Automated Parallel Simulation of Heart Electrical Activity Using Finite Element Method

Andrey Sozykin[1,2,3(✉)], Timofei Epanchintsev[1,2,3], Vladimir Zverev[1,3], Svyatoslav Khamzin[2,3], and Aleksandr Bersenev[1,3]

[1] Krasovskii Institute of Mathematics and Mechanics, Ekaterinburg, Russia
[2] Institute of Immunology and Physiology UrB RAS, Ekaterinburg, Russia
[3] Ural Federal University, Ekaterinburg, Russia
Andrey.Sozykin@urfu.ru

Abstract. In this paper we present an approach to the parallel simulation of the heart electrical activity using the finite element method with the help of the FEniCS automated scientific computing framework. FEniCS allows scientific software development using the near-mathematical notation and provides automatic parallelization on MPI clusters. We implemented the ten Tusscher–Panfilov (TP06) cell model of cardiac electrical activity. The scalability testing of the implementation was performed using up to 240 CPU cores and the 95 times speedup was achieved. We evaluated various combinations of the Krylov parallel linear solvers and the preconditioners available in FEniCS. The best performance was provided by the conjugate gradient method and the biconjugate gradient stabilized method solvers with the successive over-relaxation preconditioner. Since the FEniCS-based implementation of TP06 model uses notation close to the mathematical one, it can be utilized by computational mathematicians, biophysicists, and other researchers without extensive parallel computing skills.

Keywords: Heart simulation · Finite element method · Scalability · Krylov subspace methods · FEniCS · Parallel computing

1 Introduction

The mechanical contraction of a heart, which pumps the blood throughout the entire body, is caused by its electrical activity. In order to understand how the heart works, it is important to be able to simulate cardiac electrical processes. However, heart simulation is a complex multilevel (cell-tissue-organ) modeling task [9] that is very computationally intensive. Therefore, for a fast and accurate heart simulation, parallel computing is required.

The work is supported by the RAS Presidium grant I.33P "Fundamental problems of mathematical modeling," project no. 0401-2015-0025. Our study was performed using the *Uran* supercomputer of the Krasovskii Institute of Mathematics and Mechanics and computational cluster of the Ural Federal University.

J. Carretero et al. (Eds.): ICA3PP 2016 Workshops, LNCS 10049, pp. 365–372, 2016.
DOI: 10.1007/978-3-319-49956-7_29

However, the parallel heart simulation is impeded by the two obstacles. First, it requires a deep knowledge in a number of modern computational architectures and parallel programming technologies, which most of biophysicists do not possess. Secondly, sophisticated multilevel models are hard to implement in code, especially when the complex optimization for modern computational architectures is required. As a result, multilevel simulation software is very hard to support, while models and computational architectures are changing constantly.

Nowadays, automated scientific computing frameworks that allow software development using near-mathematical notation are becoming popular. Recently, the performance of such frameworks was significantly improved by the use of just-in-time compilers, highly efficient mathematical libraries, parallel computing, etc.

We propose an approach to simulation of the heart electrical activity using the scientific computing framework FEniCS [6], which provides the ability to automatically solve partial differential equations (PDE) using the finite element method (FEM) on MPI clusters. We use the FEniCS framework to study the space propagation of the membrane potential alternation over the left ventricle (LV) of a human heart using the *ten Tusscher–Panfilov* (TP06) cell model [10].

Due to the fact that FEM produces sparse matrices, computations heavily depend on the degree of sparsity because it allows to use various optimization techniques such as the *compressed row storage*, the *sparse matrix-vector multiplication*, or custom approaches [3]. Various models produce matrices with different degrees of sparsity. Hence, linear solvers and appropriate preconditioners demonstrate different results on parallel systems for various models. We evaluated which combination of the linear solver and the preconditioner is the most suitable for the simulation of a cardiac electrical activity using the *TP06 model*.

2 Model of the Heart Electrical Activity

There are many mathematical models of cardiac electrical activity. However, all of them contain the description of the *action potential* (AP), which is the difference of the potential between the intra- and extracellular space. We adopted the TP06 model [10,11] for the simulation of the electrical activity in the LV of a human heart. This model uses the reaction-diffusion equations to describe the space and time evolution of the action potential (V):

$$C_m \cdot \frac{dV}{dt} = \nabla \cdot (D \nabla V) - I_{ions}, \tag{1}$$

$$\frac{dS}{dt} = g(V, S), \tag{2}$$

where C_m is the capacitance of a cell membrane, D is the 3×3 diffusion matrix, I_{ions} is the sum of the ionic currents, S is the vector of the model variables that govern the ion currents, and g is the vector-valued function that describes the time evolution of each variable. The boundary conditions provide the LV electrical isolation.

On the intracellular level, the electrical potential arises from a very complicated interaction among ionic currents and cell organelles (organized structures in cells). The *TP06* model contains the equations that describe how does the state of ion channels change with time and the kinetics of intracellular concentrations of calcium, sodium and potassium, extracellular potassium, the kinetics of calcium complexes, and calcium kinetics in the organelles. All of this processes are described by 18 phase variables. System (1)–(2) is defined at each point of the heart tissue, and, consequently, we should solve it for each node of the computational mesh.

Thus, the *TP06* model is a nonlinear system of partial and ordinary differential equations (ODE) that cannot be solved analytically and, hence, must be solved on a computer using numerical techniques. This task is highly computationally intensive due to the big number of equations in the 3D domain and the stiffness of the *TP06* model.

3 FEniCS and the Finite Element Method

FEM provides a powerful methodology for discretizing differential equations, however, it produces algebraic systems the solution of which is also a challenge. Linear solvers must handle sparsity and possible ill-conditioning of the algebraic systems. In addition, modern solvers should also be able to use parallel computing systems efficiently. The FEM implementation in FEniCS is intended to automate a PDE solution. In particular, FEniCS relies on the automation of discretization, discrete solution, and error control. FEniCS provides two approaches for a PDE solution: direct and iterative. Iterative solution is more efficient because it uses less memory and is easier to parallelize [6].

The FEniCS framework is a collection of software components for the formulation of variational forms (UFL [1]), the discretization of variational forms (FIAT, FFC [4]), and the assembly of the corresponding discrete operators (UFC, DOLFIN [7]). To solve a problem, FEniCS uses several highly efficient parallel algebra backends, such as PETSc and Hypre. UFL is a domain-specific language designed for convenient and understandable formulation of variational forms using the near-mathematical notation. The discretization of variational forms is done by generation of arbitrary order instances of the Lagrange elements on lines, triangles, and tetrahedra (FIAT), and compilation of efficient low-level C++ code that can be used to assemble the corresponding discrete operator (FFC). The assembly of the discrete operators (tensors) is crucial for acceleration on parallel computing systems. The idea is to split the mesh among processing units, compute the local matrix, and insert the values back into the global matrix. The FEniCS team designed the local-to-global mapping algorithm [4] to map values between the local and global matrices.

The most computationally intensive task is solving the local linear system. Hence, optimization of this step by selecting appropriate linear solver and preconditioner can provide a significant computation speedup.

4 Model Implementation in FEniCS

In order to implement the *TP06* model in FEniCS, we transformed the nonlinear system (1)–(2) into a linear one, which let us use iterative solvers. The transformation was performed with the help of the first order operator splitting scheme (the Marchuk–Yanenko method) [5]. The scheme of computing $V(t_n)$ and $S(t_n)$ can be described as follows. Let us assume that we have already calculated the values of $V(t)$ and $S(t)$ for $t < t_n$. In order to find the values of $V(t_n)$ and $S(t_n)$, we solve Eq. (3),

$$\begin{aligned} \frac{dV^*}{dt} &= D\nabla V^*, V^*(t = t_{n-1}) = V(t_{n-1}), t \in [t_{n-1}, t_n], \\ \frac{dV^{**}}{dt} &= I_{ions}, V^{**}(t = t_{n-1}) = V^*(t_n), \\ \frac{dS^{**}}{dt} &= g(V^*, S^{**}), S^{**}(t = t_{n-1}) = S(t_{n-1}). \end{aligned} \tag{3}$$

First, we solve the diffusion PDE. After that we have to find the solution of the ODE system for cell ionic currents. We get the final values of $V(t_n)$ and $S(t_n)$ according to the rules $V(t_n) = V^{**}(t_n)$ and $S(t_n) = S^{**}(t_n)$. This method is also known as the method of splitting into physical processes. The disadvantage of the approach is the necessity to use a very small integration time step (0.0005 s) in order to capture the fast electrochemical processes.

The model was implemented in the Python language using UFL. The code fragment for the diffusion PDE problem formulation is presented in Listing 1.1. First, a finite element mesh is created and loaded from the file. After that the discrete function space for AP is defined. FEniCS uses the term *trial function* to specify the unknown function that should be approximated (the variable *v* contains a trial function and the *v0* variable contains the initial values). The next step is to define the linear variational problem for the diffusion equation. Lastly, the PDE solver is created.

Listing 1.1. Formulation of the diffusion PDE variational problem

```
mesh = Mesh()
# Code for loading mesh from the file
# Building function space for action potential
Space_AP = FunctionSpace(mesh,"Lagrange", lagrange_order)
# Define the PDE Problem
v = TrialFunction(Space_AP)
v0 = Function(Space_AP)
PdePart = (1.0/dt)*inner(v - v0, q1)*dx \
          - (-inner(D*grad(v),grad(q1)))*dx
PDEproblem = LinearVariationalProblem(lhs(PdePart),
    rhs(PdePart), v, bcs=bcs)
# Creating the PDE solver
PDEsolver = LinearVariationalSolver(PDEproblem)
```

Listing 1.2 demonstrates the code fragment for solving differential equations. The first step in the *for* loop solves the diffusion PDE using the PDEsolver. After that, the values of the state variables and AP, which was computed on the previous step, are stored. Next, the ODE system describing the cell ion currents is solved using the ODEsolver. There is no need for explicit, manual parallelization because the parallelization is provided by FEniCS. In addition to parallel computation, FEniCS provides the parallel output, during which each process writes its part of the data to a single file.

Listing 1.2. Solving the differential equation systems

```
for t in time_range[1:]:
  # Solving diffusion equation
  assign(v0, v)
  PDEsolver.solve()
  # Solving cell equations
  assign(ode_vars0, ode_vars)
  assign(ode_vars0.sub(0), v)
  ODEsolver.solve()
  # Storing data if necessary
  if steps
      v_file << (v, t)
  steps += 1
```

5 Performance Evaluation

During the experiments, we simulated the electrical activity of the human heart LV using the asymmetric anatomical model that was previously developed in our group [8] (an example of LV 3D mesh is presented in Fig. 1). We used the tetrahedral mesh with the length of the tetrahedrons from 2 to 4 mm; the mesh contained 7178 points and 26156 tetrahedrons. The GMSH software [2] was used for the initial mesh generation. Next, the mesh was converted by the DOLFIN module to the HDF5 format in order to enable parallel I/O operations.

The initial simulation conditions were the activation of a small part of LV near the apex (the potential is greater than 40 millivolt). The simulation duration was 0.3 s of physical time, because after this period the electrical activity tends to the equilibrium state in absence of an external stimulus.

The experiments were carried out on the *Uran* supercomputer of the Krasovskii Institute of Mathematics and Mechanics with the following computational nodes configuration: 2 x Intel Xeon CPU X5675 CPU, 192 GB RAM, Infiniband DDR interconnect, CentOS 7 operating system. The FEniCS version 1.6.0 was used.

The *TP06* model implementation was executed on the *Uran* supercomputer in parallel using various numbers of CPU cores, from 1 to 240. We used Krylov parallel linear solvers and preconditioners available in FEniCS (Table 1). The

Table 1. Parallel Krylov solvers and preconditioners available in FEniCS

Solver	Preconditioner
Biconjugate Gradient Stabilized	Algebraic Multigrid (amg)
Method (bicgstab)	Default preconditioner (Block Jacobi)
Conjugate Gradient method (cg)	Hypre Algebraic Multigrid (hypre_amg)
	Successive Over-relaxation (sor)

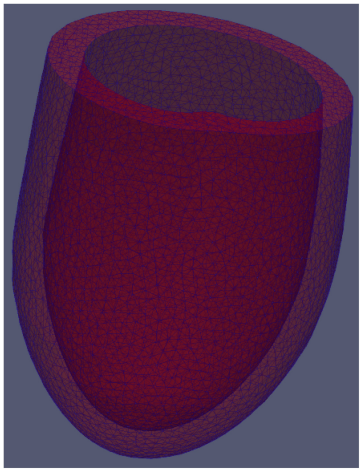

Fig. 1. An example of 3D mesh of the left ventricle (asymmetric model)

Table 2. The simulation time (minutes) using for various numbers of CPU cores

Solver and Preconditioner	Number of CPU cores									
	1	12	36	60	96	120	156	180	216	240
bicgstab + amg	1937	275	109	74	50	44	36	33	29	28
cg + amg	1930	242	96	65	44	39	35	32	29	27
bicgstab + default	1915	214	82	53	38	34	28	25	22	20
cg + default	1896	224	86	53	35	30	27	24	21	20
bicgstab + hypre_amg	1947	248	98	67	45	39	35	33	29	28
cg + hypre_amg	1925	268	106	71	49	43	35	32	29	27
bicgstab + none	2021	263	92	61	40	34	30	27	23	22
cg + none	1963	247	95	63	42	37	29	26	23	21
bicgstab + sor	1845	208	79	53	34	30	26	24	20	19
cg + sor	1839	220	85	55	36	32	26	23	20	19

simulation time with various combinations of solvers and preconditioners is presented in Table 2, the achieved speedup is demonstrated in Fig. 2.

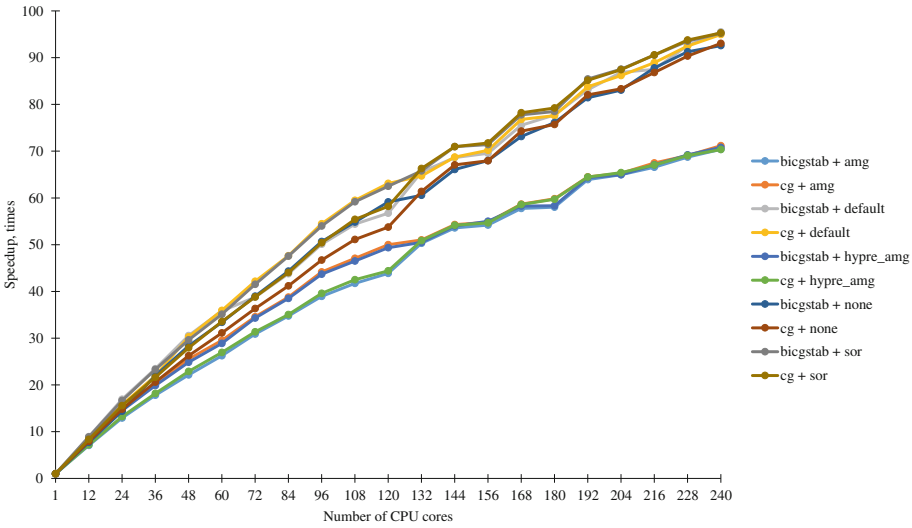

Fig. 2. Simulation speedup depending on the number of CPU cores

6 Discussion

The experiments demonstrated that the FEniCS-based *TP06* model implementation provides acceptable performance and good scalability. The best result was achieved using the *conjugate gradient method* and the *biconjugate gradient stabilized method* solvers with the *successive over-relaxation* preconditioner: 19 min of the simulation time, 95 times speedup using the 240 CPU cores.

Choosing the appropriate combination of the solver and the preconditioner is an important task. The best combination from our experiments (Table 1) provided 30% more performance on 240 CPU cores than the worst one (the *biconjugate gradient stabilized method* solver with the *algebraic multigrid* preconditioner). To save space, we presented in the paper only the best experiment results. FEniCS includes other solvers and preconditioners not listed in Table 1. Hence, in practice, the difference in performance of the best combination and other solvers and preconditioners available in FEniCS could be more than 30%.

As the number of CPU cores increases, the preconditioner's influence on performance becomes greater than the solver's. When we conducted the simulation on 132 CPU cores or more, there was no tangible difference in performance between different solvers working with the same preconditioner (Table 2).

7 Conclusion

The created implementation of the *TP06* model uses the near-mathematical notation provided by the FEniCS framework. As a result, computational mathematicians and biophysicists can use this implementation for experimenting with

the model. They can easily modify the model parameters, the initial activation conditions, and even change the model itself. Despite the usage of the near-mathematical notation, our implementation provides an acceptable performance and scales well. The possible direction of the future work is to use the *TP06* model implementation for simulation of complicated processes in LV that can cause heart diseases, such as scroll wave dynamics. Another important task is to implement the model of mechanical heart activity using FEniCS and provide the ability to simulate electro-mechanical function of the heart.

References

1. Alnæs, M.S., Logg, A., Ølgaard, K.B., Rognes, M.E., Wells, G.N.: Unified form language. ACM Trans. Math. Softw. **40**(2), 1–37 (2014)
2. Geuzaine, C., Remacle, J.F.: Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int. J. Numer. Methods Eng. **79**(11), 1309–1331 (2009)
3. Jansson, N.: Optimizing sparse matrix assembly in finite element solvers with one-sided communication. In: Daydé, M., Marques, O., Nakajima, K. (eds.) VECPAR 2012. LNCS, vol. 7851, pp. 128–139. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38718-0_15
4. Kirby, R.C., Logg, A.: A compiler for variational forms. ACM Trans. Math. Softw. **32**(3), 417–444 (2006)
5. Li, Y., Chen, C.: An efficient split-operator scheme for 2-D advection-diffusion simulations using finite elements and characteristics. Appl. Math. Model. **13**(4), 248–253 (1989)
6. Logg, A., Mardal, K.A., Wells, G.: Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book. Springer Science & Business Media, Heidelberg (2012)
7. Logg, A., Wells, G.N.: DOLFIN: automated finite element computing. ACM Trans. Math. Softw. (TOMS) **37**(2), 1–28 (2010)
8. Pravdin, S.F., Berdyshev, V.I., Panfilov, A.V., Katsnelson, L.B., Solovyova, O., Markhasin, V.S.: Mathematical model of the anatomy and fibre orientation field of the left ventricle of the heart. Biomed. Eng. Online **54**(12), 21 (2013)
9. Kerckhos, R.C.P., Healy, S.N., Usyk, T.P., McCulloch, A.D.: Computational methods for cardiac electromechanics. Proc. IEEE **94**, 769–783 (2006)
10. Ten Tusscher, K.H., Panfilov, A.V.: Alternans and spiral breakup in a human ventricular tissue model. Am. J. Physiol. Heart Circulatory Physiol. **291**(3), H1088–H1100 (2006)
11. Ten Tusscher, K.H., Panfilov, A.V., et al.: Organization of ventricular fibrillation in the human heart. Circulation Res. **100**(12), e87–e101 (2007)