Chaste: A case study of parallelisation of an open source finite-element solver with applications to computational cardiac electrophysiology simulation

Miguel O Bernabeu, James Southern, Nicholas Wilson, Peter Strazdins, Jonathan Cooper and Joe Pitt-Francis

International Journal of High Performance Computing Applications 2014 28: 13 originally published online 10 February 2013
DOI: 10.1177/1094342012474997

Article

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
2014, Vol. 28(1) 13–32

DOI: 10.1177/1094342012474997
hpc.sagepub.com
SAGE

Chaste: A case study of parallelisation of an open source finite-element solver with applications to computational cardiac electrophysiology simulation

Miguel O Bernabeu[1,2,], James Southern[3], Nicholas Wilson[3], Peter Strazdins[4], Jonathan Cooper[5] and Joe Pitt-Francis[5]**

Abstract

The simulation of cardiac electrophysiology is a mature field in computational physiology. Recent advances in medical imaging, high-performance computing and numerical methods mean that computational models of electrical propagation in human heart tissue are ripe for use in patient-specific simulation for diagnosis, for prognosis and for selection of treatment methods. However, in order to move in this direction, it is necessary to make efficient use of modern petascale computing resources.

This paper focuses on an existing open source simulation framework (Chaste) and documents work done to improve the parallel scaling on a small range of electrophysiology benchmark problems.

These benchmarks involve the numerical solution of the monodomain or bidomain equations via the finite-element method. At the beginning of this study the electrophysiology libraries within Chaste were already enabled to run in parallel and were able to solve for electrical propagation using the monodomain or bidomain equations, but parallel efficiency dropped rapidly when run on more than about 64 processors.

Throughout the course of the study, improvements were made to problem definition input; geometric mesh partitioning; finite-element assembly of large, sparse linear systems; problem-specific matrix preconditioning; numerical solution of the linear system; and output of the approximate solution. The consequence of these improvements is that, at the end of the study, Chaste is able to solve a monodomain benchmark problem in close to real time. While some of the improvements made to the parallel Chaste code are specific to cardiac electrophysiology, many of the techniques documented in this paper are generic to the parallel finite-element method in other scientific application areas.

Keywords

Cardiac electrophysiology, bidomain equations, finite-element method, parallelisation, scaling analysis, hybrid linear solver

1 Introduction

The arrival of petascale computing and the advent of the exascale era has led to a remarkable increase in computational power available to simulation scientists. This exciting technological advance has paved the way to higher complexity simulation studies in many fields of science. The added complexity comes from: a) the use of more accurate representations of computational domains already under study (e.g. finer computational meshes), b) the development of models describing larger entities, or c) model coupling (e.g. multi-scale, multi-physics modelling). The consequences of this technological shift are twofold. Firstly, there has been an increase in computational resources never seen before in the form of more CPU cores,

[1] Centre for Computational Science, University College London, UK
[2] CoMPLEX, University College London, UK
[3] Fujitsu Laboratories of Europe Ltd, Hayes, UK
[4] Research School of Computer Science, The Australian National University, Canberra, Australia
[5] Department of Computer Science, University of Oxford, UK
[**] With the Department of Computer Science, University of Oxford during this study.

Corresponding author:
Miguel O Bernabeu, CoMPLEX, University College London, Gower Street, London, WC1E 6BT, UK.
Email: miguel.bernabeu@ucl.ac.uk

larger amounts of memory, and faster interconnect technology. Secondly, a progressive increase in the volume of input data that algorithms need to handle must be also considered. Operations that do not constitute a parallel bottleneck at lower core counts become a major obstacle in the petascale era. Examples are input/output operations, synchronisations, and, in general, any operation requiring sequential execution or access to non-replicated resources.

Petascale hardware is becoming more accessible to the scientific community with a total of 20 machines—as of June 2012—achieving a sustained LINPACK performance of 1 petaflops (10^{15} floating point operations per second) or more worldwide.[1] A reduced group of high-level scientific codes have also broken the petascale barrier (e.g. physics, materials science, and chemistry (http://www.olcf.ornl.gov/2010/11/15/ornl-systems-lead-in-petascale-science/)). To date, no code using the finite-element method (FEM) for the solution of cardiac electrophysiology problems has achieved this milestone. Hence, a detailed evaluation of the current parallel technology used for FEM simulation of cardiac electrophysiology is in order, to identify the main computational kernels involved, propose effective parallelisation approaches, and highlight where parallel bottlenecks remain.

There exists a large body of literature concerning the development of parallel cardiac electrophysiology solvers (see Bordas et al. (2009); Linge et al. (2009) and Clayton et al. (2011) for surveys). Examples of early contributions to parallel solution approaches are Fishler and Thakor (1991), Pollard and Barr (1991), Winslow et al. (1993), Ng et al. (1995), Saleheen et al. (1997), Quan et al. (1998) and Cai and Lines (2002). Common amongst most of these works is that they considered explicit solution schemes for the monodomain model, which can be parallelised very efficiently, and that they used shared-memory architectures. An early contribution to distributed-memory approaches was made by Porras et al. (2000), who compared four different parallel schemes for the two-dimensional monodomain equations. More recently, Vigmond et al. (2002) presented parallel computations on shared-memory architectures with two to four processors and compared the performance of a number of direct and iterative linear solvers.

In the context of distributed-memory architectures there exists a number of works that use the Message Passing Interface (MPI) standard (http://www.mpi-forum.org) and the PETSc library (http://www.mcs.anl.gov/petsc) for the development of parallel bidomain simulators. Early examples are Vigmond et al. (2003) and Colli-Franzone and Pavarino (2004). A similar PETSc-based approach is used in dos Santos et al. (2004) combined with parallel geometric multi-grid preconditioning. In Murillo and Cai (2004) the PETSc library was also used to devise a parallel bidomain solver based on a fully implicit time discretisation. More recently, in Plank et al. (2007) the solver presented in dos Santos et al. (2004) was extended to use algebraic multi-grid preconditioning. The same solver was adapted for the solution of the monodomain equations in Niederer et al. (2011), achieving close to optimal scalability with up to 1024 cores using both explicit and semi-implicit timestepping schemes, and achieving approximately 40% parallel efficiency using the explicit scheme on 16,384 cores. With the same number of cores, Reumann et al. (2009) shows 71% paralllel efficiency using an explicit finite difference method for the solution of the monodomain equations. Finally, in Vázquez et al. (2011) a large-scale computational mechanics simulation platform was adapted to solve the monodomain equations, achieving almost linear scaling on up to 1000 processors with an explicit timestepping scheme. The interested reader can refer to Keener and Bogar (1998) for a description of the timestepping methods mentioned here.

A performance comparison of the aforementioned solvers is difficult. They use different numerical schemes, are evaluated on platforms with major architectural differences, and authors do not usually release enough information about solution timesteps and tolerances for the level of accuracy of their solutions to be compared (the use of iterative numerical methods in some of the implementations allows for computational cost to be reduced by relaxing tolerances). A performance metric often reported by authors is the ratio between the time taken to perform a simulation and the amount of time simulated (i.e. the real—time ratio). Table 1 summarises some of the values found in the literature.

Note that the solvers reported to run efficiently using hundreds of processors or more (i.e. the last four in Table 1) share one or more of the following four limitations: i) they use the monodomain model, ii) they use explicit time discretisations, iii) they use spatial discretisation methods that only allow the use of regular grids (e.g. finite difference method), and/or iv) they are not freely available to the scientific community. This paper focuses on the development of open source cardiac simulation technology that achieves a similar degree of performance in large-scale high-performance computing (HPC) infrastructures using the bidomain equations and a semi-implicit time discretisation on arbitrary spatial domains pathmanathan10. The reasons for these choices are three-fold:

- several applications of interest (e.g. human shock-induced arrhythmogenesis (Bernabeu et al., 2010b) and drug-induced alterations on the body-surface ECG (Zemzemi et al., 2011)) require the use of the bidomain equations, since the monodomain model presents limitations for their study;
- explicit time discretisation imposes a constraint on the maximum timestep directly proportional to the grid edge length, so increasing the level of detail of the geometrical models will inevitably lead to increasingly shorter timesteps. This is likely to have a negative impact on overall performance. Unconditionally stable methods (e.g. semi-implicit time discretisation) allow the mesh to become more detailed without requiring shorter timesteps;

Table 1. Ratio between the time taken to perform a simulation and the amount of time simulated.

Reference	Core count	Mesh size	Real–time ratio	Summary
Potse et al. (2006)	32	26–55 M nodes	288,000	Monodomain and bidomain; finite difference method; explicit time discretisation; Bernus et al. cell model (2002).
ten Tusscher et al. (2007)	20	13.5 M voxels	43,200	Monodomain; finite difference method; explicit time discretisation; ten Tusscher et al. cell model (2006).
Reumann et al. (2009)	16,384	128.9 M elements	13,180	Monodomain; finite difference method; ten Tusscher et al. cell model (2004).
Pope et al. (2011)	16,384	128.9 M elements	2,042	Monodomain; finite difference method; explicit time discretisation; ten Tusscher et al. cell model (2004).
Vázquez et al. (2011)	500	17 M elements	450	Monodomain; finite-element method; implicit, explicit, and Crank–Nicolson time discretisation; FitzHugh–Naguno cell model (1961).
Niederer et al. (2011)	16,384	26 M nodes	240	Monodomain; finite-element method; explicit and Crank–Nicolson time discretisation; ten Tusscher et al. cell model (2006).

- unstructured grids allow for more realistic representation of ventricular surfaces and fine-grained features than structured grids. The finite-element method is preferred for the spatial discretisation of the bidomain equations for its support of unstructured grids.

This work has been conducted in the framework of the open source Computational Biology simulation package Chaste (Pitt-Francis et al., 2009).

The rest of the paper is structured as follows. Section 2 gives a brief introduction to computational cardiac electrophysiology, including the underlying mathematical models of interest. Section 3 presents the main components of the cardiac electrophysiology solver in Chaste. The parallelisation strategy adopted for each of these components is described in Section 4. Section 5 describes the benchmark used for the evaluation of the scalability improvements presented along with its results. Finally, Section 6 discusses the results and presents the conclusions.

2 Computational cardiac electrophysiology

At a cellular level, models of cardiac electrophysiology of different species and cell types have been successfully developed and used for a variety of applications. The models include representation of the main mechanisms of ionic transport across the cell membrane and between subcellular compartments. From a mathematical point of view, the models typically consist of systems of ordinary differential equations (ODEs), with the most detailed ones (such as Iyer et al. (2007)) having over 60 ODEs. These models allow representation of the effect of mutations, drugs and disease on ion channel function.

At a tissue level, simulating propagation of electrical excitation through cardiac tissue (mainly myocytes) involves solving a system of partial differential equations (PDEs)—the bidomain equations—over an anatomically based computational grid with realistic representation of geometry and microstructure. The level of detail in the models has grown due to a better characterisation of cardiac structure provided by recent advances in medical imaging techniques (Burton et al., 2006). It is now possible to generate highly detailed representations of cardiac structures such as blood vessels, papillary muscles, the Purkinje network and fibre orientation. Preliminary studies have provided insight into the role of previously neglected cardiac structures on ventricular activation following electrical pacing and shocks (Burton et al., 2006). However, this comes at the cost of an increase in problem size and hence computational burden.

2.1 The bidomain equations

For a bidomain simulation of cardiac tissue contained in a conductive surrounding medium, (referred to as the *bath*) the magnitudes of interest are intracellular and extracellular potentials ($\phi_i(\mathbf{x},t)$ and $\phi_e(\mathbf{x},t)$), and their difference ($V(\mathbf{x},t)=\phi_i-\phi_e$). The tissue Ω and the bath Ω_b are disjoint domains with interface $\partial\Omega$, with ϕ_i—and therefore V—defined only in Ω, but ϕ_e defined throughout $\Omega\cup\Omega_b$. Keener and Sneyd (1998) showed that V and ϕ_e satisfy

$$\chi\left(C\frac{\partial V}{\partial t}+I_{\text{ion}}\right)-\nabla\cdot(\sigma_i\nabla(V+\phi_e))=-I_i^{(\text{vol})},\quad \text{in }\Omega \tag{1}$$

$$\nabla\cdot(\sigma_i\nabla(V+\phi_e)+\sigma_e\nabla\phi_e)=0,\quad \text{in }\Omega \tag{2}$$

$$\nabla\cdot(\sigma_b\nabla\phi_e)=0,\quad \text{in }\Omega_b \tag{3}$$

$$\frac{\partial\mathbf{u}}{\partial t}=\mathbf{f}(\mathbf{u},V),\text{ in }\Omega$$

where σ_b is the bath conductivity, σ_i is the intracellular conductivity tensor, σ_e the extracellular conductivity tensor, χ is the surface-area-to-volume ratio and C is the membrane capacitance per unit area. The vector $\mathbf{u}$ contains cell-level variables, such as ionic concentrations and membrane-gating variables, and $I_{\text{ion}}\equiv I_{\text{ion}}(\mathbf{u},V)$ is the ionic current per unit surface area. I_{ion} and $\mathbf{f}$ are determined by an electrophysiological cell model. The source term $I_i^{(\text{vol})}$ is the intracellular stimulus per unit volume, used to elicit propagation.

Appropriate boundary conditions are

$$\mathbf{n}\cdot(\sigma_{\mathrm{i}}\nabla\phi_{\mathrm{i}})=0,\quad \text{on } \partial\Omega \text{ (i.e. the bath-tissue boundary)} \tag{4}$$

$$\mathbf{n}\cdot(\sigma_{\mathrm{b}}\nabla\phi_{\mathrm{e}})=I^{(\mathrm{E})},\quad \text{on } \partial\Omega_{\mathrm{b}}\backslash\partial\Omega \text{ (i.e. the external bath boundary)} \tag{5}$$

where $\mathbf{n}$ is the outward-facing unit normal. Here $I^{(\mathrm{E})}$ is a stimulus current per unit area representing an external current flowing into the domain and causing an electrical shock on the tissue surface. The problem is stated without any Dirichlet boundary conditions on ϕ_{e} and therefore it is defined up to a constant. For a solution to exist, input current has to be equal to output current, i.e. $\int_{\partial\Omega_{\mathrm{b}}\backslash\partial\Omega} I^{(\mathrm{E})}\,\mathrm{d}S=0$.

Following the spatial discretisation of $\Omega\cup\Omega_{\mathrm{b}}$, let us suppose the first N nodes are contained in the closure of Ω, and the next M nodes are the remaining nodes in the bath ($\overline{\Omega}_{\mathrm{b}}\backslash\overline{\Omega}$). The semi-implicit finite-element formulation of the bidomain equations can be shown to be (e.g. see Pathmanathan et al. (2010)): find $\mathbf{V}_{(1)}^{m+1}$, $\Phi_{(1)}^{m+1}$, and $\Phi_{(2)}^{m+1}$ such that

$$\begin{bmatrix} \frac{\chi C}{\Delta t}M+K[\sigma_{\mathrm{i}}] & 0 & K[\sigma_{\mathrm{i}}] & 0\\ 0 & I_M & 0 & 0\\ K[\sigma_{\mathrm{i}}] & 0 & \mathcal{K}_{(1,1)} & \mathcal{K}_{(1,2)}\\ 0 & 0 & \mathcal{K}_{(2,1)} & \mathcal{K}_{(2,2)} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{(1)}^{m+1}\\ \mathbf{V}_{(2)}\\ \Phi_{(1)}^{m+1}\\ \Phi_{(2)}^{m+1} \end{bmatrix} = \begin{bmatrix} \frac{\chi C}{\Delta t}M\mathbf{V}_{(1)}^{m}+\mathbf{c}^{m} & \}\text{size } N\\ \mathbf{0} & \}\text{size } M\\ \mathbf{0} & \}\text{size } N\\ \mathbf{d}^{m} & \}\text{size } M \end{bmatrix} \tag{6}$$

where, at timestep m, $\mathbf{V}_{(1)}^{m}$ and $\Phi_{(1)}^{m}$ are vectors respectively containing the values of V and ϕ_{e} at the tissue nodes, $\Phi_{(2)}^{m}$ is a vector containing the values of ϕ_{e} at the bath nodes, $\mathbf{V}_{(2)}$ is a list of dummy variables (independent of m and nominally representing voltage in the bath), $\mathbf{c}^{m}$ is a vector representing the transmembrane ionic currents (I_{ion} and $I_{\mathrm{i}}^{(\mathrm{vol})}$) at the tissue nodes, and $\mathbf{d}^{m}$ is a vector representing the stimulus current $I^{(\mathrm{E})}$ at the bath nodes. Finally, M, $K[\sigma_{\mathrm{i}}]$, and

$$K[\sigma_{\mathrm{i}}+\sigma_{\mathrm{e}}]=\begin{bmatrix}\mathcal{K}_{(1,1)} & \mathcal{K}_{(1,2)}\\ \mathcal{K}_{(2,1)} & \mathcal{K}_{(2,2)}\end{bmatrix} \tag{7}$$

are the appropriate finite-element mass and stiffness matrices and I_M is the identity matrix. $\mathbf{V}_{(2)}$ and I_M are introduced for ease of implementation in order to ensure that there are two degrees of freedom associated with each mesh node (regardless of location in Ω or Ω_{b}).

For simplicity, the linear system in equation (6) can be rewritten as

$$Ax=\begin{bmatrix}A_1 & B^T\\ B & A_2\end{bmatrix}x=\mathrm{b} \tag{8}$$

with blocks $A_1=\begin{bmatrix}\frac{\chi C}{\Delta t}M+K[\sigma_{\mathrm{i}}] & 0\\ 0 & I_M\end{bmatrix}, B=\begin{bmatrix}K[\sigma_{\mathrm{i}}] & 0\\ 0 & 0\end{bmatrix}$ and $A_2=K[\sigma_{\mathrm{i}}+\sigma_{\mathrm{e}}]$

2.2 The monodomain model

Under certain circumstances (e.g. see [Keener and Sneyd(1998)]) equations (1)–(3) can be reduced to a PDE of a single unknown, V, coupled to the system of ODEs describing transmembrane ionic transport. The problem to solve, therefore, becomes

$$\begin{aligned}\chi\left(C\frac{\partial V}{\partial t}+I_{\mathrm{ion}}\right)-\nabla\cdot(\sigma\nabla V)&=-I^{(\mathrm{vol})},\quad \text{in } \Omega\\ \frac{\partial\mathbf{u}}{\partial t}&=\mathbf{f}(\mathbf{u},V),\text{in } \Omega\end{aligned} \tag{9}$$

where σ is the parallel sum of intra- and extra-cellular conductivity

$$\sigma=\frac{\sigma_{\mathrm{i}}\sigma_{\mathrm{e}}}{\sigma_{\mathrm{i}}+\sigma_{\mathrm{e}}}$$

with boundary conditions

$$\mathbf{n}\cdot(\sigma\nabla V)=0,\quad \text{on } \partial\Omega \tag{10}$$

This simplification notably reduces the computational burden associated with the numerical solution of the bidomain model. However, it is not suitable for all kinds of application studies. More precisely, More precisely, Potse et al. (2006) concluded that, in the absence of applied currents, propagating action potentials on the scale of a human heart can be studied with a monodomain model. However, note that bath-loading effects are not correctly captured, as discussed in Bishop and Plank (2011).

The final finite-element linear system to be solved for the monodomain model at each timestep can be shown to be: find $\mathbf{V}_{(1)}^{m+1}$ such that

$$\left(\frac{\chi C}{\Delta t}M+K[\sigma]\right)\mathbf{V}^{m+1}=\left(\frac{\chi C}{\Delta t}M\mathbf{V}^{m}+\mathbf{c}^{m}\right) \tag{11}$$

3 Large-scale cardiac electrophysiology simulation with Chaste

Chaste is an open source computational framework for the simulation of systems in biology, with a particular focus on cardiac electrophysiology, cancer modelling, and tissue growth. It aims to be extensible, robust, fast, accurate, maintainable and to use state-of-the-art numerical techniques, is distributed under the LGPL and BSD licenses and can be downloaded from www.cs.ox.ac.uk/chaste. For a detailed description of the Chaste project, its aims and functionality, the reader can refer to Pitt-Francis et al. (2009).

3.1 Main components of Chaste's cardiac electrophysiology solver

Running a cardiac electrophysiology simulation consists of a number of different stages, illustrated in Figure 1. In the first phase of the simulation, the simulation parameters (including simulation duration, timesteps, tolerances, cell models etc.) are defined, the mesh is loaded from file,

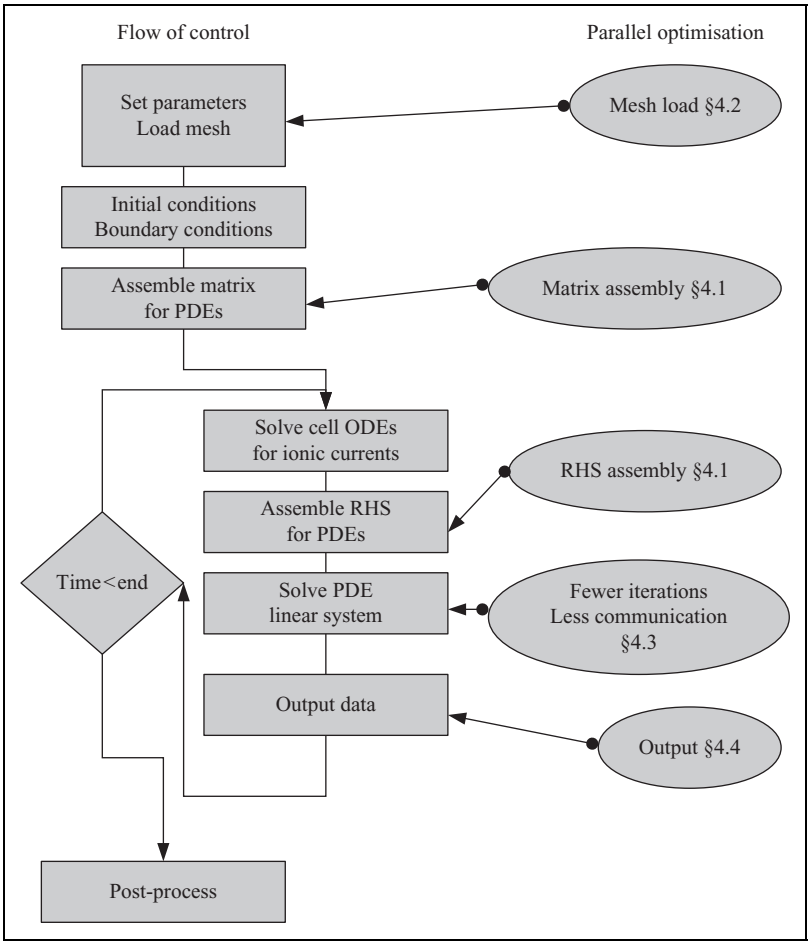

Figure 1. A schematic of the main components of the solver with cross-references to the sections describing parallel improvements made during this study.

initial and boundary conditions are specified and the matrix defined in equations (6) or (11) is assembled. Subsequently, at each timestep, individual cell models at each mesh node are integrated (ODE solution), the right-hand-side vector in equations (6) or (11) is assembled and then the resulting linear system is solved. It should be noted at this point that although Figure 1 illustrates the stages of a cardiac electrophysiology simulation code, there are aspects of the problem which are *generic* to FEM programming in general. Nevertheless, some differences may occur: e.g. many FEM solvers do not need an ODE solver stage, problems with a time-varying left-hand side would require the "Assemble matrix" component to be inside the time-loop rather than in the initial phase, non-linear PDE solvers would require the linear system stage to be inside another iterative loop, and static problems like stress analysis would need no time loop. In the following sections (as annotated in Figure 1) the schemes implemented for reducing the parallel scaling bottlenecks in the Chaste code are described.

3.2 Initial performance evaluation

In order to evaluate the performance of Chaste on large-scale supercomputers with realistic 3D cardiac models, an electrical propagation benchmark using a 4 million node anatomically-based rabbit ventricular mesh (Bishop et al., 2009) was designed. The benchmark consists of an apical stimulus followed by simulation of 100 ms of bidomain activity in the cardiac tissue only (i.e. there is no bath). The choice of 100 ms as total simulation time is a compromise between: i) simulating for a period of time that is short enough to be tractable with the lowest core count considered (i.e. 32 cores); and ii) choosing a simulation time that is representative of our applications of interest (e.g. hundreds of ms in Bernabeu et al. (2010b) and Zemzemi et al. (2011)). Table 2 summarises the experimental details.

The following parameters were used in equations (1)–(5): $\chi = 1400\ \text{cm}^{-1}$, $\mathcal{C} = 1.0\text{F/cm}^2$, $\sigma_\text{i} = \text{diag}(1.7, 1.7, 1.7)\text{ms/cm}$, and $\sigma_\text{e} = \text{diag}(6.2, 6.2, 6.2)\text{ms/cm}$, where $\text{diag}(x, y, z)$ is a 3×3 diagonal matrix with values x, y, z along the diagonal. Further, since the geometry includes no bath, $\Omega_\text{b} = \emptyset$. For solving the linear systems, Chaste uses implementations of various methods (see later for details) from PETSc 3.0.0-p8 with no changes to their default parameter values.

Prior to making any performance improvements, the benchmark was run on Phase 2a of the HECToR supercomputer, a Cray XT4 system with 3072 compute nodes (at the

Table 2. Benchmark configuration.

simulation duration	100.0 ms
stimulus type	apical
stimulus start time	0 s
stimulus duration	0.5 ms
PDE timestep	0.01 ms
cell model	Luo–Rudy (1991)
ODE timestep	0.01 ms

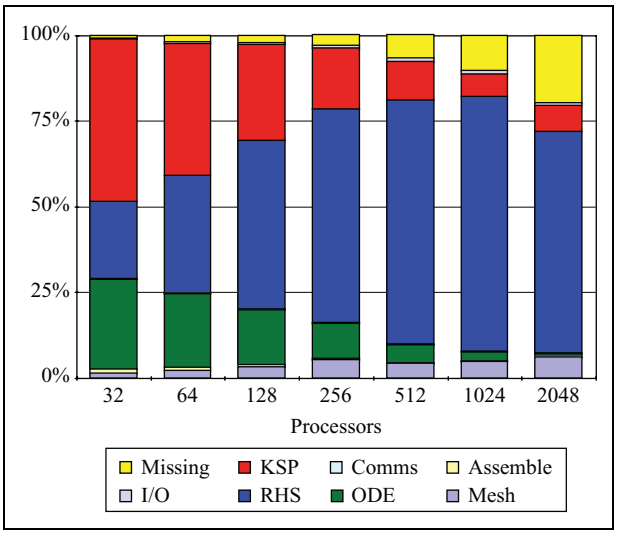

Figure 2. Original time breakdown before the improvements presented in this paper.

time of the study). Each compute node consisted of an AMD 2.3 GHz Opteron Barcelona quad-core and each quad-core socket shared 8 GB of memory and a Cray SeaStar2 chip router with 6 links used to implement a 3D-torus network topology. Figure 2 presents, for an increasing number of cores, the proportion of time spent by the benchmark simulation in each of the stages described in Section 3.1.

The first thing to note is that, as core count increases, the time within the 'Missing' section starts to increase. Further profiling confirmed that it was spent outside the main stages cited earlier and therefore it was potentially redundant. More precisely, it was identified to be unnecessary synchronisation and disk access contention when writing Chaste's log files. Interestingly, this performance degradation had passed previously unnoticed when running in small size clusters and workstations (note how it is hardly visible for $p = 32$). This is a good example of how certain operations that scale well on up to a few 10s of nodes become major bottlenecks at large scale.

It is also clear from Figure 2 that the proportion of time spent in 'RHS assembly' increases with core count and starts to dominate the total execution time for $p > 128$. This is a good indicator of poor scaling and therefore it was one of the first issues addressed (Section 4.1.2). It can also be seen that the time spent in 'Mesh load' also scaled poorly. There are two reasons for this: the sequential nature of the algorithm used for domain decomposition (METIS) and disk access contention. Section 4.2 describes how these two issues were addressed.

The two stages taking most of the time at $p = 32$ (i.e. 'System solution', labelled as KSP, and 'ODE solve') do not show major scaling problems or, where they do, it is not as severe as those seen with 'RHS assembly' or 'Mesh load'. For 'ODE solve', this is expected behaviour since the problem is embarrassingly parallel, so will scale linearly provided that an even distribution of the ionic models among the available processors is generated. This part of the code has also been tuned (see Cooper (2009)) in order to reduce ODE solution time, but this tuning has had no impact on overall parallel efficiency. In contrast, 'System solution' involves the use of tightly coupled parallel algorithms that are likely to scale suboptimally. Following the improvements introduced in the 'Missing', 'RHS assembly', and 'Mesh load' stages, the analysis presented in Figure 2 was repeated, showing 'System solution' as the next target for improvement (results not shown here). The proposed improvements are summarised in Section 4.3.

Finally, it is noted that most of the initial phases of the solver (such as the application of boundary conditions or the assembly of the system matrix) do not have a major impact in parallel scalability. The post-processing phase (which essentially involves converting data formats for visualisation and analysing data) is application-specific and was outside the scope of this publication.

4 Parallelisation

4.1 Linear system assembly

When solving systems of coupled PDEs with FEM, several fields are computed at each discretisation point (two in the case of the bidomain equations). In this context, a design decision has to be taken regarding how unknowns are arranged in the linear system. The options are to use either a 'blocked' distribution (i.e. $\mathbf{x} = [V_1, V_2, \ldots, V_n, \phi_{e1}, \phi_{e2}, \ldots, \phi_{en}]^T$ in equation (8), where V_i is the value of V at the i-th mesh node and similarly with ϕ_e), an 'interleaved' distribution (i.e. $\mathbf{x} = [V_1, \phi_{e1}, V_2, \phi_{e2}, \ldots, V_n, \phi_{en}]^T$ in equation (8)) or some hybrid approach. In sequential implementations, the choice has a moderate impact on performance often associated with matrix bandwidth reduction. In parallel, there exists an important correlation between the approach taken and the volume of communication required for the assembly and solution of FEM linear systems. Some of the implications of this design decision are discussed in this section.

Another relevant design decision concerns the way that the computational domain is partitioned. In principle, it is possible to partition either node- or element-wise. This decision has important consequences in terms of load balancing. The initial performance evaluation in Section 3.2 showed that the linear system and ODE solution stages

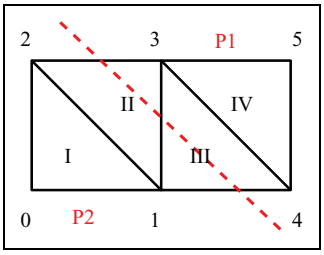

Figure 3. Model problem geometry. Dashed line shows parallel partition.

dominate total execution time for low core counts. In both cases, execution time at each subdomain is a function of the number of grid points assigned (i.e. number of degrees of freedom and number of cell models, respectively). Therefore, it makes sense to partition node-wise to ensure that the number of grid points in each subdomain stays as constant as possible. Partitioning element-wise would instead optimise the element distribution (which would have a positive impact on stages like matrix assembly). Unfortunately, it would potentially generate a suboptimal node distribution and therefore unbalance the stages that dominate total execution time.

Hence, mesh elements need to be distributed among the available processors based on node ownership. For elements located at the border between two partitions (like II and III in Figure 3) a strategy for ownership assignment is required. Some options are: i) assigning each of them to the processor owning the larger number of nodes within the element, ii) allowing multiple ownership of an element or iii) distributing them in such a way that the number of elements owned by each processor is balanced. Some of the implications of this choice are discussed later in this section.

4.1.1 Matrix assembly. A commonly used method to assemble matrix A in equation (8) is to compute the contribution of each element and then map the coefficients of this local matrix into the overall system matrix. In the notation of Wathen(1989), let $e = 1, \ldots, n_{\mathrm{el}}$ be a numbering of the elements, let n_e be the number of nodes in element e multiplied by the number of fields to solve and let n be the total number of nodes in the mesh multiplied, again, by the number of fields to solve. Further, let $E_e \in R^{n_e \times n_e}$ be the element coefficient matrix for each e and $L_e = [l_{jk}] \in R^{n_e \times n}$ where

$$l_{jk} := \begin{cases} 1, & \text{if } j-\text{th local node has global index } k, \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

that is the Boolean matrix that maps E_e into A. Then the global assembly process can be formulated as:

$$A = \sum_{e=1}^{n_{el}} L_e^T E_e L_e \tag{13}$$

where the effect of pre- and post-multiplying E_e by L_e^T and L_e can be seen as converting all the $n_e \times n_e$ matrices into a $n \times n$ matrix.

In a parallel simulation, the n_{el} mesh elements must be assigned ownership according to one of the strategies described above. For this study, multiple ownership of elements was implemented because this ensures that no data communication is required when assembling the system matrix, at the cost of replicating some computation. The overhead associated with that replication was quantified in Pathmanathan et al. (2010): for a 322, 267-element mesh and 16 processors, the average number of elements assembled by each of the 16 processors was 6.78% against a theoretical optimum of 6.25%.

More formally, let $\mathcal{E}$ be the set of mesh elements partitioned into p potentially overlapping subdomains $\mathcal{E}_i$ such that

$$\varepsilon = \bigcup_{i=1}^{p} \mathcal{E}_i, \tag{14}$$

$$|\mathcal{E}| \leq \sum_{i=1}^{p} |\mathcal{E}_i| \tag{15}$$

and consider a row-based distribution of the system matrix

$$A = \begin{bmatrix} \mathrm{A}_1 \\ \vdots \\ \mathrm{A}_p \end{bmatrix} \tag{16}$$

where $A_i \in R^{\frac{n}{p} \times n}$ is the rectangular block of A owned by processor i (assuming $n = kp$, $k \in N$ for simplicity). Let $M_{e,i} = [m^i_{jk}] \in R^{n_e \times \frac{n}{p}}$ where

$$m^i_{jk} := \begin{cases} 1, & \text{if } l_{j\hat{k}} = 1 \text{ with } \hat{k} := (i-1)\frac{n}{p} + k \text{ in (12)}, \\ 0, & \text{otherwise}, \end{cases} \tag{17}$$

be the Boolean matrix that maps E_e into A_i (i.e. maps only the rows of E_e that belong to processor i, if any). It can be seen that A_i can be assembled from local data without the need of communications, i.e.

$$A_i = \sum_{e \in \varepsilon_i} M^T_{e,i} E_e M_{e,i} \tag{18}$$

if and only if i) an interleaved ordering of unknowns and ii) multiple ownership of elements are implemented.

These two design decisions imply that for an element e owned by processor i and not located at the partition border (e.g. I and IV in Figure 3), E_e gets mapped onto A_i rows only. For an element e' located at the partition border (e.g. II and III in Figure 3), $E_{e'}$ will be assembled by all the processes meeting at that border, but the boolean matrix $M_{e,i}$ will only map onto A_i the rows of $E_{e'}$ that are locally owned by i.

4.1.2 Right-hand side assembly. In Chaste's original design, vector **b** in equation (8) was assembled analogously to equation (13)

$$\mathbf{b} = \sum_{e=1}^{n_{el}} L_e^T b_e L_e \tag{19}$$

where e, n_{el}, and L_e have been previously defined and $b_e \in R^{n_e}$ are the element-wise contributions to the right-hand side of equations (6) or (11).

It was shown in Pathmanathan et al. (2010) that, provided I_{i} and I_{ion} are known point-wise at the nodes, the main computational kernel in equations (6) and (11) can be conveniently recast as

$$\frac{\chi C}{\Delta t} M(\mathbf{V}_{(1)}^m + \mathbf{C}^m) \tag{20}$$

where $\mathbf{C}^m$ is a vector with the nodal values of the source term at timestep m. This operation can therefore be implemented as a vector summation followed by a matrix-vector product, yielding a speedup by a factor of 68 over the element-wise evaluation in equation (19) (see Pathmanathan et al. (2010) for more details). Another valuable property of this formulation is that the values of $\mathbf{V}_{(1)}^m$ do not need to be communicated explicitly across processor borders for right-hand side (RHS) assembly since the matrix-vector product will do this implicitly, thus avoiding the usual halo communication step in FEM solvers.

One final consideration is that in the case of bidomain RHS assembly, equation (20) only accounts for the first N entries (out of the total size $2(N+M)$) of the RHS of equation (6). However, its assembly requires solving $I_{\text{ion}}(\mathbf{u}^n, V^n)$ at each grid point (ODE system solution stage). In order to achieve good load balance, ownership of the grid points (and therefore each ODE system) is distributed evenly among the available processors. With the original data layout proposed in equation (6)

$$\mathbf{b} = \begin{bmatrix} \frac{\chi C}{\Delta t} M(\mathbf{V}_{(1)}^m + \mathbf{C}^m) \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{d}^m \end{bmatrix} \tag{21}$$

and a data partition compatible with equation (16)

$$\mathbf{b} = \begin{bmatrix} \mathrm{b}_1 \\ \vdots \\ \mathrm{b}_\mathrm{p} \end{bmatrix} \tag{22}$$

with $\mathbf{b}_i \in R^{\frac{n}{p}}$ being the subvector owned by processor i (assuming $n = kp$, $k \in N$ for simplicity), then processors owning nodes $N+M+1$ to $2(N+M)$ would have to communicate I_{ion} values computed locally to the processors owning the first N rows for them to finish assembling the system right-hand side. Therefore, it is advantageous to rearrange the equations in way that $\mathbf{C}^m$ is assembled from values of I_{ion} computed locally, therefore avoiding communication. Similarly, efficiency would increase if all the processors cooperate in the evaluation of the matrix-vector product, not only those owning the first N rows. Using an interleaved unknown ordering (as described at the beginning of this section (Section 4.1)) satisfies the previous two requirements ensuring good load balance.

Finally, evaluating $\mathbf{d}^m$ in equation (21) is a potential source of load imbalance, since it involves computing surface integrals over a subset of the mesh faces (i.e. application of Neumann boundary conditions) at certain timesteps. The size of the subset is potentially different between simulations so generic solutions for load balancing are difficult. Nevertheless, the subset is often small and the operations involved considerably less expensive than equation (20).

4.1.3 Implementation details. Chaste uses PETSc parallel data structures (Balay et al., 2010) wherever possible. When constructing parallel matrices, PETSc allows for data to be generated non-locally. At the last stage of construction (known as assembly), PETSc will work out the appropriate owner and migrate the data. This process requires several rounds of parallel reductions in order to ensure consistency among all the processes, even when no data is being migrated. It is possible to disable this check provided that all the data is generated locally, reducing the number of parallel reductions required with an important impact in parallel scalability. In PETSc version 3.0 this can be done for matrices with the following function call:

`MatSetOption(lhs_matrix, MAT_IGNORE_OFF_PROC_ENTRIES, PETSC_TRUE)` where `lhs_matrix` is a parallel matrix, in this case the one storing the system matrix A. The same consideration applies to the assembly of vectors. In this case, the portion of vector $\mathbf{C}^m$ in equation (20) owned by each process can be assembled from local data and therefore there is no need to check for non-locally generated data. In PETSc version 3.0 this can be enforced with the following call:

`VecSetOption(petsc_vector, VEC_IGNORE_OFF_PROC_ENTRIES, PETSC_TRUE)` where `petsc_vector` is any PETSc vector.

4.2 Mesh load and partitioning

The first step in running a simulation is to read in the mesh representing the system geometry from a file. In a parallel simulation it is then also necessary to partition this mesh between the available processes. In general, this one-time cost is relatively low for a sequential simulation—and, hence, little consideration has been given to optimising it.

At the beginning of this work, Chaste used the METIS-based mesh partitioning algorithm described in Pathmanathan et al. (2010). This method requires each process to perform a sequential partition of the mesh in order to determine which nodes it owns. This makes METIS unsuitable for partitioning meshes that are too large to fit into memory on a single core. Further, since each process calls METIS sequentially and the temporal cost of the partitioning algorithm is a function of the number of partitions, the overall time to obtain the partition increases (even for a fixed mesh) with the number of processes. Hence, for HPC simulations, Amdahl's law (Amdahl, 1967) means that the cost of loading the mesh rapidly increases relative to all other parts of the code (the work for which *can* be distributed). Modifying the algorithm to use ParMETIS (Schloegel

Algorithm 1: Original mesh load algorithm (omitting loading of boundary element files). ASCII file format requires the mesh reader to visit every entry of a file, even when the node or element it relates to is owned by a different process. Constructing the mesh requires two loops over the entire element file and one loop over the entire node file. Performing the METIS partitioning requires a further loop through the element file.

Input: Mesh files (ASCII format): list of nodes (.node), list of elements (.ele).
Output: Chaste mesh object, including node objects for local and halo nodes, element objects (with pointers to contained nodes) for local elements.

```
1   ComputeMetisPartitioning();
2   for element = 0 to num_elements − 1 do                    /* Loop over all elements */
3     ReadNextElementFromFile();
4     for element_node=0 to num_nodes_per_element do
5       if element_node ∈ owned_nodes then ownedElements.insert(element);
6       else possible_halo_nodes.insert(elememt_node);
7     if element ∈ owned_elements then haloNodes.insert(possible_halo_nodes);
8   for node = 0 to num_nodes − 1 do                          /* Loop over all nodes */
      // Previous element file loop identifies halo nodes.
9     ReadNextNodeFromFile();
10    if node ∈ ownedNodes then ConstructNode(node);
11    else if node ∈ haloNodes then ConstructHaloNode(node);
12  ResetElementFile();
13  for element = 0 to num_elements − 1 do                    /* Loop over all elements */
      // Previous node file loop creates node objects to be pointed to.
14    ReadNextElementFromFile();
15    if element ∈ ownedElements then
16      ConstructElement(element);
17      for element_node=0 to num_nodes_per_element do
18        SetupPointerFromElementToContainedNode(element_node);
```

et al., 2002), the parallel version of METIS, is relatively straightforward (in Chaste, ParMETIS is instead accessed via PETSc wrapper functions that provide the functionality required to partition based on nodes that is absent when calling ParMETIS directly).

A good mesh partitioning will not only balance the amount of geometry on each compute node (and thus balance the work load), but will also minimise the communication boundaries and reduce the skyline of the main matrix. The effect of the mesh partition on the matrix structure results in improvements to the matrix assembly (Section 4.1.1), to the RHS assembly (Section 4.1.2) and the system solution (Section 4.3). We have previously quantified these improvements in Pathmanathan et al. (2010).

The importance of the mesh partitioning step has been previously acknowledged in the literature (e.g. see Devine et al. (2005) and Teresco et al. (2006)). Furthermore, it has become of increasing interest due to the substantial increment in the number of cores available in emerging architectures (Devine et al., 2006; Zhou et al., 2012). In particular, the use of ParMETIS-based partitioning algorithms for parallel finite-element method simulations has been widely reported in the literature (Piggott et al., 2008; Sahni et al., 2009; Bekas et al., 2010; Shadid et al., 2010; Niederer et al., 2011). In the current section, we also consider an often neglected aspect of the problem: the design of a scalable algorithm that, given a ParMETIS partition, reads the mesh from disk and creates the relevant data structures.

The original file format used in Chaste to represent meshes consisted of separate ASCII files containing lists of node coordinates (.node file), a list of the nodes contained in each element (.ele file) and a list of the nodes contained in each surface element (.face file). For large meshes these files can grow to be very large, e.g. for a mesh containing approximately 4 million nodes and 24 million elements the file sizes are 129 MB (node), 958 MB (element) and 33 MB (face). Further, non-constant field length in ASCII files makes it difficult to implement random access, so it is necessary for each process to read the three files in their entirety, determine what information it needs to retain and discard the rest. In practice—since the software must be able to deal with files that are too large to fit in the memory available to a single process—the element file must be read several times, as can be seen in Algorithm 1, which shows the initial Chaste mesh load algorithm (excluding the face file read, which is equivalent to the final element file read).

In order to reduce the amount of data that each process was required to read, the files were converted to binary format. This has two consequences: the files are smaller (and hence can be read in less time) and each entry is a fixed size (allowing random access, meaning that each process can jump straight to the entries it needs to read). However, since a process owns an element if it also owns one or more of its nodes, it is necessary for that process to interrogate each element in turn to determine whether or not it owns it—and (as seen in the first loop over elements in Algorithm 1) this necessitates a complete pass through the element file (generally the largest of the mesh files). Even when using a binary file format this is expensive and does

Algorithm 2: New mesh load algorithm (ommitting loading of boundary element files). Binary file format allows random file access and .ncl file provides a fast way of determining locally owned elements. Constructing the mesh requires locally owned elements to be read from file twice each, locally owned nodes to be read from file twice each and halo nodes to be read from file once each. Performing the ParMETIS partitioning requires further file accesses to the element file—but does not require each process to read the entire file.

Input: Mesh files (binary format): list of nodes (.node), list of elements (.ele), list of elements containing each node (.ncl).
Output: Chaste mesh object, including node objects for local and halo nodes, element objects (with pointers to contained nodes) for local elements.

```
1  ComputeParmetisPartitioning();
2  for local_node = 0 to num_local_nodes − 1 do                    /* Loop over local nodes */
3    elements_containing_node = GetContainingElementsFromNclFile(local_node);
4    ownedElements.insert(elements_containing_node);
5  for local_element = 0 to num_local_elements − 1 do              /* Loop over local elements */
6    ReadElementFromFile(local_element);
7    for element_node=0 to num_nodes_per_element do
8      if element_node ∉ owned_nodes then haloNodes.insert(element_node);
9  for local_node = 0 to num_local_nodes − 1 do                    /* Loop over local nodes */
10   ReadNodeFromFile(local_node);
11   ConstructNode(local_node);
12 for halo_node = 0 to num_halo_nodes − 1 do                      /* Loop over halo nodes */
13   ReadNodeFromFile(halo_node);
14   ConstructHaloNode(halo_node);
15 for local_element = 0 to num_local_elements − 1 do              /* Loop over local elements */
16   ReadElementFromFile(local_element);
17   ConstructElement(local_element);
18   for element_node = 0 to num_nodes_per_element do
19     SetupPointerFromElementToContainedNode(element_node);
```

not scale in parallel. Thus, a fourth (binary) mesh file was introduced for the largest meshes. This is the reverse of the element file: containing a list of which elements each node is contained in (known as the node-connectivity list or .ncl file). Each process can then access the parts of the .ncl file that correspond to the nodes that it owns and very rapidly construct a list of the elements that it owns. The process is then able to access directly only the parts of the element file that it needs to. Note that for a simulation on n processes, each process can be expected to own around $1/n$-th of the total number of elements—so introducing the node-connectivity file is key to making the mesh load scale. The latest Chaste mesh load algorithm is shown in Algorithm 2.

The improvements presented in this section only refer to algorithmic problem considerations, that is, ensuring that the volume of data read by each process decreases with the number of processes involved (for a fixed problem size). The authors believe that the techniques presented are generic enough to be usable by other parallel finite-element codes. However, we also acknowledge that achieving a good interaction with the underlying parallel file system will greatly influence the algorithm implementation performance. Such a task requires substantial knowledge of the underlying parallel file system and architecture and is beyond the scope of this work.

4.3 Linear system solution

Most authors choose the conjugate gradient (CG) algorithm for the solution of mono/bidomain FEM linear systems (e.g. Colli-Franzone and Pavarino (2004), Plank et al. (2007), Pennacchio and Simoncini (2009) and Pathmanathan et al. (2010)). However, other Krylov subspace methods such as GMRES (e.g. Whiteley (2006)) or Bi-CGSTAB (e.g. Potse et al. (2006)) have been successfully applied as well. In order to study the parallel efficiency of different iterative solvers, the different computational kernels involved must be considered individually. In the case of Krylov subspace methods, these are: i) vector inner products, ii) vector–vector linear combination (axpy) operations, iii) matrix–vector products, and iv) preconditioner application. axpy operations do not compromise parallel scalability since they can be performed without need of communication (assuming a consistent parallel distribution of all the vectors involved). Matrix–vector products require a certain degree of communication, but, as shown in Pathmanathan et al. (2010), reduction of the matrix bandwidth through the use of graph-based domain decomposition techniques increases scalability. The scalability of the preconditioning step is determined by the preconditioning technique of choice: preconditioners such as point Jacobi or block Jacobi with incomplete factorisation at each subblock do not require communications. Whole matrix incomplete factorisation or multigrid techniques are tightly coupled algorithms that require a higher degree of communication. Finally, vector inner products are communication-intensive operations as they are often implemented as parallel reductions. Parallel reductions are also required when checking for convergence if the stop criteria are based on the evolution of the l^2-norm of the residual.

4.3.1 Inner product reduction. Several authors (e.g. Barrett et al. (1994) and Gutknecht and Röllin (2002)) have proposed the use of the Chebyshev Iteration (CI) method (Golub and Van Loan, 1996) for the solution of the symmetric linear systems

$$A\mathbf{x}^* = \mathbf{b}, \quad A \in \mathbb{R}^{n\times n}, \quad \mathbf{x}^*, \mathbf{b} \in \mathbb{R}^n \tag{23}$$

hence avoiding inner products which often become a performance bottleneck in parallel hardware. The method requires enough knowledge about the spectrum of the preconditioned operator $M^{-1}A$ that an interval $[a, b]$ enveloping all the eigenvalues can be defined. More precisely, given the preconditioned system

$$M^{-1}A\mathbf{x}^* = M^{-1}b \tag{24}$$

with $M \in R^{n\times n}$ and its spectral factorisation

$$M^{-1}A = Q\Lambda Q^T \tag{25}$$

where

$$QQ^T = I, \quad \Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n), \quad \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \tag{26}$$

the method requires knowledge about the interval $[a, b]$ such that

$$a \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \leq b \tag{27}$$

With this information, the method defines a family of Chebyshev polynomials p_i for the interval $[a, b]$, scaled so that $p_i(0) = 1$. Based on the three-term recurrence relation for p_i and an initial guess $\mathbf{x_0}$, the method computes a sequence of approximations $\mathbf{x}_i, i \geq 1$ that converge to $\mathbf{x}^*$. A detailed description is given in Manteuffel (1977).

The following result will be useful in later discussion: let $r_i = b - Ax_i, i \geq 0$ be the residual vectors, and express r_0 in the basis of the eigenvectors $q_j \in R^n, Q = [q_1, \ldots, q_n]$

$$r_0 = \sum_{j=1}^{n} \alpha_j q_j \tag{28}$$

Then, it can be shown (e.g. see Calvetti et al. (1994)) that

$$r_i = \sum_{j=1}^{n} \alpha_j p_i(\lambda_j) q_j \tag{29}$$

Based on a survey of the literature, it appears that the CI method has never been successfully applied to the solution of the mono/bidomain equations using modern parallel hardware. Two possible reasons for this are: i) the intrinsic difficulty of estimating $[a, b]$ for large matrices, and ii) the fact that in practice CG may converge to the solution more quickly than CI due to its superlinear convergence properties, which cannot be expected from CI templates. This reduction in the total number of iterations required may compensate for the inferior scalability, yielding an overall shorter execution time. Also note that, in realistic 3D simulations, A is on the order of millions of degrees of freedom and therefore explicitly computing the eigenfactorisation in equation (25) is unfeasible. Iterative methods such as the Lanczos algorithm can compute a subset of Λ at a much more reduced cost.

In Calvetti and Reichel (1996) a hybrid iterative method that alternates CG and Richardson iterations for the solution of symmetric positive definite linear systems is proposed. The method starts by performing m CG iterations and, thanks to the well-known relationship between CG and the Lanczos algorithm (e.g. see see Golub and Van Loan (1996)), also computes a tridiagonal matrix $T_m \in R^{m\times m}$ that can be used to approximate Λ. The eigenvalues of T_m, $\{\hat{\lambda}_i^k\}$, satisfy

$$\lambda_1 \leq \hat{\lambda}_1^k \leq \hat{\lambda}_2^k \leq \ldots \leq \hat{\lambda}_m^k \leq \lambda_n \tag{30}$$

(with associated eigenvectors $\hat{q}_i$) where k is the number of times that CG has been used to approximate the spectrum. After that, the interval $[a^k = \hat{\lambda}_1^k, b^k = \hat{\lambda}_m^k]$ is used to initialise the Richardson iteration and iterate until the solution is found or until slow convergence is detected. A degradation in convergence rate after, say, p iterations indicates that the residual r_{m+p} is orthogonal to all, or almost all, the eigenvectors $\hat{q}_i$, i.e. $p_{m+p}(\hat{\lambda}_i) = 0$ in equation (29). At this point, m new CG iterations will be performed in order to generate a new interval $[a^{k+1}, b^{k+1}]$ such that

$$\lambda_1 \leq a^{k+1} \leq a^k, \qquad b^k \leq b^{k+1} \leq \lambda_n \tag{31}$$

The Richardson iteration is then resumed with the new interval. The hybrid scheme alternates between CG and Richardson iterations as described until a sufficiently converged solution of equation (23) is found. Note that the definition of m has been intentionally left out from the discussion—a discussion and an empirical evaluation of the optimal number can be found in Calvetti and Reichel (1996).

In the current work, this idea is extended to the solution of linear systems with multiple—but not simultaneously available—right-hand sides, as is the case in the FEM solution of the mono/bidomain equations and other systems of PDEs including time derivatives. Interleaving complete CG and CI solves is proposed, rather than interleaving CG and Richardson iterations within the same solve. This is motivated by the fact that efficient parallel implementations of CG and CI solvers are readily available and can be used as black boxes. In this case, CG is used both for solving the first timestep (and possibly later ones) and for computing (via the aforementioned Lanczos connection) the interval $[a, b]$ that is used to initialise subsequent CI solves. The width of the interval—and therefore the proportion of the spectrum contained in it—depends on the number of CG iterations performed. Unlike the algorithm in Calvetti and Reichel (1996), a fixed number of CG iterations is not specified here. Rather, iterations continue until $||r_i|| < r_{\mathrm{tol}}$, with r_{tol} being the tolerance specified for the solution of the FEM linear systems. Table 3 tabulates $\hat{\lambda}_1^l$, $\hat{\lambda}_m^l$ and $||r_l||$ after l CG iterations for a bidomain simulation with a realistic 3D geometry.

The choice of $[a, b]$, driven in this case by the choice of r_{tol} in the first CG solve, determines the convergence

Table 3. Solution residual $||r_l||$ after l CG iterations, smallest eigenvalue (λ_1^l), and largest eigenvalue (λ_m^l) simultaneously computed with the Lanczos algorithm. First simulation timestep.

l	$\hat{\lambda}_1^l$	$\hat{\lambda}_m^l$	$\|\|r_l\|\|$
1	0.696	0.696	141.10
2	0.525	0.965	39.13
3	0.481	1.048	12.49
4	0.451	1.181	4.79
5	0.425	1.336	1.62
10	0.380	1.510	0.0257
15	0.3667	2.203	0.000124
16	0.3662	2.237	0.0000529
20	0.117	2.243	0.0000168
25	0.0924	2.24485	0.00000206
30	0.0878	2.24487	0.000000273
40	0.085868	2.59049	0.00000000389

Table 4. Number of iterations taken by CI initialised with $[a = \lambda_1^l, b = \lambda_m^l]$ for a range of values of l (from Table 3). Second simulation timestep.

l	number of iterations
3	30
5	25
16	21
19	28
22	40
54	45
58	101

rate of subsequent CI solves. Underestimating Λ, by not including eigenpairs $(\hat{\lambda}_j, \hat{q}_j)$ associated with large values of α_j in equation (28), can lead to convergence stagnation. Overestimating it can lead to slow convergence due to the need to compute polynomials of higher degree. Table 4 shows the number of iterations taken by the first CI solve initialised with $[a = \hat{\lambda}_1^l, b = \hat{\lambda}_m^l]$ for a range of values of l.

It can be observed that the minimum number of iterations is achieved with $a = \hat{\lambda}_1^{16}, b = \hat{\lambda}_m^{16}$. One may think that the optimal choice of parameters a and b is somehow arbitrary or at best requires an expensive tuning process. However, it can be seen in Table 3 that after 16 iterations $||r|| < 10^{-4}$, which corresponds to the tolerance used in the Chebyshev linear solver. It can, therefore, be concluded that the optimal choice of parameters a and b corresponds to the values of $\hat{\lambda}_1^l, \hat{\lambda}_m^l$ computed by CG when configured to solve to the same accuracy as the Chebyshev linear solver.

Finally, it cannot be expected that the choice of parameters a and b will remain optimal throughout the whole simulation, especially if sudden changes to $||r_i||$ happen. Therefore, the interval $[a, b]$ needs to be reevaluated every certain number of timesteps. In Calvetti and Reichel (1996), this is done by monitoring $||r_i||$ evolution. Here, however, the intention is to minimise the number of $||\cdot||$ operations since they require global reduction operations

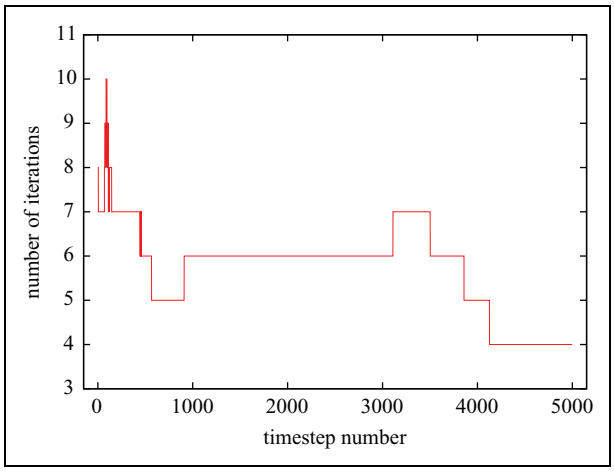

Figure 4. Number of iterations taken by CI ($r_{tol} = 10^{-6}$) at each PDE timestep for a bidomain simulation with the University of California, San Diego (UCSD) rabbit model (Vetter and McCulloch, 1998.)

which compromise parallel scalability. The following subsection describes the approach taken to achieve this.

4.3.2 l^2-Norm reduction. Even in the cases when an inner-product free iterative method (such as CI) is used, one global reduction per iteration is required when the stop criteria of the method is based on the evolution of the l^2-norm of the residual. The current work proposes an algorithm that overcomes this bottleneck by interleaving: a) solves with default stop criteria and b) solves where a fixed number of iterations is performed—therefore avoiding l^2-norm computation—with a $1 : s$ ratio (i.e. s consecutive CI solves with fixed number of iterations per each CI solve with default stop criteria). This approach is based on the evidence that, for accurate enough values of a, b, the number of iterations required by successive CI solves varies smoothly as shown in Figure 4. Note that this may require periodic reevaluations of $[a, b]$ as already mentioned.

Sudden changes in iteration count are triggered by events such as stimuli application (both user-defined and coming from self-stimulating cells), the presence in the domain of polarisation/repolarisation wavefronts and, in general, any event inducing sudden changes in $||r_i||$. For some of these events, the precise moment when they occur is known at the beginning of the simulation. Therefore, it is possible to reset the $1 : s$ ratio so that solution is not under-approximated. For others, such as the beginning of a repolarisation wavefront, there is no easy way of anticipating them and the solution will be under-approximated until the next convergence-based solve.

4.3.3 A hybrid CG–Chebyshev method. Algorithm 4.3.3 summarises the techniques presented in Sections 4.3.1 and 4.3.2 for the reduction of inner products and l^2-norms required by standard Krylov subspace methods. Note the two different uses of the CI linear solver: in line 7, the algorithm iterates until $||r_i|| < r_{tol}$ and records the number of iterations

Algorithm 3: Hybrid CG–Chebyshev method for symmetric systems with multiple, not simultaneously available, right-hand sides.

Input : System matrix $A \in R^{n \times n}$, initial guess $\mathbf{x}_{-1} \in R^n$, tolerance r_{tol}, number of timesteps n_t, 1:s ratio.
Output: $\{\mathbf{x}_i\} : A\mathbf{x}_i = \mathbf{b}_i$, $0 \leq i < n_t$

```
1   for i = 0 to n_t − 1 do
2     b_i := assemble_rhs(i, x_{i−1});
3     switch i (mod s) do
4       case 0
5         [x_i, a, b] := CG(A, b_i, r_tol);
6       case 1
7         [x_i, num_its] := Chebyshev1(A, b_i, a, b, r_tol);
8       otherwise
9         x_i := Chebyshev2(A, b_i, a, b, num_its);
10      end
11    end
12  end
```

num_its taken, whereas in line 9 a fixed number of iterations num_its is performed.

In order to evaluate the error introduced by the use of a fixed number of iterations, a benchmark consisting of 500 ms of bidomain activity following an apical stimulus on the UCSD rabbit model (Vetter and McCulloch, 1998) was designed. Figure 5(a) plots the infinity norm of the error against time for a range of values of $1 : s$ when compared with a reference solution computed with CG. It can be seen that for the largest portion of the simulation, the error stays within the linear solve tolerance, $r_{\text{tol}} = 10^{-6}$ in this case. Only during periods when depolarisation and repolarisation wavefronts travel through the domain is the error induced is larger than r_{tol}, mainly within one order of magnitude. Once these events have ended the error goes below r_{tol} again, which indicates that error is not accumulated in time. Figure 5(b) shows that there is not a huge variation in the error induced for the different values of s studied.

4.3.4 Preconditioning. Bidomain preconditioning has become an active field of research over the past years (dos Santos et al., 2004; Pennacchio and Simoncini, 2009; Plank et al., 2007; Pavarino and Scacchi, 2008). Algebraic multi-grid (AMG) preconditioning has been successfully applied to the efficient solution of bidomain linear systems. However AMG does not exhibit mesh-independent convergence properties, i.e. the convergence rate of an AMG-preconditioned linear solver will deteriorate with refinement of the computational domain. In order to address this issue, mesh-independent bidomain preconditioners have been proposed (Pennacchio and Simoncini, 2009; Bernabeu et al., 2010a). In Bernabeu et al. (2010a), we propose a preconditioner that exploits the block structure of the matrix in order to reduce computation time and achieve mesh-independent convergence. In a more recent publication (Bernabeu and Kay, 2011), we present scalable parallel implementations of the techniques described in Bernabeu et al. (2010a) and identify a dependency of the most efficient bidomain preconditioner on the coefficient between the PDE solution timestep and the square of the spatial discretisation ($\Delta t/h^2$).

4.4 Output

Output of the results of a massively parallel simulation is not as straightforward as in the sequential case or for a small number of processes. In these cases periodically writing the values of the variables of interest to a single file is likely to be relatively cheap compared to the cost of computing the solution. However, this strategy does not scale as the number of processes increases: each process in turn has to open the file, write its data and then close the file—and, hence, the time required to do this increases for a larger number of processes. This problem can be compounded if the computational resource being used has a distributed file system (e.g. Lustre or NFS), requiring additional time to be spent communicating the data to the location at which it is to be stored. So, small tasks (such as updating a file that tracks the progress of the simulation) that can be completed in a very small time by each process, and barely show up in profiling in parallel, even when using 10 s of processes can become very severe bottlenecks at larger scales as processes compete to write to disk. Hence, where possible, global information should be calculated and written to file by a single process (without communication) even though this may lead to other processes remaining idle and may result in less readable code.

However, for output of the simulation's variables (which are distributed over all processes), bandwidth limitations mean that it is not efficient to concentrate data on a single process for output. Hence, Chaste was designed to make use of the HDF5 parallel I/O library (http://www.hdfgroup.org/HDF5). Based on MPI-IO, this provides a set of routines that efficiently write complex data structures to disk in parallel. The results of this have proved to be mixed: while the use of HDF5 does seem to prevent the time required to perform I/O from increasing significantly as the number of processes increases, the code does not appear to scale in parallel when used within Chaste and timings (on any number of processes) are extremely inconsistent.

5 Results

Following the initial performance evaluation presented in Section 3.2 and the description of the proposed improvements in Section 4, we now turn our attention to the evaluation of those improvements.

5.1 Scalability improvements

5.1.1 Mesh load. Figure 6 shows the time taken by the mesh-load stage against the number of cores for five different versions of the code. In the original version, the mesh-load time increased rather than decreased with the number

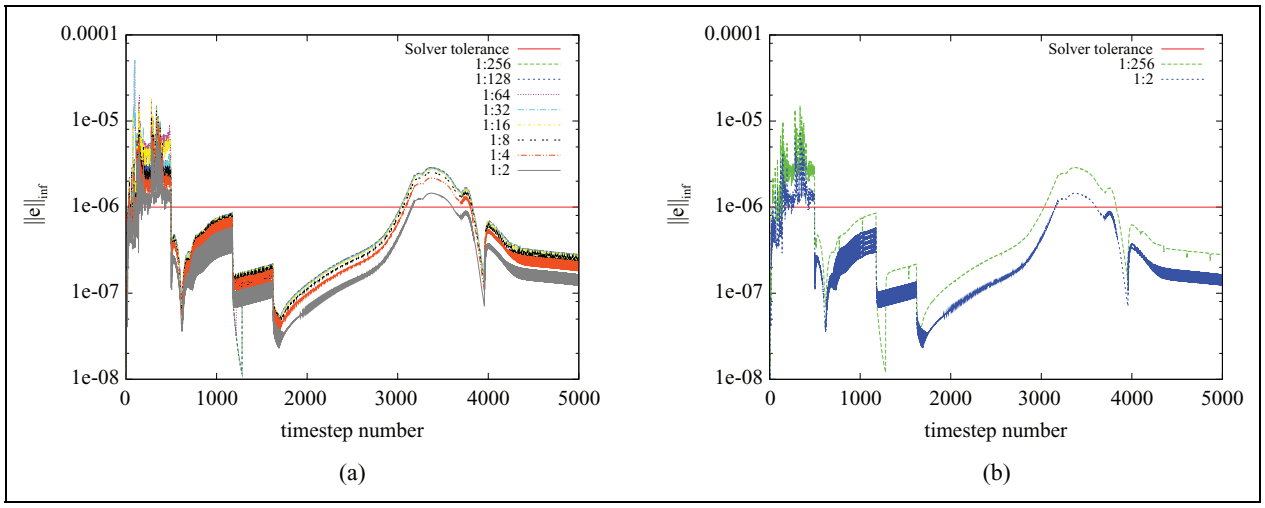

Figure 5. Error introduced by several configurations of the ratio $1 : s$ in the hybrid CG–CI algorithm compared with a reference solution computed with CG. Panel (a) shows the error for a values of s between 2 and 256. Panel (b) focuses on the configurations presenting the largest and smallest error.

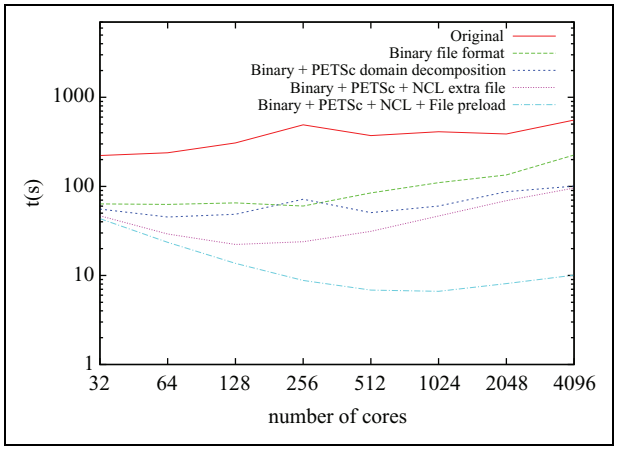

Figure 6. Mesh load time for different numbers of cores.

of processors. There were two reasons for this: i) mesh files were read in their entirety by all the processors and ii) a sequential algorithm was used to compute the partitions. In this scenario, mesh-load time will, at best, remain constant. However, the fact that all the processors were concurrently accessing the entire set of files produced disk access contention due to the large number of read operations issued at large core counts. Further, in the case of ii) the asymptotic cost of the algorithm is directly proportional to the number of partitions to be generated (i.e. to the number of processors running the simulation).

Following the introduction of a binary file format, mesh-read times were reduced by a factor of 4.21 on average, mainly due the fact that data representation was more compact and direct access could be performed on the node file. However, scalability still remained an issue. Next, the addition of PETSc-based partitioning (instead of raw calls to METIS) allowed for the partitioning step to be performed in parallel, greatly reducing the execution time for large core counts: a factor of 2.23 for $p = 4096$. Nevertheless, it was still necessary for the element file to be read in its entirety by all the processors, introducing a sequential portion of code that dominated the total execution time. Next, the introduction of .ncl files allowed for parallelisation of the element file read. It can be seen from Figure 6 that the total mesh-read time scales well up to 128 processors when using binary file format, PETSc-based partitioning and .ncl files. However, for larger core counts the mesh-read time increases again due to disk access contention.

In the final version of the code (labelled "Binary + PETSc + NCL + File preload" in Figure 6) a warm-up run is performed before the actual simulation being timed in order to validate the hypothesis about the file access contention. The rationale behind this is that the files will be cached by the distributed file system before the time the second run starts, therefore reducing latency and hiding disk access contention. When compared with the previous version, it can be seen that the mesh-load time stays fairly constant for $p = 32$ (46 s vs 42 s) but is greatly reduced for larger core counts: scalability is very good up to 1024 cores, achieving 92% and 80% scalability for 64 and 128 cores respectively. It can be seen that the total read time saturates at around 6 s for $p = 1024$ and slightly increases for $p > 1024$. This is due to some portions of this stage not being parallelised and to the fact that the problem is of fixed size and, therefore, the proportion of halo nodes and elements increases with p, making the use of large numbers of cores ineffective if the ratio of number-of-nodes to number-of-processors becomes too small. Hence, if the mesh size in the benchmark was increased good scalability up to $p = 4096$ and beyond would be expected.

5.1.2 Right-hand side assembly. The time taken by the RHS assembly stage is plotted against number of cores for three different versions of the code in Figure 7. In the original

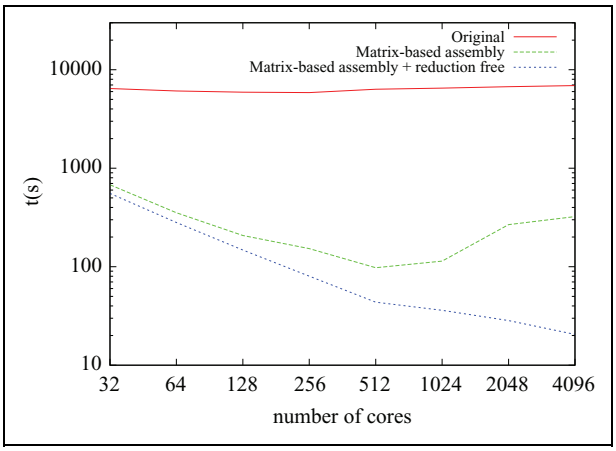

Figure 7. Right-hand side assembly time for different numbers of cores.

implementation, it can be seen that execution time remained constant no matter how many cores were used. Recall that this problem had been identified in Figure 2. The poor scaling resulted from the distributed vector containing the solution at the previous timestep ($\mathbf{V}^m_{(1)}$ and $\mathbf{V}^m$ in the bidomain and monodomain formulation, respectively) having to be explicitly gathered on each processor involved in the simulation at the beginning of the assembly phase. During earlier stages of development, this design had been proved to scale well on low core counts, making it a valid alternative to the more complex task of programming an explicit halo-value exchange across subdomain borders for low core counts. However, the results presented here demonstrate that it did not scale beyond a few 10s of processes.

Even for low core counts, the original implementation turned out to be one order of magnitude slower than later improvements. The first of these improvements (plotted as "Matrix-based assembly" in Figure 7) comes from the elimination of global gather operations and from the switch from element-wise assembly to matrix-based assembly described in Section 4.1.2. It can be seen that the solution scales well for up to 512 cores; however for larger core counts, the execution time increases. Further profiling showed that this was due to the increasing time spent performing parallel reductions during the assembly of $\mathbf{C}^m$ in equation (20) in the monodomain case. This scalability issue is also identified in Tallent et al. (2009). In order to reduce the number of global reduction operations required, the optimisation described in Section 4.1.3 was used. Note that this optimisation is only possible due to the use of the permutation described at the beginning of Section 4.1, which ensures that all the entries of $\mathbf{C}^m$ are generated locally.

5.1.3 System solution. At the beginning of this work, CG was Chaste's default linear solver. However, when the code was ported to HECToR other available linear solvers were evaluated. It was found that PETSc's implementation of the SYMMLQ algorithm was competitive. Finally, the hybrid CG–Chebyshev algorithm presented in Section 4.3 was also successfully ported.

Table 5. Linear solve time (s) for different numbers of cores and the three linear solvers considered.

Number of processors	CG	SYMMLQ	Hybrid CG–Chebyshev 1 : 16
64	619	666	535
128	313	337	262
256	145	159	123
512	78.8	80	65.8
1024	48.4	48	35.9
2048	30.8	29.4	22.4
4096	28.6	25.5	18.8

Table 5 shows the time taken by the system solution stage against the number of cores for the three linear solvers mentioned. It can be seen that the SYMMLQ algorithm is faster than CG for 1024 or more cores. This was unexpected since SYMMLQ needs to perform extra work to overcome the potential indefiniteness of the linear system. The magnitude of the difference in time suggests that this speedup is most likely to be due to implementation characteristics. Further profiling needs to be performed in order to validate this hypothesis. The hybrid CG–Chebyshev method is faster than any of the previous methods for all the core counts considered. For $p = 64$ the method is around 15% faster than CG. This shows that for low core counts the gains from the reduction in the number of inner products and $||\cdot||$ operations are modest and mainly due to fewer arithmetic operations being performed. For large core counts (e.g. $p = 4096$), the hybrid method is around 52% faster than CG and 35% faster than SYMMLQ. This demonstrates that as the number of cores increases the proportion of time spent performing inner products and $||\cdot||$ operations increases, mainly due to the fact that the time required to perform a global reduction is a function of p. Figure 8 evaluates this time reduction in terms of speedup improvement.

5.2 Final time breakdown

Following optimisation, the proportion of time spent by the benchmark simulation in each of the stages described in Section 3.1 is shown, for a range of processor counts, in Figure 9. This figure is exactly equivalent to the analysis before optimisation shown in Figure 2.

Firstly, note that the time initially reported as 'Missing' has been successfully removed. Secondly, the 'RHS assembly' stage has been greatly improved: it now takes no more than 8.6% of the total execution time (against 25–65% in the original time breakdown) and also scales almost linearly up to 1024 cores, degrading only slightly for 2048 and 4096 cores. Thirdly, the proportion of time spent reading in the mesh has been greatly improved, being almost unnoticeable for 32–1024 cores. However, the increasing proportion of

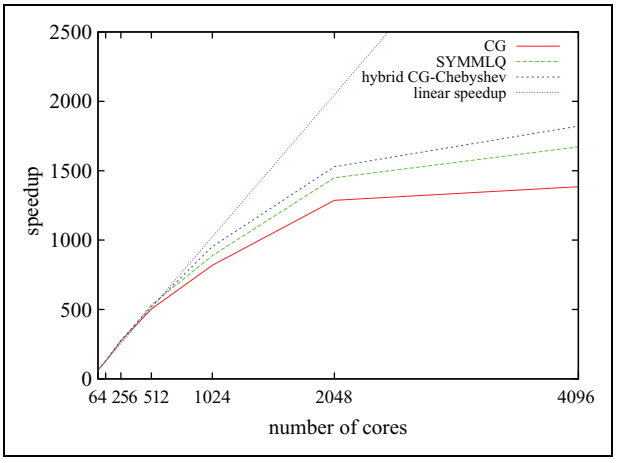

Figure 8. Parallel speedup for the three linear solvers considered and an increasing number of processors.

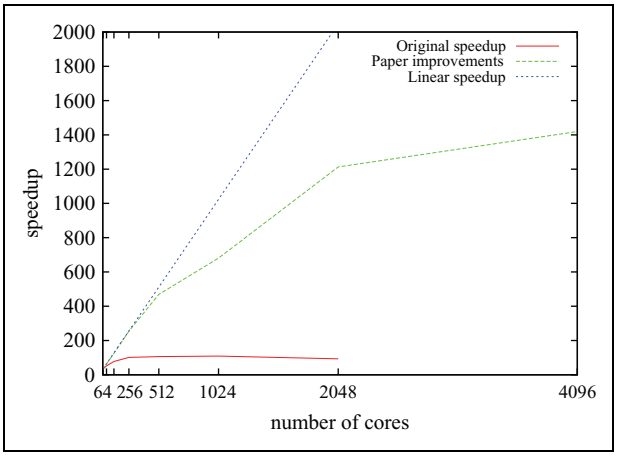

Figure 10. Chaste's scalability for a typical bidomain simulation before (solid red line) and after (broken green line) the improvements presented in Section 4.

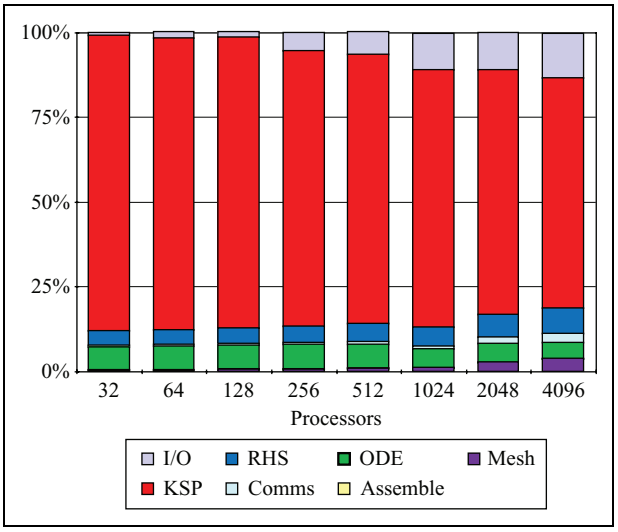

Figure 9. Final time breakdown after the improvements presented in this paper.

time spent for 2048 and 4096 cores highlights the fact that the absolute time spent in this stage is actually higher than for 1024, indicating that the execution is probably suffering from file access contention. Nevertheless, the 'Mesh' time for $p = 4096$ has been reduced by a factor of 60 when compared with the initial execution. Next, it can seen that linear system solution ('KSP') and 'ODE' scale well and 'KSP' now takes the greatest proportion of total execution time, as one would expect from a bidomain simulation with a fairly simple ionic model, like the Luo–Rudy model (Luo and Rudy, 1991).

Finally, two operations that were almost unnoticeable in the initial time breakdown increase their presence at large core counts: 'I/O' and 'Comms' (which cover writing out the simulation results and performing certain synchronisations and distributed error checks). There are two reasons for this: i) both operations involve either synchronisation or access to resources with a low degree of replication (such as Lustre I/O nodes), and ii) the benchmark used consists of around 8 million degrees of freedom, at $p = 4096$—so the number of degrees of freedom per core is only around 2000, which may not be enough to ensure full utilisation of all the functional units available. This situation is common when performing strong scaling analysis (i.e. quantifying how the solution time varies with number of processors for a fixed problem size). In fact i) is also a direct consequence of ii), since operations like 'I/O', 'Comms' or 'Mesh', and in general any task that is intrinsically sequential or with an asymptotic cost greater than $O(n/p)$, will take an increasing proportion of total time as the number of degrees of freedom per processor decreases.

6 Discussion and conclusions

In this paper, effective parallelisation strategies for Chaste's bidomain solver have been described. The code was initially ported to HECToR (the UK's high-end computational resource), where simulations could be run using two orders of magnitude more processors than ever before. Initial profiling highlighted a number of scalability issues only identifiable at a large scale (e.g. mesh read, RHS assembly, linear system solution). Section 4 describes the techniques implemented to address these issues, including novel computational and numerical techniques as well as techniques at the interface between them. Of particular interest is the novel hybrid CG–Chebyshev linear solver presented in Section 4.3. Although the hybrid concept was first introduced in the late 1980s, a literature survey indicates that the Chebyshev Iteration method has not been successfully applied to the reduction of linear system solution time using modern parallel hardware.

Figure 10 compares parallel speedup before and after the work presented in Section 4. It can be seem that in the code described in Section 3.2 (solid red line), speedup saturated at around 100 no matter the number of processors used. The optimisations allowed for speedups of over 1400 (compared

Table 6. Ratio between the time taken to perform a simulation and the amount of time simulated.

Solver	Core count	Mesh size	Real–time ratio	Architecture
Chaste bidomain	4096	3.7 M nodes	210	HECToR Phase 2a
Chaste monodomain	4224	3.7 M nodes	45	HECToR Phase 2b

to sequential execution) and resulted in optimal or near-optimal scalability for up to 512 cores. They yield a combined improvement (compared to the initial code) by a factor of over 140 in the fastest possible execution time for the benchmark, meaning that Chaste's bidomain solver is now two orders of magnitude faster than at the beginning of the work described here. Further optimisations relating to sequential performance (with no impact on parallel scaling) have also been described in Southern et al. (2011). Table 6 summarises the real–time ratios achieved as a result of all the optimisations. The values represent an improvement with respect to those in Table 1. One limitation of our work is that scalability is only examined under sinus rhythm propagation and not during reentry. During reentry multiple wavefronts coexist and interact in the domain. How this affects load balancing and scalability needs to be determined.

It can, therefore, be concluded that this work has brought Chaste to the level of parallel performance necessary to run simulation studies with state-of-the-art whole-organ geometrical models (including human ventricular models). As a result of the improvements described here, it has been possible to conduct further studies (Bernabeu et al., 2008, 2009; Zemzemi et al., 2011) that would otherwise have been impossible due to the prohibitive computational expense required. Furthermore, given that Chaste is one of the first open source software platforms for bidomain simulation, the improvements are likely to have an immediate impact in the community, by making large-scale computational cardiac electrophysiology simulation more accessible, thus avoiding the overhead of developing multiple, often repetitive, in-house codes.

Acknowledgements

The authors would like to thank Dr Martin Bishop for providing some of the geometries used in this study, HECToR's helpdesk, and the Chaste development team (http://www.cs.ox.ac.uk/chaste/theteam.html).

Funding

The work of Miguel O Bernabeu, James Southern, Nicholas Wilson, Jonathan Cooper and Joe Pitt-Francis, part of the preDiCT project, was supported (entirely or partially) by a grant from the European Commission Directorate General for the Information Society (grant number 224381). The work of Jonathan Cooper was also partially supported by the European Commission under the VPH NoE project (grant number 223920).

Note

1. According to www.top500.org.

References

Amdahl GM (1967) Validity of the single processor approach to achieving large scale computing capabilities. In: *spring joint computer conference (AFIPS '67)*, Atlantic City, USA, 18–20 th April 1967, vol. 30, pp. 483. New York: ACM Press.

Balay S, Brown J, Buschelman K, Eijkhout V, Gropp WD, Kaushik D, et al. (2010) PETSc users manual. Technical Report ANL-95/11, Argonne National Laboratory, USA.

Barrett R, Berry M, Chan TF, Demmel J, Donato J, Dongarra J, et al. (1994) *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia: SIAM.

Bekas C, Curioni A, Arbenz P, Flaig C, Van Lenthe GH, Müller R, et al. (2010) Extreme scalability challenges in micro-finite element simulations of human bone. *Concurrency and Computation: Practice and Experience* 22(16): 2282–2296.

Bernabeu MO, Bishop MJ, Pitt-Francis J, Gavaghan D, Grau V and Rodriguez B (2008) High performance computer simulations for the study of biological function in 3D heart models incorporating fibre orientation and realistic geometry at para-cellular resolution. In: *computers in cardiology*, Bologna, Italy, 14–17 September 2008, vol. 35, pp. 721–724. Piscataway: IEEE Press.

Bernabeu MO, Corrias A, Pitt-Francis J, Rodriguez B, Bethwaite B, Enticott C, et al. (2009) Grid computing simulations of ion channel block effects on the ECG using 3D anatomically-based models. In: *computers in cardiology*, Park City, USA, 13–16 April 2009, pp. 213–216. Piscataway: IEEE Press.

Bernabeu MO and Kay D (2011) Scalable parallel preconditioners for an open source cardiac electrophysiology simulation package. *Procedia Computer Science* 4: 821–830.

Bernabeu MO, Pathmanathan P, Pitt-Francis J and Kay D (2010a) Stimulus protocol determines the most computationally efficient preconditioner for the bidomain equations. *IEEE Transactions on Biomedical Engineering* 57(12): 2806–2815.

Bernabeu MO, Wallman M and Rodriguez B (2010 b) Shock-induced arrhythmogenesis in the human heart: A computational modelling study. In: *IEEE annual international conference of the engineering in medicine and biology society (EMBC)*, Buenos Aires, Argentina, 31 August–4 September 2010, pp. 760–763. Piscataway: IEEE Press.

Bernus O, Wilders R, Zemlin CW, Verschelde H and Panfilov AV (2002) A computationally efficient electrophysiological model of human ventricular cells. *American Journal of Physiology: Heart and Circulatory Physiology* 282(6): 2296–2308.

Bishop MJ and Plank G (2011) Representing cardiac bidomain bath-loading effects by an augmented monodomain approach: Application to complex ventricular models. *IEEE Transactions on Biomedical Engineering* 58(4): 1066–1075.

Bishop MJ, Plank G, Burton RA, Schneider JE, Gavaghan DJ, Grau V et al. (2009) Development of an anatomically-

detailed MRI-derived rabbit ventricular model and assessment of its impact on simulation of electrophysiological function. *American Journal of Physiology: Heart and Circulatory Physiology* 298(2): 699–718.

Bordas R, Carpentieri B, Fotia G, Maggio F, Nobes R, Pitt-Francis J et al. (2009) Simulation of cardiac electrophysiology on next-generation high-performance computers. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367(1895): 1951–1969.

Burton R, Plank G, Schneider J, Grau V and Ahammer H, Keeling S et al. (2006) 3-D models of individual cardiac histoanatomy: tools and challenges. *Annals of the New York Academy of Sciences* 1380: 301–319.

Cai X and Lines GT (2002) Enabling numerical and software technologies for studying the electrical activity in human heart. In: *6th international conference on applied parallel computing advanced scientific computing (PARA '02)*, Espoo, Finland, 15–18 June 2002, pp. 3–17. London: Springer-Verlag.

Calvetti D, Golub G and Reichel L (1994) An adaptive Chebyshev iterative method for nonsymmetric linear systems based on modified moments. *Numerische Mathematik* 67: 21–40.

Calvetti D and Reichel L (1996) A hybrid iterative method for symmetric positive definite linear systems. *Numerical Algorithms* 11: 79–98.

Clayton R, Bernus O, Cherry E, Dierckx H, Fenton F, Mirabella L, et al. (2011) Models of cardiac tissue electrophysiology: Progress, challenges and open questions. *Progress in Biophysics and Molecular Biology* 104(1–3): 22–48.

Colli-Franzone P and Pavarino LF (2004) A parallel solver for reaction-diffusion systems in computational electrocardiology. *Mathematical Models and Methods in Applied Sciences* 14(6): 883–912.

Cooper J (2009) *Automatic validation and optimisation of biological models*. PhD Thesis, University of Oxford, UK.

Devine KD, Boman EG, Heaphy RT, Hendrickson BA, Teresco JD, Faik J, et al. (2005) New challenges in dynamic load balancing. *Applied Numerical Mathematics* 52(2–3): 133–152.

Devine KD, Boman EG and Karypis G (2006) Partitioning and load balancing for emerging parallel applications and architectures. In: Heroux M, Raghavan A and Simon H (eds) *Frontiers of Scientific Computing*. Philadelphia: SIAM.

dos Santos R, Plank G, Bauer S and Vigmond E (2004) Parallel multigrid preconditioner for the cardiac bidomain model. *IEEE Transactions on Biomedical Engineering* 51(11): 1960–1968.

Fishler MG and Thakor NV (1991) A massively parallel computer model of propagation through a two-dimensional cardiac syncytium. *Pacing and Clinical Electrophysiology* 14(11): 1694–1699.

FitzHugh R (1961) Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal* 1(6): 445–466.

Golub GH and Van Loan CF (1996) *Matrix Computations*. Baltimore: Johns Hopkins University Press.

Gutknecht MH and Röllin S (2002) The Chebyshev iteration revisited. *Journal of Parallel Computing* 28(2): 263–283.

Iyer V, Hajjar R and Armoundas A (2007) Mechanisms of abnormal calcium homeostasis in mutations responsible for catecholaminergic polymorphic ventricular tachycardia. *Circulation Research* 100(2): e22–31.

Keener J and Sneyd J (1998) *Mathematical Physiology*. New York: Springer.

Keener JP and Bogar K (1998) A numerical method for the solution of the bidomain equations in cardiac tissue. *Chaos* 8(1): 234.

Linge S, Sundnes J, Hanslien M, Lines G and Tveito A (2009) Numerical solution of the bidomain equations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367(1895): 1931–1950.

Luo CH and Rudy Y (1991) A model of the ventricular cardiac action potential - depolarisation, repolarisation and their interaction. *Circulation Research* 68: 1501–1526.

Manteuffel TA (1977) The Tchebychev iteration for nonsymmetric linear systems. *Numerische Mathematik* 28: 307–327.

Murillo M and Cai XC (2004) A fully implicit parallel algorithm for simulating the non-linear electrical activity of the heart. *Numerical Linear Algebra with Applications* 11(2–3): 261–277.

Ng KT, Hutchinson SA and Gao S (1995) Numerical analysis of electrical defibrillation: The parallel approach. *Journal of Electrocardiology* 28(1): 15–20.

Niederer S, Mitchell L, Smith N and Plank G (2011) Simulating human cardiac electrophysiology on clinical time-scales. *Frontiers in Physiology* 2: 14.

Pathmanathan P, Bernabeu MO, Bordas R, Cooper J, Garny A, Pitt-Francis JM, et al. (2010) A numerical guide to the solution of the bidomain equations of cardiac electrophysiology. *Progress in Biophysics and Molecular Biology* 102(2–3): 136–155.

Pavarino LF and Scacchi S (2008) Multilevel additive schwarz preconditioners for the bidomain reaction-diffusion system. *SIAM Journal on Scientific Computing* 31(1): 420–443.

Pennacchio M and Simoncini V (2009) Algebraic multigrid preconditioners for the bidomain reaction–diffusion system. *Applied Numerical Mathematics* 59(12): 3033–3050.

Piggott MD, Gorman GJ, Pain CC, Allison PA, Candy AS, Martin BT, et al. (2008) A new computational framework for multi-scale ocean modelling based on adapting unstructured meshes. *International Journal for Numerical Methods in Fluids* 56(8): 1003–1015.

Pitt-Francis J, Pathmanathan P, Bernabeu MO, Bordas R, Cooper J, Fletcher AG, et al. (2009) Chaste: A test-driven approach to software development for biological modelling. *Computer Physics Communications* 180(12): 2452–2471.

Plank G, Liebmann M, dos Santos RW, Vigmond EJ and Haase G (2007) Algebraic multigrid preconditioner for the cardiac bidomain model. *IEEE Transactions on Biomedical Engineering* 54: 585–596.

Pollard A and Barr R (1991) Computer simulations of activation in an anatomically based model of the human ventricular conduction system. *IEEE Transactions on Biomedical Engineering* 38(10): 982–996.

Pope B, Fitch B, Pitman M, Rice J and Reumann M (2011) Performance of hybrid programming models for multiscale

cardiac simulations: Preparing for petascale computation. *IEEE Transactions on Biomedical Engineering* 58(10): 2965–2969.

Porras D, Rogers J, Smith W and Pollard A (2000) Distributed computing for membrane-based modeling of action potential propagation. *IEEE Transactions on Biomedical Engineering* 47(8): 1051–1057.

Potse M, Dubé B, Richer J, Vinet A and Gulrajani RM (2006) A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart. *IEEE Transactions on Biomedical Engineering* 53(12): 2425–2435.

Quan W, Evans S and Hastings H (1998) Efficient integration of a realistic two-dimensional cardiac tissue model by domain decomposition. *IEEE Transactions on Biomedical Engineering* 45(3): 372–385.

Reumann M, Fitch B, Rayshubskiy A, Keller D, Seemann G, Dossel O, et al. (2009) Strong scaling and speedup to 16,384 processors in cardiac electro-mechanical simulations. In: *IEEE annual international conference of the engineering in medicine and biology society (EMBC)*, Minneapolis, USA, 2–6 September 2009, pp. 2795–2798. Piscataway: IEEE Press.

Sahni O, Zhou M, Shephard MS and Jansen KE (2009) Scalable implicit finite element solver for massively parallel processing with demonstration to 160 k cores. In: *22nd conference on high performance computing networking, storage and analysis (SC '09)*, Portland, USA, 14–20 November 2009, article no. 68. New York: ACM Press.

Saleheen H, Claessen P and Ng K (1997) Three-dimensional finite-difference bidomain modeling of homogeneous cardiac tissue on a data-parallel computer. *IEEE Transactions on Biomedical Engineering* 44(2): 200–204.

Schloegel K, Karypis G and Kumar V (2002) Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience* 14(3): 219–240.

Shadid J, Pawlowski R, Banks J, Chacón L, Lin P and Tuminaro R (2010) Towards a scalable fully-implicit fully-coupled resistive MHD formulation with stabilized FE methods. *Journal of Computational Physics* 229(20): 7649–7671.

Southern J, Bernabeu MO, Cooper J, Wilson N and Pitt-Francis J (2011) Progress towards real-time simulation of cardiac electrophysiology for in silico drug testing. In: *international supercomputing conference (ISC'11)*, Hamburg, Germany, 19–23 June 2011.

Tallent NR, Mellor-Crummey JM, Adhianto L, Fagan MW and Krentel M (2009) Diagnosing performance bottlenecks in emerging petascale applications. In: *22nd conference on high performance computing networking, storage and analysis (SC '09)*, Portland, USA, 14–20 November 2009, article no. 51. New York: ACM Press.

ten Tusscher KH, Hren R and Panfilov AV (2007) Organization of ventricular fibrillation in the human heart. *Circulation Research* 100(12): e87–e101.

ten Tusscher KHWJ, Noble D, Noble PJ and Panfilov AV (2004) A model for human ventricular tissue. *American Journal of Physiology: Heart and Circulatory Physiology* 286(4): 1573–1589.

ten Tusscher KHWJ and Panfilov AV (2006) Alternans and spiral breakup in a human ventricular tissue model. *American Journal of Physiology: Heart and Circulatory Physiology* 291(3): 1088–1100.

Teresco JD, Devine KD and Flaherty JE (2006) Partitioning and dynamic load balancing for the numerical solution of partial differential equations. In: Bruaset AM, Tveito A, Barth TJ, Griebel M, Keyes DE, Nieminen RM, Roose D and Schlick T (eds) *Numerical Solution of Partial Differential Equations on Parallel Computers*. Berlin: Springer, vol. 51, pp. 55–88.

Vázquez M, Arís R, Houzeaux G, Aubry R, Villar P, Garcia-Barnés J, et al. (2011) A massively parallel computational electrophysiology model of the heart. *International Journal for Numerical Methods in Biomedical Engineering* 27(12): 1911–1929.

Vetter FJ and McCulloch AD (1998) Three-dimensional analysis of regional cardiac function: a model of rabbit ventricular anatomy. *Progress in Biophysics and Molecular Biology* 69(2–3): 157–183.

Vigmond EJ, Aguel F and Trayanova NA (2002) Computational techniques for solving the bidomain equations in three dimensions. *IEEE Transactions on Biomedical Engineering* 49(11): 1260–1269.

Vigmond EJ, Hughes M, Plank G and Leon LJ (2003) Computational tools for modeling electrical activity in cardiac tissue. *Journal of Electrocardiology* 36(1): 69–74.

Wathen AJ (1989) An analysis of some element-by-element techniques. *Computer Methods in Applied Mechanics and Engineering* 74(3): 271–287.

Whiteley J (2006) An efficient numerical technique for the solution of the monodomain and bidomain equations. *IEEE Transactions on Biomedical Engineering* 53: 2139–2147.

Winslow RL, Kimball AL, Varghese A and Noble D (1993) Simulating cardiac sinus and atrial network dynamics on the connection machine. *Physica D: Nonlinear Phenomena* 64(1–3): 281–298.

Zemzemi N, Bernabeu MO, Saiz J and Rodriguez B (2011) Simulating drug-induced effects on the heart: From ion channel to body surface electrocardiogram. In: Metaxas DN and Axel L (eds) *Functional Imaging and Modeling of the Heart*. New York: Springer, vol. 6666, pp. 259–266.

Zhou M, Sahni O, Xie T, Shephard M and Jansen K (2012) Unstructured mesh partition improvement for implicit finite element at extreme scale. *The Journal of Supercomputing* 59: 1218–1228.

Author biographies

Miguel O Bernabeu received his DPhil degree in Computational Biology from the University of Oxford, UK, in 2011 and his MSc and BEng degrees from the Universitat Politècnica de València, Spain, in 2005 and 2007. He is currently a 2020 Science Research Fellow at the Centre for Computational Science and CoMPLEX, University College London, UK. He has worked on various UK and EU projects developing HPC software for different aspects of

cardiovascular modelling and simulation. His research interests include software engineering, parallel computing, and numerical methods with applications to cardiac electrophysiology and brain haemodynamics.

James Southern received his BA, MSc, MA and DPhil degrees from the University of Oxford in 2001, 2002, 2005 and 2007, respectively. He is currently a Principal Researcher in the Technical Computing Research Division at Fujitsu Laboratories of Europe, Hayes, UK, where his research interests are in HPC simulations and the development of advanced numerical algorithms for massively parallel application software.

Nicholas Wilson received his MA degree in Chemistry from the University of Oxford and a PhD in Theoretical Chemistry from the University of Birmingham. He is currently a Principal Researcher in the Technical Computing Research Division at Fujitsu Laboratories of Europe. His research interests mainly focus on performance optimization of scientific applications.

Peter Strazdins received his PhD in Computer Science from The Australian National University in 1990. Since then he has been with the Department of Computer Science at the Australian National University, and was closely associated with the ANU–Fujitsu CAP Parallel Computing Project over the years 1990–2002. His research interests include parallel numerical algorithms and libraries, computer architecture and operating systems for high-performance computers, and computer simulation, modelling and performance analysis. He is currently a Senior Lecturer and the Associate Director of Education at the Research School of Computer Science.

Jonathan Cooper is interested in the application of software engineering to computational biology—using technology well to enable better science. In particular, he looks at the use of domain-specific languages to describe mathematical models of biology, and the tools required to make these usable for researchers. His current research interests lie in developing the concept of "functional curation", which supports the specification of experiments through a protocol language, allowing an in silico version of a wet lab experiment to be run on a range of alternative models and the results compared. He is active in all areas of the Chaste project, developed support for the use of CellML, and contributes C++ , Python, and software design expertise, as well as having been largely responsible for developing the project infrastructure.

Joe Pitt-Francis received a first class degree in mathematics from Queen Mary and Westfield College, London, UK, in 1990 and received a PhD degree in mathematics (applied to fluid dynamics) from the University of Oxford in 1994. He is currently a Research Fellow in the Computational Biology Group within the Department of Computer Science. His research interests include software development, plastic deformation, robot path-planning, geometric modelling, parallel computing, and computational physiology.