A PARALLEL-IN-TIME ALGORITHM FOR VARIABLE STEP MULTISTEP METHODS

ROBERT D. FALGOUT*, MATTHIEU LECOUVEZ†, AND CAROL S. WOODWARD*

Abstract. This paper presents a multigrid reduction in time (MGRIT) algorithm using BDF methods for achieving time parallelism for nonlinear, differential-algebraic equations of index 1. This MGRIT approach transforms the linear multistep methods into single step methods applied to groups of time steps. Stability considerations are addressed through lowering of the order on coarse grids. The methods are presented for fixed and variable time step formulations. Numerical results show moderate speedups for the resulting methods on both heat equation and IEEE power grid test problems. Performance of MGRIT with BDF is compared to performance of MGRIT with Runge-Kutta methods of the same orders for both fixed and variable step methods[1].

Key words. parallel in time, differential-algebraic systems, power grid simulations, multigrid reduction in time

AMS subject classifications. 65L06, 65L80, 65M55, 65Y05

1. Introduction.

Because clock speeds are no longer increasing, the sequential time marching approach used in nearly all science simulation codes is becoming a bottleneck. Parallel time integration is a way of creating concurrency in a simulation that can be exploited to remove this bottleneck and reduce computing time, sometimes dramatically [10, 8]. However, parallel-in-time algorithms represent a radical departure from established practice, and although significant progress has been made on a variety of research fronts, there are still many outstanding hurdles to overcome to make this approach a practical alternative.

One major issue is the question of convergence. With time stepping, the term convergence usually relates to the accuracy of the discrete equations. Parallel-in-time methods are nonlinear iterative methods over the time domain, so they have the added difficulty that the iteration itself must converge efficiently to the discrete space-time solution. When developing such methods, the character of the underlying system greatly influences algorithmic choices. In addition, it is important to accommodate essential features of modern simulation codes, such as adaptivity and error estimation, while recognizing that these codes are often extremely complex and represent many person-years of development effort.

Most physics-based applications involve the solution of time-dependent (partial) differential equations. The most common form of differential equations are *ordinary differential equations* (ODEs) in their explicit form

$$\dot{u} = f(t, u), \quad u(t_0) = u_0, \tag{1}$$

where u is the (vector) time dependent unknown, and f may be nonlinear. A more general variant of these ODEs is the implicit form

$$\mathcal{F}(t, u, \dot{u}) = 0, \quad u(t_0) = u_0, \tag{2}$$

*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551 (falgout2@llnl.gov, woodward6@llnl.gov). 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-JRNL-739759.

†Centre d'études scientifiques et techniques d'Aquitaine, CEA/Cesta, 15 Avenue des Sablières, 33114 Le Barp, France (matthieu.lecouvez@cea.fr)

[1]This paper shares some introductory material with [20] on the description of the MGRIT algorithm.

where the Jacobian of $\mathcal{F}$ with respect to $\dot{u}$, $\frac{\partial \mathcal{F}}{\partial \dot{u}}$, is assumed to be nonsingular for all t. With this assumption, it is easy to see that these two forms are equivalent. When the Jacobian may be singular, we have *differential-algebraic equations* (DAEs), and many theoretical results for ODEs are not valid anymore (see [2]). DAEs arise in various fields of science, from astrophysics to power grid simulations and mechanics. In most of these applications, the function, $\mathcal{F}$, is nonlinear.

Much work has been done on developing efficient time stepping methods for DAEs [2, 5, 12]. This work has resulted in very efficient variable step and (in the case of linear multistep methods) variable order adaptive methods and software for these systems [13]. In particular, the linear multistep methods have been shown to be very efficient in a number of application areas, especially circuit [15] and power grid simulation [23, 3]. A challenge with parallel-in-time methods has been developing strategies that accommodate these highly efficient adaptive methods.

In this paper, we focus on the parallel-in-time solution of DAEs discretized with multistep backward difference formula (BDF) time integration on variably-spaced temporal grids. Research in this area has been limited to date. The approach we consider here is called multigrid reduction in time (MGRIT) [7], which has the advantage of being relatively non-intrusive on existing application codes and allows for significant reuse of existing simulation software and technology. The MGRIT approach was first applied to DAEs in [20] where speedups were demonstrated for a simple single-machine, infinite-bus power grid problem discretized with a high-order Runge-Kutta method. In the current paper, the focus is instead on BDF methods and variable step time grids, and more complex power grid problems are considered[1]. Since the DAEs considered here are power system problems, it is insightful to compare and contrast the MGRIT approach to some of the previous work done in this area. In the interest of space, we mention only two methods here, one early [18, 17] and the other more recent [11].

The method introduced in [18] was well ahead of its time and was one of the first to demonstrate potential for speeding up power grid simulations with a parallel-in-time approach. Like MGRIT, the algorithm uses a sequence of coarse grids and coarse-grid problems. It does nested iteration in an effort to provide better and better initial guesses on each grid level, starting on the coarsest grid and moving up the hierarchy to the finest grid. A variety of relaxation/Newton methods were studied as the iterative solver on each grid level [17]. MGRIT differs primarily through the FCF-relaxation scheme (see Section 2) and the different grid cycling strategies used. In general, nested iteration with relaxation will not produce an optimal $O(N)$ algorithm, where N is the number of discrete points on the time grid, and more complex cycling strategies such as F-cycles (Figure 2) are needed.

The method in [11] is based on the popular parareal algorithm [21]. Like MGRIT, parareal is also non-intrusive and easy to integrate with existing serial time-stepping codes, and, as mentioned above, it is equivalent to MGRIT when using only two time grids and just F-relaxation. The work in [11] demonstrates the potential for speedup in power grid simulations by running serial code and assuming that parallel communication costs are negligible. In general, scalability of multigrid algorithms in parallel requires more than two grid levels. A more extensive discussion of this along with detailed parallel performance analyses can be found in [7, 8].

In Section 2, we introduce the MGRIT algorithm, and in Section 3, we introduce time integration methods for DAEs. In Section 4, we describe our approach for solving BDF discretizations with MGRIT and discuss the stability of the method. In Section 5, we provide numerical results for the 2D heat equation and several power

grid problems on uniform and variable-step temporal grids. Section 6 gives concluding remarks and discusses future work. Lastly, Appendix A provides additional details on the stability analysis in Section 4.

2. Multigrid Reduction in Time algorithm. Research on parallel-in-time methods started with the work of Nievergelt in 1964 [22]. Since then, a variety of approaches have been developed. However, relatively little work has been done in this area overall, especially considering that more than 50 years have passed since it was first explored. For a recent review of the literature, see [10]. The method considered here is called multigrid reduction in time (MGRIT) [7, 8, 9, 6, 20]. It is based on multigrid reduction techniques [24, 25] and is relatively non-intrusive on existing codes.

Let $u(t)$ be the solution to some time-dependent problem (for example, a DAE) on time interval $[0, T]$. Let $0 = t_0 < t_1 < \cdots < t_N = T$ be a discrete temporal mesh on that interval, and let u_i be an approximation to $u(t_i)$. A one-step time discretization is then given by

$$u_0 = g_0, \quad u_i = \Phi_i(u_{i-1}) + g_i, \quad i = 1, 2, ..., N. \tag{3}$$

A traditional time stepping method solves for u_1 through u_N in sequence. To motivate the MGRIT approach, first consider the simple linear case where each function Φ_i is a matrix. Then, time stepping is equivalent to a forward solve of the block linear system $A\mathbf{u} = \mathbf{g}$ given by

$$\begin{pmatrix} I & & & \\ -\Phi_1 & I & & \\ & \ddots & \ddots & \\ & & -\Phi_N & I \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_N \end{pmatrix}. \tag{4}$$

The idea is to replace the $O(N)$ sequential time stepping method with an $O(N)$ *multigrid* [26] iterative solver that is highly parallel. One well-known direct method for solving tridiagonal systems is *cyclic reduction*. The MGRIT multigrid method is a kind of approximate block cyclic reduction algorithm that utilizes a sequence of coarser temporal systems to accelerate the solution of the fine grid problem in (4). Although the above motivation assumes a linear system, the algorithm applies to the full nonlinear setting. It is also important to note that we solve the same discrete space-time system as in (3); that is, the MGRIT approach is discretization agnostic and converges to the solution produced by sequential time stepping on the finest grid.

The coarse grids in MGRIT are formed from the original fine grid by successively coarsening with factor $m > 1$. The coarsening of a grid induces a decomposition of that grid into two sets called C-points (points that align with the coarse grid) and F-points (everything else). Figure 1 provides an illustration in the case of the finest grid level. With this decomposition in hand, we can describe *relaxation* and *coarse-grid correction*, the two main components of multigrid.

In the case of MGRIT, relaxation alternates between so-called F-relaxation and C-relaxation. F-relaxation updates the F-point values, u_i, on interval (T_j, T_{j+1}) by propagating the C-point value, u_{mj}, (i.e., u_i where $i = mj$) across the interval using the time propagators Φ_i in sequence. Although this process is sequential, the F-intervals are independent from each other and can be computed in parallel. Similarly, C-relaxation updates the C-point value, u_{mj}, based on the F-point value, u_{mj-1}, and these updates can also be computed in parallel. We define FCF-relaxation as

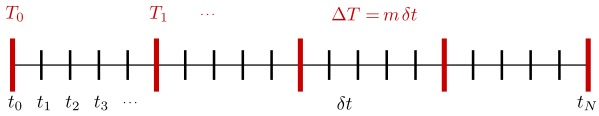


FIG. 1. *Fine grid (t_i) and coarse grid (T_j) for coarsening factor $m = 5$. The coarse grid induces a decomposition of the fine grid into C-points (red) and F-points (black).*

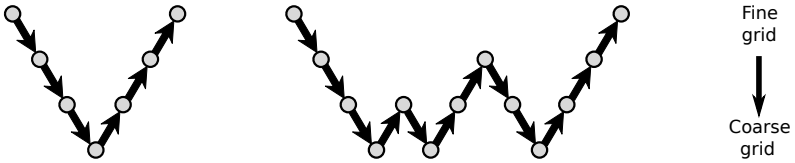


FIG. 2. *Multigrid V-cycle (left) and F-cycle (right). Cycles represent the order in which different time grids are visited in the recursive MGRIT algorithm. Going down to a coarser grid corresponds to step 2 of the MGRIT algorithm, while going up corresponds to step 5.*

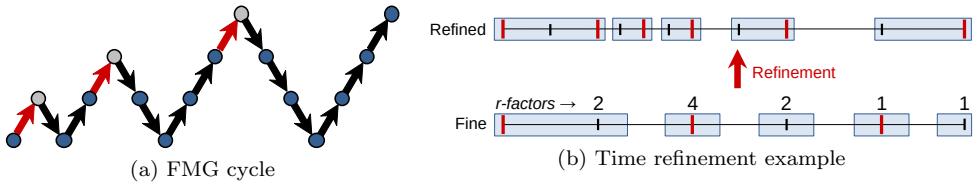
the composition of successive F-, C-, and F-relaxations. Simple injection is used to transfer values between grids (Steps 2 and 5 below).

The two-grid MGRIT algorithm is then as follows:

1) Apply FCF-relaxation to $A(\mathbf{u}) = \mathbf{g}$.
2) Restrict the fine grid approximation and residual to the coarse grid:
   $u_{\Delta,j} \leftarrow u_{mj}, \quad r_{\Delta,j} \leftarrow g_{mj} - A(\mathbf{u})_{mj}$.
3) Solve the coarse system $A_\Delta(\mathbf{v}_\Delta) = A_\Delta(\mathbf{u}_\Delta) + \mathbf{r}_\Delta$.
4) Compute coarse error approximation: $\mathbf{e}_\Delta = \mathbf{v}_\Delta - \mathbf{u}_\Delta$.
5) Correct $\mathbf{u}$ at C-points: $u_{mj} = u_{mj} + e_{\Delta,j}$.
6) Apply F-relaxation.

The multilevel algorithm uses the two-grid method in a recursive fashion to solve the system in Step 3. A variety of standard multigrid cycling strategies may be applied, including V-cycles and F-cycles (see Figure 2). For a fairly comprehensive reference on multigrid methods and techniques, see [26]. One important aspect of MGRIT is that the user only needs to define Φ_i, which corresponds to the original time stepping method. Hence, most of a users' original time integration code can be used as is. This property makes the MGRIT method relatively non-intrusive.

2.1. XBraid. Through the open-source library XBraid [1], users gain access to MGRIT parallel time integration capabilities by writing only a small amount of new code that wraps their existing time-stepping method, error estimation techniques, etc. The variable step methods explored in this paper utilize XBraid's time adaptivity feature, which uses a Full MultiGrid (FMG) cycle as depicted in Figure 3. As opposed to starting on the fine grid as described above, in FMG the user specifies an initial coarse temporal grid (often this is just a uniform grid). Then as the system is solved, the user tells XBraid if a time interval should be refined. This decision about refinement uses the same error estimation techniques employed in the original time-stepping code. From this user-provided information, XBraid then constructs a refined grid and redistributes the time intervals across the processors (Figure 3b) so that each processor owns approximately the same number of time points. This process continues until all of the time intervals are adequately refined. Once the the fine time grid is developed, XBraid will conduct any iterations needed to ensure the 2-norm of the relative residual of (4) is below a given tolerance. During this FMG cycle,



(a) FMG cycle

(b) Time refinement example

FIG. 3. *(a) Temporal adaptivity is achieved in XBraid via an FMG-cycle. The fine grid is solved using either V-cycles or F-cycles, then it is refined (red arrows) to create a new fine grid with better approximation properties, and the process continues until the final grid is considered satisfactory. Here, one V-cycle is shown for each fine grid problem solution. (b) User-provided refinement factors are used to construct refined grids which are automatically redistributed by XBraid. A 5 processor example is shown here. Refinement factors indicate needed refinement of the interval to the left of the mark.*

spatial adaptivity is also possible. XBraid provides all of the software infrastructure for passing data between grids and applying distributed memory parallelism through MPI in the multigrid algorithm.

3. Introduction to methods for DAEs. In the numerical solution of systems of DAEs, the solution is approximated on a discrete time grid $t_0 < t_1 < \cdots < t_N = T$, and for all $n \leq N$, we let u_n be the approximation of $u(t_n)$, the solution at time t_n. Traditionally, and starting from the initial condition u_0, the values u_n are computed one-by-one using previous steps. In the following sections we describe two main classes of time integrators.

3.1. Runge-Kutta solvers. Runge-Kutta integrators are one-step, multi-stage methods [19]. An s-stage Runge-Kutta method of order q is given by its so-called Butcher tableau,

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \dots & a_{ss} \\ \hline (q) & b_1 & \dots & b_s \end{array} . \tag{5}$$

The computation of u_n from the previous step u_{n-1} is given by

$$u_n = u_{n-1} + h_n \sum_{i=1}^{s} b_i K_i, \tag{6}$$

where $h_n = t_n - t_{n-1}$, and the K_i, $i = 1, ..., s$ are solutions of the following nonlinear problem

$$\mathcal{F}\left(t_{n-1} + c_i h_n,\ u_{n-1} + h_n \sum_{j=1}^{s} a_{ij} K_j,\ K_i \right) = 0, \quad \forall\ 1 \leq i \leq s. \tag{7}$$

All the properties (especially the order of accuracy) of the Runge-Kutta method are determined by its Butcher tableau. From (7), we can see that one step of an s-stage Runge-Kutta method requires the solution of an $sn_u \times sn_u$ nonlinear system (where n_u denotes the size of the unknown vector u). For large problems or high order methods, the cost of the nonlinear solve may become prohibitive. Specific Runge-Kutta methods have been developed to overcome this issue. Indeed, if the coefficients

(a_{ij}) of the Butcher tableau form a lower triangular matrix, then solving (7) would require s nonlinear solves of $n_u \times n_u$ problems. Furthermore, if the coefficients on the diagonal are all equal, $a_{11} = \cdots = a_{ss} = a$, then a modified quasi-Newton solver can be efficiently used to solve the s nonlinear problems to further reduce the computational cost (the same Jacobian can be used for all the s nonlinear solves).

Adaptive step time integration methods are critical to efficient solution processes in many application areas. To allow variable time step sizes, one needs, at any given point in time, to be able to predict the required size of the following time step. This prediction requires an error estimate on the solution computed at the current step. Embedded Runge-Kutta methods are well suited for that purpose. These schemes consist of two Runge-Kutta methods of different orders, q and $q-1$, sharing the same coefficients (a_{ij}) and (c_i). We can combine their Butcher tableaux as follows

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \dots & a_{ss} \\ \hline (q) & b_1 & \dots & b_s \\ (q-1) & \widetilde{b}_1 & \dots & \widetilde{b}_s \end{array} . \tag{8}$$

Because the two schemes share coefficients, the solutions of the systems in (7) are computed only once, then two solutions of different orders of accuracy are computed

$$\begin{aligned} u_n &= u_{n-1} + h_n \sum_{i=1}^{s} b_i K_i \quad \text{(order } q\text{)} \\ \widetilde{u}_n &= u_{n-1} + h_n \sum_{i=1}^{s} \widetilde{b}_i K_i \quad \text{(order } q\text{-1)}, \end{aligned} \tag{9}$$

and the next step size, h_{n+1}, is determined to satisfy

$$\left(\frac{h_{n+1}}{h_n}\right)^q \|e_n\| \approx \varepsilon, \tag{10}$$

where $e_n = u_n - \widetilde{u}_n$ is an error estimation, and ε is a given tolerance. Using this technique, one can see that the step sizes are determined through the estimation of a local error of order $q-1$, even though the solutions are of order q. The norm used is usually either the L2-norm (if all components of the solution vector are commensurate) or a weighted root-mean-square (WRMS) norm. In the latter case (10) becomes

$$\left(\frac{h_{n+1}}{h_n}\right)^q \|e_n\|_{WRMS} \approx 1, \quad \|e_n\|_{WRMS} = \sqrt{\frac{1}{n_u}\sum_{i=1}^{n_u} \frac{e_{n,i}}{\varepsilon_{rel}|u_{n,i}| + \varepsilon_{abs,i}}} \tag{11}$$

ε_{rel} is a relative tolerance, and ε_{abs} is a vector of absolute tolerances. Note that this norm is based on the current value of the solution u_n. This is the norm that we used in our numerical simulations presented in section 5.

3.2. Backward Difference Formula (BDF) integrators. Unlike the one-step, multistage Runge-Kutta methods, backward difference formula (BDF) methods are multistep methods. These methods require several previous steps to compute the solution at a new point in time, by means of polynomial interpolation. A BDF

method is given by a finite difference approximation of $\dot{u}_n$

$$\beta_{0,n}\, h_n \dot{u}_n = \sum_{i=0}^{q} \alpha_{i,n} u_{n-i}, \tag{12}$$

and this approximation is used to obtain the nonlinear problem

$$g(u_n) = \mathcal{F}\left(t_n,\, u_n,\, \frac{1}{\beta_{0,n} h_n} \sum_{i=0}^{q} \alpha_{i,n} u_{n-i}\right) = 0 \tag{13}$$

to solve for u_n at each time step. In the case of uniform time steps, the coefficients $\beta_{0,n}$ and $\alpha_{i,n}$ do not depend on n anymore and are given in Table 1 for $q \leq 6$. When time steps are not uniform, the coefficients need to be adapted. To that purpose, two main techniques are available with different stability properties and implementation advantages. It is worth noting that both methods reduce to the original BDF method when uniform time steps are used.

3.2.1. Fully variable coefficients. This is the most stable strategy. Here, coefficients are derived directly from finite differences based on unequally spaced time points,

$$\dot{u}_n = [u_n, u_{n-1}] + h_n\, [u_n, u_{n-1}, u_{n-2}] + h_n h_{n-1}\, [u_n, u_{n-1}, u_{n-2}, u_{n-3}] + \dots \tag{14}$$

where the divided differences are defined by the recursion formula

$$\begin{array}{rcl} [u_n] & = & u_n \\ [u_n, u_{n-1}, \cdots, u_{n-i}] & = & \frac{[u_n, u_{n-1}, \cdots, u_{n-i+1}] - [u_{n-1}, \cdots, u_{n-i}]}{t_n - t_{n-i}} \end{array} \tag{15}$$

Table 2 presents the resulting coefficients for the BDF-2 method. We can see that all the coefficients (including β_0) vary with step size. This variance has an important implication on the implementation and cost of the method. Indeed, when solving the (usually) nonlinear (13) with a Newton-like method, the Jacobian J_g arises, given by (since $\alpha_{0,n} = 1$)

$$J_g = \frac{\partial \mathcal{F}}{\partial u} + \frac{1}{\beta_{0,n}} \frac{\partial \mathcal{F}}{\partial \dot{u}}. \tag{16}$$

Since computing the Jacobian is expensive, most efficient time integrators re-use the same Jacobian over several time steps to solve (13). Having the coefficient β_0 vary at each time step makes this technique more difficult to implement, and the Jacobian would require many more updates.

3.2.2. Fixed leading coefficient. The fixed leading coefficient method has been designed to overcome the issue of the Jacobian that appears with the fully variable method [14]. Here, the leading coefficient, β_0, is fixed to the one used for fixed time steps. To compensate for the fixed coefficient and to keep the correct order of accuracy, one extra term is needed in (12). We add this term, $\beta_{1,n} h_n \dot{u}_{n-1}$, to the left of the equation to get

$$\beta_{0,n}\, h_n \dot{u}_n + \beta_{1,n}\, h_n \dot{u}_{n-1} = \sum_{i=0}^{q} \alpha_{i,n} u_{n-i}. \tag{17}$$

TABLE 1
Coefficients of the BDF methods for uniform time steps.

Order q	β_0	α_0	α_1	α_2	α_3	α_4	α_5	α_6
1	1	1	-1					
2	$\frac{2}{3}$	1	$-\frac{4}{3}$	$\frac{1}{3}$				
3	$\frac{6}{11}$	1	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$			
4	$\frac{12}{25}$	1	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$		
5	$\frac{60}{137}$	1	$-\frac{300}{137}$	$\frac{300}{137}$	$-\frac{200}{137}$	$\frac{75}{137}$	$-\frac{12}{137}$	
6	$\frac{60}{147}$	1	$-\frac{360}{147}$	$\frac{450}{147}$	$-\frac{400}{147}$	$\frac{225}{147}$	$-\frac{72}{147}$	$\frac{10}{147}$

With this method, one extra vector $\dot{u}_{n-1}$ needs to be saved to compute a new step. This method is based on polynomial interpolation. First a predictor polynomial w^p is built that fits the previously computed values,

$$\begin{cases} w^p(t_{n-i}) = u_{n-i}, & i = 1, ..., q \\ \dot{w}^p(t_{n-1}) = \dot{u}_{n-1}. \end{cases} \tag{18}$$

This polynomial w^p gives a first estimate $u_n^{(0)} = w^p(t_n)$ of u_n, and a first estimate $\dot{u}_n^{(0)} = \dot{w}^p(t_n)$ of $\dot{u}_n$. Then, a corrector polynomial w^c, which coincides with w^p on uniformly spaced time values ($t = t_n - ih_n, i = 1, \ldots, q$), is defined so that with $w^c(t_n) = u_n$

$$\begin{cases} w^c(t) = w^p(t) + c(t)\left(u_n - u_n^{(0)}\right) \\ c(t_n - h_n) = \cdots = c(t_n - qh_n) = 0, \quad c(t_n) = 1. \end{cases} \tag{19}$$

Note that from the definition of the polynomial, c, it follows that $\beta_0 h_n \dot{c}(t_n) = 1$. Finally, the BDF formula simply reads $\dot{u}_n = \dot{w}^c(t_n)$, or

$$\beta_0 h_n \dot{u}_n = \beta_0 h_n \dot{u}_n^{(0)} + h_n \left(u_n - u_n^{(0)}\right). \tag{20}$$

Table 2 shows the resulting BDF coefficients for the second order BDF method, using uniform steps or variable steps for both approaches. The main advantages of the fixed leading coefficient approach are first the possibility to reuse the Jacobian (16) during several time steps since $\beta_{0,n}$ is now independent of n. The second advantage comes from the predictor polynomial w^p. This polynomial gives an order q estimate $u_n^{(0)}$ of the (order q) solution u_n. Hence, the error estimate $e_n = u_n - u_n^{(0)}$ is also of order q and can be used together with (10) to control the size of the steps to achieve a required accuracy, for no extra computational cost.

3.3. Solving nonlinear systems. As indicated in the last two sections, solving nonlinear DAEs (or ODEs with implicit schemes) requires the solution of at least one nonlinear problem per time step. To solve these systems, we use a modified Newton method (see Algorithm 1). This method is a simple Newton algorithm in which the Jacobian matrix is evaluated only once at the beginning of the time step. This modification greatly reduces the cost of the nonlinear solve since the Jacobian is expensive to evaluate. Moreover, this algorithm allows us to factorize the Jacobian in the initialization step, reducing the overall cost of the linear solves. The Newton

TABLE 2
Coefficients of the BDF-2 method for uniform steps (US), fully variable coefficient (FVC), and fixed leading coefficient (FLC) methods. Here $\rho_n = h_n/h_{n-1}$ is the ratio between the current and previous step sizes.

Method	β_0	β_1	α_0	α_1	α_2
US ($\rho_n = 1$)	$\frac{2}{3}$	0	1	$-\frac{4}{3}$	$\frac{1}{3}$
FVC	$\frac{\rho_n + 1}{2\rho_n + 1}$	0	1	$-\frac{(\rho_n+1)^2}{2\rho_n+1}$	$\frac{\rho_n^2}{2\rho_n+1}$
FLC	$\frac{2}{3}$	$\frac{1-\rho_n}{3}$	1	$-1-\frac{\rho_n^2}{3}$	$\frac{\rho_n^2}{3}$

algorithm we employ is given in Algorithm 1. In the case of Singly Diagonally Implicit Runge-Kutta methods (with a lower triangular Butcher tableau and the same coefficient $a_{11} = \cdots = a_{ss} = a$ on the whole diagonal), the same Jacobian matrix, J, is reused for all of the Runge-Kutta stages.

Algorithm 1 Modified Newton algorithm

Require: Function g, Jacobian function G, initial guess x_0, tolerance ε.
Ensure: $\|g(x^*)\| < \varepsilon$

```
x_k ← x_0
r ← g(x_k)
J ← G(x_k)
while ‖r‖ ≥ ε do
    x_k ← x_k − J^{-1} r
    r ← g(x_k)
end while
x* ← x_k
```

4. BDF trick. The MGRIT algorithm described previously is developed for general one-step methods, such as Runge-Kutta methods. The algorithm requires the solution u_n at time t_n be computed from the solution u_{n-1} at time t_{n-1} through a step function Φ_n

$$u_n = \Phi_n(u_{n-1}). \tag{21}$$

Φ_n can be multi-stage (like high order Runge-Kutta methods) and involve nonlinear solves. This framework is, however, restricted to one-step methods, and multistep methods, such as BDF, cannot be used. Nonetheless, a redefinition of the problem can be used to write multistep methods as a general one-step method. This trick first appeared in [8] but only BDF-2 methods were considered, and stability issues and variable-step grids were not investigated.

Let Ψ be a general q-step integrator, meaning that the solution u_n at time t_n is computed from the previous steps u_{n-1}, ..., u_{n-q} through the q-variable function Ψ_n,

$$u_n = \Psi_n\left(u_{n-1}, u_{n-2}, \ldots, u_{n-q}\right). \tag{22}$$

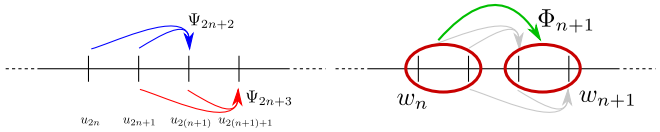


FIG. 4. *Representation of how a two-step method (left) can be redefined into a one-step method by gathering unknowns (right).*

Writing q steps of this process leads to

$$\begin{aligned} u_{n+1} &= \Psi_{n+1}\left(u_n, u_{n-1}, \ldots, u_{n-q+1}\right), \\ u_{n+2} &= \Psi_{n+2}\left(u_{n+1}, u_n, \ldots, u_{n-q+2}\right), \\ &\cdots \\ u_{n+q-1} &= \Psi_{n+q-1}\left(u_{n+q-2}, u_{n+q-3}, \ldots, u_{n-1}\right). \end{aligned} \tag{23}$$

We now redefine the problem by gathering q unknowns together. Let w_n be the vector of *gathered unknowns*

$$w_n = \begin{bmatrix} u_{qn} \\ u_{qn+1} \\ \cdots \\ u_{qn+q-1} \end{bmatrix}. \tag{24}$$

It is easy to see that with this notation, all values of w_{n+1} can be computed from values of w_n by doing a series of sequential steps similar to (22)-(23). Let Φ_{n+1} denote this process. We then have

$$w_{n+1} = \Phi_{n+1}(w_n), \tag{25}$$

and our redefined integrator fits the framework of the MGRIT algorithm. Figure 4 represents this gathering process.

4.1. Stability concerns. Using the redefined problem in a multigrid context, however, has consequences on the MGRIT algorithm and more specifically on the coarse time grids. Since the gathering technique described previously is done on the finest level, the coarsening process involves only the new set of larger unknowns. The resulting coarse time grid is not uniform, and the discrepancy between the step sizes may become extremely large. An illustration of the issue is represented in Figure 5 for the case of a 3-step method and a coarsening factor of 2. A simple recurrence shows that the ratio, ρ, between the largest step and the smallest step on level, l, using a coarsening factor, c, and a q-step method is given by

$$\rho_l(q, c) = q\left(c^l - 1\right) + 1 \tag{26}$$

and may be arbitrarily large. This discrepancy has an important effect on the stability of the time integrator on the coarse levels. The stability regions for BDF-2 and BDF-3 with fully variable coefficients and fixed leading coefficients are presented in Figures 6 and 7, respectively. See Appendix A for more details on the computation of the stability region.

If the BDF-2 method with fully variable coefficients (Figure 6.a-d) shows a surprisingly good stability on all coarse grids, this is definitely not the case for the other

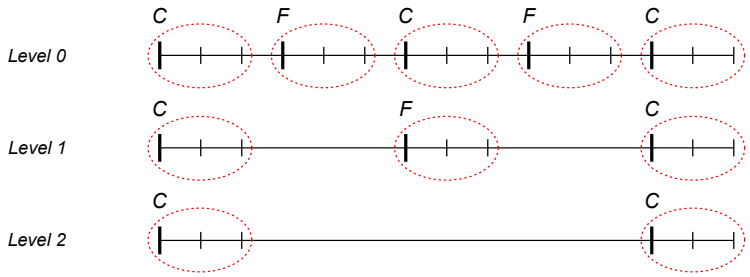
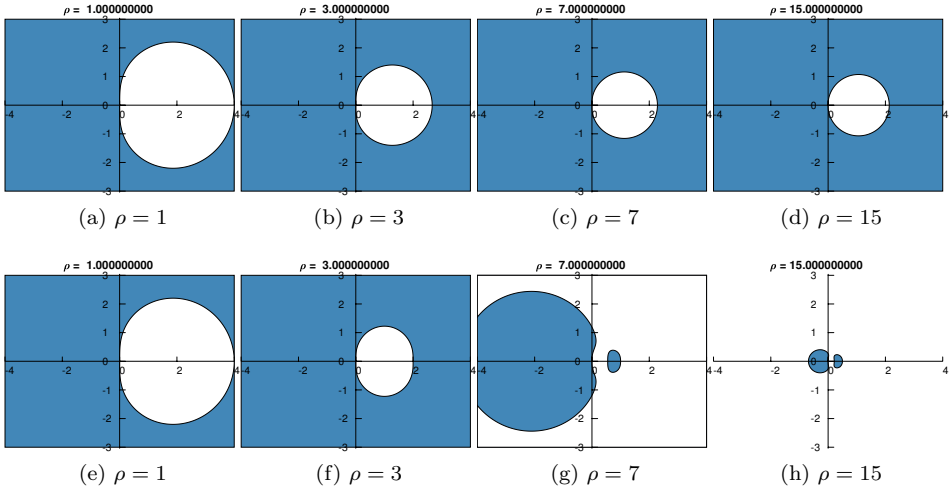


FIG. 5. *Resulting coarse grids when using a 3-step method and a coarsening factor of 2. Each tick represents an actual point in time. The dotted circles show how points in time are gathered, and the thick ticks are the time points as seen by the MGRIT algorithm (labelled C or F for coarse points and fine points, respectively). The coarsest grid gives the largest discrepancy in time step sizes.*



(a) $\rho = 1$ (b) $\rho = 3$ (c) $\rho = 7$ (d) $\rho = 15$

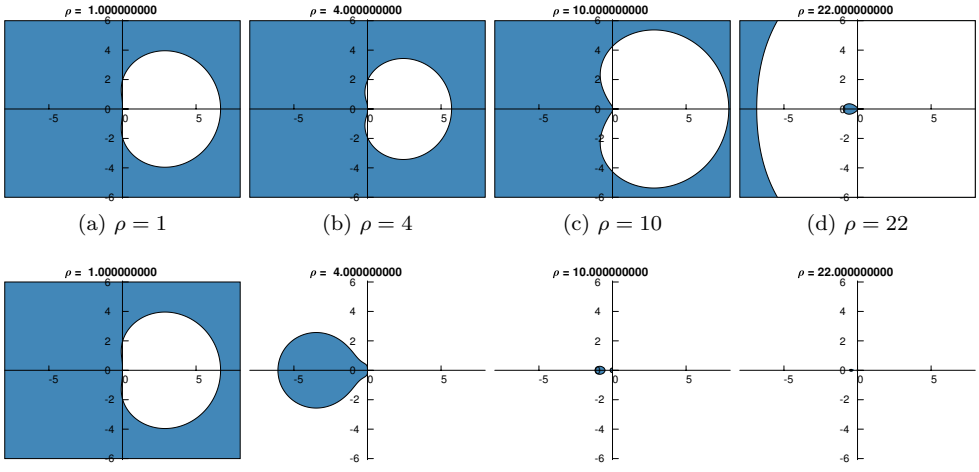
(e) $\rho = 1$ (f) $\rho = 3$ (g) $\rho = 7$ (h) $\rho = 15$

FIG. 6. *Stability region (in blue) of the BDF 2 method, using fully variable coefficient (top) or fixed leading coefficient (bottom) methods. From left to right, we represent the stability region for the time grid on levels 0 to 3, with a coarsening factor of 2. Axes are real and imaginary part of $h\lambda$, h being the size of the step (on the finest grid) and λ is the complex parameter of the scalar test equation $\dot{u} = \lambda u$.*

methods. In particular, using the fixed leading coefficient strategy results in great instability, with a stability region shrinking at least as fast as ρ^{-1}. Such a behavior is obviously a great concern for the MGRIT algorithm. It is indeed necessary to keep the integration method stable on coarse grids to obtain a good coarse grid correction. To that purpose, we propose the two following strategies to be applied within the embedded cycle method within FMG.

- The first few "coarse" grids are not coarsened in time, but a lower order BDF method is used, until reaching a BDF method of order 1, then the time grid is coarsened the usual way.
- The grid hierarchy is built the usual way, coarsening in time from level 0, but the order of the BDF method is decreased simultaneously.

The first strategy is the most stable, as the method simply reduces to a backward Euler method on uniform steps before coarsening the time grid. It is also the most



(a) $\rho = 1$ (b) $\rho = 4$ (c) $\rho = 10$ (d) $\rho = 22$

(e) $\rho = 1$ (f) $\rho = 4$ (g) $\rho = 10$ (h) $\rho = 22$

FIG. 7. *Stability region (in blue) of the BDF 3 method, using fully variable coefficient (top) or fixed leading coefficient (bottom) methods. From left to right, we represent the stability region for the time grid on levels 0 to 3, with a coarsening factor of 2. Axes are real and imaginary part of $h\lambda$, h being the size of the step (on the finest grid) and λ is the complex parameter of the scalar test equation $\dot{u} = \lambda u$.*

computationally expensive as the integration is performed several times on the finest grid. Numerical experiments show that the second strategy is stable enough in almost all cases, and, for that reason, this strategy will be used in the following sections.

5. Numerical results. We present numerical results for simple diffusion problems and an application to power grid systems. We are interested in comparing our implementation of the MGRIT algorithm using the LLNL XBraid implementation to an optimal (in terms of number of function evaluations) but sequential time stepping implementation. To that purpose, several quantities of interest are presented

- The time-to-solution is the time needed to obtain the solution over the entire time interval considered.
- The crossover point is the number of cores required by our algorithm to break even with traditional sequential time stepping. With a lower number of cores, MGRIT takes more time than sequential time stepping to get the solution.
- Best speedup: this is the ratio between the time-to-solution for the sequential method and the best time to solution for MGRIT. It describes how much faster MGRIT is compared to sequential time stepping.

From the description of the MGRIT algorithm in Section 2, it is easy to see that the solution only needs to be saved at C-points, even for multistep integrators. Hence, an efficient implementation will not save F-point solutions. This is one of the storage options available in XBraid (see [9] for more discussion on storage costs). However, to use adaptive time refinement in XBraid, some extra information may be needed when computing the gathered unknowns. For instance, one cannot assume uniform time steps, and it becomes necessary to save the inner time step values at all points (at C-points as well as F-points, on all grids). This extra storage remains negligible for large problems since only q extra time step values per gathered time point are needed, where q is the order of the BDF method on the finest grid. Parallel com-

munications are needed mainly during F- and C-relaxations and are performed with MPI (Message Passing Interface). XBraid is implemented to reduce the cost of communication by overlapping these communications with computations. For example, during an F-relaxation, each processor starts with its rightmost coarse interval (see the grid layout in Figure 1) then sends the result to its right neighbor processor. While this communication is occurring, each processor works on its interior coarse intervals. Once each processor receives the communication from its left neighbor processor, the leftmost coarse interval can be completed. A similar strategy is used for C-relaxation and other components of the code.

In all of our numerical examples, we use F-cycles (see Figure 2). In the adaptive case, we use one F-cycle to solve each intermediate FMG grid level, and otherwise the number of cycles is determined by a relative residual stopping tolerance of 10^{-8}. To initialize the algorithm, we start on a uniform initial grid, taken to be decidedly coarse in the adaptive case. Then, each point in time is initialized to the value of the initial condition: $u_n^0 = u_0$. Additional parameters, such as the coarsening factor m, can be problem-dependent. Below we use the coarsening factor that gave the best overall results for each problem. For all variable time step cases below we use 10^{-8} for the time integrator tolerance in (10).

All runs below were conducted on the "Cab" Linux cluster at Lawrence Livermore National Laboratory. This machine is an Intel Xeon-based system with over 1,200 nodes, each with two 8-core CPUs in a shared memory configuration. Speedup results are presented for a number of cores ranging from 1 to 2,048.

5.1. Heat equation. This first test case corresponds to solving the heat equation on a square domain, $\Omega = [0, \pi]^2$, with a uniform spatial grid of $n_x \times n_y$ using finite differences in space. Dirichlet boundary conditions are imposed on all sides of the square. The continuous problem is given by

$$\left\{ \begin{array}{rcll} \dfrac{\partial u}{\partial t} - \Delta u & = & f(\mathbf{x}, t) & \text{in} \quad \Omega \times [0, T] \\ u(\mathbf{x}, t) & = & 0 & \text{on} \quad \partial\Omega \times [0, T] \\ u(\mathbf{x}, 0) & = & u_0(\mathbf{x}) & \text{in} \quad \Omega \end{array} \right. , \tag{27}$$

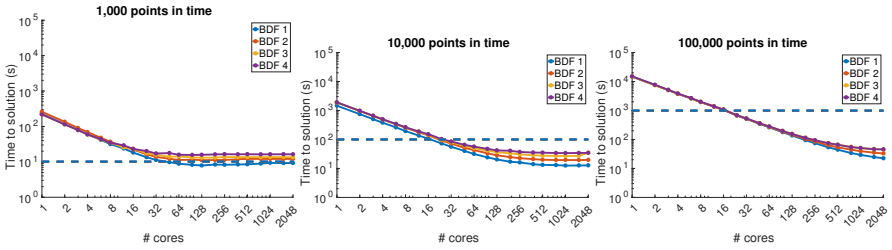
where $\mathbf{x} = (x, y)$ denotes a point in Ω. In this test, the initial condition is a sine: $u_0(\mathbf{x}) = \sin(x)\sin(y)$, and we also choose a sine right-hand side: $f(\mathbf{x}, t) = u_0(\mathbf{x})\sin(2t)$. We integrate on the interval $[0, T]$, $T = 5s$. Two sets of experiments are conducted using BDF methods of order 1 to 4. The first set of experiments uses uniform time steps, and the second uses adaptive time steps. For this diffusion problem, we use a coarsening factor of 4.

5.1.1. Uniform time steps. Here we use uniform time steps, with the number of steps ranging from $1,000$ to $100,000$ points in time. Figure 8 and Table 3 show the scalability results of our implementation of the MGRIT algorithm compared to sequential time stepping with the same integrator. Note that because uniform time steps are used, the order of the BDF method does not influence the time-to-solution of the sequential time stepping.

In the case of $1,000$ time points (Figure 8a), MGRIT can merely reach the sequential times, and no real speedup can be achieved. This lack of speedup is due to the small number of time points. The computational time becomes rapidly negligible compared to communications. However, a higher number of points in time allows for a more efficient use of parallelism, and larger speedups can be obtained. We see a speedup of at least 3 for the $10,000$ point test case (Figure 8b) and at least 21.9 for

TABLE 3
For each BDF method and number of points on the time grid, we show the best time-to-solution achieved by MGRIT, the corresponding speedup compared to the sequential time stepping, and the crossover point (i.e. the number of cores required to reach the sequential time).

# Points		1,000	10,000	100,000
Sequential time		≈ 10.1s	≈ 100s	≈ 993s
BDF 1	time	7.88s	12.54s	22.36s
	speedup	1.3×	7.9×	43.7×
	crossover	≈ 82	≈ 17	≈ 16
BDF 2	time	11.16s	19.45s	32.88s
	speedup	0.9×	5.2×	30.2×
	crossover	–	≈ 22	≈ 16
BDF 3	time	13.11s	26.94s	44.80s
	speedup	0.8×	3.7×	22.2×
	crossover	–	≈ 25	≈ 16
BDF 4	time	15.63s	33.82s	45.44s
	speedup	0.7×	3.0×	21.9×
	crossover	–	≈ 26	≈ 16



(a) 1,000 pts (b) 10,000 pts (c) 100,000 pts

FIG. 8. *Scalability results for the 2D heat equation problem on uniform time grids. The time for sequential time integration is represented by the dotted line. Note that for a fixed number of points, the sequential time is similar for all orders of the BDF method.*

the 100,000 point test case (Figure 8c). Also, it is worth mentioning that the order of the BDF method has an influence on the speedup results. Lower order methods perform slightly better on large numbers of cores. This behavior can be explained by the "trick" we have used to re-write a multistep method into a one step method. For a method of order q, q points in time are gathered into a single, larger unknown. Thus, if N points in time are initially present on the fine grid, after gathering, the MGRIT algorithm only has N/q unknowns, hence limiting the scalability. Also, since the unknowns are q times larger, communications are more expensive.

5.1.2. Variable time steps. In this section, we use the error estimation capability of the BDF methods (recall Section 3.2.2) to do adaptivity in time with MGRIT. Starting with a coarse initial grid, the time grid is refined where needed based on the estimation of the error made during a step. The refinement factor is computed using

the ratio of the current step size and the optimal one given by (10), leading to the formula

$$rfactor = \frac{h_n}{h_{n+1}} = \left\lceil \left(\frac{\|e_n\|}{\varepsilon} \right)^{\frac{1}{q}} \right\rceil, \tag{28}$$

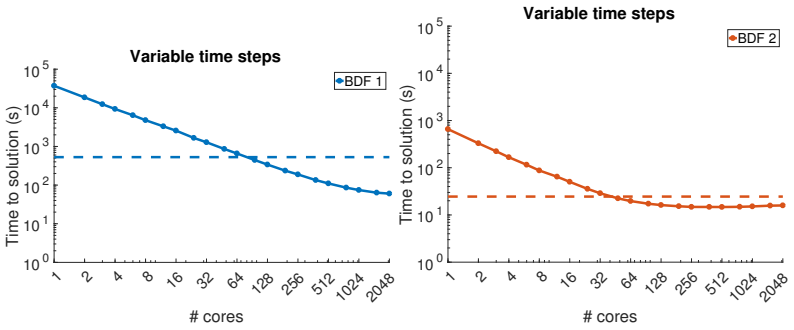
where $\lceil \cdot \rceil$ is the ceiling function (as refinement factors must be integers in XBraid) and e_n is the error estimate. Here we take absolute and relative tolerances of 10^{-8}. This refinement leads to a non-uniform time grid. We present scalability results for two different time domains. In the first case, we integrate on the time domain $[0,\ 5s]$ (Table 4 and Figure 9), and in the second case, the time domain is $[0,\ 50s]$ (Table 5 and Figure 10). As above, we use BDF methods of orders 1 to 4.

In the first case (time interval of $5s$), the scalability is quite limited. A speedup of about 8 can be obtained for the first order method and of about 1.7 for the second order method. No speedup can be achieved for higher order BDF methods. The cause of the poor scalability is easy to understand as BDF methods are highly efficient when using time adaptivity, and only a few points in time are needed to reach the required accuracy. This condition is particularly true for high order methods. For instance, when using a fourth order method, the final fine grid only has 85×4 time points. It is not reasonable to expect speedup for such a small number of time points.

The second case uses a time interval of $50s$, leading to more time points on the final fine grid. As expected, the scalability results are better. Using a first order method leads to a large number of time points (about 700k), so scalability is good. A speedup of about 20 can be obtained with 2,048 cores, but Figure 10a shows that using more cores would lead to more speedup. Using a second order method, we obtain a speedup of about 6.8, and a speedup of about 1.8 is obtained using a third order method. However, no speedup can be obtained using a fourth order method. The number of points (934×4) is still too low.

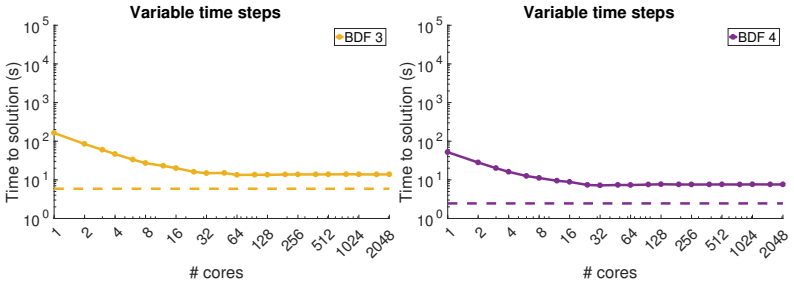
5.2. GridDyn. GridDyn is an electric power transmission grid research code developed at LLNL by P. Top [16]. It uses the SUNDIALS IDA package [13] as its sequential time integrator for DAE systems, which has been replaced by custom BDF and Runge-Kutta time integrators, interfaced with XBraid to achieve parallelism in time. Two test cases are presented here. The first test is the IEEE 39 bus problem described in [4]. The second problem is the reduced WECC system, which features 179 buses. Unless otherwise specified, we use absolute and relative tolerances of 10^{-8} (for the control of step sizes as well as for XBraid stopping criteria) and a coarsening factor of 10. Also, F-cycles are used and no limit on the number of levels in the XBraid hierarchy is imposed. We compare the behavior of the BDF method of order 2, to a Runge-Kutta method of the same order. For fixed time steps, the Runge-Kutta method is SDIRK-2, and for variable time steps, the Runge-Kutta method is an embedded method of order 2 and 4 (meaning two solutions are computed of order 2 and 4, giving an error estimate of order 2, on which the timestep sizes are based).

5.2.1. IEEE 39 bus test case. This test case has 39 buses, 46 links, and 10 generators resulting in 215 total unknowns. A sine load is applied with a period of 2s and an amplitude of 0.1. A BDF2 method is compared to an embedded Runge-Kutta method of order 2-4 using variable time steps for two time intervals, 20s and 100s. Figure 11a shows the evolution of the time to solution for both methods when the interval of integration is 20s. For this run, the final time grid has about 10.5k points using BDF and 16k points using Runge-Kutta. We can see that with the



(a) BDF 1

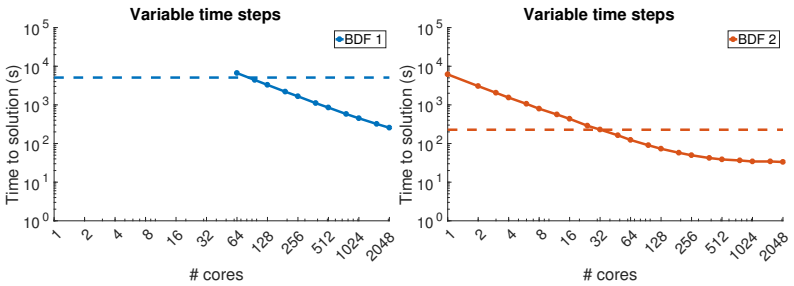
(b) BDF 2



(c) BDF 3

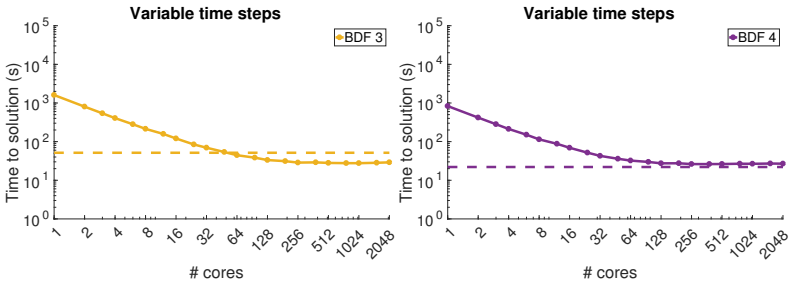
(d) BDF 4

FIG. 9. *Scalability results for the 2D heat equation problem using adaptive time steps and* $T = 5s$. *Sequential time is represented by the dotted line.*



(a) BDF 1

(b) BDF 2



(c) BDF 3

(d) BDF 4

FIG. 10. *Scalability results for the 2D diffusion problem, using variable time steps and* $T = 50s$. *Sequential time is represented by the dotted line.*

TABLE 4

Scalability results on the 2D heat equation problem using variable time steps and $T = 5s$. We indicate in each case the number of points on the final fine time grid, the time-to-solution, and, for the MGRIT algorithm, the speedup and crossover point compared to the sequential case.

		Sequential	MGRIT
BDF 1	# points	54,017	79,923
	time	529.3s	60.2s
	speedup	–	8.8×
	crossover	–	≈ 78
BDF 2	# points	2,443	1,591×2
	time	24.5s	14.7s
	speedup	–	1.7×
	crossover	–	≈ 53
BDF 3	# points	566	217×3
	time	5.9s	13.4s
	speedup	–	0.4×
	crossover	–	–
BDF 4	# points	215	85×4
	time	2.5s	7.2s
	speedup	–	0.3×
	crossover	–	–

sequential approach, the Runge-Kutta method is much more expensive than the BDF method (about 3 times). On the contrary, using MGRIT with XBraid, the BDF method becomes about 2 times more expensive than the Runge-Kutta method, and no speedup can be achieved with the BDF method. A speedup of 5.7 is observed for Runge-Kutta.

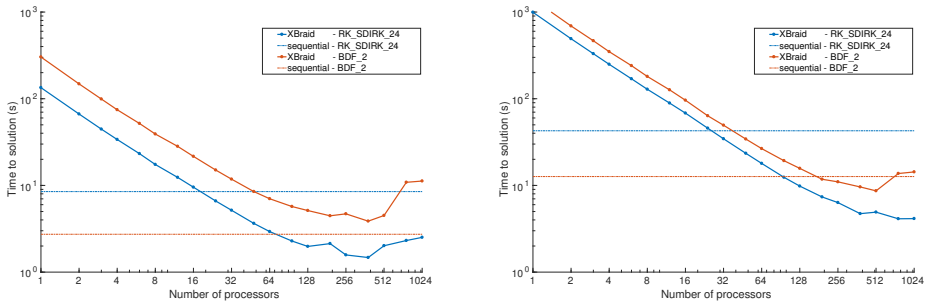
On a longer time integration (Figure 11b), we can make similar remarks. While in serial, the Runge-Kutta method is more expensive, it becomes cheaper using MGRIT with XBraid. A speedup of more that 10 is observed. With the BDF method, the speedup is modest, about 1.5. The poor performance of the BDF method can be explained by a degraded convergence rate on the IEEE 39 bus problem. It takes more than 20 iterations for XBraid to converge (1 iteration to develop the fine grid, and 19 iterations on that grid to get to convergence), while with the Runge-Kutta method, XBraid converges within 5 to 7 iterations.

5.2.2. Reduced WECC system. This problem has 179 buses, 255 links, and 29 generators resulting in 793 total unknowns. The load is a triangular pulse of period 3s and an amplitude of 0.2. Three different test cases were run with parameters described in Table 6. All other parameters, in particular XBraid parameters, are fixed: coarsening factor of 5, tolerance of 10^{-8}, F-cycles, and as many levels as possible. The absolute and relative tolerance for the Newton solver and the error estimate are also 10^{-8}. Two time intervals are used for the integration. For fixed time steps, we use a time interval of 10s, which leads to 1ms or 0.1ms steps. For variable time steps, we use a longer time interval of 100s. Scalability results are presented in Figure 12 and summarized in Table 7.

TABLE 5

*Scalability results for the 2D heat equation problem, using variable time steps and $T = 50s$. We indicate in each case the number of points on the final fine time grid, the time-to-solution, and, for the MGRIT algorithm, the speedup and crossover point compared to the sequential case. *In the case of BDF 1, the limits of scalability are not reached at 2,048 cores; using more cores would lead to better speedup and lower time-to-solution; see Figure 10a*

		Sequential	MGRIT
BDF 1	# points	518,996	796,014
	time	5,092.1s	256.2*s
	speedup	–	19.9*×
	crossover	–	≈ 82
BDF 2	# points	22,911	13,768×2
	time	227.5s	33.3s
	speedup	–	6.8×
	crossover	–	≈ 31
BDF 3	# points	5,182	1,976×3
	time	51.5s	27.7s
	speedup	–	1.8×
	crossover	–	≈ 57
BDF 4	# points	2,183	936×4
	time	22.0s	26.2s
	speedup	–	0.8×
	crossover	–	–



(a) Variable steps, Tmax=20s

(b) Variable steps, Tmax=100s

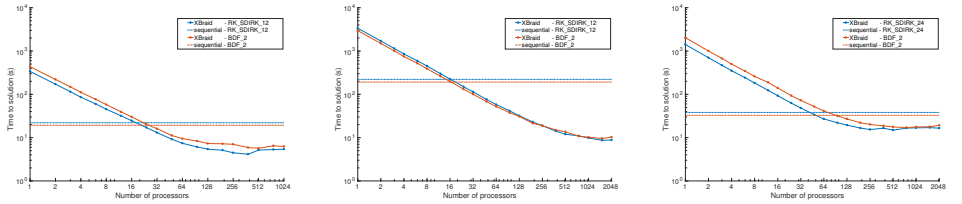
FIG. 11. *Time to solution for the IEEE 39 bus test case versus the number of cores using variable step sizes for different lengths of total time interval. Horizontal lines represent sequential time stepping for reference.*

For fixed time steps, good speedups can be obtained even for 1ms steps. The Runge-Kutta method leads to a 5.4 speedup, while the BDF method gives a 3.4

[2]Note that using variable time stepping, XBraid tends to build grids that are sub-optimal in terms of number of points. This is because at the beginning of the algorithm, the solution is not accurate and we may refine in parts where it would not be actually needed. This behaviour tends to be worse for BDF methods in general.

TABLE 6
Description of the test cases for the Reduced WECC System test problem.

		Time integration	Number of steps Sequential	XBraid	Variable steps?
#1		$10s$	10,000		No
#2		$10s$	100,000		No
#3	RK-24	$100s$	13,315	15,755 [2]	Yes
	BDF-2		16,368	23,872	



(a) 10k pts, Tmax=10s (b) 100k pts, Tmax=10s (c) Variable steps, Tmax=100s

FIG. 12. *Time to solution for the WECC test case (179 buses) versus the number of cores, using fixed and variable step sizes. Horizontal lines represent the sequential time stepping for reference.*

speedup. Using 100k points in time leads again to much better speedups for both methods. In the case of Runge-Kutta, we obtain a speedup of 25.5, while for BDF we see about 20. These speedups are for 2,048 cores. The difference between the Runge-Kutta and the BDF method can be explained here by the fact that Runge-Kutta methods are slightly more expensive, which impacts its sequential time but only mildly the parallel time since more parallelism is available.

Again, the behavior is different for variable time step methods. Here, we use an embedded Runge-Kutta method of order 2 and 4 and a BDF method of order 2. The Runge-Kutta method produces a second order as well as a fourth order solution and compares them to estimate the local error made at each step. Hence, we get an error estimate of second order. Similarly, with the BDF method, both the polynomial estimate and the solution are second order, leading also to a second order error estimate. This means that the Runge-Kutta method produces a number of time points of the same order as the BDF method (since both are based on second order error estimates), even though the Runge-Kutta solution is more accurate. This behaviour is typical of Runge-Kutta methods as the solution and the error estimate are of different order.

If the time to solution for the sequential time stepping shows a difference between Runge-Kutta and BDF (Runge-Kutta is a bit more expensive), this difference is inverted using the MGRIT algorithm (BDF becomes slightly more expensive than Runge-Kutta on 1 core). In the end, the speedup obtained for the Runge-Kutta method is about 2.6 and 1.9 for BDF.

In general, the BDF method performs slightly less well than the Runge-Kutta method when using variable time steps. This difference is due to two factors. First, the BDF methods are highly efficient in sequential mode, with few points in time and are quite cheap (only one nonlinear solve per step). The second point comes from a degraded convergence rate of XBraid, leading to more iterations.

TABLE 7
Summary of the scalability results for the reduced WECC system

	Steps :	Fixed (10k)	Fixed (100k)	Variable
RK	Speedup	5.4	25.5	2.6
	# Iterations	4	4	1
	# Refinements	–	–	7
BDF	Speedup	3.4	20	1.9
	# Iterations	6	4	3
	# Refinements	–	–	9

6. Concluding remarks and future work. This paper developed an MGRIT formulation for variable step BDF methods. The key to the algorithm is a gathering of variables into packets with the same number of time steps as the order of the method. This gathering technique allows the method to be written as a one step method which can easily be incorporated into the MGRIT framework. The resulting time-parallel BDF methods show parallel speedups for large numbers of fine grid time steps. In the variable step case, the high efficiency of the sequential BDF methods, especially for higher orders, does not provide many time steps over which to parallelize resulting in less speedup than achieved by the Runge-Kutta methods. For sequential time stepping, the BDF methods are faster than the Runge-Kutta methods. However, with MGRIT parallelization, the Runge-Kutta methods were shown to be faster.

While the results here show significant progress in incorporating state of the art adaptive linear multistep methods into a parallel-in-time framework, future work will focus on further improvements to the adaptive fine grid development. This work will include investigation of alternate coarsening strategies that will break the gathered time step packets. Such methods will allow for fewer points on the final fine time grid. In addition, we will investigate methods that will emphasize refinement and derefinement as the fine grid is formed in order to have smaller numbers of time points on the fine grid that allow for accurate time stepping.

Acknowledgments. The authors would like to thank Philip Top at LLNL for his invaluable help with the GridDyn package. The authors also thank the Department of Energys Office of Electricity Delivery and Energy Reliability Advanced Grid Modeling (AGM) program for the funding support. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC.

Appendix A. Stability region of a variable step BDF2 method. To determine the stability region of a method, we study its behavior on the test equation

$$\dot{u} = \lambda\, u, \quad \lambda \in \mathbb{C}. \tag{29}$$

Assuming a uniform fine grid with step size h, then by construction of the grid hierarchy (see Section 4 and (26)), on level l the time step size alternates between h and $h\rho_l(2,c) = h\rho$. A general 2-step integrator is written as

$$\beta_{0,n} h_n \dot{u}_n + \beta_{1,n} h_n \dot{u}_{n-1} = \alpha_{0,n} u_n + \alpha_{1,n} u_{n-1} + \alpha_{2,n} u_{n-2} \tag{30}$$

or, using (29)

$$u_n = \underbrace{-\frac{\beta_{1,n}\lambda h_n - \alpha_{1,n}}{\beta_{0,n}\lambda h_n - \alpha_{0,n}}}_{A_n} u_{n-1} + \underbrace{\frac{\alpha_{2,n}}{\beta_{0,n}\lambda h_n - \alpha_{0,n}}}_{B_n} u_{n-2}. \tag{31}$$

Note that, besides the explicit dependency λh_n, the coefficients A_n and B_n depend only on the ratio $\rho_n = h_n/h_{n-1}$, whose value alternates between ρ and $1/\rho$. It follows that

$$A_{2n} = A_2, \quad A_{2n+1} = A_3, \quad B_{2n} = B_2, \quad B_{2n+1} = B_3. \tag{32}$$

Similarly, one gets

$$\begin{aligned} u_{n+1} &= A_{n+1}u_n + B_{n+1}u_{n-1} \\ &= (A_{n+1}A_n + B_{n+1})u_{n-1} + A_{n+1}B_n u_{n-2}. \end{aligned} \tag{33}$$

So, the gathering process described in Section 4 for a 2-step method formally can be written as

$$\underbrace{\begin{bmatrix} u_{2n} \\ u_{2n+1} \end{bmatrix}}_{w_n} = \underbrace{\begin{bmatrix} B_{2n} & A_{2n} \\ A_{2n+1}B_{2n} & A_{2n+1}A_{2n} + B_{2n+1} \end{bmatrix}}_{M_n} \cdot \begin{bmatrix} u_{2n-2} \\ u_{2n-1} \end{bmatrix}. \tag{34}$$

From (32), it comes that $M_{n+1} = M_n$, and the stability of the integration process of w_n (and thus u_n) is directly linked to the eigenvalues of M_n. More specifically, the process is stable if and only if

$$\max |\mathrm{eig}(M_n)| < 1. \tag{35}$$

Also, the eigenvalues are given by

$$\mathrm{eig}^{\pm}(M_n) = \frac{A_2A_3 + B_2 + B_3}{2} \pm \frac{1}{2}\sqrt{(A_2A_3 + B_2 + B_3)^2 - 4B_2B_3}. \tag{36}$$

In our case, $h_n = \rho h$, $h_{n+1} = h$, and using the coefficients given in Table 2 with $\rho_n = \rho$ and $\rho_{n+1} = 1/\rho$, it is easy to evaluate the stability region of the method. If the exact formula is of little interest, it is worth mentioning the behavior for large ρ (i.e. for the coarsest grids). This behavior is different depending on the method (fully variable coefficients or fixed leading coefficient). In both cases, one of the eigenvalues tends to zero. The other eigenvalue is

$$\begin{aligned} &\mathrm{FVC} : \mathrm{eig}^+ \sim \frac{1}{\lambda h - 1} && \text{for } \rho \to +\infty \\ &\mathrm{FLC} : \mathrm{eig}^- \sim \frac{\lambda h}{2(3 - 2\lambda h)}\rho && \text{for } \rho \to +\infty \end{aligned}, \tag{37}$$

and the condition (35) becomes (still for $\rho \to +\infty$) for the fully variable coefficient method

$$\mathrm{FVC} : |\lambda h - 1| > 1 \ . \tag{38}$$

For large ρ (or coarse grids), the stability region of the BDF 2 method with fully variable coefficients corresponds to the exterior of a circle centered at $z = 1$ and of

radius 1. This region means the method remains A-stable on all the coarse grids. On the contrary, the eigenvalue associated to the BDF 2 method with fixed leading coefficient grows as ρ, leading to stability region whose size shrinks as ρ^{-1}. The computation of the stability region for BDF 3 follows the same principle but leads to more complex formulas of little interest and is omitted here.

REFERENCES

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

[2] U. Ascher and L. Petzold, *Computer Methods for Ordinary Differential Equations amd Differential-Algebraic Equations*, SIAM, 1998.

[3] J. Astic, A. Bihain, and M. Jerosolimski, *The mixed adams-bdf variable step size algorithm to simulate transient and long term phenomena in power systems*, IEEE Transactions on Power Systems, 9 (1994), pp. 929–935.

[4] T. Athay, R. Podmore, and S. Virmani, *A practical method for the direct analysis of transient stability*, IEEE Transactions on Power Apparatus and Systems, PAS-98 (1979), pp. 573–584, http://dx.doi.org/10.1109/TPAS.1979.319407.

[5] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical solution of initial-value problems in differential-algebraic equations*, SIAM, 1995.

[6] V. Dobrev, T. Kolev, N. Petersson, and J. Schroder, *Two-level convergence theory for multigrid reduction in time (MGRIT)*, SIAM J. Sci. Comput., 39 (2017), pp. S501–S527. LLNL-JRNL-692418.

[7] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel time integration with multigrid*, SIAM J. Sci. Comput., 36 (2014), pp. C635–C661. LLNL-JRNL-645325.

[8] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, J. B. Schroder, and S. Vandewalle, *Multigrid methods with space-time concurrency*, Computing and Visualization in Science, (2017), http://dx.doi.org/10.1007/s00791-017-0283-9, https://doi.org/10.1007/s00791-017-0283-9. LLNL-JRNL-678572.

[9] R. D. Falgout, T. A. Manteuffel, B. O'Neill, and J. B. Schroder, *Multigrid reduction in time for nonlinear parabolic problems: A case study*, SIAM J. Sci. Comput., 39 (2017), pp. S298–S322. LLNL-JRNL-692258.

[10] M. J. Gander, *50 years of time parallel time integration*, in Multiple Shooting and Time Domain Decomposition Methods, T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, eds., Springer Verlag, 2015, pp. 69–114.

[11] G. Gurrala, A. Dimitrovski, S. Pannala, S. Simunovic, and M. Starke, *Parareal in time for fast power system dynamic simulations*, IEEE Trans. Power Syst., 31 (2016), pp. 1820–1830.

[12] E. Hairer, C. Lubich, and M. Roche, *The numerical solution of differential-algebraic systems by Runge-Kutta methods*, vol. 1409, Springer, 2006.

[13] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, *SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers*, ACM Transactions on Mathematical Software (TOMS), 31 (2005), pp. 363–396.

[14] K. R. Jackson and R. Sacks-Davis, *An alternative implementation of variable step-size multistep formulas for stiff ODEs*, ACM Transactions on Mathematical Software (TOMS), 6 (1980), pp. 295–318.

[15] E. R. Keiter, T. Mei, T. V. Russo, E. L. Rankin, R. L. Schiek, H. K. Thornquist, J. C. Verley, D. A. Fixel, T. S. Coffey, R. P. Pawlowski, et al., *Xyce parallel electronic simulator: reference guide.*, tech. report, Sandia National Laboratories, 2012.

[16] B. M. Kelley, P. Top, S. G. Smith, C. S. Woodward, and L. Min, *A federated simulation toolkit for electric power grid and communication network co-simulation*, in Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2015 Workshop on, April 2015, pp. 1–6, http://dx.doi.org/10.1109/MSCPES.2015.7115406.

[17] M. La Scala and A. Bose, *Relaxation/Newton methods for concurrent time step solution of differential-algebraic equations in power system dynamic simulations*, Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 40 (1993), pp. 317–330.

[18] M. La Scala, A. Bose, D. J. Tylavsky, and J. S. Chai, *A highly parallel method for transient stability analysis*, IEEE Trans. Power Syst., 5 (1990), pp. 1439–1446.

[19] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems*, Wiley, 1991.

[20] M. Lecouvez, R. D. Falgout, C. S. Woodward, and P. Top, *A parallel multigrid reduc-*

tion in time method for power systems, in Power and Energy Society General Meeting (PESGM), 2016, IEEE, 2016, pp. 1–5. LLNL-CONF-679148.

[21] 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.

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

[23] C. Parmer, E. Cotilla-Sanchez, H. K. Thornquist, and P. D. Hines, *Developing a dynamic model of cascading failure for high performance computing using trilinos*, in Proceedings of the first international workshop on High performance computing, networking and analytics for the power grid, ACM, 2011, pp. 25–34.

[24] M. Ries and U. Trottenberg, *MGR-ein blitzschneller elliptischer löser*, Tech. Report Preprint 277 SFB 72, Universität Bonn, 1979.

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

[26] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, San Diego, 2001.