CHASTE: incorporating a novel multi-scale spatial and temporal algorithm into a large-scale open source library

BY MIGUEL O. BERNABEU 1, RAFEL BORDAS 1, PRAS PATHMANATHAN 1, JOE PITT-FRANCIS 1, JONATHAN COOPER 1, ALAN GARNY 2, DAVID J. GAVAGHAN 1, BLANCA RODRIGUEZ 1, JAMES A. SOUTHERN 1,3 AND JONATHAN P. WHITELEY 1,*

1Oxford University Computing Laboratory, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK
2Department of Physiology, Anatomy and Genetics, University of Oxford, Sherrington Building, Parks Road, Oxford OX1 3PT, UK
3Fujitsu Laboratories of Europe Ltd, Hayes, Middlesex UB4 8FE, UK

Recent work has described the software engineering and computational infrastructure that has been set up as part of the Cancer, Heart and Soft Tissue Environment (CHASTE) project. CHASTE is an open source software package that currently has heart and cancer modelling functionality. This software has been written using a programming paradigm imported from the commercial sector and has resulted in a code that has been subject to a far more rigorous testing procedure than that is usual in this field. In this paper, we explain how new functionality may be incorporated into CHASTE. Whiteley has developed a numerical algorithm for solving the bidomain equations that uses the multi-scale (MS) nature of the physiology modelled to enhance computational efficiency. Using a simple geometry in two dimensions and a purpose-built code, this algorithm was reported to give an increase in computational efficiency of more than two orders of magnitude. In this paper, we begin by reviewing numerical methods currently in use for solving the bidomain equations, explaining how these methods may be developed to use the MS algorithm discussed above. We then demonstrate the use of this algorithm within the CHASTE framework for solving the monodomain and bidomain equations in a three-dimensional realistic heart geometry. Finally, we discuss how CHASTE may be developed to include new physiological functionality—such as modelling a beating heart and fluid flow in the heart—and how new algorithms aimed at increasing the efficiency of the code may be incorporated.

Keywords: cardiac modelling; scientific computing; software engineering; numerical techniques
1. Introduction

Computational modelling is widely accepted to be a useful tool in the life sciences. One of its most mature areas is computational modelling of cardiac electrophysiology, with work in this field dating back to the first model of a cardiac action potential developed in 1962 (Noble 1962). This study laid the foundations for future computational studies that have provided significant insight into cardiac electrophysiological behaviour in both health and disease. Over the last four decades, mathematical models of cellular cardiac electrophysiology have been developed for different cell types and species, including human atrial and ventricular action potential models. Many such models are archived in the CellML repository (http://www.cellml.org/models/).

More recently, advancements in computational power have also allowed the simulation of cardiac electrophysiology at the whole organ level. Complex multi-scale (MS) electrophysiological models of the ventricles and the atria have been developed based on anatomical data obtained from different imaging modalities (Trayanova et al. 2002; Fenton et al. 2005; Tranquillo et al. 2005; Potse et al. 2006; ten Tusscher et al. 2007; Plotkowiak et al. 2008). Computer simulations using those models have enabled investigation into the mechanisms of cardiac arrhythmias, and predictions to be made on their therapies and diagnosis, including, among others, ischaemia-induced alterations in cardiac electrophysiology (Rodríguez et al. 2006; Potse et al. 2007), ventricular fibrillation in the human heart (ten Tusscher et al. 2007), defibrillation (Rodríguez et al. 2005; Vigmond & Clements 2007) and atrial fibrillation (Jacquemet et al. 2005; Seemann et al. 2006).

The studies listed above all model tissue-level electrophysiology. This is usually described mathematically by the bidomain or monodomain equations (e.g. Keener & Sneyd 1998). The numerical solution of the bidomain equations on a realistic three-dimensional computational domain is widely accepted to pose a significant computational challenge. Potse et al. (2006) report that, using the processing speed available in 2006, 2 days on a 32 processor distributed memory machine are required to simulate a single human heartbeat using the bidomain equations. It should be noted that this work used a model of the electrophysiology described by Bernus et al. (2002); this model is a simplification of the more complex electrophysiological models and requires less computational effort. Many research groups are developing cardiac electrophysiological models that include more details of the underlying physiology. These more detailed models will require more computational effort and, therefore, more computational time.

To fully use the predictive power of the complex models that are being developed, software packages should be developed, which are based on numerical algorithms that are both reliable and efficient. The MS nature of cardiac electrophysiology results in the physical processes being described by stiff systems of differential equations. The use of fully explicit time integration numerical methods is clearly an attractive option to anyone writing a cardiac simulation package, as these techniques do not require the solution of any nonlinear systems. However, the performance of such explicit numerical methods is notoriously prone to degradation by stability issues when these methods are applied to stiff systems of differential equations (Iserles 1996). As a consequence, the time step used by these solvers is usually driven by stability issues and not by
the time scales of the physiological processes that are being modelled. An alternative choice is to use implicit time integration numerical methods. These methods are, however, more difficult to incorporate into software as they require implementing methods for solving large systems of non-linear equations. This presents a difficult decision for anyone writing cardiac simulation software: the most appropriate numerical algorithm requires writing—and maintaining—a code that describes complex mathematical algorithms. Even if a complex numerical algorithm is successfully implemented, a future developer may struggle to maintain the integrity of the software due to a lack of understanding of the algorithm implemented. Less complex code can be written, but at the cost of using algorithms whose performance is significantly lower than is possible.

A further complication that has recently arisen for writers of cardiac simulation software is the introduction into cardiac models of additional features such as tissue deformation in response to the electrophysiology and the resulting flow of fluid (e.g. Watanabe et al. 2004). The systems of partial differential equations (PDEs) that model these coupled multi-physics problems are generally of mixed hyperbolic–elliptic–parabolic type and include constraints such as incompressibility of tissue. Simple numerical methods for discretizing these equations are often unconditionally unstable—i.e. they will never converge no matter how small the time step. Under these conditions, simple mathematical algorithms are not appropriate for computing the numerical solutions of these equations, and more complex numerical algorithms should be used instead. This has been demonstrated when a simple model of tissue deformation has been coupled with electrophysiology (Whiteley et al. 2007; Niederer & Smith 2008). Software written for these simulations therefore requires writing code for complex algorithms. As described above, such a code is difficult to write and maintain.

The two previous paragraphs have highlighted the need for reliable software packages for cardiac simulations. This, in turn, requires reliable software engineering methods for writing this software. An attempt to initiate software engineering techniques borrowed from the commercial sector in this field has recently been made through the Cancer, Heart And Soft Tissue Environment (CHASTE) software package currently being developed within the Computational Biology Group at the University of Oxford (Pitt-Francis et al. 2008). This package has been developed using an agile programming technique (e.g. Beck & Andres 2004). One key feature of this software is the large number of focused tests included; each of these tests covers a small portion of code (typically 5–10 lines), and each line of the code is covered by at least one test. When the code is modified, all tests are run and any failing test identifies very precisely the location of bugs in the software. This allows software to be developed with the minimum of degradation of earlier functionality. A second key feature is that a decision has been made to distribute the code under an open source software licence so that it can be downloaded, used and modified by other research groups. Any worker who suspects erroneous published results based on CHASTE is free to download the package himself/herself and search for the error and—if one is found—to alert the authors to allow the code and the associated research to be corrected.

In this paper, we will describe how CHASTE has been developed to combine this rigorous software engineering approach with a recently published numerical algorithm (Whiteley 2008). In §2, we describe numerical algorithms that have been used by other workers, justifying the choice of our numerical algorithm.
In §3, we briefly summarize the software development process and tools that are used by CHASTE, explaining why this results in reliable software that may form the basis for the next generation of cardiac modelling software. The accuracy and efficiency of the algorithm when implemented in this software is verified in §4. Finally, in §5, we discuss future challenges for computational biologists in this area, and discuss how we envisage developing the CHASTE package to meet these challenges.

2. An efficient algorithm for solving the bidomain equations

In this section, we will begin by writing down the bidomain equations. We will then briefly review numerical methods that have been used to compute an approximate solution of these equations, explaining why we have chosen to implement one particular algorithm.

(a) The bidomain equations

The bidomain equations are given by (e.g. Henriquez 1993; Keener & Sneyd 1998)

\[ \chi \left( C_m \frac{\partial V_m}{\partial t} + I_{\text{ion}}(u, V_m) \right) - \nabla \cdot (\sigma_i \nabla (V_m + \phi_e)) = 0, \]  
(2.1)

\[ \nabla \cdot ((\sigma_i + \sigma_e) \nabla \phi_e + \sigma_i \nabla V_m) = 0, \]  
(2.2)

and

\[ \frac{\partial u}{\partial t} = f(u, V_m), \]  
(2.3)

where \( V_m(x, t) \) is the transmembrane potential; \( \phi_e(x, t) \) is the extracellular potential; \( u(x, t) \) is a vector containing various gating variables and chemical concentrations; \( \sigma_i \) is the intracellular conductivity tensor; \( \sigma_e \) is the extracellular conductivity tensor; \( \chi \) is the surface to volume ratio; \( C_m \) is the membrane capacitance per unit area; and \( I_{\text{ion}} \) is the transmembrane ionic current. The precise forms of \( I_{\text{ion}} \) and \( f \) are prescribed by an electrophysiological cell model comprising a system of nonlinear ordinary differential equations (ODEs) that depend on various chemical concentrations, gating variables and the transmembrane potential.

Both (2.1) and (2.2) contain spatial derivatives and require boundary conditions. These are given by

\[ \mathbf{n} \cdot (\sigma_i \nabla (V_m + \phi_e)) = I_{s_i}, \]  
(2.4)

and

\[ \mathbf{n} \cdot (\sigma_e \nabla \phi_e) = I_{s_e}, \]  
(2.5)

where \( \mathbf{n} \) is the outward pointing unit normal vector to the tissue; \( I_{s_i} \) is the intracellular stimulus current applied across the boundary; and \( I_{s_e} \) is the extracellular stimulus current applied across the boundary. The system is then solved by specifying initial conditions for \( V_m \) and \( u \). As \( \phi_e \) appears only in these equations through spatial derivatives, this quantity is defined only up to an additive constant. When a bath is modelled, it is treated as an extension of the interstitial fluid (Vigmond et al. 2008).
Numerical methods for solving the bidomain equations

We now briefly describe some of the numerical methods that have been used to solve the bidomain equations, highlighting the strengths and weaknesses of each approach.

(i) Explicit numerical schemes

As discussed earlier, explicit numerical schemes are the easiest to implement. Whether a finite-difference method or finite-element method is used to discretize the PDEs (2.1) and (2.2) in space, these methods may be written (Plank et al. 2007; Vigmond et al. 2008) as

\[
V_{k+1} = V_k + \Delta t A_i \phi_k,
\]

(2.6)

\[
V_{k+1} = V_k - \frac{\Delta t}{C_m} I_{ion}(u_k, V_{k+1}),
\]

(2.7)

\[
(A_i + A_e)\phi_{k+1} = -A_i V_{k+1}
\]

(2.8)

and

\[
u_{k+1} = u_k + \Delta t f(u_k, V_{k+1}),
\]

(2.9)

where \(\Delta t\) is the time step used when discretizing in time; \(V_k, \phi_k\) and \(u_k\) are the spatial discretizations of \(V_m, \phi_e\) and \(u\) in space at time \(k\Delta t\), respectively; \(V_{k+1}\) is an intermediate quantity computed when solving equation (2.1); and \(A_i\) and \(A_e\) are the spatial discretizations of the operators

\[
\frac{1}{\chi C_m} \nabla \cdot (\sigma_i \nabla) \quad \text{and} \quad \frac{1}{\chi C_m} \nabla \cdot (\sigma_e \nabla).
\]

We refer to this scheme as an ‘explicit numerical scheme’ as the time derivatives are all modelled explicitly—although we note that computation of \(\phi_{k+1}\) in equation (2.8) requires the solution of a linear system, as there is no time derivative in equation (2.2).

To date, the explicit method described by equations (2.6)–(2.9) is the most common technique used when solving the bidomain or monodomain equations on a realistic three-dimensional geometry (Trudel et al. 2004; Potse et al. 2006; Plank et al. 2007; Vigmond et al. 2008). As we have explained in §1, these techniques have the advantage of being relatively easy to implement. The use of explicit approximations to time derivatives does, however, require that the time step used is chosen to satisfy stability conditions—as these equations are stiff, it is likely that this time step is much shorter than that required to capture the physiological processes that are being modelled. The time step used is usually of the order of 0.01 ms (Quan et al. 1998; Qu & Garfinkel 1999; Skouibine et al. 2000; Vigmond et al. 2002; Weber dos Santos et al. 2004; Austin et al. 2006).

(ii) Semi-implicit numerical schemes

The stability restriction on time steps that is a feature of explicit numerical schemes may be avoided by the use of an implicit numerical scheme. A fully implicit numerical scheme, however, prevents us from uncoupling any of the bidomain equations, equations (2.1)–(2.3). Furthermore, the resulting system of discretized equations is nonlinear, and so we must both compute and store a large
Jacobian matrix when solving this large system. This is not practical for a computational mesh of the size seen when solving the bidomain equations on a realistic three-dimensional geometry, and so fully implicit techniques are seldom used for problems of this size.

In order to avoid the drawbacks associated with fully explicit or fully implicit techniques, we propose the use of a semi-implicit technique (Keener & Bogar 1998; Whiteley 2006, 2008), where the linear diffusion terms in equations (2.1) and (2.2) are modelled implicitly and the reaction term explicitly. Using the notation first used in equations (2.6)–(2.9), the PDEs (2.1) and (2.2) may be discretized as

\[
\begin{pmatrix}
\Delta t A_i - I & \Delta t A_i \\
A_i & A_i + A_c
\end{pmatrix}
\begin{pmatrix}
V^{k+1} \\
\phi^{k+1}
\end{pmatrix}
= 
\begin{pmatrix}
-V^k + \frac{\Delta t}{C_m} I_{ion}(u^k, V^k) \\
0
\end{pmatrix}.
\] (2.10)

The ODEs, equation (2.3), may then be solved using any appropriate technique (see Iserles (1996) for a description of these techniques). Although the discretization described in equation (2.10) is not unconditionally stable, in practice, the time step used can be chosen to capture the physiological processes rather than with stability in mind. It may seem counter-intuitive to solve equation (2.10) as a coupled system for \( V^{k+1} \) and \( \phi^{k+1} \) instead of uncoupling these solvers. However, it has been reported that solving the coupled system is more computationally efficient than uncoupling this linear system (Pennacchio & Simoncini 2002; Sundnes et al. 2005), and we therefore solve the coupled system.

The semi-implicit technique described here allows a longer time step to be used without any significant loss of accuracy. When using this technique, or a related technique, the time step used is usually approximately 0.1 ms, i.e. an order of magnitude larger than that used in conjunction with the explicit method (Sundnes et al. 2001, 2005, 2006; Trew et al. 2005; Whiteley 2006, 2007). We note that the only additional complexity introduced when using a semi-implicit technique instead of an explicit technique is that a larger linear system must be solved on each time step.

(iii) Operator splitting techniques

The explicit numerical scheme (2.6)–(2.9) can be thought of as a very simple operator splitting scheme. Other authors have used more sophisticated operator splitting schemes (Qu & Garfinkel 1999; Sundnes et al. 2005; Hanslien et al. 2007). Using this approach, the spatial and temporal operators are uncoupled and solution techniques are used for each of the uncoupled operators separately. This reduces the complexity of the numerical methods required to solve the uncoupled systems while maintaining the highest possible order of convergence of the global solver. The main drawback of the operator splitting approach is the difficulty of ensuring numerical stability of the resulting scheme, although this has been achieved for the relatively simple Luo–Rudy I model (Hanslien et al. 2007).

Phil. Trans. R. Soc. A (2009)
Adaptive numerical schemes

There has recently been much interest in the numerical analysis community in adaptive numerical schemes (e.g. Eriksson et al. 1995; Houston & Süli 2001; Gavaghan et al. 2006). Rather than using a fixed computational mesh, these methods allow a different mesh to be used on each time step with a high density of nodes being used only when the solution varies rapidly. These methods have the advantage that computational effort is directed only towards regions of the computational domain where it is required. Adaptive numerical methods have not been particularly popular in the field of electrophysiological modelling. This is probably because: (i) significant computational effort is required to compute the mesh required for each time step, (ii) when the mesh is altered, the matrices $A_i$ and $A_e$ used in equations (2.6), (2.8) and (2.10) must be re-computed, and this itself requires a large volume of computations, (iii) each node introduced at refinement requires the generation of state from a cell model, and (iv) in modelling fibrillation, the refinement is required to change rapidly. There is, however, a limited selection of literature devoted to applying an adaptive numerical method to the monodomain or bidomain equations (e.g. Cherry et al. 2003; Pennacchio 2004; Colli Franzone et al. 2006; Whiteley 2007). None of these, however, have yet managed to demonstrate an increase in computational efficiency when applied to a realistic three-dimensional computational domain.

MS numerical schemes

A numerical method has recently been developed, which uses the MS nature of the problem (Whiteley 2008). This numerical algorithm uses the observation that only a very small number of quantities vary on a short time scale and a short length scale, and computes only these quantities at a high resolution. The key to implementing this algorithm is the use of two meshes, as shown in figure 1. A fine mesh—represented by thin lines—has nodal spacing $h_{\text{fast}}$, and rapidly varying variables are approximated using this mesh. Other variables are computed on a coarser mesh—represented by thicker lines. Linear interpolation is used to transfer the slower variables onto the fine mesh when required.

When using most electrophysiological models, the rapidly varying variables are those associated with the fast sodium current that causes the rapid action potential upstroke, specifically the transmembrane potential $V_m$, the extracellular potential $\phi_e$, the sodium $m$-gate and the sodium $h$-gate. Approximating these quantities on the fine mesh and all other quantities on the coarse mesh allows equation (2.10) to be written as

$$
\begin{pmatrix}
\Delta t A_i - I \\
A_i
\end{pmatrix}
\begin{pmatrix}
A_i \\
A_i + A_e
\end{pmatrix}
\begin{pmatrix}
V^{k+1} \\
\phi^{k+1}
\end{pmatrix}
= 
\begin{pmatrix}
I_1 + I_2 \\
0
\end{pmatrix},
$$

(2.11)

where $I_1$ is computed from variables approximated on the fine mesh and $I_2$ is dependent only on variables approximated on the coarse mesh. This system is solved with a time step $\Delta t_{\text{fast}}$, although, as the contributions to $I_2$ vary on a slower time scale, this vector is updated less frequently on a time scale $\Delta t_{\text{slow}}$. The terms included in $I_2$ are generally the more computationally expensive terms—updating this quantity on a slower time step should therefore give a significant computational saving.
Having computed $V_m$ and $\phi_v$ using equation (2.11), we now turn our attention to computing the solution of the ODEs given by equation (2.3). The equations governing the sodium $m$-gate and the sodium $h$-gate must be computed at every node of the fine mesh in order to calculate the contribution to $I_1$ in equation (2.11). All other ODEs need to be solved only at the nodes of the coarse mesh. As there are nodes fewer in orders of magnitude on the coarse mesh compared with the fine mesh, this also gives a substantial computational saving.

This algorithm has been implemented in a simple two-dimensional geometry and, using the electrophysiological cell model described by Noble et al. (1998), an increase in computational efficiency of approximately two orders of magnitude was achieved (Whiteley 2008). Later in this paper, we will investigate how this algorithm performs when applied to a realistic three-dimensional geometry.

(vi) Other methods for the numerical solution of the bidomain equations

In addition to the techniques described earlier in this section, other numerical analysis tools have been applied to the numerical solution of the bidomain equations. Skouibine et al. (2000) used a predictor–corrector scheme: an explicit numerical method was used as a prediction of the solution at the next time level. This prediction was used as an initial guess to the solution of the nonlinear system that arises from the implicit discretization of the bidomain equations. Domain decomposition was used by Quan et al. (1998); this allowed different time steps to be used in different regions of the computational domain.

3. Writing reliable software

Writing reliable software has been a focus of the commercial sector for many years, resulting in a plethora of tools and techniques that may be used to aid software development. Some of these techniques have recently been imported into the computational biology domain through the CHASTE project (Pitt-Francis et al. 2008). In this section, we begin by briefly reviewing the software...
engineering approach and computational infrastructure that has been used by CHASTE in order to generate high-quality software. We then describe various tools included in CHASTE for the purpose of (i) automating code generation from descriptions of electrophysiological models stored in the CellML format and (ii) automating code optimization.

(a) The software engineering approach adopted by CHASTE

In this section, we briefly describe the software engineering approach followed when developing CHASTE (for more details, see Pitt-Francis et al. (2008)). CHASTE is written in C++ using an agile programming technique known as ‘eXtreme programming’ (Beck & Andres 2004). Although the methodology used to develop CHASTE increases development time in the short term, it is anticipated that savings will be made in the medium to long term when we reap the benefits of the features offered through this software engineering approach. Specific features of this programming methodology, which are relevant to the scientific computing domain, are listed below.

(i) Test-driven development

Before any new incremental functionality is added, a code that tests this functionality is written. The code is then developed to incorporate the new functionality. After this, all tests including the new test are run. Should any test fail, then the code is corrected before any subsequent development takes place. As a consequence, many tests are written that cover very specific parts of the code, and all lines of the code are tested. Should a test fail, then it is relatively easy to identify the location of the problem.

(ii) Continuous integration

All developers integrate their separate strands of a code into a central repository on a regular basis. When this is done, all the tests described above are run to ensure no incompatibilities or regressions have been introduced.

(iii) Pair programming and collective code ownership

A code is usually developed by a team of two programmers working on a single computer. The composition of pairs is frequently rotated, allowing all developers to become acquainted with all aspects of the code, and all developers are familiar with the whole of the code. This facilitates collective code ownership, where no single person is responsible for any section of the code. This prevents development from stagnating owing to a developer leaving the team or spending less time on development work due to other responsibilities.

(iv) Refactoring

The large number of tests that covers small sections of the code allows any section of the code to be rewritten, to improve either the quality or performance of the code. After the code has been refactored, all tests are run to prevent degradation of quality due to the introduction of errors.
(b) The computational infrastructure employed by CHASTE

The latest version of the CHASTE software is stored on a central server equipped with the Subversion (http://subversion.tigris.org) version control system and backed up to central university resources. Any development is carried out by a pair of programmers ‘checking out’ the latest version—i.e. copying the latest version onto their local machine. When incremental development has been successfully carried out on the local machine, the code is ‘checked in’—any alterations made on the local machine are duplicated on the latest version of CHASTE stored on the central server. As more than one pair of developers may be working on the code at a given time, checking in code may introduce incompatibilities in the latest version stored on the central server. These incompatibilities are eradicated by ensuring all tests submitted pass before any more codes may be checked in.

Many established tools and libraries are used by the CHASTE project. Progress with individual programming tasks is tracked using the Trac project management tool (http://trac.edgewall.org). CHASTE developers use the Eclipse development environment (http://www.eclipse.org) for code development, CxxTest (http://cxxtest.sourceforge.net) for unit testing and the SCons software construction tool (http://www.scons.org) for build automation. See Pitt-Francis et al. (2008) for more details.

Rather than developing our own mathematical libraries from scratch, we use established, maintained libraries where possible. Large linear systems are solved using the PETSc parallel and sequential libraries (http://www.mcs.anl.gov/petsc). The Boost Ublas C++ libraries (http://www.boost.org/) are used for the smaller vectors and matrices that arise from assembling the contribution to the global system from a single element.

(c) Increasing the efficiency and reliability of software automatically

The complexity of the electrophysiology models that prescribe the $I_{\text{ion}}$ term in equation (2.1) and the size and complexity of the system of ODEs in equation (2.3) have grown substantially in models developed in recent years. In these recent models, $I_{\text{ion}}$ typically consists of several separate currents, and is dependent on tens of quantities, each modelled by a separate ODE. Furthermore, each of these ODEs includes several constant parameters. When one research group develops a model as complex as this, it is clearly difficult for another group to use this model, as coding this model without introducing errors in any of the constants is a near-impossible task. In order to facilitate sharing and reusing of these models, the CellML markup language was developed (Cuellar et al. 2003). CellML is an XML-based language that represents electrophysiological models (among others) as a structured document that may be read by both humans and computers. For details of the current state of CellML, see Garny et al. (2008).

Use of electrophysiological models stored in the CellML format should clearly be an essential feature of any electrophysiological simulation software. Although development of CellML functionality is independent of the CHASTE project, CHASTE is fortunate in that some CHASTE developers are also developers of a collection of relevant CellML tools. As a consequence, CHASTE is fully able to use CellML tools. We list below the CellML tools that we use in CHASTE.
(i) **Automatic code generation**

Tools exist to automatically generate C++ code from a CellML file representing a given model. Automatically generated code from the PyCML (https://chaste.ediamond.ox.ac.uk/cellml/) tool is used in CHASTE to compute the $I_{\text{ion}}$ term in equation (2.1) and the right-hand side of the ODEs in equation (2.3). This feature allows any electrophysiological model represented by a valid CellML file to be used in a CHASTE simulation.

(ii) **Lookup tables**

Many expressions that are included in the $I_{\text{ion}}$ term and the system of ODEs in equation (2.3) are functions of only the transmembrane potential $V_m$ and contain exponential terms that are expensive to compute. These expressions may be computed more efficiently by using lookup tables: the value of this expression is pre-computed at a number of points, and interpolation of these values is used to evaluate this expression during the course of the simulation. This technique has been reported to give a substantial increase in computational efficiency while introducing only negligible error (Victorri *et al.* 1985; Cooper *et al.* 2006).

(iii) **Partial evaluation**

Many terms that are generated through the automatic code generation described above contain expressions that may be pre-computed at compilation time to avoid repeated computation of an identical quantity. This may be achieved using partial evaluation. This technique gives a significant increase in computational efficiency, especially when combined with lookup tables (Cooper *et al.* 2006).

(iv) **Unit checking**

All of the variables and constants used in an electrophysiological cell model have associated units, for example a length may be quoted in the unit millimetres. It is crucial for an accurate simulation that compatible units are used for all quantities (we cannot add a length to an area, for instance), and that quantities are given in the same units at either side of an interface between components of the simulator. Fortunately, this may be automatically verified through the use of a CellML tool, and more advanced tools (Cooper & McKeever 2008) are being developed, allowing for automatic conversion between compatible units.

4. **Results**

A number of basic simulations were carried out in CHASTE, in order to assess the computational efficiency and accuracy of various approximations to the solution of the full cell ODE systems. The approximations were based on the semi-implicit discretization in equation (2.10), but used different numerical schemes to update the ODEs in the cell models. These numerical schemes range from an explicit solution with a time-step size determined by stability requirements to the linear interpolation scheme in the MS algorithm described...
in Whiteley (2008). These simulation experiments were designed to demonstrate the performance of the approximations when implemented in a generic high-performance computing code, and applied to more realistic geometries.

(a) Description of simulations

In two dimensions, the computational meshes were chosen to be the same as those in Whiteley (2008). The simulations were carried out on a square occupying the region \(0 < x, y < 20\) mm in an isotropic medium. The electrophysiological model used was that described in Noble et al. (1998). A spatial nodal spacing of 0.1 mm was used in each of the \(x\) and \(y\)-directions. In the MS algorithm, a node spacing of \(h_{\text{slow}} = 1.0\) mm was used for the coarse mesh (with \(h_{\text{fast}} = 0.1\) mm). This resulted in 40401 nodes on the fine mesh and 441 nodes on the coarse mesh. The nodes along the \(x=0\) edge were stimulated for the first 0.5 ms, generating an action potential that propagated across the square. The simulation was run for 0.35 s, allowing the entire domain to be excited and to return to its resting state. For each simulation, the transmembrane potential was recorded at all nodes for the duration of the simulation at intervals of 0.1 ms. The traces from the points \(x = 10\) mm, \(y = 10\) mm (midpoint node) and \(x = 20\) mm, \(y = 10\) mm (furthest edge from the stimulus) were analysed. The conduction velocity between the points was calculated. The action potential duration to 90 per cent repolarization (APD90) and maximum upstroke velocity were calculated at the midpoint node.

In three dimensions, a cube mesh was chosen that generalized the mesh used in two dimensions and had a similar number of nodes. The mesh contained the region \(0 < x, y, z < 3\) mm in an isotropic medium. In common with the two-dimensional simulations, \(h_{\text{fast}} = 0.1\) mm and \(h_{\text{slow}} = 1.0\) mm, leading to 29791 nodes on the fine mesh and 64 nodes on the coarse mesh. A stimulus was applied along the face \(x = 0\) mm and the transmembrane potential was analysed at the midpoint \((1.5, 1.5, 1.5)\) and on the \(x = 3\) mm face at the midpoint \((3, 1.5, 1.5)\).

On a uniform grid in three dimensions, with a large number of nodes in each spatial direction, the choice of \(h_{\text{fast}}\) and \(h_{\text{slow}}\) used in these simulations would lead to the ratio of the number of coarse nodes to fine nodes being approximately 1 : 1000. For the two-dimensional simulations, this ratio would be approximately 1 : 100. The ratio is closer to 1 : 500 for the uniform mesh used in these simulations due to the relatively small number of nodes in each direction. For our two-dimensional simulation that uses a larger number of nodes in each spatial direction, we see that we are closer to the ratio of 1 : 100 expected on a grid with a large number of nodes.

As a further indicator of how well these methods scale to realistic geometries, we also ran a bidomain simulation on a full ventricular mesh with 63885 nodes. This simulation may be seen on the CHASTE webpage (http://www.comlab.ox.ac.uk/chaste). The MS algorithm used a coarse mesh of 173 nodes, which was produced from the original ventricular geometry via a decimation algorithm. The decimation scheme allows for the removal of surface nodes if the local change in volume is below a threshold. This means that some of the nodes in the original fine mesh are outside the volume of the decimated coarse mesh. However, all nodes of the coarse mesh are guaranteed to be coincident with nodes on the fine mesh. The ratio of coarse to fine nodes (approx. 1 : 400) is higher than that used on the regular three-dimensional slab experiments, but was selected because

*Phil. Trans. R. Soc. A* (2009)
further decimation of the mesh was not possible without changing the topology of the ventricular cavities. Unfortunately, there are some complications with the MS algorithm on these meshes, since at the fine mesh nodes outside the volume of the coarse mesh interpolation becomes extrapolation or projection. If the values of slow gating variables and ionic concentrations are extrapolated, there is potential for them to become unphysiological (small and negative). We are still exploring a physiologically valid solution to this problem.

(i) The computational efficiency of the current code

It is worth noting that prior to these experiments, the developers had been profiling and optimizing the code. Considerable effort had been put into resolving common bottlenecks, particularly the assembly and the solution of the linear system from the diffusion PDE. Most of the steps in the solution method (ODE solution, mesh reading, output and assembly of the linear system) are linear in the number of nodes; an exception is the solution of the linear system itself, which usually runs in a time that is near quadratic in the number of nodes.

Since CHASTE is designed to be scalable to meshes with large numbers of nodes (approx. $3 \times 10^6$), work focused on reducing the amount of time spent in the linear solver. It has been documented elsewhere (e.g. Plank et al. 2007) that the choice of linear solver and preconditioner is critical in reducing the solution time of the bidomain equations. We have found empirically that a suitable selection is a (block) Jacobi preconditioner with a symmetric LQ variant (SYMMLQ) of the conjugate gradient method. The SYMMLQ solver converges sufficiently rapidly so that the tolerance of the solution may be varied by an order of magnitude with little change to the overall run time. We found that the amount of time spent solving the bidomain system was four or five times that of solving the monodomain system, since the bidomain linear system has double the number of unknowns.

The other large computational bottleneck is in the assembly of the linear system. Some considerable time has been spent on profiling this code. During the course of writing code for these experiments, we managed to optimize the assembly routines so that they took no more time than ODEs or linear solvers.

In all the results presented in this paper, the physiological parameters used are those tabulated in table 1. Conductivities $\sigma_i$ and $\sigma_e$ were isotropic (with no dominant fibre direction).

(ii) Types of simulation

Throughout the following set of results, we refer to the four numerical schemes as in table 2. Note that $\Delta t$ refers to the time step used both to update the cell model ODEs and to solve the bidomain or monodomain PDEs. See §3 for a brief

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_m$</td>
<td>$0.01 \mu F \ mm^{-2}$</td>
</tr>
<tr>
<td>$\chi$</td>
<td>$140 \ mm^{-1}$</td>
</tr>
<tr>
<td>$\sigma_i$</td>
<td>$0.175 \ mS \ mm^{-1}$</td>
</tr>
<tr>
<td>$\sigma_e$</td>
<td>$0.7 \ mS \ mm^{-1}$</td>
</tr>
</tbody>
</table>
description of the optimizations (LU) and Whiteley (2006) for an explanation of the backward Euler (BE) scheme.

(b) Results of simulations

(i) Accuracy of the simulations

Figures 2–5 present the action potential (i.e. the transmembrane potential through time) at the midpoint of the mesh over the course of each simulation. It is worth noting that the cell was excited earlier in three dimensions (2–3 ms) than in two dimensions (approx. 17–21 ms) simply because the midpoint of the node is 1.5 mm away from the stimulus, rather than 10 mm in the two-dimensional mesh.
Figure 3. Bidomain two dimensions: electrical activation at $x=10$ and $y=10$ mm. (FE, solid curve; LU, dashed curve; BE, short-dashed curve; MS, dotted curve. The curves for BE and LU are visually indistinguishable.)

Figure 4. Monodomain three dimensions: electrical activation at $x=1.5$, $y=1.5$ and $z=1.5$ mm. (FE, solid curve; LU, dashed curve; BE, short-dashed curve; MS, dotted curve. The curves for BE and LU are visually indistinguishable.)

Phil. Trans. R. Soc. A (2009)
As might be expected, the results of the simulations using the forward difference method with a small stable time step of $\Delta t=0.01$ ms (FE and LU) are indistinguishable. Activation happens at a slightly later time when the semi-implicit scheme is used with a longer time step (BE with $\Delta t=0.1$ ms). This result concurs with the results presented in Whiteley (2006). In the MS simulations, the onset of electrical activity happens before the equivalent straight semi-implicit BE run, although later than the FE and LU times. This result concurs with that presented in Whiteley (2008).

One can also see that in each trace in figures 2–5, although activation may happen at a slightly different time, the shape and duration of the action potentials are qualitatively the same. We now turn our attention to more quantitative measures of the physiological properties of the simulations. These outputs (calculated from the traces at the middle node and from the activation at a right-hand node) are presented in tables 3 and 4.

In table 3, we tabulate the APD90, conduction velocity and maximum upstroke velocity described in §4a for each of the two-dimensional simulation experiments. We see that the difference in the APD90 calculated during the monodomain experiments is small, with all values being within $\pm 0.6$ per cent of the mean. The difference between the conduction velocity figures is also small and all figures are within $\pm 4$ per cent of the mean. The maximum upstroke velocity figures show a greater degree of variation, but are still within

![Figure 5. Bidomain three dimensions: electrical activation at $x=1.5$, $y=1.5$ and $z=1.5$ mm. (FE, solid curve; LU, dashed curve; BE, short-dashed curve; MS, dotted curve. The curves for BE and LU are visually indistinguishable.)](image-url)
per cent of the mean. The differences in the properties for the bidomain experiments are similar to those for the monodomain. The APD90 calculated for each experiment is within \( \pm 0.6 \) per cent of the mean, the conduction velocity is within \( \pm 4 \) per cent of the mean and the maximum upstroke velocity is within \( \pm 8 \) per cent of the mean.

Conduction velocities in three dimensions are necessarily calculated to a lower accuracy because the probe nodes are closer together (1.5 mm rather than 10 mm) with the same nodal spacing \( h_{fast} = 0.1 \) mm.

(ii) Comparison of computational timings

Figures 6 and 7 show computation time results for the two-dimensional simulations described above. Monodomain and bidomain results are presented. We instrumented CHASTE to record wall clock times for all potentially significant parts of the code to an accuracy of 1 ms. However, in all cases 90–99\% of the simulation time was taken by three main components: assembling the linear system (matrix and right-hand side); ODE solution; and linear system solution. We have dropped less significant parts of the code such as pre- and post-processing and data output from the charts. All times were recorded by running the code on a single core of a 64 bit Xeon 3 GHz server.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>APD90 (ms)</th>
<th>Conduction velocity (mm ms(^{-1}))</th>
<th>Max dV/dt (mV ms(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monodomain FE</td>
<td>229.0</td>
<td>0.592</td>
<td>192.6</td>
</tr>
<tr>
<td>Monodomain LU</td>
<td>229.0</td>
<td>0.592</td>
<td>192.6</td>
</tr>
<tr>
<td>Monodomain BE</td>
<td>228.8</td>
<td>0.556</td>
<td>172.8</td>
</tr>
<tr>
<td>Monodomain MS</td>
<td>230.7</td>
<td>0.568</td>
<td>172.6</td>
</tr>
<tr>
<td>Bidomain FE</td>
<td>229.0</td>
<td>0.532</td>
<td>192.2</td>
</tr>
<tr>
<td>Bidomain LU</td>
<td>229.0</td>
<td>0.532</td>
<td>192.2</td>
</tr>
<tr>
<td>Bidomain BE</td>
<td>228.7</td>
<td>0.5</td>
<td>169.6</td>
</tr>
<tr>
<td>Bidomain MS</td>
<td>230.7</td>
<td>0.508</td>
<td>170.4</td>
</tr>
</tbody>
</table>

Table 3. Results of the two-dimensional simulations.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>APD90 (ms)</th>
<th>Conduction velocity (mm ms(^{-1}))</th>
<th>Max dV/dt (mV ms(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monodomain FE</td>
<td>229.4</td>
<td>0.652</td>
<td>195.8</td>
</tr>
<tr>
<td>Monodomain LU</td>
<td>229.4</td>
<td>0.652</td>
<td>195.8</td>
</tr>
<tr>
<td>Monodomain BE</td>
<td>229.2</td>
<td>0.6</td>
<td>171.4</td>
</tr>
<tr>
<td>Monodomain MS</td>
<td>231.0</td>
<td>0.652</td>
<td>178.5</td>
</tr>
<tr>
<td>Bidomain FE</td>
<td>229.3</td>
<td>0.6</td>
<td>193.6</td>
</tr>
<tr>
<td>Bidomain LU</td>
<td>229.3</td>
<td>0.6</td>
<td>193.6</td>
</tr>
<tr>
<td>Bidomain BE</td>
<td>229.2</td>
<td>0.56</td>
<td>173.0</td>
</tr>
<tr>
<td>Bidomain MS</td>
<td>231.0</td>
<td>0.58</td>
<td>179.7</td>
</tr>
</tbody>
</table>

Table 4. Results of the three-dimensional simulations.

\[ PHIL. TRANS. R. SOC. A (2009) \]
Figure 6. Monodomain two dimensions: performance of the different numerical schemes presented. Total computation time and most time-consuming parts shown.

Figure 7. Bidomain two dimensions: performance of the different numerical schemes presented. Total computation time and most time-consuming parts shown.
The performance improvement in ODE solving introduced by the use of partial evaluation and lookup tables can be observed by comparing the solid bar in FE and LU. These optimizations made the ODE solution approximately 10 per cent faster, yielding a more modest overall improvement of approximately 6 per cent. Nevertheless, its use in a numerical scheme such as BE (not suffering from such a long assembling time) would lead to a higher overall improvement.

Switching from an explicit (FE) to an implicit (BE), let us choose a bigger $\Delta t$ without affecting stability. The assembly time is highly dependent on $\Delta t$, as can be seen by comparing the second bar on both schemes. This is because the right-hand side of the linear system must be reassembled at each time step. The speed-up factor of 10 matches the ratio between $\Delta t_{\text{FE}}$ and $\Delta t_{\text{BE}}$, showing the assembly process to be a linear function of $\Delta t$. The reduction in ODE solving time (solid bar) comes straightforwardly from the fewer updates of the cell state needed.

Compared with BE, our MS algorithm reduces by a factor of 7 the time spent in ODE solving (corresponding to the ratio of 10 between $\Delta t_{\text{slow}}$ and $\Delta t_{\text{fast}}$). Unfortunately, the complexity of other parts of the code grows due to the interpolation step needed. The overall improvement factor is approximately 3. The complete set of optimizations presented (from FE to MS) shows an order of magnitude speed-up. This speed-up was mainly owing to switching to a more stable implicit solution technique.

---

Figure 8. Monodomain three dimensions: performance of the different numerical schemes presented. Total computation time and most time-consuming parts shown.

The performance improvement in ODE solving introduced by the use of partial evaluation and lookup tables can be observed by comparing the solid bar in FE and LU. These optimizations made the ODE solution approximately 10 per cent faster, yielding a more modest overall improvement of approximately 6 per cent. Nevertheless, its use in a numerical scheme such as BE (not suffering from such a long assembling time) would lead to a higher overall improvement.

Switching from an explicit (FE) to an implicit (BE), let us choose a bigger $\Delta t$ without affecting stability. The assembly time is highly dependent on $\Delta t$, as can be seen by comparing the second bar on both schemes. This is because the right-hand side of the linear system must be reassembled at each time step. The speed-up factor of 10 matches the ratio between $\Delta t_{\text{FE}}$ and $\Delta t_{\text{BE}}$, showing the assembly process to be a linear function of $\Delta t$. The reduction in ODE solving time (solid bar) comes straightforwardly from the fewer updates of the cell state needed.

Compared with BE, our MS algorithm reduces by a factor of 7 the time spent in ODE solving (corresponding to the ratio of 10 between $\Delta t_{\text{slow}}$ and $\Delta t_{\text{fast}}$). Unfortunately, the complexity of other parts of the code grows due to the interpolation step needed. The overall improvement factor is approximately 3. The complete set of optimizations presented (from FE to MS) shows an order of magnitude speed-up. This speed-up was mainly owing to switching to a more stable implicit solution technique.
Figures 8 and 9 present computational timings for the three-dimensional simulations. Most of the trends presented for the two-dimensional simulations also apply in three dimensions. The savings made in the ODE solution method from FE, through LU and BE, to MS are roughly the same as those savings made in the two-dimensional simulations. There is again a speed-up by a factor of 10 between LU and BE, which corresponds to the difference in $\Delta t$.

It must be noted that in the three-dimensional simulation experiments, the linear solver takes up to 60 per cent of the overall time for bidomain LU (20% with the smaller monodomain system). This is due to the higher connectivity of the mesh in three dimensions, which leads to the main matrix structure being less sparse than that for the equivalent two-dimensional problem. Enforcing the use of implicit schemes to reduce the time step, and thus the number of PDE solution steps, is mandatory here.

The timing results from the test on the more realistic heart-based geometry are not presented here, but they show the same trend between the semi-implicit BE run and the MS algorithm. There was an eightfold speed up in the ODE solution, which is similar to that obtained in two dimensions.

5. Discussions and conclusions

We have described the implementation of a MS numerical algorithm into CHASTE, a software package we have written, which may be used for—among other things—tissue-level electrophysiological simulations. Although we did not
achieve the same increase in computational efficiency seen for a model problem in Whiteley (2008), we are not disheartened. CHASTE has a very general purpose, and the flexibility and reliability offered by this software exclude the use of handcrafted coding that may be used to optimize the performance of the software. Furthermore, using a realistic geometry requires that we use an unstructured mesh—ordering the nodes to generate a well-conditioned matrix is a much tougher task under these circumstances.

Automating the production of as much of the code as possible has many advantages. Assuming this code meets high software engineering standards, the object-oriented features of this code allows us to replace one numerical algorithm by another without compromising the integrity of other parts of the code. This has been used by the CHASTE development team during the implementation of the MS algorithm described here—it is now a simple task for a user of CHASTE to specify which of the solvers offered by CHASTE they wish to use. Of course, should a user wish to run multiple simulations using a fixed electrophysiological cell model, then it may be appropriate to perform some code optimizations by hand. This may be achieved by using the profiling tools that are available within CHASTE. These tools may be used to identify bottlenecks in the code and these bottlenecks may then be hand-coded to optimize performance.

It is hoped that the CHASTE package will be developed to allow a wider range of simulations, drawn from both heart modelling and more general computational biology, to be carried out. In this paper, we have focused on the development of CHASTE for cardio-electrophysiological modelling. A separate development strand has focused on cancer modelling, and one notable achievement of this work is an MS model of cancer growth, which links feedback between the multiple scales in both directions (van Leeuwen et al. in press). It should be made clear that the cancer strand has benefited from the development of functionality for heart modelling and vice versa. The computational infrastructure is common to both strands and there is also much overlap in the modelling functionality required, such as the solution of ODEs. The generality that characterizes development of CHASTE allows any functionality already included to be used by any other application.

It is envisaged that the generality described above will aid us in the development of CHASTE for new problems, such as a multi-physics model of the heart that includes models of electrophysiology, force generation within cardiac fibres, tissue deformation and fluid mechanics. As more modelling functionalities are generated for different applications, it is likely that there will be more scope for code reuse for novel applications. This will make developing code for complex multi-physics problems a much less daunting task.

Any interested reader may download CHASTE under an open source licence from http://www.comlab.ox.ac.uk/chaste.

Development work on CHASTE is supported by the Engineering and Physical Sciences Research Council (EPSRC) of the UK as part of a software for high-performance computing project (grant reference EP/F011628/1). R.B. is supported by an EPSRC-funded Life Sciences Interface Doctoral Training Centre studentship (grant reference EP/E501605/1). P.P. is supported by the EPSRC-funded OXMOS project (grant reference EP/D048400/1). J.C. is supported by an EU-funded Network of Excellence project (grant reference 223920). J.A.S. is supported by an EU-funded preDiCT project (grant reference 224381). A.G. is supported by a grant from the Biotechnology and Biological Sciences Research Council (BBSRC) of the UK (grant reference BB/E024955/1). B.R. is supported by a Medical Research Council (MRC) of the UK Career Development Award (grant reference G0700278).
References


Rodríguez, B., Li, L., Eason, J., Efimov, I. & Trayanova, N. 2005 Differences between left and right ventricular geometry affect cardiac vulnerability to electric shocks. *Circ. Res.* **97**, 168–175. (doi:10.1161/01.RES.0000174429.00987.17)


