SIAM J. Sci. Comput.
Vol. 30, No. 6, pp. 2922–2947

LONG-TIME SIMULATIONS ON HIGH RESOLUTION MESHES TO MODEL CALCIUM WAVES IN A HEART CELL*

MATTHIAS K. GOBBERT†

Abstract. A model for the flow of calcium on the scale of one heart cell is given by a system of time-dependent reaction-diffusion equations coupled by nonlinear reaction terms. Calcium ions enter into the cell at release units distributed throughout the cell and then diffuse. At each release unit, the probability for calcium to be released increases along with the concentration of calcium, thus creating a feedback loop of waves regenerating themselves repeatedly. The validation of this model requires simulations on the time scale of several repeated waves and on the spatial scale of the entire cell. This requires long-time studies on spatial meshes that need to have a high resolution to resolve the positions of the calcium release units throughout the entire cell. We detail the development of a special-purpose numerical method and parallel implementation for this problem. Parallel performance studies demonstrate the scalability of the implementation on a distributed-memory cluster with low-latency interconnect. Convergence studies verify convergence to analytical expectations and confirm the appropriateness of all numerical parameters. Application studies on the desired time and length scales confirm that the model exhibits the desired feedback mechanism for calcium currents through the release units at suitable high levels, but the long-time studies demonstrate also that the current model with its present parameters leads to excessive calcium concentrations over time. This phenomenon could only be observed using a computational method able to reach laboratory scale final times for a domain on the scale of a complete cell.

Key words. reaction-diffusion equation, nonsmooth data, finite element method, matrix-free implementation, cluster computing

AMS subject classifications. 35K57, 65F10, 65M60, 65Y05, 92C45

DOI. 10.1137/070692261

1. Introduction.

Diffusion waves of calcium ions in a heart cell are part of the normal functioning of the heart, but can also trigger arrhythmias (irregular heart beat) [11, 12, 13]. See these sources as well as the appendix in [8] for more references to background material. The model for the calcium flow is given by a system of coupled, time-dependent reaction-diffusion equations

$$\frac{\partial u^{(i)}}{\partial t} - \nabla \cdot \left(D^{(i)} \nabla u^{(i)} \right) + a^{(i)}\, u^{(i)} = r^{(i)} + \big(- J_{\text{pump}} + J_{\text{leak}} + J_{\text{SR}} \big)\, \delta_{i0} + f^{(i)} \tag{1.1}$$

for the concentrations $u^{(i)}(\mathbf{x}, t)$ of the n_s chemical species $i = 0, 1, \ldots, n_s - 1$ as functions of space $\mathbf{x} \in \Omega \subset \mathbb{R}^3$ and time $0 \leq t \leq t_{\text{fin}}$. In the application problem, (1.1) is coupled with no flow boundary conditions, and the concentrations at the initial time are given.

The time and space derivatives on the left-hand side of (1.1) model the diffusive transport of each chemical species with diffusivities given by the diagonal, positive definite matrices $D^{(i)} \in \mathbb{R}^{3\times3}$, $i = 0, 1, \ldots, n_s - 1$. The reaction terms $r^{(i)} \equiv r^{(i)}(u^{(0)}, \ldots, u^{(n_s-1)})$ on the right-hand side are, in general, nonlinear functions of all species and couple all reaction-diffusion equations in (1.1). In the application problem, crucial effects related to the calcium species, labeled with index

*Received by the editors May 16, 2007; accepted for publication (in revised form) May 28, 2008; published electronically October 13, 2008. This work was partially supported by National Science Foundation under grant DMS–0215373 for the computational hardware used in the numerical studies.
http://www.siam.org/journals/sisc/30-6/69226.html

†Department of Mathematics and Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250 (gobbert@math.umbc.edu).

$i = 0$, are contained in the right-hand side terms associated with the Kronecker delta function δ_{i0} (defined as $\delta_{ij} = 0$ for all $i \neq j$ and $\delta_{ij} = 1$ for $i = j$). In (1.1), the application problem is combined with additional terms $a^{(i)}u^{(i)}$ with constant $a^{(i)} \geq 0$ and given function $f^{(i)} \equiv f^{(i)}(\mathbf{x}, t)$ that incorporates the scalar linear test problem $u_t - \nabla \cdot (D\nabla u) + au = f(\mathbf{x}, t)$ in the formulation. This combined formulation of the problems allows one to switch the code from one problem to the other by turning off terms and is useful in testing to ensure correctness of the code and associated postprocessing routines.

The problem of calcium flow in a cell is a multiscale problem in both space and time. On the scale of a cell, the points where calcium ions are injected into the cell are represented as point sources, i.e., mathematically by highly nonsmooth Dirac delta distributions, each located at a discrete point of size 0, which will be explained in detail in section 2, and specifically in section 2.1 the source term J_{SR} that includes the Dirac delta distributions. In fact, calcium channels have a size of about 10 to 30 nm, and the opening and closing of the channels take place on the scale of microseconds [18, p. 1848]. On the time scale of the waves we intend to simulate, 100 ms, this appears as instantaneous switching in time. The differences in spatial and time scale highlight that different works can have different foci. For instance, [18] considers one channel only (located on a membrane at the boundary of the domain) and models the release of calcium in detail (with a different, mathematically smooth mechanism). To simulate to desired final times of nearly 10 s, while resolving the opening transient, their numerical method uses variable time-stepping. However, due to the focus on one channel, the spatial domain of size $8 \times 8 \times 5$ $\mu\mathrm{m}^3$ includes only the region surrounding the channel.

By contrast, for the simulation of calcium waves through an entire cell, the domain needs to be on the scale of the cell, which is on the order of 10×10 $\mu\mathrm{m}^2$ in cross section and at least between 50 and 100 μm in length. On this scale, the spatial extent of calcium channels appears indeed as a point of size 0. The mathematical model based on a highly nonsmooth Dirac delta distribution for each channel is appropriate on this scale, as it captures the relevant effect on this scale of an injection of an amount of calcium ions per unit of time that can be assessed experimentally. Thus, we choose the domain $\Omega \subset \mathbb{R}^3$ of the differential equation model (1.1) as the interior of one cell as $\Omega = (-6.4, 6.4) \times (-6.4, 6.4) \times (-32.0, 32.0)$ in units of μm. Clearly, a rectangular shape is not the shape of a real cell, but this choice reflects the final goal of simulating calcium waves, for which the exact shape of the cell is not crucial. Moreover, the intermediate goal is actually to validate and tune of the coefficients in the model. Specifically, all functions $r^{(i)}$, J_{pump}, J_{SR}, etc. contain parameters, many of which cannot be measured directly but are inferred indirectly from measurements of other observations. So, before using the model (1.1) as a predictive tool, it needs to be validated against experiments.

To this end, our task is to demonstrate that the model and its computer implementation are capable of replicating the phenomena observed in the laboratory. What has to be achieved for the present purpose is (i) to be able to compute to final times comparable to the laboratory scales and (ii) to do this with a domain that has a realistic size comparable to a real cell. Specifically, a final time on the order of 1,000 ms is our goal, because it will allow enough time for waves to self-organize and for several waves of calcium to traverse the cell, given their typical time scale of 100 ms; see section 2.1. We use a domain with one long dimension of 64 μm and a cross section of 12.8×12.8 $\mu\mathrm{m}^2$, which is a good representation of the scale and the fundamental shape of a cell in this context [11, 13]. A particular example of quantities under still-active validation is the positioning of and distances between the release units of calcium; see, e.g., [2] and [12] that approach the problem from the experimental and computational side, respectively. Specifically in [12], numerical studies analyze the influence of various choices on the initiation of a calcium wave on a short cross section of the cell, on the order of 2 μm in length, with a final simulation time of 100 ms.

By contrast, this work presents a special-purpose simulation platform that is for the first time capable of simulating several repetitions of calcium waves through an entire cell. To accomplish this, we choose a numerical method that is mathematically appropriate for the problem and takes advantage of any special structure of the problem for its efficiency, while still representing the salient features of the problem and keeping the goals of the simulations in mind, and to implement this method efficiently. This paper presents an approach to this problem in this spirit that uses the properties of the application problem to design an efficient special-purpose code for reaction-diffusion equations of the form (1.1) on a domain of rectangular shape such as Ω above. Since our code and its parallelization are new, we also demonstrate how to gain confidence in its efficacy by parallel performance studies and its accuracy by convergence studies, before applying it to the application problem. In these studies, the mesh is refined down to mesh spacings on the order of 0.0625 μm; see sections 4.2 and 4.3. This quantity reaches just about the scale of the calcium channel of up to 30 nm cited above and is as fine as the differential equation model (1.1) can be considered valid. These studies confirm on the one hand that the method converges, even in the face of the highly nonsmooth Dirac delta distribution as source term. On the other hand, it also confirms that the model does not lose or gain mass, that is, the relevant effect of the calcium injection is handled correctly and accurately on the spatial scale of the cell, where each channel appears as a discrete point only.

Other authors suggest the use of local refinement around the location of calcium channels. This is useful if one wishes, e.g., to plot additional detail near the location of the calcium channels. However, this *numerical* fine resolution does not improve the validity of the differential equation model, which by its nature of representing the calcium channels as point sources cannot provide any more meaningful detail close to the source. In fact, by definition of the Dirac delta distribution that necessarily tends to infinity at the channel location, the solution also approaches infinity at the location of each channel, and the value of the solution at the channel location will never converge; this is an inherent feature of the *mathematical* model involving the Dirac delta distributions and cannot be overcome by any numerical method. Notice that the weak form of (1.1) used for the finite element method, as detailed in section 3.1, replaces the Dirac delta distribution by a weak form with a finite constant at that node, and thus the finite element method can be formulated for this problem. However, the classical convergence theory for the finite element method still does not apply, because that assumes the right-hand side of (1.1) to be in the function space $L^2(\Omega)$ spatially at every time. It is for this reason that we check the mass conservation in section 4.3 and that careful convergence tests are necessary to gain confidence in the method [6, 8].

The underlying issue here is a classical aspect of multiscale problems, where one can use the model only on the scale for which it is valid, which is here on the scale of a cell (or substantial part thereof), but with only a Dirac delta distribution; no detailed information is contained in the model itself about the solution structure at or very close to the channel (i.e., within the size of 30 nm of the channel). Rather, the model by design captures only the total amount of calcium ions injected at the channel and the location of the injection. Looking ahead to potential future research, the use of nonuniform meshes would actually be rather to use a fine mesh only near positions of *open* channels, as opposed to using a fine mesh at the positions of all channels, whether open or closed. This amounts to transient mesh refinement and coarsening, for which code is necessarily complicated to program. In the absence of this, our uniform finest mesh can be considered to be locally as fine as other fine meshes close to each channel location, but with unnecessarily many points in between them. Unfortunately, [12] does not seem to state numerical values for the mesh spacing close to the calcium channels, but considering their Figure 1C (zoom), it appears that the smallest mesh spacing (next to channel location) is about 1/16 of the distance Λ between two channels, which is on the order of $\Lambda = 1$ μm. Hence, the smallest mesh spacing used in their mesh may be on the order of $\Lambda/16 = 1/16 = 0.0625$ μm. This value equals, in fact, the mesh spacing of the finest mesh used in the convergence studies in sections 4.2 and 4.3, with the difference that we also have a fine mesh between the channels.

This work can be viewed as enabling the extension of studies of the type of [12] in space and time scale, with the use of comparable computational resources (number of parallel nodes, wall clock time). Thus, the focus is, in the end, on comparing computational cost, and we therefore restrict ourselves to the same model as used there and to a particular set of model parameters. Since the scientific goal of analyzing the influence of parameters on the initiation of calcium waves on a cross section of the cell in [12] is different from our goal of providing a simulator for long-time studies on the whole cell, we do not recompute their studies here. However, to enable any comparisons on the numerical methods, we use the same model and model parameters as in [12]. Our method is able to accomodate generalizations of the model, for instance by adding additional species. One application would be to model the amount of calcium available for insertion at the CRUs explicitly, instead of assuming an unlimited store. This and other variations, e.g., of parameter values, are the subject of future research, which can be attempted after this paper focuses on establishing that the numerical method is convergent and its implementation efficient, and that crucial aspects of the model work correctly, e.g., mass conservation and the self-generation of calcium waves. To focus on this validation is another reason why we restrict ourselves to the same model and model parameters as in [12].

The work [12] uses the code MPSalsa (www.cs.sandia.gov/CRF/MPSalsa, [20]) from Sandia National Laboratories as a simulation platform. This general-purpose code is designed for more general problems than (1.1) by including additional phenomena (such as heat and mass transfer, etc.) and uses unstructured finite element meshes; thus it is potentially much more flexible in representing realistic geometries. It will become apparent in the detailed comparisons in section 3 that some of our choices of numerical methods are remarkably analogous, such as a low-order finite element method in space, fully implicit time stepping, the Newton method as nonlinear solver, and use of a Krylov subspace method as linear solver, but with certain meaningful differences, which makes a comparison of the methods interesting: One key difference is our use of a variable-order time-stepping method with automatic step-size control that decreases the step size when warranted to control transients in time as well as allows the step size to become large when possible. This is crucial to resolve the opening and closing transients of calcium channels and to also treat the very smooth diffusion effects as efficiently as possible with high method orders and large time steps (see section 3.2). Another key difference is that we developed a completely matrix-free implementation of the numerical method (see section 3.4). It is this approach that decreases the memory usage dramatically and thus enables the numbers of degrees of freedom to be in the millions (see Table 3.1 in section 3.1). This approach was originally proposed and the derivation of the terms detailed in [8], but that work did not split the unknowns for each species across the parallel computer and thus did not have the necessary efficiency for long-time studies of the whole cell. Another key benefit of the matrix-free methodology is that the analytical Jacobian matrix in the Newton method is exact and automatically up-to-date in each iteration without any associated cost, both of which improve the convergence behavior of the Newton iterations (see section 3.3). Together with the use of a Krylov subspace method as linear solver, this is known as the Jacobian-free Newton–Krylov (JFNK) method; see the review paper [14] and the references therein. These key features of our code permit us to compute to large final times on the entire cell within about the same wall clock times on similar computational resources as used in [12]. Another difference between the works is the shape of the domain. We use an elongated brick shape as generic approximation of the shape of a cell, while [12] uses a cylinder. Both shapes are regular in nature, and there is no inherent difference between the computational cost of approximating either of them. In the same way, [12] considers a range of values for various parameters, which also does not influence the computational cost of the simulation significantly.

Along with the software MPSalsa, mentioned above and used in [12], there exist various other packages, such as SUNDIALS (www.llnl.gov/casc/sundials, [9]) and PETSc (www.mcs.anl.gov/petsc, [1]), that contain state-of-the-art numerical algorithms for problems including (1.1) considered here. It will become clear in the following sections that we made very similar algorithmic choices as these packages. However, we still decided to implement our own special-purpose code for this application in order to have full control of all algorithmic details. One set of issues relates to the term J_{SR} in (1.1) that drives the calcium release: On the one hand, for theoretical reasons, we need to carefully ensure convergence of the spatial discretization in the face of the highly nonsmooth Dirac delta distributions in J_{SR}. We use the finite element method to this end, which results in a mass matrix in the ODE system and thus needed an ODE solver that incorporates a mass matrix. It is also important that we can test various versions of the mass matrix before confirming that a lumped mass matrix is sufficient to guarantee accuracy and convergence [8]. On the other hand for practical reasons, we will want to control the time stepping of the ODE method to respect the spark times at which the calcium release units can open or close, which we are able to do in our own code more easily. It will also become clear that we need to be very mindful to limit the use of memory as much as possible to facilitate the solution of problems on the desired high resolution meshes, and this is easier in a special-purpose code. Additionally, we have and are continuing to investigate method choices that make the code more efficient for this application, for which full control of the code is necessary. One example of this is the use of the NDFk method family for time-stepping following [21], instead of the more traditional BDFk family in some of the alternative packages such as SUNDIALS. The idea is that the NDFk methods can use significantly larger time steps than the BDFk methods [21], resulting in significant savings of wall clock time. This is clearly very important for the long-time studies intended here and has been confirmed to be effective for this diffusion-dominated flow. Another example is the use of the QMR method as linear solver instead of the more typically chosen, more general GMRES, in order to limit the memory usage more tightly.

The following section 2 introduces the application problem in more detail and highlights the numerical challenges that need to be addressed. Section 3 explains the choices for the components of our numerical method in detail and discusses some comparisons with the simulations in [12]. Section 4 presents results in four sections: Parallel performance studies in section 4.1 demonstrate that we can solve problems of the desired size efficiently on a distributed-memory cluster with low-latency interconnect. Sections 4.2 and 4.3 present convergence studies with smooth and nonsmooth source terms, respectively, to validate the method, its implementation, and our choice of numerical parameters. Finally, section 4.4 shows long-time simulations for two cases, one in which no calcium waves self-organize and one in which two waves self-organize and traverse the domain. Movies of the results are available at the author's website www.math.umbc.edu/˜gobbert/calcium. Our conclusions are summarized in section 5.

2. The application problem.

2.1. The model of the spark mechanism. The key term of the model is the term $J_{\text{SR}}(u^{(0)}, \mathbf{x}, t)$ in the equation for the calcium concentration (labeled as species $i = 0$) in (1.1) that describes the release of calcium at the calcium release units (CRUs), referred to as spark events [11, 13]. On the spatial scale of a cell, the CRUs appear as discrete points distributed uniformly throughout the cell. Specifically, we take the arrangement of the CRUs as a three-dimensional lattice with spacings of $\Delta x_s = \Delta y_s = 0.8$ μm in the x- and y-dimensions and of $\Delta z_s = 2.0$ μm in the z-dimension of the cell with no CRUs on the boundary of the cell [13, p. 105]. These and all other coefficient values and their units are collected in Table 2.1. For our domain of $\Omega = (-6.4, 6.4) \times (-6.4, 6.4) \times (-32.0, 32.0)$ with length units of μm, this means that the CRUs form a $15 \times 15 \times 31$ lattice with a total of 6,975 CRUs in the cell. It is immediately clear that one of the challenges for simulations of the calcium flow throughout an entire cell is the need for a numerical mesh that resolves this lattice of points, where the sources of calcium are located.

The release of calcium concentration at each CRU is modeled as a point source on the spatial scale of the cell, mathematically represented as a Dirac delta distribution $\delta(\mathbf{x}-\hat{\mathbf{x}})$ for a CRU located at $\hat{\mathbf{x}}$ [11, p. 89]. The Dirac delta distribution is understood here in a three-dimensional sense for short, that is, $\delta(\mathbf{x}-\hat{\mathbf{x}}) := \delta(x-\hat{x})\,\delta(y-\hat{y})\,\delta(z-\hat{z})$, where we also write $\mathbf{x} = (x, y, z)$ and $\hat{\mathbf{x}} = (\hat{x}, \hat{y}, \hat{z})$. We recall that $\delta(\mathbf{x})$ is defined by requiring (i) $\delta(\mathbf{x} - \hat{\mathbf{x}}) = 0$ for all $\mathbf{x} \neq \hat{\mathbf{x}}$ and (ii) $\int \psi(\mathbf{x})\,\delta(\mathbf{x} - \hat{\mathbf{x}})\,d\mathbf{x} = \psi(\hat{\mathbf{x}})$ for any continuous function $\psi(\mathbf{x})$; this definition implies, in particular, that $\delta(\hat{\mathbf{x}})$ tends to ∞ and is thus not a function in the mathematical sense. The amount of calcium injected into the cell at one point $\hat{\mathbf{x}}$ is given by the flux density σ, that is, $\int_\Omega \sigma\,\delta(\mathbf{x}-\hat{\mathbf{x}})\,d\mathbf{x} = \sigma$, by the definition of the delta distribution, gives the amount of calcium released into the cell in 1 ms. The effect of a CRU switching on and off is incorporated by an indicator function in time. More specifically, let the set $\Omega_s = \{\hat{\mathbf{x}} \in \Omega \,|\, \hat{\mathbf{x}} \text{ is a CRU}\}$ denote the set of all CRU locations. Then [12, p. 96]

$$J_{\text{SR}}\left(u^{(0)}, \mathbf{x}, t\right) = \sum_{\hat{\mathbf{x}} \in \Omega_s} \sigma\, S_{\hat{\mathbf{x}}}\left(u^{(0)}, t\right)\, \delta\left(\mathbf{x} - \hat{\mathbf{x}}\right) \tag{2.1}$$

is the superposition of the release of calcium at all CRUs. Since the Dirac delta distribution is a highly nonsmooth term, one critical question for the numerical method is whether its spatial discretization converges.

The indicator function $S_{\hat{\mathbf{x}}}(u^{(0)}, t)$ in each term of the sum in (2.1) houses the stochastic aspect of the sparking mechanism of the CRU at $\hat{\mathbf{x}}$. The model allows the

Table 2.1
Coefficients of the application problem with $n_s = 3$ species. The concentration unit M is short for mol/L (moles per liter).

Domains in space and time		
$\Omega = (-6.4, 6.4) \times (-6.4, 6.4) \times (-32.0, 32.0)$ in units of μm		
$0 \leq t \leq t_{\text{fin}}$ with $t_{\text{fin}} = 1{,}000$ in units of ms		
Reaction-diffusion equation (1.1)		
$D^{(0)} = \text{diag}(0.15, 0.15, 0.30)$ $\mu\text{m}^2/\text{ms}$		
$D^{(1)} = \text{diag}(0.01, 0.01, 0.02)$ $\mu\text{m}^2/\text{ms}$		
$D^{(2)} = \text{diag}(0.00, 0.00, 0.00)$ $\mu\text{m}^2/\text{ms}$		
$a^{(i)} = 0$, $f^{(i)} \equiv 0$ for all i		
$u^{(0)}_{\text{ini}} = 0.1$ μM,	$u^{(1)}_{\text{ini}} = 45.9184$ μM,	$u^{(2)}_{\text{ini}} = 111.8182$ μM
CRU coefficients, (2.1), and (2.2)		
$\Delta x_s = 0.8$ μm,	$\Delta y_s = 0.8$ μm,	$\Delta z_s = 2.0$ μm
$\sigma = \begin{cases} 51.8213655\ \mu\text{M}\ \mu\text{m}^3/\text{ms} & \text{for } I_{\text{SR}} = 10 \text{ pA} \\ 103.6430533\ \mu\text{M}\ \mu\text{m}^3/\text{ms} & \text{for } I_{\text{SR}} = 20 \text{ pA} \end{cases}$		
$F = 96{,}485.3$ C/mol Faraday constant		
$P_{\max} = 0.3/\text{ms}$,	$K_{\text{prob}} = 15.0$ μM,	$n_{\text{prob}} = 1.6$
$\Delta t_s = 1.$ ms,	$t_{\text{open}} = 5.0$ ms,	$t_{\text{closed}} = 100.0$ ms
Reaction terms (2.4)		
$k_1^+ = 0.08/(\mu\text{M ms})$,	$k_1^- = 0.09/\text{ms}$,	$\overline{u}_1 = 50.0$ μM
$k_2^+ = 0.10/(\mu\text{M ms})$,	$k_2^- = 0.10/\text{ms}$,	$\overline{u}_2 = 123.0$ μM
Pump and leak terms (2.6)		
$V_{\text{pump}} = 0.2$ μM/ms,	$K_{\text{pump}} = 0.184$ μM,	$n_{\text{pump}} = 4$
$J_{\text{leak}} = 0.016048418$ μM/ms		

CRU to open with probability

$$J_{\text{prob}}\left(u^{(0)}\right) = \frac{P_{\max}\left(u^{(0)}\right)^{n_{\text{prob}}}}{(K_{\text{prob}})^{n_{\text{prob}}} + \left(u^{(0)}\right)^{n_{\text{prob}}}} \tag{2.2}$$

as a function of the local calcium concentration $u^{(0)}$ [13, p. 104]. This probabilistic model is checked at the spark times that are every unit in time $\Delta t_s = 1$ ms apart. If a CRU opens at a time $t = \hat{t}$, it stays open for a duration $t_{\text{open}} = 5$ ms; that is, mathematically the indicator function $S_{\hat{\mathbf{x}}}$ is set to 1 for $t \in [\hat{t}, \hat{t} + t_{\text{open}}]$. The desired effect of this design is that the calcium released at one CRU diffuses to a neighboring CRU, whose probability for opening increases with the increased calcium concentration. If the calcium concentration then reaches a third CRU and it opens, the effect is that of a wave forming throughout the cell [8, 13]. After a CRU closes again, it cannot release calcium again for a time period $t_{\text{closed}} = 100$ ms. Therefore, calcium waves through the cell are separated in time by at least 100 ms. We see that to simulate a sequence of repeated calcium waves, we need to be able to calculate for long times, such as, up to the final time $t_{\text{fin}} = 1{,}000$ ms. The studies in section 4.4 use the model parameters in Table 2.1 with a maximum probability $P_{\max} = 0.3/\text{ms}$ for a CRU to open and spark intervals of $\Delta t_s = 1$ ms, following the original work [13]. These results were checked against several theoretically equivalent choices, including among others a maximum probability of $P_{\max}/4$ every 0.25 ms. These simulations gave equivalent results, aside from expected variations due to different sequences of pseudo random numbers in each simulation.

The experimentally obtained coefficient σ models the amount of calcium released at one CRU [11, 13]. It is a function of the calcium current I_{SR} by $\sigma = I_{\text{SR}}/(2F)$, where F denotes the Faraday constant. The range of I_{SR} from 10 to 20 pA is "back-calculated from the size of sparks" [12, p. 96]. This quantity has crucial influence on whether calcium waves self-organize or not because it determines how much calcium is released into the cell at one CRU, σ, which via diffusion raises the value of J_{prob} in (2.2) at nearby CRUs and thus influences strongly whether they open or not. One interesting validation for this model of calcium waves is to consider the extreme values of this range for I_{SR} and observe whether calcium waves self-organize.

In summary, there are several key challenges for the numerical method and its implementation: The spatial discretization needs to resolve the cell domain with a fine mesh, and we need to ensure its convergence in the face of the Dirac delta distributions as highly nonsmooth source terms. The time discretization needs to ensure small error when CRUs are switched on and off, and needs to be efficient enough to allow long-time simulations within reasonable wall clock times.

2.2. The other terms of the application problem. The model for the calcium flow involves, in addition to calcium C, labeled as species $i = 0$, additionally an endogenous calcium buffer $\mathrm{F}^{(1)}$, labeled $i = 1$, and a fluorescent indicator dye $\mathrm{F}^{(2)}$, labeled $i = 2$. The reversible binding/unbinding of the indicator and buffer species are modeled by the reaction model for the $n_s = 3$ species [11]

$$\mathrm{F}^{(i)} + \mathrm{C} \rightleftharpoons \mathrm{G}^{(i)}, \qquad \text{for } i = 1, \dots, n_s - 1, \tag{2.3}$$

where $\mathrm{F}^{(i)}$ denotes the free molecules of species i, C denotes the calcium species, and $\mathrm{G}^{(i)}$ the molecules of species i that are bound with C. The reaction model, together with the no flow boundary conditions, assures the conservation of the total concentrations of $\mathrm{F}^{(i)}$ and $\mathrm{G}^{(i)}$ together at a value $\overline{u}_i$ [11] that is determined by the initial conditions. With $u^{(i)}$ denoting the concentration of $\mathrm{F}^{(i)}$, the concentration of $\mathrm{G}^{(i)}$ is then $\overline{u}_i - u^{(i)}$. And with $u^{(0)}$ as the concentration of C, the reaction rates are

$$R^{(i)} = -k_i^+ u^{(0)} u^{(i)} + k_i^- \left(\overline{u}_i - u^{(i)}\right) \qquad \text{for } i = 1, \dots, n_s - 1. \tag{2.4}$$

The reaction terms $r^{(i)}$ in the unified notation of (1.1) are then for all species

$$r^{(i)}\left(u^{(0)}, \dots, u^{(n_s-1)}\right) := \begin{cases} \sum\limits_{j=1}^{n_s-1} R^{(j)}\big(u^{(0)}, u^{(j)}\big), & \text{for } i = 0, \\ R^{(i)}\left(u^{(0)}, u^{(i)}\right), & \text{for } i = 1, \dots, n_s - 1. \end{cases} \tag{2.5}$$

These reaction terms introduce nonlinearity into the problem and constitute the coupling between the equations in (1.1). Notice that we follow the convention of [11, 12, 13] of tracking the buffer species in their bound state. By contrast, [18] tracks the buffers in their unbound state, resulting in apparent sign reversions in all reaction terms.

Finally, calcium also leaves the cell through the nonlinear drain term [11, p. 89]

$$J_{\text{pump}}\left(u^{(0)}\right) = \frac{V_{\text{pump}}\,\left(u^{(0)}\right)^{n_{\text{pump}}}}{(K_{\text{pump}})^{n_{\text{pump}}} + \left(u^{(0)}\right)^{n_{\text{pump}}}}. \tag{2.6}$$

At rest, that is, for the calcium concentration $u^{(0)} = 0.1\ \mu$M, the constant source term J_{leak} balances the pump term such that $J_{\text{leak}} = J_{\text{pump}}(u^{(0)})$ [11, p. 89]. (In [12, p. 96], there is a typo in the sign of the pump term; cf. the original source [11, p. 89].) The rest concentration $u^{(0)} = 0.1\ \mu$M is chosen as the initial condition for the calcium concentration, and the initial conditions for the other reactive species are computed such that $r^{(i)} = 0$ for all reactions.

TABLE 3.1
A finite element mesh with $N_x \times N_y \times N_z$ elements has $N = (N_x + 1)(N_y + 1)(N_z + 1)$ nodes and $n_{eq} = n_s N = 3N$ degrees of freedom (DOF) for the application problem with $n_s = 3$ species.

$N_x \times N_y \times N_z$	N	$n_{eq} = \text{DOF}$
$16 \times 16 \times 64$	18,785	56,355
$32 \times 32 \times 128$	140,481	421,443
$64 \times 64 \times 256$	1,085,825	3,257,475
$128 \times 128 \times 512$	8,536,833	25,610,499
$256 \times 256 \times 1024$	67,700,225	203,100,675

3. The numerical method.

The problem (1.1), for which a numerical method needs to be developed, is a system of reaction-diffusion equations coupled through the nonlinear reaction terms. We use a method of lines approach to develop a special-purpose code that can handle the Dirac delta distributions in (2.1) and that is suitable for long-time simulations on the desired fine meshes.

3.1. Spatial discretization.

Recall that $\Omega = (-X, X) \times (-Y, Y) \times (-Z, Z)$ with $X = Y = 6.4$ and $Z = 32.0$ has a regular shape and that the calcium release units (CRUs) at which the Dirac delta distributions are centered form a regular lattice Ω_s inside the domain Ω with spacings of Δx_s, Δy_s, and Δz_s between the CRUs in the three coordinate directions. We take advantage of this structure by using a uniform numerical mesh $\Omega_h \subset \overline{\Omega}$ with mesh spacings Δx, Δy, and Δz chosen such that Ω_h includes the CRU locations as mesh points; that is, we have $\Omega_s \subset \Omega_h \subset \overline{\Omega}$ by design. More specifically, the mesh has $N_x \times N_y \times N_z$ brick-shaped elements, and thus there are a total of $N = (N_x+1)(N_y+1)(N_z+1)$ nodes in the mesh. For a problem with $n_s = 3$ chemical species as in the application problem, there will be $n_{eq} = n_s N = 3N$ degrees of freedom (DOF) to be solved for in each time step. Table 3.1 lists several resolutions that we will use in the following and their associated number of nodes N and degrees of freedom n_{eq}. The simulations shown in section 4.4, in fact, use the $64 \times 64 \times 256$ mesh and have thus "only" over 3 million DOFs. However, the finer meshes are crucial to study the convergence of the spatial discretization, and the reference solution in sections 4.2 and 4.3, in fact, used the $256 \times 256 \times 1024$ mesh, albeit in a single-species run, so having as DOFs the nodes N of "only" nearly 68 million listed in the table. What these numbers demonstrate is that an off-the-shelf software such as COMSOL Multiphysics (formerly known as FEMLAB; www.comsol.com) cannot be used to solve this problem: COMSOL is a serial code, thus all variables used must fit into the memory of the computer used. Due to large numbers of auxiliary variables used inside its general-purpose implementation of the finite element method on an unstructured mesh, it runs out of memory rather rapidly, such as with about 20,000 elements on 0.5 GB of memory [5] or 200,000 elements on 1.0 GB of memory [6]; these observations depend on the polynomial order of finite elements used as well as on various other factors such as choice of solver, and should only be considered a rough guide. However, an additional limitation is that in time-dependent problems COMSOL stores all solutions over time in memory. Even if this is restricted to the points in time, where the output of the solution is desired, this would mean about 1,000 solutions for the present problem, if we store the solution merely at every spark time; this clearly becomes infeasible for any reasonable mesh resolution. All this should not be taken as an indictment of COMSOL. Rather, its strength lies in providing ready access to a variety of different finite elements, many higher-order than the linear elements used here, whose implementation is difficult and justifiably requires the large numbers of auxiliary variables used. But a problem with special features such as the one under consideration might require a special-purpose code to achieve the desired resolution.

The need to discretize the Dirac delta distributions in (2.1) motivates the use of the finite element method (FEM), because the weak formulation of the FEM is obtained by integrating (1.1) over Ω against a smooth test function, which is well-defined for the Dirac delta distributions in J_{SR}. We choose trilinear nodal FEM basis functions $\varphi_k(\mathbf{x})$, $0 \le k < N$, which satisfy $\varphi_k(\mathbf{x}_\ell) = \delta_{k\ell}$ for all nodes $\mathbf{x}_\ell \in \Omega_h$ and are linear functions of each variable between nodes. Using φ_k as a test function in the weak formulation allows for an explicit evaluation of the J_{SR} term in the discretization, and we find

$$(3.1) \qquad \Sigma_k := \int_\Omega J_{\text{SR}}\, \varphi_k \, d\mathbf{x} = \sigma \sum_{\hat{\mathbf{x}} \in \Omega_s} S_{\hat{\mathbf{x}}} \int_\Omega \delta(\mathbf{x} - \hat{\mathbf{x}})\, \varphi_k(\mathbf{x})\, d\mathbf{x} = \sigma \sum_{\hat{\mathbf{x}} \in \Omega_s} S_{\hat{\mathbf{x}}}\, \varphi_k(\hat{\mathbf{x}}).$$

Since $\hat{\mathbf{x}} \in \Omega_s \subset \Omega_h$ by construction of the mesh, $\hat{\mathbf{x}}$ is a node, and we obtain $\varphi_k(\hat{\mathbf{x}}) = 1$ if and only if $\hat{\mathbf{x}} = \mathbf{x}_k$. Moreover, $S_{\hat{\mathbf{x}}}(u^{(0)}, t) = 1$ if the CRU at $\hat{\mathbf{x}}$ is open and 0 otherwise, so $\Sigma_k = \sigma$ if $\mathbf{x}_k$ is a CRU and it is open, and 0 otherwise.

The discretization of the other terms in (1.1) is standard, see, e.g., [16, 23] for general information, and see [8] for the concrete derivation of all terms. Let $\mathbf{u}_k^{(i)}(t) \approx u^{(i)}(\mathbf{x}_k, t)$ denote the approximation to the true solution at node $\mathbf{x}_k \in \Omega_h$ at time t, then the finite element solution is denoted by $u_h^{(i)}(\mathbf{x}, t) = \sum_k \mathbf{u}_k^{(i)}(t)\, \varphi_k(\mathbf{x})$. The methods of lines approach yields the semi-discrete problem in vector form for $\mathbf{u}^{(i)} = (\mathbf{u}_k^{(i)})$ as

$$(3.2) \quad \hat{M}\, \frac{d\mathbf{u}^{(i)}}{dt} = -\left(K^{(i)} + M_a^{(i)}\right) \mathbf{u}^{(i)} + \mathbf{r}^{(i)} + \delta_{i0}\, (\mathbf{j}_{\text{pl}} + \Sigma) + \mathbf{f}^{(i)}, \quad i = 0, 1, \ldots, n_s - 1.$$

Here, $K^{(i)}$ denotes the stiffness matrix, $\hat{M}$ is the lumped mass matrix (which is the same for all species) [23], and $M_a^{(i)}$ is mass matrix involving the constant $a^{(i)}$ (and is thus species dependent). The remaining vector terms come from the discretization of their corresponding terms in (1.1), with Σ from above and the pump and leak terms combined in $\mathbf{j}_{\text{pl}}$.

The error in the finite element solution $u_h(\cdot, t)$ of the semi-discrete problem (3.2) is measured in the $L^2(\Omega)$-norm. Under standard assumptions [16, 23], it has the form

$$(3.3) \qquad \|u_h(\cdot, t) - u(\cdot, t)\|_{L^2(\Omega)} \le C\, h^q, \quad \text{as } h \to 0,$$

where $q > 0$ for convergence and C denotes a generic constant of moderate size independent of the maximum mesh spacing $h := \max\{\Delta x, \Delta y, \Delta z\}$. To guarantee the optimal convergence order for linear FEM shape functions of $q = 2$ in (3.3), the source terms of (1.1) need to lie in the function space $L^2(\Omega)$. This is not the case for the application problem because of the Dirac delta distributions in J_{SR}. In [8], we have in the past argued heuristically and demonstrated computationally that convergence does hold for linear basis functions also in this case, with a convergence order of $q = 0.5$. Notice that we cannot expect better convergence for higher-order elements, which explains our use of the low-order linear finite elements. More recent work extends these heuristic arguments rigorously and presents numerical tests in all space dimensions [6].

The simulations in [12] use the same finite elements, but on an unstructured mesh of tetrahedra and hexahedra with about 200,000 DOFs. A small difference is that their domain is of cylindrical shape with a diameter slightly smaller than our x-y cross section. The significant difference is that their domain is really a cross section of the cell in the z-direction, encompassing only two x-y planes of CRUs [12, pp. 96–97].

3.2. Time discretization. The next step in the method of lines approach is the time discretization for (3.2), which becomes a standard initial value problem $M^{(ode)}y'(t) = f^{(ode)}(t, y)$ with mass matrix, if the unknown vectors $\mathbf{u}^{(i)}(t)$ for all species are concatenated into one vector $y(t)$ of unknowns. This is one large problem of $n_{eq} = n_s N$ nonlinear ordinary differential equations (ODEs). For reaction-diffusion equations with second-order spatial derivatives, the time step restrictions due to the CFL condition are generally considered too severe to allow for explicit time-stepping methods. One might also try various splitting methods that decouple the ODEs and solve the ODEs for only one species $\mathbf{u}^{(i)}(t)$ at a time. In [8], we tried a very simple splitting method by lagging the reaction terms in time, which also makes each smaller ODE system linear for this application problem. It turns out that this approach has degraded stability properties and might also not conserve mass as well as desired. In the terminology of [17], which reports extensive tests of time-stepping methods for problems of the type considered here, a "fully implicit and balanced" method is preferable, in which the ODE system is discretized fully implicitly (and thus does not decouple by species) and the discretization has all reaction terms at the same time. The same applies to the pump and leak terms; however, the CRU term J_{SR} is necessarily taken explicitly to implement its probabilistic model.

The reaction-diffusion problem (1.1) is very smooth as such, hence when using an implicit time stepping, we expect to be able to take fairly large time steps most of the time. However, the J_{SR} term in (2.1) also includes the $S_{\hat{\mathbf{x}}}$ functions at all CRUs that may switch from 0 to 1 or vice versa at the spark times. At these times, a small time step will be necessary to enable the nonlinear and linear solvers to converge and also to control the ODE error reasonably. Therefore, to reach the very large final times desired, we must adopt some variable time-stepping strategy with automatic error control. It is understood that the theory behind any error control will probably not be rigorously valid at the spark times due to the discontinuities in the source terms. One approach would be to simply set Δt to a small value manually at these times. We are not using any special strategy at the moment, because the error control appears to overcome the spark times successfully without it. We must now expect that the computer code to be developed will be quite complex. With this being the case anyway, we decided to also implement a variable-order scheme with automatic selection of the ODE method order. Concretely, we use the numerical differentiation formulas (NDFk) with method order $1 \leq k \leq 5$ from [21], which are generalizations of the well-known BDFk methods and are the basis of MATLAB's `ode15s` function (www.mathworks.com).

We use relative and absolute tolerances of 10^{-6} and 10^{-8}, respectively, on the relative and absolute error estimators of the ODE method. In the application studies in section 4.4, the steps vary widely in size, with small steps on the order of 10^{-5} ms $= 0.01$ μs immediately after CRUs open or close; this means that the time scales of channel openings or closings of about 1 μs [18] are well-resolved. But since the reaction-diffusion system is very smooth away from these times, the time steps increase steadily afterwards up to 10^{-1} ms, all the while the error controller ensures that the total error incurred from the time-stepping remains bounded by the selected tolerances. This high variation in step size allows the solver to reach the desired final time of 1,000 ms in under 60,000 time steps. The most interesting observation regarding the ODE solver is that the average method order is about 3, with typical orders ranging from 2 to 4. This shows that we are definitely profiting significantly from the variable order method, as compared to using fixed order methods such as implicit Euler or the trapezoidal rule in time. This is the case in [12], where the trapezoidal rule with a fixed step size of 0.01 without error controller takes 10,000 steps to reach the final time of 100 ms. Analogously, if we only computed to 100 ms, we would need about 6,000 time steps and would still have a smaller error, owing to the smaller time steps used, whenever opening or closing channels introduce the largest variation in the solution.

3.3. The nonlinear solver. One price of the fully implicit time discretization is that we have to solve the fully coupled nonlinear system of $n_s N$ equations that arises from discretizing (3.2) in time. We choose the Newton method and follow the control structure for managing error control and convergence in MATLAB's `ode15s` function. We also profit from the low-order spatial discretization on a uniform mesh used, because we are able to compute analytically all matrices in (3.2) [8]. We use this here to supply an analytic Jacobian to the Newton method. Since the matrix-vector products in the Krylov subspace method used as linear solver below are implemented in matrix-free form, this Jacobian is automatically evaluated at the current Newton step without any additional cost. This combination of a Newton method with a Krylov subspace method as linear solver and a matrix-free Jacobian is known as the Jacobian-free Newton–Krylov (JFNK) method [14]. Besides the obvious memory savings associated with *not* storing any matrix, the up-to-date Jacobian also has the potential to accelerate convergence of the nonlinear solver.

A classical problem encountered when solving reaction-diffusion equations numerically is the problem maintaining the nonnegativity of the numerically computed concentrations. Clearly, they should be nonnegative physically. Also, it can be rigorously established that the unique true solution to the problem (1.1) along with its boundary and initial conditions is nonnegative [10]. In combination with a suitable spatial discretization, such as ours, it can be shown that an implicit Euler time discretization admits a nonnegative solution [10]. Notice, however, that it is well-understood that despite this fact, the Newton method does not necessarily find this nonnegative solution, even when started with the nonnegative solution at the previous time step as initial guess [10, 19].

Techniques such as clipping, in which negative solution components are set to 0, clearly degrade mass conservation properties, because these corrections add mass whenever they increase a component from a negative value to zero. Recently, MATLAB has introduced a nonnegativity preserving feature in its `ode15s` function, discussed in [22]. Its idea is to modify the right-hand side function of the ODE problem (3.2) such that it returns 0 instead of any negative slope; the intention is to have the solution "follow the constraint" [22]. This technique is applied on the level of the ODE solver, though. This means that the Newton steps themselves can still have negative components. This can be a problem, for instance, if the right-hand side function of the ODE involves square roots or other expressions that become invalid even for small negative values. Therefore, we are using a line-search strategy for the Newton step: If a negative solution component is encountered in a full Newton step, it determines the smaller step size necessary to ensure nonnegativity of all components [7]. This strategy is also applied to the computation of the initial guess of each Newton solve, which thus can still be based on the predictor formula usually used in the NDFk methods [21], as opposed to having to use the solution at the previous time step. Analogously to MATLAB's strategy, our approach is free of cost if all components are nonnegative and only of modest cost otherwise. We note that we have run studies with and without nonnegativity preservation and found that it is not crucial for this application problem, as none of the coefficient functions becomes invalid, and the solution in fact recovers on its own from negative components. We use a relatively tight tolerance of 10^{-8} in the Newton solver, but it still converges typically within 2 steps on average due to the tight ODE tolerance and the good initial guess thus available. This agrees with the observations in [12], where also 1 to 2 Newton steps are reported.

3.4. The linear solver. At each Newton step, we need to solve a linear system of equations, also of size $n_{eq} = n_s N$. Here, we profit directly from the fact that we are able to compute analytically all matrices in (3.2), because this enables a matrix-free implementation of products of vectors with any of these matrices, which is all that is needed for each iteration of the Krylov subspace methods. The obvious advantage of not storing any system matrix is its memory savings. For the finite element method used, there would be $3 \times 3 \times 3 = 27$ nonzero bands in the system matrix, each requiring about $n_{eq} = n_s N$ entries to store, where Table 3.1 lists these numbers for various mesh resolutions. Moreover, as already explained in the previous section, another advantage of our matrix-free implementation is that the system matrix in each iteration of the Newton method is automatically evaluated at the most recent solution without any cost incurred for this feature.

Based on a decision tree in [3, p. 321] (nonsymmetric system matrix, with matrix transpose available) and backed up by tests of various Krylov subspace methods in MATLAB, we choose the quasi-minimum residual (QMR) method for our nonsymmetric problem. We usually use a tolerance of 10^{-3} on the relative residual and a stagnation tolerance of 10^{-14} for the iterative linear solver.

We do not use any preconditioning presently, because a matrix-free preconditioner that avoids parallel communications was not readily apparent. This is probably not a significant problem, since the tests show that the linear solver tends to converge within fewer than 4 iterations on average due to the tight ODE and Newton tolerances enforced. This compares reasonably to the results reported in [12], where a GMRES method, with an approximate incomplete LU preconditioner on each parallel subdomain, took fewer than 10 iterations.

3.5. Parallel implementation. Parallel computing is a crucial ingredient to our code for two reasons: (i) It enables the simulations on the fine meshes listed in Table 3.1, and (ii) it speeds up the computations sufficiently to enable simulations up to large final times within a reasonable amount of wall clock time. The code is written in C with MPI commands for the parallel communications for maximum portability. We split the domain Ω into nonoverlapping subdomains, with one on each of the P parallel processes, by cutting in the long dimension of Ω. This choice makes the x-y planes of nodes whose values need to be exchanged between neighboring processes as small as possible, i.e., is the optimal decomposition in a graph partitioning sense. Our code is capable of using any number of parallel processors. The communications between neighbors occur in each matrix-vector product and are implemented by nonblocking `MPI_Isend/MPI_Irecv` commands. These have proven to be faster than blocking communication commands on our system and as fast as any other MPI point-to-point communication commands available. `MPI_Allreduce` commands are needed for all norm computations as well as for various diagnostic quantities such as minimum and maximum values of the solutions. The wall clock times are measured using `MPI_Barrier` and `MPI_Wtime` in sequence. We use the `genrand()` function from

[15], with different seeds on each parallel process, to generate sequences of uniformly distributed pseudo-random numbers.

The simulations reported below were performed on the cluster kali in the Department of Mathematics and Statistics at the University of Maryland, Baltimore County. This distributed-memory cluster has 32 compute nodes, each with two 2.0 GHz Intel Xeon CPUs and 1 GB of memory. One of the compute nodes also serves as storage node and is connected to a 0.5 TB RAID array. The compute nodes are connected by a low-latency Myrinet interconnect as well as by a fast ethernet for file serving. An additional management and user node connects the cluster to the outside network. The cluster runs the Linux RedHat EL3 operating system, and the Intel compiler suite is used for this code. See www.math.umbc.edu/~gobbert/kali for more information. For the application studies, we typically use approximately 16 dual-processor nodes. For the simulations up to the final time $t_{\text{fin}} = 1{,}000$ ms in section 4.4, the code ran about 2 to 4 hours for the $32 \times 32 \times 128$ mesh and about 20 to 40 hours for the $64 \times 64 \times 256$ mesh. Thus, for a simulation to only 100 ms, we would expect about 2 to 4 hours on 16 dual-processor nodes for the finer mesh. This is clearly faster than the times in the range of 12 to 36 hours reported in [12] using comparable computational resources of from 4 to 16 dual-processor nodes with processor speeds from 1 to 3 GHz.

4. Results.

The numerical results are presented in four sections. The results in section 4.1 analyze first what size of problem the parallel code will be able to solve and that it does so efficiently. This is used in the following sections 4.2 and 4.3 because to compute the reference solution on the finest mesh requires the parallel code. These sections summarize results from convergence studies to confirm that the numerical method is reliable, for a linear test problem with a smooth source term in section 4.2 and with a single calcium release unit in section 4.3. Finally, section 4.4 presents two representative long time simulations of the full application problem up to the large final time $t_{\text{fin}} = 1{,}000$ that show that our method allows for the thorough investigation of the given model.

4.1. Parallel performance results.

The first purpose of parallel computing is to enable the solution of larger problems. To demonstrate this ability, Table 4.1 (a) estimates the amount of memory in MB per process required to solve the application problem with $n_s = 3$ species on the meshes listed in Table 3.1 that lists the degrees of freedom n_{eq} for each mesh. Table 4.1 (a) is based on a count of the variables with major memory usage in the code. This count reveals that the ODE solver uses $13 + K$ vectors of length n_{eq}. Here, $1 \leq K \leq 5$ denotes a chosen maximum method order for the NDFk, $1 \leq k \leq K$, method, which is commonly $K = 5$; we use this value, as well, but it has turned out that the application problem uses only orders up to 4, so we could save a little memory here by selecting K smaller. By reusing 5 auxiliary vectors from the ODE solver, the QMR method requires only 3 additional vectors. Thus the total number of vectors of length n_{eq} in the code is $16 + K = 21$ for $K = 5$. For each of the lengths $n_{eq} = n_s N$ specified in Table 3.1 and with 8 B per double-precision number, we obtain the total amount of memory needed for each mesh resolution in the $P = 1$ processor column of Table 4.1 (a). We note already that resolutions finer than $64 \times 64 \times 256$ cannot be accommodated on a single node of our system. This memory gets divided into the P processors as shown in the following columns.

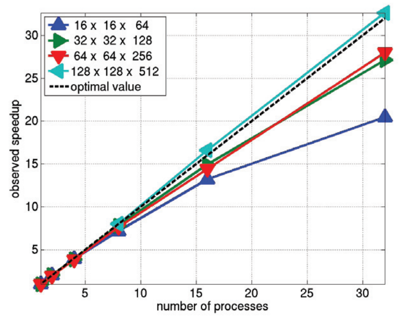
Tables 4.1 (b) and (c) list the memory in MB per process actually used by the code for each case, as observed by monitoring the output of the Linux utility `top` during the program execution. Table 4.1 (b) considers the case of "dedicated" nodes, that is, only one CPU per node is used with the second one idling. This dedicates the

TABLE 4.1
Memory usage in MB per process for the application problem with $n_s = 3$ species. The storage requirements are 21 vectors, each of length n_{eq} from Table 3.1.

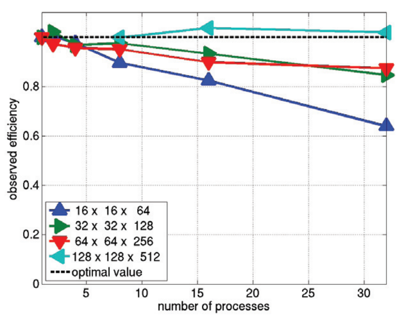
(a) Predicted memory usage in MB per process							
	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$	$P = 64$
$16 \times 16 \times 64$	9	5	2	1	1	< 1	< 1
$32 \times 32 \times 128$	68	34	17	8	4	2	1
$64 \times 64 \times 256$	522	261	130	65	33	16	8
$128 \times 128 \times 512$	4103	2052	1026	513	256	128	64
$256 \times 256 \times 1024$	32540	16270	8135	4068	2034	1017	508
(b) Observed memory usage using dedicated nodes							
	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$	$P = 64$
$16 \times 16 \times 64$	11	27	25	23	23	22	N/A
$32 \times 32 \times 128$	71	57	40	31	27	25	N/A
$64 \times 64 \times 256$	523	283	153	91	57	41	N/A
$128 \times 128 \times 512$	N/A	N/A	N/A	541	285	157	N/A
(c) Observed memory usage using nondedicated nodes							
	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$	$P = 64$
$16 \times 16 \times 64$	11	8	27	25	25	24	N/A
$32 \times 32 \times 128$	71	39	42	33	29	27	26
$64 \times 64 \times 256$	523	267	155	93	60	43	35
$128 \times 128 \times 512$	N/A	N/A	N/A	N/A	287	159	98

entire memory of each node to the process and avoids any contention for resources of the node among the 2 CPUs, but is also wasteful in the sense that half of the CPUs reserved by the scheduler for the job are idling. Table 4.1 (c) shows the memory usage observed in the case of "nondedicated" nodes, that is, with both CPUs on each node being used. Thus, the two CPUs do not have dedicated memory, but rather share the memory of the node and might also suffer from resource contention, e.g., for the use of the single Myrinet port on the node, of the memory of the node, and of the bus that connects both CPUs to all components of the node. The results in Tables 4.1 (b) and (c) are in agreement with the predictions in Table 4.1 (a), with a modest amount of baseline usage associated probably with the use of MPI functions; notice the much smaller overhead for the $P = 1$ cases. These considerations indicate problems of which size we will be able to attack, and it gives confidence in the proper understanding of the code's memory usage.

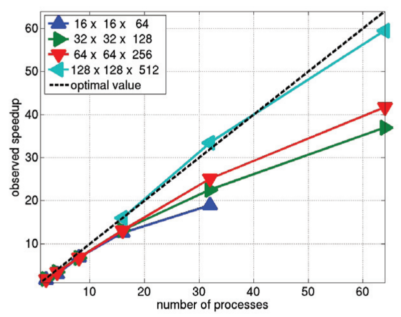
The second purpose of parallel computing is to solve a problem of a given size faster. Ideally, a run using P processors should be P times as fast as the 1-processor run. To quantify this, we first need to time the code. In the context of a true parallel code for which the processors need to communicate with each other, the correct measure of time is wall clock time T_P when using P processors. This time includes both the calculation time associated with arithmetic and similar operations that are local to a CPU and the communication time associated with the sending and receiving of messages between the parallel processes. For a fixed problem size, speedup defined as $S_P := T_1/T_P$ quantifies how much faster the P-processor run is over the 1-processor one; for the $256 \times 256 \times 1024$ mesh, the definition of speedup is modified to $S_P := 8T_8/T_P$, since the 8-processor case is the first one to fit in memory. The optimal value of S_P is P. Thus, by plotting S_P vs. P, one can get a visual impression how fast the actual performance deteriorates from the ideal one. Figure 4.1 (a) shows the speedup observed for simulations with up to 32 dedicated nodes, as described above. We see that the scalability of the code is excellent and gets better as the size of the problem increases. For the finest mesh, the results plotted are better than optimal



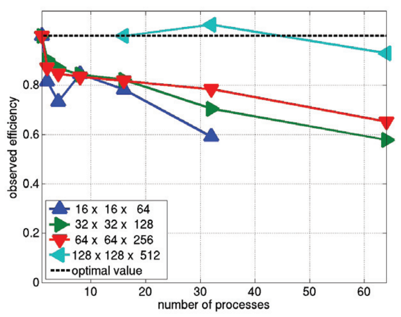
(a) Speedup



(b) Efficiency



(c) Speedup



(d) Efficiency

FIG. 4.1. (a) *Observed speedup and* (b) *observed efficiency up to* 32 *processors using dedicated nodes (only one CPU per node used).* (c) *Observed speedup and* (d) *observed efficiency up to* 64 *processors using nondedicated nodes (both CPUs per node used). Notice the different scales on the axes.*

by a small margin. This reflects the slight variability of timing results. Another way to quantify how close the speedup S_P is to its optimal value P is to plot efficiency $E_P = S_P/P$ vs. P, whose optimal value is 1. Even though Figure 4.1 (b) is directly derived from the data in the previous plot, an efficiency plot is very useful to bring out whether there is any abrupt deterioration of the parallel performance for small values of P, where a speedup plot is too cluttered to read accurately. In Figure 4.1 (b), there is no noticeable deterioration in that area; rather, the efficiency decreases slowly, but is at very respectable levels of over 80% throughout for all meshes but the coarsest one.

Figures 4.1 (c) and (d) contain the speedup and efficiency plots for the studies with nondedicated nodes associated with Table 4.1 (c). The speedup is nearly as good up to 32 processors as for the dedicated nodes; notice the different scale on the axes here. However, the efficiency plot clearly brings out an abrupt deterioration of performance, as we go from 1 processor to 2 processors; this demonstrates the usefulness of the efficiency plot, because this effect is not readily visible in the speedup plot. This effect is caused by the code running on both CPUs on one node competing for the node's resources, in particular, the single bus connecting both CPUs to memory. We have been able to reproduce this effect by running two completely independent serial jobs running on one node, so we conclude that the problem is *not* associated with the Myrinet interconnect. This effect is quite typical on commodity clusters because

Table 4.2
Convergence study for scalar test problem with smooth source term.

(a) Error on Ω against true solution (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	4.0121e–02	5.6277e–02	7.0185e–02
$32 \times 32 \times 128$	1.0100e–02 (1.990)	1.4148e–02 (1.992)	1.7650e–02 (1.992)
$64 \times 64 \times 256$	2.5074e–03 (2.010)	3.5055e–03 (2.013)	4.3869e–03 (2.008)
$128 \times 128 \times 512$	6.0012e–04 (2.063)	8.3361e–04 (2.072)	1.0591e–03 (2.050)
(b) Error on Ω against reference solution (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	3.9999e–02	5.6112e–02	6.9959e–02
$32 \times 32 \times 128$	9.9770e–03 (2.0033)	1.3984e–02 (2.0046)	1.7424e–02 (2.0054)
$64 \times 64 \times 256$	2.3849e–03 (2.0647)	3.3408e–03 (2.0655)	4.1608e–03 (2.0661)
$128 \times 128 \times 512$	4.7750e–04 (2.3204)	6.6881e–04 (2.3205)	8.3285e–04 (2.3207)

the two CPUs on one node indeed share all other components on the node with each other. One might think now that therefore it would be best to use always only one CPU per node, and this is true when comparing runs using the same total number of *CPUs*. But in practice, a user often reserves *nodes* from the scheduler in order to have dedicated access to their memory, and then a run using both CPUs per node will use twice as many CPUs and be faster than one that only uses one CPU per node, though not necessarily twice as fast. Hence, unless the memory of a node cannot accommodate a run with using both CPUs, one should use both CPUs when available. The parallel runs in the following sections use this approach.

4.2. Scalar test problem with smooth source term. We first consider briefly the scalar linear test problem $u_t - \nabla \cdot (\nabla u) = 0$, already used to test an earlier version of this code [4], obtained from (1.1) by setting $n_s = 1$ to get a scalar problem and then $D = 1$, $a = 0$, $f \equiv 0$, and all application-related functions to 0. We consider this problem on the same domain as the application problem $\Omega = (-X, X) \times (-Y, Y) \times (-Z, Z)$ with $X = Y = 6.4$ and $Z = 32.0$. This test problem also continues to use the no-flow boundary conditions of the application problem. The initial condition is specified in agreement with the chosen true solution

$$u(x, y, z, t) = \frac{1 + \cos(\lambda_x x)\, e^{-D_x \lambda_x^2 t}}{2} \, \frac{1 + \cos(\lambda_y y)\, e^{-D_y \lambda_y^2 t}}{2} \, \frac{1 + \cos(\lambda_z z)\, e^{-D_z \lambda_z^2 t}}{2}$$

using the notations $\lambda_x = x/X$, $\lambda_y = y/Y$, $\lambda_z = z/Z$, and $\mathbf{x} = (x, y, z) \in \overline{\Omega}$.

The finite element solution for this problem using linear basis functions satisfies (3.3) with $q = 2$ at every point in time t. Therefore, we expect the L^2-error of the numerical solution against the true solution to decrease by a factor 4, whenever the mesh spacings Δx, Δy, and Δz are halved. This is born out by the L^2-errors listed in Table 4.2 (a) at three representative times $t = 2, 3, 4$. Notice that this confirms that the ODE tolerances are chosen small enough to ensure that the spatial errors dominate. To formally estimate the convergence order q in (3.3) from numerically observed errors, one can use the estimation formula

$$q^{(est)} = \log_2 \left(\frac{\|u_{2h}(\cdot, t) - u(\cdot, t)\|_{L^2(\Omega)}}{\|u_h(\cdot, t) - u(\cdot, t)\|_{L^2(\Omega)}} \right). \tag{4.1}$$

The results of this formula are listed in the parentheses in Table 4.2 (a), and we note them to be consistent with $q = 2$.

Table 4.3
Convergence study for scalar test problem with nonsmooth source term.

(a) Error on Ω against reference solution (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	1.8651e+03	1.8503e+03	1.8415e+03
$32 \times 32 \times 128$	1.7120e+03 (0.124)	1.6974e+03 (0.124)	1.6951e+03 (0.120)
$64 \times 64 \times 256$	1.4537e+03 (0.236)	1.4531e+03 (0.224)	1.4529e+03 (0.222)
$128 \times 128 \times 512$	9.6843e+02 (0.586)	9.6832e+02 (0.586)	9.6829e+02 (0.585)
(b) Error on $\Omega \backslash \Omega_0$ against reference solution (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	2.6478e–01	9.8039e–01	2.2494e+00
$32 \times 32 \times 128$	1.2526e–01 (1.080)	3.7741e–01 (1.377)	6.6924e–01 (1.749)
$64 \times 64 \times 256$	3.7324e–02 (1.747)	1.1385e–01 (1.729)	1.8863e–01 (1.827)
$128 \times 128 \times 512$	8.0971e–03 (2.205)	2.3743e–02 (2.262)	3.8368e–02 (2.298)
(c) Error in species mass (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	1.4332e+00	3.6992e+00	4.4979e+00
$32 \times 32 \times 128$	1.2033e+00 (0.252)	1.2032e+00 (1.620)	1.2032e+00 (1.902)
$64 \times 64 \times 256$	3.1122e–01 (1.951)	3.1111e–01 (1.951)	3.1110e–01 (1.951)
$128 \times 128 \times 512$	7.8784e–02 (1.982)	7.8737e–02 (1.982)	7.8707e–02 (1.983)

The above procedure uses the true solution $u(\mathbf{x}, t)$, which is not available in practice. An alternative in that case is to use the numerical solution on the finest possible mesh as reference solution in place of $u(\mathbf{x}, t)$. We use here the numerical solution on the mesh $256 \times 256 \times 1024$ with maximum mesh spacing $h = \max\{\Delta x, \Delta y, \Delta z\} = 0.0625$. The results of the observed errors and the convergence order estimates in Table 4.2 (b) show agreement with the results based on the true solution above. The purpose of these tests was to validate the choice of ODE tolerances, to carefully test the code, and to confirm that the postprocessing procedure to estimate the convergence order $q^{(est)}$ only using available numerical data is reliable.

4.3. Scalar test problem with nonmooth source term. In this section, we solve the same scalar test problem as in the previous section, except that the right-hand side of (1.1) contains now the term J_{SR} from (2.1) on a CRU lattice of one single CRU centered at $\mathbf{x} = 0$ that opens at $t = 1$ and stays open for the duration of the simulation. We again use the domain of the application problem in this test. Notice that this is a scalar problem containing only the calcium equation; since the reaction terms are 0, there is no coupling to the other species. The initial condition is now $u = 0.1$, also as in the application problem. In the terminology of the application, we have 1 centered CRU in the cell that starts firing at $t = 1$ and stays open for the duration of the test. For this case, the classical finite element theory does not apply, but considerations and computational results in [6, 8] lead us to expect $q = 0.5$ in (3.3). Since no true solution is available for this problem on a finite domain, the errors in Table 4.3 (a) against a reference solution on the finest mesh $256 \times 256 \times 1024$ with maximum mesh spacing $h = \max\{\Delta x, \Delta y, \Delta z\} = 0.0625$. are computed by the postprocessing procedure tested in the previous section. They do converge, but it is hard to tell by what ratio the errors decrease. But the $q^{(est)}$ in the parentheses show that the errors decrease with a convergence order that is consistent with the expected number $q = 0.5$.

Another way to check convergence is possible by combining the results of Tables 4.3 (b) and (c). Table 4.3 (b) considers the L^2-norm on the domain Ω with a small area centered about the calcium release unit removed. Specifically, let $\Omega_0 :=$

$(-\varepsilon_x, \varepsilon_x) \times (-\varepsilon_y, \varepsilon_y) \times (-\varepsilon_z, \varepsilon_z)$ with ε_x, ε_y, ε_z chosen as the mesh spacings Δx, Δy, Δz of the coarsest mesh $16 \times 16 \times 64$, then we consider the $L^2(\Omega \backslash \Omega_0)$-norm. The errors in this norm listed in Table 4.3 (b) clearly converge quadratically again. Since we have removed Ω_0 from consideration, the question remains whether the solution at the center of the domain, where the single CRU is located, can be trusted. We answer this question by considering the total mass in the system $m_h(t) := \int_\Omega u_h(\mathbf{x}, t)\, d\mathbf{x}$ at time t; we indicate by the subscript h that $m_h(t)$ is the mass associated with the numerical solution u_h on mesh Ω_h. For this scalar problem with no flow boundary conditions and the release unit as the only source, we know that this mass should equal the mass at the initial time $m(0) = \int_\Omega u_{\text{ini}}(\mathbf{x})\, d\mathbf{x}$ plus the mass released into the cell up to time t given by $\int_0^t \sigma\, n_{\text{open}}(t')\, dt'$, where $n_{\text{open}}(t')$ denotes the number of open CRUs at time t'. Since the sole CRU in this system opens at $t = 1$, the latter integral is equal to $(t-1)\,\sigma$ for $t \geq 1$. Table 4.3 (c) lists the errors $|m_h(t) - m(0) - (t-1)\,\sigma|$ for $t = 2, 3, 4$, and we observe quadratic convergence for this quantity; note that the apparently large values for the mass error must be viewed in the context of the large size of the domain, which is (12.8)(12.8)(64.0) = 10,485.76. Thus, we conclude from Table 4.3 (c) that mass is conserved by the numerical method, which shows together with the convergence on $\Omega \backslash \Omega_0$ in Table 4.3 (b) that also the error at $\mathbf{x} = 0$ is converging.

4.4. Case studies for the full application problem. In this section, we present two case studies for the full application problem described in section 2. Recall from section 2.1 that the calcium current through a CRU, I_{SR}, determines the amount of calcium released into the cell through an open CRU and thus influences whether a calcium wave self-organizes or not [12]. We consider here the values $I_{\text{SR}} = 10$ pA and $I_{\text{SR}} = 20$ pA resulting in the corresponding values of σ in Table 2.1. The initial concentration of calcium is chosen at rest, $u^{(0)} = 0.1$ μM, throughout the cell and the other species concentrations computed such that $r^{(i)} = 0$ for all i. Also, we have $J_{\text{leak}} = J_{\text{pump}}$ for $u^{(0)} = 0.1$ μM. Initially, all CRUs are closed; hence, $J_{\text{SR}} = 0$. The numerical parameters used for the studies are specified and discussed in section 3.

Since no CRUs are triggered artificially, the first test for the model is whether any CRUs will open randomly and whether the diffusion of the calcium released causes any neighboring CRUs to open. Figures 4.2 and 4.3 show results for the case study with $I_{\text{SR}} = 10$ pA. Figure 4.2 shows the plots indicating any CRU open in the domain at ten selected times from 100 ms to 1,000 ms. Each dot indicates that the CRU at the spatial point is open (and does not represent the value of any quantity). We observe that a number of CRUs open randomly over time, and thus the probability mechanism of the model works. However, no systematic opening of CRUs develops. Figure 4.3 shows isosurface plots of the calcium concentration $u^{(0)}$ throughout the cell with a critical isolevel of 65 μM indicated by each surface. It is clear that the concentration around open CRUs does increase. It then diffuses and thus the concentration in the area falls below the critical isolevel again. We notice that the level of calcium concentration does not rise high enough anywhere to cause more CRUs in the area to open.

Figures 4.4 and 4.5 show results for the case study with $I_{\text{SR}} = 20$ pA. We see in Figure 4.4 that at time $t = 100$ ms a number of CRUs are open without any discernible pattern. But by $t = 200$ ms, two waves of CRUs opening have self-organized in this case. At some time between 100 and 200 ms, the concentration near the left end of the domain as well as just to the right of center at the top has reached higher levels that caused several neighboring CRUs to open at the same time. In turn, more neighboring CRUs of those opened and by $t = 200$ ms we have one wave traveling

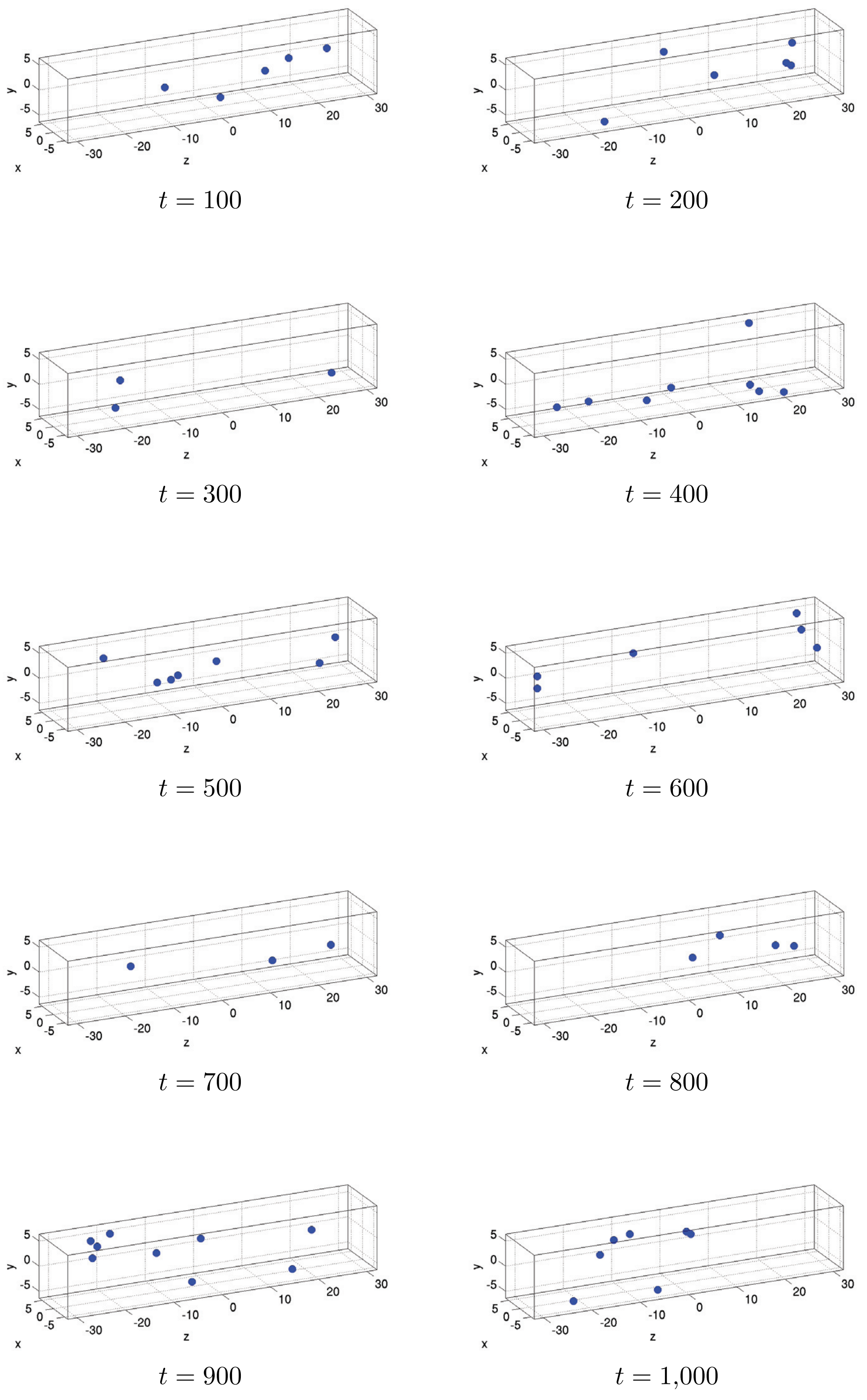


FIG. 4.2. *Open calcium release units throughout the cell with $I_{SR} = 10$ pA.*

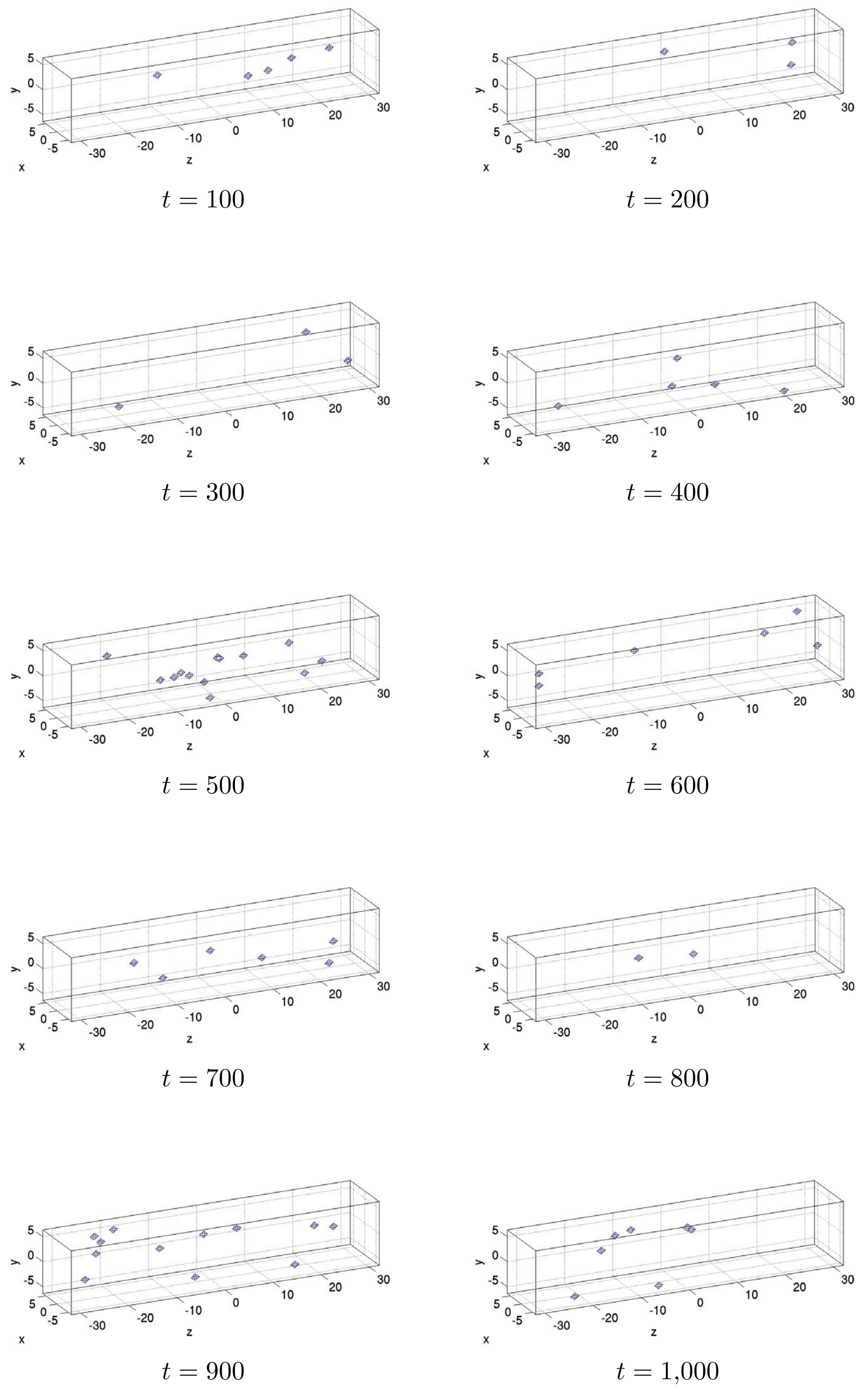


FIG. 4.3. *Isosurface plots of the calcium concentration throughout the cell with $I_{SR} = 10$ pA.*

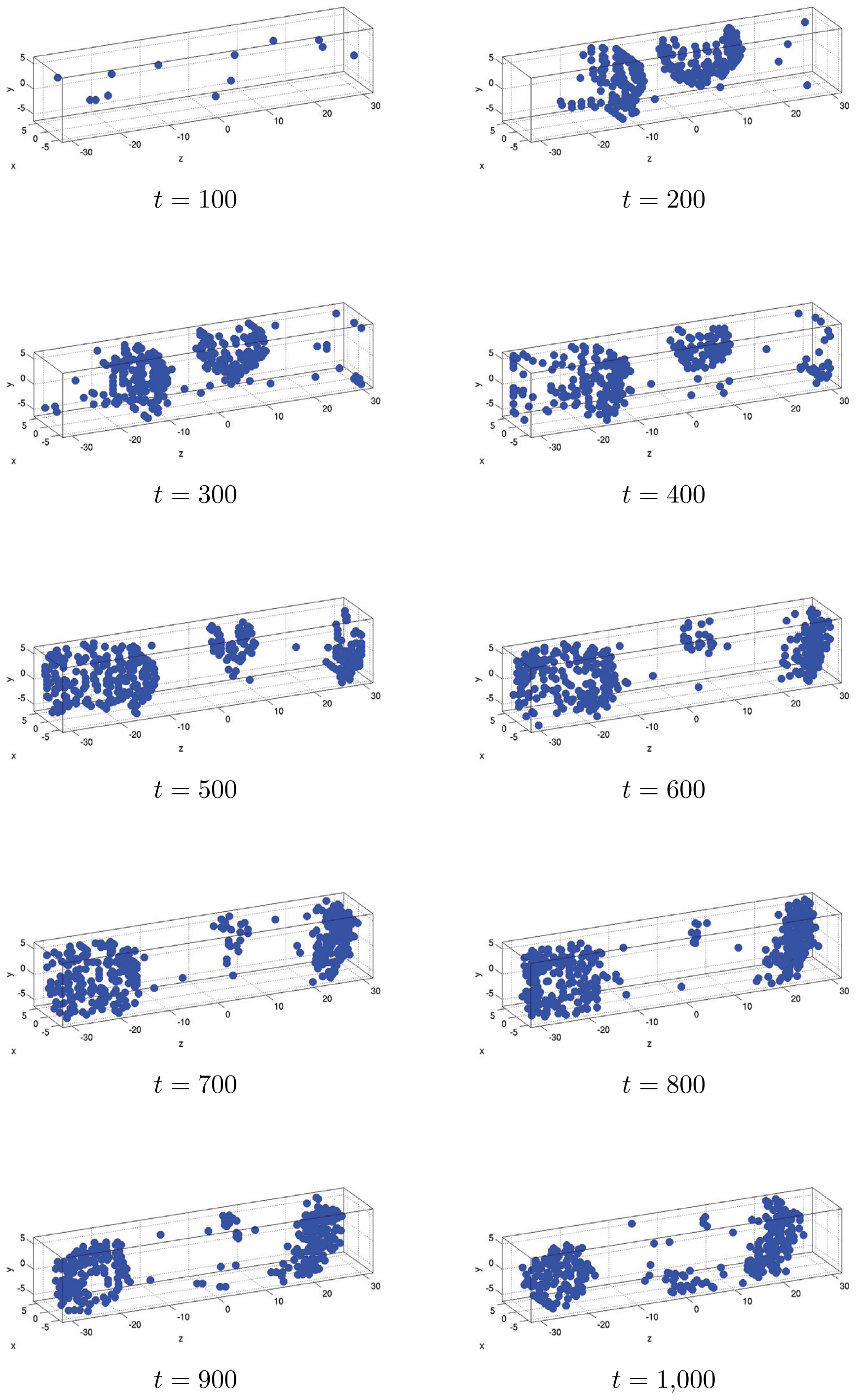


FIG. 4.4. *Open calcium release units throughout the cell with* $I_{SR} = 20$ *pA.*

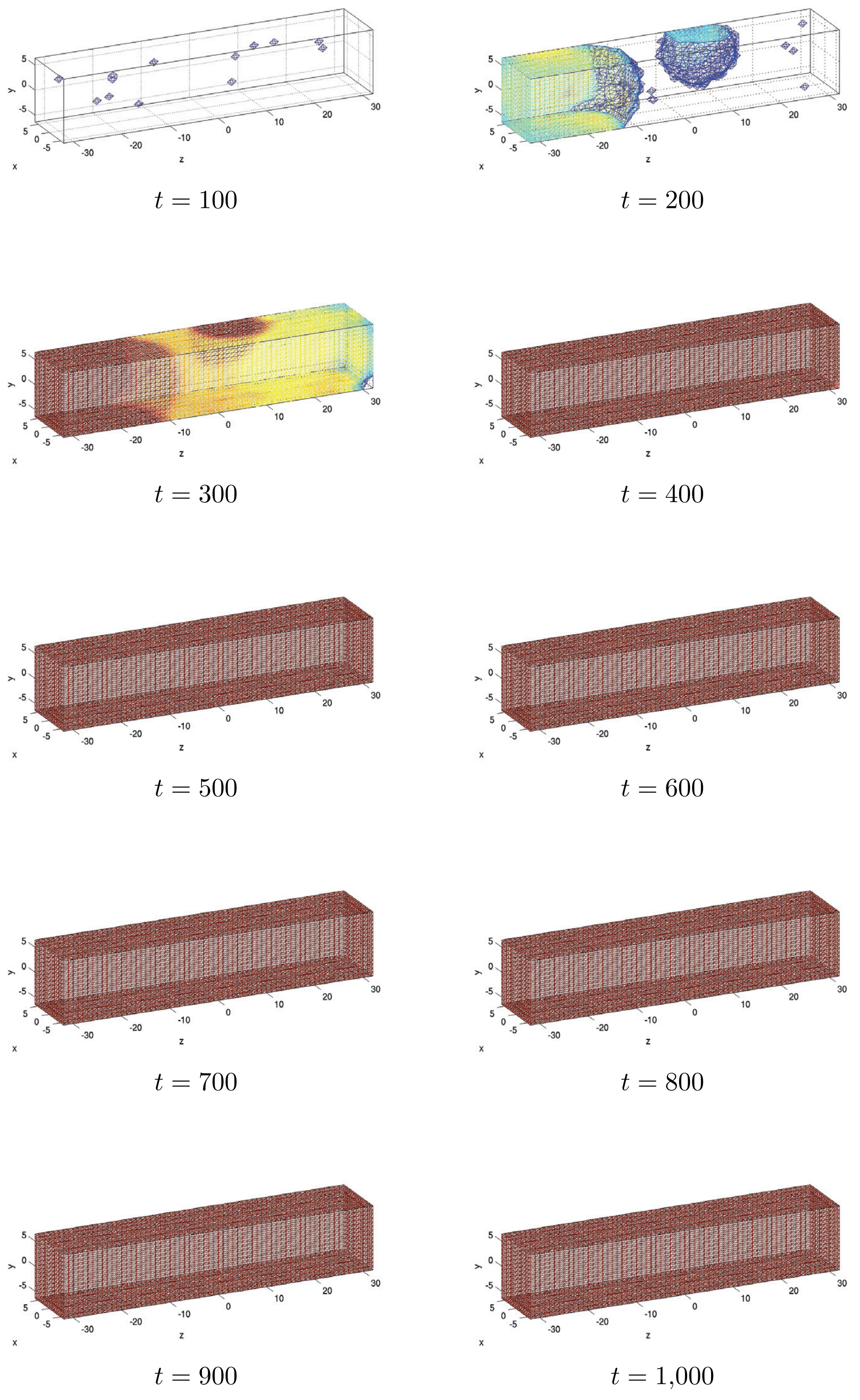


FIG. 4.5. *Isosurface plots of the calcium concentration throughout the cell with* $I_{SR} = 20$ *pA.*

left to right with its front at about $z = -12$ and a second wave expanding from a center at about $(x, y, z) = (5, 5, 5)$. The formation of the waves is clearly visible in the movies available at the website mentioned at the end of section 1. After the CRUs, where the two waves started, have been closed for a time period of $t_{\text{closed}} = 100$ ms, they open again and restart a wave similar to each first one. This process is repeated several times throughout the simulation. The reason why several of the snapshots look remarkably similar is that the time between them of 100 ms approximates the period of time between wave initiations of $t_{\text{open}} + t_{\text{closed}} = 105$ ms. These results demonstrate that the model at this higher value of current $I_{\text{SR}} = 20$ pA promotes the self-organization of waves, as intended by the model. Notice that it took some time of between 100 and 200 ms for the first wave to form, which demonstrates the importance of being able to perform long-time simulations for this application problem. Even more so, the effect of waves traveling through the cell several times could only be seen by simulating to a large final time such as 1,000 ms.

Figure 4.5 shows isosurface plots of the calcium concentration $u^{(0)}$ throughout the cell with a critical isolevel of 65 μM. We see at $t = 100$ ms that the concentration has crossed this isolevel only around a few CRUs that happen to be open. By $t = 200$ ms, however, in the wake of both waves, we see significantly increased levels of calcium, which by $t = 400$ ms have reached levels above the critical isolevel throughout the domain. Clearly, the model exhibits the feedback mechanism of open CRUs releasing calcium and the diffused calcium in turn promoting the initiation of a new wave after the time period of closure t_{closed} has passed. In fact, we note from our log files that about 300 CRUs are open at any given moment in time for all times $t \geq 200$ ms. Plots of the maximum concentration vs. time for each species (not shown here) demonstrate that the calcium concentration grows without bound, and the other free species are practically completely consumed. It appears that this behavior is not physical, because the calcium concentration should not grow without bound, in a cell. This indicates that some effect is missing from the model or coefficient values being not appropriate. Notice here that this fact is simply not apparent until simulations to times significantly larger than $t_{\text{closed}} = 100$ ms are performed, because we have to wait for waves to travel through the cell several times to see this effect. This demonstrates the key advantage of our simulation platform for the validation of models for this application problem.

5. Conclusions.

We consider a model for calcium waves in a heart cell proposed in [11, 12, 13]. To validate this model and recreate the conditions of an experiment, it is necessary to be able to perform simulations up to large final times, to allow for waves to regenerate several times, and on a domain that encompasses the entire cell. This requires a high resolution mesh to adequately resolve the given lattice of calcium release units throughout the cell. Section 3 presents all choices for the development of a simulation platform for this model from the ground up. The keys to our ability to perform the desired long-time simulations on a high resolution mesh are the use of a variable-order, variable step-size ODE solver and the matrix-free implementation of all linear solves.

The code is applied to the application problem in section 4.4. The results show that the model successfully allows for the self-organization of a wave at a random location in the cell, without any artificial triggering of calcium release. To see this result, it was already necessary to simulate up to final times over 100 ms. But to see waves regenerate and travel through the cell several times, we need to be able to reach final times of at least 1,000 ms. This is possible for our special-purpose code, and the results indicate that the model, in the form stated in [11, 12, 13] and with the model parameters available from these references, may eventually accumulate more calcium in the cell than is physically reasonable. This observation can only be made because our code can compute to sufficiently large final times.

We note that our application studies follow careful convergence studies in sections 4.2 and 4.3 for a scalar linear test problem with smooth and nonsmooth source term, respectively, designed to evaluate the accuracy and reliability of our method and its implementation as thoroughly as possible. The results also demonstrate that the method converges, and results on a coarser uniform mesh give reliable answers for the physical effect under consideration. Notice that a convergence study for the full application problem including the probabilistic term would be problematic, because its behavior is influenced by the random number generator and also by the calcium concentration, which is not exactly the same at different resolutions. Finally, section 4.1 demonstrated the effectiveness of using parallel computing to solve this large problem and the scalability of our implementation. In summary, our simulation platform is demonstrated to be able to solve the given model reliably and within a reasonable time frame on actually available computational resources; thus it is perfectly suited to investigate which parts of the model need improvements so that its results better match experimental results.

In section 3, we also include a number of comparisons to the simulator used in [12]. One difference between [12] and this work is actually the use of a general-purpose code vs. the development of a special-purpose one. This distinction is independent of the mesh used, because, as long as the mesh is regular in some way and fixed in time, we can still compute all matrices analytically and implement all linear solves in matrix-free form, which is one key to the efficiency of our code. But we believe this is not the direction of most promise to pursue, because this still does not take full advantage of the knowledge about the application: A fine mesh around the location of a calcium release unit is needed only if that release unit is open, not when it is closed, as is the case far most of the time (compare $t_{\text{open}} = 5$ ms with $t_{\text{closed}} = 100$ ms). So, the most significant advance in efficiency could be achieved by an automatic mesh refinement and coarsening strategy that uses a finer mesh only around those release units that are open. In this context, the current simulation platform can serve to produce reference solutions for any more sophisticated code.

REFERENCES

[1] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc users manual*, Technical report ANL–95/11 — Revision 2.1.5, Argonne National Laboratory, 2004.

[2] Y. Chen-Izu, S. L. McCulle, C. W. Ward, C. Soeller, B. M. Allen, C. Rabang, M. B. Cannell, C. W. Balke, and L. T. Izu, *Three-dimensional distribution of ryanodine receptor clusters in cardiac myocytes*, Biophys. J., 91 (2006), pp. 1–13.

[3] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[4] M. K. Gobbert, *Configuration and performance of a Beowulf cluster for large-scale scientific simulations*, Comput. Sci. Eng., 7 (2005), pp. 14–26.

[5] M. K. Gobbert, *A technique for the quantitative assessment of the solution quality on particular finite elements in COMSOL Multiphysics*, in Proceedings of the COMSOL Conference 2007, V. Dravid, ed., Boston, MA, 2007, pp. 267–272.

[6] M. K. Gobbert, M. Kružík, and T. I. Seidman, *Numerical approximation of a heat equation with measure-valued data*, in preparation.

[7] M. K. Gobbert, M. Muscedere, T. I. Seidman, and R. J. Spiteri, *A non-negativity preserving Newton method for high-order implicit time stepping*, submitted.

[8] A. L. Hanhart, M. K. Gobbert, and L. T. Izu, *A memory-efficient finite element method for systems of reaction-diffusion equations with non-smooth forcing*, J. Comput. Appl. Math., 169 (2004), pp. 431–458.

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

[10] W. Hundsdorfer and J. Verwer, *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*, Springer Series in Computational Mathematics 33, Springer-Verlag, Berlin, 2003.

[11] L. T. Izu, J. R. H. Mauban, C. W. Balke, and W. G. Wier, *Large currents generate cardiac Ca^{2+} sparks*, Biophys. J., 80 (2001), pp. 88–102.

[12] L. T. Izu, S. A. Means, J. N. Shadid, Y. Chen-Izu, and C. W. Balke, *Interplay of ryanodine receptor distribution and calcium dynamics*, Biophys. J., 91 (2006), pp. 95–112.

[13] L. T. Izu, W. G. Wier, and C. W. Balke, *Evolution of cardiac calcium waves from stochastic calcium sparks*, Biophys. J., 80 (2001), pp. 103–120.

[14] D. A. Knoll and D. E. Keyes, *Jacobian-free Newton-Krylov methods: A survey of approaches and applications*, J. Comput. Phys., 193 (2004), pp. 357–397.

[15] M. Matsumoto and T. Nishimura, *Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulation, 8 (1998), pp. 3–30.

[16] A. Quarteroni and A. Valli, *Numerical Approximation of Partial Differential Equations*, Springer Series in Computational Mathematics 23, Springer-Verlag, Berlin, 1994.

[17] D. L. Ropp, J. N. Shadid, and C. C. Ober, *Studies of the accuracy of time integration methods for reaction-diffusion equations*, J. Comput. Phys., 194 (2004), pp. 544–574.

[18] S. Rüdiger, J. W. Shuai, W. Huisinga, C. Nagaiah, G. Warnecke, I. Parker, and M. Falcke, *Hybrid stochastic and deterministic simulations of calcium blips*, Biophys. J., 93 (2007), pp. 1847–1857.

[19] A. Sandu, *Positive numerical integration methods for chemical kinetic systems*, J. Comput. Phys., 170 (2001), pp. 589–602.

[20] J. Shadid, A. Salinger, R. Schmidt, T. Smith, S. Hutchinson, G. Hennigan, K. Devine, and H. Moffat, *MPSalsa: A finite element computer program for reacting flow problems; Part* 1*—Theoretical development*, Technical report SAND98–2864, Sandia National Laboratories, 1998.

[21] L. F. Shampine and M. W. Reichelt, *The MATLAB ODE suite*, SIAM J. Sci. Comput., 18 (1997), pp. 1–22.

[22] L. F. Shampine, S. Thompson, J. A. Kierzenka, and G. D. Byrne, *Non-negative solutions of ODEs*, Appl. Math. Comput., 170 (2005), pp. 556–569.

[23] V. Thomée, *Galerkin Finite Element Methods for Parabolic Problems*, Springer Series in Computational Mathematics 25, 2nd ed., Springer-Verlag, Berlin, 2006.