LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-681918

PARALLEL-IN-TIME FOR MOVING MESHES

R. D. Falgout, T. A. Manteuffel, B. Southworth, J. B. Schroder

February 4, 2016

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

PARALLEL-IN-TIME FOR MOVING MESHES

R.D. FALGOUT *, T.A. MANTEUFFEL †, B. SOUTHWORTH †, AND J.B. SCHRODER *

Abstract. With steadily growing computational resources available, scientists must develop effective ways to utilize the increased resources. High performance, highly parallel software has become a standard. However until recent years parallelism has focused primarily on the spatial domain. When solving a space-time partial differential equation (PDE), this leads to a sequential bottleneck in the temporal dimension, particularly when taking a large number of time steps. The XBraid parallel-in-time library was developed as a practical way to add temporal parallelism to existing sequential codes with only minor modifications. In this work, a rezoning-type moving mesh is applied to a diffusion problem and formulated in a parallel-in-time framework. Tests and scaling studies are run using XBraid and demonstrate excellent results for the simple model problem considered herein.

1. Introduction. Future computer architectures are trending towards an increase in processors and memory, not faster individual processors. This means that for software to effectively utilize modern architectures, it must incorporate increased parallelism. For the most part, parallelism in scientific computing is focused on the spatial domain of a problem, while time stepping remains sequential, which imposes a limit on the possible concurrency. Adding parallelism to the temporal dimension has been considered in a number of works [4–6, 10–12], and only recently has it become increasingly important due to the direction of computer architecture development. One particular parallel-in-time method is the XBraid multigrid reduction in time C-library, developed at Lawrence Livermore National Laboratory [2].

A brief overview of the theory behind XBraid can be found in Section 2.3, and a detailed description in [5, 6]. The objective of XBraid is to be 'non-intrusive,' in the sense that with only minor modifications, existing codes with sequential time stepping routines can be wrapped with XBraid to add parallelism in the temporal dimension. One consequence of this approach is a larger total cost to solve a space-time PDE, but this cost can be efficiently distributed over an increased number of processors. XBraid is unique in that it can easily be applied to existing sequential codes. There has been other work on parallel-in-time solvers, most notably parareal [11] and PFASST [4], but they generally lack either the non-intrusive feature of XBraid (PFASST), or practical speedup over sequential time stepping (parareal). Nonetheless, for scientists to wrap their existing code with XBraid, they must be confident that XBraid is well suited for their problem and code. In this regard, it must be demonstrated that XBraid is compatible with standard numerical methods, including adaptive mesh methods.

Adaptive mesh methods are common in large scale numerical simulations, wherein the mesh underlying the physical PDE is altered in some manner that is advantageous to solving the PDE numerically. Such methods can be broadly broken into two categories: mesh refinement and moving meshes. Moving mesh methods can also be grouped in two categories: the 'quasi-Lagrange' approach and the 'rezoning approach.' In quasi-Lagrange methods, mesh points move continuously in time, wherein time derivatives of the physical PDE are transformed onto mesh trajectories and an additional convective term is introduced corresponding to mesh movement [13]. This approach is standard in computational fluid dynamics, where the mesh moves with

*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore,CA 94551. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-TR-******

†Department of Applied Mathematics, University of Colorado at Boulder, Boulder, Colorado

fluid flow. The rezoning approach keeps mesh points fixed at discrete time steps, and at each step they are relocated to an optimal location in space with respect to the current solution [13]. A rezoning-type moving mesh is the focus of this work.

In this paper, a parallel-in-time moving mesh is applied to a 1-dimensional diffusion PDE and wrapped with XBraid. Section 2.1 introduces the physical PDE and discretization, and Section 2.2 covers background theory on moving meshes and the moving mesh PDE (MMPDE) implemented. The parallel-in-time framework is introduced in Section 2.3, along with an appropriate formulation of the PDE and MMPDE of interest to be wrapped with XBraid. Numerical results, including test problems and scaling studies, are presented in Section 3, and a short discussion on conclusions and future work can be found in Section 4.

2. Problem formulation.

2.1. Physical PDE. Consider the 1-dimensional diffusion equation subject to Dirichlet boundaries and a space-time dependent forcing function

$$\begin{aligned} u_t &= k u_{xx} + f(x,t), \\ u(0,t) &= u(1,t) = 0, t > 0 \\ u(x,0) &= u_0(x). \end{aligned} \tag{2.1}$$

A method of lines approach is used to form a corresponding discrete problem, wherein the spatial term, ku_{xx}, is discretized using a finite element weak form. The discrete problem is then given as the time dependent ODE $M\mathbf{u}_t + A\mathbf{u} = \mathbf{b}$, for mass matrix M, stiffness matrix A, and vector $\mathbf{b}$. Discretizing $\mathbf{u}_t$ using backward Euler, and defining $\delta\mathbf{u} := \frac{\mathbf{u}^{(t+1)} - \mathbf{u}^{(t)}}{\delta t}$ gives

$$\begin{aligned} & M\delta\mathbf{u} + A(\mathbf{u}^{(t)} + \delta t \delta\mathbf{u}) = \mathbf{b}^{(t+1)}, \\ \iff \quad & (M + \delta t A)\delta\mathbf{u} = -A\mathbf{u}^{(t)} + \mathbf{b}^{(t+1)}. \end{aligned} \tag{2.2}$$

The finite element discretization of Eq. (2.1) is done in the MFEM finite element library [1], and all linear solves are performed using HYPRE [9].

2.2. Moving mesh. A physical PDE as in Eq. (2.1) has some physical domain, Ω_p, associated with it, in this case a 1-dimensional space $x \in [a, b]$. Now, for a fixed time consider a corresponding 1-dimensional computational domain, Ω_c, with coordinates, $\zeta \in [0, 1]$, and an invertible map

$$\begin{aligned} x &= x(\zeta), \\ \zeta &= \zeta(x). \end{aligned}$$

In solving a discrete physical PDE for solution $\mathbf{u}$, this transformation is picked such that, at a given time, t, the solution in the transformed spatial variable,

$$\hat{\mathbf{u}}(\zeta, t) = \mathbf{u}(x(\zeta, t), t),$$

is smooth and easy to approximate using a uniform mesh over ζ. A moving mesh as a function of time on Ω_p, $\mathcal{T}_h(t)$, is defined as

$$\mathcal{T}_h(t): \quad x_j(t) = x(\zeta_j, t), \quad j = 1, ..., n,$$

for nodes $j = 1, ..., n$, and a fixed uniform mesh on Ω_c,

$$\mathcal{T}_h^c : \quad \zeta_j = \frac{j-1}{n-1}, \quad j = 1, ..., n.$$

A *mesh density function*, $K(x, t_i) > 0$, is then chosen that provides some measure of how well-suited a mesh is for a given approximation to our physical PDE at time t_i. For a given physical mesh, $\mathcal{T}_h(t_i) : 0 = x_1 < ... < x_n = 1$ and mesh density function, K, the optimal mesh is defined to satisfy the equidistribution principle, that is

$$\int_{x_1}^{x_2} K(x, t_i)dx = ... = \int_{x_{n-1}}^{x_n} K(x, t_i)dx.$$

Several popular choices of mesh density functions are arc length and curvature, as well as optimal linear error interpolation under the L^2 and H^1-norms [13]. In this work arc length, given in Eq. (2.3), is used as a simple and effective choice of mesh density function,

$$K(x, t) = \sqrt{1 + u_x^2}. \tag{2.3}$$

Fig. 1 demonstrates why an equidistributed mesh with respect to arc length is desirable over a uniform mesh. Note that in some literature, $K(x, t)$ is referred to as the monitor function. Herein, $K(x, t)$ is referred to as the mesh density function and $K(x, t)^2$ as the monitor function, as this is consistent with moving mesh terminology in higher dimensions [13].

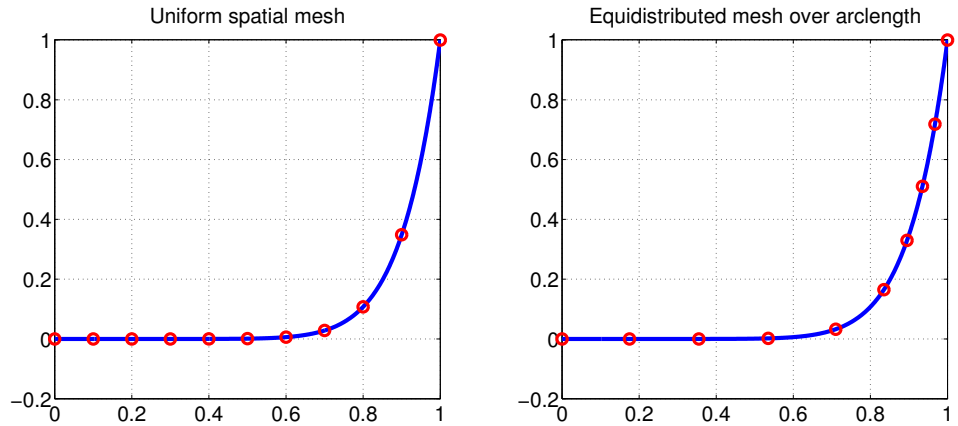

Fig. 1: Two 10-point meshes over $f(x) = x^{10}, x \in [0, 1]$. The equidistributed mesh with respect to arclength is shown on the right – notice that it includes more points close to the single feature in the function than a uniform mesh, making it a easier to approximate the continuous form.

For a time-dependent physical PDE, a coordinate transformation can be constructed to generate an optimal mesh at each time step. However, this method is computationally expensive, and can also result in large mesh movement on progressive time steps. The solution at each time must be interpolated to the new mesh, and too much movement between successive meshes degrades accuracy in interpolation. Both of these issues can be addressed by formulating and solving equations for the velocity of each mesh node at time t, and updating nodes at $t + \delta t$ accordingly. This

is (i) computationally less expensive than coordinate transformations at each time step, and (ii) provides a degree of temporal smoothing in mesh movement. A mesh equation involving the mesh speed is referred to as a moving mesh PDE (MMPDE), and is the strategy utilized in this work [13].

A full derivation of MMPDEs can be found in [13]. Conceptually, the MMPDE is constructed so that mesh nodes move in the direction of equidistribution with respect to $K(x,t)$. A general MMPDE form is given in Eq. (2.4), where P is some positive-definite operator to be chosen by the user, and $\tau > 0$ is a user-specified parameter controlling how rapidly mesh movement responds to changes in K,

$$\frac{\partial x}{\partial t} = \frac{1}{\tau}\frac{\partial x}{\partial \zeta} P\left(K\frac{\partial x}{\partial \zeta}\right)^{-2}\left(\frac{\partial x}{\partial \zeta}\right)^{-1}\frac{\partial}{\partial \zeta}\left(K\frac{\partial x}{\partial \zeta}\right). \tag{2.4}$$

There are many possible choices of P, discussed in detail in [13]. Choosing $P = (Kx_\zeta)^2$ gives the MMPDE

$$x_t = \frac{1}{\tau}\frac{\partial}{\partial \zeta}\left(K\frac{\partial x}{\partial \zeta}\right). \tag{2.5}$$

The right hand side of Eq. (2.5) forces the mesh towards equidistribution with respect to the mesh density function K. When equidistribution is satisfied, this term is zero and thus there will be zero mesh velocity and movement. For a fixed physical domain, boundary nodes must remain stationary, and so zero Dirichlet boundary conditions are applied to Eq.(2.5).

Eq. (2.5) is discretized using backward Euler in time and central finite difference schemes in space. Central differences give a tridiagonal operator that is second order in space and satisfies the CFL condition with a given first-order, time-stepping method like backward Euler. The specific discretization in Eq. (2.6) also gives an elliptic operator, which is well-suited for solving with spatial multigrid [3]. A second order central difference scheme is used to approximate $(u_x)_j$, which yields

$$\begin{aligned}
&\frac{x_j^{(i)} - x_j^{(i-1)}}{\delta t} = \frac{1}{\tau}\left[\frac{\partial}{\partial \zeta}\left(K^{(i)}\frac{\partial x^{(i)}}{\partial \zeta}\right)\right]_j, \qquad \text{where} \\
&\left[\frac{\partial}{\partial \zeta}\left(K\frac{\partial x}{\partial \zeta}\right)\right]_j = \frac{(K_{j+1}+K_j)(x_{j+1}-x_j)}{2(\Delta\zeta)^2} - \frac{(K_j+K_{j-1})(x_j-x_{j-1})}{2(\Delta\zeta)^2}, \\
&K_j = \sqrt{1+(u_x)_j^2}.
\end{aligned} \tag{2.6}$$

2.3. Braid Implementation:. A space-time PDE can be formulated using the method of lines as an ODE

$$\begin{aligned}
&\mathbf{u}^{(i)} = \Phi\mathbf{u}^{(i-1)} + \mathbf{g}^{(i)}, \quad i = 1, 2..., N_t \\
&\mathbf{u}^{(0)} = \mathbf{u}(0),
\end{aligned} \tag{2.7}$$

where Φ is an operator to advance spatial points one time step, δt, constructed from a spatial discretization and time stepping scheme. This forms a system of equations over the temporal domain, $t_0, ..., t_{N_t}$, as

$$A\mathbf{u} = \mathbf{g} \quad \Longrightarrow \quad \begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix}\begin{pmatrix} \mathbf{u}^{(0)} \\ \mathbf{u}^{(1)} \\ \vdots \\ \mathbf{u}^{(N_t)} \end{pmatrix} = \begin{pmatrix} \mathbf{g}^{(0)} \\ \mathbf{g}^{(1)} \\ \vdots \\ \mathbf{g}^{(N_t)} \end{pmatrix}. \tag{2.8}$$

In a standard multigrid fashion, points in time are split into a fine grid, F, and coarse grid C. The form of A in Eq. (2.8) allows for explicit forming of the ideal interpolation and restriction operators, $P = \begin{pmatrix} -A_{ff}^{-1}A_{fc} & I_c \end{pmatrix}^T$, $R = \begin{pmatrix} -A_{cf}A_{ff}^{-1} & I_c \end{pmatrix}$ [8], respectively, and a Petrov-Galerkin coarse grid operator is given by

$$A_\Delta = RAP = \begin{pmatrix} I & & & \\ -\Phi^{k_1+1} & I & & \\ & \ddots & \ddots & \\ & & -\Phi^{k_{N_c-1}+1} & I \end{pmatrix}, \tag{2.9}$$

where $k_i \geq 1$ is the number of F-points between the ith and $(i+1)$th C-points and N_c the size of the coarse grid. Notice that at the ith time step, the Galerkin coarse grid operator (Eq. (2.9)) takes $k_i + 1$ steps of the fine grid step size, which is equivalent to the action of the fine grid operator. This defeats the purpose of a coarse grid, so consider the non-Galerkin operator

$$B_\Delta = \begin{pmatrix} I & & & \\ -\Phi_{\Delta_1} & I & & \\ & \ddots & \ddots & \\ & & -\Phi_{\Delta_{N_c}} & I \end{pmatrix}, \tag{2.10}$$

where Φ_{Δ_i} takes a time step of size $\Delta_i = \delta t(k_i + 1)$. XBraid solves Eq. (2.8) using a full approximation storage (FAS) multigrid scheme [3], with ideal interpolation and the coarse grid operator in Eq. (2.10). The user must only provide a function $\Phi(\delta t)$ that takes a time step of size δt, and a few other simple routines.

To apply XBraid to the moving mesh problem, the discretized physical PDE and MMPE are coupled and the solution is stored in a block vector format $\mathbf{v}^{(i)} = \begin{pmatrix} \mathbf{x}^{(i)} & \mathbf{u}^{(i)} \end{pmatrix}^T$, where $\mathbf{x}^{(i)}$ is the mesh and $\mathbf{u}^{(i)}$ the physical solution at time t_i. Notice in Eq. (2.6) that solving for the ith mesh is implicit upon evaluating the mesh density function $K(x,t)$ with $\mathbf{x}^{(i)}, \mathbf{u}^{(i)}$. Coupling the MMPDE and physical PDE in this fashion results in a fully nonlinear, time-stepping routine. Instead of solving the fully nonlinear PDE, the system is linearized by evaluating $K(x,t)$ on $\mathbf{u}^{(i)}$ and $\mathbf{x}^{(i-1)}$ when solving for $\mathbf{x}^{(i)}$. The basic outline of $\Phi(\delta t)$ for the (linearized) moving mesh algorithm used in this paper is then

1. Move the mesh with respect to the current approximation, $u^{(i)}$.
2. Interpolate solution values from the old mesh to the new mesh.
3. Take a time step in the physical PDE.

3. Numerical results.

3.1. Test problems. Example 1: Consider the problem

$$u_t - \frac{u_{xx}}{2} = f(x,t), \tag{3.1}$$

$$u(0,t) = u(1,t) = 0, \tag{3.2}$$

$$u(x,0) = 0, \tag{3.3}$$

where $f(x,t)$ is defined as a Gaussian bump moving across the spatial domain in time, given by

$$f(x,t) = \begin{cases} e^{\frac{-1}{1-\left(\frac{x-0.5(t+0.25)}{0.05}\right)^2}} & t \in [0, 1.5] \\ 0 & t > 1.5 \end{cases}. \tag{3.4}$$

At any fixed time, $t \leq 1.5$, $f(x,t)$ is a Gaussian bump of width $w = 0.05$, centered at $c = \frac{t+0.25}{2}$. Hypre is used to perform the linear solve associated with each time step [9], and the time-stepping routine is wrapped in XBraid to add parallelism to the temporal dimension via multigrid V-cycles in time. Results of applying XBraid V-cycles in time to this problem with 100 time steps over $t \in [0, 2.4]$ and 30 spatial unknowns are given in Fig. 2. The average convergence factor for this problem is CF ≈ 0.05.

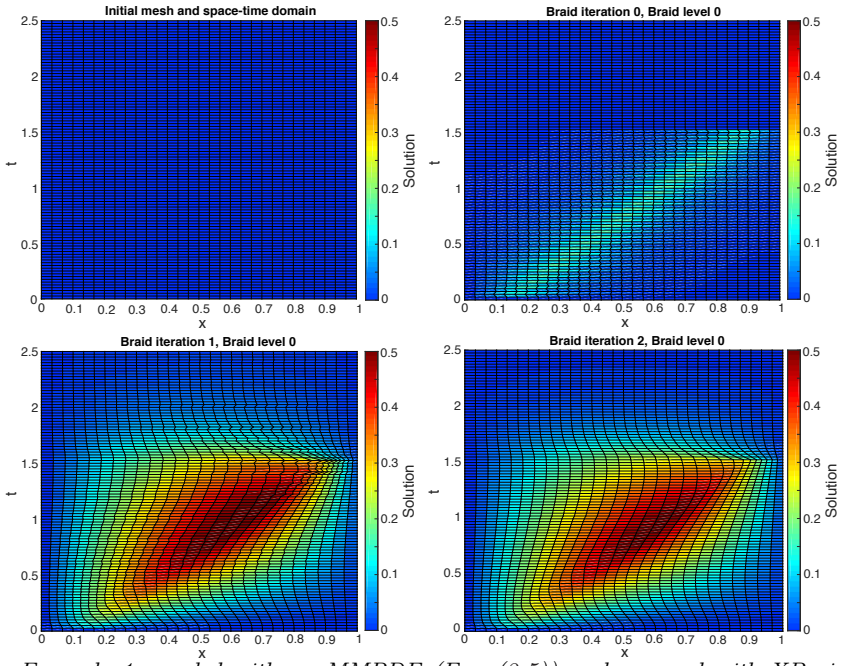

Fig. 2: Example 1 coupled with an MMPDE (Eq. (2.5)) and wrapped with XBraid. The physical solution in space-time is shown as a heat map, and the space-time mesh is shown as black lines overlaying the physical PDE solution.

Fig. 2 is a proof of concept, but uses a very small spatial and temporal problem size. As problem size increases, stability with regards to mesh location is problematic using standard V-cycles. Specifically, the mesh can be erroneously moved outside of the physical domain during initial V-cycles due to a poor current approximation to the solution. This can lead to a degradation in convergence or even divergence of iterations. However, this problem can be addressed by using full multigrid (FMG) cycles in time, an option provided in XBraid. FMG cycles allow the solver to begin resolving mesh location and the physical solution on a coarse temporal grid. Experimentally, when the finest temporal grid is reached, each solution has a sufficiently accurate approximation that mesh movement will remain inside the physical domain. If this does not immediately resolve a mesh stability issue, the amount of mesh movement can also be adjusted using the parameter τ in Eq. (2.6).

Example 2: Now, consider the same problem as Ex. 1 with a set of five time-dependent Gaussian bump sources with domains $[x_0, x_1] \times [t_0, t_1]$ and strength (leading constant) S given by

$$f_1 : \quad [0.85, 0.95] \times [0.05, 0.15], \quad S = 1500,$$

$$
\begin{aligned}
f_2: &\quad [0.15, 0.45] \times [0.05, 0.45], \quad S = 900, \\
f_3: &\quad [0.20, 0.80] \times [0.50, 0.70], \quad S = 200, \\
f_4: &\quad [0.70, 0.90] \times [0.50, 1.10], \quad S = 1200, \\
f_5: &\quad [0.10, 0.50] \times [0.80, 1.00], \quad S = 900.
\end{aligned}
$$

The purpose of Example 2 is to create a solution with sharper gradients, making the physical PDE, and particularly mesh movement, more difficult. Applying FMG to this problem with $(N_x, N_t) = (40, 40^2)$, convergence in the 'eyeball norm' takes 5 iterations, and convergence to 10^{-9} accuracy in the discrete l^2-norm takes 8 iterations. The solution on iterations $0, 1, 2$, and 4 is given in Fig. 3. Examples 1, 2 are done on a small problem size, primarily for visualization purposes to see how the space-time mesh moves with the physical solution. Success on larger problems can be seen in scaling studies in Sec. 3.2.

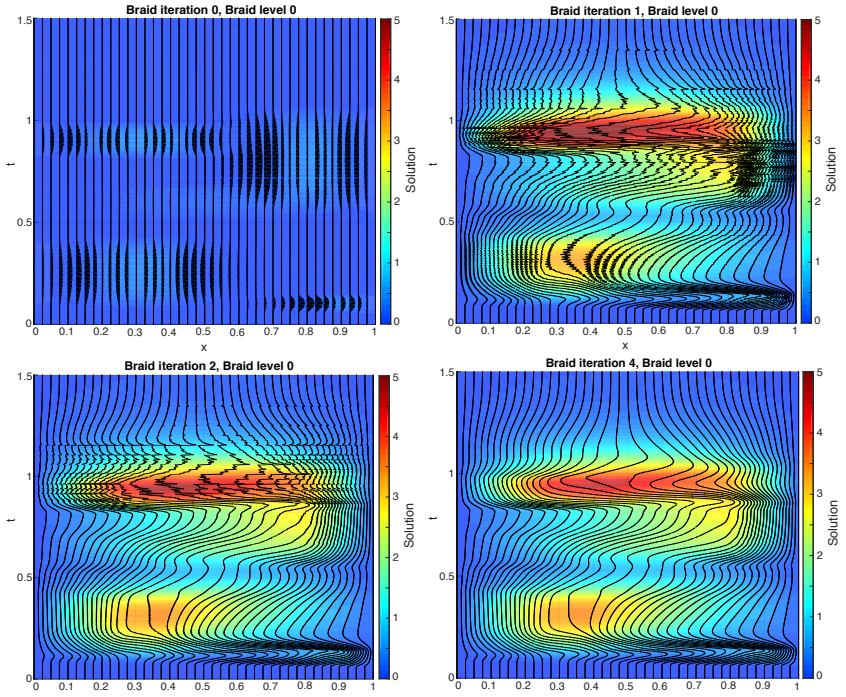

Fig. 3: Example 2 coupled with an MMPDE (Eq. (2.5)) and wrapped with XBraid. The physical solution in space-time is shown as a heat map, and the spatial mesh is shown as black lines overlaying the physical PDE solution.

3.2. Scaling.. A strong scaling study is solving a fixed problem size and considering the speedup as the number of processors is increased, while a weak scaling study is fixing the problem size per processor and increasing the number of processors. For each of these studies, a relative error tolerance is used as the stopping criterion. Theoretically, the stopping criterion is given by residual tolerance in the L^2 norm,

$$
\|r\|_{L^2} = \sqrt{\int_{\Omega_t} \int_{\Omega_x} |r|^2 dx dt} < \text{tol}, \tag{3.5}
$$

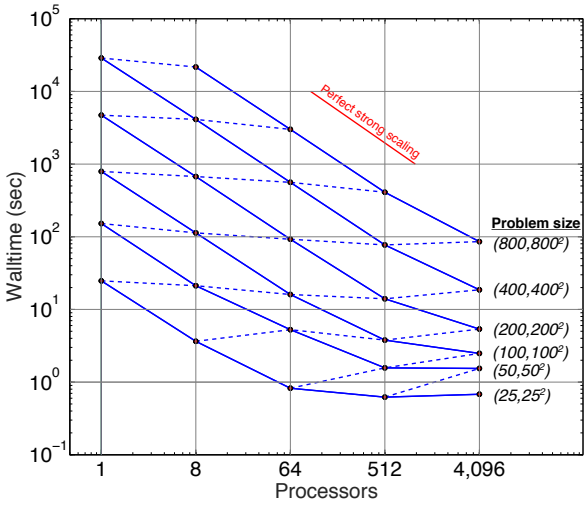

Fig. 4: Strong scaling in solid lines and weak scaling in dotted lines, with perfect strong scaling shown in red and perfect weak scaling being flat. Total problem size is shown as (N_x, N_t). Observe that for many problem sizes and processors, near perfect strong and weak scaling is obtained. The scaling is suboptimal when the problem size is not sufficiently large for the number of processors, e.g. distributing a problem of size $(50, 50^2)$ across 4096 processors.

for some continuous residual, r, and spatial and temporal domains, Ω_x and Ω_t, respectively. Let $\hat{\mathbf{r}}$ be the discrete residual computed in XBraid. Then Eq. (3.5) is approximated as $\|r\|_{L^2} \approx \|\hat{\mathbf{r}}\|_{l^2} \cdot \sqrt{\delta t \Delta x}$, and the relative stopping tolerance defined as

$$\widehat{\text{tol}} = \frac{\text{tol}}{\sqrt{\delta t \Delta x}},$$

where $\text{tol} = 10^{-10}$ is the absolute tolerance.

Scaling studies are run on Example 2, with smallest problem size $(N_x, N_t) = (25, 25^2)$. The number of spatial points, N_x, is then increased by factors of two and the ratio $\delta t = \Delta x^2$ maintained, up to a largest size of $(800, 800^2)$. Note that the algorithm used is spatially serial, and thus, for the scaling study all processors are used in the temporal dimension. This is primarily for ease of implementation, as a spatially parallel moving mesh requires a nontrivial communication algorithm to interpolate between meshes stored on different processors, and because this work is focused on demonstrating parallel-in-time capabilities.

Results of the strong and weak scaling studies are shown in Fig. 4. Observe that for sufficiently large problem sizes, XBraid achieves near perfect strong and weak scaling. From an algorithmic standpoint, how the number of required FMG iterations scales with increased problem size is also of interest. Table 1 shows the FMG iterations for all problem sizes considered for Examples 1 and 2. Observe that the number of iterations actually improves as the problem size increases. A likely explanation for this is that the non-linear solve becomes better conditioned as the grid is refined and that this is particularly beneficial for the simple linearization used here.

Grid size:	$(25, 25^2)$	$(50, 50^2)$	$(100, 100^2)$	$(200, 200^2)$	$(400, 400^2)$	$(800, 800^2)$
Ex. 1	7	6	6	5	4	3
Ex. 2	13	12	10	9	8	6

Table 1: FMG iterations to converge to relative tolerance, $\widehat{tol}$, based on an absolute tolerance of $tol = 10^{-10}$.

Scaling studies measure the parallel efficiency of a numerical method, but it is also important to consider how the method performs relative to other methods. Comparing the wall time required to solve the problem using XBraid with the wall time of a temporally serial implementation (all parallelism done spatially) gives the speedup that parallel-in-time provides over a serial time stepping scheme. Because the moving mesh algorithm used herein is currently serial, this comparison has yet to be done. For the 1-dimensional model problem considered, it is likely that spatial parallelism will be faster than temporal due to the small problem sizes. However, for higher dimensions and more difficult problems with significantly more unknowns, the parallel-in-time approach is expected to overcome spatial parallelism with respect to wall time, e.g. see [5, 7].

Generally, for a given spatial problem size, there is a point at which it is no longer beneficial to add processors to the spatial domain. In this case a bottleneck develops in the temporal dimension, wherein a substantial part of the code remains strictly serial, while increasing the number of processors does not decrease wall times. Such a bottleneck is particularly problematic for numerical schemes that require a substantial number of time steps, perhaps due to a CFL condition, $\delta t = \Delta x^2$, or simply due to a large temporal domain. This is where XBraid comes in – XBraid requires more floating point operations to solve a space-time problem than a temporally serial implementation and is, thus, not optimal for small problems. However, when sufficient computing resources are available and the problem size is sufficiently large, XBraid can offer a scalable speedup and increased parallelism.

4. Conclusions and future work. This work investigates the implementation of a parallel-in-time moving mesh algorithm applied to a 1-dimensional diffusion PDE. Initial results are promising, successfully coupling the physical PDE and MMPDE and wrapping the system with the XBraid parallel-in-time library. Near perfect strong and weak scaling are achieved and stability issues are resolved using FMG cycles in time. However, the model problem used thus far is only one dimensional in space and relatively smooth. Future work will involve implementing a moving mesh algorithm in higher dimensions, and coupling this with more complicated physical PDEs. It will be important to implement a spatially parallel moving mesh and consider the speedup of XBraid over sequential time stepping. Due to the difficult communication algorithms required for a spatially parallel moving mesh, this speedup may occur at a reasonably small number of processors. Tangential work will involve applying XBraid to alternative adaptive mesh methods, including quasi-Lagrange moving mesh, adaptive time steps, and full space-time adaptive mesh refinement, in order to show XBraid's applicability to all standard types of adaptive mesh algorithms.

REFERENCES

[1] *MFEM: Modular finite element methods*. `mfem.org`.

[2] *XBraid: Parallel multigrid in time*. `http://llnl.gov/casc/xbraid`.

[3] W.L. Briggs, S.F. McCormick, and V.E. Henson, *A multigrid tutorial*, Siam, 2000.

[4] M. Emmett and M. Minion, *Toward an efficient parallel in time method for partial differential equations*, Communications in Applied Mathematics and Computational Science, 7 (2012), pp. 105–132.

[5] R.D. Falgout, S. Friedhoff, Tz. V. Kolev, and S.P. MacLachlan, *Parallel time integration with multigrid*, SIAM Journal on ..., (2014).

[6] R.D. Falgout, S. Friedhoff, Tz. V. Kolev, S.P. MacLachlan, J.B. Schroder, and S. Vandewalle, *Multigrid methods with space-time concurrency*, SIAM Journal on Scientific Computing, (2015).

[7] R.D. Falgout, A. Katz, Tz. V. Kolev, J.B. Schroder, A.M. Wissink, and U.M. Yang, *Parallel time integration with multigrid reduction for a compressible fluid dynamics application*, (2014).

[8] R.D. Falgout and P.S. Vassilevski, *On Generalizing the Algebraic Multigrid Framework*, SIAM Journal on Numerical Analysis, 42 (2004).

[9] R.D. Falgout and U.M. Yang, *hypre: A library of high performance preconditioners*, Computational Science—ICCS 2002, (2002).

[10] Martin J Gander, *50 years of time parallel time integration*, in Householder Symposium XIX June 8-13, Spa Belgium, p. 81.

[11] J. Lions, Y. Maday, and G. Turinici, *A "parareal" in time discretization of pde's*, Comptes Rendus de l'Academie des Sciences Series I Mathematics, 332 (2001), pp. 661–668.

[12] S. Vandewalle and E. Van de Velde, *Space-time concurrent multigrid waveform relaxation*, Annals of Numer. Math, 1 (1994), pp. 347–363.

[13] H. Weizhang and R.D. Russell, *Adaptive Moving Mesh Methods*, vol. 174 of Applied Mathematical Sciences, Springer Science & Business Media, New York, NY, Oct. 2010.