PARALLEL IMPLEMENTATION OF MULTIPHYSICS MULTIBLOCK FORMULATIONS FOR MULTIPHASE FLOW IN SUBSURFACE*

QIN LU †, MANISH PARASHAR ‡, MAŁGORZATA PESZYŃSKA , AND MARY F. WHEELER§

Abstract. We present the design, implementation and experimental evaluation of parallel solutions that support adaptive, multiblock and multiphysics applications arising in subsurface modeling. In these applications, the underlying computational domain is subdivided into blocks (subdomains) and the most appropriate physical model is independently applied to each block. The models are coupled using numerical algorithm based on mortar spaces, which provides a rigorous formulation of the overall problem. Efficient parallel implementation of this application presents significant computational challenges. The solutions presented in this paper include (1) MACE computational engine that provides programming, dynamic data-management and mortar-based coupling support, (2) a communication substrate that provides communication contexts for model-based interactions and (3) model-sensitive load-balancing strategies. These solutions are implemented within the IPARS reservoir simulation framework, and support arbitrary numbers of blocks, models, and processors. Results presented in this paper demonstrate efficiency and parallel scalability of the implementation.

Key words. Parallel subsurface modeling, multiphase flow, multiblock, multimodel, multiphysics, mortar spaces, model sensitive load balancing.

AMS subject classifications. 65Y05 65M55 76T30 76S05

1. Introduction. Modeling of oil and gas recovery as well as the study of contamination scenarios in unsaturated zones often require large scale computations to understand the design and to optimize the output of the proposed engineering system. The overall goal of the new generation of reservoir simulators is to support realistic, high-resolution reservoir studies with a million or more grid elements, with modeling elements dynamically selected based on the current characterization and state of reservoir.

The underlying phenomena are those of multiphase flow and multicomponent transport in subsurface. Mathematical models of such phenomena are highly nonlinear, coupled, time-dependent, systems of partial differential equations which locally may be of parabolic, hyperbolic or elliptic type. Moreover, these subsurface processes frequently have a highly local spatial and temporal character which in one part can be described by fewer or different type variables than in another part. However, even though availability of multiple models has become an industrial standard and is popular in academic and other research institutions [36, 16, 44], when modeling a phenomenon occurring in a large connected domain, scientists using traditional simulation tools must select one physical model that is a superset of all the local models. Naturally, computational requirements of a physical model depend on its complexity - typically, a more complicated model requires more computational time and resources. Using the traditional approach leads therefore to inefficiency: since even if a complex model is necessary for only a small part of the domain, it will have to be used for the

* supported by NSF EIA-0120934 (ITR), EIA 0103674 (NGS) and ACI 9984357 (CAREERS), and DOE ASCI/ASAP (Caltech) PC295251, also by DOE: DE-FG03-99ER25371, DOD: PET2/ UTA01-347, and the NSF grants: SBR 9873326, ITR EIA-0121523, NPACI 10181410

†Landmark Graphics Corp.qlu@lgc.com

‡The Applied Software Systems Laboratory, Department of Electrical &Computer Engineering, Rutgers, The State University of New Jersey,parashar@caip.rutgers.edu

§Texas Inst. for Comp. and Appl. Math., Univ. of Texas at Austin mpesz@ticam.utexas.edu, mfw@ticam.utexas.edu

entire domain. This drawback is somewhat similar to using a uniform fine grid over entire computational domain when such fine resolution is necessary only in a small region in the domain, but that may be alleviated with local adaptive grid refinement. In this paper we present the *multiblock multiphysics* solution methodology that allows physical models to be adaptively selected to match the characteristics of local domain. Moreover, this methodology can be combined with local grid refinement and time stepping, and used for upscaling [33]. While such a formulation leads to advantageous cost/accuracy ratios [19], it also leads to nontrivial modeling and numerical issues. Furthermore, scalable parallel implementation of these simulations presents significant challenges in programming, data-management, runtime management and load-balancing.

The parallel adaptive multiblock, multimodel formulation presented in this paper lifts the traditional paradigm of reservoir simulation and allows for great increase of computing efficiency without loss of accuracy. The methodology and solutions are an integral part of IPARS (Integrated Parallel Accurate Reservoir Simulator) framework developed at the Center for Subsurface Modeling at the University of Texas at Austin [25, 40, 43]. A unique feature of IPARS is the ability to handle multiple physical models assigned to different blocks. Specifically, we handle N_B blocks (subdomains) which are gridded independently and which are separated by N_I interfaces. Assume that we are given N_M models, with $N_M \leq N_B$. From computational point of view, each of the blocks can be considered as an independent domain and implemented as a separate object. However, physically they are part of the same reservoir. Therefore, these blocks and models need to be connected by (i) underlying physical principles of continuity of relevant and appropriate quantities such as mass, momentum, emergy etc., (ii) numerical algorithm defining the discrete form of this continuity and (iii) a computational object that serves as "glue" positioned between these blocks. In this paper we focus on mass flow and transport in subsurface and in i) the continuity of mass and momentum is enforced. Second, in ii) we use *mortar spaces* for mixed finite element methods [2]. Finally, for iii), we use Multiblock Adaptive Computational Engine (MACE). MACE implements a mortar space as a dynamic, semantically specialized distributed shared memory that enables the blocks to transparently share information in spite of their distributions.

Denote by N_C the total number of cells in computational domain which will be considered fixed in this paper, with $N_C(b), N_C(p), N_C(M)$ denoting number of cells assigned to a block b, processor p or model M. Assume that we are given $N_P > 1$ processors. Our parallel implementation builds on the Message Passing Interface (MPI) [38, 15]. The communication substrate defines the parallel operators required to (i) update primary unknowns and other variables in ghost layers (UPDATE) as well as to (ii) communicate scalar properties across processors (GLOBAL_REDUCE). For $N_B = N_M = 1$ we follow the SPMD (Single Program Multiple Data) paradigm where all processors execute the same code and, aside of synchronization issues, most communication calls occur from the same point in the program. Next, assume $N_M = 1, N_B > 1$. This is the case of multiblock single-model execution mode, typical for previous generation of domain decomposition simulators [10, 9] where typically it is required that $N_P = N_B$. In our case N_P is arbitrary. A key issue here is that the different blocks interact on different levels and in different ways. Ensuring physical and numerical continuity across blocks requires communications between processors working on adjacent blocks and it is realized using mortar spaces and communications realized in MACE. Finally, use arbitrary N_B, N_M, N_P, in which case different models

execute on different blocks whose cells are distributed across processors and we follow the MPMD (Multiple Program Multiple Data) paradigm. Since processors execute different models or codes, it is critical to preserve the independence of block and model decomposition from parallel decomposition and enable proper interactions between the different models to avoid conflicts or deadlocks. This problem is addressed by the multiblock communication substrate, which builds on the MPI multi-communicator support. Each model and mortar maintains its own communication context. Some global interaction contexts are also defined to ensure, in particular, global mass balance. The substrate is tuned for parallel scalability. Finally, since different models have different computational cost and simulation speeds and run on grid of different sizes, model-sensitive load balancing is critical to ensure efficient parallel execution. In particular, in this paper we are concerned with decreasing overall computational time $\mathcal{CT}(N_C, N_P, N_B, N_M)$, mainly considering the parallel MPMD issues for $N_M > 1$ and increasing N_P. Solver or grid adaptivity issues will be discussed elsewhere.

Our experience has been mainly with subsurface modeling applications. However, the ideas presented in this paper can be applied to many other *heterogeneous domain decomposition* problems like those in *fluid-structure interactions* [13], or biomedical applications. In addition, there is a wide variety of applications in which different codes are coupled by some form of interface, e.g. see [11, 1]. Our methodology may be applicable to parallel implementation of many of such tight couplings.

The rest of the paper is organized as follows. In Section 2 we define the computational problem. In Section 3 we describe building blocks of our multiblock multiphysics parallel implementation. In Section 4 we present results demonstrating the performance and scalability of the implementation. Section 5 presents our conclusions and an outline of our current and future work.

Results presented in this paper come from multiblock multiphysics implementation under IPARS framework. We would like to acknowledge contributions of John Wheeler, Carter Edwards and Ivan Yotov to the code and to algorithmic development to this implementation.

2. Problem description: multiphase flow and transport in subsurface. In this section we briefly discuss a general model of isothermal multiphase multicomponent flow and transport in subsurface. Discretization of such equations must be conservative and it must accurately describe transfer of mass of individual components between parts of the domain. The complexity of the physical model and low regularity of its solutions constrain choice of the grid as well as the order of approximation. Here we focus on cell centered finite differences (CCFD) which remain probably the most common numerical technique in reservoir simulation. CCFD are conservative as well as have optimal order of convergence for model problems, and are easily implemented as "ijk" grids. The disadvantage of CCFD grids is in their small ability to be adapted to the geological features of a reservoir which may include irregular boundaries, faults, and permeability layers or barriers, or to allow for local refinement around wells. In addition, in traditional setup, all parts of a reservoir are described by the same model. These disadvantages can be lifted by employing the *multiblock* and *multiphysics* paradigm described below.

2.1. Physical and numerical models in a single block. Consider a subsurface reservoir $\Omega \subset \mathbb{R}^3$ whose pores are filled with fluids which can flow or be immobile. We distinguish between *phases* and *components* within these fluids which are denoted by subscripts c and C, respectively. Assume isothermal and equilibrium conditions and no chemical reactions and no adsorption. Use general conservation of

mass equation for each component [27] as follows:

$$\frac{\partial(\phi N_C)}{\partial t} + \nabla \cdot \boldsymbol{V}_C = q_C. \tag{2.1}$$

where concentration N_C of component C is defined as $N_C = \sum_c \rho_c S_c n_{cC}$, with phase density ρ_c and saturation S_c for phase c. Since a component can exist in more than one phase, we define n_{cC} as the mass fraction of component C in phase c. Source term is denoted by q_C and flux $\boldsymbol{V}_C$ is the overall mass flux of component C. Porosity $\phi(x,t), x \in \Omega, t > 0$ is a spatially and pressure-dependent property of the porous medium. Consider advective flux of a component $\boldsymbol{V}_C = \sum_c \rho_c n_{cC} \boldsymbol{U}_c$, where $\boldsymbol{U}_c$ is the velocity of phase c, defined by Darcy's law which expresses momentum conservation and states

$$\boldsymbol{U}_c = -\mathbf{K}\lambda_c(\nabla P_c - \rho_c G \nabla D). \tag{2.2}$$

In general, Forchheimer, Navier-Stokes, or other law, can be used. Here $\mathbf{K}(x), x \in \Omega$ denotes a general (intrinsic) permeability tensor which in this work for simplicity is assumed diagonal and isotropic in both horizontal directions with anisotropy between horizontal and vertical directions; $\mathbf{K} = diag(\mathbf{K}_{hor}, \mathbf{K}_{hor}, \mathbf{K}_{vert})$. $D(x)$ denotes depth and G is the gravity constant. Each phase has an associated pressure P_c, relative permeability k_c and viscosity μ_c, with mobility $\lambda_c = \frac{k_c}{\mu_c}$. Note that $\sum_c S_c = 1, \sum_c n_{cC} = 1$. System is closed by adding capillary pressure relationships and equations of state which specify dependence of density ρ_c and viscosity μ_c on pressure P_c and composition n_{cC}. Model is complemented by a set of appropriate initial and boundary conditions. Finally, it has associated a set of *primary unknowns* Υ which can be used to uniquely identify the state of the system, i.e., to compute values of all the other unknowns. The number of variables in Υ is decided upon with Gibbs phase rule [39, 18] and in general, the choice of Υ is not unique.

This general model includes many submodels. When a component (phase) is not present at $x \in \Omega$, then its concentration (phase saturation) $N_C(x) = 0$ ($S_c(x) = 0$) (Here we ignore the distinction between stationary and absent components/phases.) A specific (sub)model arises when it is clear i) which phases and components are present and flowing, ii) what restrictions are placed on n_{cC}, and iii) which application specific constitutive relationships are used. Some of such specific models are very well-known, understood, and widely used, in both petroleum and environmental engineering [27, 18]. These include in particular 1) a popular single phase incompressible flow modeled by an elliptic equation, 2) a two-phase flow model which is a parabolic-hyperbolic system of 2 equations, and 3) a black-oil model: a coupled system of 3 equations [8, 32, 19, 7, 6] and 4) a compositional model [8, 18] with large number of components. The single phase flow problem is solved for one variable only (pressure of phase c) so $\Upsilon = (P_c)$. Two-phase flow problem is solved simultaneously for two variables (typically, one phase pressure and one other unknown), for example, $\Upsilon = (P_o, N_O)$ (oil phase pressure and oil component concentration). Black-oil model has three primary unknowns (typically, one pressure and two other unknowns), e.g., $\Upsilon = (P_o, N_O, N_G)$ with the latter unknown denoting gas component concentration.

Numerical model. Consider now computational domain Ω_h covering Ω, which is composed of rectangular ijk grid cells. For simplicity we assume $\Omega_h = \Omega$, In general, Ω_h doesn't have to be a parallelepiped and any curved boundary can be approximated by keyed-out cells. Moreover, the grid must only be locally logically rectangular [4, 3].

We apply cell centered finite difference formulation (CCFD) to (2.1) to get

$$\frac{\partial \phi^{ijk} N_C^{ijk}}{\partial t} + (\nabla \cdot \boldsymbol{V}_C)^{ijk} = q_C^{ijk}. \tag{2.3}$$

Next, we discretize Eq. (2.2) in order to define $(\nabla \cdot \boldsymbol{V}_C)^{ijk}$. Note that, under certain circumstances, CCFD is equivalent, up to quadrature error, to the *expanded mixed finite element method* using RT0 spaces [34, 5, 3]. In particular, discrete form of (2.2) in i'th direction reads, with gravity terms omitted for simplicity

$$\boldsymbol{U}_c^{i+1/2,j,k} = -\frac{\mathbf{K}^{i+1/2,j,k} \lambda_c^{i+1/2,j,k}}{\delta x_{i+1/2}} \left(P_c^{i+1,j,k} - P_c^{i,j,k} \right), \tag{2.4}$$

where $\mathbf{K}^{i+1/2,j,k}, \lambda_c^{i+1/2,j,k}$ on the edge $i+1/2, j, k$ between cells i, j, k and $i+1, jk$, are obtained by harmonic averaging of permeabilities $\mathbf{K}_{hor}^{i,j,k}$ and $\mathbf{K}_{hor}^{i+1,j,k}$, and by upwinding, respectively. Definition of $(\nabla \cdot \boldsymbol{V}_C)^{ijk}$ follows in a straightforward manner, see [30] for details. In general, boundary conditions are accounted for in (2.4).

Now, assume that values of primary unknowns Υ_h at time t_0 are given, and define an appropriate temporal discretization, in other words, define how $\Upsilon_h^{n+1}, n = 0, 1, \ldots$ is computed. Usually, additional equations and constraints are directly incorporated to eliminate the need for extra unknowns. For example, an implicit scheme stems from backward-Euler time discretization

$$\frac{\phi^{n+1,ijk} N_C^{n+1,ijk} - \phi^{n,ijk} N_C^{n,ijk}}{\partial t} + (\nabla \cdot \boldsymbol{V}_C)^{n+1,ijk} = q_C^{n+1,ijk}, \tag{2.5}$$

and requires solution of a nonlinear algebraic problem for Υ_h^{n+1} using a Newton-Raphson algorithm [30]. In general, other temporal discretizations, splittings, and time-lagging procedures are possible; each has its specific advantages related to its stability, convergence, and computational complexity. Regardless of details, most share common structure presented in the Figure 2.1.

Step n: given primary unknowns $\Upsilon_h^n =: \Upsilon_h^{n+1,0}$
  Iterate $K = 1, \ldots K_{conv}$
    i) compute nonlinear properties (all other unknowns) using $\Upsilon_h^{n+1,K-1}$
    ii) incorporate boundary conditions and sources at t^{n+1}, use $\Upsilon_h^{n+1,K-1}$
    iii) assemble matrix and rhs of a linear system
    iv) solve linear system and update primary unknowns $\Upsilon_h^{n+1,K}$
Step $n+1$: computed $\Upsilon_h^{n+1} := \Upsilon_h^{n+1,K_{conv}}$

FIG. 2.1. *Time stepping for a single model over a single block. K_{conv} is determined by Newtonian convergence criterium for implicit models, $K_{conv} = 1$ for sequential procedures.*

Computational time $\mathcal{CT}$ required for one time step of such a numerical model on a single processor machine can be roughly estimated as

$$\mathcal{CT}(N_C, 1, 1, 1) = \mathcal{CT}(N_C, 1, 1, \{M\}) \approx C_M {N_C}^{\alpha_M} \tag{2.6}$$

where α_M, C_M depend on model M. In particular, α_M depends mainly on the iterative linear and nonlinear solvers and associated preconditioners, in our experience usually $1 \leq \alpha_M < 2$. C_M includes dependence on the number n_M of (primary) unknowns in Υ for model M.

2.2. Multiblock mortar formulation. Here we consider a domain decomposition method in which the domain Ω is decomposed into non-overlapping subdomains (blocks) Ω_b so that $\Omega = \bigcup_{b=1}^{N_B} \Omega_b$. We assume that each $\Omega_b = (\Omega_b)_h$. Interfaces between blocks are denoted by $\Gamma_l, l = 1 \dots N_I$ and for each l there is $(b_i, b_j) : \Gamma_l = \partial\Omega_{b_i} \cap \partial\Omega_{b_j}$. Denote $\Gamma = \bigcup_{l=1}^{N_I} \Gamma_l$. The original problem (2.1) posed over Ω is formulated as an interface problem on Γ in which one seeks values of *interface primary unknowns* $\boldsymbol{\Lambda}(s), s \in \Gamma$ so that a certain complementary condition is satisfied which we write as $\boldsymbol{B}(\boldsymbol{\Lambda}) = \mathbf{0}$. Values of $\boldsymbol{\Lambda}(s)|_{\Gamma \cap \partial\Omega_b}$ are understood as boundary conditions for subdomain (block) problem on block b which are solved for (subdomain) primary unknowns $\Upsilon|_{\Omega_b}$. Values of $\boldsymbol{B}(\boldsymbol{\Lambda})$ are found from $\Upsilon|_{\Omega_b}, b = 1 \dots N_B$.

In discrete formulation, if the grids are matching, the individual subdomains communicate their data by way of Lagrange multipliers $\boldsymbol{\Lambda}_h$ which are defined at appropriate grid points on Γ. For grids that are non-matching across the interface we follow the development of *mortar spaces* which provide a numerically convergent technique of coupling the blocks together [2]. Figure 2.3 shows a multiblock domain and mortars on block interfaces. Note that mortar grid is completely independent of the subdomain grids. (In general, it has to satisfy some conditions [2]). Regardless whether grids are matching or not, here for simplicity we assume mortar spaces are always used. The algorithm with the use of mortars follows the interface formulation $\boldsymbol{B}(\boldsymbol{\Lambda}) = \mathbf{0}$ and it is enhanced by operations defined between mortar grids and subdomain grids and vice versa. (These operations are counterparts of L^2 projections between finite element spaces defined on subdomains and on mortar grids.) Mortar interface problem $\boldsymbol{B}_h(\boldsymbol{\Lambda}_h) = \mathbf{0}$ is formulated and solved.

For example, consider multiblock formulation for the incompressible single-phase flow problem. A transmission problem for this elliptic equation [37] requires that on the interface we impose i) continuity of pressures (which can be used as Dirichlet boundary condition for the subdomains) and ii) continuity of fluxes (which can be used as the complementary condition). Mortar multiblock formulation for mixed methods for this problem, with $\boldsymbol{\Lambda}_h$ defined as (mortar) pressures and $\boldsymbol{B}_h(\boldsymbol{\Lambda}_h)$ defined as jump of the flux across the interface, has been proven to be optimally convergent [2]. Symbolically, we write $\boldsymbol{\Lambda} = (P_w)$ and $\boldsymbol{B}(\boldsymbol{\Lambda}) = ([\boldsymbol{V}_W])$ where $[\cdot]$ denotes jump (in appropriate weak sense).

Next, consider a general model M as in Eq. (2.3). Here the set of interface primary unknowns $\boldsymbol{\Lambda}(s), s \in \Gamma$, must have a one-to-one correspondance to $\Upsilon(s)$ and it represents values of any unknowns which can be used to impose a boundary condition. One choice is to use $\boldsymbol{\Lambda} \equiv \Upsilon$. For example, in implicit two-phase oil-water model mentioned above, this choice gives $\boldsymbol{\Lambda} = (P_o, N_O)$ and $\boldsymbol{B}(\boldsymbol{\Lambda}) = ([\boldsymbol{V}_O], [\boldsymbol{V}_W])$. Another choice is $\boldsymbol{\Lambda} = (P_w, N_O)$ and $\boldsymbol{B}(\boldsymbol{\Lambda}) = ([\boldsymbol{V}_W], [\boldsymbol{V}_O])$. Since the problem (2.1), in general, is nonlinear, the choice of $\boldsymbol{\Lambda}$ and $\boldsymbol{B}(\boldsymbol{\Lambda})$ may not be straightforward and in implementation some choices may be more efficient than others, e.g., see discussion for two-phase immiscible flow in [29, 30]. After all additional constraints are eliminated, the system $\boldsymbol{B}(\boldsymbol{\Lambda}) = \mathbf{0}$ should be square with dimension of $\boldsymbol{\Lambda}(s)$ equal to the number of components n_M specific to a model M following from Gibbs' rule which is equal to the number of component fluxes.

For numerical solution, we solve $\boldsymbol{B}_h(\boldsymbol{\Lambda}_h) = \mathbf{0}$ which is square as well, with dimension of $\boldsymbol{\Lambda}_h$ equal to $n_M \times n_h$ where n_h denotes the number of (mortar) degrees of freedom. In fact, at every time step t_n we solve $\boldsymbol{B}_h^n(\boldsymbol{\Lambda}_h^n) = 0$. Time-stepping for interface formulation doesn't have to match the time-stepping in individual subdomains. In general, it follows the iterative procedure shown in Figure 2.2 with $I = 1, \dots I_{conv}$

iterations where I_{conv} is determined by an interface stopping criterium based on verifying whether appropriate norm of $\| \boldsymbol{B}_h^n(\boldsymbol{\Lambda}_h^{n,I_{conv}}) \| \approx 0$.

Step n: given $\boldsymbol{\Lambda}_h^{n+1,1} := \boldsymbol{\Lambda}_h^n$ and $\Upsilon_h^n|_{\Omega_b}, b = 1 \dots N_B$
  Iterate $I = 1, \dots I_{conv}$: given current guess $\boldsymbol{\Lambda}_h^{n+1,I}$
    a) solve in all blocks $b = 1 \dots N_B$: iterate $K = 1, \dots K_{conv}(I)$
      i)... ii) ... use $\boldsymbol{\Lambda}_h^{n+1,I}$ as boundary condtitions for block b
      iii)... iv) ... find $\Upsilon_h^{n+1,K_{conv}(I)}|_{\Omega_b}$
    b) compute $\boldsymbol{B}_h(\boldsymbol{\Lambda}_h^{n+1,I})$ and determine new guess $\boldsymbol{\Lambda}_h^{n+1,I}$
Step $n+1$: computed $\boldsymbol{\Lambda}_h^{n+1} := \boldsymbol{\Lambda}_h^{n+1,I_{conv}}$
  and $\Upsilon_h^{n+1}|_{\Omega_b} := \Upsilon_h^{n+1,K_{conv}(I_{conv})}|_{\Omega_b}, b = 1 \dots N_B$

FIG. 2.2. *Time stepping for a multiblock single model*

We remark that in our implementation, the number of iterations $K_{conv}(I)$ in finding $\Upsilon_h^{n+1,K_{conv}(I)}|_{\Omega_b}$ is actually the same for all blocks. This is because, even though blocks are decoupled, the stopping criterium applied in all blocks involves a sum (or maximum) of norms over all blocks. In other words, the overall nonlinear and linear systems solved is block diagonal, with each block corresponding to a subdomain. Therefore, even if residuals in one block are close to zero, iterative linear solver will keep working until residuals in all remaining subdomains are zero. This procedure is easy in implementation and guarantees parallel scalability. Additionally, we assume for simplicity here that vectors $\boldsymbol{\Lambda}(s)$, $\boldsymbol{B}(\boldsymbol{\Lambda})(s)$, and consequently $\boldsymbol{\Lambda}_h$, $\boldsymbol{B}_h(\boldsymbol{\Lambda}_h)$ have the same number and type of components for all $s \in \Gamma$. All these constraints including making $K_{conv} = K_{conv}(b)$ could easily be lifted in implementation, if needed.

Extending Eq. (2.6), we get

$$\mathcal{CT}(N_C, 1, N_B, \{M\}) \approx I_{conv} \left(C_I + C_M \sum_{b=1}^{N_B} N_C(b)^{\alpha_M} \right) \tag{2.7}$$

where, C_I is the cost of operations between blocks and mortars per interface iteration. I_{conv} depends on the quality of interface solver and preconditioners and it is a function of $n_M \times n_h$ on Γ. It our applications it turns out that C_I is dominated by the other terms therefore $\mathcal{CT}(N_C, 1, N_B, \{M\}) \approx I_{conv} C_M \sum_{b=1}^{N_B} N_C(b)^{\alpha_M}$.

2.3. Multiphysics multiblock formulation. For many reasons it is advantageous to apply different models in different parts of the domain. For example, large parts of subsurface reservoirs are filled with water only which can be modelled using a simple single-phase flow model. At the same time, such parts may be connected to oil and gas reservoirs which require a compositional or a black-oil model. Such conditions may persist throughout the life of a reservoir or at least throughout the desired simulation time. It is frequently possible to predict *a-priori* what models should be assigned to what parts of the domain.

In a simple example, consider a reservoir Ω filled with oil O and water W described by a comprehensive two-phase immiscible flow model. In this case, consider multiblock formulation with $\boldsymbol{\Lambda}(s) = (P_w, N_O)$ and $\boldsymbol{B}(\boldsymbol{\Lambda})(s) = ([\boldsymbol{V}_W, \boldsymbol{V}_O]), s \in \Gamma$. Consider time t, and single-phase model M_1 and two-phase model M_2 and a decomposition of $\Omega = \tilde{\Omega}^1(t) \cup \tilde{\Omega}^2(t)$ defined as follows: $\tilde{\Omega}^1(t) = \{x : N_O(x;t) = 0\}, \tilde{\Omega}^2(t) = \Omega \backslash \tilde{\Omega}^1(t)$. Physical

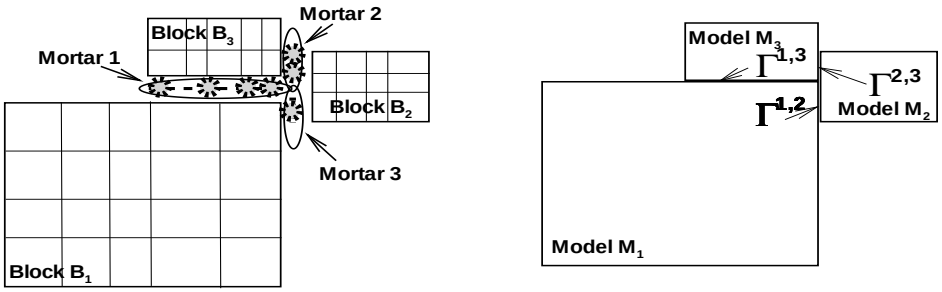

FIG. 2.3. *Left: mutliblock grid. Right: multiphysics assignment of models to blocks. Choice of unknowns* $\mathbf{\Lambda}|_{\Gamma_{2,3}}$ *must be consistent with* $\mathbf{\Lambda}|_{\Gamma_{1,2}}$ *because they both provide boundary conditions to model* M_2*. Mortar grids on each part of* Γ *are independent of each other and of block grids.*

meaning of these sets is that $\tilde{\Omega}^1(t)$ is a domain containing water only which can be described by model M_1. Note that at a given t, each of $\tilde{\Omega}^1(t), \tilde{\Omega}^2(t)$ (but not both) may be empty, and that in most reservoirs water (or at least some of its residual quantities) is present everywhere. Assume therefore $\tilde{\Omega}^1(t) \neq \emptyset$. Define the *free boundary* $\tilde{\Gamma}^{1,2}(t) = \partial\tilde{\Omega}^1(t) \cap \partial\tilde{\Omega}^2(t)$ between the single- and two-phase region. In general, it is difficult to track $\tilde{\Gamma}^{1,2}(t)$ in numerical computations and it would be difficult and impractical to assign a specific model to a time-varying domain $\tilde{\Omega}^1(t)$. Therefore, consider the time-independent decomposition of $\Omega = \Omega^1 \cup \Omega^2$ where $\Omega^1 \subseteq \bigcap_{t\in(0,T)} \tilde{\Omega}^1(t)$. The actual definition of Ω^1 is allowed to vary, see below. For completeness, $\Omega^2 = \Omega \setminus \Omega^1$. Define the interface between the two models $\Gamma^{1,2} = \partial\Omega^1 \cap \partial\Omega^2$.

The main idea is to apply models M_1, M_2 in Ω^1 and Ω^2, and to couple them by interface conditions on $\Gamma^{1,2}$. Note that Ω^1 can be chosen in any way that is convenient for modeling or computational purposes. In particular, it is desirable for Ω^1 to be as large as possible in order to save computational time and so that $\Gamma^{1,2}$ has convenient geometry. In addition, consider multiblock decomposition of Ω so that Ω^1 and Ω^2 are covered each by a separate union of N_B^1 and N_B^2 subdomains (blocks), respectively. Consider $\Gamma = \Gamma^{1,1} \cup \Gamma^{2,2} \cup \Gamma^{1,2}$ so that $\Gamma^{1,1} = \Gamma \cap \Omega^1 - \Gamma^{1,2}$ and $\Gamma^{2,2} = \Gamma \cap \Omega^2 - \Gamma^{1,2}$.

To reformulate the interface conditions locally on different parts of Γ, we note first that $\mathbf{\Lambda} = \Upsilon = (P_w, N_O)$ has to be retained on $\Gamma^{2,2}$. However, there is no need to account for oil-related components inside Ω^1. In fact, since $\Gamma^1 := \Gamma^{1,1} \cup \Gamma^{1,2} \subset \tilde{\Omega}^1(t), \forall t$, we use $\mathbf{\Lambda}|_{\Gamma^1} = (P_w, N_O|_{\tilde{\Omega}^1(t)} = 0)$ and construct a map $\mathbf{\Lambda}|_{\Gamma^1} \mapsto \Upsilon|_{\Omega^2}|_{\Gamma^1}$ which follows from definition of $\tilde{\Omega}^1(t)$. There is also a natural map $\mathbf{\Lambda}|_{\Gamma^1} \mapsto \Upsilon|_{\Omega^1}|_{\Gamma^1}$ which follows by ignoring the second coordinate. In summary, $\mathbf{\Lambda}|_{\Gamma^1} = (P_w, 0)$ can be used as a boundary condition in both Ω^1, Ω^2. Next, in the definition of $\boldsymbol{B}(\mathbf{\Lambda})|_{\Gamma^1}$ we set the jump of the flux of the (missing) oil component to zero.

In general, assume that N_M sub-models can be defined on N_B blocks with $N_B \geq N_M$, each on at least one block, with primary unknowns $\Upsilon^M \in \mathbb{R}^{n_M}$ for model M. There are also $l = 1, \ldots N_I$ interfaces. The numbering of models is arbitrary and the choice of unknowns on each interface Γ_l has to be considered on a case-by-case basis depending on the physical conditions and on the models defined on both sides of this interface, see Figure 2.3, also see [19, 41, 28] for other examples and modeling details. Time stepping for a multiphysics run on a single processor is shown in Figure 2.4.

Now, assume that there is a comprehensive model M_{compr} defined over Ω for which $\Upsilon \in \mathbb{R}^{n_{compr}}, n_{comp} \geq n_M, \forall M,$. The multiphysics procedure in the discrete form amounts to reduction of the algebraic system $\boldsymbol{B}_h(\boldsymbol{\Lambda}_h) = \mathbf{0}$, with $\boldsymbol{\Lambda}_h$ of size $n_{compr} \times n_h$ on Γ as in Eq. (2.7), to a different system, of size $n_l \times n_h$ locally on each interface Γ_l. Note that this reduction really amounts to removing some columns and rows from original system $\boldsymbol{B}_h(\boldsymbol{\Lambda}_h) = \mathbf{0}$ and that this way, we reduce I_{conv} from Eq. (2.7). Moreover, overall computational complexity is decreased in Eq. (2.7) because subdomain solvers assigned to each block are faster than a comprehensive model solver. In fact, dropping C_I, we have

$$\mathcal{CT}(N_C, 1, N_B, N_M) \approx I_{conv} \left(\sum_{M=1}^{N_M} C_M (N_C(M))^{\alpha_M} \right). \tag{2.8}$$

Overall, this last feature dominates the efficiency of multiphysics procedures, see [33, 19]. Finally, in case the comprehensive model doesn't exist or has trouble describing degenerate conditions, the multiphysics coupling procedure described here provides a template for a tight coupling of separate models.

Step n: given $\boldsymbol{\Lambda}_h^{n+1,1} =: \boldsymbol{\Lambda}_h^n$ and $\Upsilon_h^n|_{\Omega_b}, b = 1 \ldots N_B$
  Iterate $I = 1, \ldots I_{conv}$: given current guess $\boldsymbol{\Lambda}_h^{n+1,I}$
    ma) solve for each model $M = 1 \ldots N_M$
      a) solve in blocks $b = 1 \ldots N_B(M)$: iterate $K = 1, \ldots K_{conv}(I, M)$
        i).. iv)
    mb) compute $\boldsymbol{B}_h(\boldsymbol{\Lambda}_h^{n+1,I})$ and determine new guess $\boldsymbol{\Lambda}_h^{n+1,I}$
Step $n+1$: computed $\boldsymbol{\Lambda}_h^{n+1} := \boldsymbol{\Lambda}_h^{n+1,I_{conv}}$
  and $\Upsilon_h^{n+1}|_{\Omega_b} := \Upsilon_h^{n+1,K_{conv}(I_{conv},M)}|_{\Omega_b}, b = 1 \ldots N_B$

FIG. 2.4. *Time stepping for a multiphysics multiblock problem*

3. Parallel implementation. The multiblock, multimodel formulation described in previous section has been implemented in reservoir simulation framework IPARS and tuned for parallel performance. IPARS (Integrated Parallel Accurate Reservoir Simulator) framework provides the infrastructure common to most simulators, such as memory management, grid generation, free-form keyword input, parallel runtime support, and formatted output for visualization, and allows the user to concentrate on the physical model of interest. This paper describes enhancements that enable IPARS to handle multiple blocks and multiple models and an arbitrary number of processors. Each physical or numerical model may have different primary variables, units system, numerical discretization and solutions method (e.g. its own time-stepping or solver) associated with it. The enhancements include (1) the MACE computational engine that provides programming, dynamic data-management and mortar-based coupling support, (2) communication contexts for interactions within and across physical models, especially when they span multiple blocks and processors, and (3) model sensitive load-balancing strategies that take into account computational heterogeneity of the different physical models used while balancing load.

In what follows we first describe the parallel implementation of a single-block single-model under IPARS. We then describe implementation of multiblock simula-

tions using MACE, and implementation of multimodel simulation including model-sensitive load-balancing.

3.1. Single-block implementation. Here $N_B = 1$ and, hence, $N_M = 1$. Consider a time step for a single-block single-model algorithm as shown in Figure 2.1 and an ijk grid. For $N_P > 1$, use a spatial decomposition, assigning pieces of grid to each processor so that dividing lines between processors are aligned with cell boundaries. Since the values of primary unknowns are cell-centered, each cell belongs to at most one processor. Each processor computes the same single-model algorithm as others on its own part of the grid, therefore we realize a SPMD paradigm.

This implementation requires appropriate communications between the processors. First, CCFD have associated a certain stencil such as 7-pt stencil in Eq. (2.4) which requires that each cell communicates with its six neighbors (larger stencils are also possible). To enable this communication, each processor pads its part of the grid with layers of *ghost cells*. These cells serve as place holders for values that are computed on adjacent processors and these are updated using basic MPI[38, 15] point-to-point message passing primitives `send()`, `receive()` encapsulated in an UPDATE call provided by IPARS. Next, the overall algorithm relies on iterative solution of an algebraic problem, and therefore it requires some scalar values such as the total mass of a component, or the maximum residual of a linear or nonlinear system, to be computed. The GLOBAL_REDUCE operation encapsulated in IPARS is realized using either MPI collective communication primitives such as MPI_ALLREDUCE or simple asynchronous `send` and `receive` operations. Note that GLOBAL_REDUCE requires a *rendez-vous* synchronization for all N_P processors. See Figure 3.1 for summary. All processors are part of one general *MPI communicator* group which we will denote $C_0 = \{P_1, \ldots P_{N_P}\}$.

Step n: given $\Upsilon_h^n =: \Upsilon_h^{n+1,0}$. UPDATE($\Upsilon_h^n$)
  Iterate $K = 1, \ldots K_{conv}$
    after i) UPDATE(other unknowns)
    after ii) UPDATE(source terms, boundary conditions)
    after iii) UPDATE(matrix,residual)
    in iv) UPDATE(solution), GLOBAL_REDUCE(stop for linear solver)
  GLOBAL_REDUCE(stop for nonlinear iteration, global mass)
Step $n+1$: computed $\Upsilon_h^{n+1} := \Upsilon_h^{n+1,K_{conv}}$

FIG. 3.1. *Parallel operations in SPMD implementation for a single model over a single block*

Efficiency of this implementation depends on communication costs and on load balancing. Accordingly, the number of ghost cells should be minimized and each processor should be assigned approximately the same amount of work. For single-model algorithms and most of the models that we're concerned with in this paper, the work (load) per cell is approximately the same throughout the whole simulation. Exceptional cases with complicated well or boundary conditions, with severe local heterogeneity which require extra work from the solver, or with entirely explicit models with reaction phenomena, require dynamic load balancing and will not be discussed here. Disregarding these cases, static and grid-based load balancing strategy described below works quite well for most subsurface applications.

Consider an ijk grid superimposed over a reservoir Ω and aligned with the permeability layers. Most reservoirs are much larger in horizontal (aerial) direction than in vertical direction where "vertical" is the gravity direction or one close to it (here assumed as the k direction). Grid spacing in direction k is usually much smaller than in the aerial directions $i \times j$ in order to properly account for gravity effects, including fluid segregation due to density differences. At the same time, anisotropy ratio $\mathbf{K}_{hor}/\mathbf{K}_{vert}$ may be as large as 10, which makes flow in k direction slow. Overall, the maximum number of cells in the vertical dimension N_z is typically much smaller than those in horizontal directions N_x, N_y. IPARS solvers exploit this fact and are optimized to handle N_z in its internal loops; they may constrain the shape of a region assigned to a processor [17].

Now consider aerial $i \times j$ view of the grid and evenly divide the cells in this 2D view between processors so that $N_C(p)$ cells on each processor p form a region as close as possible in shape to a ball (this minimizes the number of ghost cells). For an almost rectangular Ω for which total number of cells $N_C \approx N_x N_y N_z$, we have

$$N_C(p) \approx \frac{N_C}{N_P} = (N_x N_y / N_P) N_z, p = 1, \ldots N_P. \tag{3.1}$$

This grid-based load balancing strategy has been shown to lead to nearly optimal, and in some cases, to super-linear parallel scalability [42]. By linear scalability we mean that the speedup $s(N_P) = \frac{CT(N_C,1,1,1)}{CT(N_C,N_P,1,1)}$ is close to N_P. Here we do not consider scaled speedup [14] in which number of cells per processor is fixed primarily because, if N_C increases with grid refinement, the conditioning of the linear system deteriorates and scaled speedup depends critically on quality of preconditioner which is beyond the scope of this paper.

3.2. Multiblock implementation. Multiblock Adaptive Computational Engine (MACE) is part of a family of unifying *computational engines* aimed at enabling interoperability between application frameworks and solution methodologies which support a family of adaptive PDE solution techniques with multiple structures, multiple scales, and multiple physics [24]. MACE implements a semantically specialized distributed shared memory, extending simple access semantics to dynamic, heterogeneous, and physically distributed data objects spanning different storage types. It encapsulates distributions, communications, coordination, and load balancing. MACE [25, 40, 26] supports multiblock grids where multiple distributed and adaptive blocks with heterogeneous discretizations are coupled together with lower dimensional (also distributed and adaptive) mortar grids.

A separation of concerns in the design of MACE leads to a layered architecture. The lowest layer implements a Hierarchical Distributed Dynamic Array (HDDA), which provides array semantics to hierarchical and physically distributed data. The next layer adds application semantics, implementing objects such as grids, meshes and trees, and providing an object-oriented programming interface for directly expressing multiscale, multi-resolution computations. As the application- and method-specific data objects are based on a common HDDA object, different adaptive solution methods can interact and be combined within a single application. The primary objective for defining such a generalized array data-structure is that most application domain algorithms are formulated as operations on grids and their implementation is defined as operations on arrays. Providing an array interface to the dynamic data-structures allows implementations of new parallel and adaptive algorithms to reuse existing kernels at each level of the HDDA hierarchy. A key feature of HDDA is its ability to

extract out the data locality requirements from the application domain and maintain this locality despite its distribution and dynamics. Two important concepts that underlie the design [23, 20, 22] are hierarchical index spaces using space-filling mappings [35] and extendible hashing storage mechanism [12].

The computational engines provides intuitive programming abstractions and associated operations upon which different formulations can be simply and directly implemented [21]. The abstractions themselves are independently and efficiently implemented on target systems. MACE provides four key abstractions, see Figure 2.3. The *Block* abstraction defines a distributed and adaptive grid hierarchy for each block that is logically rectangular and can be independently refined. It encapsulates distribution, load-balancing and communications. Operations on this abstraction class include adding, deleting and clustering refinements, as well as accessing components grids at a particular level. The *Mortar Grid (Mortar)* abstraction represents the interface grid between *Blocks*. It is essentially a dynamic, semantically specialized, distributed shared space for information exchange between adjacent interfaces of fault blocks. Each *Block* and its associated processors can write to their corresponding portion of the *Mortar* and can read the entire *Mortar*. *Mortars* support refinement and coarsening for mutigrid and adaptive methods. The *Grid Function* abstraction represents application fields defined on the *Block* or *Mortar* grids and defines application data elements and variables. Locally, this abstraction can be viewed as an array that can be indexed in the usual way. Finally, the *Grid Geometry* abstractions include points and boxes and provide a convenient tool for addressing regions in the computational domain.

In implementation, at startup, *Blocks* and *Mortars* are identified and defined along the adjacent interfaces Γ_l. Application specific *Grid Functions* are defined independently on each *Mortar*. The *Mortar Grid* and its associated functions are shared by all processors that have parts of adjacent interfaces. During setup, appropriate interaction contexts and schedules are setup using MPI communicator C_{MACE}. These include contexts to enable processors to update the corresponding portion of the shared *Mortar* or to read from the entire *Mortar*, and to enable processors which own block cells on either side of the *Mortar* to synchronize. Consistency of mortar data is implicitly managed. At runtime MACE defines operations for *Block-Mortar* and *Mortar-Block* interactions, see Figure 3.2. In particular, *Block-Mortar* interactions consist of transferring data from *Blocks* to the *Mortar*. In this case each processor updates only its part of the *Mortar*. Local updates are gathered and aggregated by MACE and can then by committed onto the shared *Mortar*. Now processors on adjacent blocks can read data from this shared object. The actual communication handlers underlying these operations build using asynchronous send and receive operators provided by MPI and avoid the expensive collective communication operators.

Finally, we note that as shown in Figure 2.2, the interface operations and subdomain operations involve two levels of iterations which are sequential with respect to each other. Therefore, for $N_P > 1$, the interface code never locks the subdomain (*Block*) operations. Overall paradigm is MPMD with *Blocks* executing one "program", *Mortars* executing another "program", and *Block-Mortar* interactions representing yet another "program".

As for computational cost and load-balancing for multiblock single-model implementation, we notice that interface costs C_I in Eq. (2.7) now include MACE communication costs. However, in our experience these are very small compared to the subdomain costs which represent the remaining terms in Eq. (2.7). For that reason,

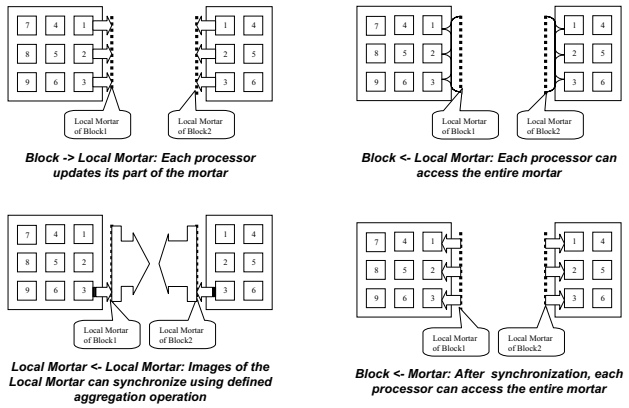

FIG. 3.2. *Multiblock interactions in MACE.*

we extend the grid-based load-balancing strategy to handle multiple blocks by through a straightforward generalization of Eq (3.1): each processor handles approximately

$$N_C(p) = (\sum_{n=1}^{N_B} N_{x_n} N_{y_n})/N_P) N_z, p = 1 \dots N_P. \tag{3.2}$$

where we have assumed for simplicity that approximately $N_z \approx N_{z1} \approx \cdots N_{zN_B}$.

This strategy leads to an almost ideal speedup $s(N_P) = \frac{CT(N_C,1,N_B,1)}{CT(N_C,N_P,N_B,1)} \approx N_P$, with N_B independent of N_P, as reported in [33].

3.3. Multimodel implementation. As mentioned above, IPARS framework has been designed for multiple physical models that can be defined, implemented, and combined within a single executable. Each model defines its own time-stepping and selects the most appropriate solver from a selection of linear solvers and preconditioners. The models may execute simultaneously over multiple subdomains (blocks) which may be connected or disconnected. If they are connected, the mortar interface formulation is used to solve $\boldsymbol{B}(\boldsymbol{\Lambda}) = \mathbf{0}$ and MACE abstractions are actively used at run-time. A user assigns models to blocks and, if they are connected, must decide on the choice of interface variables $\boldsymbol{\Lambda}|_{\Gamma_l}$ for every interface Γ_l.

Consider the algorithm for handling multiple models presented in Figure 2.4 which is straightforward to carry out with $N_P = 1$ or with $N_M = 1$. In general, parallel execution for $N_M > 1$ must be carefully designed to prevent locking and to allow for reasonable parallel scalability. Since different cells execute different algorithms, the underlying paradigm is one of MPMD. We avoid message conflicts by using multiple MPI communicators[15] which require synchronization only between processors within the same communicator.

First consider a simple example with two blocks and two models M_A and M_B running on $N_P = 7$ processors, see Figure 3.3, left. Assume first the blocks are not connected so $N_I = 0$ and no *Mortars* are necessary. Values in the ghost cells belonging to model M_A need only be communicated to other cells in that same model and the stopping criterium for solver associated with this model checks only cells in model M_A. As a result, any model specific UPDATE and GLOBAL_REDUCE operations only apply to model M_A and any *rendez-vous* required by model M_A only involves

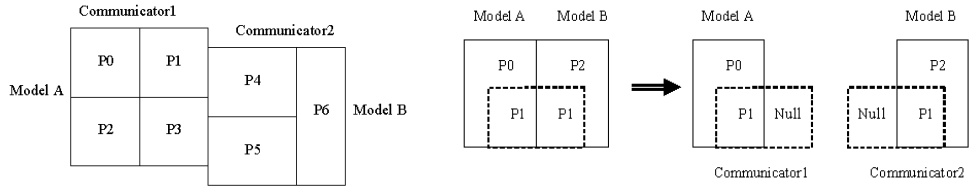

FIG. 3.3. *Multiple communicators for two models. Left: general idea. Right: overlapping communicators. P_1 executes two models, but P_0 and P_2 execute one model only. When P_1 is working as a member of communicator $C_1 = \{P_0, P_1\}$, it does not impact model B*

processors $P_0..P_3$. Similar observations apply to model M_B. Therefore it is only natural to define two communicators: communicator $C_1(M_A) = \{P_0, P_1, P_2, P_3\}$ and $C_2(M_B) = \{P_4, P_5, P_6\}$. Furthermore, since no mass flows between the two models, no global communication is necessary.

Now consider the case in which the two blocks are connected. In addition to the block calculations, there is need for interface calculations. Therefore, a MACE specific communicator C_{MACE} is used. Moreover, mass flows between blocks and therefore, communication between some processors from communicators C_1 and C_2 must be allowed. This is achieved by introducing a global communicator C_0 which groups the first of the processors in each C_1 and C_2, in this case $C_0 = \{P_0, P_4\}$.

In general, in order to ensure complete independence of N_M, N_B, N_P, we must allow for overlapping communicators. For example, only one processor $N_P = 1$ may be available at run-time with $N_B > 1$. Or, consider situation in Figure 3.3, right, with $N_B = 2 = N_M$ and $N_P = 3$. Overlapping communicators are considered for flexibility and portability only because, in general, they lead to poor parallel performance.

In summary, the above strategy is realized as follows. At the beginning of simulation, we split processors into multiple communicator groups: there is a communicator C_M assigned to each model M. In addition, we define a global communicator C_0 and we also have MACE communicator C_{MACE}. At each instant, we use the concept of "current communicator": only processors belonging to it are allowed to communicate. The current communicator is changed and message tags are reset each time i) a model changes or ii) a global communicator or iii) MACE communicator is necessary. Each processor executes the algorithm shown in Figure 3.4. Of course, each processor executes only the steps which are relevant to it: in other words, it works only on the models and blocks that it owns.

3.4. Load Balancing. Traditional grid-based load balancing strategy as described by Eq. (3.1) works reasonably well for single model simulations, also for $N_B > 1$ [33]. However, a straightforward application of this strategy to multimodel case may lead to overlapping communicators which lead to poor parallel performance. Therefore, we develop a model-sensitive (model-based) load balancing strategy. First, if possible, we ensure that a processor does not handle more that one model. Second, we attempt to make the number of processors assigned to a model proportional to its computational complexity.

Consider an example with $N_B = 3, N_M = 2, N_P = 4$ as in Figure 3.5. The first and the third block have model M_A, the middle block has model M_B. First, consider the traditional grid-based distribution (shown on the left) which requires overlap in communicators $C_1(A) \cap C_2(B) \neq \emptyset$ as $C_1(A) = \{P_0, P_1, P_2, P_3\}, C_2 = \{P_1, P_2\}$. All processors are busy during a time step executed by model M_A. On the other hand,

Step n: given $\mathbf{\Lambda}_h^{n+1,1} =: \mathbf{\Lambda}_h^n$ and $\Upsilon_h^n|_{\Omega_b}, b = 1 \ldots N_B$
  Iterate $I = 1, \ldots I_{conv}$: given current guess $\mathbf{\Lambda}_h^{n+1,I}$, choose C_{MACE}
    pma) Solve $M = 1 \ldots N_M$. Choose $C(M)$ as current communicator.
      a) Solve in blocks $b = 1 \ldots N_B(M)$: iterate $K = 1, \ldots K_{conv}(I, M)$
        i).. iv) Use $C(M)$ in all UPDATE and GLOBAL_REDUCE calls.
    pmb) Compute $\boldsymbol{B}_h(\mathbf{\Lambda}_h^{n+1,I})$ and determine new guess $\mathbf{\Lambda}_h^{n+1,I}$
  Choose C_0 as current communicator, GLOBAL_REDUCE(global mass).
Step $n+1$: computed $\mathbf{\Lambda}_h^{n+1} := \mathbf{\Lambda}_h^{n+1,I_{conv}}$
    and $\Upsilon_h^{n+1}|_{\Omega_b} := \Upsilon_h^{n+1,K_{conv}(I_{conv},M)}|_{\Omega_b}, b = 1 \ldots N_B$

FIG. 3.4. *Multiphysics multiblock problem in parallel: use of multiple communicators*

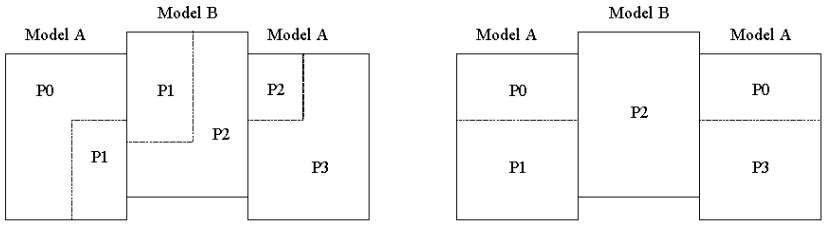

FIG. 3.5. *Load balancing for a 3-block, 2-model case on 4 processors. Left: traditional. Right: model-sensitive*

during a time step for model M_B processors P_0 and P_3 are idle.

Next, consider model-sensitive load balancing. Here each processor is assigned only one model but may span more than one block, for example processor P_0 in Figure 3.5 (right) owns part of block 1 and block 3. In this case $C_1(M_A) = \{P_0, P_1, P_3\}$, and $C_2(M_B) = \{P_2\}$. There is no overlap of communicators and both groups of processors can simultaneously execute time steps for their respective models. However, the loads for processors associated with different models may be different and therefore some processors may remain idle while others work to complete their job. Therefore, if complexity of model M_A is higher than that of model M_B, we should proportionally allocate more processors to model M_A than to model M_B. In other words, the size (rank) of the communicators should be proportional to the relative computational complexity of the models. Ideally, we would assign $N_P(M)$ processors to every Ω^M so that, for each two models M_1, M_2

$$\frac{N_P(M_1)}{N_P(M_2)} \approx \frac{\mathcal{CT}(N_C(M_1), 1, N_B(M_1), \{M_1\})}{\mathcal{CT}(N_C(M_2), 1, N_B(M_2), \{M_2\})}.$$

Note that to satisfy the above strategy, the total number of processors must be $N_P = \sum_M N_P(M)$. In reality, the total number of processors is limited. In addition, it is difficult to precisely estimate the load ratio between models *prior* to their use in a coupled simulation. This is because the physical conditions with and without the coupling are not the same. Moreover, the load depends on static as well as on time-varying factors. Static factors which can be determined once for simulation include number of cells, number of unknowns per cell and of those per ghost cell that need to be communicated in UPDATE calls, and character of time-stepping (implicit,

sequential, semi-implicit). Time-varying factors include efficiency of a linear solver and of the underlying preconditioners, interaction of the model with sources, with boundary conditions on $\partial\Omega$ as well as on interface Γ, convergence tolerance and others. In one example from our experience, the ratio of execution times for the same physical problem simulated by a two-phase implicit and black-oil model ranges between 1.5 and 4 when they are coupled and between 3.5 and 8 when they are not.

These difficulties can be overcome by careful tuning of parallel decomposition as well as of model distribution.

4. Numerical Examples. In this section we focus on demonstrating efficiency of parallel implementation understood as scaling of $\mathcal{CT}(N_C, N_P, N_B, N_M)$ for increasing N_P. In fact, we show that a linear speedup $s(N_P) = \frac{\mathcal{CT}(N_C,1,N_B,N_M)}{\mathcal{CT}(N_C,N_P,N_B,N_M)} \approx N_P$, can be obtained. We note that single-processor efficiency of multiphysics procedures understood as $\mathcal{CT}(N_C, 1, N_B, N_M) < \mathcal{CT}(N_C, 1, 1, \{M_{compr}\})$ was shown elsewhere [19, 33] for M_{compr} being black-oil model. Also, it is clear from Eq. (2.8) that $\mathcal{CT}(N_C, 1, N_B, N_M) < \mathcal{CT}(N_C, 1, N_B, \{M_{compr}\})$. In particular, for cases reported below the $\mathcal{CT}(N_C, 1, 3, \{M_2, M_3\}) = 7046$ while $\mathcal{CT}(N_C, 1, 3, \{M_3\}) = 11850$. Combination of these facts suggest that it is critical to study $s(N_P)$ to determine that efficiency of multiphysics implementation is not destroyed by suboptimal load-balancing.

Our examples involve three blocks (subdomains) as shown in Figure 4.1. On each block we use rectangular grid $60x40' \times 60x40' \times 10x3'$. Subdomains are connected by relatively small interfaces with piecewise linear mortar space on grid 1x1 per each interface. The total number of cells is 108K; it would be almost twice as much without multiblock approach.

We consider two relatively simple examples based on this grid. The choice of such a simple grid and of relatively simple physical conditions is motivated by the focus of this paper on parallel implementation; in general, our simulator is capable of handling much more sophisticated geometry and modeling issues [19, 33]. The examples are run on a 64-node PC cluster connected by a 1.28 GB/sec Myrinet network, with total 32 GB RAM and over 200 GB collective storage; each node is a 300 MHz Intel Pentium II processor. We only use up to $N_P = 16$; communication costs dominate for larger N_P even in SPMD mode.

We consider couplings between the following three models: an implicit black-oil model M_3, a two-phase implicit model M_2, and a sequential two-phase model M_1. Number of subdomain iterations $K_{conv}(M)$ for the implicit models M_2, M_3 depends on a combination of absolute and relative tolerance for Newtonian solver in subdomain related to a constant $\nu_{conv}(M) = 10^{-8}$ and in cases shown here was between 1 and 3. For sequential model, $K_{conv}(M_1) = 1$. For simplicity, we run both examples using the same time steps in all models.

First example involves the coupling of M_3 and M_2 which are assigned, respectively, to the blocks at opposite ends of reservoir and to the middle block, respectively. This assignment is motivated by physical conditions assumed in this example which involve i) difference in the depth over all blocks, and ii) well placement: two production wells completed at opposite ends of the reservoir and one injection well completed in the middle block. See pressure solution in Figure 4.2. Most of the flow occurs from the middle block out to the ends of reservoir. Reservoir is initially filled with water and oil, the latter saturated with gas. During production, pressure is lowered and free gas (gas phase) starts coming out of the solution, see oil concentration in Figure 4.2. Average I_{conv} per time step here is 1.3.

Our second example involves two-phase flow across reservoir with two wells, both

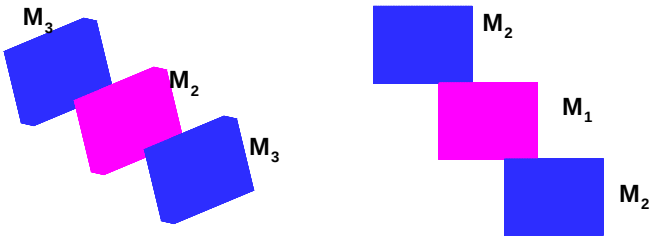

FIG. 4.1. *Geometry, grids and assignment of models. Left: 3D view and assignment of models in example 1. Right: 2D view and assignment of models in example 2.*

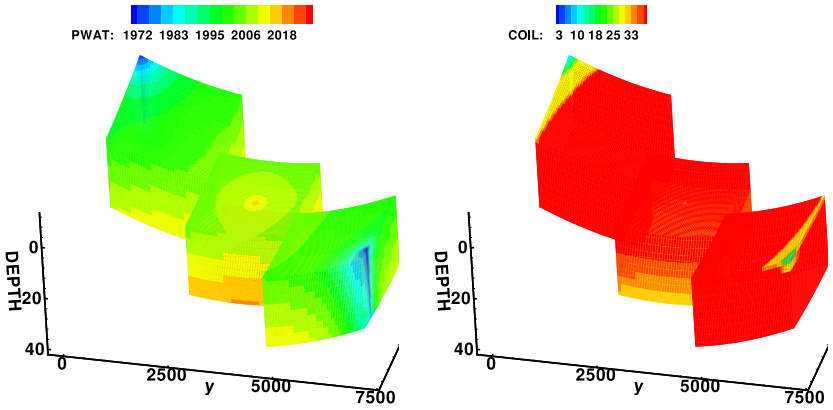

FIG. 4.2. *Results for example 1. Left: pressure solution. Right: oil concentration. Injection wells are marked by high pressure region and production well are marked by low pressure region. Oil contours show effects of gravity (more oil collects on top than at the bottom) and presence of free gas (gas collects at production wells where pressure is low) and displaces oil*

at opposite ends of reservoir, and with no depth variation across the field. In this case we employ the coupling of two different numerical models for two-phase flow: M_2 is used in the subdomains which contain the wells and M_1 is used in the middle block where no well is present. Motivation behind this assignment follows from the fact that high velocities around wells are handled in a more accurate and stable fashion by an implicit model than by a sequential model which is constrained by CFL condition. More details and examples on this multinumeric coupling are provided in [31]. For lack of space, we do not show the solution.

From the point of view of parallel simulation, in spite of very different physical conditions, both examples are somewhat similar because of similar speed-ratio between models, see Table 4.1, left. This is reflected in our tests of parallel speedup in which we consider both the traditional as well as the model-sensitive load-balancing strategies. Table 4.1, right, shows the number of processors assigned to different models and blocks. Figure 4.3 shows parallel speedup obtained for both examples.

The results reported in Figure 4.3 show that the speedup with model-sensitive load balancing is close to the ideal linear speedup and that in some cases it is superlinear. Also, in general, traditional load balancing in multimodel setting may not scale even though for $N_P = 3, 6, 9$ it leads processor decomposition not crossing model

$N_C =$	1000	10000	100000
M_1	1.20	17.70	301.59
M_2	5.62	86.57	1527.48
M_3	22.93	271.72	4986.74

N_P	$N_P(b)$	
	$b = 1, 3$	$b = 2$
1	1	1*
2	1	2**
3	1	1
4	2	1
5	2	1
6	2	2
8	3	2
9	3	3
11	4	3
16	6	4

TABLE 4.1

Left: $CT(N_C, 1, 1, \{M\})$ (in seconds) for $M = M_1, M_2, M_3$ and $N_C = 1K, 10K, 100K$ cells, using different models operating in comparable physical conditions, running for 100 days, with the same time stepping and compatible tolerances. Right: number of processors per block assigned to different models in examples 1 and 2.

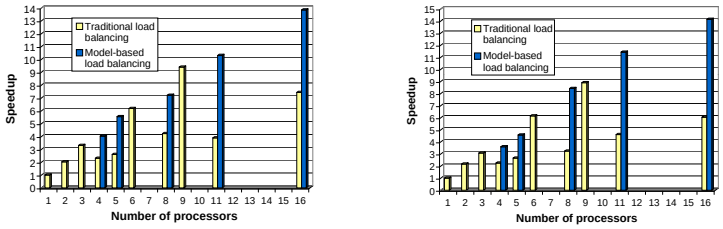

FIG. 4.3. *Parallel speedup in multimodel coupling. Left: speedup for coupling of M_3 and M_2. Right: speedup for coupling of M_2 and M_1.*

boundaries. For small N_P therefore it gives similar speedup as model-sensitive load balancing.

Another observation is that the parallel scaling was better in the first example than in the second. This can be explained by the fact that given a fixed total number of processors N_P the speed ratio for first example fits the given distribution of processors better than the one in second example.

5. Conclusions. We have discussed parallel implementation of multiblock multiphysics algorithms for subsurface flow and transport and shown that it leads to optimal parallel scalability. The strategy employed uses MPI and its multi-communicator concept as well as MACE library for block-mortar interactions and it was implemented in subsurface flow simulator IPARS. This implementation allows an arbitrary number and combination of processors and models, Model-based load balancing strategies have been applied to achieve ideal speedup; in some cases, superlinear speedup can be obtained. Ideas and solutions presented in this paper can be applied to many heterogeneous domain decomposition problems especially those that involve tight couplings of different models or codes.

Future work includes couplings with compositional model and adaptive aspects of

mortar multiphysics formulation and implementations; for example, varying n_h as well as $N_C(b), N_C(M), K_{conv}(b)$ which may require dynamic load balancing. Moreover, a large scale computational study of $\mathcal{CT}(N_C, N_P, N_B, N_M)$, with all the parameters varying simultaneously, is underway. Open questions include definition of multiphysics efficiency for cases where a comprehensive model doesn't exist.

REFERENCES

[1] V. Aizinger and C. N. Dawson. Discontinuous Galerkin methods for two-dimensional flow and transport in shallow water. *Advances in Water Resources*, 25:67–84, 2002.

[2] T. Arbogast, L. C. Cowsar, M. F. Wheeler, and I. Yotov. Mixed finite element methods on non-matching multiblock grids. *SIAM J. Numer. Anal.*, 37:1295–1315, 2000.

[3] T. Arbogast, C. N. Dawson, P. T. Keenan, M. F. Wheeler, and I. Yotov. Enhanced cell-centered finite differences for elliptic equations on general geometry. *SIAM J. Sci. Comp.*, 19(2):404–425, 1998.

[4] T. Arbogast, P. T. Keenan, M. F. Wheeler, and I. Yotov. Logically rectangular mixed methods for Darcy flow on general geometry. In *Thirteenth SPE Symposium on Reservoir Simulation, San Antonio, Texas*, pages 51–59. Society of Petroleum Engineers, Feb. 1995. SPE 29099.

[5] T. Arbogast, M. F. Wheeler, and I. Yotov. Mixed finite elements for elliptic problems with tensor coefficients as cell-centered finite differences. *SIAM J. Numer. Anal.*, 34(2):828–852, 1997.

[6] L. Bergamaschi, S. Mantica, and G. Manzini. A mixed finite element–finite volume formulation of the black-oil model. *SIAM J. Sci. Comput.*, 20(3):970–997 (electronic), 1999.

[7] Z. Chen. Formulations and numerical methods for the black oil model in porous media. *SIAM J. Numer. Anal.*, 38(2):489–514 (electronic), 2000.

[8] K. H. Coats, L. K. Thomas, and R. G. Pierson. Compositional and black oil reservoir simulator. In *the 13th SPE symposium on reservoir simulation*, San Antonio, Texas, Feb 12-15 1995.

[9] L. C. Cowsar, J. Mandel, and M. F. Wheeler. Balancing domain decomposition for mixed finite elements. *Math. Comp.*, 64:989–1015, 1995.

[10] L. C. Cowsar, A. Weiser, and M. F. Wheeler. Parallel multigrid and domain decomposition algorithms for elliptic equations. In D. E. Keyes et al., editors, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 376–385. SIAM, Philadelphia, 1992.

[11] M. Discacciati, E. Miglio, and A. Quarteroni. Mathematical and numerical models for coupling surface and groundwater flows. submitted to Applied and Numerical Mathematics, 2001.

[12] R. Fagin. Extendible hashing-a fast access mechanism for dynamic files. *ACM TODS*, 4:315–344, 1979.

[13] C. Farhat, J. Mandel, and F.-X. Roux. Optimal convergence properties of the FETI domain decomposition method. *Comput. Methods Appl. Mech. Engrg.*, 115(3-4):365–385, 1994.

[14] I. Foster. *Designing and Building Parallel Programs.* Addision-Wesley, 1995.

[15] W. Group, E. Luak, and A. Skyjellum. *Using MPI.* The MIT Press, 1994.

[16] J. Killough and D. Commander. Scalable parallel reservoir simulation on a Windows NT-based workstation cluster. In *SPE paper 51883 presented at the Fifteenth SPE Symposium on Reservoir Simulation, February, 1999*, pages 41–50, 1999.

[17] S. Lacroix, Y. Vassilevski, and M. F. Wheeler. Iterative solvers of the implicit parallel accurate reservoir simulator (IPARS). *Numerical Linear Algebra with Applications*, 4:537–549, 2001.

[18] L. W. Lake. *Enhanced oil recovery.* Prentice Hall, 1989.

[19] Q. Lu, M. Peszyńska, and M. F. Wheeler. A parallel multi-block black-oil model in multi-model implementation. *SPE Journal*, 7(3):278–287, September 2002. SPE 79535.

[20] M. Parashar and J. C. Browne. Distributed dynamic data-structures for parallel adaptive mesh-refinement. In *Proceedings of the International Conference for High Performance Computing*, pages 22–27. IEEE Computer Society Press, Dec. 1995.

[21] M. Parashar and J. C. Browne. Object-oriented programming abstractions for parallel adaptive mesh-refinement. In *Parallel Object-Oriented Methods and Applications (POOMA)*, Santa Fe, NM, Feb. 1996.

[22] M. Parashar and J. C. Browne. On partitioning dynamic adaptive grid hierarchies. In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences (HICSS 96)*, pages 604–613, Maui, Hawaii, Jan. 1996. IEEE Computer Society Press.

[23] M. Parashar and J. C. Browne. System engineering for high performance computing software:

The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement. In S. B. Baden, N. P. Chrisochoides, D. B. Gannon, and M. L. Norman, editors, *IMA Volume 117: Structured Adaptive Mesh Refinement (SAMR) Grid Methods*, pages 1–18. Springer-Verlag, Jan. 2000.

[24] M. Parashar, J. C. Browne, C. Edwards, and K. Klimkowsky. A common data management infrastructure for parallel adaptive algorithms for pde solutions. In *Proceedings of Supercomputing '97*, San Jose, CA, Nov. 1997. ACM Sigarch and IEEE Computer Society, IEEE Computer Society Press.

[25] M. Parashar, J. A. Wheeler, G. Pope, K. Wang, and P. Wang. A new generation EOS compositional reservoir simulator. Part II: Framework and multiprocessing. In *Fourteenth SPE Symposium on Reservoir Simulation, Dalas, Texas*, pages 31–38. Society of Petroleum Engineers, June 1997.

[26] M. Parashar and I. Yotov. An environment for parallel multi-block, multi-resolution reservoir simulations. In *Proceedings of the 11th International Conference on Parallel and Distributed Computing and Systems (PDCS 98)*, pages 230–235, Chicago, IL, Sep. 1998. International Society for Computers and their Applications (ISCA).

[27] D. W. Peaceman. *Fundamentals of numerical reservoir simulation.* Elsevier Scientfic Publishing Company, Amsterdam-Oxford-New York, first edition, 1977.

[28] M. Peszyńska. Multiphysics coupling of three-phase and two-phase models of flow in porous media. submitted.

[29] M. Peszyńska. Advanced techniques and algorithms for reservoir simulation III. Multiphysics coupling for two phase flow in degenerate conditions. In J. Chadam, A. Cunningham, R. E. Ewing, P. Ortoleva, and M. F. Wheeler, editors, *IMA Volume 131: Resource Recovery, Confinement, and Remediation of Environmental Hazards*, pages 21–40. Springer, 2002.

[30] M. Peszyńska, E. Jenkins, and M. F. Wheeler. Boundary conditions for fully implicit two-phase flow model. In X. Feng and T. P. Schulze, editors, *Recent Advances in Numerical Methods for Partial Differential Equations and Applications*, volume 306 of *Contemporary Mathematics Series*, pages 85–106. American Mathematical Society, 2002.

[31] M. Peszyńska, Q. Lu, and M. F. Wheeler. Coupling different numerical algorithms for two phase fluid flow. In J. R. Whiteman, editor, *MAFELAP Proceedings of Mathematics of Finite Elements and Applications*, pages 205–214, Uxbridge, U.K., 1999. Brunel University.

[32] M. Peszyńska, Q. Lu, and M. F. Wheeler. Multiphysics coupling of codes. In L. R. Bentley, J. F. Sykes, C. A. Brebbia, W. G. Gray, and G. F. Pinder, editors, *Computational Methods in Water Resources*, pages 175–182. A. A. Balkema, 2000.

[33] M. Peszyńska, M. F. Wheeler, and I. Yotov. Mortar upscaling for multiphase flow in porous media. *Computational Geosciences*, 6:73–100, 2002.

[34] T. F. Russell and M. F. Wheeler. Finite element and finite difference methods for continuous flows in porous media. In R. E. Ewing, editor, *The Mathematics of Reservoir Simulation*, pages 35–106. SIAM, Philadelphia, 1983.

[35] H. Sagan. *Space-Filling Curves.* Springer-Verlag, 1994.

[36] Schlumberger Technology Corporation. *Eclipse-100, Reference Manual*, 1998.

[37] R. E. Showalter. *Hilbert space methods for partial differential equations.* Pitman, London, 1977. Monographs and Studies in Mathematics, Vol. 1.

[38] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dong arra. *MPI: the complete reference.* The MIT Press, 1996.

[39] J. W. Tester and M. Modell. *Thermodynamics and Its Applications.* Prentice Hall, 1997.

[40] P. Wang, I. Yotov, M. F. Wheeler, T. Arbogast, C. N. Dawson, M. Parashar, and K. Sepehrnoori. A new generation EOS compositional reservoir simulator. Part I: Formulation and discretization. In *Fourteenth SPE Symposium on Reservoir Simulation, Dalas, Texas*, pages 55–64. Society of Petroleum Engineers, June 1997.

[41] M. F. Wheeler and M. Peszyńska. Computational engineering and science methodologies for modeling and simulation of subsurface applications. Advances in Water Resources, in press.

[42] M. F. Wheeler, M. Peszyńska, X. Gai, and O. El-Domeiri. Modeling subsurface flow on PC cluster. In A. Tentner, editor, *High Performance Computing*, pages 318–323. SCS, 2000.

[43] M. F. Wheeler, J. A. Wheeler, and M. Peszyńska. A distributed computing portal for coupling multi-physics and multiple domains in porous media. In L. R. Bentley, J. F. Sykes, C. A. Brebbia, W. G. Gray, and G. F. Pinder, editors, *Computational Methods in Water Resources*, pages 167–174. A. A. Balkema, 2000.

[44] M. White, M. Oostrom, and R. Lenhard. Modeling fluid flow and transport in variably saturated porous media with the STOMP simulator. 1. Nonvolatile three-phase model description. *Advances in Water Resources*, 18(6), 1995.