time-dependent problems to include time as a dimension that can be decomposed across processors. This way, all time steps can be solved in parallel. This adds a very large amount of parallelism, making these methods a good match for current and future HPC systems.

However, using a parallel-in-time approach does not come for free: it typically exhibits higher overheads and hence is only suitable when used on a large number of processing cores. Determining the exact cross-over point at which a parallel-in-time approach outperforms a traditional sequential solver is not trivial and typically requires a lot of trial and error. Furthermore, with time now another dimension that can be parallelized, determining the right tradeoff between resources used in the spatial dimensions versus the time dimension is an equally unanswered problem. Solving both questions, however, is critical for the successful use of parallel-in-time solutions.

In this work, we make the following contributions:

- We derive a general performance model for MGRIT [1], a parallel-in-time solver based on multigrid. This model allows us to determine both the correct cross-over point in terms of scale as well as a suitable processor distribution in space vs. time.
- We validate our model on three different platforms using two case studies, a 2D heat transfer problem and an advection-diffusion problem.
- Our model accurately predicts the best distribution of processes in space and time about 85 percent of the time, for a given problem, process count, and coarsening factor. In most of the cases where it did not, the prediction was close to the best one.
- Most importantly, our work lays the foundation for the practical use of parallel-in-time methods. Without a predictive model to guide when to use these methods, and how much parallelism to devote to space and time, getting the best performance requires a lot of guesswork that can be a real burden to users.

The remainder of the paper is organized as follows. Section II discusses the basics of the MGRIT algorithm. Section III introduces our performance model, and Section IV gives specifics on how we adapt the model so that it can be used to make predictions. Section V discusses our results, and Section VI discusses future directions for further development of the model. Related work is overviewed in Section VII, and our conclusions are in Section VIII.

II. Multigrid-in-Time Basics

As mentioned before, parallel-in-time methods work by treating time as yet another dimension that can be decomposed across processors. To visualize this, consider solving a first-order system of ordinary differential equations,

$$\mathbf{u}'(t) = \mathbf{f}(t, \mathbf{u}(t)), \quad \mathbf{u}(0) = \mathbf{g}_0, \quad t \in [0, t_{\max}].$$

For simplicity, consider discretization on a uniform temporal mesh of T time steps with spacing $\delta t = t_{\max}/T$ and time values of $t_i = i\delta t$ for $i = 0, 1, \ldots, T$. A general one-step time discretization of this system can then be written as

$$\mathbf{u}_i = \Phi_i(\mathbf{u}_{i-1}) + \mathbf{g}_i, \quad i = 1, 2, \ldots, T,$$

where $\mathbf{u}_0 = \mathbf{g}_0$. For instance, one application of Φ_i could be forward or backward Euler. To further simplify the presentation, we assume $\mathbf{f}$ is linear. [1] Then the discretized system of ODEs is equivalent to the linear system

$$A = \begin{bmatrix} I & & & \\ -\Phi_i & I & & \\ & \ddots & \ddots & \\ & & -\Phi_T & I \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_T \end{bmatrix} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_T \end{bmatrix} = \mathbf{g}. \tag{1}$$

Solving the system of ODEs with sequential time-stepping is equivalent to a direct block-triangular solve of (1). However, triangular solvers scale poorly in parallel. This has led to the use of other solvers for leveraging parallelism in time, such as multigrid, which has the desirable property of ideal algorithmic scalability. One such solver is multigrid-reduction-in-time (MGRIT), which works by non-intrusively adding parallelism in time to an existing sequential time-marching code [1]. This is accomplished through a multigrid reduction [3], [4] strategy that uses coarse time grids to accelerate convergence to the solution on the finest time grid. MGRIT is implemented in the software package XBraid [5], which has already seen use in a compressible fluid dynamics application problem [6]. XBraid allows the user to non-intrusively wrap pre-existing time-stepping and problem initialization routines. A call to a driver routine then starts the MGRIT solver. We use the XBraid implementation of MGRIT in our experiments.

We now briefly describe a two-grid MGRIT process, but note that the process can be done recursively for a multilevel solver. Let the coarse time grid be formed with a coarsening factor of c_t, as depicted in Figure 1. MGRIT first proceeds with a local relaxation process, alternating between F- and C-points (fine and coarse grid points as depicted in Figure 1). An F-relaxation consecutively updates all $\{\mathbf{u}_j\}$ on an interval (τ_i, τ_{i+1}) through applications of Φ_j. Each F-interval

[1]Nonlinear problems can also be solved, see Falgout et al. [2].

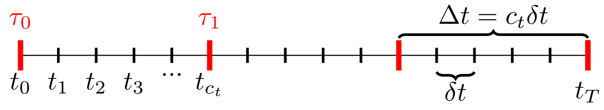

Fig. 1. Fine and coarse time-grids. Fine grid points t_i are present on only the fine grid, whereas coarse grid points τ_i (red) are on both the fine and coarse grids.

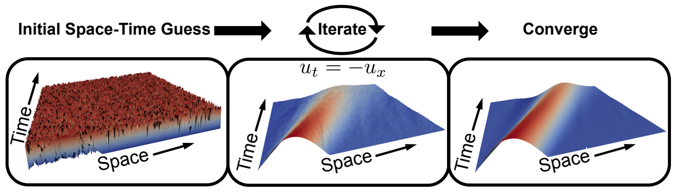

Fig. 2. Example MGRIT solution process for the 1D linear advection equation. For illustration, the initial space-time guess is uniformly random, and then the MGRIT algorithm iteratively updates the entire space-time solution until convergence.

is sequentially updated, but intervals are independent and done in parallel. C-relaxation updates each C-point in parallel with an application of Φ_j. In practice, relaxation is either an F-sweep or a more powerful FCF-sweep.

After relaxation, the residual $\mathbf{r} = \mathbf{g} - A\mathbf{u}$ is restricted (with injection) to the coarse time-grid. Next, the coarse grid residual equation $A_\Delta \mathbf{e}_\Delta = \mathbf{r}_\Delta$ is solved, where $\cdot_\Delta$ represents coarse grid quantities. The matrix A_Δ has the same structure as A, only it represents just the T/c_t coarse time-steps and uses a coarse grid time propagator $\Phi_{i,\Delta}$ which is typically a rediscretization in time using $\Delta t = c_t \delta t$. For example, $\Phi_{i,\Delta}$ could be backward Euler using Δt, while Φ_i is backward Euler using δt. This is the key approximation made in the method. The error correction $\mathbf{e}_\Delta$ is then harmonically interpolated [1] to the fine-grid and updates the solution, completing a two-grid cycle. Figure 2 depicts this cycling process on the fine-grid for the 1D linear advection equation.

In summary, MGRIT exposes concurrency in the time dimension with multigrid. It is non-intrusive and leaves the time discretization unchanged. The user simply wraps their existing time-stepping routine to form the Φ operator. As such MGRIT, converges to the same solution as sequential time-stepping. The method is optimal for a variety of parabolic problems, regardless of the coarsening factor or the spatial dimension of the problem being solved [7], [1]. In a simple two-level setting with F-relaxation, MGRIT is equivalent [8] to the Parareal method [9], which is perhaps the most popular parallel-in-time method.

Regarding performance, sequential time-stepping and MGRIT are both linear in the number of time steps. More specifically, they are both linear in the number of rounds of communication and computation required, but time-stepping has a lower constant for the computation term and a significantly higher constant for the communication term than MGRIT. However, MGRIT is able to utilize far more processors to parallelize its computational term. Hence, there is a performance cross-over point between the two methods [1]. For processor counts less than the cross-over, sequential time-stepping is faster, but for larger processor counts, MGRIT is faster. Determining this cross-over point can be time-consuming and costly. Thus, a major goal of this paper is to determine the cross-over *a priori* with a model, and to furthermore predict the optimal space-time processor layout for a fixed processor count.

III. Basic Modeling

We build our model of MGRIT in a modular fashion, to allow for a wide range of possible time-stepping routines Φ. In fact, we assume that both Φ and the problem being solved are arbitrary. For a given problem, each application of Φ incurs some computation and some communication. This might be as simple as a combination of a nearest neighbor stencil computation with its associated communication, in the case of an explicit Φ, or as involved as an iterative solver with multiple rounds of communication and computation, plus collective communication to check convergence, in the case of an implicit Φ that uses an iterative solver. The definitions of the modeling terms are summarized in Table I.

We count the time spent in the involved operations as follows. Let $\phi(N)$ be the time spent in computation when applying Φ to N spatial points in serial. Let $\psi(N, P)$ be the time spent in point-to-point communication when applying Φ to N points with P processes, and let $\xi(P)$ be time spent in collective communication when applying Φ with P processes. These values will depend on the machine, the time-stepping routine, and the problem being solved. Assuming like before that we take T time steps, the time spent when using sequential time-stepping is

$$T_{\text{seq}} = T\left(\frac{\phi(N)}{P} + \psi(N,P) + \xi(P)\right). \tag{2}$$

When moving to multigrid in time, we will look at the operation count in terms of relaxations and matrix-vector multiplications. Assume we are using $P = P_x P_t$ processes, where P_x is the number of processes in space and P_t is the number of processes in time. The process layout is shown in Figure 3 with $P_x = 4$ and $P_t = 4$, compared with sequential time-stepping with four processes in space. Moving to parallel-in-time requires the storage of several time steps at once, which leads to larger storage requirements than is the case for sequential time-stepping, where only one time step needs to be stored. To save memory, MGRIT stores only the C-points.

Assuming, as before, that we are coarsening in time with factor c_t, relaxation using the matrices in (1) takes time

$$T_{\text{C}} = \frac{T}{c_t P_t}\left(\frac{\phi(N)}{P_x} + \psi(\frac{N}{P_x}, P_x) + \xi(P_x)\right) \tag{3}$$

for C-relaxation, and

$$T_{\text{F}} = \frac{T}{P_t}\left(1 - \frac{1}{c_t}\right)\left(\frac{\phi(N)}{P_x} + \psi(\frac{N}{P_x}, P_x) + \xi(P_x)\right) \tag{4}$$

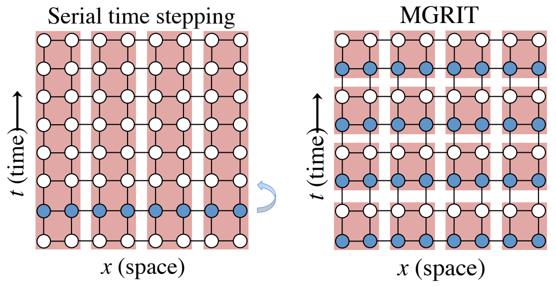

Fig. 3. Processor layout comparison, where the blue dots represent parts of the domain being actively computed and stored, and the pink boxes represent the data that a process owns.

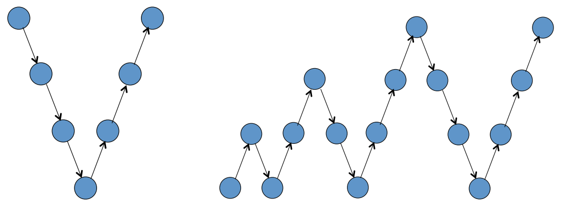

Fig. 4. Multigrid V-cycle (left), and full multigrid cycle (right). The V-cycle progresses from the finest grid to the coarsest grid, and then back to the finest grid. The full multigrid cycle starts on the coarsest grid, and consists of a series of V-cycles of progressively increasing depth until the final one, which starts on the finest grid.

for F-relaxation. A matrix-vector multiplication using these matrices takes time $T_C + T_F$, to cover both the C-points and F-points.

To expand this to a multigrid V-cycle, which is pictured in Figure 4, we consider each level in time separately. This requires us to introduce a coarsening factor in space, c_x. The reason for this is that when Φ represents an explicit time-stepping routine, the spatial grid needs to be coarsened along with the temporal grid to keep the relative grid spacings from violating the CFL limit, which would prevent the solver from obtaining the correct answer [10].

Define expressions for the time spent in C-relaxation and F-relaxation on grid i to be:

$$\begin{aligned}
T_C^i &= \max\left\{\frac{T}{c_t^i P_t}, 1\right\}\left(\frac{\phi(N)}{c_x^{i-1}P_x} + \psi(\frac{N}{P_x}, P_x) + \xi(P_x)\right) \\
T_F^i &= \max\left\{\frac{T}{c_t^{i-1}P_t}\left(1 - \frac{1}{c_t}\right), c_t - 1\right\}\cdot \\
&\quad \left(\frac{\phi(N)}{c_x^{i-1}P_x} + \psi(\frac{N}{P_x}, P_x) + \xi(P_x)\right)
\end{aligned}$$

Both of these enforce minimums on the number of consecutive relaxations performed. For C-relaxation, which can be performed completely in parallel, this minimum is simply 1, i.e., at least one C-relaxation has to be performed. F-relaxation, however, requires relaxation on the previous F-point to be complete before proceeding to the next one, making the minimum equal to the number of F-points between each C-point.

Relaxation on a particular grid can be a number of different combinations of C-relaxation and F-relaxation. Here, we as-

TABLE I
DEFINITIONS OF MODELING TERMS.

Parameter	Definition
N	Number of spatial points
T	Number of time steps
P	Total number of processes
P_x	Number of spatial processes
P_t	Number of processes in the time direction
$\phi(N)$	Computation time for applying Φ to N points in serial
$\psi(N,P)$	Point-to-point communication time for applying Φ to N points using P processes
$\xi(P)$	Collective communication time for applying Φ using P processes
T_C	Time for C-relaxation
T_F	Time for F-relaxation
T_C^i	Time for C-relaxation on grid i
T_F^i	Time for F-relaxation on grid i
L	Number of multigrid cycles in time
T_{FMG}	Time for a full multgrid cycle

sume an F-relaxation sweep followed by a C-relaxation sweep and another F-relaxation sweep, which we denote as FCF-relaxation. Matrix-vector multiplication with the A operator in (1), which forms the residual before proceeding to the next coarsest grid, is treated as a sweep of F-relaxation and a sweep of C-relaxation. After the coarse grid correction is determined, there is a another sweep of F-relaxation before interpolating the result to the next finest grid, except on the finest level. We also charge a C-relaxation sweep to account for the application of the coarse-grid operator when forming the right-hand-side for the Full Approximation Storage (FAS) coarse-grid [11]. [2] The cycle time is then

$$3T_C^1 + 3T_F^1 + \sum_{i=2}^{L-1}(3T_C^i + 4T_F^i) + 2T_C^L + 3T_F^L, \tag{5}$$

where $L \leq \lfloor \log_{c_t} T \rfloor$ is the number of multigrid levels in time.

We can further extend the model to the full multigrid cycle, which is also pictured in Figure 4, as follows. First, we express (5) as a V-cycle from grid i to grid j:

$$V_i^j = 3T_C^i + 3T_F^i + \sum_{k=i+1}^{j-1}(3T_C^k + 4T_F^k) + 2T_C^j + 3T_F^j \tag{6}$$

The time taken by a full multigrid cycle can then be expressed as

$$T_{\text{FMG}} = \sum_{i=1}^{L-1} V_i^L, \tag{7}$$

which is a sum of V-cycles of depths $1, \ldots, L$, with L being the coarsest grid for each cycle.

IV. MAKING PREDICTIONS

The baseline performance model of MGRIT, introduced above, is very general in terms of how the cost of the time-stepping routine depends on the problem, the number of points, and the number of processes. This does not automatically allow for us to build a performance model that can be used in a practical setting in order to determine how many processes to use in space and how many to use in time. To build such a model, we focus on the relative costs of communication and computation. This enables us to make simplifying assumptions about the terms in the model in the previous section, which then leads to a way to measure them and make predictions.

Our first simplifying assumption deals with the communication involved in applying Φ. We assume that the communication involved in an application of Φ for the problem being solved takes a constant amount of time $\hat{\psi}$. This assumes that latency and overhead dominate the point-to-point communication involved in time-stepping, which is not unreasonable when doing time-stepping for the targeted settings, where the available parallelism in space has been exhausted, leading to a large number of small, latency dominated messages. We make one more assumption, which is to relate the time spent in collectives to the point-to-point time by the relation

$$\xi(P) = (a \log_2 P)\hat{\psi}. \tag{8}$$

This relates the time spent in collectives to the point-to-point communication cost, expressing the collective communication cost in terms of point-to-point messages along a binary tree, a typical implementation of collective operations, again especially for small message sizes. Finally, to aid in parameter measurement, we define a parameter $\hat{\xi} = a\hat{\psi}$. We do this because we will be using linear equations to measure the parameters, which will not be possible with the unknown coefficient a in the equations.

The parameters we end up measuring are $\phi(N)$, $\hat{\psi}$, and $\hat{\xi}$ for the problem we are solving. Given our earlier definitions of them, if we take m measurements $T_1, \ldots, T_m$ of the time-stepping routine Φ for different numbers of processes $P_1, \ldots, P_m$, we can solve the least squares problem

$$\begin{bmatrix} \frac{1}{P_1} & 1 & \log_2 P_1 \\ \vdots & \vdots & \vdots \\ \frac{1}{P_m} & 1 & \log_2 P_m \end{bmatrix} \begin{bmatrix} \phi(N) \\ \hat{\psi} \\ \hat{\xi} \end{bmatrix} = \begin{bmatrix} T_1 \\ \vdots \\ T_m \end{bmatrix} \tag{9}$$

for $\phi(N)$, $\hat{\psi}$, and $\hat{\xi}$. If $P_i = 1$ for some row i, we set the second and third entries in that row to 0.

It is possible that we know the value of $\hat{\xi}$ will be zero, or that we get an impossible value for $\hat{\xi}$ when performing this measurement. This can happen with an explicit time-stepping routine that has no collective communication, or a network with highly optimized and fast collective operations. In that case, we set $\hat{\xi}$ to zero, which also makes the coeffecient a relating it to $\hat{\psi}$ zero, and solve the least squares problem

$$\begin{bmatrix} \frac{1}{P_1} & 1 \\ \vdots & \vdots \\ \frac{1}{P_m} & 1 \end{bmatrix} \begin{bmatrix} \phi(N) \\ \hat{\psi} \end{bmatrix} = \begin{bmatrix} T_1 \\ \vdots \\ T_m \end{bmatrix} \tag{10}$$

for $\phi(N)$ and $\hat{\psi}$. If $P_i = 1$ for some row i, we set the second entry in that row to 0. Then, for a given process count and the solution to either (9) or (10), we can use (2) to estimate the time spent when using sequential time-stepping and (5)

[2]FAS is nonlinear multigrid, allowing XBraid to solve nonlinear problems.

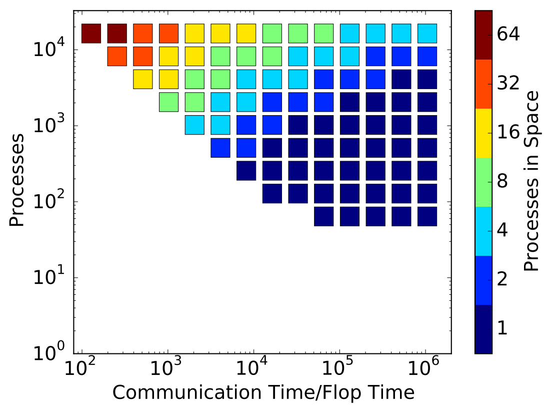

Fig. 5. Best process configurations for 128×128 spatial problem solved over 4096 time steps with explicit time-stepping. Colored areas correspond to using MGRIT with the corresponding number of processes in space. Areas with no color mean that sequential time-stepping is preferable.

multiplied by the number of cycles to estimate the time spent when using MGRIT.

This approach is based on the idea that the variance in computation and communication time across different process counts is too complicated to directly measure, and so we should instead strive for measuring average-case values that reflect the overall relative costs of communication and computation. These relative costs can then be used to give a reasonable idea of which configuration of processes in space and time will result in the best performance. Though the resulting performance model is relatively simple, it explains the observed performance, while avoiding unnecessary complexity that makes a performance model harder to use in practice. [3]

The underlying principle behind our approach is illustrated in Figure 5, which plots the estimated process configurations given the total number of processes used and the ratio of the time per communication round in Φ to the time per floating-point operation, assuming a model problem with 128×128 points in space solved with an explicit time-stepping scheme over 4096 time steps. For simplicity, no spatial coarsening is assumed, and the collective communication term $\hat{\xi}$ is assumed to be zero. The number of processes at which a switch to MGRIT over sequential time-stepping is beneficial, as well as the expected best number of processes to use in space and time, depends on the relative costs of communication and computation. Our prediction scheme is equivalent to finding where on the plot we are based on the time-stepping routine (x-axis) and the total process count (y-axis), and then looking up the process configuration that is expected to result in the best performance.

[3]One omitted complexity, which is not required for good modeling accuracy, is communication in the time dimension. This is because the problems considered here have a large spatial problem size that makes communication and computation in space dominant. However, it is possible that accounting for communication in time will be required in the future for problems with a small spatial dimension.

V. Results

We tested MGRIT and our model-based prediction scheme on two different problems on three different machines. The two problems are the 2D heat equation and a 2D advection-diffusion problem. We chose these problems as opposed to 3D problems with more demands on parallelism in space in order to free up resources to test a variety of parallel-in-time process configurations. As mentioned before, the convergence of MGRIT does not depend on the spatial dimension, so our approach is also applicable to 3D problems. We chose different time-stepping schemes and cycle types for each to highlight the versatility of MGRIT and the predictive performance model for making decisions on the process distribution in space and time. For each problem, we used our runs with sequential time stepping at the tested process counts to provide the measurements of the time-stepping routine that are needed by the performance model.

A. Machine Descriptions

Vulcan is an IBM Blue Gene/Q at Lawrence Livermore National Laboratory, consisting of 24,576 nodes with one 16-compute core 1.6 GHz processor per node. Experiments were run on up to 16,384 cores. All experiments use the IBM compiler, version 12.1, and the MPI implementation is an IBM-derived version of MPICH2.

Cab is a Linux cluster at Lawrence Livermore National Laboratory. It consists of 1,296 nodes with two eight-core 2.6 GHz Intel Xeon E5-2670 processors per node. The nodes are connected by an Infiniband QDR interconnect, organized as a two-level fat-tree. Experiments were run on up to 4,096 cores. All experiments use the Intel compiler, version 14.0.3. The MPI implementation is MVAPICH2.

Eos is a Cray XC30 at Oak Ridge National Laboratory, consisting of 744 nodes with two eight-core Intel Xeon E5-2670 processors per node. Experiments were run on up to 8,192 cores. All experiments use the Intel compiler, version 15.0.2, and the MPI implementation is Cray's native MPI.

Each machine has the capability to run multiple threads per core, four in the case of Vulcan and two in the case of Cab and Eos. For our experiments, we chose to only run one thread per core.

B. Heat Equation

Our first test problem is the 2D heat equation

$$u_t = u_{xx} + u_{yy}$$

on the unit square, with homogeneous Dirichlet boundary conditions, discretized using a 5-point stencil, with a 128×128 spatial grid, run for 4,096 time steps. The time-stepping routine is backward Euler, which necessitates solving a linear system of equations at each time step. The spatial solver is the semicoarsening geometric multigrid solver PFMG [12], [13] in the hypre library [14]. V-cycles were used for the multigrid-in-time solves, and we tested four different coarsening factors in time. The number of V-cycles required for MGRIT to converge

TABLE II
MGRIT CONFIGURATION PREDICTIONS FOR THE 2D HEAT EQUATION ON VULCAN.

P	T_{seq}	Model Best			Actual Best		
		$P_x \times P_t$	c_t	T_{par}	$P_x \times P_t$	c_t	T_{par}
128	53.02	128×1	–	53.02	1×128	4	53.01
256	49.93	2×128	8	41.22	2×128	8	41.22
512	48.88	2×256	4	31.46	2×256	4	31.46
1024	48.37	4×256	4	30.46	4×256	8	25.14
2048	47.53	8×256	4	21.21	4×512	4	20.80
4096	47.22	8×512	4	17.98	8×512	4	17.98
8192	46.63	16×512	4	16.77	8×1024	4	16.39
16384	46.32	16×1024	4	15.27	16×1024	4	15.27

was 9 for $c_t = 2$, and 8 for the others. Our XBraid convergence tolerance was 10^{-6}.

Results for each machine are in Figure 6. We compared sequential time-stepping with several different MGRIT configurations. The modeled best and actual best MGRIT configurations are highlighted with solid lines. In most cases, the model predicted a good mix of processes in space and time to use, often the best one. Speedups over the fastest time-stepping solution reached 3x on Vulcan, 3.6x on Cab, and 3.8x on Eos. In all cases, MGRIT enabled continued scalability well after the available spatial parallelism was exhausted.

We also utilized the model to predict the best overall configuration, treating the coarsening factor in addition to the number of processes in space and the number of processes in time as a quantity we needed to find. Finding the right coarsening factor (see Falgout et al. [1]) can by itself be a resource and time consuming task. Our predictions, compared with the actual best configurations, are in Tables II-IV. The first two columns list the process count and the runtime T_{seq} when using sequential time-stepping. The next three show the mix of parallelism in space and time and temporal coarsening factor predicted by the model to have the best performance, along with the runtime T_{par} for that configuration. The last three show the mix of parallelism in space and time and temporal coarsening factor that resulted in the best performance in our experiments, along with the corresponding runtime T_{par}. The uninteresting cases, where sequential time-stepping results in the best performance and is predicted to do so by the model, are omitted. Our model did not find the exact best configuration every time, but when it did not, it was still able to find a configuration that did not perform much worse than the best one. In addition, the configurations it selected resulted in improved performance compared to both sequential time-stepping and the configuration selected by the model for the next smallest number of processes.

C. Advection-Diffusion

For a more sophisticated example, we consider the advection-diffusion driver, `drive-05`, from XBraid, which implements

$$u_t - \nu \Delta \mathbf{u} + \mathbf{b} \cdot \nabla u = 0 \text{ for } u \in \Omega,$$

where $\nu = 0.2$ and $\mathbf{b} = [\sqrt{2/3}; \sqrt{1/3}]$. Figure 7 depicts the initial condition, the 2D hexagonal domain Ω and the periodic

TABLE III
MGRIT CONFIGURATION PREDICTIONS FOR THE 2D HEAT EQUATION ON CAB.

P	T_{seq}	Model Best			Actual Best		
		$P_x \times P_t$	c_t	T_{par}	$P_x \times P_t$	c_t	T_{par}
128	4.91	2×64	8	4.88	2×64	8	4.88
256	6.17	2×128	8	3.37	2×128	8	3.37
512	6.25	2×256	4	2.61	4×128	8	2.38
1024	8.46	4×256	4	1.83	4×256	4	1.83
2048	9.22	4×512	4	1.56	8×256	8	1.50
4096	11.2	8×512	4	1.28	8×512	4	1.28

TABLE IV
MGRIT CONFIGURATION PREDICTIONS FOR THE 2D HEAT EQUATION ON EOS.

P	T_{seq}	Model Best			Actual Best		
		$P_x \times P_t$	c_t	T_{par}	$P_x \times P_t$	c_t	T_{par}
128	5.42	2×64	8	4.96	2×64	8	4.96
256	7.43	2×128	8	3.44	4×64	8	3.38
512	7.42	2×256	4	2.65	4×128	8	2.37
1024	7.86	4×256	4	1.79	4×256	4	1.79
2048	7.97	4×512	4	1.54	8×256	4	1.41
4096	9.39	8×512	4	1.21	8×512	4	1.21
8192	10.2	8×1024	4	1.12	8×1024	4	1.12

spatial boundary conditions, visible at time-step 2500 as the solution begins to wrap around the spatial domain. The time domain is $t \in [0, 2.0]$.

To show generality, our implementation wraps the MFEM package [15] with XBraid, and we choose the fourth-order Runge-Kutta (RK4) time-stepper and linear discontinuous Galerkin finite elements in space. The use of an explicit time-stepping scheme requires simultaneous space-time coarsening so that the aforementioned CFL limit is not violated on coarse time grids, which results in an unstable and divergent solution. The CFL ratio $\delta t/h^2$, where h is the mesh width in space, must be kept below a fixed value on all time-grids, which for this discretization and PDE coefficients was experimentally determined to be around 0.16. For our experiments we used $\delta t/h^2 \approx 0.145$.

To accomplish this, we used an MGRIT solver with 4 time levels, where each successively coarser time-grid corresponded to a uniformly coarsened spatial mesh. That is, the spatial mesh size h grows by a factor of 2 when going to the next coarser time-grid, which corresponds to an overall spatial coarsening factor $c_x = 4$. To complement this, we choose the coarsening factor $c_t = 4$ in time, such that $\delta t/h^2$ remains roughly fixed on all time-grids. In this sense, explicit methods place restrictions on the available parameter choices to XBraid. Another restriction is that MGRIT converges best when the finest time-grid is close to the CFL limit, which is what we do here. This is not an obstacle, as running close to the CFL limit reduces the overall number of time steps and is how most sequential time-stepping codes operate. We also used full multigrid cycles for MGRIT as opposed to V-cycles, again for the purpose of obtaining the best convergence.

We ran `drive-05` for 16,384 time steps on each of the three machines, comparing sequential time-stepping with several different MGRIT configurations, analogous to the heat

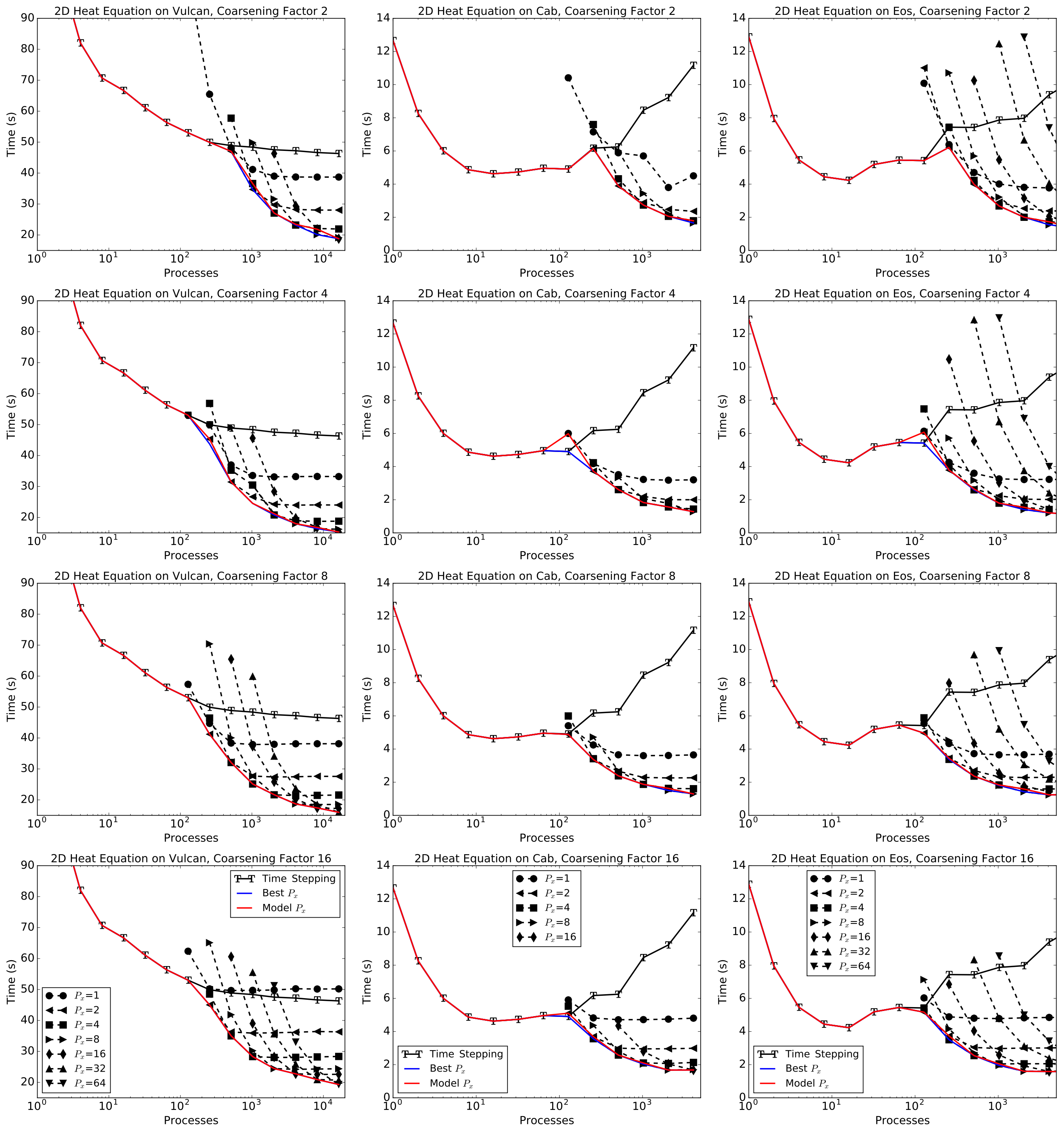

Fig. 6. Sequential time-stepping versus MGRIT when solving the 2D heat equation on Vulcan, Cab, and Eos for coarsening factors of 2 through 16.

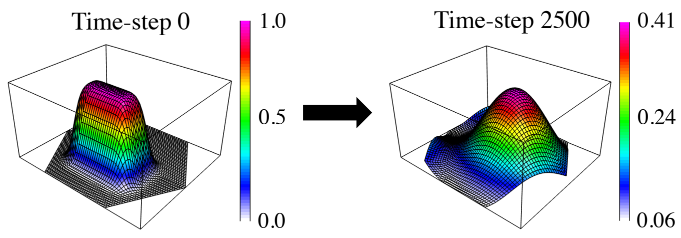

Fig. 7. Time-evolution of an advection-diffusion problem in XBraid. Note how the initial condition both convects and diffuses as time marches on.

TABLE V
MGRIT CONFIGURATION PREDICTIONS FOR THE ADVECTION-DIFFUSION PROBLEM ON VULCAN.

P	T_{seq}	Model Best		Actual Best	
		$P_x \times P_t$	T_{par}	$P_x \times P_t$	T_{par}
1024	10.6	1024×1	10.6	1024×1	10.6
2048	9.52	2048×1	9.52	2048×1	9.52
4096	7.82	4096×1	7.82	4096×1	7.82
8192	–	16×512	6.71	8×1024	6.39
16384	–	16×1024	4.43	16×1024	4.43

equation. The $\hat{\xi}$ term in the performance model was not explicitly set to zero. Ten MGRIT cycles were required for the solver to converge, and again, the XBraid tolerance we used was 10^{-6}. The results are in Figure 8. In almost every case, the model predicted either the best configuration or one very close to it. Model-predicted versus actual best configurations are listed for each machine in Tables V-VII. The first two columns list the process count and runtime when using sequential time-stepping. The next four show the mix of parallelism in space and time and corresponding runtime for the configuration predicted to have the best performance by the model and the configuration that resulted in the actual best runtime, respectively. As before, the uninteresting cases are omitted. We make special note of the fact that MGRIT enabled the use of process counts that could not be used with sequential time-stepping. For these counts, there were too many processes and too few spatial points for the mesh partitioner to assign to them, meaning that the simulation did not run. MGRIT, however, enabled scalability beyond this point. The highest speedups over the best time-stepping performance were 1.8x on Vulcan, 4.8x on Cab, and 6.4x on Eos.

VI. Future Directions

The performance model we developed can make decisions on the amount of parallelism to devote to space and the amount to devote to time in MGRIT, as well as suggest coarsening factors that result in superior performance. Still, there are many avenues for future work to build on our current foundation.

TABLE VI
MGRIT CONFIGURATION PREDICTIONS FOR THE ADVECTION-DIFFUSION PROBLEM ON CAB.

P	T_{seq}	Model Best		Actual Best	
		$P_x \times P_t$	T_{par}	$P_x \times P_t$	T_{par}
128	3.96	128×1	3.96	256×1	3.96
256	4.08	256×1	4.08	256×1	4.08
512	4.65	2×256	3.61	2×256	3.61
1024	6.55	2×512	1.93	4×256	1.90
2048	8.31	4×512	1.11	2×1024	1.10
4096	9.18	4×1024	0.71	4×1024	0.71

TABLE VII
MGRIT CONFIGURATION PREDICTIONS FOR THE ADVECTION-DIFFUSION PROBLEM ON EOS.

P	T_{seq}	Model Best		Actual Best	
		$P_x \times P_t$	T_{par}	$P_x \times P_t$	T_{par}
128	2.89	128×1	2.89	128×1	2.89
256	3.54	256×1	3.54	256×1	3.54
512	4.18	512×1	4.18	2×256	3.34
1024	5.45	2×512	1.80	4×256	1.76
2048	6.18	4×512	1.00	4×512	1.00
4096	7.61	4×1024	0.59	4×1024	0.59
8192	–	8×1024	0.45	8×1024	0.45

First, there are optimizations that can be done to get more performance out of MGRIT, such as reducing the solve tolerance for implicit time-stepping routines on coarse grids or adjusting the relaxation routine [1], [16]. Extending the model to cover them would provide additional helpful guidance to users to get the best possible performance.

Second, there are also more complicated problems and time-stepping schemes to consider. Nonlinear problems can be solved using MGRIT, but then each time step involves a nonlinear solver such as Newton's method, and the time taken by such a solver can vary substantially between time steps. Modeling this will be very useful, but is also a challenge. Time-stepping schemes that adjust the step size during a simulation have the potential to introduce load imbalance into the parallelism in time. A model that can cover this will help in tackling the resulting performance issues.

Finally, there is the interaction between MGRIT and the underlying computer architecture. The XBraid implementation of MGRIT expresses time-parallelism through message passing alone. Other forms of parallelism are now widely available, most notably in the form of multicore nodes and accelerators. Adapting MGRIT to express parallelism in more programming models than just message passing will become important as the amount of on-node concurrency increases. Also, there is the interaction of MGRIT with time-stepping routines that themselves use other programming models. In this scenario, it is possible that a mix of parallelism in space and time other than what would be suggested when only using message passing would make better use of the machine. Extending the model to cover both of these cases will be important as the on-node parallelism of machines continues to increase.

When it comes to modeling all of these additional features and future directions of MGRIT, there is also the need to balance detail with the ability to apply such a model in practice. There are a number of factors involved in doing so; Gahvari et. al. [17] discusses this in depth for a different solver. It is important that end users are able to select the features and mix of space/time parallelism they need for their particular applications without having to go through a lot of trial and error to find the right parameters. We designed

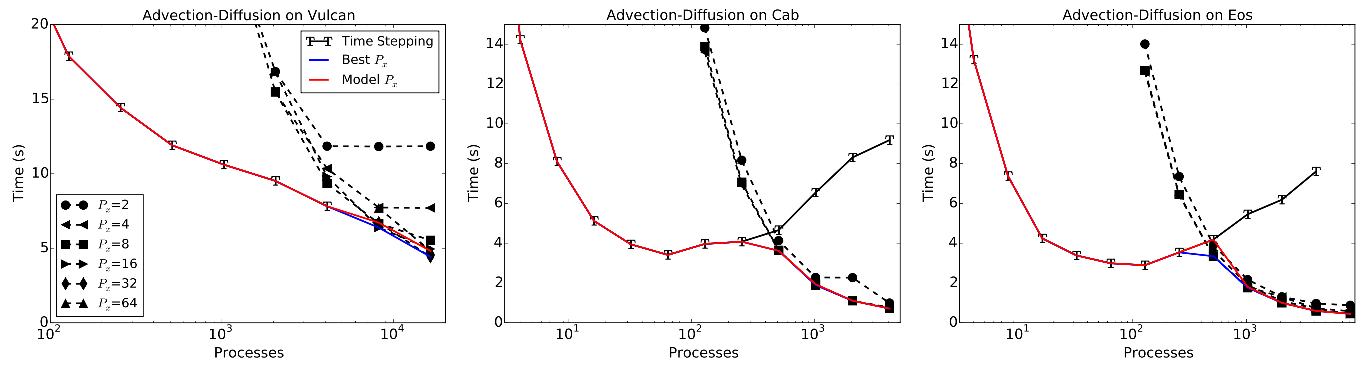

Fig. 8. Sequential time-stepping versus MGRIT when solving the advection-diffusion problem on Vulcan, Cab, and Eos.

our performance model with this goal in mind, and plan on adapting it for actual runtime use.

VII. Related Work

A. Parallel-in-time Methods

The field of parallel-in-time goes back at least 50 years [18] and includes a variety of approaches including direct methods as well as iterative approaches such as multiple shooting, domain decomposition, waveform relaxation, and multigrid. For an introduction to this history, see the review paper by Gander et al. [19]. Our work focuses on multigrid approaches (and MGRIT in particular) because of multigrid's optimal algorithmic scaling for both parallel communication and number of operations. An additional attraction of MGRIT is its non-intrusive nature.

Regarding the use of multigrid-in-time methods in the HPC setting, Carracciuolo et al. [20] propose the use of MGRIT across nodes and spatial parallelism within nodes, facilitated by implementing this framework within an existing software library. However, no results are presented.

B. Setting Application Parameters with Performance Models

The rapid pace of evolution in computer architecture necessitates software being able to adapt to a changing environment. There are frequently tunable parameters that need to be adjusted to ensure good performance. One very common example is tiling in linear algebra operations to maximize the amount of computation performed in registers or fast memory. Lang [21] gives an overview of existing work in this area. Much of it involves linear algebra kernels, but some more substantial applications have also seen benefits. Examples given are host/accelerator workload distribution in the conjugate gradient method [22], tree depth in the fast multipole method [23], tiling in explicit predictor-corrector Runge-Kutta ODE solvers [24], algorithm and parameter choice in sorting [25], and load balance and keeping in data in cache during molecular dynamics visualization [26]. Our work falls in this same general area, but tackles a more difficult application in that the time-stepping routine can be arbitrary, ranging from a simple calculation to an iterative solve, whereas past work has focused on specific algorithms or choice among a finite number or specific class of algorithms.

VIII. Conclusions

We presented a performance model that gives guidance on what mix of processes to use in space and time for the parallel-in-time solver MGRIT, and demonstrated the effectiveness of the model on a simple finite difference problem and a more complicated finite element problem. The two problems were solved using two very different time-stepping routines and cycling strategies. The former used the implicit backward Euler time-stepper and V-cycle multigrid, and the latter used the explicit RK4 time-stepper and full multigrid.

Performance models like the one we presented will be key to making parallel-in-time methods accessible to users. In addition to MGRIT, there are opportunities for the development of additional methods as well as methods tailored to emerging massively parallel machines. With their ability to strong-scale time-dependent simulations well beyond the capabilities of sequential time-stepping, parallel-in-time methods will become increasingly important as the parallelism of machines increases. The range of possible mixes of parallelism in space and time will be substantial, and features designed to take the underlying architecture into account will add even more knobs. Without a performance model to provide guidance, users will have to invest significant time and resources tuning in order to get the best performance out of their applications.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. It used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. LLNL-CONF-701995.

References

[1] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder, "Parallel Time Integration with Multigrid," *SIAM Journal on Scientific Computing*, vol. 36, pp. C309–C334, 2014.

[2] R. D. Falgout, A. Katz, T. Kolev, J. B. Schroder, A. Wissink, and U. M. Yang, “Parallel time integration with multigrid reduction for a compressible fluid dynamics application.” *Lawrence Livermore National Laboratory Technical Report, LLNL-JRNL-663416*, 2015.

[3] M. Ries, U. Trottenberg, and G. Winter, “A note on MGR methods,” *Linear Algebra Appl.*, vol. 49, pp. 1–26, 1983.

[4] M. Ries and U. Trottenberg, “MGR-ein blitzschneller elliptischer löser,” Universität Bonn, Tech. Rep. Preprint 277 SFB 72, 1979.

[5] “XBraid: Parallel multigrid in time,” http://www.llnl.gov/casc/xbraid.

[6] R. D. Falgout, A. Katz, T. V. Kolev, J. B. Schroder, A. M. Wissink, and U. M. Yang, “Parallel Time Integration with Multigrid Reduction for a Compressible Fluid Dynamics Application,” Lawrence Livermore National Laboratory, Tech. Rep. LLNL-JRNL-663416, October 2014.

[7] V. Dobrev, T. Kolev, N. A. Petersson, and J. Schroder, “Two-level convergence theory for parallel time integration with multigrid,” *SIAM Journal on Scientific Computing, LLNL-JRNL-692418*, 2016 (submitted).

[8] M. J. Gander and S. Vandewalle, “Analysis of the parareal time-parallel time-integration method.” *SIAM Journal on Scientific Computing*, vol. 29, pp. 556–578, 2007.

[9] 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.*, vol. 332, no. 7, pp. 661–668, 2001.

[10] R. Courant, K. Friedrichs, and H. Lewy, “On the Partial Difference Equations of Mathematical Physics,” *IBM Journal of Research and Development*, vol. 11, pp. 215–234, 1967.

[11] A. Brandt, “Multi-level adaptive computations in fluid dynamics,” 1979, technical Report AIAA-79-1455, AIAA, Williamsburg, VA.

[12] S. F. Ashby and R. D. Falgout, “A Parallel Multigrid Preconditioned Conjugate Gradient Algorithm for Groundwater Flow Simulations,” *Nuclear Science and Engineering*, vol. 124, pp. 145–159, 1996.

[13] R. D. Falgout and J. E. Jones, “Multigrid on Massively Parallel Architectures,” in *Lecture Notes in Computational Science and Engineering volume 14. Multigrid Methods VI: Proceedings of the Sixth European Multigrid Conference Held in Gent, Belgium, September 27-30, 1999*. Springer, 2000, pp. 101–107.

[14] “*hypre*: High performance preconditioners,” http://www.llnl.gov/CASC/hypre/.

[15] “MFEM: Modular finite element methods,” www.mfem.org.

[16] R. Falgout, S. Friedhoff, T. Kolev, S. MacLachlan, J. Schroder, and S. Vandewalle, “Multigrid Methods with Space-Time Comcurrency,” *SIAM Journal on Scientific Computing, LLNL-JRNL-678572*, 2015 (submitted).

[17] H. Gahvari, “Improving the Performance and Scalability of Algebraic Multigrid Solvers through Applied Performance Modeling,” Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2014.

[18] J. Nievergelt, “Parallel methods for integrating ordinary differential equations,” *Comm. ACM*, vol. 7, pp. 731–733, 1964.

[19] M. J. Gander, *50 years of Time Parallel Time Integration*, ser. Multiple Shooting and Time Domain Decomposition. Springer, 2015, in press.

[20] L. Carracciuolo, L. D’Amore, and V. Mele, “Toward a fully parallel Multigrid in Time algorithm in PETSc environment: a case study in ocean models,” in *2nd International Workshop on High Performance Computing for Weather, Climate, and Solid Earth Sciences (HPC-WCES 2015)*, 2015.

[21] J. Lang, “Data-aware tuning of scientific applications with model-based autotuning,” *Concurrency and Computation: Practice and Experience*, To appear.

[22] J. Lang and G. Rünger, “An execution time and energy model for an energy-aware execution of a conjugate gradient method with cpu/gpu collaboration,” *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2884–2897, 2014.

[23] H. Dachsel, M. Hofmann, J. Lang, and G. Rünger, “Automatic Tuning of the Fast Multipole Method Based on Integrated Performance Prediction,” in *14th International Conference on High Performance Computing and Communications*, 2012, pp. 617–624.

[24] N. Kalinnik, M. Korch, and T. Rauber, “Online auto-tuning for the time-step-based parallel solution of ODEs on shared-memory systems,” *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2722–2744, 2014.

[25] X. Li, M. J. Garzarán, and D. Padua, “A Dynamically Tuned Sorting Library,” in *2004 International Symposium on Code Generation and Optimization*, 2004.

[26] Y. L. Nelson, B. Bansal, M. Hall, A. Nakano, and K. Lerman, “Model-Guided Performance Tuning of Parameter Values: A Case Study with Molecular Dynamics Visualization,” in *22nd IEEE International Parallel and Distributed Processing Symposium*, 2008.