Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 196 (2004) 645–679
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Operator splitting and adaptive mesh refinement for the Luo–Rudy I model

John A. Trangenstein [*,1], Chisup Kim [2]

Department of Mathematics, Center for Multi-Scale Modeling and Distributed Computing, Duke University, Physics Building Room 024D, Durham, NC 27708-0320, USA

Received 22 May 2003; received in revised form 11 November 2003; accepted 13 November 2003

Abstract

We apply second-order operator splitting to the Luo–Rudy I model for electrical wave propagation in the heart. The purpose of the operator splitting is to separate the nonlinear but local reaction computations from the linear but globally coupled diffusion computations. This approach allows us to use *local nonlinear* iterations for the stiff nonlinear reactions and to solve *global linear* systems for the implicit treatment of diffusion. For computational efficiency, we use dynamically adaptive mesh refinement (AMR), involving hierarchies of unions of grid patches on distinct levels of refinement. The linear system for the discretization of the diffusion on the composite AMR grid is formulated via standard conforming finite elements on unions grid patches within a level of refinement and aligned mortar elements along interfaces between levels of refinement. The linear systems are solved iteratively by preconditioned conjugate gradients. Our preconditioner uses multiplicative domain decomposition between levels of refinement; the smoother involves algebraic additive domain decomposition between patches within a level of refinement, and Gauss–Seidel iteration within grid patches. Numerical results are presented in 1D and 2D, including spiral waves.

Keywords: Reaction–diffusion; Excitable media; Adaptive mesh refinement; Operator splitting; Finite elements; Multigrid; Domain decomposition

1. Introduction

1.1. Modeling electrical activity in the heart

Theoretical studies of electrical activity of the heart have focused on many important issues. Examples include propagation of action potentials [31,56], development of spiral waves [45,59], and

* Corresponding author.
E-mail address: johnt@math.duke.edu (J.A. Trangenstein).
[1] Work supported by NSF Grant DMS-98-70384.
[2] Work supported by NSF grant DMS-97-09608.

0021-9991/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2003.11.014

effects of strong electric shocks such as those used in electrical termination of arrhythmias (defibrillation) [19,27,33,43,55,58]. Accurate calculation of the response of the heart to electrical stimuli must take into account realistic fiber geometry, anisotropy of cardiac conductivities, detailed membrane properties, microscopic tissue structure, and the inhomogeneous nature of myocardium [36,43,45,54].

1.2. Survey of numerical methods

The objective of this paper is to increase the computational efficiency of such simulations by employing advanced numerical methods. We have chosen to study the simple Luo–Rudy I model [34], which can still produce reasonably realistic simulations. LRI involves a single membrane potential, and eight reactions for the calcium concentration and seven gating variables.

The numerical methods previously used in simulations of cardiac electrical activity can be divided into two groups. The methods of the first, larger group are non-adaptive: they use fixed spatial grids and fixed time-steps. These methods are implemented both on parallel computers [47,56] and ordinary workstations [2,35,44]; they use both explicit [15,35,44,47] and semi- or fully implicit time-stepping techniques [2,32,42,56,57]. While clever implementation allowed the researchers to conduct some sample 3D simulations, the resulting modeling tool is very expensive. For example, active anisotropic models that run on parallel machines (256 processor CM5) have been reported to require more than 15 h to simulate 100 ms of action potential propagation in a $0.8 \times 0.8 \times 0.1$ cm piece of the cardiac muscle [47]. The main reason for this expense is the weakness common to all the methods from this group. When algorithms fail to recognize regions of high electrical activity and to apply separate numerical treatment depending on the level of electrical activity, the resulting codes suffer stringent limits on the size of the time-step and spatial resolution for the whole region of calculations.

The methods of the second group employ spatial adaptivity, temporal adaptivity, or both in order to avoid this limitation. The intent of these algorithms is to concentrate the computational work in regions of high electrical activity. Because the positive-definite conductivity tensors σ_i and σ_e are relatively small compared to the reaction rate, these regions of high electrical activity will have a small width.

At best, an adaptive scheme could reduce the total computational work by the factor given by the ratio of the domain volume to the total volume of the regions of high electrical activity. This upper bound on the adaptive speedup is problem-dependent. If the reaction rates are large compared to the conductivities, these regions of high electrical activity will be small, and the potential speedup from an adaptive mesh refinement (AMR) scheme is large.

Several authors have used such adaptive algorithms successfully. A domain-decomposition method combined with an alternating direction implicit (ADI) Rush–Larsen method implemented in [41] dynamically tracks active regions, decomposes the region of computation into small subdomains, and uses explicit time-stepping in the subdomains (locally) and implicit method for global integration. This algorithm is temporally adaptive, but not spatially adaptive. The authors report time savings on the order of 3–17, compared with a non-adaptive technique. Significantly larger 2D models with active membrane dynamics (LR phase II was used in the paper) can therefore be used for simulations. A similar approach, involving a combination of an implicit integration technique with multigrid iteration, is employed in [38]. A large modular code has recently been developed to accommodate a wide variety of existing cardiac models and numerical approaches in [40]. This code is not adaptive in space. It allows the user to choose an adaptive time-integration technique among a few explicit, semi-implicit, and implicit methods. The linear algebra is handled by a choice of an iterative method (CG, GMRES) with a preconditioner (SOR, incomplete Cholesky). An irregular grid option allows for complex geometries. Finally, the code is organized to allow parallelization. The lack of spatial adaptivity prevents the direct extension of these methods into three spatial dimensions. Electric stimulation of the tissue by strong shocks especially calls for spatial adaptivity

in the regions near the electrodes and thus makes the method less than optimal for defibrillation studies even in two dimensions.

The Rush–Larsen temporal adaptation technique was also considered in [39]. Here, it was combined with explicit, semi-implicit, and fully implicit methods, as well as with a spatial adaptation technique. The active front was tracked dynamically. All the nodes were marked as either "active" (near the front) or "inactive" and the calculations were performed only at the "active" nodes. This approach does not allow control of the error in the transmembrane potential as it accumulates over time. This model also seems to be only applicable for propagation studies.

Methods that are adaptive both temporally and spatially are only beginning to be developed. A space-time adaptive approach that uses finite differences and is explicit in time has already been shown to produce a factor of 5 reduction in 2D computational time and memory expense [14], relative to the non-adaptive explicit algorithm.

1.3. Overview of the paper

The principal numerical developments in this paper are the construction of an effective multiplicative domain decomposition algorithm for such a hierarchical grid and the development of effective synchronization strategies for adaptation in time. The issues involve both the use of multigrid-like iteration on a nested hierarchical grid for efficient solution of the diffusion equation and the avoidance of boundary layers at interfaces between coarse and fine grid.

In Section 2 we will describe the Luo–Rudy I model and our numerical implementation of its model functions. In particular, we will describe how we avoid unnecessary overflow and cancellation error as well as excessive computational cost in evaluating these model functions. Next, we will describe a second-order operator splitting of reaction and diffusion in Section 3. The purpose of the operator splitting is to separate the nonlinear but local reaction computations from the linear but globally coupled diffusion computations. We will describe our techniques for integrating the reactions in Section 4 and our techniques for integrating the diffusion in Section 5. We will use a stiff diagonally-implicit Runge–Kutta (SDIRK) scheme to integrate the reactions; this scheme is both L-stable and A-stable. We will use a piecewise linear finite element method with implicit Crank–Nicolson time integration to integrate the diffusion equation. For adaptive mesh refinement (AMR) purposes, we will also describe how we formulate our discretization of diffusion on locally refined domains in Section 5.4.

These computational techniques for uniform grids will give us the basic tools for developing our adaptive mesh refinement (AMR) scheme in Section 6. We describe modifications to time-step selection in Section 6.1. We will describe how the AMR algorithm chooses to relocate the finer mesh in Section 6.2. Techniques for communicating between scales in the mesh hierarchy are described in Section 6.3; this section also discusses some techniques to avoid taking fine time-steps on coarse grid. We describe modifications to the iterative linear algebra for AMR in Section 7. The linear system on the hierarchical grid involves appropriate use of mortar finite elements to determine the composite grid equations; this is described in Section 5.4. The iterative scheme uses a conjugate gradient iteration, preconditioned by multiplicative domain decomposition, which is described in Section 7.1. In order to take advantage of the organization of the grid hierarchy, in Section 7.2 we will describe a smoother that uses additive domain decomposition. This smoother works well on an array of grid patches and extends well to distributed computing.

Finally, we will present some numerical results in Section 8.

2. Description of the Luo–Rudy I model

The Luo–Rudy I model is a reaction–diffusion system of the form

$$\forall \mathbf{x} \in \Omega,\ \forall t > 0, \quad \frac{\partial \mathbf{u}}{\partial t} \equiv \frac{\partial}{\partial t}\begin{bmatrix} \mathbf{g} \\ [\mathrm{Ca}]_{\mathrm{i}} \\ V \end{bmatrix} = \begin{bmatrix} \mathbf{a}(V) - (\mathbf{a}(V) + \mathbf{b}(V))\mathbf{g} \\ \rho_{[\mathrm{Ca}]_{\mathrm{i}}}(\mathbf{g}, [\mathrm{Ca}]_{\mathrm{i}}, V) \\ \rho_V(\mathbf{g}, [\mathrm{Ca}]_{\mathrm{i}}, V) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \nabla_x \cdot D(\mathbf{x}) \nabla_x V \end{bmatrix}$$
$$\equiv R(u) + D(V), \tag{1a}$$

$$\forall \mathbf{x} \in \partial\Omega,\ \forall t > 0, \quad \mathbf{n} \cdot \nabla_x V = 0, \tag{1b}$$

$$\forall \mathbf{x} \in \Omega, \quad \mathbf{u}(\mathbf{x}, 0) \quad \text{given.} \tag{1c}$$

The vector $\mathbf{u}$ of unknowns consists of the vector $\mathbf{g}$ of gating variables (dimensionless), the intracellular calcium concentration $[\mathrm{Ca}]_{\mathrm{i}}$ (in millimolar), and the membrane potential V (in millivolts). The vectors $\mathbf{a}(V)$ and $\mathbf{b}(V)$ are vectors of reaction rates for the gating variables, and are described in Eqs. (A.5a)–(A.10b). The notation $(\mathbf{a} + \mathbf{b})\mathbf{g}$ represents the componentwise product of the two vectors $\mathbf{a} + \mathbf{b}$ and $\mathbf{g}$.

These functions have been modified slightly from the definitions in [34] to avoid unnecessary overflow and discontinuity; for details, see Appendix A. We do not guard against underflow. Even with our modifications, overflow can occur for very extreme values of V. On our machine, the first double-precision overflow occurs for negative V in $\beta_m(V)$ around $V = -7807$, and for positive V in $\beta_{K_1}(V)$ around $V = 8831$ mV. (Voltages of such magnitude are highly destructive, causing at least electroporation, and should not be reached.) With the original Luo–Rudy I model functions, the earliest overflow occurred in $\beta_{K_1}(V)$ for $V < -1472$ or $V > 1288$. In single precision, overflow in the original Luo–Rudy I parameter $\beta_{K_1}(V)$ would occur for $V < -184$ or $V > 161$ mV; this is close to the range of values that occur in our simulations.

The functional forms (A.2) and (A.5a)–(A.10b) are very expensive to evaluate during the numerical solution of the differential equation (1). We use Hermite cubic spline interpolants to these functions on 20 uniform intervals over $-100 \leqslant V \leqslant 100$. These splines are as accurate in representing the Luo–Rudy I model functions as those functions were accurate in representing the physical data. Further, these splines are continuously differentiable and much less expensive to evaluate.

3. Operator splitting

The system of equations (1) involves both reaction and diffusion. Our approach to integrating these equations will be to use a second-order operator splitting [48,49,53]:

$$\text{solve} \quad \begin{cases} \frac{\partial \mathbf{u}^{R,n+\frac{1}{2}}}{\partial t} = R(\mathbf{u}^{R,n+\frac{1}{2}}) & \forall \mathbf{x} \in \Omega,\ \forall t^n \leqslant t \leqslant t^n + \frac{1}{2}\Delta t^{n+\frac{1}{2}}, \\ \mathbf{u}^{R,n+\frac{1}{2}}(\mathbf{x}, t^n) = \mathbf{u}(\mathbf{x}, t^n) & \forall \mathbf{x} \in \Omega, \end{cases} \tag{2a}$$

$$\text{solve} \quad \begin{cases} \frac{\partial \mathbf{u}^{D,n+1}}{\partial t} = D(\mathbf{u}^{D,n+1}) & \forall \mathbf{x} \in \Omega,\ \forall t^n \leqslant t \leqslant t^n + \Delta t^{n+\frac{1}{2}}, \\ \mathbf{n} \cdot \nabla_x V^{D,n+1}(\mathbf{x}, t) = 0 & \forall \mathbf{x} \in \partial\Omega,\ \forall t^n \leqslant t \leqslant t^n + \Delta t^{n+\frac{1}{2}}, \\ \mathbf{u}^{D,n+1}(\mathbf{x}, t^n) = \mathbf{u}^{R,n+\frac{1}{2}}(\mathbf{x}, t^n + \frac{1}{2}\Delta t^{n+\frac{1}{2}}) & \forall \mathbf{x} \in \Omega, \end{cases} \tag{2b}$$

$$\text{solve} \quad \begin{cases} \frac{\partial \mathbf{u}^{n+1}}{\partial t} = R(\mathbf{u}^{n+1}) & \forall \mathbf{x} \in \Omega,\ \forall t^n + \frac{1}{2}\Delta t^{n+\frac{1}{2}} \leqslant t \leqslant t^n + \Delta t^{n+\frac{1}{2}}, \\ \mathbf{u}^{n+1}(\mathbf{x}, t^n + \frac{1}{2}\Delta t^{n+\frac{1}{2}}) = \mathbf{u}^{D,n+1}(\mathbf{x}, t^{n+1}) & \forall \mathbf{x} \in \Omega. \end{cases} \tag{2c}$$

Here, we use the centered notation $\Delta t^{n+\frac{1}{2}} = t^{n+1} - t^n$. This splitting of the problem is second-order accurate in time. The advantage of this approach is that we can integrate the reactions using stiff solvers with *localized* time-step control at individual grid nodes and we can integrate the diffusion using fast *linear* solvers.

4. Integration of reactions

Recall from Eqs. (2a) and (2c) that the operator-split reaction problems have the form

$$\frac{\partial \mathbf{u}}{\partial t} = R(\mathbf{u}) \quad \forall \mathbf{x} \in \Omega, \ \forall t^n < t < t^n + \Delta t,$$
$$\mathbf{u}(\mathbf{x}, t^n) \text{ given } \forall \mathbf{x} \in \Omega.$$

In practice these ordinary differential equations are stiff, meaning that $\partial R/\partial \mathbf{u}$ has a wide range of eigenvalues. As a result, we will use appropriate stiff solvers.

4.1. Singly diagonally implicit Runge–Kutta scheme

To integrate these ordinary differential equations, we will use a second-order SDIRK scheme, described in [18]. Our basic integration scheme takes the form

$$\mathbf{k}_1 = R(\mathbf{u}^n + \mathbf{k}_1 \gamma \Delta t), \tag{3a}$$

$$\mathbf{k}_2 = R(\mathbf{u}^n + [1 - 2\gamma]\mathbf{k}_1 \Delta t + \gamma \mathbf{k}_2 \Delta t), \tag{3b}$$

$$\mathbf{u}^{n+1} = \mathbf{u}^n + (\mathbf{k}_1 + \mathbf{k}_2)\frac{\Delta t}{2}. \tag{3c}$$

Here, $\gamma \equiv 1 - \sqrt{1/2}$, so that the scheme is both L-stable and A-stable. Extensions of this scheme to higher order can be found in [28,37].

4.2. Solving the nonlinear systems

The SDIRK scheme equations (3a) and (3b) require the solution of nonlinear equations of the form

$$\mathbf{f}(\mathbf{k}) \equiv \mathbf{k} - R(\mathbf{u} + \mathbf{k}\gamma \Delta t) = 0$$

for fixed $\mathbf{u}$ and fixed Δt. Newton iteration to solve such a nonlinear system replaces $\mathbf{k}$ by $\mathbf{k} + \mathbf{s}$, where $\mathbf{s}$ solves

$$\frac{\partial \mathbf{f}}{\partial \mathbf{k}}\mathbf{s} = -\mathbf{f}(\mathbf{k}).$$

Note that the Jacobian matrix $\partial \mathbf{f}/\partial \mathbf{k}$ has a special structure for the Luo–Rudy I model:

$$\frac{\partial \mathbf{f}}{\partial \mathbf{k}} = \begin{bmatrix} I + \operatorname{diag}(\mathbf{a}) + \operatorname{diag}(\mathbf{b}) & 0 & -\mathbf{a}' + (\mathbf{a}' + \mathbf{b}')\mathbf{g} \\ -\frac{\partial \rho_{[\mathrm{Ca}]_i}}{\partial \mathbf{g}} & 1 - \frac{\partial \rho_{[\mathrm{Ca}]_i}}{\partial [\mathrm{Ca}]_i} & -\frac{\partial \rho_{[\mathrm{Ca}]_i}}{\partial V} \\ -\frac{\partial \rho_V}{\partial \mathbf{g}} & -\frac{\partial \rho_V}{\partial [\mathrm{Ca}]_i} & 1 - \frac{\partial \rho_V}{\partial V} \end{bmatrix} \equiv \begin{bmatrix} \mathbf{D} & \mathbf{G} \\ \mathbf{F} & \mathbf{T} \end{bmatrix},$$

where $\mathbf{D} = I + \mathrm{diag}(\mathbf{a}) + \mathrm{diag}(\mathbf{b})$ is 7×7 and diagonal, and the vectors $\mathbf{a}$ and $\mathbf{b}$ are as in Eq. (1a) for the gating variables. As a result, it is easy to use the first seven equations to eliminate the increments for the gating variables from the system of nine Newton equations, and reduce to a system of two equations involving the increments for the intracellular calcium $[\mathrm{Ca}]_i$ and the membrane potential V. Specifically, in order to solve

$$\begin{bmatrix} \mathbf{D} & \mathbf{G} \\ \mathbf{F} & \mathbf{T} \end{bmatrix} \begin{bmatrix} \mathbf{s_g} \\ \mathbf{s}_2 \end{bmatrix} = - \begin{bmatrix} \mathbf{f_g} \\ \mathbf{f}_2 \end{bmatrix},$$

we compute

$$\begin{aligned} \mathbf{M} &= \mathbf{T} - \mathbf{F}\mathbf{D}^{-1}\mathbf{G} \in \mathbf{R}^{2\times 2}, \\ \mathbf{s}_2 &= -\mathbf{M}^{-1}(\mathbf{f}_2 - \mathbf{F}\mathbf{D}^{-1}\mathbf{f_g}) \in \mathbf{R}^2, \\ \mathbf{s_g} &= -\mathbf{D}^{-1}(\mathbf{f_g} + \mathbf{G}\mathbf{s}_2) \in \mathbf{R}^7. \end{aligned}$$

Any linear systems involved in Newton's method are either 7×7 diagonal (involving $\mathbf{D}$) or 2×2 (involving $\mathbf{M}$).

4.3. Comments on reaction integrators

Our use of second-order integration methods for reaction is consistent with our use of second-order operator splitting of the reaction and diffusion (described in Section 3), with our second-order integration of diffusion (described in Section 5), and with our use of continuously differentiable cubic splines to represent the gate rates $\mathbf{a}(V)$ and $\mathbf{b}(V)$ (described in Section 2). However, stimulus currents $I_{\mathrm{st}}(\mathbf{x}, t)$ that involve discontinuities in time can prevent our reaction integrations from exhibiting second-order accuracy.

It is reasonable to ask whether higher-order integration schemes for the reactions would be appropriate. In theory, this would make sense only if we were able to produce a fully third-order algorithm for integrating the Luo–Rudy I model equations. In order to produce a third-order algorithm for the Luo–Rudy I model, it would be necessary to use twice continuously differentiable splines to represent $\mathbf{a}(V)$ and $\mathbf{b}(V)$ (to increase the regularity of the solution of the differential equations), to use third-order integration of diffusion and third-order operator splitting of reaction and diffusion. If the computational domain is rectangular, then it would also be necessary to perform local refinement near the corners to prevent degradation of accuracy due to restricted regularity of the solution near the corners; alternatively, we could modify the finite element method for the diffusion to include basis functions that represent the non-smooth behavior near the corners.

In practice, sometimes accuracy is a more important consideration than order. In other words, it is possible that a reaction integrator with higher than second-order accuracy might obtain the desired absolute accuracy with less work than our SDIRK scheme. Such decisions depend on other factors, such as the choice of mesh width and time-step. For example, with relatively coarse meshes the propagating reaction front will be very sharp, and the reaction integration will have to deal with very large changes in the solution in one time-step. On the other hand, with relatively fine meshes the diffusion will spread the reaction front over a moderate number of grid cells, and the reaction integration will see much smaller changes in the solution in one time-step.

We have implemented a Richardson extrapolation of our SDIRK integration, in order to achieve a desired accuracy, possibly through high order. Richardson extrapolation involves recursive time-step halving, so the work goes up substantially with each extrapolation. For example, the second-order SDIRK integration requires two nonlinear system solves, so the third-order extrapolant requires an additional four nonlinear solves, fourth-order requires an additional eight nonlinear solves, and so on. Numerical exper-

iments indicate that these extrapolants almost always achieve the desired order of accuracy; only abrupt changes in the stimulus current or occasional straddling of spline interpolation points prevents high order in extrapolation. The reason for using the Richardson extrapolation with the SDIRK scheme is to test the usefulness of higher-order reaction integration. We will discuss this issue further with our numerical results in Section 8.

In order to determine an appropriate tolerance for the reaction integration, we first determined a model for the error due to operator splitting and spatial discretization of the diffusion equation. In other words, we chose very small tolerances for the reaction integration, for the nonlinear system solves and for the iterative solution of the linear systems for the diffusion equation. We performed mesh refinement studies to see how the error behaved as a function of the mesh width. (See Section 8.1 for a description of this mesh refinement study.) This gave us the following empirical formula for the reaction tolerance:

$$\text{tolerance} = \min\{10^{-3},\ \max\{10^4\epsilon, 40\Delta x^2\}\},$$

where ϵ is machine roundoff. In other words, Newton iteration is continued until

$$|\mathbf{f}_2(\mathbf{k})| \leqslant \text{tolerance}\max\{|\mathbf{k}_2|, |R_2|\},$$

and Richardson extrapolation of the SDIRK scheme is continued until

$$|\Delta V| \leqslant \text{tolerance}|V|,$$

where ΔV is the estimated error in the Richardson extrapolation for the membrane potential V.

5. Integration of diffusion

In order to integrate the diffusion Eq. (2b) in second-order operator splitting, we will develop a finite element discretization that is second-order accurate in both space and time. On a grid without local refinement, we will use piecewise-linear functions for spatial discretization and Crank–Nicolson for temporal discretization. This approach is described in Sections 5.2 and 5.3. With local refinement refinement, we will use mortar elements to form the composite grid equations in Section 5.4.

5.1. Weak form on the full domain

Given initial data $V_0 \in H^0(\Omega)$, the weak form of the diffusion problem (2b) is to find $V(x,t)$ so that $\forall t > 0 V(\cdot,t) \in H^1(\Omega)$, $V(\cdot,0) \in L^2(\Omega)$ and

$$\begin{aligned} &\forall w \in H^1(\Omega), \quad 0 = \frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega w(x)V(x,t)\,\mathrm{d}x + \int_\Omega \nabla_x w(x)\cdot D(x)\nabla_x V(x,t)\,\mathrm{d}x \equiv \frac{\mathrm{d}}{\mathrm{d}t}(w,V) + B(w,V),\\ &\forall w \in L^2(\Omega), \quad 0 = (w, V - V_0). \end{aligned} \tag{4}$$

Here, $H^1(\Omega)$ represents the usual Sobolev space, consisting of functions whose derivatives of order up to one are square-integrable on Ω. We assume that in any number of dimensions, the diffusion coefficient is bounded and uniformly positive-definite:

$$\exists \overline{D} \geqslant \underline{D} > 0 \quad \forall x \in \Omega,\ \forall y \neq 0,\ \underline{D} \leqslant \frac{y^{\mathrm{T}} D(x) y}{y^{\mathrm{T}} y} \leqslant \overline{D}.$$

As a result, the bilinear form B is self-adjoint, bounded and coercive on $H^1(\Omega)$. This problem is known to be equivalent to

$$\frac{\partial V}{\partial t} = \nabla_x \cdot D(\mathbf{x}) \nabla_x V \quad \forall \mathbf{x} \in \Omega,\ \forall t > 0,$$

$$\mathbf{n} \cdot \nabla_x V = 0 \quad \forall \mathbf{x} \in \partial\Omega,\ \forall t > 0,$$

$$V(\mathbf{x}, t) = V_0(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega,$$

provided that the initial data V_0, the diffusion coefficient D and the domain Ω are sufficiently smooth.

5.2. Galerkin methods on the full domain

To integrate the diffusion equation in the operator-splitting subproblem (2b), we use a conforming finite element method. An excellent reference for this technique applied to diffusion problems is [50]. We begin by choosing a finite-dimensional subspace $M_h \subset H^1(\Omega)$; our choice of M_h is described below. The semi-discrete Galerkin approximation is a function $V_h(x,t) \in M_h$ for all $t > 0$ that satisfies the weak equations

$$\begin{aligned} \forall W_h \in M_h, \quad 0 &= \frac{\mathrm{d}}{\mathrm{d}t} \int_\Omega W_h(x) V_h(x,t)\,\mathrm{d}x + \int_\Omega \nabla_x W_h(x) \cdot D(x) \nabla_x V_h(x,t)\,\mathrm{d}x \\ &\equiv \frac{\mathrm{d}}{\mathrm{d}t}(W_h, V_h) + B(W_h, V_h), \end{aligned} \tag{5a}$$

$$\forall W_h \in M_h, \quad 0 = (W_h, V_0 - V_h). \tag{5b}$$

In (5b), we have used the L^2-projection of the initial data for the semi-discrete problem. Note that if $V_0 \in M_h$, then the L^2-projection implies that $V_h(x,0) = V_0(x)$.

A fully discrete Galerkin approximation uses an appropriate stiffly stable method for solving initial value problems for ordinary differential equations. A finite element method for approximating the solution of (4) is a Galerkin method that chooses the finite-dimensional subspace M_h to consist of piecewise polynomials on some mesh of width h. In our applications, we will choose M_h to consist of continuous functions that are linear in each coordinate on some given rectangular mesh. We will let h denote the maximum cell width in any coordinate direction. It is well known that for quasi-uniform meshes, M_h approximates functions in $H^2(\Omega)$ with order h^2. We will use the trapezoidal rule to integrate in time; this leads to the familiar Crank–Nicolson method for the diffusion equation:

$$\forall W_h \in M_h, \quad 0 = \frac{1}{\Delta t}(W_h, V_h(\cdot, t^{n+1}) - V_h(\cdot, t^n)) + \frac{1}{2} B(W_h, V_h(\cdot, t^{n+1}) + V_h(\cdot, t^n)), \tag{6a}$$

$$\forall W_h \in M_h, \quad 0 = (W_h, V_0 - V_h(\cdot, 0)). \tag{6b}$$

It is possible to show [50] that the error in the fully discrete method satisfies

$$\begin{aligned} \|V(\cdot,t) - V_h(\cdot,t)\|_{H^0(\Omega)} \leqslant C \Bigg[& \mathrm{e}^{-\lambda_1 t} \|V(\cdot,0) - V_h(\cdot,0)\|_{H^0(\Omega)} + h^2 \int_0^t \left\| \frac{\partial V}{\partial t}(\cdot,s) \right\|_{H^2(\Omega)} \mathrm{d}s \\ & + \Delta t^2 \int_0^t \left\| \frac{\partial^3 V}{\partial t^3}(\cdot,s) \right\|_{H^0(\Omega)} \mathrm{d}s + \Delta t^2 \int_0^t \left\| \frac{\partial^2 \nabla_x \cdot D \nabla_x V}{\partial t^3}(\cdot,s) \right\|_{H^0(\Omega)} \mathrm{d}s \Bigg]. \end{aligned}$$

Here, C is a constant depending on Ω and D, and λ_1 is the smallest eigenvalue of $-\nabla_x \cdot D\nabla_x$ on Ω. The scheme is basically second-order in space and time for $t = \mathrm{O}(1)$; at early time, the order of the error depends on the accuracy with which the initial data can be approximated by the finite element space. Note that the influence of the error in the initial data fades exponentially as time increases.

It is well known that the weak equations using Crank–Nicolson time integration lead to linear systems of the form

$$\left(\mathbf{M} + \mathbf{K}\frac{\Delta t^{n+\frac{1}{2}}}{2}\right)\mathbf{V}^{n+1} = \left(\mathbf{M} - \mathbf{K}\frac{\Delta t^{n+\frac{1}{2}}}{2}\right)\mathbf{V}^{n}. \tag{7}$$

Here, $\mathbf{M}$ is the mass matrix and $\mathbf{K}$ is the stiffness matrix. We will develop the details of these matrices in Section 5.3.

5.3. Numerical implementation

In 1D, note that the nonzero contributions to the mass or stiffness matrices from element (x_k, x_{k+1}) only involve nodal basis functions W_k and W_{k+1}. For piecewise constant diffusion coefficient D, the integrals can be performed exactly; otherwise, it is sufficient to approximate the integrals using a one-point Gaussian quadrature rule (i.e., the midpoint rule). This suggests that we form the diffusive quadratures

$$\begin{aligned}
\begin{bmatrix} q_0 \\ q_1 \end{bmatrix}_{k+\frac{1}{2}} &\equiv \int_{x_k}^{x_{k+1}} \begin{bmatrix} W_k(x) \\ W_{k+1}(x) \end{bmatrix} [W_k(x) W_{k+1}(x)]\,\mathrm{d}x \begin{bmatrix} v_k^{n+1} - v_k^n \\ v_{k+1}^{n+1} - v_{k+1}^n \end{bmatrix} \\
&\quad + \Delta t^{n+\frac{1}{2}} \int_{x_k}^{x_{k+1}} \begin{bmatrix} \frac{\partial W_k}{\partial x} \\ \frac{\partial W_{k+1}}{\partial x} \end{bmatrix} D(x) \begin{bmatrix} \frac{\partial W_k}{\partial x} & \frac{\partial W_{k+1}}{\partial x} \end{bmatrix} \mathrm{d}x \begin{bmatrix} \frac{v_k^{n+1} + v_k^n}{2} \\ \frac{v_{k+1}^{n+1} + v_{k+1}^n}{2} \end{bmatrix} \\
&= \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} v_k^{n+1} - v_k^n \\ v_{k+1}^{n+1} - v_{k+1}^n \end{bmatrix} \frac{\Delta x_{k+\frac{1}{2}}}{6} + \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} v_k^{n+1} + v_k^n \\ v_{k+1}^{n+1} + v_{k+1}^n \end{bmatrix} \frac{D_{k+\frac{1}{2}} \Delta t^{n+\frac{1}{2}}}{2\Delta x_{k+\frac{1}{2}}} \\
&\equiv \mathbf{M}_{k+\frac{1}{2}} \begin{bmatrix} v_k^{n+1} - v_k^n \\ v_{k+1}^{n+1} - v_{k+1}^n \end{bmatrix} + \mathbf{K}_{k+\frac{1}{2}} \frac{\Delta t^{n+\frac{1}{2}}}{2} \begin{bmatrix} v_k^{n+1} + v_k^n \\ v_{k+1}^{n+1} + v_{k+1}^n \end{bmatrix}.
\end{aligned} \tag{8}$$

The finite element equations at internal nodes take the form

$$(q_0)_{k+\frac{1}{2}} + (q_1)_{k-\frac{1}{2}} = 0. \tag{9}$$

In general, the sum, of the quadratures associated with a given node, over elements within the computational domain is zero. The mass matrix $\mathbf{M}$ and stiffness matrix $\mathbf{K}$ in Eq. (7) are formed by combining the elementwise mass and stiffness matrices from Eq. (8) into the finite element Eq. (9).

In 2D, the non-zero contributions to the mass or stiffness matrices from element $(x_k, x_{k+1}) \times (y_\ell, y_{\ell+1})$ involve only nodal basis functions $W_{k,\ell}$, $W_{k+1,\ell}$, $W_{k,\ell+1}$ and $W_{k+1,\ell+1}$. For piecewise constant diffusion coefficient D, the integrals can be performed exactly. This suggests that we form the diffusive quadratures

$$
\begin{aligned}
\begin{bmatrix} q_{00} \\ q_{10} \\ q_{01} \\ q_{11} \end{bmatrix}_{k+\frac{1}{2},\ell+\frac{1}{2}}
&\equiv \int_{y_\ell}^{y_{\ell+1}} \int_{x_k}^{x_{k+1}} \begin{bmatrix} W_{k,\ell} \\ W_{k+1,\ell} \\ W_{k,\ell+1} \\ W_{k+1,\ell+1} \end{bmatrix} [W_{k,\ell} W_{k+1,\ell} W_{k,\ell+1} W_{k+1,\ell+1}] \,\mathrm{d}x \begin{bmatrix} v_{k,\ell}^{n+1} - v_{k,\ell}^{n} \\ v_{k+1,\ell}^{n+1} - v_{k+1,\ell}^{n} \\ v_{k,\ell+1}^{n+1} - v_{k,\ell+1}^{n} \\ v_{k+1,\ell+1}^{n+1} - v_{k+1,\ell+1}^{n} \end{bmatrix} \\
&\quad + \frac{\Delta t^{n+\frac{1}{2}}}{2} \int_{y_\ell}^{y_{\ell+1}} \int_{x_k}^{x_{k+1}} \begin{bmatrix} (\nabla_x W_{k,\ell})^{\mathrm{T}} \\ (\nabla_x W_{k+1,\ell})^{\mathrm{T}} \\ (\nabla_x W_{k,\ell+1})^{\mathrm{T}} \\ (\nabla_x W_{k+1,\ell+1})^{\mathrm{T}} \end{bmatrix} D [\nabla_x W_{k,\ell} \nabla_x W_{k+1,\ell} \nabla_x W_{k,\ell+1} \nabla_x W_{k+1,\ell+1}] \,\mathrm{d}x \,\mathrm{d}y \\
&\quad \times \begin{bmatrix} v_{k,\ell}^{n+1} + v_{k,\ell}^{n} \\ v_{k+1,\ell}^{n+1} + v_{k+1,\ell}^{n} \\ v_{k,\ell+1}^{n+1} + v_{k,\ell+1}^{n} \\ v_{k+1,\ell+1}^{n+1} + v_{k+1,\ell+1}^{n} \end{bmatrix} \\
&= \begin{bmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 2 & 2 \\ 2 & 2 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{bmatrix} \begin{bmatrix} v_{k,\ell}^{n+1} - v_{k,\ell}^{n} \\ v_{k+1,\ell}^{n+1} - v_{k+1,\ell}^{n} \\ v_{k,\ell+1}^{n+1} - v_{k,\ell+1}^{n} \\ v_{k+1,\ell+1}^{n+1} - v_{k+1,\ell+1}^{n} \end{bmatrix} \frac{\Delta x_{k+\frac{1}{2}} \Delta y_{\ell+\frac{1}{2}}}{36} \\
&\quad + \frac{\Delta t^{n+\frac{1}{2}}}{2} \left\{ \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_{k+\frac{1}{2}} & 0 \\ 0 & \Delta y_{\ell+\frac{1}{2}} \end{bmatrix}^{-1} \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix} \begin{bmatrix} \Delta x_{k+\frac{1}{2}} & 0 \\ 0 & \Delta y_{\ell+\frac{1}{2}} \end{bmatrix}^{-1} \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \right. \\
&\quad \left. + \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \frac{\mathrm{tr}\left(\begin{bmatrix} \Delta x_{k+\frac{1}{2}} & 0 \\ 0 & \Delta y_{\ell+\frac{1}{2}} \end{bmatrix}^{-1} \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix} \begin{bmatrix} \Delta x_{k+\frac{1}{2}} & 0 \\ 0 & \Delta y_{\ell+\frac{1}{2}} \end{bmatrix}^{-1} \right)}{3} [1 \quad -1 \quad -1 \quad 1] \right\} \\
&\quad \times \begin{bmatrix} v_{k,\ell}^{n+1} + v_{k,\ell}^{n} \\ v_{k+1,\ell}^{n+1} + v_{k+1,\ell}^{n} \\ v_{k,\ell+1}^{n+1} + v_{k,\ell+1}^{n} \\ v_{k+1,\ell+1}^{n+1} + v_{k+1,\ell+1}^{n} \end{bmatrix} \frac{\Delta x_{k+\frac{1}{2}} \Delta y_{\ell+\frac{1}{2}}}{4} \\
&\equiv \mathbf{M}_{k+\frac{1}{2},\ell+\frac{1}{2}} \begin{bmatrix} v_{k,\ell}^{n+1} - v_{k,\ell}^{n} \\ v_{k+1,\ell}^{n+1} - v_{k+1,\ell}^{n} \\ v_{k,\ell+1}^{n+1} - v_{k,\ell+1}^{n} \\ v_{k+1,\ell+1}^{n+1} - v_{k+1,\ell+1}^{n} \end{bmatrix} + \mathbf{K}_{k+\frac{1}{2},\ell+\frac{1}{2}} \frac{\Delta t^{n+\frac{1}{2}}}{2} \begin{bmatrix} v_{k,\ell}^{n+1} + v_{k,\ell}^{n} \\ v_{k+1,\ell}^{n+1} + v_{k+1,\ell}^{n} \\ v_{k,\ell+1}^{n+1} + v_{k,\ell+1}^{n} \\ v_{k+1,\ell+1}^{n+1} + v_{k+1,\ell+1}^{n} \end{bmatrix}. \qquad (10)
\end{aligned}
$$

In these equations, "tr" denotes the trace of a matrix. The finite element equations at internal nodes then take the form

$$(q_{00})_{k+\frac{1}{2},\ell+\frac{1}{2}} + (q_{10})_{k-\frac{1}{2},\ell+\frac{1}{2}} + (q_{01})_{k+\frac{1}{2},\ell-\frac{1}{2}} + (q_{11})_{k-\frac{1}{2},\ell-\frac{1}{2}} = 0. \tag{11}$$

In general the sum, over elements within the computational domain, of the quadratures associated with a given node is zero. The mass matrix $\mathbf{M}$ and stiffness matrix $\mathbf{K}$ in Eq. (7) are formed by combining the elementwise mass and stiffness matrices from Eq. (10) into the finite element Eq. (11).

5.4. Weak form with sub-domains

In AMR, we will use a hierarchy to define our computational grid. Our AMR algorithm will be recursive, always working with at most two grids in any region of space. Thus, it will suffice locally to assume that our computational domain Ω satisfies $\overline{\Omega} = \overline{\Omega_f} \cup \overline{\Omega_c}$, where Ω_f and Ω_c are disjoint open sets (the fine and coarse domains). Let $S = \partial\Omega_f \cap \partial\Omega_c$ be the interface between the two domains, and let $\mu_f = \mathbf{n}_f \cdot D\nabla_x V$ be the diffusive flux, where $\mathbf{n}_f$ is the unit outer normal on Ω_f. We can introduce a Lagrange multiplier λ_f to force continuity of the solutions on the two domains and develop the following weak formulation of the diffusion equation:

$$\forall w_f \in H^1(\Omega_f) \quad \frac{d}{dt}\int_{\Omega_f} w_f V_f \, dx + \int_{\Omega_f} \nabla_x w_f \cdot D\nabla_x V_f \, dx - \int_S w_f \mu_f \, dx = 0, \tag{12a}$$

$$\forall w_c \in H^1(\Omega_c) \quad \frac{d}{dt}\int_{\Omega_c} w_c V_c \, dx + \int_{\Omega_c} \nabla_x w_c \cdot D\nabla_x V_c \, dx + \int_S w_c \mu_f \, dx = 0, \tag{12b}$$

$$\forall \lambda_f \in H^{-\frac{1}{2}}(S) \quad \int_S \lambda_f(-V_f + V_c) \, dx = 0. \tag{12c}$$

If we choose finite-dimensional subspaces $M_f \subset H^1(\Omega_f)$, $M_c \subset H^1(\Omega_c)$ and $\Lambda_f \subset H^{-\frac{1}{2}}(S)$, we can use the weak formulation on sub-domains to develop fully discrete finite element equations. The resulting fully discrete system will have the form

$$\left(\mathbf{M}_f + \mathbf{K}_f \frac{\Delta t^{n+\frac{1}{2}}}{2}\right)\mathbf{V}_f^{n+1} - \mathbf{C}_f \mathbf{m}_f^{n+\frac{1}{2}} \Delta t^{n+\frac{1}{2}} = \left(\mathbf{M}_f - \mathbf{K}_f \frac{\Delta t^{n+\frac{1}{2}}}{2}\right)\mathbf{V}_f^{n}, \tag{13a}$$

$$\left(\mathbf{M}_c + \mathbf{K}_c \frac{\Delta t^{n+\frac{1}{2}}}{2}\right)\mathbf{V}_c^{n+1} - \mathbf{C}_c \mathbf{m}_c^{n+\frac{1}{2}} \Delta t^{n+\frac{1}{2}} = \left(\mathbf{M}_c - \mathbf{K}_c \frac{\Delta t^{n+\frac{1}{2}}}{2}\right)\mathbf{V}_c^{n}, \tag{13b}$$

$$-\mathbf{C}_f^{\mathrm{T}} \mathbf{V}_f^{n+1} + \mathbf{C}_c^{\mathrm{T}} \mathbf{V}_c^{n+1} = 0. \tag{13c}$$

The specific form of these equations is dimensionally dependent and will be elaborated below in Eqs. (14) and (16). We will refer to the entries of $\mathbf{m}_f^{n+\frac{1}{2}}$ as the mortars [9].

In 1D, we will take the finite-dimensional subspace Λ_f of Lagrange multipliers to be the set of all scalar values at x_i. Suppose that $x_i = x_I \in S$ is a node shared by the fine and coarse domains, with the fine domain to the left of the coarse. With piecewise linear elements, our discrete equations take the form

$$\begin{aligned}
&(q_1)_{i-\frac{1}{2}} - \mu_i = 0,\\
&(q_0)_{I+\frac{1}{2}} + \mu_I = 0,\\
&-\upsilon_i + \upsilon_I = 0.
\end{aligned} \tag{14}$$

The quadratures q_0, q_1 in these equations were defined in Eq. (8). In addition, we have incorporated the time-step $\Delta t^{n+\frac{1}{2}}$ into the mortar μ_i. The third equation enforces continuity between the coarse and fine grid solutions and the other two equations suggest that we define the coarse grid mortar at node $x_I = x_i \in S$ in terms of the fine mortar by

$$\mu_I = \mu_i.$$

These equations say that the composite grid equation at $x_i = x_I \in S$ is

$$(q_0)_{I+\frac{1}{2}} + \mu_I = 0, \quad \text{where } \mu_I = \mu_i = (q_1)_{i-\frac{1}{2}}.$$

Note that $(q_0)_{I+\frac{1}{2}}$ is a linear function of the coarse grid unknowns, and $(q_1)_{i-\frac{1}{2}}$ is a linear function of the fine grid unknowns and the coarse interface unknown υ_I (since continuity requires that $\upsilon_i = \upsilon_I$). Similar linearity conditions apply at other grid nodes. In general, the composite grid equations have the form of a linear system

$$\begin{bmatrix} \mathbf{A}_{\mathrm{ff}} & \mathbf{A}_{\mathrm{fc}} \\ \mathbf{A}_{\mathrm{cf}} & \mathbf{A}_{\mathrm{cc}} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\mathrm{f}} \\ \mathbf{u}_{\mathrm{c}} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{\mathrm{f}} \\ \mathbf{b}_{\mathrm{c}} \end{bmatrix}, \tag{15}$$

where $\mathbf{u}_{\mathrm{f}}$ is the vector of unknowns on the fine grid and $\mathbf{u}_{\mathrm{c}}$ is the vector of unknowns on the coarse grid.

In 2D, we will take Λ_{f} to be the set of all discontinuous piecewise linear functions on the fine mesh restricted to S. Our basis functions in Λ_{f} will be chosen to be dual to the piecewise linear basis functions in M_{f}, restricted to S. As an example, suppose that S consists of a line segment parallel to the y-axis. Then, if (x_i, y_j) is a boundary node, we want

$$\int_{y_0}^{y_N} V_{i,j}(x_i, y)\lambda_{j'}(y)\,\mathrm{d}y = \frac{1}{2}\delta_{j,j'} \begin{cases} y_{j+1} - y_{j-1}, & 0 < j < N, \\ y_1 - y_0, & j = 0, \\ y_N - y_{N-1}, & j = N. \end{cases} \tag{16}$$

We will break S into intervals associated with the fine mesh and assume that fine mortars at the endpoints of S are identical with the coarse grid values there. In this case, it is easy to see that the dual basis functions at nodes j interior to S are

$$\lambda_j(y) = \begin{cases} 2 - 3\frac{y-y_j}{y_{j+1}-y_j}, & y_j < y < y_{j+1}, \\ 2 + 3\frac{y-y_j}{y_j-y_{j-1}}, & y_{j-1} < y < y_j, \\ 0, & \text{otherwise.} \end{cases}$$

The dual basis functions at the end nodes of S are the indicator functions for the end elements. With this choice for the basis functions in Λ_{f}, we find that $\mathbf{C}_{\mathrm{f}}$ is diagonal.

Suppose that Ω_{f} is to the left of S at fine node (x_i, y_j), and that the coarse domain lies to the right of coarse node (x_I, y_J), where $x_i = x_I$. Also suppose that exactly two fine grid sides of equal length are contained in each coarse grid side within S. (See Fig. 1.) Then, our discrete equations are:

$$(q_{11})_{i-\frac{1}{2},j-\frac{1}{2}} + (q_{10})_{i-\frac{1}{2},j+\frac{1}{2}} - \mu_{i,j} = 0,$$

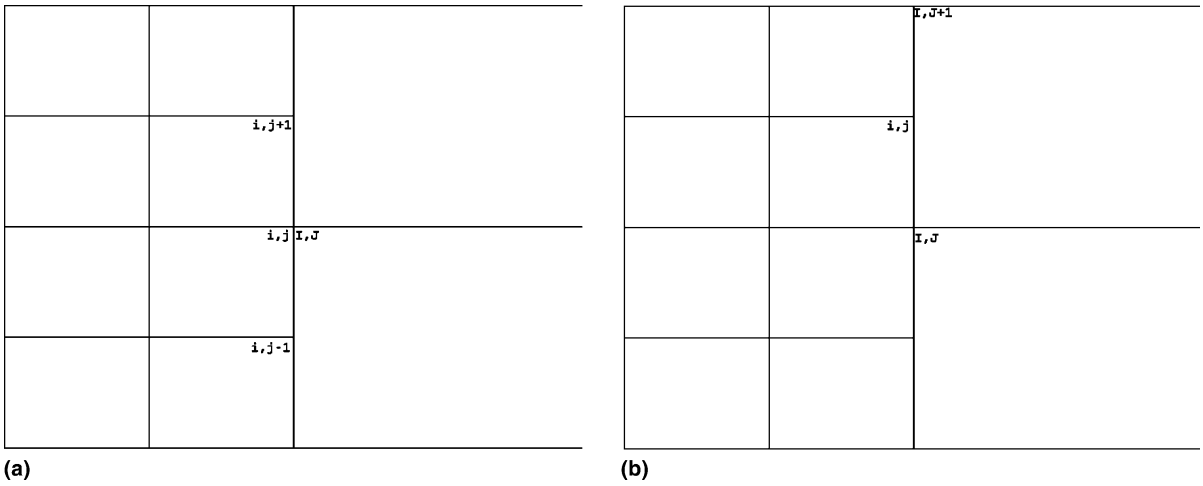

Fig. 1. Example of interface S between coarse and fine grids: (a) fine corner at coarse corner, (b) fine corner between coarse corners.

$$(q_{01})_{I+\frac{1}{2},J-\frac{1}{2}} + (q_{00})_{I+\frac{1}{2},J+\frac{1}{2}} + \tfrac{1}{2}\mu_{i,j-1} + \mu_{i,j} + \tfrac{1}{2}\mu_{i,j+1} = 0, \quad \text{where } y_j = y_J,$$

$$-v_{i,j} + \begin{cases} v_{I,J}, & y_j = y_J, \\ \frac{1}{2}(v_{I,J} + v_{I,J+1}), & y_{j+1} = y_J \end{cases} = 0.$$

Here, we have incorporated the time-step $\Delta t^{n+\frac{1}{2}}$ and the mesh length into the mortars $\mu_{i,j}$. Similar equations hold at corners of the domains, or at physical boundaries. In general, the continuity condition implies that the fine grid nodal values on S are equal to the restriction of the coarse basis functions to S; in the current example,

$$v_{i,j} = \begin{cases} v_{I,J}, & y_j = y_J, \\ \frac{1}{2}(v_{I,J} + v_{I,J+1}), & y_{j+1} = y_J. \end{cases}$$

The other equations imply that at a coarse node in S we should define the coarse grid mortar to be the fine grid mortar at that same node, plus half the sum of the fine grid mortars at neighboring fine grid nodes in S. In the current example, if $(x_i, y_j) = (x_I, y_J)$, then

$$\mu_{I,J} = \tfrac{1}{2}\mu_{i,j-1} + \mu_{i,j} + \tfrac{1}{2}\mu_{i,j+1}.$$

These give us equations to determine fine grid boundary values from coarse grid values and to determine coarse grid mortars from fine grid mortars. These equations say that the composite grid equation at $(x_i, y_j) = (x_I, y_J) \in S$ is

$$\begin{aligned} &(q_{01})_{I+\frac{1}{2},J-\frac{1}{2}} + (q_{00})_{I+\frac{1}{2},J+\frac{1}{2}} + \mu_{I,J} = 0, \quad \text{where} \\ &\mu_{I,J} = \tfrac{1}{2}\mu_{i,j-1} + \mu_{i,j} + \tfrac{1}{2}\mu_{i,j+1} \\ &\quad = \tfrac{1}{2}\left((q_{11})_{i-\frac{1}{2},j-\frac{3}{2}} + (q_{10})_{i-\frac{1}{2},j-\frac{1}{2}} \right) + (q_{11})_{i-\frac{1}{2},j-\frac{1}{2}} + (q_{10})_{i-\frac{1}{2},j+\frac{1}{2}} \\ &\qquad + \tfrac{1}{2}\left((q_{11})_{i-\frac{1}{2},j+\frac{1}{2}} + (q_{10})_{i-\frac{1}{2},j+\frac{3}{2}} \right). \end{aligned}$$

Note that the coarse quadratures are linear functions of the coarse grid unknowns, and the fine quadratures are linear functions of the fine grid unknowns and the coarse interface unknowns (since continuity requires that the fine grid interface solution is the restriction of the coarse solution). Similar linearity conditions apply at other grid nodes. Recall that the composite grid equations have the form (15), where $\mathbf{u}_f$ is the vector of unknowns on the fine grid and $\mathbf{u}_c$ is the vector of unknowns on the coarse grid. The first block equation in this system represents the finite element equations at fine grid unknowns and the second block equation represents the finite element equations at coarse grid nodes, include nodes in the coarse-fine interface S.

Since the AMR grids are designed so that coarse grid cells are aligned with fine grid cells, the mortar equations produce the same composite grid system as the conforming finite element method itself. The mortars serve the purpose of organizing the communication between grids in the AMR hierarchy.

6. Adaptive mesh refinement

There are various forms of adaptive mesh refinement (AMR), ranging from static refinement [1,46] to dynamic refinement in space [12,13,16,20–26] and to dynamic refinement in both space and time [5,8,14,30,51]. Our approach follows, as closely as possible, the basic principles developed by Berger and Colella [5,8].

In this paper, we will describe an AMR algorithm that dynamically selects refinement in both space and time. It will use a nested hierarchy of grids, each selecting appropriate time-steps chosen for needs of local accuracy and synchronization with coarser computations. The grid at a given level of refinement within the hierarchy will be a union of logically rectangular arrays of grid cells. These rectangular arrays of grid cells are called patches and form the basic organizational unit for computation. Communication between patches is regular and small compared to the work within the patch, so the algorithm extends nicely to distributed computation.

We assume that we are given some initial coarse mesh. We will call this mesh the coarsest level of refinement and denote it by L_0. Finer levels L_ℓ for $\ell > 0$ are defined recursively from the coarser level $L_{\ell-1}$.

We assume that each level L_ℓ consists of a finite array of patches, each of which is a logically rectangular array of cells. Here, "logically rectangular" means that the array of cells can be mapped to a rectangular grid by a continuous coordinate transformation. The grid patches themselves can be non-rectangular in space; we only require that each patch be rectangular as a data array.

We assume that if a coarse cell is refined in any part of its physical space, then it is refined everywhere. As a result, the boundary of any fine patch on level L_ℓ coincides with the boundary of a logically rectangular array of coarse grid cells on level $L_{\ell-1}$. This assumption can restrict the grid generation for curvilinear grids; see [3] for an alternative.

We assume that we are given an integer refinement ratio r. Whenever a coarse cell on level L_ℓ is refined, it is subdivided into r cells in each logical coordinate direction on level $L_{\ell+1}$. The refinement ratio is a power of 2 and is commonly chosen to be 2 or 4. Note that the assumptions of a fixed refinement ratio and of grid alignment imply that on any level L_ℓ with $\ell > 0$, in any patch, the number of cells in any coordinate direction is an integer multiple of the refinement ratio.

We assume that the union of fine patches on level $L_{\ell+1}$ is contained in the interior of the union of coarse patches on level L_ℓ. However, an individual fine patch is not required to lie inside any single coarse patch. Note that this assumption implies that the coarsest level L_0 must completely cover the entire physical domain. This in turn implies that we must be able to provide a logically rectangular grid on the coarsest level. This assumption can restrict the range of application of this form of AMR. For modifications of the AMR technique that extend to more general problem geometry, see [4,6].

After we advance the data on patches in a coarse level L_ℓ to some time, we assume that the data on patches in a finer level $L_{\ell+1}$ are advanced by as many time-steps as required by stability and accuracy to reach exactly the same time as the coarser level L_ℓ. This assumption implies that coarse patches are integrated before fine patches; it also implies that the time-stepping algorithm must be applied recursively within each time-step on all but the finest level.

6.1. Time-step selection for AMR

Since we are using stiffly stable integrators for reaction and diffusion, there is no a priori choice for the time-step on a given grid level. Since we are moving the mesh during AMR, however, we do need to make sure that the dynamical features requiring refinement do not move off the fine grid region before the next regridding event.

The Luo–Rudy I reaction–diffusion system can generate traveling waves with reasonably well-determined speeds. From numerical experiments, we have determined the front speed to be $v = 6.6$ cm/ms for our model parameters. Thus, on any level of mesh refinement, we require $\Delta t \leqslant \Delta x/v$. In order to enforce synchronization between fine and coarse computations, we also require the time-step on a fine level $L_{\ell+1}$ to be an integer multiple of the time-step on its coarser level L_ℓ. If the finer level can be refined, then the number of time-steps on the fine level must also be an integer multiple of 2, so that we can perform Richardson error estimation for regridding, if necessary.

In performing the integration routines recursively, first we determine the initial time-step size as described previously in Section 6.1. Then, we loop over time-steps until we synchronize with the coarser level, if it exists. Within each time-step, we use operator splitting to integrate the reaction and diffusion equations; this is described in Section 6.3. Afterwards, we use recursion to repeat the process on the finer levels. At appropriate times, we regrid the finer levels as described in Section 6.2. Finally, we determine the next time-step size.

6.2. Regridding

Because we are interested in solving time-dependent problems, we allow the mesh refinement to move in time. We use an error estimation procedure (described below) to determine where the unacceptably large errors occur on a given level. The tagged cells are organized into grid patches using the Berger–Rigoutsos algorithm [7]. The same process is used to generate the initial grid refinement as is used at later times. By using an error estimator, rather than a gradient detector, we are able to place mesh refinement where discontinuities in the variables are about to form, or where the algorithm is not able to produce second-order accuracy for some other reason, such as a lack of smoothness in the equation of state. Note that this error estimator will use a Richardson error estimation process, naturally combining errors due to both spatial and temporal differences. Since the time-step is related to the mesh width through the front speed, as described just above in Section 6.1, this error estimation process is natural. The use of this front speed is also crucial in determining new grid patches, which are required to contain the front until the next regridding event.

Let us describe how grid cells are selected for refinement. First, cells in the patches belonging to some given coarse level L_ℓ are tagged if their global integration error is too large. This procedure uses both Richardson extrapolation to estimate the local truncation error in the integration, and a simple device to estimate the number of time-steps to be performed on this level of refinement. This error estimation procedure is a standard procedure in the numerical integration of ordinary differential equations [17]. Suppose that at each time-step in cell i we commit an error of magnitude ϵ_i (principally the local truncation error); further, suppose that the computation permits a bound M on the growth of these errors. Note that the Crank–Nicolson scheme finite difference equation (7) shows that the computational solution essentially

amounts to applying a perturbation of the identity operator to the previous solution; it is therefore reasonable to expect M to be close to 1 for smooth flow and sufficiently fine mesh. Then the error $e_i^{(n)}$ in cell i at step n satisfies

$$e_i^{(0)} \leqslant \epsilon_i, \quad e_i^{(n)} \leqslant \epsilon_i + M e_i^{(n-1)} \quad \text{for } n > 1.$$

An argument by induction shows that

$$e_i^{(n)} \leqslant \epsilon_i \sum_{j=0}^{n-1} M^j = \epsilon_i \frac{M^n - 1}{M - 1}.$$

If $M \approx 1 + \mu\epsilon_i$ is close to one, then for small n the error bound will be approximately $n\epsilon_i$.

Suppose that the local truncation error satisfies

$$\epsilon_i = C\Delta t^{k+1},$$

where k is the expected global order of the scheme. (Note that our spatial and temporal error orders are equal.) Then, the error in taking one coarse step of size $\rho\Delta t$ is

$$e_{i,\mathrm{c}}^{(n)} \approx C\rho^{k+1}\Delta t^{k+1}.$$

On the other hand, if we take ρ fine time-steps of size Δt, the error is

$$e_{i,\mathrm{f}}^{(n)} \approx C\rho\Delta t^{k+1}.$$

If $\rho > 1$, then this allows us to estimate the local error of a fine time-step by

$$\epsilon_i \approx \frac{e_{i,\mathrm{c}}^{(n)} - e_{i,\mathrm{f}}^{(n)}}{\rho^{k+1} - \rho} = \frac{w_{i,\mathrm{c}}^{(n)} - w_{i,\mathrm{f}}^{(n)}}{\rho^{k+1} - \rho},$$

where w is the quantity being monitored for errors. This gives us a computable estimate for the local truncation error. The accumulated error over N time-steps can be estimated by multiplying the local error by the anticipated number of time-steps N. We can approximate $N \approx L/c\Delta t$, where L is some length scale associated with the problem, c is the speed of the reaction front (see Section 6.1) and Δt is the current time-step. As a result, cell i is tagged for refinement if the relative error satisfies

$$\frac{|w_{i,\mathrm{c}}^{(n)} - w_{i,\mathrm{f}}^{(n)}|}{\max_j |w_{j,\mathrm{f}}^{(n)}|} \frac{L}{(\rho^{k+1} - \rho)c\Delta t} > \text{tolerance}.$$

In the computations described later in this paper, we always chose the tolerance to be 0.1.

Note that error estimation is performed on pseudo-patches that potentially lie in index spaces between the current level of refinement and the next coarser level. This is because the pseudo-patches are coarsened by a factor of the regrid interval $\rho = 2$, and mesh refinement uses an integer multiple $r = 2$ or 4 of the regrid interval. This reduces the work in comparing the errors, compared to estimating the error on the current level and something even finer.

It is important to consider the implementation of this error estimation strategy on a recursively refined mesh. Note that errors on coarse and fine meshes are estimated at different times, namely at one step forward on each individual level. At first glance, the fact that these times are different would appear to be undesirable. However, the alternative of comparing errors on all levels at the same time actually leads to much wasted work and larger refined regions. This is because the error estimation on the coarse mesh places the refined cells where the disturbance will be moving, plus buffer cells. Thus it is only necessary to buffer by

$\rho - 1$ cells (where ρ is the regrid interval), since that is the number of time-steps that will be taken between the times when the errors are estimated, and when the mesh will next be moved. If the errors had been computed at the same times, then it would be necessary to buffer by ρ cells on each level. Furthermore, the error estimation would have to proceed through more than one time-step, with recursive calls to integration on finer levels in order to provide data for finer grids. Since the mesh is going to be moved, this is extra work being performed for data that are only going to be discarded.

6.3. Advancing the data

When we advance the data at a given level of refinement L_ℓ, we have available to us a grid hierarchy consisting of the current level of refinement and all coarser levels $L_{\ell-1}, \ldots, L_0$. We need to perform the steps of operator splitting for the reaction–diffusion system on level L_ℓ. This means that we react for a half-step, solve the diffusion equation for a full step and then react for a half-step. This process, together with recursive integration of finer levels and infrequent regridding, continues until we have completed that number of steps required for synchronization with a coarser level $L_{\ell-1}$.

There are important issues to examine in this process. One issue regards how to handle the initial reaction on the composite grid; this step provides the initial data for the diffusion step. We discuss the details of this initial reaction step in Section 6.3.1. Another issue concerns the iterative solution of the linear system for the diffusion; this computation is discussed in Section 7. Note that this linear system involves unknowns from patches on levels $L_\ell, \ldots, L_0$. A third issue involves the relationship of the second reaction to the synchronization of fine data with coarse; this is discussed in Section 6.3.2.

6.3.1. Initial operator-split reaction

We expect the error estimation process to select refinement in the neighborhoods of the strong reaction front. This is because the stiff reaction in the Luo–Rudy I problem is not near steady-state there, so the errors in integrating the reactions are largest in that region. Away from the reaction front, we expect the potential to be varying slowly near equilibrium values for the reaction; both the diffusion and the reaction will involve small changes in the solution of the equations.

This suggests that we can use time interpolation on coarser levels $L_{\ell-1}, \ldots, L_0$ to determine the initial data for the linear system on those levels. We admit that this initial data should be the result, via operator splitting, of the current (fine) half-step of reaction; the effect of the time interpolation will be to approximate the result at the half-time of both reaction and diffusion on the coarse grid. However, in regions away from the reaction front, the fast reactions are nearly at steady state and the diffusion is relatively small. As a result, the difference in computational results between the two choices of initial data is negligible, while the difference in computational cost may be substantial for more complicated heart models.

Whenever we take a time-step on level L_ℓ, we use the SDIRK scheme to compute the results after a half-step of reaction. Note that this SDIRK scheme involves a local time-stepping strategy (at each point in space) to guarantee a desired accuracy in the reaction integration (see Section 4.3.).

6.3.2. Final operator-split reaction and synchronization

After we solve the linear system for the diffusion, we perform a second half-step of reaction in order to obtain overall second-order accuracy for the operator splitting process. This second reaction is performed on the current level L_ℓ in the same way as it was performed before solving the diffusion equation. However, there are important differences in how the computations proceed on coarser levels $L_{\ell-1}, \ldots, L_0$.

In general, there is no need to perform the second reaction step on the coarser levels. These levels will use the results from their own coarse time-steps to determine, via time interpolation, the initial data for the next linear system for diffusion on the current level L_ℓ. The exception occurs when the current level reaches the

same time as its coarser level $L_{\ell-1}$, because it is necessary to synchronize the data between levels. At that point, it is necessary to perform a half-step of reaction on all synchronized levels.

We also need to adjust the coarse results near the boundary with the fine grid, in order to avoid boundary layers. The results of the reactions for synchronization replace the previous results from the reaction–diffusion–reaction step on the coarser levels. However, the iterative solution of the diffusion equation on the composite grid does not provide us with results for the solution on those portions of the coarse level $L_{\ell-1}$ overlying the current level L_ℓ. (Recall that the multigrid algorithm computes *increments* to the solution on coarser levels.) The coarse grid results in regions overlying the current level are updated by the following process.

We want to replace the coarse results with appropriately coarsened results from the current level wherever the coarse and fine grids overlap. Given our nodal finite element data organization, this is easy. The coarse finite element space is nested in the fine space, so we can copy the fine results from fine grid nodes to co-incidental coarse nodes. This corresponds to replacing coarse results with fine results on coarse particle paths at the end of the second reaction in operator splitting.

7. Iterative linear algebra

During each time-step on level L_ℓ, we solve a linear system for the Crank–Nicolson discretization of the diffusion equation. This linear system involves unknowns on levels $L_\ell, \ldots, L_0$. The initial data for this diffusion is provided by operator splitting on level L_ℓ, and by time interpolation on levels $L_{\ell-1}, \ldots, L_0$. All that remains in the description of the algorithm is the formulation and solution of the linear system for the diffusion equation on the hierarchical grid.

There are several parts to the description of this linear system. We use a preconditioned conjugate gradient iteration to solve the linear system on the composite grid. In Section 5.4 we described the formulation of the linear system on a composite grid, composed of grid cells chosen from the grid hierarchy. These composite grid equations were related to a simple application of a popular technique called mortar finite elements. That discussion determines how we compute the residual and matrix–vector multiplies needed in the conjugate gradient algorithm.

In Section 7.1 we will describe a multiplicative domain decomposition process used to precondition the conjugate gradient iteration. If there were no local refinement, this multiplicative domain decomposition would be what is commonly called a multigrid algorithm.

7.1. Multiplicative domain decomposition

We will describe the basic ideas of multiplicative domain decomposition in this section. For more details on the convergence of the algorithm, the reader can consult [11,12]. This multiplicative domain decomposition involves the same steps as multigrid: residual calculation, smoothing, restriction and prolongation. The residual calculation is completely determined by the composite grid equations in Section 5.4. The prolongation involves injection of coarse grid finite element function values into fine grid interiors and mortar continuity conditions on fine grid boundaries. The restriction involves the adjoints of the prolongation. The smoother operation is performed within a given level of refinement on a list of grid patches. It involves both non-overlapping additive domain decomposition between grid patches and Gauss–Seidel iteration within a grid patch.

Let us consider the simple case where we have two domains $\Omega_f \subset \Omega_c$. For example, Ω_f may correspond to a fine grid in an adaptive approximation to the solution of some partial differential equation and Ω_c may correspond to the coarse grid. Since the union of the grid patches in level$L_{\ell+1}$ are contained in the interior of the union of the grid patches in the coarser level L_ℓ, the general algorithm can be reduced to recursive

consideration of this two-level case. Specifically, there are no places in the composite grid where neighboring cells differ by more than one level of refinement.

Conceptually, we will think of the true unknown $\overline{u}_f$ and $\overline{u}_c$ on $\Omega_c \setminus \Omega_f$ as providing the best solution to the problem where it is available. Our composite grid linear system would then take the form (15).

Multiplicative domain decomposition is a particular iterative improvement algorithm for solving the composite grid equations (15). Without loss of generality, we will assume that our initial guess for the solution of the composite grid equations (15) is zero; if not, we apply the algorithm to the system with the same matrix and the initial residual as right-hand side. The multiplicative domain decomposition algorithm on level L_ℓ with $\ell > 0$ takes the form

$$
\begin{aligned}
&\text{if on finest level, compute the initial fine residual} \quad \mathbf{r}_f^{(0)} = -\mathbf{b}_f,\\
&\text{pre-smooth on } L_\ell \quad \mathbf{u}_f^{(1)} = -\mathbf{S}_f \mathbf{r}_f^{(0)},\\
&\text{compute the composite grid residual on } L_\ell \text{ and } L_{\ell-1} \quad \begin{bmatrix} \mathbf{r}_f^{(1)} \\ \mathbf{r}_c^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_f^{(0)} \\ -\mathbf{b}_c \end{bmatrix} + \begin{bmatrix} \mathbf{A}_{ff} \\ \mathbf{A}_{cf} \end{bmatrix} \mathbf{x}_f^{(1)},\\
&\text{restrict the residual from } L_\ell \text{ to } L_{\ell-1} \quad \mathbf{r}_{c\leftarrow f} = \mathbf{R}\mathbf{r}_f,\\
&\text{recursively apply the algorithm on } L_{\ell-1} \quad \begin{bmatrix} \mathbf{u}_{cf} \\ \mathbf{u}_c \end{bmatrix} = -\begin{bmatrix} \mathbf{V}_{ff} & \mathbf{V}_{fc} \\ \mathbf{V}_{cf} & \mathbf{V}_{cc} \end{bmatrix} \begin{bmatrix} \mathbf{r}_{c\leftarrow f} \\ \mathbf{r}_c^{(0)} \end{bmatrix},\\
&\text{prolong the fine grid correction from } L_{\ell-1} \text{ to } L_\ell \text{ and update} \quad \mathbf{u}_f^{(2)} = \mathbf{u}_f^{(1)} + \mathbf{P}\mathbf{u}_{cf},\\
&\text{compute the residual on } L_\ell \quad \mathbf{r}_f^{(2)} = \mathbf{r}_f^{(0)} + [\,\mathbf{A}_{ff} \quad \mathbf{A}_{fc}\,]\mathbf{u}_f^{(2)},\\
&\text{post-smooth on } L_\ell \quad \mathbf{u}_f = \mathbf{u}_f^{(2)} - \mathbf{S}_f^{\mathrm{T}} \mathbf{r}_f^{(2)}.
\end{aligned}
\tag{17}
$$

Here, $\mathbf{R}$ is an inter-grid transfer operator that maps values on the fine grid to values on the coarse grid. We choose $\mathbf{R}^{\mathrm{T}}$ to be injection from the coarse finite element space to the fine space.

On the coarsest level L_0, the multiplicative domain decomposition algorithm takes the form

$$\text{solve } \mathbf{A}\mathbf{u} = \mathbf{r}^{(0)}. \tag{18}$$

This system of equations corresponds to a discretization of the diffusion equation on the entire domain. The residual $r^{(0)}$ was computed when the composite grid residual was evaluated on the next finer level L_1. This linear system is relatively small and can be solved effectively by a conjugate gradient iteration in multiple dimensions, or by a direct solve in one dimension.

In our multiplicative domain decomposition algorithm, the smoother $\mathbf{S}_f$ corresponds to approximating the solution of the linear equations for the differential equation on the subdomain Ω_f with appropriate boundary conditions. At physical boundaries, these boundary conditions should be specified. If Ω_f is smaller than the physical domain, then it is necessary to select appropriate internal boundary conditions for the definition of $\mathbf{S}_f$. We will describe our choice for the smoother in Section 7.2.

7.2. AMR smoother

The only remaining piece of the iterative solution method is the smoother. As we have already mentioned, in 1D we use a direct solver for the smoother. This means that on any grid patch that does not touch the boundary of the physical domain, we have Dirichlet boundary data coming from the coarse grid, and we solve directly for the interior unknowns. On any grid patch that does touch the physical boundary, we have Neumann data at that boundary and must solve for the solution at the boundary.

In multiple dimensions, the smoother must take a different form. We do not want to use a direct solver for the smoother in multiple dimensions, because the work is not proportional to the number of unknowns. Furthermore, the boundary nodes for a grid patch in level L_ℓ may include nodes on the interface between levels L_ℓ and $L_{\ell-1}$, and nodes in the interior of the union of the patches on level L_ℓ. We need to choose a smoother that will perform well within the multiplicative domain decomposition algorithm and still allow distributed computation.

We have chosen to use a Gauss–Seidel iteration at patch nodes other than at interior patch boundaries. At nodes on interior patch boundaries that are also interior to the union of the fine patches, we use a Jacobi iteration. At nodes on the interface with a coarser grid, a boolean mask tells us to ignore the correction from the Jacobi iteration.

Unlike the Gauss–Seidel iteration, Jacobi iteration does not depend on the order of processing of the unknowns. This means that if all patches on some level of refinement compute the same residual at shared nodes, then distributed Jacobi iterations will compute the same correction to the solution on all patches.

Our approach to the smoother corresponds to an additive domain decomposition, in the following sense. Consider the case in which there are two patches on level L_ℓ. We could order the equations and unknowns to obtain a linear system of the form

$$\begin{bmatrix} \mathbf{A}_{11} & 0 & \mathbf{A}_{1i} \\ 0 & \mathbf{A}_{22} & \mathbf{A}_{2i} \\ \mathbf{A}_{i1} & \mathbf{A}_{i2} & \mathbf{A}_{ii} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_i \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_i \end{bmatrix}.$$

Here, $\mathbf{u}_1$ is the vector of unknowns in the first patch, other than unknowns on the interface between the two patches. Similarly, $\mathbf{u}_2$ is the appropriate vector of unknowns on the second patch and $\mathbf{u}_i$ is the vector of unknowns on the interface between the two patches. We perform a block-Jacobi iteration on this system, where Gauss–Seidel iteration is used on the first two block equations (for the interior unknowns on each patch), and Jacobi iteration is used on the last block (for the unknowns on the interface between the patches).

We remark that it is straightforward to verify that condition A of [10] is satisfied by the multigrid scheme described in this paper.

8. Numerical results

We have considered 1D and 2D simulations of electrical wave propagation in a model heart. All computations used a refinement ratio of 4, second-order operator splitting and the Crank–Nicolson scheme for time integration. The diffusion coefficient (membrane conductance/membrane capacitance) as chosen to be the constant $D(x) = 1.25 \times 10^{-3}$ cm^2/ms. The heart tissue was 6 cm long in each coordinate direction.

For the first millisecond, we applied a stimulus current of -100 μA/cm^2 along a band of tissue within 1/40 of the tissue width from the left-hand side. In two dimensions, from time 315 to 316 ms we applied a second stimulus of -50 μA/cm^2 in the lower left-hand quarter of the tissue. This generated spiral waves in 2D, and interesting AMR.

In order to hasten the relaxation of the tissue after the initial stimulus, we have modified three parameters in the Luo–Rudy I model. The fast sodium current, slow inward current and time-dependent potassium current were changed from the original model in [34] to be

$$I_{\mathrm{Na}}(\mathbf{g}, V) = 16m^3hj(V - 54.4),$$
$$I_{\mathrm{si}}(\mathbf{g}, [\mathrm{Ca}]_{\mathrm{i}}, V) = 0.052df(V - 7.7 + 13.0287 \log([\mathrm{Ca}]_{\mathrm{i}})),$$
$$I_{\mathrm{K_1}}(\mathbf{g}, V) = 0.423K_1(V + 87.26).$$

The changes involve only the leading coefficient in the right-hand side expressions.

Our coarsest grid consisted of 20 grid cells in each coordinate direction. Without refinement, this grid is too coarse and causes pinning of the reaction front. As the mesh is refined, the reaction front gains speed until a limiting value is reached. With 10–12 grid cells inside the reaction front, the computed front speed is accurate to about two digits.

8.1. Comparison with unsplit method

In order to verify that the operator splitting used in this paper produces reliable results, we also programmed an unsplit algorithm. Since the reactions in the Luo–Rudy I model are so stiff, it was useful to use implicit time integration. Since the reactions are nonlinear and our multigrid iteration is designed for linear systems, we did not program this unsplit scheme for use in AMR.

We discretized the diffusion in the Luo–Rudy I model in space and used the method of lines to integrate in time. We used the LSODE algorithm due to Hindmarsh [29] for time integration, with relative error tolerance of 10^{-10}. The LSODE algorithm uses linear multistep methods with variable time-step and order control. Since LSODE uses a band solver for linear systems in the nonlinear iterations, we restricted comparisons between this unsplit approach and our operator split approach to 1D. In order to save code development time in this comparison, we asked LSODE to compute Jacobians numerically. This meant that LSODE had to make extra evaluations of the right-hand side in the method of lines occasionally, in order to approximate the Jacobian. As a result, the unsplit code took roughly 48 times longer than our operator split code on a uniform grid with 1000 cells. The combined cost of the Jacobian evaluations and the banded system solver would have made 2D runs with this method of lines code prohibitively expensive.

In order to measure the convergence rates with the unsplit and split methods, we chose to examine the numerical location of the middle of the reaction front at 80 ms. Specifically, we looked at the numerical results at that time and linearly interpolated the membrane potential versus space values to determine the spatial location of $V_m = -36.75$. The unsplit scheme was run with 62, 125, 250, 500 and 1000 grid cells, and the results from the two finest simulations were extrapolated to produce the predicted limiting position of the reaction front. Afterward, we computed the differences between the simulated position of the reaction front, and the extrapolated value. We performed the same computations with our operator split code, except that we added simulations with 2000 and 4000 grid cells. Both methods appear to be converging to the same front location at this time, as illustrated in Fig. 2, although the unsplit results approach from the left and the operator split results approach from the right. Figs. 3 and 4 shows the errors in the simulated reaction front location with these two methods. Both methods appear to be second-order accurate.

8.2. Numerical results in 1D

The wave propagation was simulated for a total of 120 ms in 1D. This is slightly more than the amount of time it takes for the reaction front due to the initial stimulus to cross the tissue. After this time, the AMR algorithm removes all refinement, and takes large and efficient time-steps on the coarsest level L_0 only. If we had run the comparisons between uniform grid and AMR simulations to later time, we would have made the uniform grid computations look comparatively worse. In Fig. 5 we show some computational results for uniform and AMR simulations in 1D. These show the jump in membrane potential propagating to the right, with good agreement between uniform and AMR computations.

In 1D it is possible to run very detailed uniform grid computations in a reasonable amount of time. A uniform grid computation with 5120 cells took 907 s, or slightly more than 15 min, on our Dec Alpha machine. The corresponding AMR simulation took 124 s, or slightly more than 2 min.

Without AMR, we expect that refinement by a factor of r will roughly cost a factor of r^2 more work. This is because the work per time-step is proportional to the number of grid cells, and the time-step is

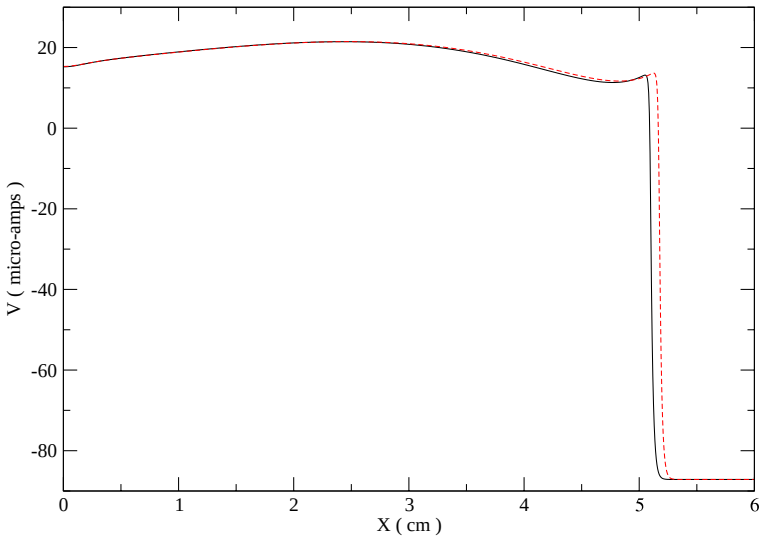

Fig. 2. Membrane potential versus position. Solid curve, unsplit method; dashed curve, operator split method.

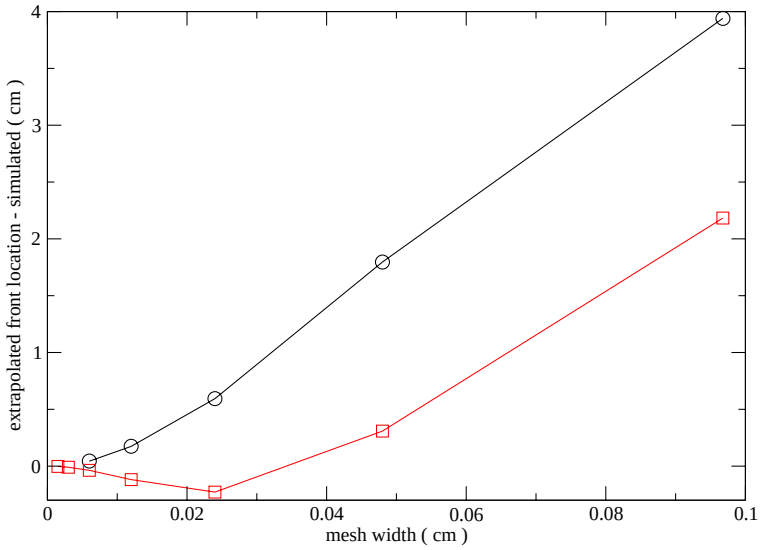

Fig. 3. Error in simulated front position versus mesh width. Circles, unsplit method; squares, split method.

proportional to the cell width. (If we used explicit time integration for the diffusion as in [14], the time-step would have been proportional to the cell width squared for numerical stability, and the work should roughly increase by a factor of r^3 as a result of refinement.) Thus, refinement by a factor of 4 should increase the cost of uniform grid computations by a factor of 16. In practice, we observed that the computational time increased by somewhat different factors, as shown in Table 1. The observed ratios can be larger than predicted because we change the tolerance for the reaction computations as we refine the mesh, as described in Section 4.3. In addition, as the mesh is refined it becomes evident that the reaction front has

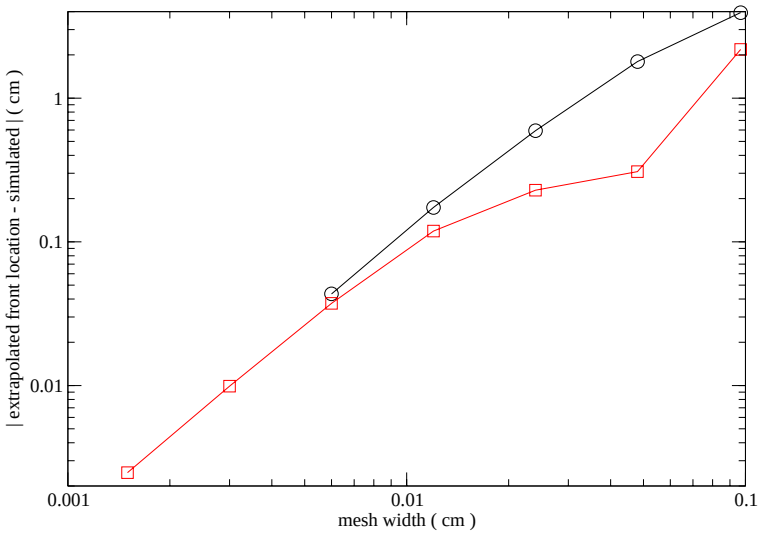

Fig. 4. Absolute value of error in simulated front position versus mesh width. Circles, unsplit method; squares, split method.

a non-zero width in space that requires increasingly more grid cells to resolve. At low resolution, the computational time can be less than predicted because the reaction wave is not propagating properly (if at all), and the solution is not changing much from one time-step to the next.

The computational results for 1D uniform grid simulations show that most of the computational time is spent in integrating the reactions. For problems with at least 80 grid cells, 90–95% of computational time is spent in this task. It should be noted, however, that the solution of the linear system for the diffusion equation is especially simple in 1D, because we can use a tridiagonal system solver. We should not expect the reaction computations to dominate to such an extent in multiple dimensions.

The computational results for 1D AMR computations show a different scaling with mesh refinement. Suppose that the propagating front were refined with a fixed number of grid cells normal to the front on each level of refinement. Then, adding another level of refinement should roughly increase the computational work by a factor of the refinement ratio r, because the time-step on the fine grid would be decreased by that factor. In practice, we observed that the computational time increased by a factor greater than r, as shown in Table 2. These factors are greater than r for at least two reasons. First, the tolerances for both the reactions and the iterative solution of the linear systems for the diffusion are tightened as we refine the mesh, as described in Section 4.3. Second, the reaction is more carefully described as having a fixed width in space, so additional levels of refinement involve increasingly more grid cells within the reaction front. Nevertheless, the computational time increases much less rapidly with AMR than with uniform grid refinement. This indicates that for sufficiently refined computations, the AMR computations should take less computational time than the uniform grid computations. In Fig. 6 we display some timings for AMR runs with different refinement ratios and number of levels. Each calculation was run with 20 grid cells on level 0 and refined by a fixed refinement ratio within a curve.

The computational results for 1D AMR simulations show that about 1/3 of the computational time is spent in integrating the reactions and about 2/3 of the total time in solving the linear systems for diffusion. About 12% of simulation time is spent in error estimation for regridding; this includes time for solving linear systems and for integrating reactions. In 1D, the linear system solves with AMR take relatively more work than uniform grid computations for two reasons. First of all, with AMR there is less work to do in integrating the reactions because there are fewer grid cells. Second, there is more work to do in the linear

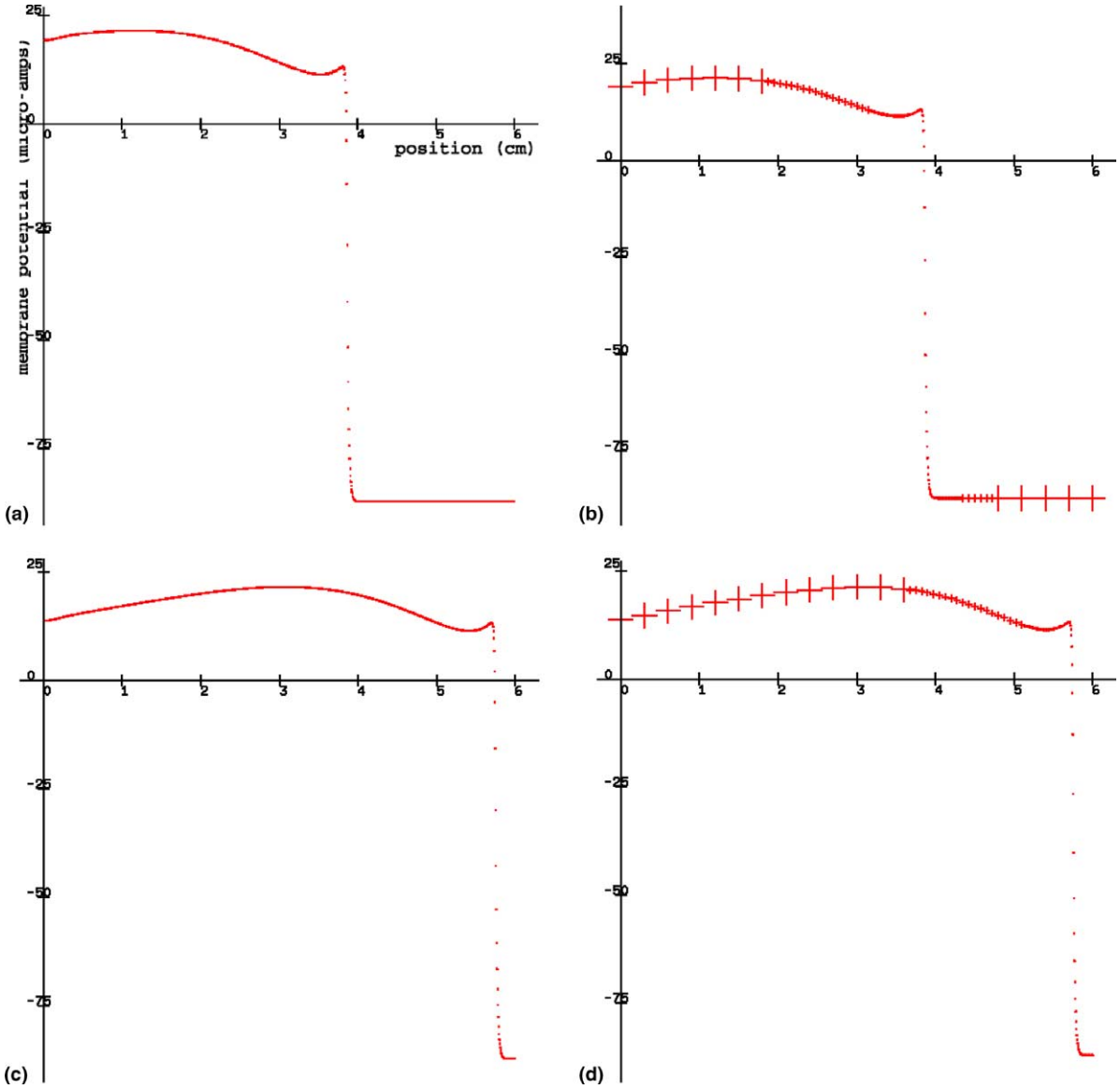

Fig. 5. Membrane potential versus position. (a) Uniform grid of 1280 cells at 60 ms; (b) AMR with four levels at 60 ms; (c) uniform grid of 1280 cells at 90 ms; (d) AMR with four levels at 90 ms.

system solve, because the preconditioned conjugate gradient iteration using multiplicative domain decomposition for AMR cannot be performed as rapidly as a single tridiagonal system solve. This is in spite of that fact that the AMR algorithms (conjugate gradients preconditioned by multiplicative domain decomposition) involve work proportional to the number of unknowns in each iteration, and the number of iterations grows very slowly as the mesh is refined.

In Fig. 7 we show the speedup due to AMR, compared to uniform grid computations. With 80 grid cells on a uniform grid compared to 20 grid cells refined adaptively once by a factor of 4, we find that the AMR

Table 1
Computational times for 1D uniform grid calculations on a Dec Alpha

Cells	Time (s)	Ratio
20	0.0401	
		7.78
80	0.3116	
		11.17
320	3.48	
		14.89
1280	51.83	
		17.5
5120	907	

Table 2
Computational times for 1D AMR grid calculations on a Dec Alpha; refinement ratio = 4, 20 cells on coarsest grid

Number levels	Time (s)	Ratio
2	0.4937	
		6.15
3	3.036	
		6.63
4	20.13	
		6.16
5	124	

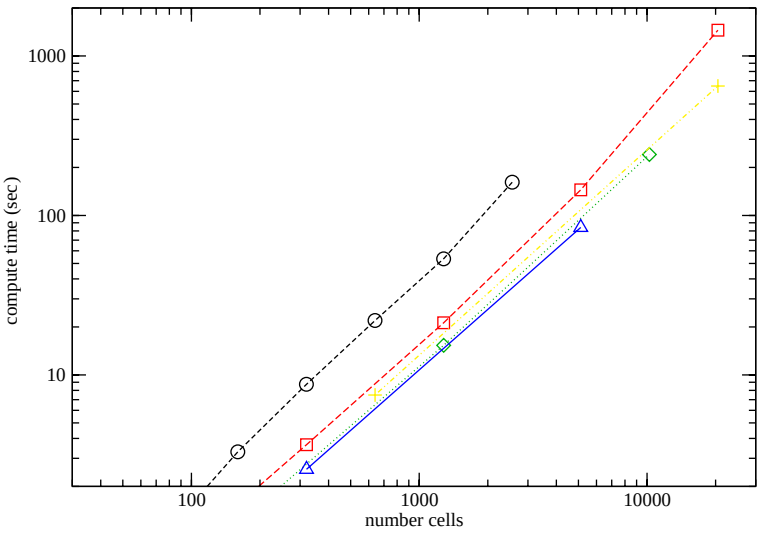

Fig. 6. Computational time versus equivalent number of grid cells. Circles, refinement ratio 2; squares, ratio 4; diamonds, ratio 8; triangles, ratio 16; plus, ratio 32. Horizontal axis is $20 \times r^{\ell}$, where r is the refinement ratio and ℓ is the finest level number. Computations performed on a 1.4 GHz Athlon processor.

code runs slower than the uniform grid computation. However, if we compare a uniform grid of 5120 cells to AMR with five levels using a refinement ratio of 4, we find that the AMR computation runs a factor of 7.29 faster than the uniform grid computation.

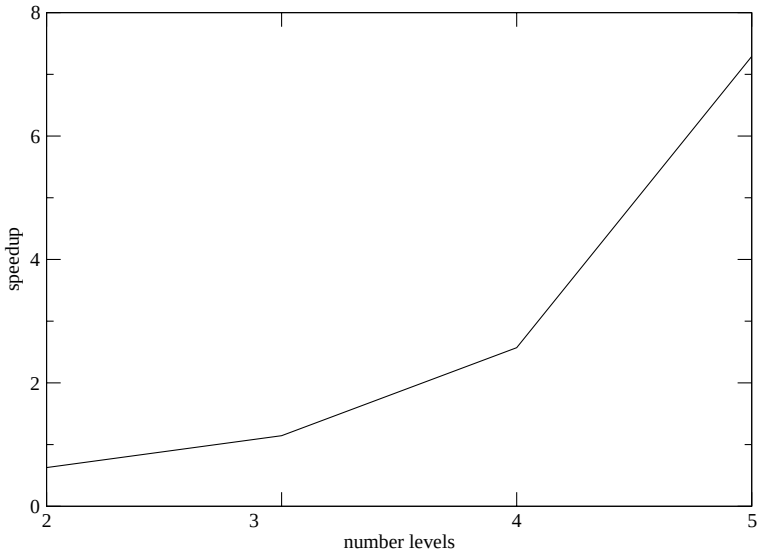

Fig. 7. Speedup due to AMR versus levels of refinement in 1D. Level 0 involved 20 cells; refinement by a factor of 4.

8.3. Numerical results in 2D

In 2D, the wave propagation was simulated for a total of 460 ms. This is slightly more than the amount of time it takes for the spiral wave due to the second stimulus to circle around the tissue. After this time, the spiral wave dissipates and AMR removes all refinement. Between roughly 120 and 315 s in simulation time, the tissue is relaxing from the initial stimulus, and AMR is able to remove all refinement. Uniform grid computations waste a great deal of simulation time in this time interval.

In Fig. 8 we show some computational results for AMR computations in 2D. As a result of the first stimulus along the left-hand side of the domain, the initial jump in membrane potential propagates to the right, then the tissues relaxes. After the second stimulus in the lower left-hand quadrant of the tissue, a spiral wave moves clockwise around the domain. The contour plots use a hot–cold color map (blue for low values, red for high), with the grid patches drawn over the contours.

In 2D we are not able to perform uniform grid calculations with more than 640 cells in each coordinate direction, because of the size of available memory (1 GB). A uniform grid computation with 320 cells in each direction took 4863 s, or slightly more than 81 min, on our Dec Alpha machine. The corresponding AMR simulation using five levels and a refinement ratio of 2 took 2812 s, or slightly less than 47 min. AMR using three levels and a refinement ratio of 4 took 1113 s, or about 18.5 min. In general, it is more efficient to use a refinement ratio of 4 than a ratio of 2.

Without AMR, we expect that refinement by a factor of r will roughly cost a factor of r^3 more work. This is because the work per time-step is proportional to the number of grid cells (which multiplies the work by $O(r^2)$), and the time-step size is proportional to the cell width (which multiplies the work by $O(r)$). If we had used explicit time integration for the diffusion, the time-step would have been proportional to the square of the cell width for numerical stability, and the work would roughly increase by a factor of r^4 as a result of refinement. Thus, refinement by a factor of 4 should increase the cost of uniform grid computations by a factor of 64, and refinement by a factor of 2 should increase the cost by a factor of 8. In practice, we observed that the computational time increased as shown in Table 3. We believe that the observed ratios were larger than predicted in some cases for the same reasons as in 1D.

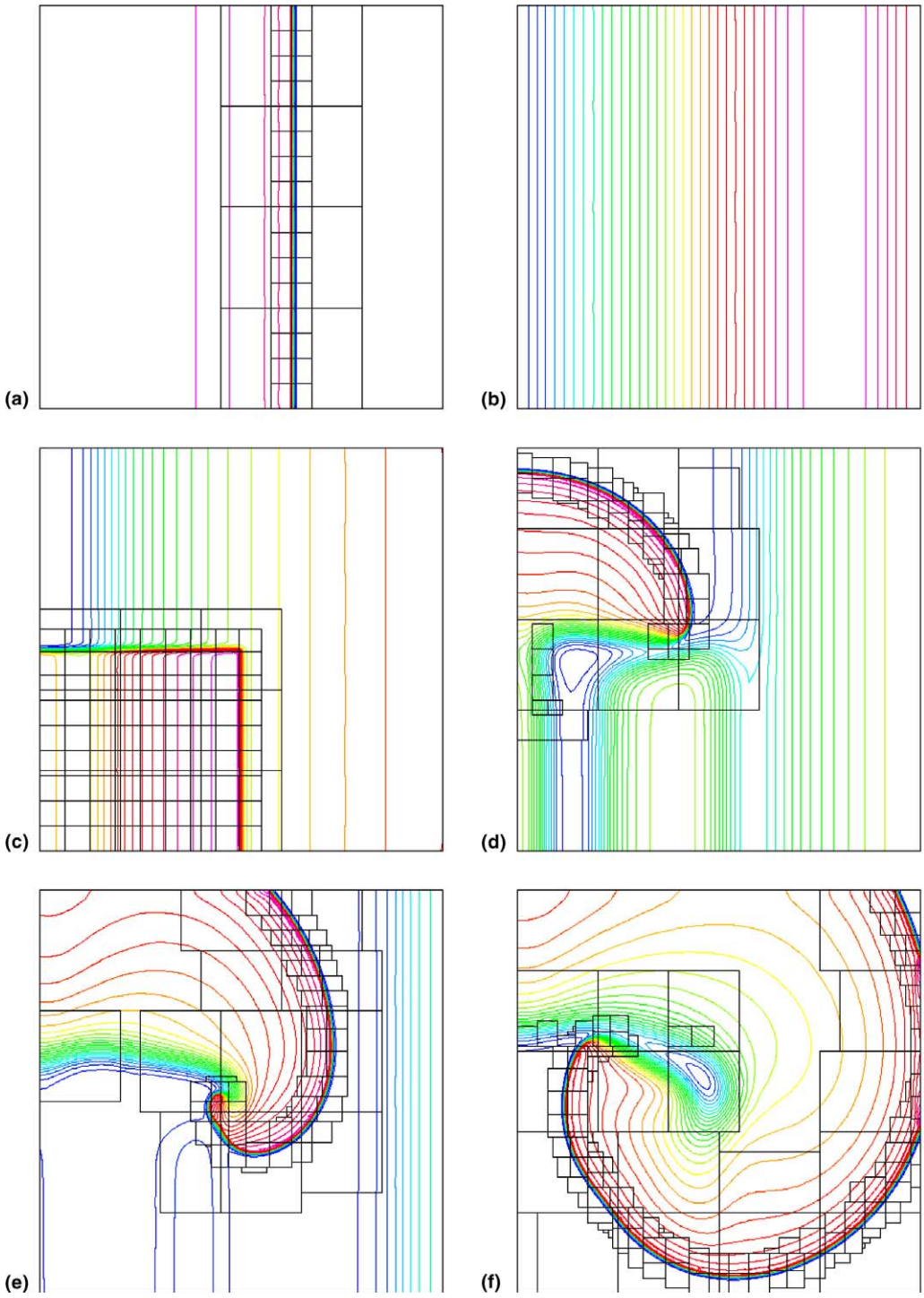

Fig. 8. Contours of membrane potential, AMR, three levels, 40 cells on coarsest level, 30 equally spaced contours: (a) 60 ms; (b) 120 ms; (c) 318 ms; (d) 360 ms; (e) 390 ms; (f) 420 ms.

Table 3
Computational times for 2D uniform grid calculations on a Dec Alpha

Cells	Time (s)	Ratio
20×20	0.8258	
		5.34
40×40	4.41	
		7.61
80×80	33.56	
		8.28
160×160	277.9	
		17.5
320×320	4863	

The computational results for 2D uniform grid simulations again show that most of the computational time is spent in integrating the reactions. For all of the 2D simulations (using 20–640 cells in each direction), 90% of computational time is spent in this task. This is in spite of the fact that we are using a simple conjugate gradient iteration (without preconditioning) to solve the uniform grid linear system. However, as the mesh is refined the cost of the reaction integrations decreases slightly, as the conjugate gradient iteration begins to take more iterations to converge.

The computational results for 2D AMR computations show a different scaling with mesh refinement. Suppose that the propagating front were refined with a fixed number of grid cells normal to the front on each level of refinement. Then, adding another level of refinement should roughly increase the computational work by a factor of the refinement ratio r^2, because the time-step on the fine grid would be decreased by factor of r and we would use a factor of r more grid cells tangential to the front. (If we had used explicit time integration for the diffusion, as in [14], the work would be expected to scale proportional to r^3.) Thus with a refinement ratio of 2, we expect that the computational cost should increase by a factor of 4 with each new level of AMR. In practice, we found that the computational time increased as shown in Table 4. Note that the computational time increases less rapidly with AMR than with uniform grid refinement. This indicates that for sufficiently refined computations, the AMR computations should take less computational time than the uniform grid computations.

The computational results for 2D AMR simulations show that about 3/4 of the computational time is spent in integrating the reactions, and about 1/4 of the total time in solving the linear systems for diffusion. In practice, the preconditioned conjugate gradient iteration worked very well, typically using 1 or 2 iterations to converge to the tolerance described in Section 4.3. About 10% of simulation time is spent in error estimation for regridding; this includes time for solving linear systems and for integrating reactions.

Table 4
Computational times for 2D AMR grid calculations on a Dec Alpha; refinement ratio = 2, 20×20 cells on coarsest level

Number levels	Time (s)	Ratio
2	11.91	
		4.62
3	55.04	
		9.14
4	503	
		5.59
5	2812	

In Fig. 9 we show the speedup due to AMR, compared to uniform grid computations. With 40 grid cells in each direction on a uniform grid compared to 20 grid cells refined adaptively once by a factor of 2, we find that the AMR code runs slower than the uniform grid computation. However, if we compare a uniform grid of 320 cells in each direction to an AMR grid with five levels using a refinement ratio of 2, we find that the AMR computation runs a factor of 1.72 faster than the uniform grid computation. The speedup is significantly greater when AMR uses a refinement ratio of 4. For example, a uniform grid of 320 cells in each direction takes a factor of 4.35 more computational time than an AMR simulation using three levels refined from a coarse grid with 20 cells in each direction. We were not able to compare AMR using four or more levels to uniform grid computations, because the latter would not fit in the available memory.

8.4. Extensions

It is reasonable to ask how the techniques in this paper could be applied to more realistic simulations of electrical wave propagation in the heart. The extension of the AMR techniques in this paper to structured 3D grids is straightforward. In fact, the AMR code described in this paper has already been applied to 3D simulation of viscous instabilities for single-phase flow in porous media [52]. That application involves an elliptic equation with rapidly varying coefficients, much as the elliptic equation in the bidomain model for electrical wave propagation in the heart. The multigrid iterative solver for that elliptic equation takes roughly a fixed number of iterations to solve the linear system with different numbers of levels of refinement, so that the work per time-step is roughly proportional to the number of grid cells. At any rate, we do not foresee any difficulties in extending the methods in this paper to 3D or the bidomain equations.

For problems in which the heart is in a state of chaotic electrical activity, the AMR techniques described in this paper will not be computational advantageous. In such cases, the AMR code will naturally refine almost everywhere, at greater cost than a uniform grid computation.

The most significant roadblock preventing us from simulating electrical wave propagation in the full (3D) heart are the kinds of datasets available for describing the heart itself. Some datasets are basically staircase grids, providing at best a first-order geometric description of the heart. Other datasets are tet-

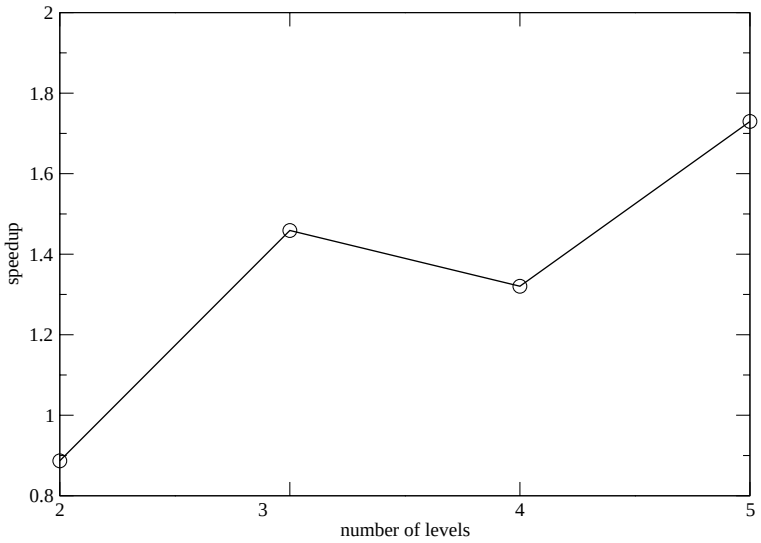

Fig. 9. Speedup due to AMR versus levels of refinement in 2D. Level 0 involved 20 cells; refinement by a factor of 2.

rahedral and are best suited for unstructured grid simulations. We are searching for ways to represent the heart with structured smooth grids, or access to existing datasets of this type, before we extend this code to 3D.

9. Conclusions

Iterative solution of linear systems within AMR leads to significant additional complexity for the AMR code. Further, many of the details of this code implementation cannot be carried over to other AMR applications.

For example, we have previously developed iterative methods for solving linear systems arising from incompressible flow in porous media within AMR. In order to conserve fluid volume and treat random permeability via harmonic averaging, we used a non-conforming hybrid mixed finite element method combined with mortar elements. In that porous media application, the volume conservation naturally gave rise to fine grid unknowns as master to the coarse unknowns at interfaces between coarse and fine grids. In this application of AMR to a diffusion equation, we used a conforming finite element method, in which it was natural for the coarse grid unknowns on the interface to be master to the fine. This reversal of communication rippled through a fair amount of the multiplicative domain decomposition preconditioner.

In spite of the difficulties in code development, we are encouraged by the relative performance advantage of AMR, when compared to uniform grid computations. Because of the difficulty in programming AMR for these applications, we have made the code available to the public at http://www.math.duke.edu/~johnt/duke_amr.html.

This software consists of eight libraries and four application examples. This code has been programmed for distributed computing; we will report on the success of that development in a later paper.

Acknowledgements

We thank Kenneth Eaton and Craig Henriquez of Duke University for many helpful comments during the development of this application of AMR to the Luo–Rudy I model.

Appendix A

We will present the details of our implementation of the LR I reactions below. These functions have been modified slightly from the original LR I model for improved numerical performance.

The rate of change of intracellular calcium $[\mathrm{Ca}]_\mathrm{i}$ is

$$\rho_{[\mathrm{Ca}]_\mathrm{i}}(\mathbf{g}, [\mathrm{Ca}]_\mathrm{i}, V) \equiv -10^{-4} I_{\mathrm{si}}(\mathbf{g}, [\mathrm{Ca}]_\mathrm{i}, V) + 0.07(10^{-4} - [\mathrm{Ca}]_\mathrm{i}), \tag{A.1}$$

where $\mathbf{g}$ is the vector of gating variables, V is the membrane potential and the slow inward current I_{si} (in $\mu\mathrm{A}/\mathrm{cm}^2$) is described in [34]. The rate of change of the membrane potential is as described in Eq. (1) of [34]:

$$\rho_V(\mathbf{g}, [\mathrm{Ca}]_\mathrm{i}, V, t) \equiv -\frac{1}{C}\left[I_{\mathrm{Na}}(\mathbf{g}, V) + I_{\mathrm{si}}(\mathbf{g}, [\mathrm{Ca}]_\mathrm{i}, V) + I_{\mathrm{K}}(\mathbf{g}, V) + I_{\mathrm{K_1}}(\mathbf{g}, V) + I_{\mathrm{K_p}}(V) + I_{\mathrm{b}}(V) + I_{\mathrm{st}}(t)\right],$$

where I_{Na} is the fast sodium current, I_{si} is the slow inward current, I_{K} is the time-dependent potassium current, $I_{\mathrm{K_1}}$ is the time-independent potassium current, $I_{\mathrm{K_p}}$ is the plateau potassium current, I_{b} is the time-independent background current and I_{st} is the stimulus current. These currents are described in Table 1 of

[34] and have units of $\mu A/cm^2$. The membrane capacitance per area is $C = 1\ \mu F/cm^2$, and the membrane diffusion coefficient $D(x)$ is the membrane conductance divided by the membrane capacitance, with units of square cm per millisecond. We have taken $D(x)$ to be identically 0.00125 cm/ms in this paper.

The vector of gating variables has the form

$$\mathbf{g}^{\mathrm{T}} = [\, K_1 \quad X \quad h \quad j \quad m \quad f \quad d \,]. \tag{A.2}$$

These gating variables are dimensionless. In addition, the stimulus current $I_{\mathrm{st}}(\mathbf{x}, t)$ is some specified function of space and time. We described our stimulus current in Section 8.

The inactivation gate X_i of the time-dependent sodium current I_{K} and the activation gate K_{p} of the plateau potassium current $I_{\mathrm{K_p}}$ are assumed to reach steady state very rapidly, so they are functions of V only:

$$X_i(V) = \begin{cases} 2.837 e_1(-v_{X_i}(V)) \exp(v_{X_i}(V) - \delta_{X_i}(V)), & v_{X_i}(V) \geqslant 0, \\ 2.837 e_1(v_{X_i}(V)) \exp(-\delta_{X_i}(V)), & v_{X_i}(V) < 0 \quad \text{and} \quad V > -100, \\ 1., & V \leqslant -100, \end{cases} \tag{A.3a}$$

where $v_{X_i}(V) \equiv 0.04(V + 77)$ and $\delta_{X_i}(V) \equiv 0.04(V + 35)$

$$K_{\mathrm{p}}(V) = \begin{cases} \exp(-\delta_{K_{\mathrm{p}}}(V))/[1 + \exp(-\delta_{K_{\mathrm{p}}}(V))], & \delta_{K_{\mathrm{p}}}(V) \geqslant 0, \\ 1/[1 + \exp(\delta_{K_{\mathrm{p}}}(V))], & \delta_{K_{\mathrm{p}}}(V) < 0, \end{cases} \tag{A.3b}$$

where $\delta_{K_{\mathrm{p}}}(V) \equiv (V - 7.488)/5.98$.

In order to avoid unnecessary cancellation errors, we have used the function

$$e_1(x) \equiv \frac{\mathrm{e}^x - 1}{x}.$$

For $|x| \leqslant 1/4$, we use the Padé approximation

$$e_1(x) \approx \frac{15120 + x(840 + x(420 + x(20 + x)))}{15120 - x(6720 - x(1260 - x(120 - 5x)))}, \tag{A.4}$$

with errors close to roundoff in double precision on our machine.

Recall from Eq. (1a) that the other gating variables satisfy rate equations of the form $\mathrm{d}\mathbf{g}/\mathrm{d}t = \mathbf{a} - (\mathbf{a} + \mathbf{b})\mathbf{g}$. We use the notation

$$\mathbf{a}^{\mathrm{T}} = [\, \alpha_{K_1} \quad \alpha_X \quad \alpha_h \quad \alpha_j \quad \alpha_m \quad \alpha_f \quad \alpha_d \,], \quad \mathbf{b}^{\mathrm{T}} = [\, \beta_{K_1} \quad \beta_X \quad \beta_h \quad \beta_j \quad \beta_m \quad \beta_f \quad \beta_d \,]$$

for the vectors of rates in these rate equations. The rates for the inactivation gate K_1 of the time-independent potassium current $I_{\mathrm{K_1}}$ are:

$$\alpha_{K_1}(V) = \begin{cases} 1.02 \exp(-\delta_{K_1}(V))/[1 + \exp(-\delta_{K_1}(V))], & \delta_{K_1}(V) \geqslant 0, \\ 1.02/[1 + \exp(\delta_{K_1}(V))], & \delta_{K_1}(V) < 0, \end{cases} \tag{A.5a}$$

where $\delta_{K_1}(V) \equiv 0.2385(V + 87.26 - 59.215)$;

$$\beta_{K_1}(V) = \begin{cases} \frac{0.49124 \exp(v_{K_1,1}(V) - \delta_{K_1}(V)) + \exp(v_{K_1,2}(V) - \delta_{K_1}(V))}{1 + \exp(-\delta_{K_1}(V))}, & \delta_{K_1}(V) \geqslant 0, \\ \frac{0.49124 \exp(v_{K_1,1}(V)) + \exp(v_{K_1,2}(V))}{1 + \exp(\delta_{K_1}(V))}, & \delta_{K_1}(V) < 0, \end{cases} \tag{A.5b}$$

where $v_{K_1,1}(V) \equiv 0.08032(V + 87.26 + 5.476)$, $v_{K_1,2}(V) \equiv 0.06175(V + 87.26 - 594.31)$, $\delta_{K_1}(V) \equiv -0.5143(V + 87.26 + 4.753)$.

The rates for the activation gate X of the time-dependent potassium current I_K are:

$$\alpha_X(V) = \begin{cases} 0.0005\exp(v_X(V) - \delta_X(V))/[1+\exp(-\delta_X(V))], & \delta_X(V) \geqslant 0, \\ 0.0005\exp(v_X(V))/[1+\exp(\delta_X(V))], & \delta_X(V) < 0, \end{cases} \tag{A.6a}$$

where $v_X(V) \equiv 0.083(V+50)$, $\delta_X(V) \equiv 0.057(V+50)$;

$$\beta_X(V) = \begin{cases} 0.0013\exp(v_X(V) - \delta_X(V))/[1+\exp(-\delta_X(V))], & \delta_X(V) \geqslant 0, \\ 0.0013\exp(v_X(V))/[1+\exp(\delta_X(V))], & \delta_X(V) < 0, \end{cases} \tag{A.6b}$$

where $v_X(V) \equiv -0.06(V+20)$, $\delta_X(V) \equiv -0.04(V+20)$.

The rates for the inactivation gate h of the fast sodium current I_{Na} are:

$$\alpha_h(V) = 0.135\exp(-(V+80)/6.8), \tag{A.7a}$$

$$\beta_h(V) = \begin{cases} \phi_h[3.56\exp(0.079V) + 3.1\times 10^5\exp(0.35V)], & V < -40, \\ (1/0.13)/[1+\exp(-(V+10.66)/11.1)], & V \geqslant -40, \end{cases} \tag{A.7b}$$

where ϕ_h chosen so that β_h is continuous.

The rates for the slow inactivation gate j of the fast sodium current I_{Na} are:

$$\alpha_j(V) = \begin{cases} -\frac{[\omega_{j1}\exp(v_{j1}(V)-\delta_j(V))+\omega_{j2}\exp(v_{j2}(V)-\delta_j(V))](V+37.78)}{1+\exp(-\delta_j(V))}, & \delta_j(V) \geqslant 0, \\ -\frac{[\omega_{j1}\exp(v_{j1}(V))+\omega_{j2}\exp(v_{j2}(V))](V+37.78)}{1+\exp(\delta_j(V))}, & \delta_j(V) < 0, \end{cases} \tag{A.8a}$$

where $\omega_{j1} = 1.2714\times 10^5$, $\omega_{j2} = 3.474\times 10^{-5}$, $v_{j1}(V) = 0.2444V$, $v_{j2}(V) = -0.04391V$, $\delta_j(V) = 0.311(V+79.23)$;

$$\beta_j(V) = \begin{cases} 0.3, & V > 72, \\ 0.3\exp(-2.535\times 10^{-7}V)/[1+\exp(-0.1(V+32))], & -40 \leqslant V \leqslant 72, \\ \phi_j\exp(-0.0152V)/[1+\exp(-0.1378(V+40.14))], & V < -40, \end{cases} \tag{A.8b}$$

where ϕ_j chosen so that β_j is continuous.

The rates for the activation gate m of the fast sodium current I_{Na} are:

$$\alpha_m(V) = \begin{cases} 3.2/e_1(-\delta_m(V)), & \delta_m(V) \geqslant 0, \\ 3.2\exp(\delta_m(V))/e_1(\delta_m(V)), & \delta_m(V) < 0, \end{cases} \tag{A.9a}$$

where $\delta_m(V) = 0.1(V+47.13)$;

$$\beta_m(V) = 0.08\exp(-V/11.0). \tag{A.9b}$$

The rates for the inactivation gate f of the slow inward current I_{si} are:

$$\alpha_f(V) = \begin{cases} 0.12\exp(v_f(V) - \delta_f(V))[1+\exp(-\delta_f(V))], & \delta_f(V) \geqslant 0, \\ 0.12\exp(v_f(V))[1+\exp(\delta_f(V))], & \delta_f(V) < 0, \end{cases} \tag{A.10a}$$

where $v_f(V) = -0.008(V+28.)$, $\delta_f(V) = 0.15(V+28.)$;

$$\beta_f(V) = \begin{cases} 0.0065\exp(v_f(V) - \delta_f(V))[1+\exp(-\delta_f(V))], & \delta_f(V) \geqslant 0, \\ 0.0065\exp(v_f(V))[1+\exp(\delta_f(V))], & \delta_f(V) < 0, \end{cases} \tag{A.10b}$$

where $v_f(V) = -0.02(V + 30.)$, $\delta_f(V) = -0.2(V + 30.)$.The rates for the activation gate d of the slow inward current I_{si} are:

$$\alpha_d(V) = \begin{cases} 0.095\exp(v_d(V) - \delta_d(V))/[1 + \exp(-\delta_d(V))], & \delta_d(V) \geqslant 0, \\ 0.095\exp(v_d(V))/[1 + \exp(\delta_d(V))], & \delta_d(V) < 0, \end{cases} \tag{A.11a}$$

where $v_d(V) = -0.01(V - 5)$, $\delta_d(V) = -0.072(V - 5)$;

$$\beta_d(V) = \begin{cases} 0.07\exp(v_d(V) - \delta_d(V))/[1 + \exp(-\delta_d(V))], & \delta_d(V) \geqslant 0, \\ 0.07\exp(v_d(V))/[1 + \exp(\delta_d(V))], & \delta_d(V) < 0, \end{cases} \tag{A.11b}$$

where $v_d(V) = -0.017(V + 44.)$, $\delta_d(V) = 0.05(V + 44.)$.

In these expressions, only α_h and β_m are the identical to the equations in [34].

References

[1] R.E. Bank, A posteriori error estimates, adaptive local mesh refinement and multigrid iteration, in: W. Hackbusch, U. Trottenberg (Eds.), Proceedings of the 2nd European Conference on Multigrid Methods, Cologne, October 1–4, 1985, 1986.

[2] John P. Barach, A simulation of cardiac action currents having curl, IEEE Trans. Biomed. Engrg. 40 (1993) 49–58.

[3] J.B. Bell, P. Colella, J. Trangenstein, M. Welcome, Godunov methods and adaptive algorithms for unsteady flow. in: Proceedings of the 11th International Conference on Numerical Methods in Fluid Dynamics, June 27–July 1, 1988.

[4] M. Berger, J. Melton, An accuracy test of a cartesian grid method for steady flow in complex geometries. in: Proceedings of the 5th International Conference on Hyperbolic Problems, Stonybook, NY, June, 1994.

[5] M.J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1989) 64–84.

[6] M.J. Berger, R.J. LeVeque, Stable boundary conditions for Cartesian grid calculations, Comput. Syst. Engrg. 1 (1990) 305–331.

[7] M.J. Berger, I. Rigoutsos, An algorithm for point clustering and grid generation, IEEE Trans. Systems Man Cybern. 21 (1991) 1278–1286.

[8] M.J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. 53 (1984) 484–512.

[9] C. Bernardi, Y. Maday, A.T. Patera, A new nonconforming approach to domain decomposition: the mortar element method, in: H. Brezis, J.L. Lions (Eds.), Nonlinear Partial Differential Equations and Their Applications, Pitman Research Notes on Mathematics, vol. 290, Longman Scientific & Technical, Harlow, UK, 1989, pp. 13–51.

[10] J.H. Bramble, X. Zhang, in: P.G. Ciarlet, J.L. Lions (Eds.), Handbook of Numerical Analysis, vol. VII, The Analysis of Multigrid Methods, North-Holland, Amsterdam, 2000, pp. 173–415.

[11] J.H. Bramble, Multigrid Methods, in: Pitman Research Notes in Mathematics, vol. 294, Longman, London, 1993.

[12] J.H. Bramble, R.E. Ewing, R.R. Parashkevov, J.E. Pasciak, Domain decomposition methods for problems with partial refinement, SIAM J. Sci. Statist. Comput. 13 (1992) 397–410.

[13] J.H. Bramble, R.E. Ewing, J.E. Pasciak, A.H. Schatz, A preconditioning technique for the efficient solution of problems with local grid refinement, Comput. Methods Appl. Mech. Engrg. 67 (1988) 149–159.

[14] E.M. Cherry, H.S. Greenside, C.S. Henriquez, A space-time adaptive method for simulating complex cardiac dynamics, Phys. Rev. Lett. 84 (2000) 1343–1346.

[15] M. Courtemanche, A. Winfreem, Re-entrant rotating waves in a Beeler–Reuter based model of two-dimensional cardiac electrical activity, Int. J. Bifurc. Chaos 1 (1991) 44431–44446.

[16] H.K. Dahle, M.S. Espedal, R.E. Ewing, O. Savareid, Characteristic adaptive subdomain methods for reservoir flow problems, Numer. Methods Partial Differential Equations 6 (1990) 279–309.

[17] G. Dahlquist, Å. Björck, Numerical Methods, Prentice-Hall, Englewood Cliff, 1974 (Transl. by N. Anderson).

[18] K. Dekker, J.G. Verwer, Stability of Runge–Kutta Methods for Stiff Nonlinear Differential Equations, North-Holland, Amsterdam, 1984.

[19] Emilia Entcheva, Natalia A. Trayanova, Fritz J. Claydon, Patterns and mechanisms for shock-induced polarization in the heart: a bidomain analysis, IEEE Trans. Biomed. Engrg. 46 (1999) 260–270.

[20] R.E. Ewing, R.D. Lazarov, T.F. Russell, P.S. Vassilevski, Analysis of the mixed finite element method for rectangular Raviart–Thomas elements with local refinement, in: T.F. Chan, R. Glowinski, J. Periaux, O. Widlund (Eds.), Domain Decomposition Methods for Partial Differential Equations, SIAM, Philadelphia, PA, 1990, pp. 98–114.

[21] R.E. Ewing, in: J.E. Flaherty (Ed.), Adaptive Methods for Partial Differential Equations, Adaptive Grid Refinements for Transient Flow Problems, SIAM, 1989, pp. 194–205.

[22] R.E. Ewing, R.D. Lazarov, T.F. Russel, P.S. Vassilevski, Local refinement via domain decomposition techniques for mixed finite element methods with rectangular Raviart–Thomas elements, in: Tony Chan (Ed.), Third International Symposium on Domain Decomposition for Partial Differential Equations, SIAM, 1989.
[23] R.E. Ewing, J. Wang, Analysis of mixed finite element methods in locally refined grids, Numer. Math. 63 (1992) 183–194.
[24] Richard Ewing, Raytcho Lazarov, Panayot Vassilevski, Local refinement techniques for elliptic problems on cell-centered grids I. Error analysis, Math. Comput. 56 (194) (1991) 437–461.
[25] R. Ewing, R. Lazarov, P. Vassilevski, Local refinement techniques for elliptic problems on cell-centered grids. III. algebraic multilevel BEPS preconditioners, Numer. Math. 59 (1991) 431–452.
[26] Richard Ewing, Raytcho Lazarov, Panayot Vassilevski, Local refinement techniques for elliptic problems on cell-centered grids. II. Two-grids iterative methods, Numer. Lin. Alg. Appl. 1 (1) (1994) 337–368.
[27] M.G. Fishler, Syncytial heterogeneity as a mechanism underlying cardiac far-field stimulation during defibrillation-level shock, J. Cardiovasc. Electrophys. 9 (1998) 384–394.
[28] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II Stiff and Differential Algebraic Equations, Springer-Verlag, 1991.
[29] A.C. Hindmarsh, in: R.S. Stepleman et al. (Ed.), Scientific Computing, ODEPACK, A Systematized Collection of ODE Solvers, North-Holland, Amsterdam, 1983.
[30] R.D. Hornung, J.A. Trangenstein, Adaptive mesh refinement and multilevel iteration for flow in porous media, J. Comput. Phys. 136 (1997) 522–545.
[31] J.P. Keener, An eikonal-curvature equation for action potential propagation in myocardium, J. Math. Biol. 29 (1991) 629–651.
[32] J.P. Keener, K. Bogar, A numerical method for the solution of the bidomain equations in cardiac tissue, Chaos 8 (1998) 234–241.
[33] W. Krassowska, M.S. Kumar, The role of spatial interactions in creating the dispersion of transmembrane potential by premature electric shocks, Ann. Biomed. Engrg. 25 (1997) 949–963.
[34] C.-H. Luo, Y. Rudy, A model of the ventricular cardiac action potential: depolarization, repolarization, and their interaction, Circ. Res. 68 (1991) 1501–1526.
[35] A.L. Muzikant, C.S. Henriquez, Paced activation mapping reveals organization of myocardial fibers, J. Cardiovasc. Electrophys. 8 (1997) 281–294.
[36] J.C. Newton, S.B. Knisley, X. Zhou, A.E. Pollard, R.E. Ideker, Review of mechanisms by which electrical stimulation alters the transmembrane potential, J. Cardiovasc. Electrophys. 10 (1999) 234–243.
[37] H. Olsson, Runge–Kutta solution of initial value problems, PhD thesis, Department of Computer Science, Lund Institute of Technology, Lund University, 1999.
[38] A.L. Pardhanani, G.F. Carey, Efficient simulation of complex patterns in reaction–diffusion patterns, J. Comput. Appl. Math. 74 (1996) 295–311.
[39] A.E. Pollard, N. Hooke, C.S. Henriquez, Cardiac propagation simulation, Crit. Rev. Biomed. Engrg. 20 (1992) 171–210.
[40] J.B. Pormann, A modular simulation system for the bidomain equations, PhD thesis, Department of Electrical and Computer Engineering, Duke University, 1999.
[41] W. Quan, S.J. Evans, H.M. Hastings, Efficient integration of a realistic two-dimensional cardiac tissue model by domain decomposition, IEEE Trans. Biomed. Engrg. 45 (1998) 372–385.
[42] J.M. Rogers, A.D. McCulloch, A collocation-Galerkin finite element model of cardiac action potential propagation, IEEE Trans. Biomed. Engrg. 41 (1994) 743–757.
[43] B.J. Roth, Mechanisms for electrical stimulation of excitable tissue, Crit. Rev. Biomed. Engrg. 22 (1994) 253–305.
[44] B.J. Roth, A mathematical model of make and break electrical stimulation of cardiac tissue by a unipolar anode or cathode, IEEE Trans. Biomed. Engrg. 42 (1995) 1174–1184.
[45] B.J. Roth, W. Krassowska, The induction of reentry in cardiac tissue. The missing link: how electric fields alter transmembrane potential, Chaos 8 (1998) 204–220.
[46] U. Rüde, Mathematical and Computational Techniques for Multilevel Adaptive Methods, SIAM, 1993.
[47] H.I. Saleheen, K.T. Ng, A new three-dimensional finite-difference bidomain formulation for inhomogeneous anisotropic cardiac tissues, IEEE Trans. Biomed. Engrg. 45 (1998) 15–24.
[48] G. Strang, On the construction and comparison of difference schemes, SIAM J. Numer. Anal. 5 (1968) 506–517.
[49] Michael Taylor, Partial Differential Equations III, in: Applied Mathematical Sciences, vol. 117, Springer-Verlag, 1996.
[50] Vidar Thomée, Galerkin Finite Element Methods for Parabolic Problems, Springer-Verlag, 1997.
[51] J.A. Trangenstein, Adaptive mesh refinement for wave propagation in nonlinear solids, SIAM J. Sci. Statist. Comput. 16 (1995) 819–839.
[52] John A. Trangenstein, Zhuoxin Bi, Multi-scale iterative techniques and adaptive mesh refinement for flow in porous media, Adv. Water Resour. 25 (2002) 1175–1213.
[53] J.A. Trangenstein, K. Skouibine, Operator splitting and adaptive mesh refinement for the Fitzhugh–Nagumo problem, Technical Report, Department of Mathematics, Duke University, 2000.

[54] N. Trayanova, B.J. Roth, Cardiac tissue in an electric field: a study in electrical stimulation, Comput. Cardiol. 31 (Suppl.) (1992) 695–698.
[55] N. Trayanova, K. Skouibine, Modeling defibrillation. Effects of fiber curvature, J. Electrocardiol. 31 (Suppl.) (1998) 23–29.
[56] S. Veronese, H. Othmer, A computational study of wave propagation in a model for anisotropic cardiac ventricular tissue, Lecture Notes Comput. Sci. 919 (1995) 248–253.
[57] E.J. Vigmond, L. Joshua Leon, Computationally efficient model for simulating electrical activity in cardiac tissue with fiber rotation, Ann. Biomed. Engrg. 27 (1999) 160–170.
[58] J. Bradley White, G.P. Walcott, A.E. Pollard, R.E. Ideker, Myocardial discontinuities – a substrate for producing virtual electrodes that directly excite the myocardium by shocks, Circulation 97 (1998) 1738–1745.
[59] A.T. Winfree, Electrical instability in cardiac muscle: phase singularities and rotors, J. Theoret. Biol. 138 (1989) 353–405.