Accepted Manuscript

Title: A fully parallel in time and space algorithm for simulating the electrical activity of a neural tissue

Author: Mathieu Bedez Zakaria Belhachmi Olivier Haeberlé
Renaud Greget Saliha Moussaoui Jean-Marie Bouteiller Serge
Bischoff

PII: S0165-0270(15)00350-7
DOI: <http://dx.doi.org/doi:10.1016/j.jneumeth.2015.09.017>
Reference: NSM 7346

To appear in: *Journal of Neuroscience Methods*

Received date: 29-6-2015
Revised date: 10-9-2015
Accepted date: 16-9-2015

Please cite this article as: Bedez M, Belhachmi Z, Haeberlé O, Greget R, Moussaoui S, Bouteiller J-M, Bischoff S, A fully parallel in time and space algorithm for simulating the electrical activity of a neural tissue, *Journal of Neuroscience Methods* (2015), <http://dx.doi.org/10.1016/j.jneumeth.2015.09.017>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

A fully parallel in time and space algorithm for simulating the electrical activity of a neural tissue

Mathieu Bedez^{a,b,c}, Zakaria Belhachmi^b, Olivier Haeberlé^c, Renaud Greget^a, Saliha Moussaoui^d, Jean-Marie Bouteiller^{a,e}, Serge Bischoff^a

^a Rhenovia Pharma SA, Mulhouse, France

^b Laboratoire LMIA EA3993, Université de Haute Alsace, Mulhouse, France

^c Laboratoire MIPS EA2332, Université de Haute Alsace, Mulhouse, France

^d Firalis SAS, Huningue, France

^e Department of Biomedical Engineering, University of Southern California, Los Angeles, CA, USA

Highlights

- Description of a method coupling parallelization in time and space on the GPU.
- Resolution of a model describing the propagation of the electrical signal in a neural tissue using the parallel algorithm on GPUs.
- Significant reduction in the calculation time to obtain an accurate solution.

Abstract

Background: The resolution of a model describing the electrical activity of neural tissue and its propagation within this tissue is highly consuming in term of computing time and requires strong computing power to achieve good results.

New method: In this study, we present a method to solve a model describing the electrical propagation in neuronal tissue, using parareal algorithm, coupling with parallelization space using CUDA in graphical processing unit (GPU).

Results: We applied the method of resolution to different dimensions of the geometry of our model (1-D, 2-D and 3-D). The GPU results are compared with simulations from a multi-core processor cluster, using message-passing interface (MPI), where the spatial scale was parallelized in order to reach a comparable calculation time than that of the presented method using GPU. A gain of a factor 100 in term of computational time between sequential results and those obtained using the GPU has been obtained, in the case of 3-D geometry. Given the structure of the GPU, this factor increases according to the fineness of the geometry used in the computation.

Comparison with existing method(s): To the best of our knowledge, it is the first time such a method is used, even in the case of neuroscience.

Conclusion: Parallelization time coupled with GPU parallelization space allows for drastically reducing computational time with a fine resolution of the model describing the propagation of the electrical signal in a neuronal tissue.

Keywords: GPU, neural activity, parareal algorithm, MPI, Partial differential equations

1 - Introduction

Brain activity results from electro-chemical reactions leading to the creation of an electric field propagating in all areas of the brain, as well as in the cranium. This electrical field, called electroencephalogram (EEG), can be measured by placing electrodes at specific locations of the skull. The EEG is used as potential biomarker in the diagnosis of some central nervous system diseases (CNS), such as epilepsy.

The main purpose of simulating EEG is to provide a better understanding of the connection between this global electrophysiological signal and the electrophysiological activity of subgroups of brain cells under physiological and/or pathological conditions.

This electrical activity can be modeled using the bidomain model, now widely used in the simulation of the electrical signal during the heart activity, which constitutes the electrocardiogram. This model was introduced for the first time in the 1970s (Schmitt 1969; Tung 1978).

Determining an analytical solution of this model, described via partial differential equations (PDEs), is not possible because of the highly complex geometry. The digital resolution requests some simplifications in order to obtain a usable result within a reasonable computation time. These sacrifices are made, depending on the choice of the model, the precision of the mesh in time and space, or by fixing the problem within a portion of the definition of space. To avoid making concessions on the results' accuracy, specific tools, which enable massively parallel computing, may be useful, distributing equitably the involved work, and thus reducing the computation time, while at the same time increasing the quality of results.

In this study, we present a fully parallelization technique in time and space, and apply it to the monodomain model case (Keener and Sneyd 2009), which constitutes a simplification of the bidomain model. This method will help us to determine the results quickly, and with an acceptable resolution in time and space. With this method, it is possible to run longer simulations within reasonable computation time, and also to perform simulations using very fine discretization grids, both in space and time. All this is not applicable in sequential methods, because calculation time would become excessively large. We used GPU computation, an approach already used in neuroscience: in 2013, Hoang has developed a CPU/GPU simulation environment for large-scale biological networks: NeoCortical Simulator. The possibility of simulating spiking neural networks on one or several GPUs was investigated by developers (Igarashi, Shouno O Fau - Fukai et al. 2011; Wang, Yan et al. 2011; Brette and Goodman 2012). The use of GPUs yielded significant improvements compared to CPUs (Baladron 2012). Such implementations perform a parallelization at the neuron model level, whereas our study aims at performing a parallelization of the solving methods, at the global problem level. We performed several simulations to validate the different algorithms and the computational setups.

2 Material and methods

2.1 - Bidomain model

As for the cardiac tissue, the neuronal tissue, defined by the domain B (see Fig. 1), can be modeled by decomposition into three distinct domains: the cells, called intracellular domain, the extracellular domain representing the outside of the cells, and the cellular membrane separating them. Each zone has an intracellular, an extracellular and a membrane potential (ϕ_i , ϕ_e and V_m respectively) and the voltage across the membrane is defined by their difference:

$$V_m = \phi_i - \phi_e \quad (1)$$

The intracellular, extracellular and membrane potentials are functions defined in B by:

$$\phi_i, \phi_e, V_m : B \rightarrow \mathbb{R}$$

In the absence of any stimulus, all cells remain in resting state, ranging between -60 to -70 mV. When a cell is excited, a depolarization phase is initiated reaching a threshold value. At a given potential, sodium channels close spontaneously, terminating the depolarization phase. At the same time, potassium channels open, causing repolarization of the plasma membrane. Then the potassium channel closure period causes a transient hyperpolarization (or relative refractory period) of the membrane potential and reach its resting potential.

Cell behavior has been described using models formulated from ordinary differential equations (see Section 2.3). Fig. 2 shows the variations of ϕ_e and V_m to a position over time.

The bidomain model might be used to describe the propagation of electrical signals along an axon, but also in the cells surrounding space. This model consists in two different types of equations: an evolution equation and an elliptic equation that describes the evolution of three potentials over time. Using a neuron model, and an appropriate conductivity tensor, these formulas can also be applied to neuronal tissue.

Throughout this research, the following assumption was made: in the intracellular and extracellular regions, current flow is resistive, resulting in a proportional relationship between the current density $\mathbf{J}$ and the potential gradient ϕ (Fick's law), where the matrix $\mathbf{M}$ is the conductivity tensor:

$$\mathbf{J} = \mathbf{M} \nabla \phi. \quad (2)$$

According to the current conservation law, the outflow from a domain must be equal to the inflow into the surrounding areas, which is described by the following equation:

$$\nabla \cdot \mathbf{J}_i = -\nabla \cdot \mathbf{J}_e. \quad (3)$$

The membrane current $\mathbf{J}_m$ is defined with respect to the flow of the potential, by combining equations (2) and (3):

$$\mathbf{J}_m = -\mathbf{M}_i \nabla \phi_i \cdot \mathbf{n}_i = \mathbf{M}_e \nabla \phi_e \cdot \mathbf{n}_i. \quad (4)$$

where $\mathbf{M}_i$ and $\mathbf{M}_e$ are conductivity tensors representing the intracellular and extracellular area, respectively, and $\mathbf{n}_i$ is the unit outward normal vector of the intracellular domain.

The behavior of the membrane is both resistive and capacitive. The perceived capacitive behavior of the membrane is due to an insulating bilipidic layer that separates the extra-cellular environment from the intra-cellular medium. The resistive part results from the transport of ions between the two environments by specific membrane proteins like voltage dependent channel. We consequently model this membrane with a resistor and a capacitor, connected in parallel, as schematically represented in Fig. 3. Associated currents are denoted $\mathbf{J}_c$ and $\mathbf{J}_{ion}$, respectively, and expressed in unit area per volume, while the membrane current is expressed in units of volume (Hodgkin and Huxley 1952). To maintain homogeneous units, one must multiply the ionic current and the membrane potential by area to volume ratio, which is denoted A_m .

The macroscopic bidomain model is obtained from a homogenization process (Neu J.C., Krassowska W. 1993), which yields a coupled parabolic-elliptic system of equations. The first equation is:

$$-\nabla \cdot (\mathbf{M}_i \nabla V_m) + A_m (C_m \partial_t V_m + \mathbf{J}_{ion}) = \nabla \cdot (\mathbf{M}_e \nabla \phi_e) \quad \text{in } B. \quad (5)$$

The second equation being:

$$\nabla \cdot ((\mathbf{M}_i + \mathbf{M}_e) \nabla \phi_e) = -\nabla \cdot (\mathbf{M}_i \nabla V_m) \quad \text{in } B. \quad (6)$$

The ionic current is determined by the resolution of one or more ordinary differential equations (ODEs), e.g. as Hodgkin-Huxley (Hodgkin and Huxley 1952; Hodgkin and Huxley 1990) or Fitzhugh-Nagumo (FitzHugh 1955; Nagumo, Arimoto et al. 1962) models, described in Section 2.3. We complete the model with Neumann boundary conditions for the extracellular potential ϕ_e and membrane potential V_m allowing for the field to be perfectly isolated:

$$\mathbf{M}_i \nabla V_m \cdot \mathbf{n}_{\partial B} = 0 \quad \text{in } \partial B \quad (7)$$

$$\mathbf{M}_e \nabla \phi_e \cdot \mathbf{n}_{\partial B} = 0 \quad \text{in } \partial B, \quad (8)$$

with $\mathbf{n}_{\partial B}$ representing the outward unit normal vector of domain B. The different variables are shown in Fig. 1, and the initial condition is given by:

$$V_m(0, x) = V_0. \quad (9)$$

2.2 - Monodomain model

The greatest difficulty in solving the bidomain problem is that the differential operator:

$$\nabla \cdot (M_i \nabla \phi_s)$$

is not explicitly defined from the expression of the membrane potential V_m (equation 5), but only through the elliptic problem (equation 6). We therefore have two problems to solve:

- (i) the dynamics problem according to V_m (equation 5),
- (ii) the elliptic problem to update the extracellular potential ϕ_s (equation 6).

The monodomain approach was introduced to simplify this two-fold problem. It is based on equal anisotropy ratios between the intra- and extra-cellular environments, i.e. the existence of a constant $\lambda > 0$ such that $M_i = \lambda * M_e$.

The elliptic problem (equation 6) then becomes:

$$(\lambda + 1) \nabla \cdot (M_e \nabla \phi_s) = -\nabla \cdot (M_i \nabla V_m) \text{ in } B, \quad (10)$$

which allows for writing:

$$A_m \left(C_m \frac{\partial V_m}{\partial t} + I_{ion} \right) = \frac{1}{1+\lambda} \nabla \cdot (M_i \nabla V_m) \text{ in } B, \quad (11)$$

with the Neumann boundary conditions in membrane potential V_m :

$$M_i \nabla V_m \cdot \mathbf{n}_{\partial B} = 0. \quad (12)$$

2.3 - Ionic current

To complete this model, we need to characterize the I_{ion} variable, which represents the ionic current. To this end, we propose to use well-established models that describe neuronal activity, and consequently define the dynamics of ionic currents. The selected model and its associated parameters are selected depending on the location within the brain. For our study, we use the Fitzhugh-Nagumo model, being a reduction of the Hodgkin-Huxley model (FitzHugh 1955; Nagumo, Arimoto et al. 1962).

The total ionic current described by the Hodgkin-Huxley model (Hodgkin and Huxley 1952; Hodgkin and Huxley 1990) is given by the following expressions:

$$I_{ion} = \sum_k I_{m,k} = I_{Na} + I_K + I_L = \bar{G}_{Na} m^3 h (V - E_{Na}) + \bar{G}_K n^4 (V - E_K) + \bar{G}_L (V - E_L), \quad (13)$$

$$\frac{\partial x}{\partial t} = \frac{x_\infty(V) - x}{\tau_x(V)}, \quad (14)$$

with:

$$x_\infty(V) = \frac{\alpha(V)}{\alpha(V) + \beta(V)}, \quad \tau_x = \frac{1}{\alpha(V) + \beta(V)}, \quad (15)$$

with I_{Na} being the sodium current, I_K being the potassium current, and I_L representing the leakage current. Variables $x = m, n$ and h (equations 11 and 12) are bounded between 0 and 1. Variables m and h describe the activation and deactivation of the sodium current, while n is the potassium

current activation. The α and β functions correspond to the transitions between open and close states of the different gates involved in each ionic channel. Although this model precisely describes neuronal activity, it is complex and numerically costly to incorporate, as being composed of three ordinary differential equations (ODEs), as a whole to resolve.

Introducing the Fitzhugh-Nagumo model, variables h and m vary over much slower time scales with respect to n and the membrane potential. Thus, one can combine the two variables in a single one, w , characterizing the degree of "refractoriness" of the system. The previous four-dimensional system (Hodgkin and Huxley 1952) is thus transformed into a two-dimensional one, having always the same dynamics characteristics.

It is composed of two variables: V_m the membrane potential and a recovery variable w :

$$A_m C_m \frac{\partial V_m}{\partial t} - \frac{1}{1+\lambda} \nabla \cdot (M_i \nabla V_m) = V_m - V_m^3 - w - I_{app} \quad (16)$$

$$\tau \frac{\partial w(t)}{\partial t} = V_m(t) - a - bw(t), \quad (17)$$

with the initial and boundary conditions (in the case of two spatial variables)

$$\begin{aligned} V_m(x, y, 0) &= w(x, y, 0) = 0 & \forall (x, y) \in B \\ \frac{\partial V_m}{\partial n}(x, y, 0) &= 0 & \forall (x, y) \in \partial B, \end{aligned}$$

where a , b and τ are the model parameters, and I is the external current being applied. Contrary to the Hodgkin-Huxley model, ionic currents do not explicitly appear, but the knowledge of the total ionic current is sufficient for the monodomain model.

Note that the values of the variables in the Fitzhugh-Nagumo model are bounded between -2 and 2, so it is necessary to rescale the obtained results in order to obtain physiological values.

Thanks to the low computational complexity of this approach, it may be used in the modeling of very large neural networks.

2.4 - Parameters values

The parameter values, for the Fitzhugh-Nagumo model used here, are (Fitzhugh 1961) :

$$\begin{aligned} a &= 0.7 \\ b &= 0.8 \\ \tau &= 13 \end{aligned}$$

We also define the parameter values for the monodomain model, assuming here that the conductivities $m_{i,x}$, $m_{\sigma,x}$ are the same in all directions (isotropic medium hypothesis):

$$\begin{aligned} m_{i,x} &= m_{\sigma,x} = 1 \text{ mS.cm}^{-1} \\ A_m &= 2000 \text{ cm}^{-1} \\ C_m &= 1 \mu\text{F.cm}^{-2} \end{aligned}$$

2.5 - Spatial discretization

The spatial discretization of the PDE characterizing the monodomain is achieved using the finite difference method. We define the problem within $[0,1]^D$ where D is the dimension of space (1 or 2 in this study), and discretize space by introducing N_x , the number of nodes along dimension x_i and of step size $\Delta x_j = 1/N_{x_j}$. The differential operator $\nabla \cdot (M \nabla V_m)$ then becomes:

$$\sum_{j=1}^D m_{i,j} \frac{V_m(x_j + \Delta x_j) - 2V_m(x_j) + V_m(x_j - \Delta x_j)}{\Delta x_j^2} \approx \sum_{j=1}^D m_{i,j} \frac{V_{m,j+1} - 2V_{m,j} + V_{m,j-1}}{\Delta x_j^2}, \quad (18)$$

where $V_{m,j}$ is the approximation of $V_m(x_j)$ and $m_{i,j}$ are the constant coefficients of the conductivity matrix M . Other discretization formulas exist, but that one has the advantage of convenience.

2.6 - Time discretization

To solve the ordinary derivative equations, we introduce a time step $\Delta t = T/N$, where N is the number of steps and T is the duration of the simulation. Time is approximated by discretized time $t_n = n * \Delta t$; $n = 0, \dots, N$ is also associated with the discrete variables representing the unknowns of the equation system: V_m^n, w^n (membrane potential and recovery variable, respectively) for the Fitzhugh-Nagumo model.

Two schemes may be used to integrate these equations: Euler or 4th order Runge-Kutta. The time step to be chosen depends on the model, and also the model parameters values. For the complete resolution of monodomain model, one must sequentially solve each part, i.e:

1- Calculate the V_m solution of the equation with the ionic current at time t_n calculated above, using Euler and Runge-Kutta of order 4 integration:

$$A_m \left(C_m \frac{\partial V_m}{\partial t} + I_{ion} \right) = \frac{1}{1+\lambda} \nabla \cdot (M_i \nabla V_m) \text{ in } B. \quad (19)$$

2 - Determine the recovery variables to update the ionic current I_{ion} :

$$\frac{\partial w}{\partial t} = f(V_m, w) \text{ in } B. \quad (20)$$

Each ionic current needs to be calculated at each node of the discretized space, resulting in a workload that varies depending on the location of the node, if one uses variable time step size. In order to keep a high level of accuracy for the solution, while still maintaining acceptable computation duration, computations may be parallelized.

2.7 Parallelization in time

To parallelize an ordinary differential equation or a partial differential equation, one can partition the time dimension. This technique appears at first look the least natural, since the resolution over time has the characteristics of being sequential. However, this technique has proven its interest in the case of very long temporal simulations. Lions et al. (Lions, Maday et al. 2001) first proposed an algorithm to solve an evolution partial differential equation problem using evolutionary time multi-grid. The convergence of this method has been proved in (Bal and Maday 2002; Gander and Vandewalle 2007).

Consider the PDE problem:

$$\frac{\partial u}{\partial t} + Au = f \quad ; \quad t \in [0, T] \quad (21)$$

$$u(t=0) = y_0,$$

where A is the discretization of an operator. The time intervals are decomposed into N equal sub-intervals $[t_i, t_{i+1}]$ where $t_i = t_0 + i * \Delta t$; $i = 1, 2, \dots, N$. We introduce the operator $G(t_{n+1}, t_n, u_n)$ that determines a coarse approximation $u(t_{n+1})$ of the solution u of the problem (equation 16) with the initial condition $u(t_n) = u_n$. This propagator G uses a step size Δt . Let $F(t_{n+1}, t_n, u_n)$ be an operator that determines a more accurate approximation of solution $u(t_{n+1})$, using a time step $\delta t \ll \Delta t$

The coarse propagator is used sequentially, then we solve in parallel, with the fine operator, the next equation over each subinterval:

$$\frac{\partial u_n}{\partial t} + A u_n = f_n ; f_n = f|_{[t_n, t_{n+1}]} , \quad (22)$$

with the initial condition:

$$u_n(t = t_n) = \lambda_n ,$$

where the functions λ_n (the coarse approximations) are defined for $n = 0, \dots, N-1$; $\lambda_0 = u_0$ and the same initial boundary conditions u .

The standard algorithm begins with an initial coarse approximation U_n^0 with $n = 0, \dots, N$ at time $t_0, t_1, \dots, t_N$ with the sequential calculation of $U_{n+1}^0 = G(t_{n+1}, t_n, U_n^0)$; $U_0^0 = u_0$, and then performs for each iteration $k = 0, 1, 2, \dots$ the corrector step :

$$U_{n+1}^{k+1} = G(t_{n+1}, t_n, U_n^{k+1}) + F(t_{n+1}, t_n, U_n^k) - G(t_{n+1}, t_n, U_n^k). \quad (23)$$

This parareal method stops when the next stop condition is satisfied:

$$\|U_{n+1}^{k+1} - U_{n+1}^k\|_{\infty} < \varepsilon , \quad (24)$$

where ε is the tolerance.

Several implementations of the parareal algorithm were compared in (Aubanel 2011), including the distributed algorithm, presented as the most effective. Unlike the standard algorithm, the coarse propagation is shared across processes. At first iteration, process 0 performs the coarse resolution $u_1 = G(u_0)$, and sends the result to the next process, which in turn continues the coarse resolution. When the process has completed his M coarse resolution, it performs fine resolution using smaller time step than for the coarse resolution, followed by the corrector step from (equation 23).

The result is the sent to process $M+1$. If $M+1 < N$, then a convergence test is performed at each iteration. If process M converges, its work is finished. Otherwise, it starts a new iteration, taking into account the correction performed by the $M-1$ process. An example with 4 processes and 4 iterations processes is given in Fig. 4.

The development of this algorithm is achieved through OpenMPI 1.4 (Message Passing Interface) that allows for exploiting the multi-processors characteristic of current CPUs. Another implementation of this algorithm was developed in CUDA, to handle the massively parallel architecture of GPUs. Unlike the MPI processes, which treat each a completely different piece of code, and therefore have to communicate their respective results, which can result in a blockage, GPU threads run the same piece of code, called the kernel, with different initial data.

2.8 - Spatial parallelization

One can also take benefit of the multi-threading capabilities of GPUs to parallelize the evolution equation V_m on the monodomain model. The temporal resolution is carried out sequentially, whereas the spatial resolution is performed in parallel. To do so, we use the discretization of the gradient V_m through the finite difference method.

In its sequential resolution, we use the matrix formulation described in section 2.5, but with GPUs, we use the standard equation using the Taylor expansion to order 2:

$$\sum_{i=1}^D \frac{u(x_i + \Delta x_i) - 2u(x_i) + u(x_i - \Delta x_i)}{\Delta x_i^2} \quad (25)$$

with $i = 1, \dots, N_x - 1$. This formula is nothing else than equation 18 applied to monodomain model, and with the conductivity parameters given in Section 2.4.

We define a CUDA grid having the same size as the mesh problem space, so that each CUDA thread can handle a single node. For example, in 1-D case, we restrict ourselves to a one-dimensional grid containing 1-D blocks thread. The dimensions of grids and blocks are defined according to the size of the problem.

2.9 - Dynamic grids

Version 5.0 of CUDA (Nvidia Corp. 2010), coupled with a GPU with a 3.5 or above micro-architecture version, has introduced dynamic parallelism. It is possible to launch kernels from threads running in the device i.e. threads can launch more threads. The interest of this technique is to launch an application in a coarse kernel, which in turn launches a finer kernel (see Fig. 5). Using this technique, the parallelism is coupled in time and space, thus enabling the use of a multi-threaded version of the monodomain problem.

The coarse kernel allows for parallel simulation time. Each thread of this coarse kernel is engaged to perform simulation in a time subinterval, and then all threads start the discrete calculation of the gradient (equation 20), and update the ionic currents (equation 13-14), on a fine kernel to parallelize computation space. By default, the grids launched within a thread block are executed sequentially: the next grid is launched only after the previous one has finished its computation. To implement a fully parallel algorithm, we must launch grid children competition. This is performed using CUDA streams.

Kernels launched in different streams are executed concurrently. With the Kepler architecture, we can run up to 32 streams in competition (against 8 with the older Fermi architecture). In other words, the time interval can be split into 32 subintervals with the current GPU architecture.

3 - Results

The simulations are performed using the Python language and mpi4py, a full-featured Python bindings package for OpenMPI 1.4 that we deployed on the Rhenovia Pharma cluster composed of DL380 HP Servers, equipped with 256 processing cores clocked at 2.5 GHz and with 128 Go of RAM.

The GPU code was developed with the parallel computing platform and programming model provided by NVIDIA: CUDA 6.0. The compilation of GPU code was performed on a NVIDIA Kepler Tesla K20c board, with compute capability 3.5, running at a frequency of 706 MHz and equipped with 5120 MB of GDDR5 memory.

We define the acceleration of the algorithm as the ratio between the sequential resolution time and the parallel resolution one. This parameter is a simple measure of the gain between two methods of resolution. In this study, we compare the time resolution obtained with parallel policies from the sequential method. If the speed-up parameter is greater than 1, then the parallel method will be faster than the sequential method.

3.1 - Neurons Models

We initially validated the parareal algorithm on the system of ODEs modeling the activity of a neuron, following the Fitzhugh and Nagumo approach.

We used the 4th order Runge-Kutta scheme with a time step $\Delta t = 0.003125 \text{ ms}$ for the coarse solver, in order to ensure stability. For the fine solver, we chose to use the explicit Euler scheme with a time step $\delta t = 1.25 \cdot 10^{-4} \text{ ms}$.

We determined the quadratic error between parareal algorithm results and those obtained with the sequential solution determined with explicit Euler scheme and a time step $\delta t = 1.25 \cdot 10^{-4} \text{ ms}$.

The tests are based on the activity of a neuron for a period of 200 ms, to which is applied an external current over a period of 100 ms. The simulation time needed by the CPU to sequentially solve the Fitzhugh-Nagumo model is 92 seconds on one node of the Rhenovia Pharma cluster. Then, we performed tests on 4, 8, 16, 20, and 32 different MPI processes.

Fig. 6(A-C) present the mean squared errors and corresponding time speed-up obtained by comparing the resolution time with sequential and parareal approaches, with MPI.

We note that for the Fitzhugh-Nagumo model, convergence takes place from the second or third iteration, according to the desired accuracy, i.e. when the mean square error is of the order of 10^{-7} . In addition, a decrease in CPU time is observed when one increases the number of processors, without having to increase the number of iterations. Note that the error increases slightly when the number of processors increases, which can lead, in the case of a large number of processors, to a greater number of iterations needed to maintain convergence. These errors increase because there is a propagation effect. When the number of processors increases, the correction computed by one processor takes longer to spread towards the other processors.

We then tested the GPU version of this algorithm, applied to the same problem, and using the same simulation parameters. Fig. 6(B) presents the results obtained using the parareal algorithm on GPU. A grid is defined in one dimension containing a single block of threads.

Using the same block, we can use the shared memory between threads and the synchronization between the threads within same block using the `__syncthread()` command. Shared and local memories are preferred, because the communication speed is significantly reduced when using global memory (up to 177 GB/s for global memory, while 1TB/s for shared memory is achievable). It also makes some blocking operations, so that the memory space dedicated to a specific thread is read too early, resulting in the recovery of a value, which has not yet been updated.

Our results indicate that the error between the reference (sequential resolution) solution and the GPU solution becomes negligible after no more than 3 iterations. The obtained error is notably larger than for the MPI parallelization (CPU) case with a small number of processes, but anyway remains below the desired accuracy. One important difference lies in the simulation time needed for additional iterations: with a GPU implementation, this time is very small, making computationally affordable the additional iterations needed to decrease the quadratic error. So, the GPU version of the parareal algorithm can drastically reduce the resolution time, while preserving a very high accuracy of the solution. Although some accuracy is to be sacrificed in order to quickly solve a problem, our results suggest that increasing the number of iterations reduces error, while still significantly decreasing the global simulation time.

Fig. 6(C) shows the average speedup between the resolution time with MPI or CUDA and the resolution time in sequential. These speedups are determined by computing the ratio between the sequential resolution time and the parallel resolution time. We observe a significant time saving with the parallel resolution, compared to the sequential resolution. But the parareal CUDA implementation drastically decreases the computation time, by a factor 10 as compared to the MPI implementation.

3.2 - Monodomain 1-D model

In this section, we consider the parallelization of the monodomain model, applied on the interval $[0,1]$ representing the spatial domain, and for a simulated period of 100 ms.

The simulation consists in applying an external current during 90 ms, using a resolution $\Delta x = 0,001$, a coarse time step size $\Delta t = 0,003125 \text{ ms}$ and a fine time step size $\delta t = 1,25 \cdot 10^{-4} \text{ ms}$. The reference solution is calculated using the explicit Euler scheme with a time step size $\delta t = 1,25 \cdot 10^{-4} \text{ ms}$. This solution is obtained in 820 seconds using one node of the Rhenovia Pharma cluster.

The results are presented in Fig. 7(A) for different numbers of MPI processes and increasing iterations.

We see that for an error in the order of 10^{-7} , we need about 4 iterations to reach convergence. The mean square error appears to increase slightly when the number of processors increases, suggesting that a higher number of iterations may be needed for a large number of processes, depending on the accuracy and degree of smoothness the user wants.

The CUDA implementation of the monodomain resolution uses the dynamic grid (see section 3.5) to parallelize also the Laplacian calculation. To get the most from competition, we use the CUDA streams. Each stream will perform problem solving on a sub-interval, launching the Laplacian calculation on a children grid. The resolution of the problem is therefore completely parallelized.

Fig. 7(B) presents the mean square errors for different sizes of sub-intervals (depending on the number of streams and hence the size of the parent grid). In all cases, the error is of the order of 10^{-6} after the 3rd iteration. Compared to the MPI resolution, we observe that the error after the first iteration is lower, being of the order of 10^{-4} , while it is around 10^{-1} for MPI. A more rapid convergence is observed for the CUDA implementation.

The speedup that depends on the number of MPI processes or CUDA streams, is shown in Fig. 7(C). The parareal simulation was completed when the error between two iterations respects a user-defined tolerance. Again, we see that there is indeed a reduction of computing time by a factor comprised between 1.6 and 9.5 when using the MPI implementation. When the CUDA implementation is used, we observe an even more dramatic reduction, by a factor of 80 to 138. The time parallelization combined with space parallelization therefore helps for solving this model almost immediately. As CUDA grids childrens depend on the size of the problem, the computation time of the monodomain model with a finer spatial resolution will not take much longer than this resolution.

3.3 - Monodomain 2-D model

We then apply the methods discussed earlier to parallelize calculations of 2-D simulation of the mono domain model. The spatial domain is defined by the square $[0,1]^2$ with a spatial resolution $\Delta x = \Delta y = 0,2$, the coarse and fine time step size being $\Delta t = 0,03125 \text{ ms}$ and $\delta t = 1,25 \cdot 10^{-3} \text{ ms}$, respectively.

The simulation represents a period of 100 ms, with application of an external current during 90 ms. The obtained results are summarized in Fig. 8, for varying numbers of MPI processes and CUDA streams.

Results from the MPI implementation are shown in Fig. 8(A). It is observed that 6 iterations are needed to obtain an error of the order of 10^{-6} between sequential and MPI parareal solution. Note that in this case, the speedup generally decreases by a factor of 2 compared to the 1-D case. It is therefore necessary to launch this integration on a large number of nodes in order to reduce the time of resolution, i.e. the time needed to reach a solution close to the reference solution.

Contrariwise, the computational time required to solve the monodomain model using dynamic grids is almost identical that the time required in 1-D, because the children grids has adapted to the size of problem. Thus, there is the same number of procedures to be performed by CUDA thread relative to the 1-D problem. Sequential solving time being logically longer, the speed-up is more important as shown Fig. 8(C). We observe that convergence seems faster than with MPI (see Fig. 8(B)), because the mean square error for the first iteration is small, while the MPI error is so significant that it has not been shown in Fig. 8(A). Indeed, it takes an average of four iterations to get to an error in the order of 10^{-7} .

3.4 - Monodomain 3-D model

Finally, we tested our method by applying the monodomain model to a 3-D geometry. The space sector is defined by the cube $[0,1]^3$ with a spatial resolution $\Delta x = \Delta y = \Delta z = 0.5$, the steps of coarse and fine time are those defined in Section 3.3, i.e. $\Delta t = 0.03125 \text{ ms}$ and $\delta t = 1.25 \cdot 10^{-3} \text{ ms}$.

The simulation is performed over a period of 100 ms, by applying to a portion of the tissue an external current during 90 ms. Sparse matrices are used to reduce the memory space allocated in the sequential and MPI resolutions. The results obtained using the different methods are summarized in Fig. 9.

Fig. 9(A) shows the results obtained using the parareal method developed using the MPI technology. Again, there is a decrease of the error during the various iterations. Note also that the error is homogeneous with respect to the choice of the number of processors. Furthermore, it becomes acceptable between the third and fourth iteration, that is to say around 10^{-6} . In the case of MPI, Fig. 9(C), we see that the speedup is of the same order of magnitude than those observed for 2-D geometry. The gain becomes non-negligible when the number of processors increases, which tends to confirm, once again, the usefulness of the parareal approach to solve the problem.

Fig. 9(B) shows the results obtained when applying the method developed on GPU. The errors seem to again be homogeneous according to the number of streams used in the parallelization of the time scale. Also, one observes a continuous decrease of the error during the various iterations, even if they are obviously somewhat larger compared to the MPI application. An error of the order of 10^{-6} is obtained after three or four iterations, which confirms the results obtained in the 2-D case. Fig. 9(C) shows the speed-up obtained using the fully parallel method. It is noted that the resolution time decreases greatly compared to sequential time and MPI, even if the gain is not as important as in 2-D. This is because the sequential resolution and MPI are performed using sparse matrices, thereby significantly reducing the calculation times, while the resolution in space GPU does not use the matrix formulation of the problem.

4 - Discussion

The size of the problem that can be set is limited by technical characteristics of the different technologies. In MPI, there is a limit to the size of the data, that commands Send and Broadcast can transfer. This limit depends on hardware characteristics of the machine (memory, processor), but also on the MPI version, which is used. For this reason, and especially in 3-D, the use of sparse matrices is required, thereby greatly reducing the memory to be used. Some data communications are however not compressible, limiting the maximum size of the problem.

The same difficulty holds when using GPU architecture. The limits are obviously different depending on the graphics card and its architecture. The sizes of the different types of memory (local, shared, global) differ from an architecture to another. As we have seen in section 3.5, the number of streams that can be launched in competition is limited to 32 on the current Kepler architecture. This is reflected in our method by dividing by 32 the maximum time scale. The next architectures will push these limits further back, as was the case during the passage of the Fermi architecture to the Kepler one (with 8 versus 32 streams, respectively). Regarding the parallelization of space, the limit, if one attaches one thread per mesh node, depends on the maximum size of the GPU grid and the maximum size of a thread block. The graphics card used during our test phase (Tesla K20) defines a grid of maximum dimension (2147483647, 65535, 65535) and a maximum size of thread block (1024, 1024, 64), already allowing a fine enough geometry for solving our present problem.

All these specifications are to be considered on a power development, which will aim to solve a PDE problem in future, especially in 3-D, as the resolution of bidomain problem.

5 - Conclusion

We have applied parallelization methods to the monodomain model, which is a simplification of the more complete bidomain model. This parallelization allows for significant reduction of the simulation time by using fine resolutions in both time and space, using GPU technology. This

method thus allows obtaining a better solution whether in space or time, but it can also perform much longer simulations on larger areas without catastrophic increase of the resolution time.

With this technology, and using a fraction only of the resources of the graphics card, it was shown that the resolution of an ordinary system of differential equations is performed in less than one second of time. Then by applying the same process to 1-D, 2-D and 3-D monodomain models, we have shown that the resolution time also decreased by increasing the size of the resource used either in time or in space. Using these methods, we can simulate this model over much longer periods, but also in larger spaces. Furthermore, they allow for keeping a high level of resolution limiting the calculation time.

GPU technology has also permitted to develop a resolution of the problem by parallelizing in space and also in time with dynamic grids. The ratio between the computation time in sequential and GPU implementation is very important compared to MPI (launched on multiple cluster nodes). This study showed that the use of a graphics card enables massive parallelization of a problem, thus requiring the full capabilities of the Tesla card. But in comparison, running MPI allows for a limited parallelization only.

These methods have proven to be helpful for simulation of a monodomain model, and the logical extension will be to apply this method to a bidomain model for a better description of the propagation of electrical activity, but requiring considerable resources and especially time calculation.

6 - Acknowledgements

Mathieu Bedez thanks the entire team of Rhenovia Pharma for the biological expertise, which enabled us to do this study, as well as for the access to the Cluster computing, and Bertrand Simon, from MIPS laboratory, for giving access to its Tesla GPU K20c, to develop and demonstrate the concept of our method.

7 - Conflict of interest

Jean-Marie C. Bouteiller has a conflict of interest. The University of Southern California holds an equity interest in Rhenovia Pharma, and has also received licensing income from Rhenovia Pharma.

Figures

Fig. 1 - Definition set of bidomain model

Fig. 2 - Membrane (V_m) and extracellular (ϕ_e) potential after bidomain simulation

Fig. 3 - The membrane is modeled with a capacitor and a resistor in parallel

Fig. 4 - The first 3 iterations in 4 processors of distributed parareal algorithm

Fig. 5 - Description of GPU dynamic grid

Fig. 6 – (A) Mean square error for the Fitzhugh-Nagumo neuron model following the stop iteration of parareal algorithm, and for different numbers of MPI process. (B) Mean square error of the simulation of Fitzhugh-Nagumo model with parareal algorithm developed in CUDA. The error is determined for different stop iterations and size of CUDA grids. (C) Average speed-up of the CUDA resolution, compared to the parareal MPI resolution.

Fig. 7 – (A) Mean square error for the 1-D monodomain model between the sequential and the MPI parareal solutions at each iteration, up to convergence. (B) Mean square error for the 1-D monodomain model between the sequential and GPU solutions, at each iteration, up to convergence. (C) Speed-up between the sequential time calculation and the MPI/CUDA time calculation.

Fig. 8 – (A) Mean square error for the 2-D monodomain model between the sequential and the MPI parareal solutions at each iteration up to convergence. (B) Mean square error for the 2-D monodomain model between the sequential and GPU solutions, at each iteration, up to convergence. (C) Speed-up between the sequential time calculation and the MPI/CUDA time calculation.

Fig. 9 – (A) Mean square error for the 3-D monodomain model between the sequential and MPI solutions at each iteration of the parareal method. (B) Mean square error for the 3-D monodomain model between the sequential and GPU solutions, at each iteration of the parareal algorithm. (C) Speed-up between the sequential time calculation and MPI/CUDA time calculation.

References

- Aubanel, E. (2011). "Scheduling of tasks in the parareal algorithm." Parallel Computing **37**: 172-182.
- Bal, G. and Y. Maday (2002). A "Parareal" Time Discretization for Non-Linear PDE's with Application to the Pricing of an American Put. Recent Developments in Domain Decomposition Methods. L. F. Pavarino and A. Toselli, Springer Berlin Heidelberg: 189-202.
- Baladron, J., Fasoli, D., and Faugeras, O. (2012), "Three Applications of GPU Computing in Neuroscience." Computing in Science & Engineering **3**: 40-47
- Brette, R. and D. F. Goodman (2012). "Simulating spiking neural networks on GPU." Network **23**(4): 167-182.
- FitzHugh, R. (1955). "Mathematical models of threshold phenomena in the nerve membrane." The bulletin of mathematical biophysics **17**: 257-278.
- Fitzhugh, R. (1961). "Impulses and Physiological States in Theoretical Models of Nerve Membrane." Biophys J **1**(6): 445-466.
- Gander, M. J. and S. Vandewalle (2007). On the Superlinear and Linear Convergence of the Parareal Algorithm. Domain Decomposition Methods in Science and Engineering XVI. O. B. Widlund and D. E. Keyes, Springer Berlin Heidelberg: 291-298.
- Hodgkin, A. L. and A. F. Huxley (1952). "Propagation of electrical signals along giant nerve fibers." Proceedings of the Royal Society of London. Series B, Biological Sciences **140**: 177-183.
- Hodgkin, A. L. and A. F. Huxley (1990). "A quantitative description of membrane current and its application to conduction and excitation in nerve. 1952." Bulletin of Mathematical Biology **52**: 25-71; discussion 25-23.
- Igarashi, J., T. Shouno O Fau - Fukai, et al. (2011). "Real-time simulation of a spiking neural network model of the basal ganglia circuitry using general purpose computing on graphics processing units." (1879-2782 (Electronic)).
- Keener, J. and J. Sneyd (2009). Mathematical Physiology. New York, NY, Springer New York.
- Lions, J. L., Y. Maday, et al. (2001). "R al. (2011). "Real-time simulation of a spiking neur" Comptes Rendus de l'Acad et al. (2001). "R al. (2011). "Real-time s **332**: 661-668.
- Nagumo, J., S. Arimoto, et al. (1962). "An Active Pulse Transmission Line Simulating Nerve Axon." Proceedings of the IRE **50**: 2061-2070.
- Neu, J.C., Krassowska W. (1993). "Homogenization of syncytial tissues." Critical Reviews in Biomedical Engineering **20**: 137-199
- Nvidia Corp. (2010). NVIDIA CUDA C Programming Guide.
- Schmitt, O. H. (1969). Biological Information Processing Using the Concept of Interpenetrating Domains. Information Processing in The Nervous System. K. N. Leibovic, Springer Berlin Heidelberg: 325-331.
- Tung, L. (1978). A bi-domain model for describing ischemic myocardial d-c potentials., Massachusetts Institute of Technology.
- Wang, M., B. Yan, et al. (2011). Simulation of large neuronal networks with biophysically accurate models on graphics processors. Neural Networks (IJCNN), The 2011 International Joint Conference on.