Simulating Electrical Action Potential Propagation in the Heart with Parallel Computation

Christa Erwin

CRPC-TR99806-S
August 1999

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Submitted November 1999

Simulating Electrical Action Potential Propagation in the Heart with Parallel Computation

Christa Erwin

Summer 1999

1 Motivation

One of the central questions in the field of cardiac dynamics is the mechanism of the decay of ventricular tachycardia, characterized by spiral waves of electrical activity, to fibrillation, characterized by "incoherent" electrical wave behavior, resulting in failure of the heart's pumping action. This problem is being chipped away at from various perspectives by various groups in physiology, mathematics, physics and engineering. Numerical simulation, coupled with experimental feedback, is a necessary and powerful tool in this area, as in many other growing areas of computational science.

The focus of this summer project was to explore the effect of geometry and fiber architecture of the left ventricle on electrical wave dynamics. This question was motivated by earlier numerical experiments performed in a rectangular slab model of the ventricle, which showed that the rotating anisotropy inherent in cardiac tissue could lead to wave instability. This rotating anisotropy is pictured in Figure 1 (from [4] and [5]). Our goal was to construct a minimally realistic geometrical model of the left ventricle and to simulate electrical wave propagation in this three-dimensional model.

Dissection results reveal a *nested* layered geometry for the left ventricle, where a single macroscopic muscle fiber bundle starting at the basal plane outside the midwall (toward the epicardium) traverses down toward the apex on an outer surface, and at some point before reaching the apex, changes direction, traverses back along an inner surface reinserting at the basal plane inside the midwall (toward the endocardium) [6]. The simple fact that electrical propagation along a fiber is several times faster than perpendicular to it suggests that this nested architecture can be important in the propogation of electrical waves in the heart. In particular, transmural propagation can still be achieved in a nested geometry even when propagation perpendicular to the fiber is weakened or cut off.

A geometrical model of the left ventricle has been developed by Sima Setayeshgar in the Applied Mathematics Department at the California Institute of Technology [1], based on previous works [2], [3]. In this model, the fiber surfaces are given by nested cones, and the fiber trajectories by geodesics on these surfaces, consistent with experimental observation. Figure 2 shows the fiber trajectories on one fiber surface.

The objective of this summer project was to investigate the effect of the realistic (a) nested conical geometry and (b) fiber architecture of the left ventricle on action potential propagation, using the above model. In the process, the project provided an introduction to the following:

- Numerical methods for scientific computing, in particular, finite difference solutions to nonlinear partial differential equations
- Parallel computing using MPI
- HTML form submission and CGI programming as a front-end for running parallel code on Caltech's Boewulf machine.

2 Model

The FitzHugh-Naguomo (FHN) model, a two-variable "caricature" of action-potential propagation in the heart, was used. We included the effect of the anisotropic conductivity of the three-dimensional mycocardium, in both the rectangular slab and conical geometries. The governing equations are:

$$\frac{\partial u}{\partial t} = f(u,v) + \boldsymbol{\nabla} \cdot (\boldsymbol{D} \cdot \boldsymbol{\nabla} u)\,, \tag{1}$$

$$\frac{\partial v}{\partial t} = g(u), \tag{2}$$

where $\boldsymbol{D}$ is the diffusion tensor, containing information about the fiber architecture, and the FHN kinetics are given by:

$$f(u,v) = 3\,u - u^3 - v \tag{3}$$
$$g(u) = \epsilon(u - \delta). \tag{4}$$

u is the action potential, and v is equivalent to a "gating variable", (describing the action of membrane pumps in transporting ions), and ϵ and δ are adjustable model parameters. No-flux boundary conditions were used at all physical boundaries.

3 Implementation

3.1 Numerical scheme

In addition to using a simple model, a simple numerical scheme was implemented in an effort to keep the computational effort tractable given the geometrical complexity of the problem. Second order finite-differencing with explicit Euler time-stepping was used in the numerical simulations. The stability restriction on the time step is onerous in this problem. As an example, it takes approximately 20 CPU hours on a single 450Mz Pentium Pro workstation to generate a single spiral period. Clearly, to use these numerical simulations as a practical experimental tool, speedup is needed. Hence, the code was parallelized.

3.2 MPI

Problems involving a large amount of computation that can be divided up into pieces are highly amenable to parallelization. Message Passing Interface (MPI), is a library of functions that can be called from within a C, C++, Java or Fortran program. These functions allow a programmer to take a serial program (code written for one computer) and parallelize the program (code written to be run on multiple computers simultaneously). The MPI functions allow for the data transmission between computers (also called processes or nodes), which is usually required to complete the overall computational task.

As an introduction to parallel computing with MPI, we started with parallelizing the code in the rectangular slab geometry. In this case, the electrical potential is represented by a three-dimensional Cartesian matrix. The matrix is divided up among a group of processors (also called nodes), introducing artificial boundaries between nodes. Since centered finite-differencing requires knowing the solution at the points to the right and left of each point, data on the artificial borders of each node must be communicated between nodes at every time step. MPI functions were used for this message passing (point-to-point communication).

MPI was also used to allow one process to take care of all the input and output. A single node (node 0) was given the task of all reading in from and printing out to files. MPI was used to distribute the input parameters to all the processes as well as to gather the data from all processes by node 0 for generating output, as well as other computing tasks requiring data from all nodes (collective communication).

One major task in the implementation of parallelization with MPI is attention to memory allocation. MPI function calls which pass information between nodes require the starting address of the data being passed. More data can be passed in fewer function calls, saving run time, if it is located contiguously in memory. For memory allocation, we used Numerical Recipes [7] utility routines. In particular, it was important to understand in detail how memory is allocated to a $3d$ tensor, `array3d`:

$$\texttt{array3d} = \texttt{d3tensor}(0, \mathrm{nx}+1, 0, \mathrm{ny}+1, 0, \mathrm{nz}+1) \tag{5}$$

where:

```
double ***d3tensor(long nrl, long nrh, long ncl, long nch,
   long ndl, long ndh)
/* allocate a double 3tensor with range:
   t[nrl..nrh][ncl..nch][ndl..ndh] */
{
   long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
   double ***t;

   /* allocate pointers to pointers to rows */
   t=(double ***) malloc((size_t)
      ((nrow+NR_END)*sizeof(double**)));
   if (!t) nrerror("allocation failure 1 in d3tensor()");
t += NR_END;
t -= nrl;

/* allocate pointers to rows and set pointers to them */
t[nrl]=(double **)malloc((size_t)
   ((nrow*ncol+NR_END)*sizeof(double*)));
if (!t[nrl]) nrerror("allocation failure 2 in d3tensor()");
t[nrl] += NR_END;
t[nrl] -= ncl;

/* allocate rows and set pointers to them */
t[nrl][ncl]=(double *)malloc((size_t)
   ((nrow*ncol*ndep+NR_END)*sizeof(double)));
if (!t[nrl][ncl]) nrerror("allocation failure 3 in d3tensor()");
t[nrl][ncl] += NR_END;
t[nrl][ncl] -= ndl;

for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
for(i=nrl+1;i<=nrh;i++) {
   t[i]=t[i-1]+ncol;
   t[i][ncl]=t[i-1][ncl]+ncol*ndep;
   for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
}

/* return pointer to array of pointers to rows */
return t;
}
```

In this function, memory for the entire three-dimensional array is contiguous if x is the slowest changing index and z the fastest. Thus, it is most convenient to parallelize in the x-direction, allowing entire yz planes to be passed between nodes by referring to the beginning address.

The parallelized code was run on the Beowulf computer at the Center for Advanced Computing Research. This computer has 64 compute nodes, each with a 300Mz Pentium Pro processor, 128Mb RAM, and 3.1Gb of hard disk memory.

3.3 Performance results

The speedup due to parallelization was measured using wall clock time and calculated according to:

speedup = serial code run time / parallelized code run time

Timing was tested for 4, 8, 16, 32, and 64 nodes, evenly dividing the number of points in the x direction (for equal load distribution among nodes). The timing results were obtained from running the code for a sufficiently large number of iterations, such that the time spent on initialization constituted a minor fraction of the total compute time. The performance of the parallelized code ranged from 1.7 times faster for 4 nodes to 10.4 times faster for 64 nodes (see Figure 3). It appears that the "optimal" number of nodes for this problem is 32, since there is not a significant gain between 32 and 64 nodes. We can identify one aspect of the computation (calculation of the tip trajectory) which can be further parallelized.

The Jumpshot program which comes with MPI software was also used to look at timing. Jumpshot provides a graph of the CPU time spent in each MPI function call. This gave a better understanding of which functions took up a small amount of program time (*eg.*, `MPI_Pack()`, `MPI_Unpack()`, `MPI_Bcast()`, used to distribute initial parameters and conditions), which took up a moderate amount of time (*eg.*, MPI_Sendrecv(), used to pass border data), and which took the most time (more than expected!) (*eg.*, MPI_Gather(), used to gather xy planes of data for computing the spiral tip at regular time intervals). Although the Jumpshot program provides a relative measure of the time spent in various MPI functions, it does not provide concrete numbers with which to calculate speed-up. In general, we found it not to be as useful as our "home-made" timing calls.

3.4 HTML interface

A web page interface for the parallelized heart code was developed. This page allows for an interactive way to run the code and view the output. The web page provides a form for the user to enter the numerical input parameters and the number of nodes. A CGI script, written in C (utilizing a CGI C library of functions) accesses the variables from the form and makes system commands to run the parallel code and to display the output. The web page can be found at:

http://naegling.cacr.caltech.edu/~ simas/

At present, the page can only be accessed by approved computers, including computers on the network of the Caltech Applied Math department as well as the Center for Advanced Computing Research.

4 Future Work

Currently the serial code for the conical model of the left ventricle has been written. I have made substantial progress in parallelizing it, although this task is not fully complete. The parallelization scheme for the code in the rectangular geometry in Cartesian coordinates follows through very closely for the code in the conical geometry in spherical coordinates. The added complication in the latter case comes from allowing for a variable mesh in the azimuthal direction as a function of the radial coordinate. The speedup achieved from parallelization of the conical code will allow investigation of the effect of geometry and fiber architecture on the dynamics of electrical waves in the ventricle.

In additional to parallelization, the numerical scheme can be improved from fully explicit to semi-implicit. We should note that the cross-derivative terms arising from anisotropic diffusion would still need to treated explicitly, even if an ADI (alternating direction implicit) scheme is used for the diagonal diffusion terms.

The web interface can be further improved to provide a nice front-end for organization and submission of runs to the Caltech Boewulf. Estimate of the run time, as well as output file display and manipulation are among the interactive features that will soon be added.

Numerical simulation in this minimally realistic geometry is almost as straightforward and accessible as that in the commonly used rectangular slab, in contrast with fully realistic whole heart finite element models. The hope is that it will be a scientifically sound and numerically tractable tool for exploring electrical wave activity in the ventricle.

5 Acknowledgements

Thanks to Tom Gottschalk at the Center for Advanced Computing Research for his sharing of resources and information regarding the use of supercomputers and programming with MPI. Thanks to Sima Setayeshgar for assistance in programming and excellent mentorship.

References

[1] S. Setayeshgar, preprint.

[2] C. S. Peskin, Communications on Pure and Applied Mathemathics **42**, 79 (1989).

[3] C. S. Peskin and D. M. McQueen, in *Case Studies in Mathematical Modeling: Ecology, Physiology, and Cell Biology*, H. J. Othmer, F. R. Adler, M. A. Lewis, and J. C. Dallon, eds., Prentice-Hall, Englewood Cliffs NJ, 309 (1996).

[4] D. D. Streeter, D. P. Patel, J. Ross and E. H. Sonnenblick, Circ. Res. **24**, 339 (1969).

[5] D. D. Streeter, W. E. Powers, M. A. Ross and F. Torrent-Guasp, in *Cardiovascular System Dynamics*, J. Baan, A. Noordergraaf and J. Raines, eds., Cambridge, M. I. T. Press, 73 (1978).

[6] C. E. Thomas, Americal Journal of Anatomy, **101**, 17 (1957).

[7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C* (Cambridge Univ. Press, 1992).

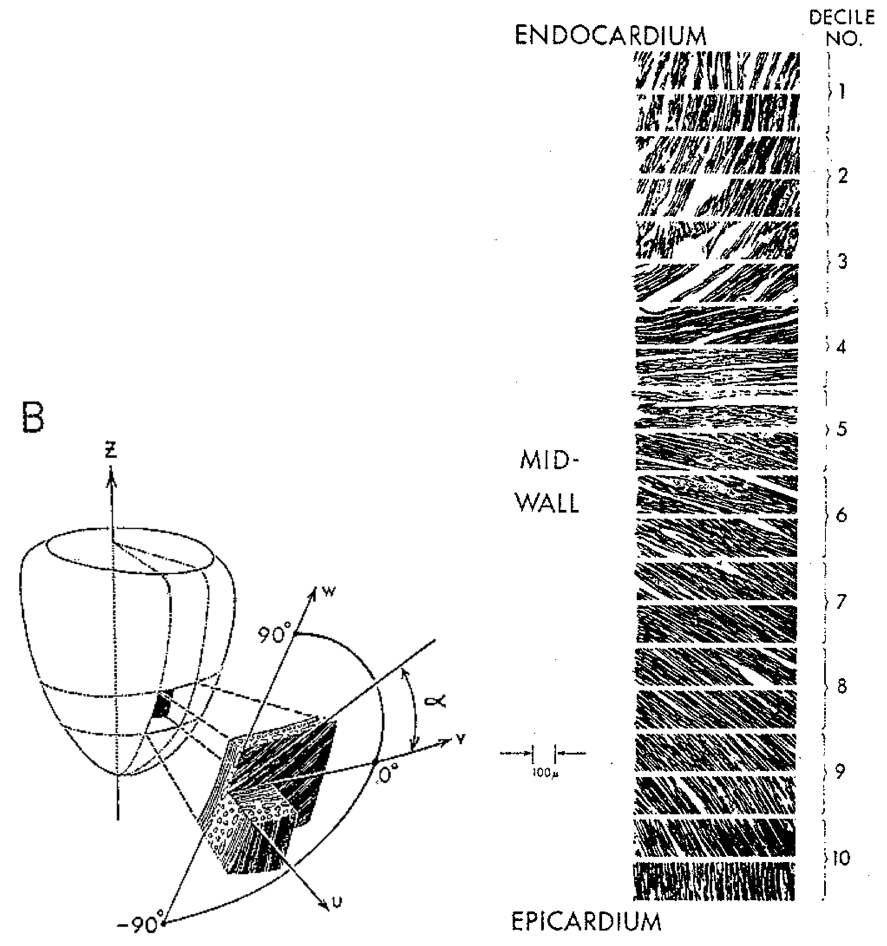

Figure 1: Left: Rectangular slab of the left ventricle. Right: Fiber direction in successive slices.

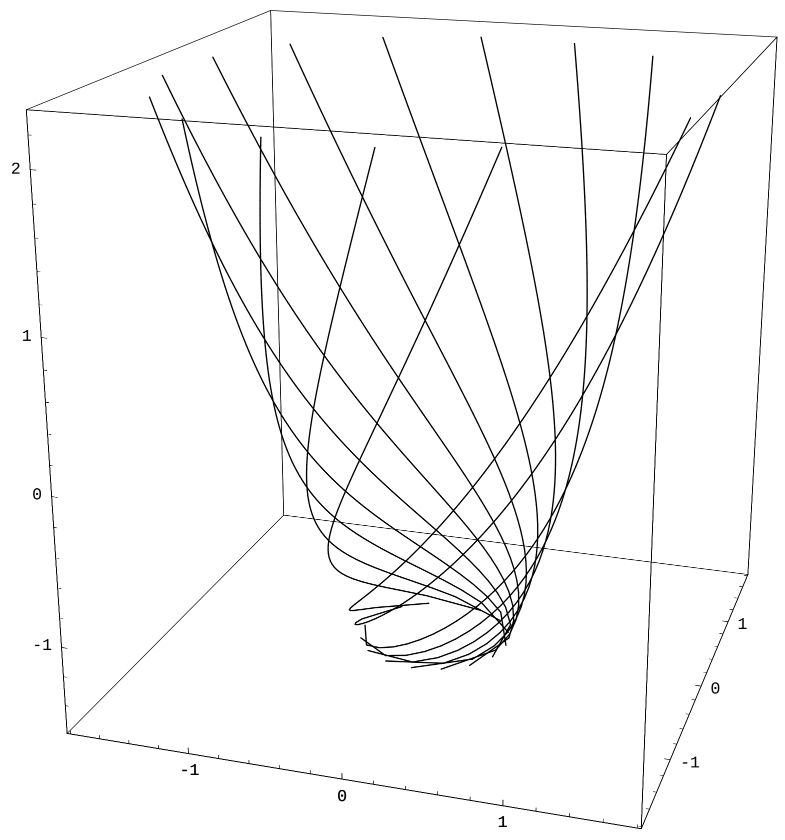

Figure 2: Fiber trajectories on a single conical surface.

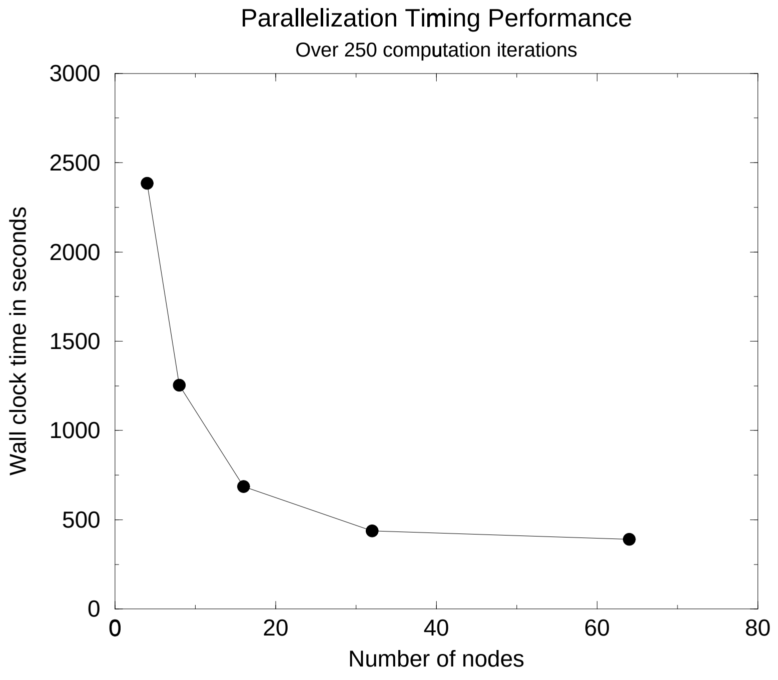

Figure 3: Parallelization timing performance.