AN ANALYSIS OF THE PARALLEL COMPUTATION OF ARBITRARILY BRANCHED CABLE NEURON MODELS*

JOSEP-L. LARRIBA-PEY†, MICHAEL MASCAGNI‡,
ÀNGEL JORBA§, AND JUAN J. NAVARRO†
†DEPARTAMENT D'ARQUITECTURA DE COMPUTADORS,
UNIVERSITAT POLITÈCNICA DE CATALUNYA, BARCELONA, SPAIN
‡SUPERCOMPUTING RESEARCH CENTER I.D.A.
BOWIE, MARYLAND, USA
§DEPARTAMENT DE MATEMÀTICA APLICADA I
UNIVERSITAT POLITÈCNICA DE CATALUNYA, BARCELONA, SPAIN

SRC-TR-94-134

November 1, 1994

Abstract. We present and analyze a parallel method for the solution of partial differential equation models of the nervous system. These models mathematically are one-dimensional nonlinear parabolic equations defined on branching domains. Implicit methods for these equations leads to numerical solution of diagonally dominant almost tridiagonal linear systems at each time step. We first review some exact methods for the solution of these linear systems that includes an **E**xact **D**omain **D**ecomposition. This **EDD** leads to the solution of many tridiagonal linear systems one for each branch. The sizes of these systems is equal to the number of grid points on the branch. Since the branches of realistic neurons vary widely in size, the decomposition leads to a very poor *a priori* load balance. This problem may be solved with the **O**verlapped **P**artition **M**ethod, a method for decomposing large diagonally dominant tridiagonal systems. We describe and analyze an algorithm based on **EDD** and **OPM** that can be load balanced.

Key words. Nerve modeling, partial differential equations, numerical method, domain decomposition, parallel algorithm, vectorization

AMS(MOS) subject classifications. 65C06, 65M55, 65Y05, 92C20

* Work supported by the Ministry of Education and Science of Spain (CICYT TIC-880/92), the CEE (ESPRIT Project APPARC 6634) and by CEPBA.

1. Introduction. Considerable effort has gone into the numerical solution of realistic models of the nervous system [1, 2, 4, 8, 9, 13]. Some of the most efficient and accurate of these methods are based on implicit finite-difference solutions of one-dimensional nonlinear parabolic partial differential equations (PDEs) that capture phenomena including the action potential and repetitive firing. The spatial domains of these one-dimensional PDEs reflect the morphology of realistic neurons and so are extensively branched. This leads to almost tridiagonal systems to be solved at each time step for an explicit solution to the PDEs. The description and analysis of a parallel algorithm for these almost tridiagonal systems is the purpose of this paper.

The plan of the paper is as follows. In §2 we will describe the PDE models of nerve conduction that we wish to solve. In addition, we will briefly review their implicit finite-difference discretizations. We then describe the geometry of the computational domains of interest and will derive the almost tridiagonal linear systems that of interest. In §3 we reference a serial algorithm for the solution of this almost tridiagonal linear system based on a reordering of the unknowns. We also explain an **E**xact **D**omain **D**ecomposition algorithm for this system based on breaking the neuronal domain into problems to be solved on individual branches, [10]. The linear systems on the branches are simple diagonally dominant tridiagonal systems; however, since a neuron's branches vary considerably in length, the sizes of these systems also vary. This inhibits an *a priori* load balance of the parallel computation. In §4 we propose a solution to load balancing by using the **O**verlapped **P**artition **M**ethod, a method for decomposing large diagonally dominant tridiagonal systems, to produce a more uniformly sized set of tridiagonal systems. We describe how to achieve an *a priori* load balance with the addition of **OPM** and we analyze the overall algorithm's complexity. In §5 we use some results from the analysis of **OPM** to improve the other phases of **EDD**. In §6 we summarize our results and conclude.

2. The Mathematical Model of Neurons. It is believed that the computational prowess of the nervous system is due to complex activity in neurons. This complex activity includes, among other activities, ensemble activation, the propagation of action potentials, and the repetitive firing of neurons, [12]. Ordinary differential equations models of activity can often capture much of this behavior, but experience teaches that the most faithful models to date are PDE models, [8]. The basic form of these equations is:

$$C\,\frac{\partial V}{\partial t} = \frac{a}{2R}\frac{\partial^2 V}{\partial x^2} - \bar{g}V. \tag{2.1}$$

By scaling $x \to x\sqrt{\frac{2R}{a}}$ and $t \to t/C$ in the above equation one can obtain the cable equation, [11]. If $\bar{g}$ is a constant, then this linear cable equation is a valid PDE model of the dendrites and a further change of variables in V reduces (1) to the heat equation. For axons $\bar{g} = \bar{g}(V, x, t, \ldots)$ is a nonlinear conductance and no such scaling in V is possible and then (1) is nonlinear parabolic PDE.

Popular algorithms for the numerical solution of (1) are finite-difference methods, [8]. The need to resolve spatially the action potential places an upper bound on the scaled Δx. Thus an explicit solution of (1) must use a step-size that satisfies $\Delta t \leq \frac{(\min \Delta x)^2}{2}$. This is overly restrictive and begs the use of an unconditionally stable implicit method. Using implicit methods requires the solution of a linear or perhaps a nonlinear equation at each time step. Different forms of nonlinear conductance require different methods of solution, [9], thus let us assume that we can advance our implicit finite-difference solution one time step by solving a single linear system.

If we consider the simplest case: solving the linear cable equation on a single one-dimensional domain, then the matrix for the linear system obtained is diagonally dominant and tridiagonal. If we use a backward-Euler time discretization and ignoring boundary conditions, the matrix is Toeplitz with diagonal $1 + 2\lambda + \gamma$ and off diagonals $-\lambda$ where $\lambda = a\Delta t/2RC(\Delta x)^2$ and $\gamma = \bar{g}\Delta t/C$. The morphology of biological neurons requires that we consider heavily branched one-dimensional domains. This leads to linear systems that are almost Toeplitz with extra off diagonals that couple the different branches together at branch points. An example showing a branching geometry and the associated linear system can be found in [10]. The purpose of this paper is the description and analysis of a double domain decomposition algorithm for the parallel solution of these types of linear systems for the complicated geometries found in the nervous system.

3. The EDD Algorithm. Real neurons are extensively branched. In fact, the anatomy of real neurons includes an extensively branched dendritic region that converges onto a single cell body. The cell body then emanates a lightly branched axon that ends in other heavily branched presynaptic regions. The linear system corresponding to an explicit finite-difference solution of such an axon is still diagonally dominant, but is very complicated indeed.

An algorithm that reduces the linear system from any loop-free neuron geometry into a logically single diagonally dominant tridiagonal system was proposed and is based on a careful numbering and reordering of the unknowns in such a system, [3, 8]. This reordering algorithm will not be described here, but it has major two shortcomings: (a) the algorithm is equivalent to Gaussian elimination without pivoting and hence is hard to vectorize/parallelize, and (b) the algorithm precludes modeling neurons with electrical synapses since their geometry is not loop free. These defects motivated an exact domain decomposition algorithm, [10] based on decomposing the linear system on the entire neuron into subproblems solved only on the branches and then only at the branch points.

All branches are connected to at least one branch point, some to two. This observation motivates the main idea behind the algorithm, namely that the solution along a branch is a superposition of at most three solutions: (i) the branch equations with branch points assumed zero; (ii) the solution with the right-hand side zero and the "left" branch point assumed one; and (iii) the homogeneous "right" branch point solution. After obtaining the solution of these tridiagonal system one creates a linear system for the branch point values. This equation is neither positive definite nor symmetric and has the sparsity pattern of the adjacency matrix for the graph of the neuron geometry. The solution of the branch point equation then gives proportionality constants to form the linear combinations of the branch tridiagonal system solutions. For more details see [10].

To review, the almost tridiagonal system can be solved by a decomposition of the domain into K unbranched domains of size n_k (for $1 \leq k \leq K$). The solution of the linear system with the domain decomposition method is found in the following way:

A) Initialization: In this phase, K to $2K$ independent tridiagonal systems have to be solved. These systems can be solved in parallel with Gaussian elimination without pivoting. Each branch has one of these systems $E_k y_k = b_k$ if it is connected to one branch point and two systems $E_k y_k = b_k$ and $E_k z_k = v_k$ if it is connected to two branch points. These are the "right" and "left" homogeneous solutions.

B) Iteration: At each time step i do:

B.1) Solve K independent tridiagonal systems $E_k d_k^{(i)} = x_k^{(i-1)}$ where E_k are diagonally dominant in structure but different in size. These K systems can be solved independently in $O(n_k)$ operations via Gaussian elimination.

B.2) Using part of the solutions of step **B.1**, solve a sparse system of equations for the branch points $P x_b^{(i)} = x_b^{(i-1)}$ where $x_b^{(i)}$ is the solution vector of the branch points at step i. This system is approximately of order K, and has no special structure. The matrix P represents a condensation of the unknowns in the problem onto only the branch points.

B.3) Perform K general `axpy` operations $x_k^{(i)} = d_k^{(i)} + x_r^{(i)} y_k + x_\ell^{(i)} z_k$. Here $x_r^{(i)}$ and $x_\ell^{(i)}$ are the solution to branch points r and l adjacent to branch k. These K operations can also be performed in parallel giving $x_k^{(i)}$, the overall solution at time step i.

While we think of **EDD** as an algorithm for the solution of a particular type of linear system, this linear system arises from a time-dependent nonlinear PDE. Thus the coefficient matrix can change each time step and so the matrices E_k and P may also change. However, many of the K branches are dendritic and are therefore governed by linear PDEs. Here the E_k matrices do not change and this provides for potential savings. This implies that the vectors y_k and z_k will remain constant throughout the computation, and so can be computed once and stored. In Hines' algorithm, no such savings are realized. We should also note that if a uniform spatial discretization is made where the cable equation constants λ and γ are made identical along a dendritic branch then y_k and z_k will be reflections of one another on that branch. Moreover, if λ and γ are the same along all the dendritic branches, then if one computes and stores y_k for the longest dendritic branch, measured in grid points, all of the y_k's and z_k's on the shorter branches are simple restrictions to this reference solution. This provides additional initialization savings.

Some comments on the **EDD** algorithm are necessary. Most important is the observation that **EDD** allows only limited *a priori* load balancing. The most time consuming computations are **B.1** and **B.3**. These are performed in parallel on the individual branches and require $O(n_k)$ operations. Anatomy and subsequent discretization give us the n_k's, the branch lengths. Determining an *a priori* load balance on Π processors requires partitioning the n_k's into Π subsets with sums as close to $\Lambda = \lfloor \sum_k n_k / \Pi \rfloor$ as possible, in general an NP-hard problem. To make matters worse, axonal branches are usually much much longer than typical dendritic branches potentially making the best possible *a priori* load balance very poor. In the case that the longest axonal branch is smaller than Λ there is no hindrance to an *a priori* load balance. However, when the longest axonal branch is larger than Λ, a method that allows further subdivision of branches into smaller problems is required. This situation occurs especially on a massively parallel processors where Π is large.

4. A Further Decomposition via OPM. We wish further to subdivide the solution of a single tridiagonal system into smaller sized problems to assist in load balancing. This can be accomplished with the **O**verlapped **P**artition **M**ethod proposed for the vectorization of diagonally dominant tridiagonal systems, [6]. **OPM** has been shown to be faster than other methods, such as Divide and Conquer, on vector computers, [5, 7] and appears to be very promising on parallel computers as well, [5]. In **OPM**, one decomposes a long diagonally dominant tridiagonal system into several overlapping tridiagonal systems that can be solved independently. The overall solution to the long system is made up of the individual solutions. Usually, the overlapped unknowns are taken from the first system they appear in, but this choice is not crucial. Iteration of **OPM** is a convergent overlapping Schwartz domain decomposition for a one-dimensional parabolic PDE. However, the interesting fact is that by choosing the overlap correctly this decomposition is as accurate as one desires in a single iteration. This fact should not astonish as the linear systems corresponding to elliptic PDEs do not have this property while those from parabolic PDEs do.

Let us state the conditions by which we can determine the overlap as a function of the desired accuracy in the overall solution. First define a measure of the diagonal dominance of the tridiagonal system with typical equation $e_i x_{i-1} + d_i x_i + f_i x_{i+1} = b_i$ by $\delta = \min_i \frac{|d_i|}{|e_i|+|f_i|}$. In a uniform linear backward-Euler discretization we clearly have $\delta > 1$ as $\delta = (1 + 2\lambda + \gamma)/(2\lambda)$, and so $\delta = 1 + O(\lambda^{-1}) > 1$. Experience with the backward Euler discretization shows that one can use a value of λ as large as 5 or 10 and still obtain qualitatively good results. However, with the unconditionally stable Crank-Nicolson method, a value of $\lambda \approx 2$ is more appropriate as larger λ values lead to unphysiological oscillations in these equations. With Crank-Nicolson we also have $\delta = 1 + O(\lambda^{-1}) > 1$. It is easy to bound the solution to such a tridiagonal system by:

$$\mu = \|x\|_\infty \le \frac{\max_i |b_i/d_i|}{1 - \delta^{-1}}. \tag{4.1}$$

With μ we can prove that for an accuracy ε we require an overlap no smaller than m where m is defined by the following bound:

$$m \ge \left\lceil \frac{1}{\log \delta^{-1}} \log \left[\frac{\varepsilon(1 - \delta^{-2})}{\mu} \right] \right\rceil. \tag{4.2}$$

Notice that the value of m can be computed *a priori* and so can be hard coded into a computation.

The use of **OPM** is only required to decompose the extremely long branches to assist in load balance. It is obviously not desirable to use **OPM** to slavishly decompose every system into subproblems of exactly the same size. In fact, if branch k has $n_k < 2m$ than **OPM** cannot be applied. Also if we have a branch that is decomposed by **OPM** into j subproblems, then there are $(j-1)m$ extra overlapped variables. Since these overlapped variables appear in tridiagonal systems the extra work required to solve this branch equation is $O((j-1)m)$. However, this is only $O(m)$ extra parallel work.

5. More Parallelism from OPM. A consequence of the analysis in **OPM** is further reduction in the work required in algorithm's initialization as well as step **B.3**. In addition, this same analysis gives us criteria for when a branch point equation can be considered diagonal. Such a branch point is decoupled from the rest and can be computed independently.

If we consider the special solutions, y_k and z_k, on linear branches, these must be combined with other solutions at each time step. It has been proved for such systems, that only the first m elements of these solutions are larger than ε with m defined by, [6]:

$$m = \left\lceil \frac{\log \varepsilon(1-\delta^{-2})}{\log \delta^{-1}} + 1 \right\rceil \tag{5.1}$$

This information helps us during both initialization and **B.3** as follows. Suppose we have an *a priori* bound on the branch point voltages, the multipliers of y_k and z_k in step **B.3**. Call this bound $\mathcal{V}$. If we calculate m from equation (4) with ε replaced by $\mathcal{V} \times \varepsilon$, we will not need more than the first m unknowns in y_k and z_k. Thus we only need compute these many unknowns initially. In addition, since only the first m unknowns in y_k and z_k are numerical significant, we need only combine these many unknown in the `axpy` computations in step **B.3**. This saves work in steps **A** and **B.3**.

The matrix P from step **B.2** represents the coupling between branch points. In [10] it was shown that this coupling is the value $y_k(\ell)$, i.e. the minimum value of the special solution. Equation (4) tells us how big this special solution can be with m interpreted as the branch length in grid points given the bound $\mathcal{V}$. When equation (4) predicts a sufficiently small value for $y_k(\ell)$, that unknown becomes decoupled and may be solved in parallel to the rest of the unknowns.

Since equation (4)'s behavior is crucial to this extra parallelism, we provide a table that contains values m for a small number of values for ε and δ. The value of δ is computed from λ assuming a uniform backward Euler discretization with $\gamma = 0$.

m	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-16}$
$\lambda = 1 \rightarrow \delta = 1.5$	22	39	90
$\lambda = 5 \rightarrow \delta = 1.1$	107	180	397
$\lambda = 10 \rightarrow \delta = 1.05$	223	364	789

6. Conclusions and Open Problems. This paper has presented the details of a parallel algorithm for solving a special linear system. This system arises from the implicit finite-difference solution of one-dimensional parabolic PDEs on extensively branched domains. This sort of problem arises in the simulation of realistic neuron models. The algorithm takes advantage of two noniterative domain decompositions. The first, **EDD** breaks the problem variables into branches and branch points. The branches lead to diagonally dominant tridiagonal systems which determine the values in an equation to be solved for the branch points. The wide range of lengths in realistic branch points motivates the use of the **OPM** to further partition the longest branches. The overall algorithm requires $O(\sum_k n_k)$ arithmetic operations, equal to the operation count for the optimal serial algorithm and provides considerable opportunities for parallelism. In addition, the analysis of **OPM** produces several additional savings as well as more opportunities for parallelism in the second phase of **EDD**.

Currently, **EDD** is part of the `Genesis` simulation package available from the Division of Biology at Caltech, [13]. It is hoped that a fully parallel version of `Genesis` will appear that incorporates the entire algorithm described in this paper. Another application for just the **OPM** in this application domain is as a method of decomposing Hines' serial algorithm for parallelism. While the asymptotic cost of a parallelized Hines' algorithm would be the same as our algorithm, it may prove more useful on large-grained implementations.

REFERENCES

[1] J. W. Cooley and F. A. Dodge, *Digital computer solution for excitation and propagation of the nerve impulse*, Biophys. J., 6 (1966), pp. 219-229.

[2] F. A. Dodge and J. W. Cooley, *Action potential of the motorneuron*, IBM J. Res. Dev., 17 (1973), pp. 219-229

[3] M. Hines, *Efficient computation of branched nerve equations*, Int. J. Bio-Medical Computing, 15 (1984), pp. 69-76.

[4] A. L. Hodgkin and A. F. Huxley, *A quantitative description of membrane current and its application to conduction in the giant axon of Loligo* , J. Physiol., 117 (1952), pp. 500-544.

[5] J.-L. Larriba-Pey. *Tridiagonal Systems on Parallel Computers.* 3rd Mons Workshop on Parallel Computing, Mons (Belgium), September 23-34 1993.

[6] J.-L. Larriba-Pey, A. Jorba and J. J. Navarro. *A Parallel Tridiagonal Solver for Vector Uniprocessors*, SIAM Conf. on Par. Proc. for Sci. Comp., Norfolk, 1993, pp.590-597.

[7] J.-L. Larriba-Pey, J. J. Navarro, O. Roig and A. Jorba. *A generalized vision of some parallel bidiagonal systems solvers.* Proceedings of the Intl. Conf. on Supercomputing 1994, ACM Press, pp. 404-411.

[8] M. Mascagni, *Numerical methods for neuronal modeling*, in Methods in Neuronal Modeling, C. Koch and I. Segev (Eds.) MIT Press, Cambridge, MA, pp. 439-484.

[9] M. Mascagni, *The backward Euler method for numerical solution of the Hodgkin-Huxley equations of nerve conduction*, SIAM J. on Num. Anal., 27 (1990) pp. 941-962.

[10] M. Mascagni, *A parallelizing algorithm for computing solutions to arbitrarily branched cable neurons*, J. of Neuroscience Methods, Elsevier Sci. Publishers 36 (1991), pp. 105-114.

[11] W. Rall, *Core conductor theory and cable properties of neurons*, in Handbook of Physiology: The Nervous System, Vol.1, Kandel, E. R. Brookhardt, J. M. and V. B. Mountcastle (Eds.), Williams and Wilkins, Co., Baltimore, MD, pp. 39-98.

[12] G. M. Shepherd, *The Synaptic Organization of the Brain*, Third Edition, Oxford University Press, New York, Oxford (1990).

[13] M. A. Wilson and J. M. Bower, *The simulation of large-scale neural networks*, in Methods in Neuronal Modeling, C. Koch and I. Segev (Eds.) MIT Press, Cambridge, MA, pp. 291-333.