SIMULATIONS OF UTERINE ELECTRICAL ACTIVITY USING PARALLEL COMPUTING

Tanguy Hedrich[(a)], Jeremy Laforet[(b)], Catherine Marque[(c)]

[(a)(b)(c)]CNRS UMR 7338
Biomecanique et Bioingenierie
Universite de Technologie de Compiegne
60200 Compiegne, France

[(a)]thedrich@etu.utc.fr, [(b)]jlaforet@utc.fr, [(c)]cmarque@utc.fr

ABSTRACT
The uterine electrical activity can be realistically modeled by representing the principal ionic dynamics at the cell level, the propagation of electric activity at the tissue level.

A simplified model based on the physiology was already presented (Laforet 2011). We were able to simulate 0-dimension to 2.5-dimension grids of muscle cells.

In this study, we implemented this model as an easy-to-use software using oriented-object python, under an open-source license. We developed a parallel integration based on shared memory by using python standard libraries to improve the computational effectiveness of the simulations.

The execution times of parallel computed simulations are up to 25 times less than the serial integration. We observe an effect of the size and the dimension of the grid on the computation time per cell.

Thanks to this improved computation time, further studies will provide bigger and more realistic simulations in a reason- able time.

Keywords: Simulation, Electrophysiological signals, uterus, Python

1. INTRODUCTION

The sequence of contraction and relaxation of the myometrium results from the electrical activity associated to the generation and propagation of cellular action potential bursts. The uterine electrical activity can be measured noninvasively on the abdomen by standard electrodes and it is referred to as electrohysterogram (EHG) [2].

At the myometrial level, the cellular action potential generation and propagation have been previously modeled as function of a large number of electro-physiological parameters related to ionic dynamic (Rihana 2009). This model is physiology-based and has been demonstrated to be representative of the main ionic mechanisms responsible for the genesis and evolution of the uterine electrical activity (Rihana 2009). However, this model is computationally demanding and therefore unsuitable for a direct integration in a multi-scale approach.

In view of this challenging objective, we have defined a new simplified physiology-based model of the electrical activity generation at the cell level and of the electrical propagation at tissue level, based on the full model described in (Laforet 2011).

Unfortunately, even the simplified model is still computationally demanding for large amount of cells. To step closer to obtain full organ simulations in a decent time, we present a parallel implementation of the models integration. We show
how this can help us reduce the computation time by a factor up to 24.

2. MODEL

In view of this challenging objective, we define a global multi-scale model, from the myometrial cell to the skin surface (Laforet 2011). This model is physiology based and takes into account the following levels:

- myometrial cell: generation of the electrical activity.
- uterine tissue: propagation of this activity from cell to cell.
- organ level: propagation of the electrical field thought the tissue layers from the uterus to the skin surface (Rabotti 2010).
- abdominal level: recording on the skin surface by an electrode grid.

In this paper, we focus on the improvements of the computational effectiveness of the uterine muscle model, as it is the most time consuming step within the global model. We previous presented a simplification of the complete model in (Laforet 2011). Starting from the 6 variables uterine cell reduced model (referred as Red6 model) (Rihana 2009), we obtain a 3 variables model (referred as Red3) defined as follow:

$$\frac{dV_m}{dt} = \frac{1}{C_m}\left(I_{stim} - I_{Ca} - I_K - I_{KCa} - I_{leak}\right) \quad (1)$$

$$\frac{dn_K}{dt} = \frac{h_{K_\infty} - n_K}{\tau_{n_K}} \quad (2)$$

$$\frac{d\left[Ca^{2+}\right]}{dt} = f_c\left(-\alpha I_{Ca} - K_{Ca}\left[Ca^{2+}\right]\right), \quad (3)$$

with Vm the transmembrane potential, nK the potassium activation variable, and [Ca2+] the intracellular calcium concentration.

The Red3 model is generally 50% faster than the Red6 model and will be used as default model in this work. However, the Red6 will be kept for studies on small number of cells when requiring a higher accuracy.

At the tissue scale, the communication between the my- ometrial cells through gap-junctions is modeled by the spatial diffusion of the electrical potential over the cells.

3. IMPLEMENTATION

We implemented both Red6 and Red3 models using Python (version 2.7) in an object oriented approach. The code will be published under an open-source license and its required dependences are only common Python modules beside the standard library (Numpy and Scipy). It can be executed under Mac-OS X , Windows, and GNU/Linux systems.

3.1. Models

First, we defined a generic cell model class from which Red6 and Red3 classes inherit. It enables us to define all common elements in the models only once, in the parent class. This structure is illustrated in figure 1.

At the tissue level, the cells are arranged into a Cartesian grid. This tissue model can be 0D (a single cell), 1D (a cable-like line of cells), 2D (a flat surface), or 2.5D (a flat surface with non-null thickness). The grid is represented directly by a N-dimensional state array where each element represent a cell which is electrically coupled with its direct neighbors (2, 4 or 6 depending on the dimensions).

For 2D and 2.5D tissue model, in order to simulate a simple model of cylindrical geometry, the border in one of the dimensions of the model may be considered as the neighbor of the opposite border.

The dimension of a cell, the membrane resistance, and different physiologic features can be modified for each cell individually along each spatial dimension. This way makes possible to add anisotropy to the model and also to take into account local variations of the parameters. It is also possible to represent non-muscular cells (ie connective tissue). These cells would be affected by the spatial diffusion as the other cells but the model defined in Eq. 1 would not be applied to generate their own response.

To simulate the pacemaker cells, an external current is applied to the cells that will act as pacemaker. This external current is modeled by a half sine wave to avoid discontinuities. It can be applied to two different areas of the model during simulation.

Finally, we added a two-layer padding on the borders to avoid side effects of the spatial filtering. These 'ghost cells' have a coefficient of diffusion 10^4 times lower than the other cells to efficiently attenuate the signals at the borders.

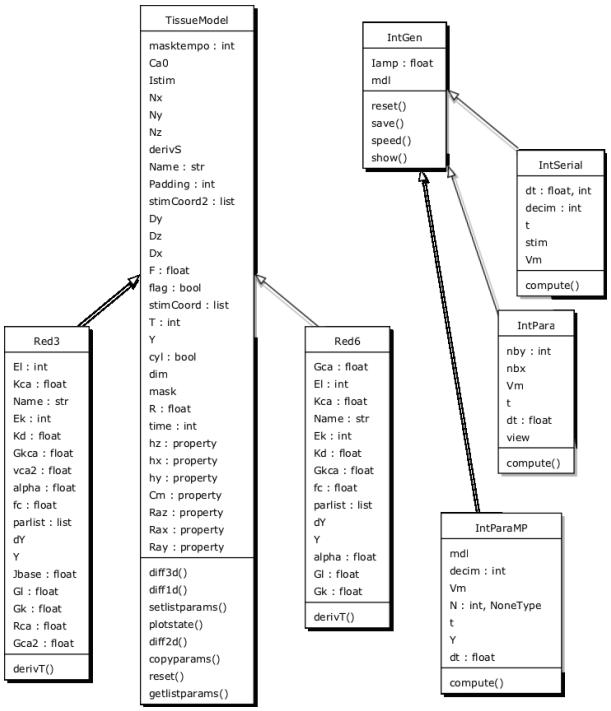

Figure 1: Main classes of uEMG.

3.2. Integration

We implemented two methods to compute the models. The first method is the serial integration, which computes the model by using only one process. This is the usual way to integrate such models, but it is ineffective on modern multi-core CPUs as only one of the cores is used. For a huge number of cells or for a long simulation duration, the computation time will rise to an unacceptable level. To improve the performance of the model (in term of computational time), parallel computing can be used. The idea is to divide the size of the tissue model into several processes which are running in parallel so that the global execution decreases.

A different process computes each part of the tissue model and for each time step, the processes share their results. For this study, we developed an integration model based on shared memory.

1) SERIAL: The serial integration computes the model using the Euler finite difference method. For each time sample, a discrete version of the cell model described in Eq. 1 is applied for every muscle cell of the tissue model.

After the muscle model, for each time step as well, the spatial diffusion model is applied to the cells to represent the electrical communication through gap-junctions. On a Cartesian grid the computation of this diffusion is a spatial Laplacian filtering which global weights are the diffusion coefficients. Those coefficients are dependent on the geometry and the membrane conductance of the cells.

We implemented an adaptive temporal step size to improve the efficiency of this simple scheme in terms of computation time. After each time step, the step size is refined according to the previously computed derivative value. The new temporal step □t at the nth iteration is computed as follows:

$$\Delta t_n = \Delta t_{min} \frac{\Delta V_{max}}{max\left(\Delta\left(V_m\right)_{n-1}\right) - Ft}$$

where ≥tmin (0.05ms) is the minimum step size accepted, ≥Vmax (1V) is the estimate of the maximum of change in Vm and F t is an adjustment constant. The minimum step has been determined from previous study, to assure the model stability.

2) PARALLEL: We used the multiprocessing module of Python standard library for the parallel implementation. It allows several processes to use a shared memory.

To deal with Numpy arrays (more convenient than the standard type of array) using shared memory, we used the module shmarray, written by David Baddeley (under BSD license).

At the beginning of the stimulation, a master process does the initializations and divides the grid of muscle cells and then generates as many processes as required (the default value being the total number of cores available) and then waits until all the processes are done.

The spatial domain is divided according to the number of working processes so that all the processes receive a sub-domain of roughly the same size (Minkoff 2002). Empirically, we found that the 1D-division presented better results than 2D- or 3D-parcellization. For that reason, only the rows of the state array are fairly divided for all the processes used since this method gives better results than dividing both the rows and the columns of the grid. Each process is dependent of its neighbors because it needs border information for the spatial diffusion process.

Each process has two rows of the cell grid in common with its neighbors so that the spatial diffusion can be computed accurately along the whole grid. In other words, a process can only modify the part of the grid that has been affected to him and can only "see" this domain as well as the two rows above and bellow. All the models of each sub-domain are computed asynchronously (both the cell and the spatial diffusion models). However all processes need to stay synchronized to process the same time step. This synchronization has been implemented as a barrier for all the processes, each of them storing its own results in the shared memory. A barrier basically blocks all participating threads until the slowest participating thread reaches the barrier call.

3.3. Tools

We also implemented several tools which could be useful for further studies as methods of the generic integration class.

1) Show(): A first need to arise was the ability to display the results of the simulation. To do so the generic integration class provides the show() method. Concerning 0D and 1D models, we used the matplotlib library (http://matplotlib.sourceforge.net/) to plot matlab-like graphics. Figure 2 shows the example of a 1D simulation, each column expresses the state of the whole model at a given time (100 cells computed here) and respectively each lines is the time course of one cell.

For higher dimensions model, we use the 3D-visualization library mayavi (http://code.enthought.com/projects/mayavi/). It allows us to show 2D and 3D animated simulations. Figures 3 and 4 show examples of simulation results displayed mayavi.

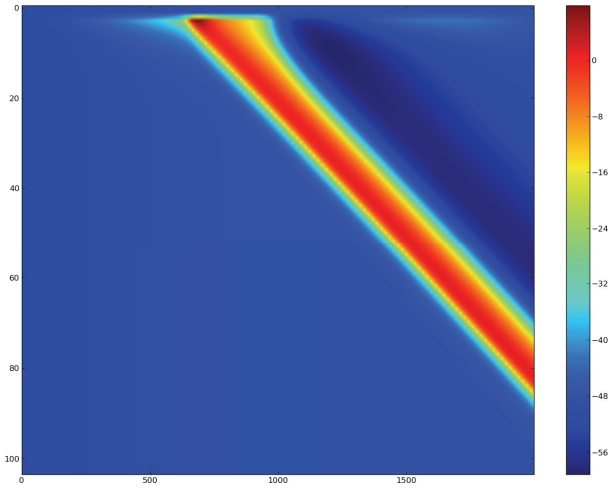

Figure 2: Representation of a 1D simulation using matplotib library. The x-axis represent the time, and the y-axis represents the cells. In this simulation, the cell number 3 the pacemaker.

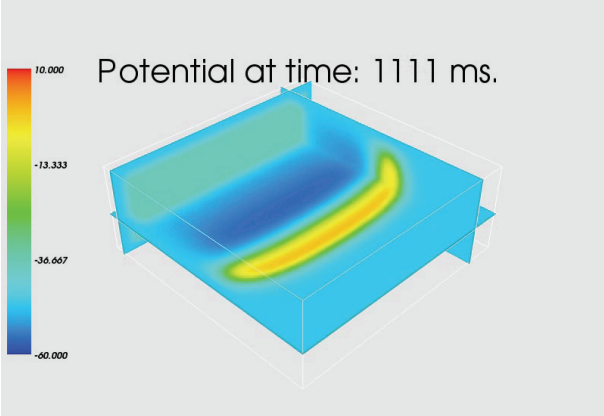

Figure 3: Representation of a 2.5D simulation using mayavi library. Electrical potential amplitude is color-coded.

2) Save(): To store the simulation results, we implemented the method save() using the file management abilities of the Numpy library. It saves a time array describing the sampling and the N-dimensional array describing the electrical potential for each saved time step. It is possible to part the results into several lighter files if the weight of the data to store is too important. A text file containing a description of the model used is saved as well.

3) Speed(): Finally, we implemented an easy way to measure the conduction velocity between two cells on the tissue model. To do so, we simply compute the delay between the time course of the two chosen cells by using the maximum of the cross-correlation. We then calculate the Euclidean distance to get the value of the conduction velocity. The existence of an actual propagation is assessed by a threshold applied on the value of the maximum of the cross-correlation.

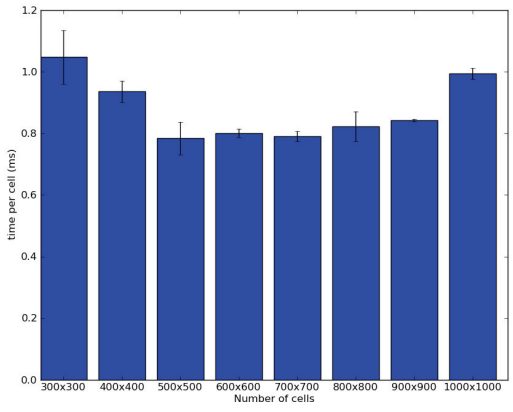

Figure 4: Time per cell (execution time divided by the number of cells) for different number of cells and 12 cores.

4. RESULTS

To test the efficiency of the parallelization of the model, the model was first run on a 8 core MacBookPro (Intel Core i7 2.2Ghz, 8GB RAM, OSX 10.6 64bits). Once our code became reliable enough for deployment, the simulations were run on a 24-core calculator (Intel Xeon X7542 2.67GHz, 256GB RAM, RedHat Enterprise 64bits). We first present the computation time per cell for simulations of a 2D models (going from 300 × 300 to 1000 × 1000 cells) with a given number of cores, to test the grid size effect. Then, we present the effect of parallel computing on a 500 × 500-cell model (250000 cells, referred later as 2D model) and on a 150 × 150 × 20-cell model (450000 cells, referred later as 2.5D model) by using 1 to 24 cores.

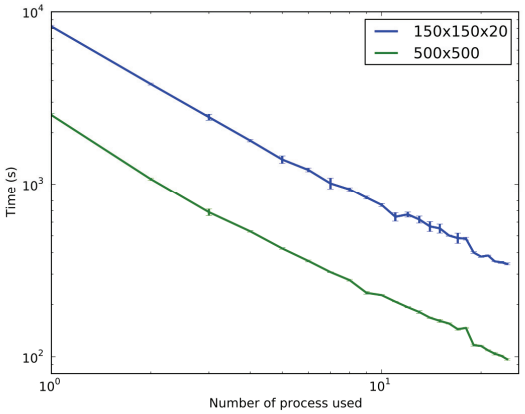

Figure 5: Execution time for 500x500 and 150x150x20 models. Curves are in Log-Log scales to simplify their reading.

Figure 5 shows the time per cell for a 1-second simulation for different grid dimensions and by using 12 processors. To assess repeatability, each simulation was run 5 times. For small dimensions, the parallelization is less efficient. Symmetrically, when the dimension of the model increases, the time of execution per cell increases as well. For both cases, it can be explained by the fact that the cost of the access to the shared memory stops being negligible compared to the gain of time due to the parallelization. For higher dimensions, one can also think that the memory hierarchy effects are less visible for these cases. The figure shows that the dimension of the grid does affect the performance of the parallel computing. Indeed, for too small or too big sizes of the grid, the benefit of the parallel computation is limited by issues of accessing the data.

Figures 6 and 7 show the mean of computation times over 5 simulations respectively for the 2D and 2.5D models, by using 1 to 24 cores. As expected, the 2.5D model takes more time than the 2D one. The time ratio between the times per cell of the two models is 1.88 (standard deviation: 0.07) which means that even with the same number of cells, a 2.5D model would be more time-consuming than a 2D model. The main reason of the time gap is attributed to the spatial diffusion model, where spatial diffusion is applied in 3 directions for the 2.5D model instead of 2 for the 2D model.

In both cases we notice a dramatic reduction of the computation time, from several hours to a few minutes.

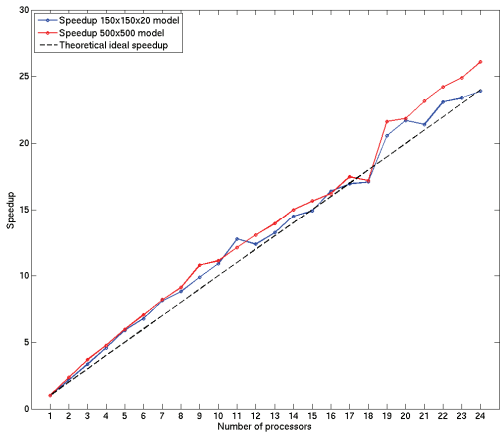

Figure 6: Speedup values for both configurations, for 1 to 24 cores.

To analyze more precisely the gain in the computation time given by the parallel computing, we calculated the speedup, which is defined as:

$$speedup(n) = \frac{T(1)}{T(n)} \quad (4)$$

where T (n) is the execution time on the parallel system with n processors, and T(1) is the execution time with only one process. In other words, speedup is defined by how much faster a simulation will run on n processors compared to a serial execution. In theory, the ideal speedup is linear, so for n processors it would be n. However in practical cases, supra-linear speedup has been noticed. Indeed, in some cases the computation time is reduced by a factor higher than the number of processors used for the parallelization (Camargos 2009). the memory hierarchy effects can explained this result, and mostly the use of memory cache that reduces the access time to the data. Figure 7 presents the results of the two models. This effect can be noticed in our data, since speedup values for both models follow a slightly supra-linear curve (results over the ideal curve).

5. CONCLUSIONS

Modeling the electric activity of the uterine muscle by realistic simulation is a challenging issue that raises the problem of computational time. We developed a complete tool to be able to parallelize time-consuming simulation. Our model is able to simulate uterine tissue model from 0D to 2.5D.

By using shared-memory parallel computing, we were able to reduce the execution time up to 25 times. In other terms, we could turn a simulation that lasted more than 2 hours into a 5-minute run. Indeed, our speedup curves follow a slightly supra-linear progression. However, for large dimensions model, the efficiency of the parallelization decreases. This will permit us to increase the dimension of the model, thus inducing an increase in the simulation duration, but keeping this increase reasonable enough. It will allow realistic simulations that could be validated by physiological experiments.

Our model still presents some limits. For instance, it is only able to simulate tissue areas of simple shapes (rect- angular, cylindrical). We will need better meshing abilities to be able to represent the complex uterine tissue structure. Another limit will be memory usage as the size of the simulated model increases. A work-around will be to support distributed memory calculators. We are developing a integrator class relying on message based parallelization (using MPI library) in this aim. Once deployed on a similar scale calculator we'll be able to confront the two approaches.

Finally, a user-friendly graphical interface will be added to the software in order to make its exploitation possible by non-specialists.

ACKNOWLEDGMENTS

This work was funded by ANR, partner of the ERASys-Bio+ initiative supported under the EU ERA-NET Plus scheme in FP7.

REFERENCES

Camargos, A., Batalha Martins Silva Soares, R. C. E. G., 2009, Super-linear speedup in a 3-d parallel conjugate gradient solver, *IEEE Transactions on Magnetics*, 45:1602 –1605

Devedeux, D., Marque Mansour Germain Duchene, C. S. G. J., 1993, Uterine electromyography: a critical review, *American Journal of Obstetrics and Gynecology*, 169:1636–1653

Laforet, J., Rabotti Terrien Mischi Marque, C. J. M. C., 2011, Toward a multiscale model of the uterine electrical activity, *IEEE Transactions on Bio-Medical Engineering*, 58:3487–3490

Minkoff, S.E. 2002, Spatial parallelism of a 3D finite difference Velocity-Stress elastic wave propagation code, *SIAM Journal on Scientific Computing*, 24:1 [5]

Rabotti C., Mischi Beulen Oei Bergmans, M. L. S. J., 2010, Modeling and identification of the electrohysterographic volume conductor by high-density electrodes, *IEEE Transactions on Biomedical Engineering*, 57:519– 527

Rihana, S., Terrien Germain Marque, J. G. C., 2009, Mathematical modeling of electrical activity of uterine muscle cells, *Medical & Biological Engineering & Computing*, 47:665–675

AUTHORS BIOGRAPHY

Tanguy Hedrich was born in Poissy (France) in 1989. In 2011 he received both the Engineering degree in computer engineering and the Master degree in biomedical engineering from Compiègne University. He is now a Master student in biomedical engineering at McGill university, Canada. His research interests include statistical signal processing and localization of epileptic spikes using distributed sources modeling.

Jeremy Laforet was born in France on Feburary 19, 1983. He recieved both the Master and Ph.D. Degrees from Montpellier 2 University, Montpellier, France, in 2006 and 2009. He's now post-doctoral researcher at Compiegne University in the biological engineering departement. His interests include neuro-musclar models and their indentification and validation.

Catherine Marque was born in France on January 15, 1958. She received the degree of Engineer from the Ecole Nationale Supérieure des Arts et Métiers, Paris, France, in 1980 and the Ph.D. degree from Compiègne University, Compiègne, France, in 1987.She is presently Professor at Compiègne University in the biological engineering department where she manages the biomedical engineering formation. Her interests include signal processing and instrumentation applied to the biomedical field.