Published in final edited form as:
IEEE Trans Biomed Eng. 2012 August ; 59(8): 2281–2290. doi:10.1109/TBME.2012.2202661.

Accelerating Cardiac Bidomain Simulations Using Graphics Processing Units

Aurel Neic,
Institute of Mathematics and Scientific Computing, Karl Franzens University of Graz, Graz, Austria

Manfred Liebmann,
Institute of Mathematics and Scientific Computing, Karl Franzens University of Graz, Graz, Austria

Elena Hoetzl,
Institute of Biophysics, Medical University of Graz, Graz A-8036, Austria

Lawrence Mitchell,
Edinburgh Parallel Computing Centre, University of Edinburgh, Edinburgh EH8 9YL, Scotland

Edward J. Vigmond [Member, IEEE],
LIRYC, L'Institut de Rythmologie et modèlisation cardiaque, University of Bordeaux1, Pessac 33604 France, and the Department of Electrical and Computer Engineering, University of Calgary, Calgary AB T2N 1N4, Canada

Gundolf Haase, and
Institute of Mathematics and Scientific Computing, Karl Franzens University of Graz, Graz, Austria

Gernot Plank
Institute of Biophysics, Medical University of Graz, Graz A-8036, Austria, and also with the Oxford e-Research Centre, University of Oxford, Oxford, OX1 2JD, U.K

Aurel Neic: aurel.neic@uni-graz.at; Manfred Liebmann: manfred.liebmann@uni-graz.at; Elena Hoetzl: elena.hoetzl@uni-graz.at; Lawrence Mitchell: lawrence.mitchell@ed.ac.u; Edward J. Vigmond: edward.vigmond@u-bordeaux1.fr; Gundolf Haase: gundolf.haase@uni-graz.at; Gernot Plank: gernot.plank@medunigraz.at

Abstract

Anatomically realistic and biophysically detailed multiscale computer models of the heart are playing an increasingly important role in advancing our understanding of integrated cardiac function in health and disease. Such detailed simulations, however, are computationally vastly demanding, which is a limiting factor for a wider adoption of in-silico modeling. While current trends in high-performance computing (HPC) hardware promise to alleviate this problem, exploiting the potential of such architectures remains challenging since strongly scalable algorithms are necessitated to reduce execution times. Alternatively, acceleration technologies such as graphics processing units (GPUs) are being considered. While the potential of GPUs has been demonstrated in various applications, benefits in the context of bidomain simulations where large sparse linear systems have to be solved in parallel with advanced numerical techniques are less clear. In this study, the feasibility of multi-GPU bidomain simulations is demonstrated by running strong scalability benchmarks using a state-of-the-art model of rabbit ventricles. The model is spatially discretized using the finite element methods (FEM) on fully unstructured grids.

Correspondence to: Gernot Plank, gernot.plank@medunigraz.at.

The GPU code is directly derived from a large pre-existing code, the Cardiac Arrhythmia Research Package (CARP), with very minor perturbation of the code base. Overall, bidomain simulations were sped up by a factor of 11.8 to 16.3 in benchmarks running on 6–20 GPUs compared to the same number of CPU cores. To match the fastest GPU simulation which engaged 20GPUs, 476 CPU cores were required on a national supercomputing facility.

Index Terms

Algebraic multigrid; domain decomposition; strong scalability; high-performance computing (HPC)

I. Introduction

The development of anatomically realistic and biophysically detailed computer models of integrated cardiac function is an important research focus. Such *in-silico* models are considered a promising approach for integrating the wealth of multiscale data available to research in the postgenomic era into quantitative mechanistic models. However, the associated computational costs remain a major limiting factor.

Despite steep increases in compute power, execution times may be prohibitive when exploring larger parameter spaces. Next generation high-performance computing (HPC) resources promise to deliver significantly better performance in the PetaFLOPS or even ExaFLOPS range. However, exploiting such resources represents a formidable challenge. As dictated by the fundamental theorem of many-core computing—increases in clock frequency are no longer worth the costs of power consumed and heat dissipated—increases in computer power can only be achieved via a dramatic increase in the number of compute cores N_p. Therefore, it is vital to devise numerical methods that strongly scale to thousands of cores.

Most current bidomain codes do not benefit from such HPC environments since strong scalability is limited, at best, to a few hundred cores [1]. A major step forward was achieved recently [2]. Employing proper grid partitioning and asynchronous parallel IO techniques, a single activation sequence in a patient-specific anatomically realistic and biophysically detailed model of a human heart was achieved within 1 min using 16 384 cores. Although this step change in performance—a speedup of more than two orders of magnitude relative to the fastest simulations reported [3]—opens new perspectives, it also indicated that strong scalability of current codes may prove insufficient to fully exploit upcoming larger HPC platforms. For instance, the real-time lag factor ξ_r was measured to be ~240 on a 0.12 PetaFLOPS machine. To achieve real-time performance, ~4 million compute cores of a 30 PetaFLOPS computer would be required, assuming strong scalability could be achieved, something challenging due to the deterioration of the surface-to-volume ratio with N_p. That is, with increasing N_p, work assigned to a core decreases, i.e., the “volume,” but the relative communication cost, associated with the boundaries of a local domain, i.e., the “surface,” increases. As soon as communication costs dominate, no further benefits from adding compute cores are achieved.

An alternative to the CPU-based HPC approach is acceleration technologies such as GPUs. GPUs outperform modern multicore CPUs by almost an order of magnitude in terms of memory bandwidth and by about factors of 3 and 10 with regard to double and single precision floating point arithmetic, respectively. Additionally, GPUs are built to schedule a vast numbers of threads and, thus, efficiently reduce latencies in their many-core chip

architectures. Therefore, properly implemented algorithms that exploit these architectural advantages can dramatically improve performance.

This study aims to assess the potential of multi-GPU codes in the context of current state-of-the-art cardiac bidomain simulations, that is, for microanatomically accurate fully unstructured 3-D FE grids executed in parallel distributed memory environments, to allow for strong scalability up to a large N_p. The main challenge, the efficient solution of linear systems, is tackled by employing advanced numerical techniques, such as algebraic multigrid. Finally, the feasibility of efficiently integrating a large simulation code (~100k lines of code) built on top of a linear algebra package with a GPU-enabled domain decomposition solver code, while minimally perturbing the code base, is demonstrated. The integration of pre-existing software is of pivotal importance, considering that simulations at the level of complexity as shown in this study, cannot be executed otherwise.

II. Methods

A. Governing Equations

The bidomain equations in the elliptic-parabolic form the equations are given by

$$\begin{bmatrix} -\nabla \cdot (\sigma_i + \sigma_e)\nabla \phi_e \\ -\nabla \cdot \sigma_b \nabla \phi_e \end{bmatrix} = \begin{bmatrix} \nabla \cdot \sigma_i \nabla V_m + I_{ei} \\ I_{eb} \end{bmatrix} \quad (1)$$

$$\beta C_m \frac{\partial V_m}{\partial t} = (\nabla \cdot \sigma_i \nabla \phi_i) - \beta (I_{ion}(V_m, \eta) - I_i) \quad (2)$$

$$V_m = \phi_i - \phi_e \quad (3)$$

where ϕ_i and ϕ_e are the intracellular and extracellular potentials, respectively, $V_m = \phi_i - \phi_e$ is the transmembrane voltage, $\boldsymbol{\sigma}_i$ and $\boldsymbol{\sigma}_e$ are the intracellular and extracellular conductivity tensors, respectively, β is the membrane surface to volume ratio, I_m is the transmembrane current density, I_{ei} and I_{eb} are extracellular stimuli applied in the interstitial space or bath, respectively, I_i is an intracellular current stimulus, C_m is the membrane capacitance per unit area, and I_{ion} is the membrane ionic current density that depends on V_m and a set of state variables $\boldsymbol{\eta}$. At tissue boundaries, no flux boundary conditions are imposed for ϕ_i, with the potential ϕ_e and the normal component of the extracellular current being continuous. At boundaries of the conductive bath surrounding the tissue, no flux boundary conditions for ϕ_e are imposed.

B. Spatio-Temporal Discretization

The PDEs given in (1) to (2) are spatially discretized using linear tetrahedral FEs [4]. Parabolic and elliptic PDE are decoupled and operator splitting based on a Strang scheme is employed to solve the parabolic PDE. The inner kernel of the compute scheme is then given by

$$\mathbf{K_{i+e}} \boldsymbol{\phi}_\mathbf{e}^{\,t+1} = -\mathbf{P}\mathbf{K_i}\mathbf{v_m}^t + \mathbf{M_e}\mathbf{I_e}^t \quad (4)$$

$$\frac{\partial \eta}{\partial t} = f(\mathbf{v_m}^t, \eta^{t+1/2}) \quad (5)$$

$$\mathbf{I_{ion}}^{t+1/2}=g(\mathbf{v_m}^{t},\eta^{t+1/2})\quad \mathbf{v_m^{t+1/2}}=\mathbf{v_m^{t}}-\frac{\Delta t}{C_m}\mathbf{I_{ion}^{t+1/2}}\quad \mathbf{M_i}\frac{\partial \mathbf{v_m^{t+1/2}}}{\partial t}=-\frac{1}{\beta C_m}\mathbf{K_i}\left(\mathbf{v_m}^{t+1/2}+\mathbf{P^{-T}}\boldsymbol{\phi}_e^{t+1}\right) \quad (6)$$

where $\mathbf{M}_\zeta$ and $\mathbf{K}_\zeta$ are FEM mass and stiffness matrices, respectively, with the subscript ζ specifying which conductivities to use, either intracellular *i*, extracellular *e*, or their sum *i* + *e*, **P** is a projection operator that transfers data between intracellular and extracellular grid, and $\boldsymbol{\eta}$ is a discrete representation of the set of state variables of the ionic model. For time discretization of the diffusion part of the parabolic PDE, the fully implicit Crank–Nicholson scheme is employed, with κ defined as $\beta C_m/\Delta t$:

$$\left(\kappa\mathbf{M_i}+\frac{\mathbf{K_i}}{2}\right)\mathbf{v_m^{t+1}}=-\mathbf{K_i}\left(\frac{\mathbf{v_m^{t+1/2}}}{2}+\mathbf{P^{-T}}\boldsymbol{\phi}_e^{t+1}\right)+\kappa\mathbf{M_i}\mathbf{v_m^{t+1/2}}. \quad (7)$$

C. Parallel Implementation Strategy

The Cardiac Arrhythmia Research Package (CARP) [5], which is built on top of the MPI-based library PETSc [6], was used as a framework for solving the cardiac bidomain equations in parallel. PETSc [6] served as the basic infrastructure for handling parallel matrices and vectors. Hypre [7] advanced algebraic multigrid methods such as BoomerAMG and ParMetis [8], graph-based domain decomposition of unstructured grids, were compiled with PETSc as external packages. An additional package, the publicly available Parallel Toolbox (pt) library (http://paralleltoolbox.sourceforge.net), [9] which can be compiled for both CPUs and GPUs, was interfaced with PETSc.

The parallelization strategy was based on recent extensions of the CARP framework [2]. Briefly, achieving good load balancing, where both computational load and communication costs are distributed as evenly as possible, is of critical importance. While this is achieved for structured grids with relative ease [10] since nodal numbering relationship is the same everywhere, mirroring the spatial relationship, it is far more challenging in the more general case of unstructured grids, which are preferred for cardiac simulations [11]. For general applicability, unstructured grids are mandatory to accommodate complex geometries with smooth surfaces, which prevent spurious polarizations when applying extracellular fields. To obtain a well-balanced grid partitioning, ParMeTis computes a k-way element-based partition of the mesh's dual graph, to redistribute finite elements among partitions. Depending on whether PETSc or pt was employed, two different strategies were devised. This was necessary due to the inherently different parallelization philosophies—PETSc partitions matrices row-wise, i.e., every node is uniquely assigned to a partition, whereas pt is a domain decomposition code where each partition holds those parts of the matrix that correspond to the local domain. Therefore, matrix rows are fragmented with pt, i.e., one matrix row might be stored across several processes. For vectors, similar differences arise. While every entry in a PETSc vector is uniquely assigned to a partition, entries in pt vectors that represent nodal values along boundaries between two or more subdomains are stored on all processes sharing the boundary. These fundamental differences are illustrated in Fig. 1.

In both scenarios, following the domain decomposition step, nodal indices in each domain were renumbered. Inner nodes were linearly numbered, forming the main diagonal block of the global matrix, which map onto the local rows of the linear system to solve. In the PETSc scenario, interface nodes have to be split and assigned to one parallel partition. Since entries in the off-diagonal of a matrix are more expensive in terms of communication cost, we aimed at evenly distributing interface nodes across the computed partitions to load balance communication. Interfacial nodes were split equally between the minimum and maximum

numbered partitions. Both grid partitioning and nodal renumbering were tightly integrated to compute partitioning information very fast in parallel on the fly. Permutation vectors kept track of the relationship between the reordered mesh and the user-provided canonical mesh.

D. Software Engineering Considerations

While the libraries Hypre and ParMeTis are straight forwardly integrated with PETSc as external packages, integration of pt is a more challenging problem due to the inherent conceptual difference. To reconcile this under the constraint of minimizing the perturbation of the code base, we wrapped the PETSc and pt matrix formats into a basic matrix data structure where C/C++ defines decided at compile time which matrix format to use. The parallel vector format remained unchanged in standard PETSc format. These decisions were based on the observation that the number of distinct matrix operations is small—in our case, only 12 functions affected—relative to the number of vector operations. Therefore, vector data structures could be reused unmodified, only additional gathering or scattering operation were required in any matrix-vector operation to communicate off-partition entries of the vector needed in a particular partition. For instance, when calling solving a linear system, a gathering operation was performed first for ghost padding of the initial guess and right-hand side vector, and, after solving the system, the solution vector was scattered back.

Following this strategy, coding effort was minimized. Modifications of the basic matrix data structure and matrix-vector operations required less than 100 lines of code, while the API for connecting CARP with pt amounted to 980 lines. Compared to the CARP code base of ~100k lines of code, these costs were negligible.

E. GPU Implementation

In our discretization scheme, there are three main contributors to the overall computational cost: solving the linear systems associated with elliptic and parabolic PDEs, and the set of ODEs representing cellular dynamics. In line with the notion of minimal perturbation, only these three time critical portions were implemented on the GPU. At each step outlined in (4)–(6), right-hand side vectors and initial guesses are transferred from the host (CPU) to the device (GPU), while the matrices associated with the various linear systems reside on the GPU for an entire simulation. The problem is solved on the device and the solution transferred back to the host. The advantage is that the code controlling the global execution, which is, generally, not time critical, remains unchanged. In our multi-GPU implementation, the bidomain equations are partitioned exactly the same way as in the multi-CPU case, where an MPI process is linked to a single GPU and vice versa.

1) GPU Implementation of the ODE Solvers—The ionic model library included in CARP, which implemented the Rush–Larsen technique [12], [13] was compiled for the GPU. The exact same code was executed either on the GPU or CPU. When executed on the GPU, the transmembrane voltage vector $\mathbf{v_m}^t$ was copied over to the GPU memory to solve (5) at each time step. State variables $\boldsymbol{\eta}$, since they are not required for solving the PDEs, were kept in GPU memory. That is, the update of the state vector, the computation of the total ionic current $\mathbf{I_{ion}^{t+1/2}}$ and the update of the voltage vector $\mathbf{v_m}^{t+1/2}$ based on the contribution of the reaction term are executed on the GPU. Subsequently, $\mathbf{v_m}^{t+1/2}$ is transferred back to the CPU memory prior to being sent back to the GPU for solving the parabolic PDE in (6). Details on the GPU-based ODE solver have been described previously [14].

2) GPU Implementation of the PDE Solvers—Solving sparse linear systems on GPUs efficiently is a more involved endeavor. Although support for GPU execution has been

added via Cusp and Thrust to the PETSc library, this applies to the PETSc base package only, but not to external packages such as Hypre [7] that provide multigrid preconditioners crucial for solving the bidomain equations efficiently [15], [16]. The key advantage of the small pt solver package is that pt can be compiled for both multi-CPU and multi-GPU environments. The library includes a highly optimized algebraic multigrid framework that can be used as a preconditioner for an iterative conjugate gradient solver and as such is very well suited for solving the elliptic PDE embedded in the bidomain equations. An additional ω-Jacobi was implemented since algebraic multigrid is too expensive for solving the parabolic PDE. The exact same interface to pt could be used as in the CPU case, no extra costs incurred.

F. Numerical Setup and Benchmarking

An anatomically detailed FE model of rabbit ventricles immersed in a conductive fluid was used for benchmarking [1], [17], [18]. The FE mesh of the ventricles comproised 4.3 million vertices and 24.1 million tetrahedra, whereas the mesh of the surrounding bath comprised 2.6 million vertices and 16.9 million tetrahedra. Cellular dynamics were modeled using a recent rabbit ventricular action potential model [19]. A propagating wavefront was initiated at the left ventricular apex using a transmembrane current pulse of 50-μA/cm^2 strength and 1-ms duration. Electrical activity was simulated over 250 ms to observe a full action potential cycle over the entire ventricular volume. A full bidomain formulation was used for all benchmarks runs, using a fixed global time step of 25 μs. All linear systems were solved using iterative solvers where the convergence crtierion was an absolute L_2 norm of 10^{-6} of the unpreconditioned residual. Execution times spent on solving elliptic PDE, parabolic PDE and ODEs were recorded.

Benchmark runs were executed employing the CARP framework, compiled for three different flavors.

1. The CARP standard setup running on CPU cores ($CARP_{std}$) [1], [16]: The elliptic PDE is solved using Boomer AMG [20] preconditioning for the CG solver (BAMG-CG). The parabolic PDE is solved using a global block Jacobi preconditioner with an incomplete Cholesky preconditioner in each subblock (BJ-ICC-CG). The set of ODEs is solved using an optimized Rush–Larsen technique [12], [13].
2. CARP coupled to pt running on CPU cores ($CARP_{pt}$): pt was engaged to solve the linear systems. A previously described AMG preconditioner for CG (ptAMG-CG) was employed for the elliptic PDE [9]. The stopping criterion for coarsening of the AMG preconditioner ensured a global size of the coarse system in the range 8000–12 000. The coarse grid system was solved in parallel using the direct solver MUMPS [21]. For the parabolic PDE, an ω-Jacobi preconditioner in combination with CG (ωJ-CG) was implemented within pt, using an ω of 2/3 and two iterations per CG iteration.
3. CARP coupled to pt running on GPU cores ($CARP_{gpu}$): PDEs are solved using the same methods as in $CARP_{pt}$, but on the GPU. A slightly different setup had to be used for solving the elliptic PDE due to the lack of a GPU-enabled direct solver: Therefore, the multigrid stopping criterion for coarsening was set to allow coarsening until the local systems were linearly independent. ODEs were solved using the same method [13], but compiled on the GPU [14].

G. Performance Metrics

The following metrics were used to evaluate parallel performance.

1. Real time lag factor ξ_r: Execution time divided by the time span simulated.
2. Parallel efficiency E: The algorithm's ability to reduce execution time as N_p increases, defined as the ratio between the factors by which calculation time has been reduced and N_p has been increased. An E value of 1 is referred to as *linear scaling*, because computation time is reduced by the same amount as N_p is increased. E values below 1 are *sublinear scaling* while above are *super linear scaling*.
3. Speedup S: The reduction in execution time of the $CARP_{pt}$ and $CARP_{gpu}$ benchmark flavors compared to the $CARP_{std}$ baseline benchmark.

H. Benchmarking Hardware

The bulk of benchmark simulations were executed on the Mephisto GPU cluster, consisting of six Supermicro Super-Server nodes—one master- and five compute-nodes—each equipped with two Intel Xeon Six-Core CPUs X5650 clocked at 2.67 GHz. Each compute node is equipped with 96 GB DDR3 ECC RAM and four Nvidia Tesla C2070 GPUs, each with 6 GB RAM, giving a total of 24 GB of GDDR5 RAM per compute node. In total, the five nodes provide 20 GPUs (8960 CUDA Cores) with a total video memory of 120 GB RAM, and 60 CPU cores with a total of 480 GB of RAM. All nodes are interconnected by a 40 GB/s QDR low-latency InfiniBand switch.

As a reference, all CPU benchmarks with the $CARP_{std}$ executable were repeated on the Phase 2B element of the wider HECToR service, the national supercomputing facility in the U.K., using 96–1536 cores. Briefly, each HECToR blade contains four compute nodes, each with two 12-core AMD Opteron 2.1 GHz Magny Cours processors. Each 12-core processor shares 16 GB of memory. The MPI point-to-point bandwidth is 5 GB/s or more and latency between two nodes is around 1–1.5 μs. More details are found online (http://www.hector.ac.uk/service/hardware/phase2b.php).

III. Results

As previously [1], a strong scalability benchmark was performed using a state-of-the-art anatomically accurate and biophysically detailed computer model of rabbit ventricles [see Fig. 2(a)]. The stimulus protocol induced electrical activity representative of a paced activation sequence under healthy conditions. This sequence covered all phases of an action potential cycle [see Fig. 2(c)] and allowed study of the impact of these phases upon performance of iterative solvers [see Fig. 2(d)].

A. Baseline Benchmarks

To allow comparison with previous reports [1], [16] benchmarks were performed employing $CARP_{std}$ and $CARP_{pt}$ using 6–60 CPU cores on the Mephisto GPU cluster. These benchmarks allowed us to assess performance for two factors: 1) the custom-tailored AMG approach, as implemented in pt [22], relative to the performance of BAMG-CG [1], [16], and, 2) GPUs, since $CARP_{pt}$ and $CARP_{gpu}$ rely upon the same solver algorithm. Measured execution times are summarized in Tables I and II. Overall, performance differences between the two implementations are comparable, with $CARP_{pt}$ being slightly faster in solving the elliptic PDE while being slower in solving the parabolic PDE, $CARP_{pt}$ being overall faster by 26–36%.

B. GPU Benchmarks

Benchmarks were repeated with $CARP_{gpu}$ where the number of GPUs was varied between 6 and 20 and summarized in Table III. Comparing performance between $CARP_{pt}$ and

$CARP_{gpu}$ reveals that there are dramatic speedups for all three major components of the computational scheme. Even the smallest GPU setup with 6 GPUs clearly outperformed the fastest CPU setup with 60 CPU cores. In a 1:1 GPU to CPU comparison, the GPU code always outperformed the CPU code by more than an order of magnitude, with S being in the range between 11.8 and 16.3.

C. Strong Scalability and Parallel Efficiency

Strong scalability is a key metric for assessing the potential of numerical methods on current and future HPC resources. Comparing $CARP_{std}$ with $CARP_{pt}$ showed very similar scalability characteristics (see columns E in Tables I and II as well as Fig. 3). Although the GPU code performed best in terms of overall execution time, scalability was inferior to CPU counterparts. For instance, the worst E with the CPU benchmarks was 81.0%, observed when all 60 CPU cores were engaged. With GPUs, E degraded to these lower value at a much smaller N_p where E dropped to 79% with only 12 GPUs (see Table III). An overview showing the scaling properties of all executables in terms of both overall compute time as well as time spent on individual components is given in Fig. 3. Despite inferior scaling of the GPU code, reductions in execution time were substantial. The number of CPU cores required to match the fastest GPU simulation run $\bar{N}_p$ turned out to be 472 cores. For the elliptic PDE, parabolic PDE and the set of ODEs, $\bar{N}_p$ was 567, 274, and 367 cores, respectively.

IV. Discussion

In this study, the potential of a multi-GPU implementation for solving the cardiac bidomain equations was investigated. Strong scalability benchmarks were performed using a state-of-the-art anatomically accurate and biophysically detailed in-silico model of rabbit ventricles that relied upon a fully unstructured FE grids for spatial discretization. The main emphasis was on employing current numerical techniques, such as algebraic multigrid, for solving large sparse linear systems of equations for which efficient GPU implementations remain a challenge. Finally, the feasibility of integrating a GPU-enabled domain-decomposition solver code with a large simulation code built on top of the PETSc linear algebra package is demonstrated that minimizes the perturbation of the code base. This is of particular importance when considering that codes for performing simulations at this level of complexity typically require decades of person years for development.

A. Previous Work on Multigrid Solvers for GPUs

There is a limited number of reports on GPU-enabled multigrid solvers suitable for the bidomain equations. FEAST [23], [24] relies on block-structured grids to take advantage of local structure on a single GPU, and is considered to be among the most advanced methods; however, fully unstructured grids are unsupported. Notay [25] proposed a geometry-based semistructured algebraic multigrid method that is also not suited for unstructured grids nor is available on the GPU. Similar limitations apply to the matrix-free multigrid solver reported by Flaig and Arbenz [26] which requires direct implementation of material parameters in the code, something not feasible in our case. GPU support has been added to the most recent PETSc release [6] that is very well suited for unstructured grids. However, no advanced preconditioners are available for GPUs and acceleration factors of ~3, with a diagonally preconditioned cg solver, are well below expectations. The Hypre [7] preconditioners, which include BoomerAMG, will not be transferred to GPUs, Trilinos [27] preconditioners are currently implemented on GPUs, but no benchmark results are available yet. An interesting approach of implementing algebraic multigrid on GPUs based on the Thrust library (CUDA) has been proposed recently [28]. The reported GPU accelerations for unstructured problems by a factor 2–3 are inferior to those achieved with pt, and the use of multiple GPUs is in its

infancy. Nevertheless, the two latter solvers are promising and may be of interest at a more mature stage. While GPU implementations of direct solvers such as MUMPS [21] and SuperLU [29] are announced for 2012, their CPU parallel performance is clearly inferior to pt, and preliminary GPU acceleration results are not too promising. However, such direct solvers may be useful in multigrid implementations for the coarsest mesh.

B. Previous GPU Work on Cardiac Tissue Simulations

Previous studies have evaluated the potential of GPUs for solving the cardiac monodomain equation. Sato *et al.* [30] demonstrated speedups by a factor of 32. While demonstrating that GPUs yield speedups for this type of equation, the methods are not applicable to bidomain simulations. To summarize the most severe limitations, regular grids were used with a matrix-free finite difference stencil for spatial discretization; the parabolic PDE was solved explicitly, that is, no linear system was solved; simulations were parallelized over GPUs using threads instead of MPI, thus limiting the number of GPUs to the number of available slots on a single mainboard; Older NVidia GT 8800 and GT 9800 GX2 cards were used that are faster for single precision floating point operations than the Tesla GPUs used in our study, but do not support double precision calculations; the phase I of the Luo–Rudy ionic model [31] was used that has far fewer variables and less stiff dynamics than the one used in this study [19]; finally, the set of ODEs was solved using an explicit Euler method that does not necessarily provide any speedup when compared to more stable ODE integration techniques such as optimized Rush–Larsen methods [12], [13].

For modeling studies that aim to capturing cardiac anatomy as faithfully as possible, unstructured grids are key to smoothly accommodate organ boundaries. In these cases, explicit methods may be disadvantageous since time stepping is severely limited by the Courant–Friedrich–Lewy condition [32] that depends on the shortest edge lengths in the mesh. Subtime stepping techniques are needed to maintain stability [2]. In [30], 4–6 subtime steps were required per ODE solver step to satisfy the CFL condition, suggesting that implicit methods may be better suited. We investigated this previously [33] where an implicit method was implemented on the GPU. FEM was employed for spatial discretization and full stiffness matrices were accumulated, necessitating the efficient implementation of matrix-vector products for the GPU. Using an unpreconditioned CG method, promising speedups were achieved with 2-D tissue models, but with unstructured 3-D bidomain simulations, the number of iterations required for convergence became prohibitive, thus rendering the method impractical in a more general context.

C. Strong Scalability of Solver Components

Independent of CPU or GPU, strong scalability is a key metric. Since almost all HPC scenarios rely on the use of a massively large number of compute cores, only strongly scalable codes will benefit. A large N_p means that for a given problem size, the local workload becomes small enough so that communication costs start to carry weight. This can be measured with strong scaling experiments, but not weak scaling experiments [10]. The largest bidomain strong scalability data have been reported recently [34] where local workload dropped to 3k nodes with $N_p = 2048$. Potse *et al.* [35] reported a smaller bidomain strong scalability benchmark for up to 96 cores on a shared memory machine. However, due to the large problem size of 55 million nodes and the small $N_p = 96$, local workload was ~573k nodes, which is too large to draw meaningful conclusions in terms of current HPC.

The three major components of the bidomain solver scheme implemented here pose quite different requirements for achieving favorable strong scalability. The easiest problem is the solution of the ODEs due to its embarrassingly parallel nature. This results in almost linear scaling, as shown in Fig. 3(c). In general, the parabolic problem is the next less expensive

system to solve. Either highly scalable explicit approaches can be used [2], or cheap iterative solvers such as ωJ-CG or BJ-ICC-CG. Due to diagonal dominance, particularly pronounced with mass lumping, convergence is fairly quick. In our benchmarks, the BJ-ICC-CG solver converged within 6–12 iterations (average 8.6) and the ωJ-CG solver within 9–16 iterations (average 11.3). The most challenging problem is the elliptic PDE. More expensive preconditioners, such as algebraic multigrid methods [9], [20] are much more efficient. Cheaper iterative solvers require an excessively large number of iterations [16]. Strong scalability is more difficult to achieve due to the rapid increase in communication costs arising at the coarser grids [34]. This is illustrated in the top panels of Fig. 3. Scalability started to level off around 768 cores as indicated by the drop in E from 63%/70% down to 35%/43% for elliptic and parabolic solvers, respectively.

D. Potential of GPU-Enabled Codes for Bidomain Studies

Strong scaling properties are governed by the communication cost relative to the computational workload per core. Due to the coprocessor model implemented, extra costs incur for GPUs due to transferring vectors between host and device. Since communication costs are slightly higher with GPUs, while solving costs are reduced, inferior strong scaling is expected, which is confirmed. Despite this, GPUs can lead to significant reductions in execution time. The various solver components benefit to different degrees. Solving the set of ODEs on GPUs does yield the expected benefits, with robust speedups in the range between 11.6 and 12.5 (compare ODE columns in Tables II and III) without any degradation of scaling characteristics compared to the CPU [see Fig. 3(c)].

Speedups achieved in solving the elliptic and parabolic PDEs were in the ranges 12.7–18.5 and 8.1–9.5, respectively. While scalability of the parabolic GPU solver was only moderately inferior to its CPU counterpart, more noticeable differences were observed with elliptic solvers. This is due to a less favorable computation-to-communication ratio and the use of a different numerical strategy on the GPU. On the CPU, MUMPS [21] was used as a coarse grid solver, whereas, on the GPU, the system was coarsened until it was linearly independent, since no direct sparse GPU solver was available. The direct solver quartered the number of CG iterations and helped reduce the dependence of convergence on the distribution of Dirichlet nodes.

While benchmark results were reported only for one specific grid, the methods used are general and independent of grid structure. As confirmed by the vast number of simulation runs performed with our code with grids of varying complexity, the important factor is the surface-to-volume ratio that depends only on grid size and number of cores, not on the structure of the grid itself, as long as mesh quality is sufficient.

E. Framework-Specific Aspects

Every numerical method must be implemented in a software package running on hardware, and, thus, every benchmark is dependent on the underlying hardware and software frameworks. In this study, CARP was used as a software framework to test the potential benefits of multi-GPU execution of bidomain simulations. While benchmarks were shown only for CARP, the approaches described are generic, allowing the implementation of these methods within other frameworks with similar benefits. To summarize, the two most important aspects to address are: 1) Independent of CPU or GPU, both compute and communication load has to be well balanced to achieve strong scalability; 2) With regard to GPU execution, the implementation of optimized sparse matrix-vector operations as well as advanced solver techniques that account for the specifics of GPU hardware are key to achieve maximal speedup in the generic scenario of fully unstructured FE grids.

All major aspects of the presented method are built upon publicly available software packages, thus facilitating the replication of these benchmarks within other software frameworks. As numerous other scientific simulation frameworks CARP relies upon the MPI-based library PETSc [6] that served as the basic infrastructure for handling parallel matrices and vectors, and external packages such as Hypre [7] provide advanced algebraic multigrid methods such as BoomerAMG. Proper load balancing relied upon the graph-based domain decomposition package ParMetis [8]. GPU-enabled solution of linear systems was built on top of the publicly available Parallel Toolbox (pt) library (http://paralleltoolbox.sourceforge.net), [9]. The only CARP-specific aspect is the interface code that had to be implemented to glue the various methods together. Compared to the overall size of a framework such as CARP, the effort invested in the interface code was virtually negligible, requiring only 980 lines of code including documentation. While this step can be replicated in any other framework, it requires an in-depth understanding of the framework used and, thus, efficient implementations can only be achieved by the developer teams of the various simulation frameworks themselves.

V. Conclusion

This study demonstrates the significant benefits of using GPUs for solving the cardiac bidomain equations. All components of a bidomain solver could be sped up by more than an order of magnitude. Despite inferior scalability, a small GPU cluster could match the performance of a substantially larger setup on national supercomputing facility, where roughly 25 CPU cores were needed to match one GPU.

Acknowledgments

G. Plank and E. J. Vigmond have partial ownership of CardioSolv, LLC. CardioSolv, LLC was not involved in this research.

This research is supported by the Austrian Science Fund FWF Grant F3210-N18, the National Institute of Health grant NIH RO1 HL 10119601 and a class 1b award of the Engineering and Physics Research Council in the U.K.

References

1. Plank G, Burton RA, Hales P, Bishop M, Mansoori T, Bernabeu MO, Garny A, Prassl AJ, Bollensdorff C, Mason F, Mahmood F, Rodriguez B, Grau V, Schneider JE, Gavaghan D, Kohl P. Generation of histo-anatomically representative models of the individual heart: Tools and application. Phil. Trans. R. Soc. A. 2009; vol. 367(no. 1896):2257–2292. [PubMed: 19414455]
2. Niederer S, Mitchell L, Smith N, Plank G. Simulating human cardiac electrophysiology on clinical time-scales. Front Physiol. 2011; vol. 2:14. [PubMed: 21516246]
3. Reumann, M.; Fitch, BG.; Rayshubskiy, A.; Keller, D.; Seemann, G.; Doessel, O.; Pitman, M.; Rice, J. Strong scaling and speedup to 16,384 processors in cardiac electro-mechanical simulations; Proc. IEEE Eng. Med. Biol. Soc. Conf; 2009.
4. Rocha BM, Kickinger F, Prassl AJ, Haase G, Vigmond EJ, dos Santos RW, Zaglmayr S, Plank G. A macro finite-element formulation for cardiac electrophysiology simulations using hybrid unstructured grids. IEEE Trans. Biomed. Eng. 2011 Apr; vol. 58(no. 4):1055–1065. [PubMed: 20699206]
5. Vigmond E, Hughes M, Plank G, Leon L. Computational tools for modeling electrical activity in cardiac tissue. J. Electrocardiol. 2003; vol. 36:69–74. [PubMed: 14716595]
6. Balay, S.; Buschelman, K.; Eijkhout, V.; Gropp, WD.; Kaushik, D.; Knepley, MG.; McInnes, LC.; Smith, BF.; Zhang, H. PETSc users manual. DuPage County, IL: Argonne Nat. Lab.; 2008. Tech. Rep. ANL-95/11 - Revision 3.0.0
7. Hypre team. Hypre—High performance preconditioners User's Manual. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory; 2006 Dec.. [Online]. Available:

https://computation.llnl.gov/casc/hypre/download/hypre-2.0.0_usr_manual.pdf, Tech. Rep. Software Version: 2.0.0.

8. Karypis, G.; Kumar, V. MeTis: Unstructured graph partitioning and sparse matrix ordering system. Minneapolis, MN: University of Minnesota; 2009. [Online]. Available: http://www.cs.umn.edu/smetis.
9. Haase, G.; Liebmann, M.; Douglas, CC.; Plank, G. A parallel algebraic multigrid solver on graphics processing units. In: Zhang, W.; Chen, Z.; Douglas, CC.; Tong, W., editors. HPCA (China), Revised Selected Papers (Lecture Notes in Computer Science Series). Vol. vol. 5938. Berlin, Germany: Springer; 2009. p. 38-47.
10. Munteanu M, Pavarino L, Scacchi S. A scalable newton-krylovschwarz method for the bidomain reaction-diffusion system. SIAM J. Sci. Comput. 2009; vol. 31:3861–3883.
11. Vigmond EJ, Weber dos Santos R, Prassl AJ, Deo M, Plank G. Solvers for the cardiac bidomain equations. Prog. Biophys. Mol. Biol. 2008; vol. 96(no. 1–3):3–18. [PubMed: 17900668]
12. Rush S, Larsen H. A practical algorithm for solving dynamic membrane equations. IEEE Trans. Biomed. Eng. 1978 Jul.vol. 25(no. 4):389–392. [PubMed: 689699]
13. Plank G, Zhou L, Greenstein JL, Cortassa S, Winslow RL, O'Rourke B, Trayanova NA. From mitochondrial ion channels to arrhythmias in the heart: Computational techniques to bridge the spatio-temporal scales. Philos. Transact. A Math. Phys. Eng. Sci. 2008; vol. 366(no. 1879):3381–3409.
14. Vigmond EJ, Boyle PM, Leon L, Plank G. Near-real-time simulations of biolelectric activity in small mammalian hearts using graphical processing units. Proc. IEEE Eng. Med. Biol. Soc. Conf. 2009; vol. 2009:3290–3293.
15. Weber dos Santos R, Plank G, Bauer S, Vigmond EJ. Parallel multigrid preconditioner for the cardiac bidomain model. IEEE Trans. Biomed. Eng. 2004 Nov.vol. 51(no. 11):1960–1968. [PubMed: 15536898]
16. Plank G, Liebmann M, dos Santos RW, Vigmond E, Haase G. Algebraic multigrid preconditioner for the cardiac bidomain model. IEEE Trans. Biomed. Eng. 2007 Apr.vol. 54(no. 4):585–596. [PubMed: 17405366]
17. Bishop MJ, Plank G, Burton RA, Schneider JE, Gavaghan DJ, Grau V, Kohl P. Development of an anatomically-detailed MRI-Derived rabbit ventricular model and assessment of its impact on simulation of electrophysiological function. Amer. J. Physiol. Heart Circ. Physiol. 2009; vol. 298(no. 2):699–718.
18. Prassl AJ, Kickinger F, Ahammer H, Grau V, Schneider J, Hofer E, Vigmond E, Trayanova NA, Plank G. Automatically generated, anatomically accurate meshes for cardiac electrophysiology problems. IEEE Trans. Biomed. Eng. 2009 May; vol. 56(no. 5):1318–1330. [PubMed: 19203877]
19. Mahajan A, Shiferaw Y, Sato D, Baher A, Olcese R, Xie L, Jam M, Chen P, Restrepo J, Garfinkel A, Qu Z, Weiss JN. A rabbit ventricular action potential model replicating cardiac dynamics at rapid heart rates. Biophys. J. 2008; vol. 94:392–410. [PubMed: 18160660]
20. Henson V, Yang U. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. Appl. Numerical Math. 2002; vol. 41:155–177.
21. Amestoy, P.; Duff, IS.; L'Excellent, J-Y.; Koster, J. Proc. 5th Int. Workshop Appl. Parallel Comput., New Paradigms for HPC in Industry and Academia. London, UK: Springer-Verlag; 2001. Mumps: A general purpose distributed memory sparse solver; p. 121-130.
22. Liebmann, M. Ph.D. dissertation, Dept. Math. Sci. Comput., Univ. Graz. 2009. Efficient PDE solvers on modern hardware with applications in medical and technical sciences.
23. Turek S, Göddeke D, Becker C, Buijssen SH, Wobker H. FEAST—Realisation of hardware-oriented numerics for HPC simulations with finite elements. Concurrency and Comput.: Practice Expecience. 2010 May; vol. 22(no. 6):2247–2265. special Issue Proceedings of ISC 2008.
24. Göddeke, D. Ph.D. dissertation. Technische Univ. Dortmund, Fakultät für Mathematik; 2010 May. Fast and accurate finite-element multigrid solvers for PDE simulations on GPU clusters. [Online]. Available: http://hdl.handle.net/2003/27243.
25. Notay Y. An aggregation-based algebraic multigrid method. Electron. Trans. Numerical Anal. 2010; vol. 37(no. 6):123–146.

26. Flaig, C.; Arbenz, P. A highly scalable matrix-free multigrid solver for μ FE analysis based on a pointer-less octree. In: Lirkov, I.; Margenov, S.; Waniewski, J., editors. Large Scale Scientific Computing, (Lecture Notes in Computer Science Series). Vol. vol. 7116. Berlin, Germany: Springer; 2012. p. 498-506.
27. Bartlett, R. Sandia National Labs; 2007. Daily integration and testing of the development versions of applications and trilinos: A stronger foundation for enhanced collaboration in application and algorithm research and development. [Online]. Available: http://www.ornl.gov/8vt/AppPlusTrilinosDev2007.pdf, Tech. Rep. SAND2007-7040.
28. Bell N, Dalton S, Olson LN. Exposing fine-grained parallelism in algebraic multigrid methods. SIAM J. Scientific Comput. 2011 (in review).
29. Li X, Demmel J. SuperLU DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. ACM Trans. Math. Softw. 2003; vol. 29(no. 2):110–140.
30. Sato D, Xie Y, Weiss JN, Qu Z, Garfinkel A, Sanderson AR. Acceleration of cardiac tissue simulation with graphic processing units. Med. Biol. Eng. Comput. 2009; vol. 47(no. 9):1011–1015. [PubMed: 19655187]
31. Luo CH, Rudy Y. A model of the ventricular cardiac action potential. Depolarization, repolarization, and their interaction. Circ Res. 1991 Jun.vol. 68(no. 6):1501–1526. [PubMed: 1709839]
32. Courant R, Friedrichs K, Lewy H. Ü ber die partiellen Differenzengleichungen der mathematischen Physik. Math. Annalen. 1928; vol. 100(no. 1):32–74.
33. Rocha B, Campos F, Amorim R, Plank G, Weber dos Santos R, Liebmann M, Haase G. Accelerating cardiac excitation spread simulations using graphics processing units. Concurrency Computat.: Pract. Exper. 2010; vol. 23:708–720.
34. Mitchell L, Bishop M, Hötzl E, Neic A, Liebmann M, Haase G, Plank G. Modeling cardiac electrophysiology at the organ level in the peta flops computing age. AIP Conf. Proc. 2010; vol. 1281(no. 1):407–410.
35. Potse M, Dub B, Richer J, Vinet A, Gulrajani RM. A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart. IEEE Trans. Biomed. Eng. 2006 Dec.vol. 53(no. 12, pt 1):2425–2435. [PubMed: 17153199]

Biographies

Aurel Neic received the M.Sc. degree in scientific computing from the St. Pölten University of Applied Science, St. Pölten, Austria, in 2008. He is currently working toward the Ph.D. degree in the Institute of Mathematics and Scientific Computing, University of Graz, Graz, Austria.

His research interests include the implementation and analysis of parallel algorithms in the context of PDE solvers.

Manfred Liebmann received the M.Sc. degree in theoretical physics and the Ph.D. degree in mathematics both from the University of Graz, Graz, Austria.

He is currently an Assistant Professor in the Institute for Mathematics and Scientific Computing, University of Graz. Prior, he was a Project Assistant at the Institute of Computational Mathematics, University of Linz, Austria, from 2005 to 2006, and a Doctoral Scholar at the Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany, from 2006 to 2007, and a Research Assistant in the Institute for Mathematics and Scientific Computing, University of Graz, from 2007 to 2009, and a Postdoctoral Research Associate in the Department of Mathematics, University of Wyoming, Laramie, in 2009. His research interests include large scale numerical simulations, parallel numerical algorithms, GPU computing, number theory, graph theory, object-oriented software development, computer graphics, quantum computation and the foundations of quantum physics.

Elena Hoetzl received the Ph.D. degree in applied mathematics from the University of Graz, Graz, Austria, in 2009.

In 2010, she was a University Assistant at the Institute of Mathematics and Scientific Computing, University of Graz. Since 2010, she has been a Postdoctoral Research Fellow at the Institute of Biophysics, Medical University of Graz. Her research interests include inverse problems in tomography, mathematical image processing, numerical mathematics, optimization, and numerical analysis.

Lawrence Mitchell received the M.Phys. degree and the Ph.D. degree in physics from The University of Edinburgh, Edinburgh, Scotland, in 2005 and 2009, respectively.

He is an Applications Scientist at the Edinburgh Parallel Computing Centre, Edinburgh, Scotland. His research interests include parallel and high-performance computing, in particular, domain-specific languages, and compilers for finite-element methods.

Edward J. Vigmond (S’96–M’97) received the B.A.Sc. degree in electrical and computer engineering, and the M.A.Sc. and Ph.D. degrees both in the Institute of Biomedical Engineering all from the University of Toronto, Toronto, Canada, in 1988, 1991, and 1996, respectively.

He was a Postdoctoral Fellow with the University of Montreal, from 1997 to 1999, and with Tulane University, from 1999 to 2001. From 2001 to 2011, he was with the Department of Electrical and Computer Engineering, University of Calgary, and served as the Director for

the Centre for Bioengineering Research and Education, from 2009 to 2011. He is currently with the University of Bordeaux 1, Bordeaux, France, in the Institut Hospitalo-Universitaire LIRYC. His research interests include numerical computation, biomedical signal processing, and modeling of nonlinear biosystems.

Gundolf Haase received the Diplomas in teaching and mathematics and the Graduate degree all from Technische Universität, Karl–Marx–Stadt/Chemnitz, Germany, in 1988, 1991, and 1993, respectively.

He has been a Postdoctoral Fellow and he became an Associate Professor at the Institute for Computational Mathematics at University Linz, Linz, Austria, after his habilitation on parallel numerical algorithms in 2001. After getting a call he moved to University Graz, Graz, Austria, in 2004 for a Full Professorship at the Institute for Mathematics and Scientific Computing. His research interests include parallel computing, algebraic multigrid, hardware acceleration of algorithms, and in applying fast methods to real life problems.

Gernot Plank received the M.Sc. and Ph.D. degrees in electrical engineering from the Institute of Biomedical Engineering, Technical University of Graz, Graz, Austria, in 1996 and 2000, respectively.

He is currently an Associate Professor with the Institute of Biophysics, Medical University of Graz, Graz, Austria, and an Academic Fellow with the Oxford e-Research Centre, University of Oxford, Oxford, U.K. Prior, he was a Postdoctoral Fellow with the Technical University of Valencia, Valencia, Spain, from 2000 to 2002, the University of Calgary, AB, Canada, in 2003, and a Marie Curie Fellow with Johns Hopkins University, from 2006 to 2008. His research interests include computational modeling of cardiac electrophysiology and mechanics in terms of both methodological as well as applied aspects.

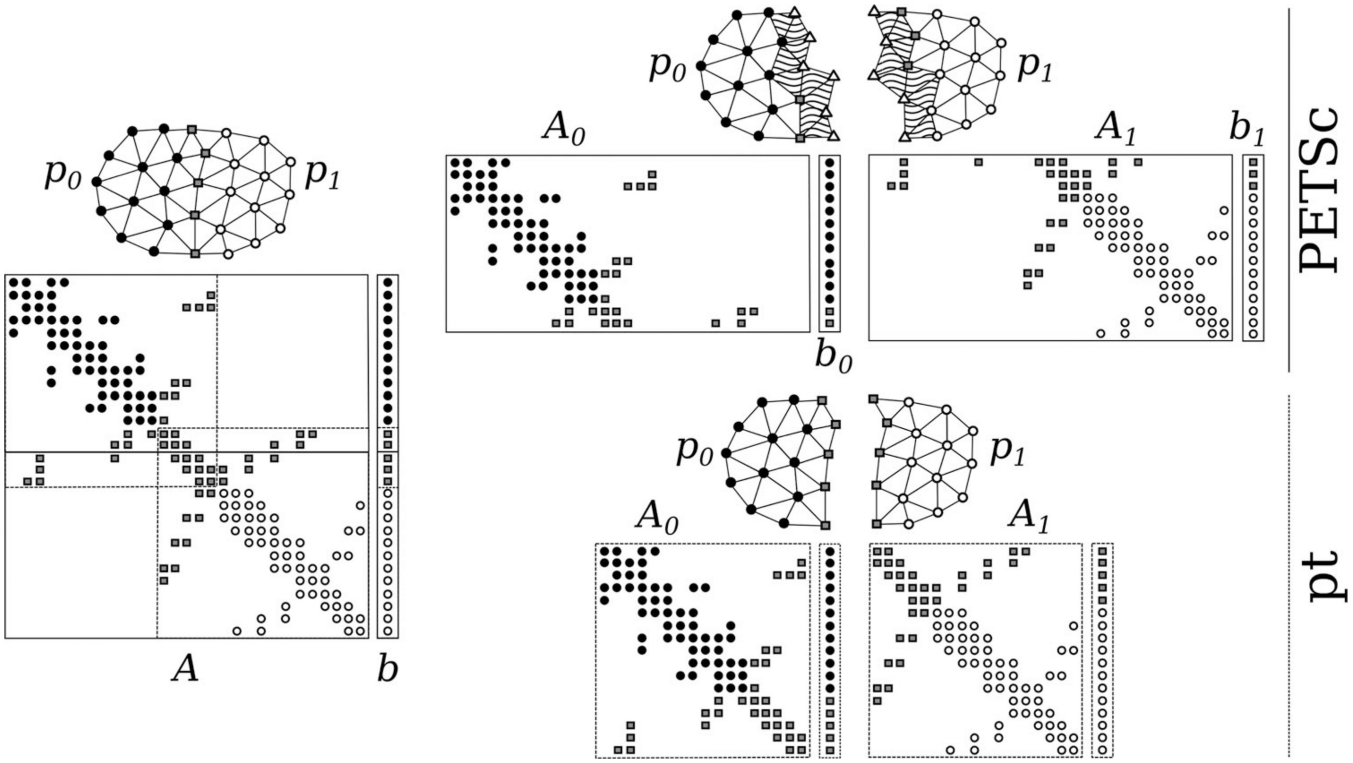

Fig. 1.

Parallelization strategies of the linear algebra package PETSc and the domain decomposition solver pt. *Left panel*: FE mesh of a domain consisting of two subdomains p_0 and p_1 and sparsity pattern of the corresponding global accumulated FE stiffness matrix A. In the FE mesh, black and white circles are inner nodes of p_0 and p_1, respectively, while squares are boundary nodes. Edges in the FE mesh correspond to nonzero entries of the stiffness matrix A. In the depicted sparsity pattern of A, white and black circles represent FE edges between inner nodes of p_0 and p_1, respectively, and squares represent FE edges from or to a boundary node. *Right upper panel*: Grid partitioning in the PETSc case. Submatrices A_0 and A_1, corresponding to p_0 and p_1, contain entire rows for all nodal indices assigned to a partition. Vector b_0 holds the entire local index range but for an operation equivalent to $A_0 \cdot b_0$ the node values corresponding to the triangles on p_0 need to be communicated, respectively, b_1 and p_1. Hatched areas in the FE mesh indicate domain overlap between p_0 and p_1. *Right lower panel*: Grid partitioning in the pt case. Domains p_0 and p_1 do not overlap. Therefore, rows corresponding to interface nodes in submatrices A_0 and A_1 are fragmented, but vectors b_0 and b_1 are complete, holding any entry corresponding to matrix entries in A_0 and A_1.

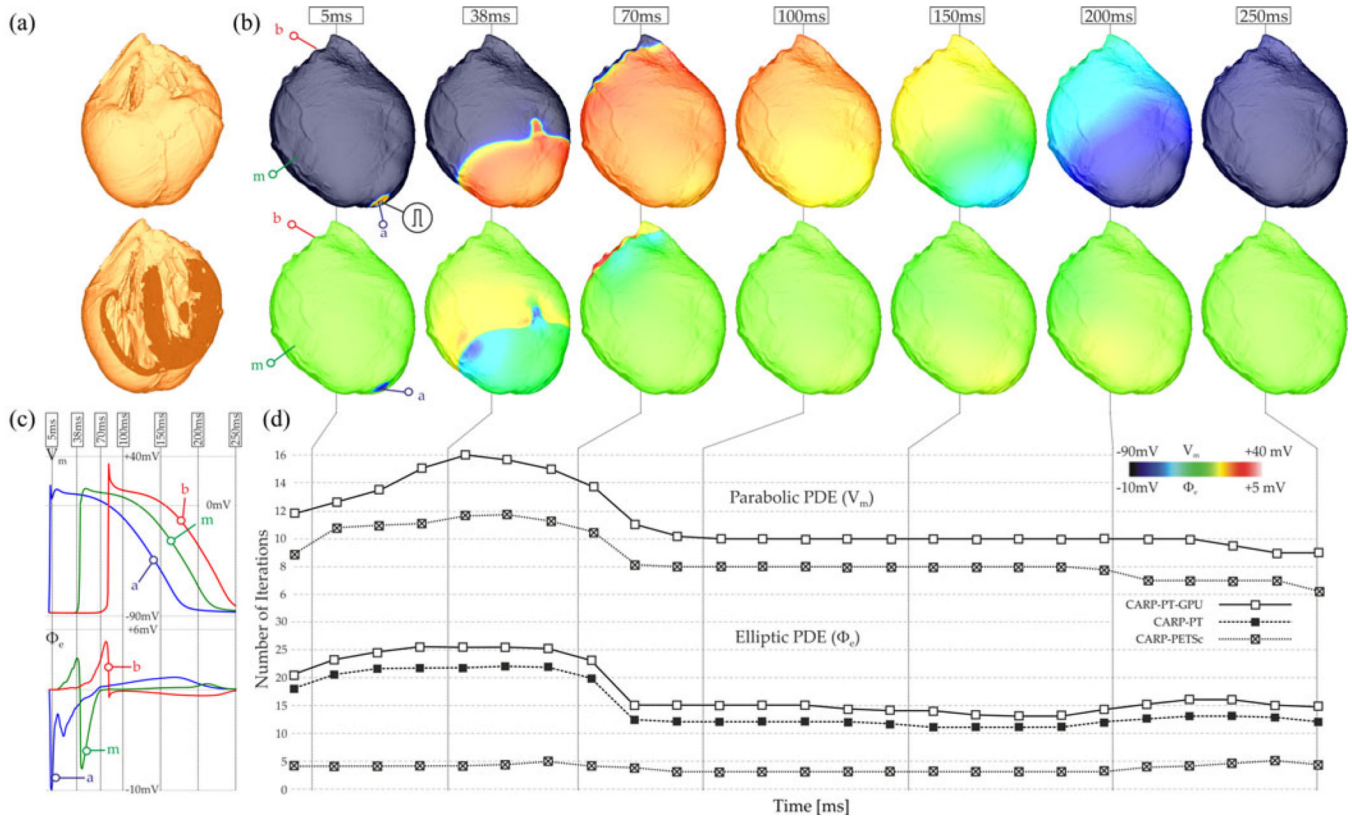

Fig. 2.

(a) Image-based high-resolution FE mesh of rabbit ventricles. Clipping plane exposes view on trabeculation and papillary muscles in the LV cavity. (b) Activation sequence used for benchmarking. Wavefront propagation is initiated by delivering a transmembrane stimulus at the LV apex. (c) Time traces are shown for V_m and Φ_e at three different sites, close to the stimulus site (blue), halfway between apex and base (green) and at the latest activating site (red) at the base of the ventricles. (d) Convergence history as a function of time.

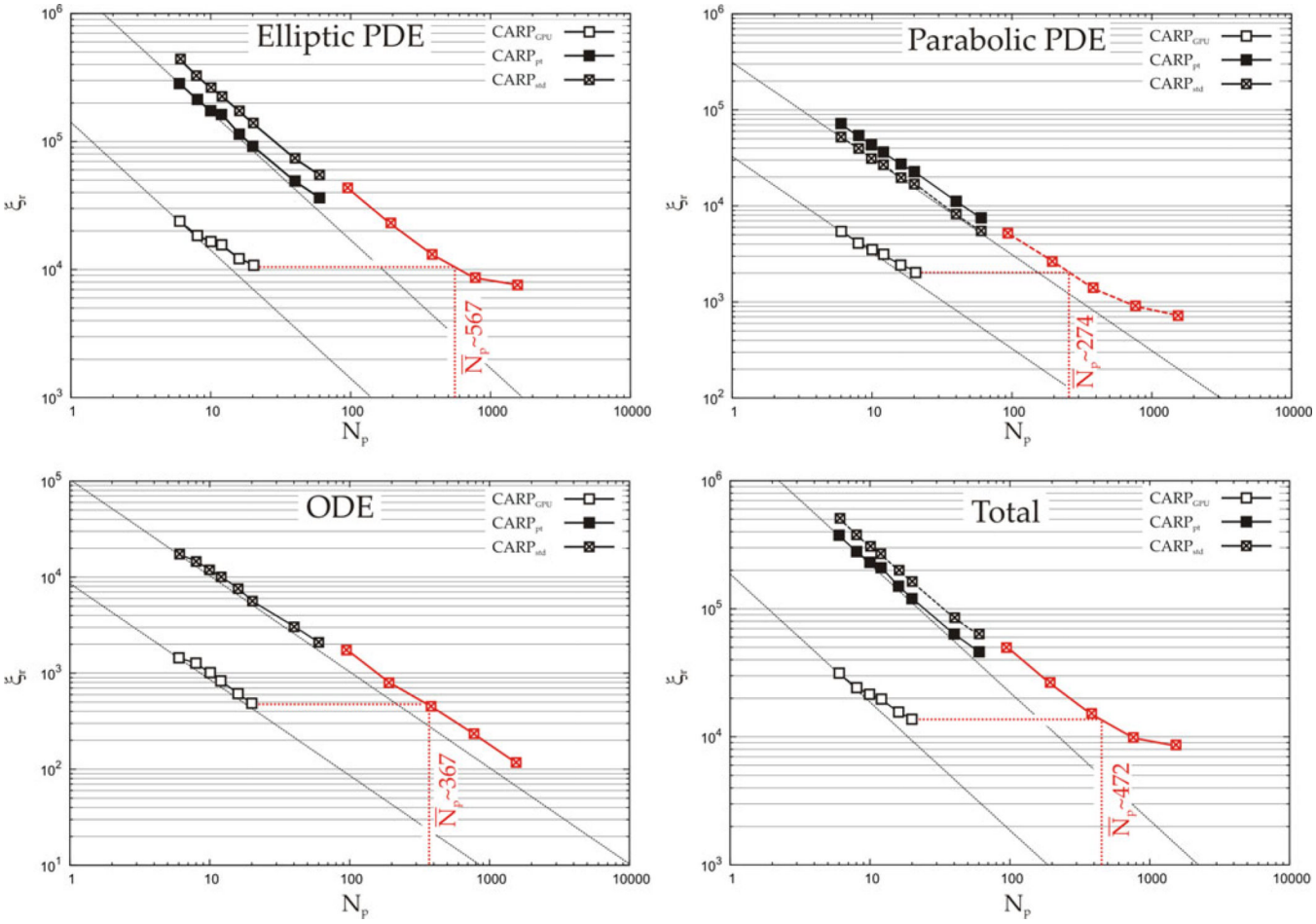

Fig. 3.

Realtime lag ξ_r as a function of N_p for the solution of elliptic PDE, parabolic PDE, ODE and overall compute time. Red traces were measured with the U.K. national supercomputing facility HECToR. Number of cores required to match fastest GPU setup $\bar{N}_p$ are indicated in each panel.

TABLE I

Benchmark Results Measured With $CARP_{std}$ Executable: Table Columns are Number of Cores N_p, Total, Elliptic, Parabolic and ODE Solvers Time and Parallel Efficiency E

	Execution Time [secs]				
N_p	Total	Elliptic	Parabolic	ODE	E
6	128127	109369.2	12981.8	4322.8	1.00
8	96285	81130.8	10031.3	3727.9	1.00
10	77286	65274.7	7651.9	2983.5	1.00
12	67359	56602.4	6860.1	2543.6	0.96
16	51069	42988.1	4837.5	1917.0	0.95
20	41748	34719.0	4312.8	1423.3	0.94
40	22457	18374.8	2021.7	777.0	0.90
60	16817	13656.3	1366.1	527.4	0.81
96	14019	10776.6	1261.7	436.7	1.00
192	8101	5700.9	676.0	200.6	0.95
384	5257	3275.9	345.4	112.5	0.84
768	3978	2153.8	225.1	58.7	0.64
1536	3746	1909.4	181.4	29.3	0.37

IEEE Trans Biomed Eng. Author manuscript; available in PMC 2013 June 30.

TABLE II

Benchmark Results Measured With $CARP_{pt}$ Executable: Table Columns are Number of Cores N_p, Total, Elliptic, Parabolic, and ODE Solver Times, Parallel Efficiency E and Speedup S

	Execution Time [secs]					
N_p	Total	Elliptic	Parabolic	ODE	E	S
6	94791	70856.8	18116.6	4354.2	1.00	1.36
8	71487	53184.1	13174.5	3727.5	1.00	1.35
10	58782	43573.6	10841.4	2986.9	0.98	1.32
12	53724	40767.6	9037.6	2557.1	0.89	1.26
16	38830	28759.6	6813.6	1926.6	0.93	1.33
20	31447	23015.6	5712.4	1419.5	0.93	1.34
40	17103	12227.1	2801.9	777.0	0.88	1.34
60	12787	9110.2	1868.1	528.9	0.81	1.35

TABLE III

Benchmark Results Measured With $CARP_{gpu}$ Executable: Table Columns are Number of Cores N_p, Total, Elliptic, Parabolic, and ODE Solver Times, Parallel Efficiency E and Speedup S

	Execution Time [secs]					
N_p	Total	Elliptic	Parabolic	ODE	E	S
6	9013	5923.8	1364.5	360.0	1.00	16.30
8	7329	4611.3	1034.2	320.5	0.96	15.60
10	6581	4132.0	882.6	249.6	0.87	14.20
12	6160	3861.1	772.2	206.9	0.79	13.42
16	5124	3039.8	613.2	154.3	0.75	12.81
20	4678	2741.2	535.0	120.5	0.68	11.76