MATHEMATICS OF COMPUTATION
Volume 75, Number 255, July 2006, Pages 1403–1428
S 0025-5718(06)01832-1
Article electronically published on February 24, 2006

CONVERGENT ITERATIVE SCHEMES FOR TIME PARALLELIZATION

IZASKUN GARRIDO, BARRY LEE, GUNNAR E. FLADMARK, AND MAGNE S. ESPEDAL

Abstract. Parallel methods are usually not applied to the time domain because of the inherit sequentialness of time evolution. But for many evolutionary problems, computer simulation can benefit substantially from time parallelization methods. In this paper, we present several such algorithms that actually exploit the sequential nature of time evolution through a predictor-corrector procedure. This sequentialness ensures convergence of a parallel predictor-corrector scheme within a fixed number of iterations. The performance of these novel algorithms, which are derived from the classical alternating Schwarz method, are illustrated through several numerical examples using the reservoir simulator `Athena`.

1. Introduction

Hydrocarbon flow in porous media is often approximated with a mathematical model involving three coupled nonlinear evolution equations for the primary variables of temperature, pressure, and molar masses, and tabulated values based on bubble and dew point curves for the secondary variables. Since this model is generally too difficult to solve analytically, a simplified model is solved numerically by decoupling the equations and discretizing them using finite volume in space and backward Euler in time. The resulting nonlinear system of equations is then solved using the iterative Newton-GMRES algorithm [3], [4].

This is the overall structure of `Athena`, our current simulator for flow in porous media. The goal of `Athena` is to simulate a wide variety of flow scenarios within reasonable accuracy, in reasonable computational time. However, to simulate realistic problems, the standard numerical algorithms are unacceptably slow. Thus, it is necessary to develop faster, parallel algorithms.

The standard numerical approach is to sequentially solve each primary variable using a fixed time-step determined by the smallest evolutionary time-scale for the primary variables, even though the time-scales for these variables may be an order of magnitude different. In particular, the time-scale for temperature and pressure is often an order of magnitude larger than that for the molar masses. Hence, the temperature and pressure can be computed with larger time-steps. To mitigate this time-step restriction, a common procedure is to compute the temperature and pressure with time-step ΔT, while substepping G times for the molar masses with

Received by the editor May 29, 2003 and, in revised form, April 20, 2005.
2000 *Mathematics Subject Classification.* Primary 65N55, 65Y05; Secondary 65M55, 65M60.
Key words and phrases. Alternating Schwarz, time parallelization, reservoir simulator, multilevel, full approximation storage.

smaller time-step $\frac{\Delta T}{G}$. However, this substepping does not lead to acceptable computation efficiency in a serial computing environment.

The goal of this paper is to develop parallelization techniques for the substep procedure, in order to speed up the computation. An ideal algorithm is one that would completely bypass the sequential nature of time evolution in a parallelized substep march. But this would go against physics, and so, may lead to poor accuracy. Hence, we propose a novel method, similar to the Parareal technique studied in [1, 8, 2], that allows parallelization in the substep procedure and is also causal. Causality is obtained by using a coarser time-step in a predictor-corrector manner. Both the Parareal technique and this proposed method have a two-level structure, with the predictor step defined on the coarse time level and the corrector step defined on the fine level. The novelty in our approach is to decrease the time domain as the predictor-corrector progresses.

This paper is organized as follows. For background and self-containment, Section 2 describes the equations of hydrocarbon flow in porous media. Given these equations, in Section 3, we review some of the existing parallel algorithms to handle evolution equations, and give a general description of a predictor-corrector (PC) algorithm for parallelizing time. This general description is given in terms of an iterative method for solving a lower block bidiagonal matrix. The details of this PC algorithm are given in Section 4. In this section, the PC method is viewed as a two-level scheme, where the coarse grid and fine grid procedures are respectively the predictor and corrector steps. Section 4 further derives defect terms that modify the predictor equations, introduces modifications to the PC algorithm that ensure convergence within a fixed number of iterations, and analyzes the computational efficiency of this two-level algorithm. Section 5 develops a multilevel extension that alleviates some of the load-balance issues of the two-level method. The intricacies of this extension are given, as well as a description of the computational costs for a cycle of this multilevel method. Numerical examples are presented in Section 6 to illustrate the performance of these algorithms. Finally, Section 7 gives the conclusions of this work.

We note that this paper concentrates on the derivation and properties of the numerical algorithm, avoiding unnecessary details of our compositional simulator `Athena`. For further details on `Athena`, we refer the interested reader to [5], [9], [10], and [11].

2. Equations of hydrocarbon flow

We are interested in a multicomponent multiphase fluid flow in a porous media region V with surface boundary S. The primary variables of interest are temperature (T), water pressure (p_w), and molar masses (N_ν), for each component (ν) of the multicomponent multiphase fluid. The phases considered are oil (o), gas (g), and water (w). The mathematical model for hydrocarbon flow in porous media involves conservation laws for the molar masses and internal energy, and a water pressure equation. Conservation of molar mass for a flow through a *porous media* region V is given by the integral relation

$$\frac{\partial}{\partial t}\int_V m_\nu \, dV + \int_V \nabla \cdot \vec{m}_\nu = -\int_V q_\nu \, dV \,, \tag{2.1}$$

where q_ν is the source/sink for molar mass density component m_ν (molar mass N_ν is the volume integral of m_ν). Conservation of energy is enforced through the heat flow equation

$$\frac{\partial}{\partial t}\int_V (\rho u) dV - \int_S (\underline{k}\nabla T)\cdot d\vec{S} = -\int_S h\rho\vec{u}\cdot d\vec{S} + \int_V q dV \ , \tag{2.2}$$

where T is the temperature, $\underline{k}$ is the bulk heat conductivity, ρ is the density, and u is the internal energy. And, an equation for the water pressure is derived by imposing that the pore volume V_p be equal to the sum volume of all three fluid phases at all times, i.e.,

$$R = V_p - \sum_{l=g,o,w} V^l$$

is zero at all times. This residual volume is a function of the water pressure p^w, the overburden pressure W, and each molar mass component N_ν. Hence, a first-order Taylor expansion of $R(t + \Delta t)$ about t, together with the chain rule applied to $\partial R(p^w, W, N_\nu)/\partial t$, leads to the water pressure equation

$$\frac{\partial R}{\partial p^w}\frac{\partial p^w}{\partial t} + \sum_{\nu=1}^{n_c}\frac{\partial R}{\partial N_\nu}\frac{\partial N_\nu}{\partial t} = -\frac{R}{\Delta t} - \frac{\partial R}{\partial W}\frac{\partial W}{\partial t} \ . \tag{2.3}$$

Equations (2.1)–(2.3) are three *coupled* nonlinear differential systems. The pressure and molar masses are clearly coupled in (2.3). The temperature is coupled to the pressure and molar masses through its Jacobian. This Jacobian is also dependent on the rock temperature, which in reservoir simulation has a much lower variation rate than the pressure and molar masses.

Now, to numerically solve this differential system, these equations are decoupled and discretized using finite volume in space and backward Euler in time. Thus, at each time-step interval, $[T^n, T^{n+1}]$, three discrete Jacobian systems of the form $\mathcal{A}^{[n]}\Delta\vec{x} = \vec{f}^{[n]}$ are solved in sequential order. These systems are solved with an iterative Newton–Raphson method using, for example, preconditioned GMRES [3], [4]. In general, at each time-step these Jacobian matrices change after each Newton iteration. But for the temperature equation, since its Jacobian matrix is majorized by the rock temperature, its Jacobian will be changed only between each time-step. Having determined the temperature at time $T^{[n+1]}$, the pressure and molar masses must be determined using a Newton–Raphson iteration. As for the pressure system, its Jacobian has off-diagonal terms that depend on the molar masses. Because of the decoupling, these off-diagonal terms will involve only time-lagged molar masses. Thus, the nonlinearities in the discrete pressure equation are restricted to the diagonal terms $\mathcal{A}^{[n]} = \mathcal{D}^{[n(s)]}$, where the superscript s corresponds to the number of Newton iterations. Turning to the molar mass, as noted in Section 1, the rate of change for the temperature and pressure are comparatively small compared to the rate of change for the molar masses. So, for each time-step interval, the molar mass equations are substepped with step-size $\frac{T^{n+1}-T^n}{G}$. Given the temperature and pressure, the nonlinear molar mass system at time-level T^n is

$$\mathcal{A}^{[n(s)]}(\vec{N}^{[(n)(s+1)]} - \vec{N}^{[n(s)]}) \quad = \quad \vec{f}^{[n(s)]}. \tag{2.4}$$

Superscript s will be omitted in the remainder of this paper.

For further details of the discretization of (2.1)–(2.3), we refer the interested reader to [7].

3. Parallel algorithms for evolutionary equations

In this section, we review some of the existing parallel algorithms for evolutionary equations. As alluded to in the previous section, the most computationally intensive procedure in hydrocarbon flow simulation is the substep march for the molar masses. If this procedure can be computed in parallel, then the computation of the molar masses can be substantially improved. In general, such parallelization can improve the computational performance for any of the three variables.

The solution procedure for an evolutionary equation time-discretized with backward Euler can be viewed as a forward substitution (or an approximation) for the lower block bidiagonal matrix system

$$\begin{pmatrix} \mathcal{L}_1 & & & & \\ \mathcal{M}_1 & \mathcal{L}_2 & & & \\ & \mathcal{M}_2 & \mathcal{L}_3 & & \\ & & \ddots & \ddots & \\ & & & \mathcal{M}_{N-1} & \mathcal{L}_N \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_N \end{pmatrix}, \tag{3.1}$$

where $\mathcal{L}_i$ is the matrix operator acting on the unknown at the i^{th} time-step and $\mathcal{M}_{i-1}$ is the matrix operator acting on the unknown of the previous time-step. A direct forward substitution is clearly sequential in time. However, to achieve some parallelism, a common approach is to parallelize in space as time is sequentially marched. This corresponds to forward substituting (3.1) with parallelism in the inversion of $\mathcal{L}_i$. For example, when the evolutionary equation is parabolic, inversion of $\mathcal{L}_i$ corresponds to solving an elliptic equation, in parallel. This standard approach definitely does not exploit full parallelism, especially when the temporal grid is much finer than the spatial grid.

Two other parallel approaches for time-dependent problems are waveform relaxation and overlapping Schwarz waveform relaxation. These methods do achieve parallelism in time. To describe these approaches, consider a one-dimensional time-dependent problem, i.e., 1-1 dimensional in space-time. Spatially discretizing this problem leads to a system of ode's, one equation for each spatial grid point:

$$\frac{du}{dt} + \mathcal{A}u = f, \tag{3.2}$$

where $\mathcal{A}$ is the matrix that describes the coupling produced by the spatial discretization. Denoting the diagonal of $\mathcal{A}$ with $\mathcal{D}$, a Jacobi-type iterative scheme for solving (3.2) is

$$\frac{du^k}{dt} + \mathcal{D}u^k = f + (\mathcal{D} - \mathcal{A})u^{k-1}, \tag{3.3}$$

which leads to a system of decoupled ode's at each iteration. Each of these time-dependent problems is then solved for the complete time interval on different CPU's. This is the basic waveform relaxation approach. A generalization of it is the overlapping Schwarz waveform scheme ([6]). Rather than subdividing the spatial domain with grid points, the overlapping Schwarz method decomposes the spatial domain into several overlapping subdomains; see Figure 1. On each subdomain, a time-dependent problem must be solved. These subdomain problems are solved in

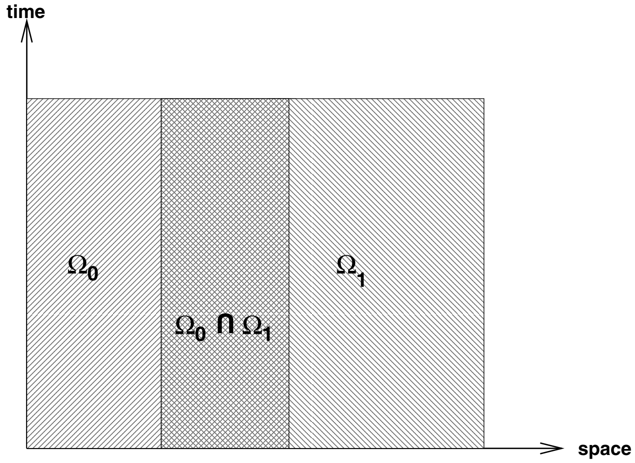

FIGURE 1. Domain decomposition for the overlapping Schwarz waveform method.

parallel, with an appropriate strategy for updating the subdomain interface values.

Waveform relaxation suffers from poor convergence. Hence, several multigrid schemes have been developed by Vandewalle et al. ([14]) to overcome this convergence problem. Overall, this scheme can be viewed as applying multigrid to a fully discretized time-dependent problem, with time viewed as another spatial variable. However, because the space-time grid is generally anisotropic (e.g., $\Delta t \ll \Delta x$), line relaxation or semi-coarsening must be used in the multigrid procedure ([13], [15]). For our 1-1 dimensional problem, line relaxation in the time direction, as the strength of connection is in the time direction when $\Delta t \ll \Delta x$, is equivalent to solving a one-dimensional evolutionary equation. The solving of this line is performed using cyclic reduction, which can be done recursively in a multigrid fashion and in parallel. Vandewalle et al. has also examined semi-coarsening and pointwise smoothing, which lead to better parallelism.

The above parallel approaches have been most successfully applied to second-order parabolic equations. However, for hyperbolic systems, the smoothing characteristic of parabolic equations is not present, and so, the overlapping Schwarz and multigrid methods may suffer. Moreover, for higher-dimensional equations, algorithmic/implementation issues can make these methods impractical.

Before introducing an alternative approach, it is insightful to examine the cyclic reduction scheme for solving (3.1). Cyclic reduction is a direct method that combines the even numbered equations with the odd numbered equations to create a system half the original size and with the same bandwidth pattern. For example, when $N = 8$ in (3.1), the reduced system is

$$\begin{pmatrix} \mathcal{L}_2 & & & \\ -\mathcal{M}_3\mathcal{L}_3^{-1}\mathcal{M}_2 & \mathcal{L}_4 & & \\ & -\mathcal{M}_5\mathcal{L}_5^{-1}\mathcal{M}_4 & \mathcal{L}_6 & \\ & & -\mathcal{M}_7\mathcal{L}_7^{-1}\mathcal{M}_6 & \mathcal{L}_8 \end{pmatrix} \begin{pmatrix} u_2 \\ u_4 \\ u_6 \\ u_8 \end{pmatrix} = \begin{pmatrix} f_2 - \mathcal{M}_1\mathcal{L}_1^{-1}f_1 \\ f_4 - \mathcal{M}_3\mathcal{L}_3^{-1}f_3 \\ f_6 - \mathcal{M}_5\mathcal{L}_5^{-1}f_5 \\ f_8 - \mathcal{M}_7\mathcal{L}_7^{-1}f_7 \end{pmatrix}.$$

The solution components of the smaller system and original system are the same, and this size reduction procedure can be repeated recursively until a small system that can be easily solved is formed. Once the solution of the smallest system has been determined, it can be used to calculate the unresolved values of the next larger system. This back substitution procedure is recursively applied up to the original system. Of course, a problem is the inversion of $\mathcal{L}_i$'s when they are submatrices, as is the case in higher-dimensional evolutionary equations. Nevertheless, this procedure gives insight into developing other methods.

One can view the above procedure as initially "ignoring" the odd numbered unknowns and modifying the original system to account for this ignoring. Suppose that instead of ignoring and modifying, approximations to the odd numbered components can be computed. Then with these values, the following reduced system can be formed:

$$\begin{pmatrix} \mathcal{L}_2 & & & \\ & \mathcal{L}_4 & & \\ & & \mathcal{L}_6 & \\ & & & \mathcal{L}_8 \end{pmatrix} \begin{pmatrix} u_2 \\ \tilde{u}_4 \\ \tilde{u}_6 \\ \tilde{u}_8 \end{pmatrix} = \begin{pmatrix} f_2 - \mathcal{M}_1 u_1 \\ f_4 - \mathcal{M}_3 \tilde{u_3} \\ f_6 - \mathcal{M}_5 \tilde{u_5} \\ f_8 - \mathcal{M}_7 \tilde{u_7} \end{pmatrix}, \tag{3.4}$$

where $\tilde{u_i}$ denotes an approximation to u_i. In fact, this reduction can be done in a coarser fashion. For example, suppose only an approximation to u_4 is known. Then the reduced or decoupled system is

$$\begin{pmatrix} \begin{bmatrix} \mathcal{L}_1 & & & \\ \mathcal{M}_1 & \mathcal{L}_2 & & \\ & \mathcal{M}_2 & \mathcal{L}_3 & \\ & & \mathcal{M}_3 & \mathcal{L}_4 \end{bmatrix} & \\ & \begin{bmatrix} \mathcal{L}_5 & & & \\ \mathcal{M}_5 & \mathcal{L}_6 & & \\ & \mathcal{M}_6 & \mathcal{L}_7 & \\ & & \mathcal{M}_7 & \mathcal{L}_8 \end{bmatrix} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \tilde{u}_5 \\ \tilde{u}_6 \\ \tilde{u}_7 \\ \tilde{u}_8 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 - \mathcal{M}_4 \tilde{u}_4 \\ f_6 \\ f_7 \\ f_8 \end{pmatrix}. \tag{3.5}$$

Each subblock is lower bidiagonal, and so, if approximations to some of the subblock unknowns are available, then this procedure can be applied recursively. Moreover, because the subblocks are decoupled, each subblock system can be performed in parallel. Naturally, since only approximate values for the lower subblocks are used, this procedure must be repeated iteratively. This is the essence of the novel parallel method we propose.

4. A two-level algorithm—the general scheme

So the idea of our parallel method is to compute approximations to the solution at certain time levels. These approximations will permit the time march to decouple over subintervals, which in turn, will lead to parallelization of the march over the whole time interval. To compute these approximations, a predictor-corrector procedure is used. But unlike a typical predictor-corrector (PC) method, where an explicit scheme predicts the solution at a fixed time and an implicit method corrects

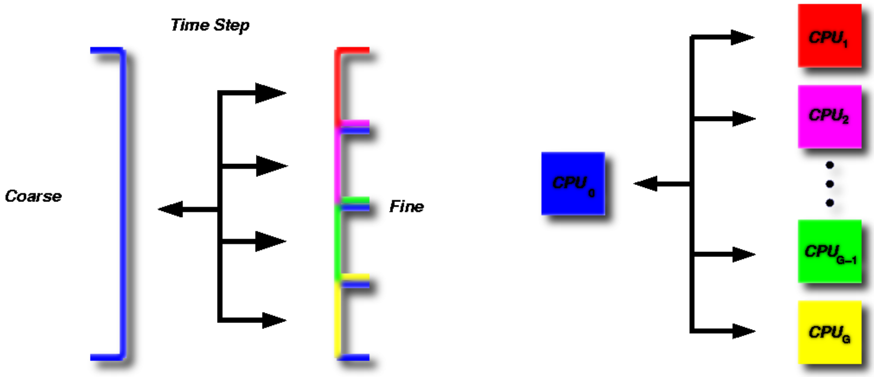

FIGURE 2. The solution over the total coarse domain (in the left) determines the BCs for each parallel-fine system.

the prediction, in our method, the solution is both predicted and corrected with backward-Euler, at a collection of time levels, and with the corrector computed in parallel. The difference between our predictor and corrector procedures lies in the size of the time-steps, since the predictor uses a coarse step whereas the corrector uses a fine step. Thus, the overall parallel procedure can be viewed somewhat as a full approximation storage (FAS) multilevel scheme [13]. We describe this method corresponding to matrix equation (3.4) first.

Consider the domain $\bar{\Omega} = \Gamma \cup \Omega$ as shown in Figure 2, where $\Gamma = T^n$ denotes the boundary where the initial boundary condition is given and $\Omega = (T^n, T^{n+1}]$. The diameter $\Delta T = (T^{n+1} - T^n)$ of Ω can be viewed as the time-step for the temperature and pressure equations in the hydrocarbon flow model. Recall that the target temporal grid for the molar mass equations has mesh size $\Delta t = \frac{\Delta T}{G}$. Besides Ω and the target temporal grid, we introduce an intermediate grid obtained by subdividing Ω into P subdomains Ω_i with mesh size $\Delta P = p\Delta t$. Hence, $\Omega = \bigcup_{i=1}^{P} \Omega_i$, $\Omega_i = (T^n + (i-1)\Delta P, T^n + i\Delta P]$, and $G = Pp$. For example, in Figure 2, we have $P = 4$, $p = 1$, and $G = 4$, whereas in Figure 3, $P = 3$, $p = 4$ and $G = 12$. Subdomains Ω_i for $i > 1$ have interface boundaries Γ_i, and Ω_1 has the actual boundary $\Gamma_1 = \Gamma$. Also, inside each subdomain, different discretizations that appropriately match along the interfaces can be used, though we opt to use the same backward Euler and finite volume discretizations in each subdomain. We now have the multilevel grid hierarchy for the algorithm.

Consider the two-level grid setup of Figure 2. In this predictor-corrector algorithm, we alternatively solve over the entire coarse domain Ω (predictor) and in the nonoverlapping fine subdomains (corrector). Let $N^{[n]}$ denote the initial boundary condition (BC) at time T^n. Tracking the predictor-corrector iterate with k, in the coarse domain, the residual boundary value problem (BVP) is

$$\begin{aligned} \mathcal{A}^{[n],k} \Delta u^{[n],k} &= f^{[n],k} & \bar{\Omega}, \\ u_{|\Gamma}^{[n],k} &= N^{[n]} & \Gamma\ , \end{aligned} \tag{4.1}$$

where $\Delta u^{[n],k} = u^{[n+1],k} - u^{[n],k}$. Having solved this BVP, the initial condition $u^{[n],k}_{|\Gamma} = u^{[n],k}_{|\Gamma_1} = N^{[n]}$ together with some approximations to the interface values $u^{[n],k}_{|\Gamma_i}, i = 2, \ldots, P$, serve as initial conditions for the subdomain problems to be solved in parallel. These interface approximations, $g_i^{[n],k}$, can be obtained by linearly interpolating $u^{[n],k}$ and the computed solution $u^{[n+1],k}$, or they can be constructed explicitly with a substep march over the intermediate grid if $p > 1$ in the grid setup. In either case, solving equation (4.1) is the predictor step, which determines boundary conditions for the fine subdomain systems

$$\left.\begin{array}{rcll} \mathcal{A}_i^{[n],k} \Delta u_i^{[n],k} & = & f_i^{[n],k} & \bar{\Omega}_i \\ u_{i|\Gamma_i}^{[n],k} & = & g_i^{[n],k} & \Gamma_i \end{array}\right\} \qquad i = 1, \ldots, P. \tag{4.2}$$

Note that subdomain Ω_1 is also solved on the fine grid.

Now, having solved (4.2) in parallel, their solutions provide a correction to the coarse predictor system. Several choices for this correction will be given in the next subsection. Letting $\mathcal{S}$ denote this correction, the overall predictor-corrector cycle can be viewed as a block Gauss-Seidel iteration: the predictor step sequentially solves the entire time-domain using correction $\mathcal{S}$ constructed from the previous corrector solution, and the corrector step immediately uses the predictor solution to determine the interface boundary values.

Iterate

- on processor 0 solve

$$\mathcal{A}^{[n],k}_{|\Omega} \Delta u^{[n],k}_{|\Omega} = f^{[n],k} + \mathcal{S}^{[n],k} - \mathcal{A}^{[n],k}_{|\Gamma} N^{[n]}; \tag{4.3}$$

- on processor i, $i = 1, \ldots, P$, solve

$$\mathcal{A}^{[n],k}_{i|\Omega_i} \Delta u^{[n],k}_{i|\Omega_i} = f_i^{[n],k} - \mathcal{A}^{[n],k}_{i|\Gamma_i} g_i^{[n],k}. \tag{4.4}$$

There are many strategies for determining when this alternating method has converged. Due to the solution smoothness in time, we stop this alternating iteration when the solutions of (4.3) and (4.4) at $i = P$ differ within some given tolerance, e.g., the tolerance can be $O(\Delta T - \Delta t)$, the difference between the order of the coarse and fine time discretizations.

4.1. **Correction terms.**

To derive a correction term for the predictor equation, we establish a relation between the predictor and corrector solutions. To accomplish this, we assume a multilevel grid setup with $p > 1$, i.e., the intermediate subdomain partitioning and the target fine grid are different. With this setup, equation (4.1) can be solved with a substep march over the subdomain partitioning. Such a march will then explicitly determine the interface boundary conditions for the fine corrector equations, instead of linearly interpolating them from $u^{[n],k}$ and $u^{[n+1],k}$. Hence, the BVPs

$$\left.\begin{array}{rcl} \mathcal{A}^{[n],k}_{|\bar{\Omega}_i} \Delta u^{[n],k} & = & f^{[n],k}_{|\bar{\Omega}_i} \\ u^{[n],k}_{|\Gamma_i} & = & N^{[n]} + \sum_{j=1}^{i-1} \Delta u^{[n],k}_{|\Omega_j} \end{array}\right\} \qquad i = 1, \ldots, P, \tag{4.5}$$

are sequentially solved in the predictor procedure. Moreover, because $p > 1$, a substep march must also be conducted in the corrector procedure. Indexing the substeps in subdomain Ω_i by $i(j)$, these fine corrector equations are

$$\left.\begin{array}{rcl} \mathcal{A}^{[n],k}_{i|\bar{\Omega}_{i(j)}}\Delta u^{[n],k}_{i(j)} & = & f^{[n],k}_{i|\bar{\Omega}_{i(j)}} \\ u^{[n],k}_{i|\Gamma_{i(j)}} & = & u^{[n],k}_{i|\Gamma_{i(1)}} + \sum_{l=1}^{j-1}\Delta u^{[n],k}_{i|\Omega_{i(l)}} \end{array}\right\} \qquad j = 1, \ldots, \tag{4.6}$$

with interface boundary condition $u_{i|\Gamma_{i(1)}} = g_i$.

To establish a relation between the predictor and corrector solutions, we add and subtract $\mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i}\Delta u^{[n],k-1}_{|\bar{\Omega}_i}$ from the first equation of (4.5) to get

$$\Delta u^{[n],k-1}_{|\bar{\Omega}_i} = \Delta u^{[n],k-1}_{|\bar{\Omega}_i} + (\mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i})^{-1}(\mathcal{A}^{[n],k}_{|\bar{\Omega}_i}\Delta u^{[n],k} - f^{[n],k}_{|\bar{\Omega}_i}). \tag{4.7}$$

Expanding $\Delta u^{[n],k-1}_{|\bar{\Omega}_i}$, (4.7) is equivalent to

$$u^{[n],k-1}_{|\Gamma_{i+1}} = u^{[n],k-1}_{|\Gamma_{i+1}} + (\mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i})^{-1}(\mathcal{A}^{[n],k}_{|\bar{\Omega}_i}\Delta u^{[n],k} - f^{[n],k}_{|\bar{\Omega}_i}). \tag{4.8}$$

Since corrector system (4.6) is solved more accurately over Ω_i than in the predictor march, we substitute $u^{[n],k-1}_{|\Gamma_{i+1}}$ on the right-hand side of (4.8) with $u^{[n],k-1}_{i|\Gamma_{i+1}}$:

$$u^{[n],k-1}_{|\Gamma_{i+1}} = u^{[n],k-1}_{i|\Gamma_{i+1}} + (\mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i})^{-1}(\mathcal{A}^{[n],k}_{|\bar{\Omega}_i}\Delta u^{[n],k} - f^{[n],k}_{|\bar{\Omega}_i}) \ ,$$

or equivalently,

$$\mathcal{A}^{[n],k}_{|\bar{\Omega}_i}\Delta u^{[n],k} = f^{[n],k}_{|\bar{\Omega}_i} + \mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i}(u^{[n],k-1}_{|\Gamma_{i+1}} - u^{[n],k-1}_{i|\Gamma_{i+1}}) \ . \tag{4.9}$$

Equation (4.9) is the *corrected* predictor equation we solve on the coarse grid to get the interface value $u^{[n],k}_{|\Gamma_{i+1}}$ for the corrector equation. Note that due to the nonlinearity of the molar mass system, the correction term $\mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i}(u^{[n],k-1}_{|\Gamma_{i+1}} - u^{[n],k-1}_{i|\Gamma_{i+1}})$ is also nonlinear, and hence, must be updated at every Newton iteration.

Another correction scheme can be derived by adding and subtracting all earlier predictor-corrector iterates to (4.8) and using the more accurate $u^{[n],j}_{i|\Gamma_{i+1}}$'s:

$$\mathcal{A}^{[n],k}_{|\bar{\Omega}_i}\Delta u^{[n],k} = f^{[n],k}_{|\bar{\Omega}_i} + \sum_{j=1}^{k-1}\mathcal{A}^{[n],j}_{|\bar{\Omega}_i}(u^{[n],j}_{|\Gamma_{i+1}} - u^{[n],j}_{i|\Gamma_{i+1}}) \ . \tag{4.10}$$

This is the Parareal correction term proposed by Maday and Lions ([1], [8], [2]). For weakly nonlinear problems, a modification of this Parareal correction term is

$$\mathcal{A}^{[n],k}_{|\bar{\Omega}_i}\Delta u^{[n],k} = f^{[n],k}_{|\bar{\Omega}_i} + \mathcal{A}^{[n],k}_{|\bar{\Omega}_i}\sum_{j=1}^{k-1}(u^{[n],j}_{|\Gamma_{i+1}} - u^{[n],j}_{i|\Gamma_{i+1}}) \ . \tag{4.11}$$

The implementation and computation of this latter correction term is greatly simplified because only the current matrix operator has to be formed.

Yet another correction scheme follows naturally by reformulating the PC scheme as a two-level FAS method [13]. In each of the above correction terms, the operator $\mathcal{A}^{[n],j}_{|\bar{\Omega}_i}$ is applied to the difference of a predictor solution and a corrector solution. Alternatively, a correction term can be formed using the operators of the two time-levels and only the more accurate corrector solution:

$$\left[\mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i} - \mathcal{A}^{[n],k-1}_{i|\bar{\Omega}_{i(p)}}\right]u^{[n],k}_{i|\Gamma_{i+1}} \ , \tag{4.12}$$

giving the corrected predictor equation

$$\mathcal{A}^{[n],k}_{|\bar{\Omega}_i} \Delta u^{[n],k} = f^{[n],k}_{|\bar{\Omega}_i} + \left[\mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i} - \mathcal{A}^{[n],k-1}_{i|\bar{\Omega}_{i(p)}} \right] u^{[n],k}_{i|\Gamma_{i+1}} . \tag{4.13}$$

Correction term (4.12) can be viewed as an approximation to the local truncation error at time Γ_{i+1}, which is the term that must be added to the right-hand side of the predictor equation to produce a coarse grid solution with fine grid discretization accuracy [13].

4.2. **Improved parallel PC algorithm.** Employing any of the correction terms given in (4.9)–(4.11) and (4.13), we have a parallel PC algorithm. The accuracy of this method is determined by the accuracy of the corrector procedure. Each iteration consists of first solving over the *whole* coarse time interval and then asynchronously solving over each subdomain. However, causality implies that the solution in the initial subdomain is not affected by the solution time subdomains at a later time. This observation permits a simple modification of the PC scheme that ensures convergence within P iterations. The idea is to decrease the *active* time interval by removing the initial subdomain after each PC iteration. In particular, in the first PC iteration, since both the predictor and corrector solve the *same* initial boundary-value problem in subdomain $\bar{\Omega}_1$, and since the solution in this subdomain is physically unaffected by the solution in the later subdomains, one can take the more accurate fine-grid corrector approximation to be the solution in $\bar{\Omega}_1$. Having determined the solution in $\bar{\Omega}_1$, this subdomain can be eliminated from the active time interval:

$$\Omega_{\text{active}} = \bar{\Omega}/\bar{\Omega}_1 \ . \tag{4.14}$$

For the next PC iteration, on initial subdomain $\bar{\Omega}_2$ of the current Ω_{active}, the computed fine grid solution $u^{[n],1}_{1|\Gamma_2}$ will be taken as the initial condition

$$u^{[n],2}_{|\Gamma_2} = u^{[n],1}_{1|\Gamma_2} \ . \tag{4.15}$$

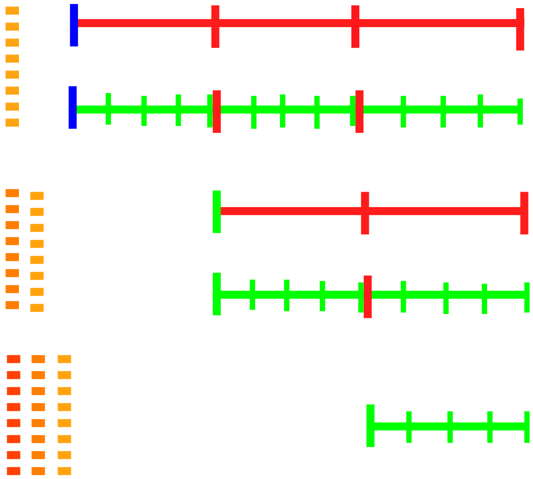

FIGURE 3. Active domain for the improved PC algorithm.

Although the right-hand side of the predictor equation in $\bar{\Omega}_2$ now has a nonzero correction term, the actual equation being solved is the fine-grid corrector equation. Besides, the correction term in the predictor equation is used only to generate accurate subdomain interface conditions for the corrector systems (cf. Section 5.1 concerning the correction term 4.13). Hence, again we take the more accurate fine-grid corrector approximation in $\bar{\Omega}_2$, and redefine the active time interval to $\Omega_{\text{active}} = \bar{\Omega}/\bigcup_{i=1}^{2}\bar{\Omega}_i$. After repeating this process $k = P-1$ times, the computed solution at the Γ_i's will be

$$u^{[n],P}_{|\Gamma_1} = g, \ldots, u^{[n],P}_{|\Gamma_P} = u^{[n],P-1}_{P-1|\Gamma_P}, \tag{4.16}$$

and the active time interval will be $\Omega_{\text{active}} = \bar{\Omega}/\bigcup_{i=1}^{P-1}\bar{\Omega}_i$. At this stage of the PC iteration, both the predictor and corrector equations have the same initial condition and are defined only over $\bar{\Omega}_P$. Once again the fine-grid corrector approximation is taken to be the solution in $\bar{\Omega}_P$. Thus, after at most P iterations, this improved algorithm converges with accuracy determined by the corrector approximations. A graphic overview of how the active time domain evolves as the method progresses is shown in Figure 3.

```
Algorithm: Improved PC Algorithm
• $\Omega = \bigcup_{i=1}^{P}\Omega_i$, $\Omega_i = \bigcup_{j=1}^{p}\Omega_{i(j)}$.
• $g = N^{[n]}$, $S^1 = 0$.
• For $k = 1, \ldots$
    – Solve sequentially at processor root for $i = k, \ldots, P$, where $S^{[n],k}_{|\bar{\Omega}_i}$ is
      given in (4.9), (4.10) or (4.11)
(4.17)    $\mathcal{A}^{[n],k}_{|\bar{\Omega}_i}\Delta u^{[n],k} = f^{[n],k}_{|\bar{\Omega}_i} + S^{[n],k}_{|\bar{\Omega}_i}$,
          $u^{[n],k}_{|\Gamma_i} = u^{[n],k}_{|\Gamma_{i-1}} + \Delta u^{[n],k}_{|\bar{\Omega}_{i-1}}$,   $i > k$,
          $u^{[n],k}_{|\Gamma} = g$,   $i = k = 1$,
          $u^{[n],k}_{|\Gamma_i} = u^{[n],k-1}_{i-1|\Gamma_i}$,   $i = k > 1$,
    – At each $i^{\text{th}}$-processor with $g_i = u^{[n],k}_{|\Gamma_i}$, $i = k, \ldots, P$, solve sequentially
      for $j = 1, \ldots, p$
(4.18)    $\mathcal{A}^{[n],k}_{i|\bar{\Omega}_{i(j)}}\Delta u^{[n],k}_i = f^{[n],k}_{i|\bar{\Omega}_{i(j)}}$,
          $u^{[n],k}_{i|\Gamma_{i(j)}} = u^{[n],k}_{i|\Gamma_{i(j-1)}} + \Delta u^{[n],k}_{i|\bar{\Omega}_{i(j-1)}}$,   $j > 1$,
          $u^{[n],k}_{i|\Gamma_{i(j)}} = g_i$,   $j = 1$,
    – Modify the active time domain
          $\Omega \leftarrow \bar{\Omega}/\bar{\Omega}_k$,
  If $\|u^{[n],k}_{P|\Gamma_{P+1}} - u^{[n],k}_{|\Gamma_{P+1}}\| \leq$ tol or $k = P$, stop.
```

Remark. The goal of our two-level PC scheme is to reach convergence in far fewer than P iterations, otherwise the parallel fine-grid corrector procedure would have benefited little, and this PC iteration would be forward substitution. In terms of matrix system (3.1), this improved PC iteration is an "iterative forward substitution" method for solving a block lower bidiagonal system. Our numerical experiments demonstrate that fewer than P iterations are needed.

4.3. **Linearized PC algorithm.** So far, our PC scheme has been developed using a coarse substep march to generate the interface boundary conditions. It is very tentative to derive these interface conditions by linearly interpolating $u^{[n],k}$ and $u^{[n+1],k}$. This would eliminate the need to substep on the coarse temporal grid, and thus, reduce the computational cost. Such an interpolation approach would be suitable for linear and weakly nonlinear problems. However, for highly nonlinear problems, because linear interpolation may poorly approximate the nonlinear nature of the system, the required number of iterations for this "linearized" PC scheme will generally be more than that for the scheme of Section 4.2, though still within P iterations. Nevertheless, at the kth iteration of this PC scheme, the linearly interpolated interface conditions are

$$g_i = \frac{P-i+1}{P-k+1} u^{[n],k}_{|\Gamma} + \frac{i-k}{P-k+1} u^{[n],k}_{|\Gamma_{P+1}}, \quad i = k, \ldots, P,$$

where Γ is the left boundary of the active time domain, and the modified corrected predictor equations are

$$\mathcal{A}^{[n],k}_{|\bar\Omega} \Delta u^{[n],k} = f^{[n],k}_{|\bar\Omega} + \mathcal{A}^{[n],k-1}_{|\bar\Omega} \left(u^{[n],k-1}_{|\Gamma_{P+1}} - u^{[n],k-1}_{P|\Gamma_{P+1}}\right), \tag{4.19}$$

$$\mathcal{A}^{[n],k}_{|\bar\Omega} \Delta u^{[n],k} = f^{[n],k}_{|\bar\Omega} + \sum_{j=1}^{k-1} \mathcal{A}^{[n],j}_{|\bar\Omega} \left(u^{[n],j}_{|\Gamma_{P+1}} - u^{[n],j}_{P|\Gamma_{P+1}}\right), \tag{4.20}$$

$$\mathcal{A}^{[n],k}_{|\bar\Omega} \Delta u^{[n],k} = f^{[n],k}_{|\bar\Omega} + \mathcal{A}^{[n],k}_{|\bar\Omega} \sum_{j=1}^{k-1} \left(u^{[n],j}_{|\Gamma_{P+1}} - u^{[n],j}_{P|\Gamma_{P+1}}\right). \tag{4.21}$$

Remark. Although the interface conditions and correction terms involve linearly interpolated values, at each Newton iteration, the $\mathcal{A}^{[n],k}_{|\bar\Omega}$'s must be updated, i.e., they are not linearized.

Algorithm: Linearized PC Algorithm

- $\Omega = \bigcup_{i=1}^{P} \Omega_i$, $\Omega_i = \bigcup_{j=1}^{p} \Omega_{i(j)}$.
- $g = N^{[n]}$, $S^1 = 0$,
- For $k = 1, \ldots,$
  - Solve sequentially at processor root over the active domain Ω, where $S^{[n],k}$ is given by (4.19), (4.20), or (4.21):

$$\begin{aligned} \mathcal{A}^{[n],k}_{|\bar\Omega} \Delta u^{[n],k} &= f^{[n],k}_{|\bar\Omega} + S^{[n],k}_{|\bar\Omega}, \\ u^{[n],k}_{|\Gamma} &= g\,, \qquad k = 1, \\ u^{[n],k}_{|\Gamma} &= u^{[n],k-1}_{k-1|\Gamma_k}\,, \qquad k > 1, \end{aligned} \tag{4.22}$$

  - At each ith-processor with $g_i = \frac{P-i+1}{P-k+1} u^{[n],k}_{|\Gamma} + \frac{i-k}{P-k+1} u^{[n],k}_{|\Gamma_{P+1}}$,’ $i = k, \ldots, P$, solve sequentially for $j = 1, \ldots, p$

$$\begin{aligned} \mathcal{A}^{[n],k}_{i|\bar\Omega_{i(j)}} \Delta u^{[n],k}_{i} &= f^{[n],k}_{i|\bar\Omega_{i(j)}}, \\ u^{[n],k}_{i|\Gamma_{i(j)}} &= u^{[n],k}_{i|\Gamma_{i(j-1)}} + \Delta u^{[n],k}_{i|\bar\Omega_{i(j-1)}}\,, \quad j > 1, \\ u^{[n],k}_{i|\Gamma_{i(j)}} &= g_i\,, \quad j = 1. \end{aligned} \tag{4.23}$$

  - Modify the active time domain

$$\Omega \leftarrow \bar\Omega / \bar\Omega_k.$$

If $\|u^{[n],k}_{P|\Gamma_{P+1}} - u^{[n],k}_{|\Gamma_{P+1}}\| \le \text{tol}$ or $k = P$, stop.

4.4. **Two-level computational efficiency.** A deficiency in the two-level method of Section 4.2 is the bottleneck in (4.17). For each PC iteration, the root processor must complete its sequential march before the child processors can start, resulting in processor idleness and communication traffic. The bottleneck in processor communication can be mitigated by looping (4.17) and (4.18) differently:

```
Algorithm: Reduced Bottleneck Loop
For $k = 1, \dots$
  For $i = k, \dots, P-1$,
      – Solve at processor root
(4.24)    $\begin{aligned} \mathcal{A}^{[n],k}_{|\bar{\Omega}_i} \Delta u^{[n],k} &= f^{[n],k}_{|\bar{\Omega}_i} + S^{[n],k}_{|\bar{\Omega}_i}, \\ u^{[n],k}_{|\Gamma_i} &= u^{[n],k}_{|\Gamma_{i-1}} + \Delta u^{[n],k}_{|\bar{\Omega}_{i-1}}, \qquad i > k, \\ u^{[n],k}_{|\Gamma_i} &= u^{[n],k-1}_{i-1|\Gamma_i}, \qquad i = k. \end{aligned}$
      – At processor $(i+1)$, with $g_{i+1} = u^{[n],k}_{|\Gamma_{i+1}}$, begin solving for
        $j = 1, \dots, p$
(4.25)    $\begin{aligned} \mathcal{A}^{[n],k}_{i+1|\bar{\Omega}_{(i+1)(j)}} \Delta u^{[n],k}_{i+1} &= f^{[n],k}_{i+1|\bar{\Omega}_{(i+1)(j)}}, \\ u^{[n],k}_{i+1|\Gamma_{(i+1)(j)}} &= u^{[n],k}_{i+1|\Gamma_{(i+1)(j-1)}} + \Delta u^{[n],k}_{i+1|\bar{\Omega}_{(i+1)(j-1)}}, \quad j > 1, \\ u^{[n],k}_{i+1|\Gamma_{(i+1)(j)}} &= g_{i+1}, \quad j = 1. \end{aligned}$
    Solve subdomain $i = P$ on the root processor
(4.26)    $\begin{aligned} \mathcal{A}^{[n],k}_{|\bar{\Omega}_P} \Delta u^{[n],k} &= f^{[n],k}_{|\bar{\Omega}_P} + S^{[n],k}_{|\bar{\Omega}_P}, \\ u^{[n],k}_{|\Gamma_P} &= u^{[n],k}_{|\Gamma_{P-1}} + \Delta u^{[n],k}_{|\bar{\Omega}_{P-1}}, \qquad i > k. \end{aligned}$
```

Now once the root processor has finished a substep, it can immediately communicate the computed interface boundary condition to the appropriate child processor, and, while this communication is occurring, the root processor can begin its next substep. As for processor idleness, it can be reduced by starting the next predictor cycle on the child processors that have completed their marches before the root processor has completed its march. This unfortunately involves complicated processor scheduling.

Suppressing this complicated scheduling, the efficiency of the parallel two-level method can be obtained by counting specific substep solves on all processors, and counting the number of blocking communications (i.e., communications that *delay* the start-up of the last running processor[1]). In particular, substeps that are solved concurrently on different processors are counted only once. This is illustrated in Figure 4, which shows the counts for three different subdomain partitionings of a fine grid with $G = 16$, e.g., for the top partitioning, since the first subdomain of both levels are solved concurrently, they are numbered 1, since the second substep of both levels and the ninth substep of the fine level are all solved concurrently, they are numbered 2, etc. Table 1 summarizes the efficiency for several time-marching schemes. First, a serial march requires no communication and only G substeps since the predictor is not needed. For the two-level method with loops (4.17)–(4.18), in the kth PC iteration, $(P - k + 1)$ substeps are needed for the predictor and p substeps are needed for the corrector. Moreover, before the last processor can start

[1] The root processor can communicate each interface value using point-to-point communications, or it can communicate all the interface values at once using a scatter communication. For large sets of data, both types of communication complete their tasks in about the same time.

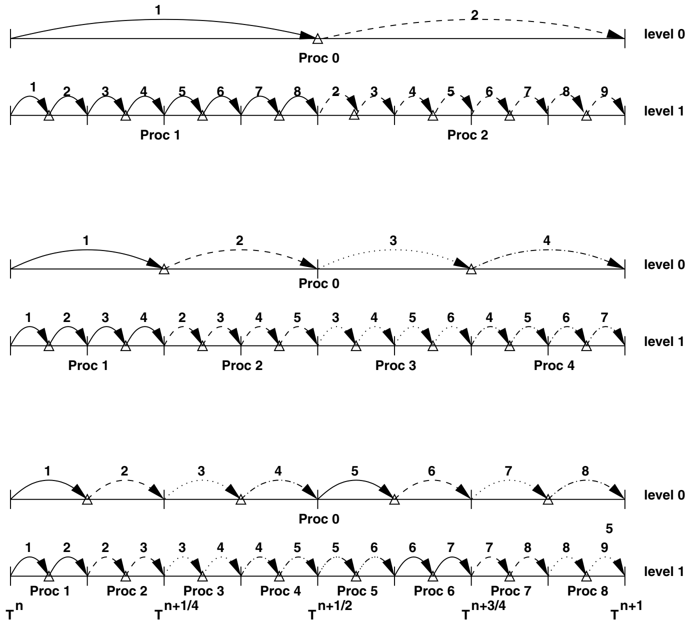

FIGURE 4. Two-level Method. Counting the substep solves in the first PC iteration for three different subdomain partitionings: $G = 16$, $(P, p) = (2, 8), (4, 4), (8, 2)$. Substep solves that occur simultaneously on different processors are counted only once.

its march, it must wait for $(P - k)$ communications. Hence, for l iterations, the efficiency of this scheme is

$$\sum_{k=1}^{l}[(P-k+1+p)\text{ solves} + (P-k)\text{ comms}]$$
$$= l\left(P+p+1-\frac{(l+1)}{2}\right)\text{solves} + \ l\left(P-\frac{l+1}{2}\right)\text{comms.}$$

Finally, using the above counting convention, in the kth PC iteration, the two-level method with loop (4.24)–(4.25) requires $(P-k+p)$ solves. Also, since the predictor march continues while communication is occurring, only the last communication is blocking. Thus, for l iterations, the efficiency of this scheme is

$$\sum_{k=1}^{l}[(P-k+p)\text{ solves} + 1\text{ comms}] = l\left(P+p-\frac{(l+1)}{2}\right)\text{solves} + l\text{ comms.}$$

TABLE 1. Efficiency for the kth iteration. The serial method performs only 1 iteration, a forward substitution.

Method	SubSteps	Comms	Efficiency
serial	G	none	G substeps
Loops (4.17)–(4.18)	$(P-k+1+p)$	$(P-k)$	$(P-k+1+p)$ substeps & $(P-k)$ comms
Loop (4.24)–(4.25)	$(P-k+p)$	$(P-k)$	$(P-k+p)$ substeps & 1 comm

Note that irrespective of the subdomain partitioning, the two-level method with loop (4.24)–(4.25) requires just 1 blocking communication. Thus, its efficiency at the kth iteration is optimized by choosing a partition that minimizes the substep count. That is, its efficiency is optimized by minimizing

$$f(P) = P - k + p = P - k + \frac{G}{P}.$$

The minimum of f occurs at $P = \sqrt{G}$, and the minimal value is $(2\sqrt{G} - k)$. Hence, this two-level scheme's optimal efficiency for l iterations is

$$l\left(2\sqrt{G} - \frac{(l+1)}{2}\right) \text{solves} + l \text{ comms.}$$

But obviously this choice for P affects the value of l, the required number of PC iterations to reach convergence.

5. Multilevel extensions

A remaining issue with the two-level method is processor idleness. This problem can be further embellished when the optimal subdomain partitioning is chosen, since then only $(\sqrt{G} + 1)$ of the total number of processors are used. Hence, there is an incentive for having many small subdomains. In this section, we develop a multilevel extension of the two-level method that uses just such partitioning yet keeps the substep count low.

To achieve better load-balance when many small subdomains are used, an initial consideration is to allocate more processors to the predictor procedure. But this returns us to the original problem of time parallelization—processors cannot start until boundary conditions are available. Undoubtedly what needs to be achieved is faster generation of interface boundary conditions. One way this is accomplished is by recursively applying the matrix method of (3.5) to its own subblocks. To describe this recursion, it is helpful to illustrate the processor scheduling using trees and grids. First, the processor scheduling for the two-level method is given by the tree shown in Figure 5.

For the two-level scheme with loop (4.24)–(4.25), processor 0 immediately fans out to a branch processor once the necessary interface boundary condition has been computed. The processor scheduling for the recursive scheme, on the other hand, leads to a tree that spins off pairs of branches at each processor node, as shown in Figure 6. After only a few levels down the root node, a majority of the processors will be active. Each pairwise branching corresponds to a newly computed interface boundary condition at an intermediate time level in (T^n, T^{n+1}). This interface boundary condition creates a processor distribution—the subdomain to the left of the intermediate time level is substepped on one processor, while the subdomain to

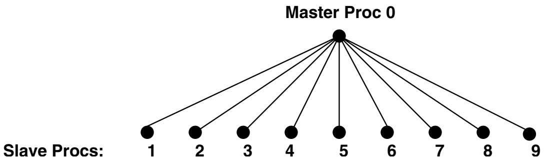

FIGURE 5. Processor scheduling for two-level method, fan-out from left to right.

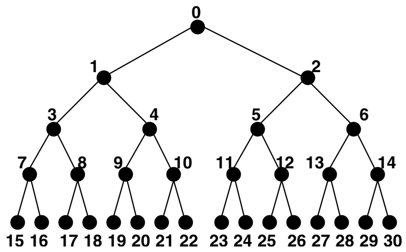

FIGURE 6. Processor scheduling for the multilevel method.

the right is substepped on another processor. Figure 7 elaborates this branching. In this diagram, there are three coarsenings of the bottom target grid. For the two-level method, the bottom grid and any one of the other three grids, depending on the subdomain partitioning, are used. For the recursive multilevel scheme, the collection of all the grids forms the multilevel grid hierarchy. Corresponding to this grid hierarchy is a processor distribution. Each processor takes a subdomain composed of two substeps. For example, Proc 0 takes the top grid with the whole time domain $[T^n, T^{n+1}]$ as one subdomain with two substeps, Proc 1 takes the left substep of Proc 0 as a subdomain and further divides it into two finer substeps, etc. To relate this figure's layout to the pairwise branching of Figure 6, consider the top grid. Here, after Proc 0 has marched one substep, it has computed an interface boundary condition that permits Proc 2 to start its march. Proc 1 also can start, though it could have started concurrently with Proc 0. Taking the delayed start-up of Proc 1, after Proc 0's first substep, a pairwise branching fires up Proc 1 and Proc 2. Next, on the second grid, once Proc 1 and Proc 2 each has completed its first substep, they pairwise branch off to Proc 3 and Proc 4, and Proc 5 and Proc 6, respectively. This branching continues down to the third grid where the next branching finally leads to the target temporal grid. The end result is a multilevel partitioning composed of many small subdomains.

This branching procedure spawns off a variety of choices for the PC algorithm. One option already alluded to is the immediate/delayed start-up of processors, with this option arising whenever a time level is common to several grids. Using immediate start-up, some of the processors alloted to the finer grids will be activated rather early in the branching cycle. For example, in Figure 7, once Proc 0 reaches $T^{n+\frac{1}{2}}$, in addition to Proc 2, Procs 5 and 11 can also start marching. Alternatively, using delayed start-up, Procs 5 and 11 respectively start only after Procs 1 and 4 have completed their *short* marches. Although delayed start-up is less efficient, an advantage it has is a construction of consistently accurate interface values on a grid level. Moreover, since delayed start-up generates several different approximations for a common time level, Richardson extrapolation can be applied to obtain better approximations. In particular, delayed start-up can be used on a minimal number of coarser grids, just enough to generate the necessary number of approximations for an application of Richardson extrapolation. The extrapolated interface value may be sufficiently accurate for use on all the remaining finer levels, i.e., immediate start-up for these levels.

Up to now, a PC iteration with this branching cycle has not been described. There are several options on how this iteration can be structured. One approach is

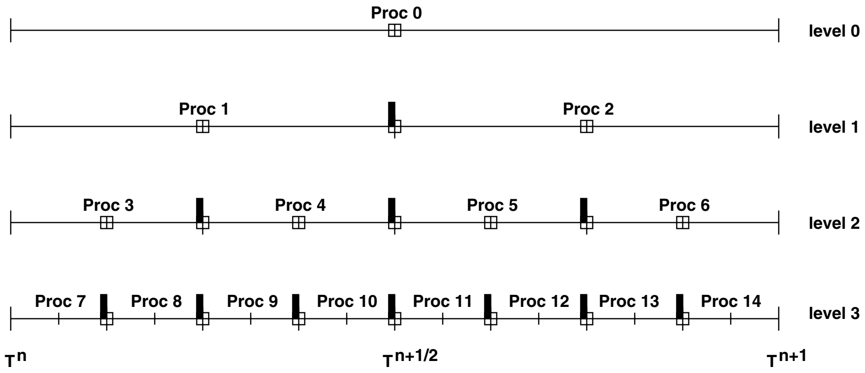

FIGURE 7. Grid layout of processor scheduling for the multilevel algorithm. Dark clips are the processor boundaries, square clips are time levels where several approximations of different grid resolutions exist. Richardson extrapolation can be performed on these approximations to generate more accurate solutions at these time levels.

to have the first PC iteration be the branching cycle, and then all future iterations be the two-level scheme applied to the two finest grid levels. The purpose of the branching cycle would be to efficiently form an accurate initial approximation over the whole time domain, which then can reduce the total number of PC iterations to reach convergence. However, although more processors will be alloted to the predictor procedure of the two-level module, the sequential march of the predictor nullifies the purpose of this allotment. Hence, this leads to the next option for the PC iteration: branch cycle for each PC iteration. After the corrector procedure of the finest grid, the next iteration begins again on the coarsest grid and branches down to the finest grid. The active domain of this cycle is modified as in the two-level method. But because the eliminated subdomains may cover only a fraction of a substep on some of the coarser grids, only a portion of a substep will be removed on these grids. Figure 8 illustrates this after two PC iterations. For iterate $k = 1$, a substep will be removed on level 2, a half substep on level 1, and a quarter substep on level 0. For iterate $k = 2$, totals of two substeps will be removed from level 2, a whole substep on level 1, and an half substep on level 0.

Two other options for the PC iteration are in the correction term and the stopping criterion. For the correction term, at right subdomain boundaries that are positioned at mutual time levels of several grids, one can continue to use the correction term

$$\mathcal{A}^{[n],k-1}_{|\bar{\Omega}_i}(u^{[n],k-1}_{|\Gamma_{i+1}} - u^{[n],k-1}_{i|\Gamma_{i+1}}),$$

or one can replace $u^{[n],k-1}_{i|\Gamma_{i+1}}$ with a Richardson extrapolated value. For the stopping criterion, one can choose a measure based on the difference between the solutions at time T^{n+1} on the coarsest and finest levels, on the two finest levels, or on the finest level and a Richardson extrapolated solution. Finally, we examine the efficiency of this multilevel method. Figure 9 shows the number of solves required in the first PC iteration for the delayed start-up option. One can see that for two consecutive levels, the numbering on the coarser level repeats itself on the left half

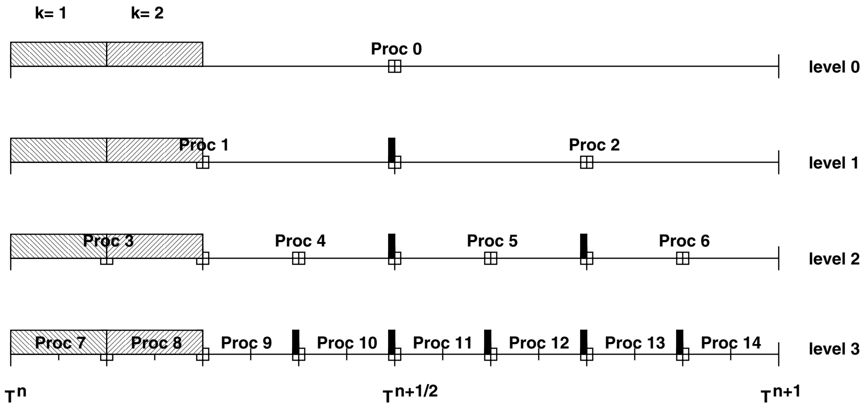

FIGURE 8. Grid layout of the multilevel improved PC iteration after the initial 2 PC iterations. For the second iteration, labelled $k = 1$, the left-hashed time interval is removed from the grid hierarchy. For the third iteration, the next hashed time interval is removed.

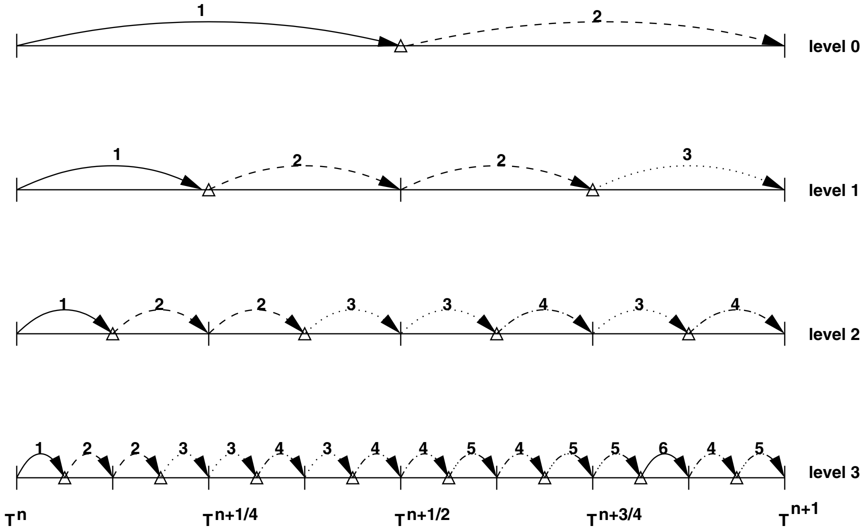

FIGURE 9. Multilevel Method. Counting the substep solves in the initial PC iteration. Substep solves that occur simultaneously on different processors are counted only once.

of the finer level. Thus, the last solve occurs on the right half of the finest level. In particular, a little reflection reveals that the last solve will occur in the second-to-last subdomain of the finest level, e.g., in Figure 9, it occurs in subdomain 7. This transpires because of the delay resulting from generating an accurate interface value for this subdomain. Assuming the number of subdomains on the target grid to be $2^n, n > 1$ (other powers can be used but different tree structures will be formed), this last solve will be numbered $2n$. Thus, since the processor performing this solve

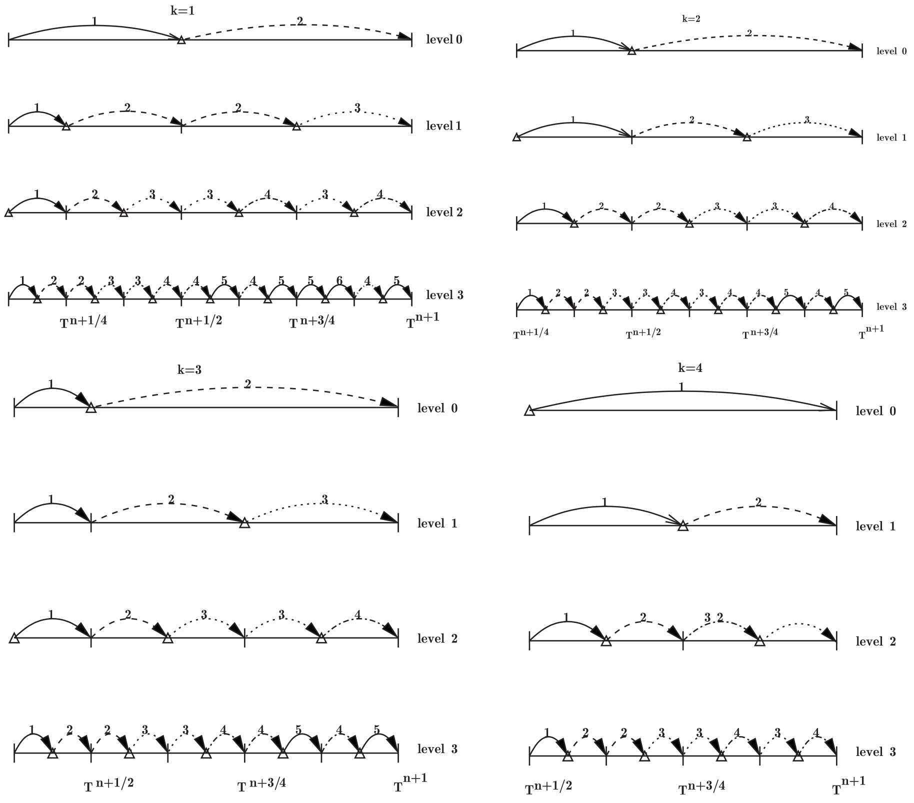

Figure 10. Counting the substep solves in the second, third, fourth, and fifth ($k = 1, 2, 3, 4$) PC iterations with the active domain decreasing. The reduction in the number of solves is minor.

has to wait for only 1 communication, the efficiency for the first PC iteration is

$$2n \text{ solves} + 1 \text{ comm.}$$

For future iterations, the number of solves only gradually decreases as the active domain is reduced (see Figure 10). Hence, for l PC iterations, an upper bound for the efficiency is

$$2nl \text{ solves} + l \text{ comms.} \tag{5.1}$$

With immediate start-up, the efficiency may improve. Figure 11 shows the solve count for an immediate start-up on the same grid hierarchy used in Figure 9. One can see that the last solve occurs on the last subdomain of the finest grid, and with a total of $2^n, n > 1$, subdomains on this grid, this solve is numbered $(n + 2)$. Also, the processor computing this solve must wait for only 1 communication, so that the efficiency for the first PC iteration is

$$(n + 2) \text{ solves} + 1 \text{ comms.}$$

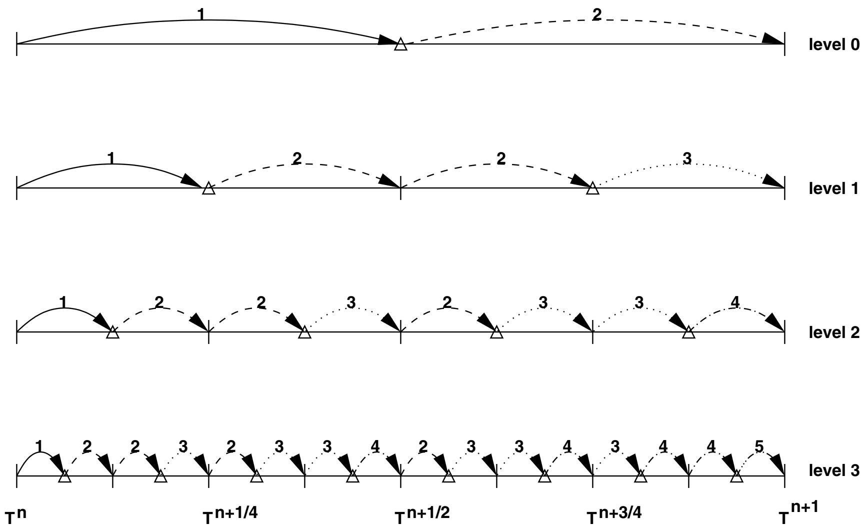

Figure 11. Counting the substep solves in the initial PC iteration for immediate start-up.

But again the number of solves slowly decreases as the active domain is reduced. Thus, for l PC iterations, the efficiency is bounded above by

$$l\ (n+2) \text{ solves} + l \text{ comms.} \tag{5.2}$$

Although this bound is smaller than the bound for delayed start-up, the overall number of iterations to reach convergence for this scheme generally will be larger.

Comparing the efficiencies of the two-level and multilevel methods, it initially appears that the latter method dramatically improves the efficiency. For example, for a target grid with $G = 1024$ substeps, the cost for l iterations for the optimal two-level method is roughly

$$l\left(64 - \frac{(l+1)}{2}\right) \text{solves} + l \text{ comms,}$$

while the costs for delay and immediate start-up are respectively

$$18l \text{ solves} + l \text{ comms}$$

and

$$11l \text{ solves} + l \text{ comms.}$$

However, the two-level method is computed on only $(\sqrt{G}+1) = 33$ processors, whereas the multilevel methods are computed on 1023 processors. Thus, these multilevel extensions do not attain processor scalability, i.e., since there are roughly 31 times the number of processors used in the multilevel methods, but the speed-up for l iterations is roughly 4-6 times. Of course, this scalability would improve if more iterations were needed in the two-level method than in the multilevel methods.

5.1. **Multilevel FAS reformulation.** This multilevel branch scheme may require more iterations than one may anticipate. Since the coarser levels generally involve much larger time-steps than the target fine grid, the correction terms given in (4.9)–(4.11) may be ineffective in generating accurate interface values. On the other hand, correction term (4.12) can generate better interface values. More important, using this correction term, the multilevel branch scheme becomes a multigrid FAS iteration for the time march. Now the task of the coarse levels is not just to obtain accurate initial values for the finer levels, but also to eliminate slow time error modes of the target fine grid approximation [13]. This can substantially improve the convergence rate of the branch scheme (see Table 2).

6. Numerical examples

In this section, we demonstrate the performance of the multilevel and two-level methods on a linear advection equation and a reservoir simulation, respectively. The goal of the advection problem is to exhibit the performance of the multilevel scheme on a massively parallel computer system; the goal of the reservoir simulation is to exhibit the applicability of our two-level method on a realistic, highly nonlinear problem.

6.1. **Linear advection equation: Multilevel schemes.**

We consider the following linear advection problem defined on the spatial domain $D = (0, 0.5)^3$ with inflow boundary $\partial D_{\text{inflow}} = \{(x, y, z) : xyz = 0\}$:

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z} &= 1 \quad (x, y, z) \in D, \\ u(x, y, z, t = 0) &= 1 \quad (x, y, z) \in D, \\ u(x, y, z, t) &= 0 \quad (x, y, z) \in \partial D_{\text{inflow}}. \end{aligned} \tag{6.1}$$

The computational mesh is uniform for both space and time, with the same spatial grid of 50 points in each direction (i.e., $\Delta x = \Delta y = \Delta z = 0.01$, a total of 125,000 spatial points) at every time-step, and with the time-step on coarsest level 0 being $\Delta t_0 = 0.01$ and on level i being $\Delta t_i = \frac{\Delta t_{i-1}}{2}, i > 0$. The discretization is finite volume in space and backward Euler in time. The operators $\mathcal{A}^{[n],k}_{|\bar{\Omega}_i}$ are nonsymmetric, and hence, at each time-step, the linear systems are solved with GMRES preconditioned with a Schaffer multigrid V cycle [12]. Lastly, the stopping criterion for the multilevel PC iteration is

$$\frac{\| u^{\text{finest}}_{P|\Gamma_{P+1}} - u^{\text{coarsest}}_{|\Gamma_{P+1}} \|}{\| u^{\text{coarsest}}_{|\Gamma_{P+1}} \|} \leq [\Delta t_0 - \Delta t_{\text{finest}}].$$

Table 2 shows the results for runs made on a parallel computer system with a total of 2096 Intel Xeon processors (2.4 GHz). For branch cycles with the correction

Table 2. Performance of the multilevel time-marching scheme on a linear advection equation. Only the FAS cycle was used on the 1023 processors run because the convergence rate of the branch cycle can already be observed to depend on the number of target fine time-steps.

# Procs	# Target Fine Subdomains	Branch Cycle	FAS Cycle
63	32	19	2
127	64	37	2
255	128	73	2
511	256	>100	2
1023	512	-	2

term given in (4.9), the number of PC iterations does not scale with respect to the number of finest level time-steps. Moreover, although the number of PC iterations is less than the number of finest level subdomains, the total computational cost for this method is unsatisfactory since the cost per branch cycle for the target time grids used in these experiments is roughly 10 solves. However, using the FAS correction term, which transforms the branch cycle to a FAS cycle, the number of PC iterations does scale with respect to the number of finest level time-steps. Moreover, the small number of iterations makes the FAS cycle extremely computationally efficient.

6.2. **Fluid flow simulation: Two-level method.** We now consider a realistic nonlinear problem. The two-level methods of Sections 4.2 and 4.3 are implemented for the molar mass equations in `Athena`, our simulator for multiphase, multicomponent, fluid flow in porous media. These experiments were conducted on a Linux cluster with PIII processors, where the interface values are communicated to the child processors only after the predictor has completed its full time march. The correction terms for the predictor equations, $S^{[n],k}_{|\bar{\Omega}_i}$, are given in (4.11).

Our experiments were carried out on a geological domain of size $1000\,\mathrm{m} \times 100\,\mathrm{m} \times 70\,\mathrm{m}$, with its upper end points located at a depth of 50 m from the earth's surface. The vertical topography of the domain consists of four layers of rock: shale, sandstone, shale, sandstone. Hence, the mathematical equations have discontinuous coefficients since the lithology for sandstone has a porosity of $\phi = 0.5$ and a permeability of

$$K_x = 500\,\mathrm{mD},\ \mathrm{K_y} = 500\,\mathrm{mD},\ \mathrm{K_z} = 500\,\mathrm{mD},$$

whereas the lithology for shale has a porosity of $\phi = 0.5$ and a permeability of

$$K_x = 5 \cdot 10^{-6}\,\mathrm{mD},\ \mathrm{K_y} = 5 \cdot 10^{-6}\,\mathrm{mD},\ \mathrm{K_z} = 5 \cdot 10^{-6}\,\mathrm{mD}.$$

Finally, the chosen boundary conditions are a left inflow flux of $5 \cdot 10^{-5}\,\mathrm{mol/m^2s}$ for oil and gas, a right outflow flux of $6.5 \cdot 10^{-4}\,\mathrm{mol/m^2s}$ for water, a top temperature value of 450 K, and a bottom temperature value of 460 K. Figures 12 and 13 display `Athena`'s output results for a simulation of a 100 years. These figures also illustrate the computational grid used in our experiments.

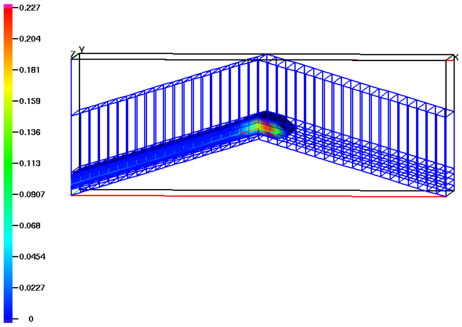

FIGURE 12. Gas saturation.

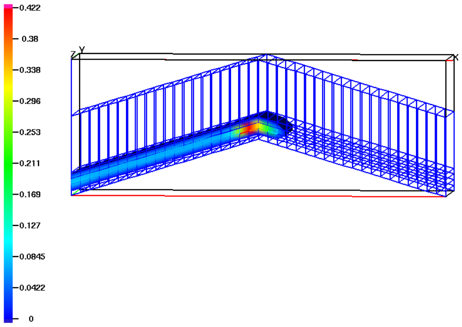

FIGURE 13. Oil saturation.

6.3. **Improved parallel PC algorithm in Athena.**

We apply the improved parallel PC algorithm of Section 4.2 to the molar mass equations.

6.3.1. *Computational results: Scalability.* In this experiment, we are interested in the processor scalability of our parallel two-level algorithm, although we remark that usually only 2 PC iterations were needed to attain convergence in our experiments. By varying the number of processors, we explore the speedup of a parallel run over a serial run. Ideally, as more processors are used, the wall-clock time is expected to decrease linearly as a function of the number of processors used. However, because of the sequentiality of the predictor march, this ideal situation is observed only when the computational costs dominate the communication costs up to the optimal subdomain partitioning.

Our implementation of the two-level method is based on an MPI master-slave processor structure, where the number of subdomains equals the number of slave processors. In our experiments, we vary the number of subdomains, or slave processors, while keeping the size of the target time-domain fixed to $G = 16$ time-steps. Therefore, successively doubling the number of subdomains i times, $i = 0, \ldots, 4$, the number of substeps in each subdomain successively halves to $p = 2^{4-i}$. Since a subdomain resides on one processor, increasing the number of subdomains decreases the amount of computation per slave processor, but increases the number of communications and the sequential computation load of the predictor step.

Results for our experiments are shown in Figures 14 and 15. On the left plots of these figures, the slave processor run-time is plotted against i, where 2^i is the number of subdomains. This slave processor run-time is the average of the times for all slave processors. As can be observed, the computational time decreases (monotonic decreasing curve) as the number of subdomains increases, but at the same time, the collective broadcasting time increases (monotonic increasing curve). Note that the computational time decreases more than expected, e.g., for $i = 1$, one would expect the computational time only to halve, whereas the actual time is quartered. There are two possibilities for this discrepancy. First, the number of

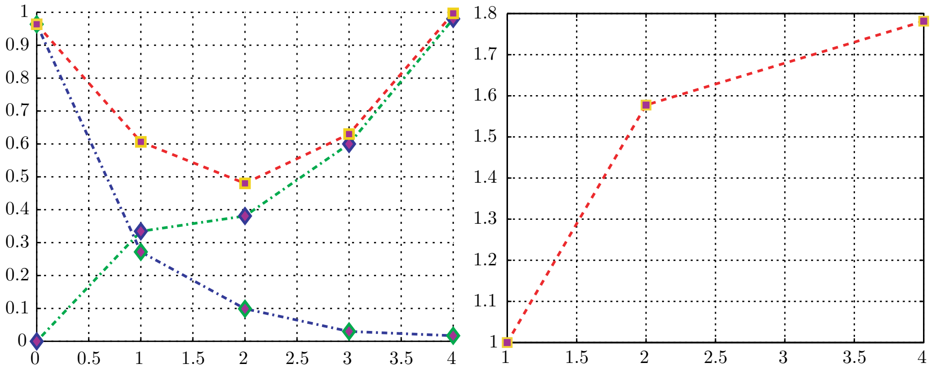

FIGURE 14. Spatial grid of 200 cells. Run time vs. i, where 2^i, $i = 0, \ldots, 4$, is the number of subdomains. On the top, timing for communication (.- increasing), calculation (.- decreasing), and the sum of both (- -) for the slave processors. On the bottom, the speedup for the full slave and master run.

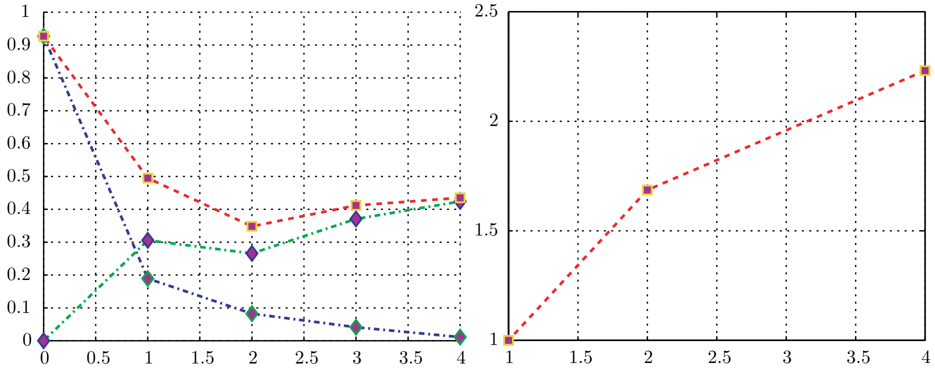

FIGURE 15. Spatial grid of 800 cells. Run time vs. i, where 2^i, $i = 0, \ldots, 4$ is the number of subdomains. On the left, timing for communication (.- increasing), calculation (.- decreasing), and the sum of both (- -) for the slave processors. On the right, the speedup for the full slave and master run.

Newton-GMRES iterations is less in the parallel runs, and second, the number of Jacobian matrix formations are clearly more in a serial run.

Now, the relevant timings are the sums of the computation and communication times. These are plotted in the square-marked curve, which shows that the method is competitive up to a degree of parallelism, i.e., to a balance between the computational and communication costs. For the small number of spatial points used in this experiment, the communication costs are relatively high.

So far, we have considered only the timings for the corrector solves. To observe the overall speedup of the two-level method, both the predictor (master processor) and corrector (slave processors) timings need to be added. This is displayed on the right-hand graph of Figure 14. There we observe that the best speedup occurs in the optimal subdomain partitioning with $P = \sqrt{16} = 2^2$.

6.3.2. *Computational results: Performance.* In this subsection, we explore the scaling when the number of spatial points is increased. As previously indicated, when the number of spatial points increases, the computational costs dominate the communication costs. This is indeed the case as can be seen from the left-hand graph of Figure 15. Hence, we have an overall improvement of the processor scalability. On the right-hand graph, we see again that the best speedup occurs with the optimal subdomain partitioning.

6.4. **Linearized parallel PC algorithm in `Athena`.** In this subsection, we repeat the experiments of Subsection 6.3.1 using the linearized parallel PC algorithm. Scalability results are plotted in Figure 16, showing better parallel speedup since there are less communications—only the last corrector solution needs to be communicated to the root processor. Also, the overall speedup is better because only one time-step is needed in the corrector march. However, this linearized method is inefficient for highly nonlinear systems because the initial values for the corrector are so badly approximated that the required number of PC iterations to reach convergence suffers an unacceptable increment.

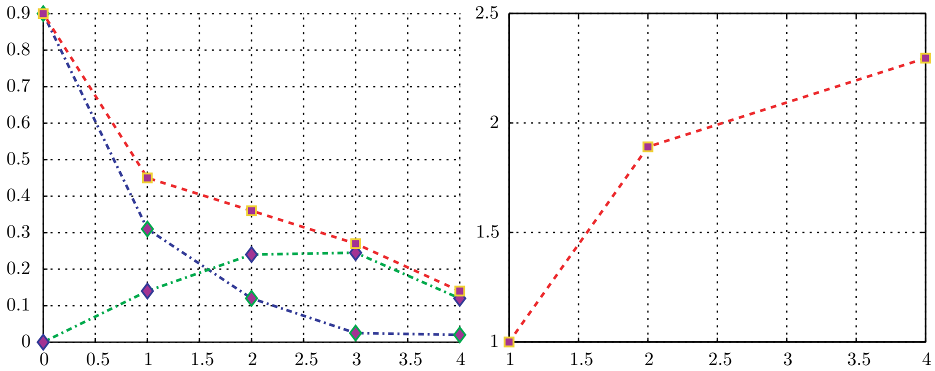

Figure 16. Spatial grid of 200 cells. Run time vs. i, where 2^i, $i = 0, \dots, 4$ is the number of subdomains. On the left, timing for communication (.- increasing), calculation (.- decreasing), and the sum of both (- -) for the slave processors. On the right, the speedup for the full slave and master run.

7. Conclusions

In this paper, several Parareal-type schemes for time parallelization were developed. We introduced several simple predictor correction terms, and a progressive domain reduction procedure that simulated forward substitution. We also extended our two-level method to multilevels. In particular, using correction term (4.12), our multilevel branching cycle converts to a FAS time marching iteration, which has better convergence properties.

The performance of these methods was demonstrated through a model linear advection problem and a realistic reservoir simulation. Results of the advection problem exhibited the performance of the multilevel methods on a large number of processors. The FAS iteration displayed superior performance. Results of the reservoir model demonstrated the applicability of the parallel time marching scheme to realistic nonlinear problems. Although only a small number of processors were used, our two-level scheme displayed encouraging performance for a highly nonlinear problem. However, the linearized two-level PC iteration did not display similar encouraging performance because of the equations' high nonlinearity.

References

1. L. Baffico, S. Bernard, Y. Maday, G. Turinici, G. Zérah, *Parallel in time molecular dynamics simulations*, 2002.
2. G. Bal, Y. Maday, *A parareal time discretization for non-linear PDE's with application to the pricing of an American put*, Lect. Notes Comput. Sci. Eng., vol. 23, Springer, Berlin, 2002. MR1962689
3. S. Bellavia, B. Morini, *A globally convergent Newton-GMRES subspace method for systems of nonlinear equations*, SIAM J. Sci. Comput., **23**, No. 3, SIAM, 2001, pp. 940–960. MR1860971 (2002h:65077)
4. P. N. Brown, Y. Saad, *Hybrid Krylov Methods for nonlinear systems of equations*, SIAM J. Sci. Stat. Comput., **11**, No. 3, SIAM, 1990, pp. 450–481. MR1047206 (91e:65069)
5. G. E. Fladmark, *Secondary Oil Migration. Mathematical and numerical modelling in SOM simulator*, In Norsk Hydro (eds.), Research centre Bergen, **R-077857**, 1997.

6. M. J. Gander, H. Zhao, *Overlapping Schwarz Waveform Relaxation for the Heat Equation in n-Dimensions*, BIT, **42**, 2002, pp. 779–795. MR1944537 (2003m:65170)
7. I. Garrido, E. Øian, M. Chaib, G. Fladmark, M. Espedal, *Implicit treatment of compositional flow*, Comput. Geosci., **8**, No. 3, Kluwer, 2004, pp. 1–19. MR2057524
8. J. L. Lions, Y. Maday, Gabriel Turinici, *Rèsolution d'EDP par un schéma en temps pararéel*, C. R. Acad. Sci. Paris, **332**, No. 1, pp. 1–6, 2001. MR1842465 (2002c:65140)
9. G. Å. Øye, H. Reme, *Parallelization of a Compositional Simulator with a Galerkin Coarse/Fine Method*, P. Amestoy and others (eds.), **1685**, pp. 586–594. Euro-Par'99, Springer-Verlag, Berlin, 1999.
10. G. Qin, H. Wang, R. E. Ewing and M. S. Espedal, *Numerical simulation of compositional fluid flow in porous media*, Z.–C. Shi Z. Chen, R. E. Ewing (eds.), Numerical treatement of multiphase flows in porous media, **552**, pp. 232–243. Springer-Verlag, Berlin, 2000. MR1876022
11. H. Reme, G.Å. Øye, *Use of local grid refinement and a Galerkin technique to study secondary migration in fractured and faulted regions*, Computing and Visualisation in Science, **2**, pp. 153–162, Springer-Verlag, Berlin, 1999.
12. S. Schaffer, *A semicoarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients* SIAM J. Sci. Comput, **20**, No. 1, pp. 228–242, 1998. MR1639126 (99d:65355)
13. U. Trottenberg, C.W. Oosterlee and A. Schuller, *Multigrid*, Academic Press, 2001. MR1807961 (2002b:65002)
14. S. Vandewalle, and G. Horton, *A space-time multigrid method for parabolic partial differential equations*, SIAM J. Sci. Comput, **16**, No. 4, pp. 848–864, 1995. MR1335894 (96d:65158)
15. P. Wesseling, *An Introduction to Multigrid Methods*, John Wiley, 1992. MR1156079 (93g:65006)

Department of Mathematics, University of Bergen, Johs. Brunsgt. 12, N-5008 Bergen, Norway
E-mail address: izaskun@mi.uib.no

CASC, Lawrence Livermore National Laboratory, Livermore, California 94551

Department of Mathematics, University of Bergen, Johs. Brunsgt. 12, N-5008 Bergen, Norway

Department of Mathematics, University of Bergen, Johs. Brunsgt. 12, N-5008 Bergen, Norway