Identifying, tabulating, and analyzing contacts between branched neuron morphologies

J. Kozloski
K. Sfyrakis
S. Hill
F. Schürmann
C. Peck
H. Markram

Simulating neural tissue requires the construction of models of the anatomical structure and physiological function of neural microcircuitry. The Blue Brain Project is simulating the microcircuitry of a neocortical column with very high structural and physiological precision. This paper describes how we model anatomical structure by identifying, tabulating, and analyzing contacts between 10^4 neurons in a morphologically precise model of a column. A contact occurs when one element touches another, providing the opportunity for the subsequent creation of a simulated synapse. The architecture of our application divides the problem of detecting and analyzing contacts among thousands of processors on the IBM Blue Gene/L™ supercomputer. Data required for contact tabulation is encoded with geometrical data for contact detection and is exchanged among processors. Each processor selects a subset of neurons and then iteratively 1) divides the number of points that represents each neuron among column subvolumes, 2) detects contacts in a subvolume, 3) tabulates arbitrary categories of local contacts, 4) aggregates and analyzes global contacts, and 5) revises the contents of a column to achieve a statistical objective. Computing, analyzing, and optimizing local data in parallel across distributed global data objects involve problems common to other domains (such as three-dimensional image processing and registration). Thus, we discuss the generic nature of the application architecture.

Introduction

Building neural tissue: The neocortical column

Since the late nineteenth century, researchers have observed biological tissue at the microscopic scale. Their studies have revealed tremendous complexity, especially in the structure and composition of neural tissue [1]. At the scale revealed by light microscopy (1–100 μm at 5- to 100-fold magnification), neural tissue is composed of two primary cell types: glia and neurons. Both glia and neurons are branched structures whose morphologies are varied and categorized by many clear subtypes [2].

In the early 1950s, studies began to elucidate the electrical dynamics of neural tissue at this same microscopic scale [3]. Glial cells exhibit simple electrical dynamics and provide a supporting matrix into which neurons grow, develop, and live. Neurons, on the other hand, produce rich electrical dynamics, whose properties also provide a basis for categorization [4]. When neurons form neural microcircuits, these dynamics in turn yield the emergent electrical and computational properties of neural tissue. Hodgkin and Huxley's studies of single-neuron electrophysiology yielded the first formal, predictive model of the dynamics of a simple isopotential neuron [5]. Researchers elaborated on this model beginning in the 1970s in order to accommodate the branching, spatially extended structure present in most neurons [6, 7]. These methods include *compartmental modeling*, or the practice of modeling neurons as branched graphs of isopotential compartments [8] (**Figure 1**).

Neurons communicate via two types of connections known as synapses. *Electrical synapses* provide a resistive electrical coupling between the interiors of compartments

0018-8646/08/$5.00 © 2008 IBM

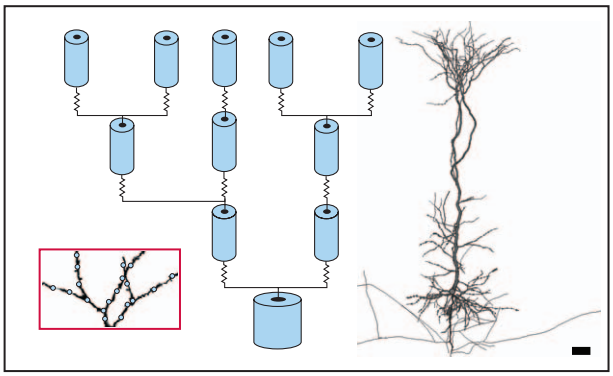

Figure 1

Neuron modeling. An anatomist captures points using online microscopy in order to describe segments (left) of a neuron (right; scale bar is 100 μm). The points are used to construct a model of a branching graph of equipotential compartments coupled by resistors (center). The term "segment" refers to two points and the line that joins them.

from different neurons. *Chemical synapses* provide a means by which electrical activity in a compartment that precedes a synapse (i.e., a presynaptic compartment) causes the release of neurotransmitter chemicals. Through diffusion and a complex series of molecular mechanisms, neurotransmitter release causes electrical activity in a compartment following the synapse (i.e., in a postsynaptic compartment). Hence, electrical synapses are bidirectional, while chemical synapses are always unidirectional.

Simulations of compartmental neuron models typically examine single neurons for which synapses may be modeled as a set of parameterized current sources [8]. In the 1980s, some researchers began composing simulated neural microcircuits from compartmental neuron models, thereby representing at least a portion of the synapses of a neuron as logical inputs from other neuron models [9]. Synapse locations specified in lists of connections between neuron compartments were used to create either arbitrary circuits or approximations of real circuits.

A great diversity of neurons exist in the neocortex and create what is known as the *neocortical microcircuit*. (The neocortex refers to the outermost layer of the cerebral hemispheres in mammals.) The Blue Brain Project, based on widely accepted neuron categories [10], classifies neocortical neurons into 21 morphological types and 20 electrophysiological types, with 32 combined electro-morphological types (**Figure 2**). The neocortical tissue is a six-layered sheet, from 1 to 4 mm in thickness and 1 to 10,000 cm^2 in lateral extent. This sheet is divided laterally into columns, each about 0.5 mm in diameter with a radial axis oriented normal to the plane of the sheet. We assemble each column using approximately 10,000 neurons that are sampled from all of the varying morphological and electrophysiological types according to neuron density functions that vary along the radial axis of the column. It is this variation that creates the layered appearance of the sheet. Packed densely into the volume of the column (which is comparable to the volume of a toothbrush bristle), these diverse branching structures are woven tightly into what Ramón y Cajal colorfully described as the "jungle . . . impenetrable and indefinable" [1].

Accurately specifying connections between all 10,000 neurons in a column is both an opportunity to map known circuit parameters into an accurate connection list and a challenging simulation task. While analytic approaches exist for estimating circuit statistics from neuron morphologies [11], precisely modeling the neocortical column allows us to derive specific circuits from real neuron morphologies. This approach is desirable both because neuron morphology data is easier to collect and, therefore, more complete than precise circuit connectivity data, and because the observed specificity in the circuit likely derives from known regularities in tissue composition. As with any endeavor in computational neuroscience, the goal involves a compromise between unknown and known parameters, as well as an attempt to derive rules and principles from biological observations that can be used to refine existing models.

BlueBuilder: The neocortical column structural modeler

To build the column, the Blue Brain Project has implemented the BlueBuilder Structural Modeler [12]. This tool provides interfaces between a database of single-neuron morphologies and a model of the neocortical tissue, as well as between the model tissue and a column specification file. A user builds the tissue by loading single-neuron morphologies from the database and distributing these neurons within a column visualization either manually or by application of various density functions. After the desired tissue composition has been crafted, the software creates a column specification file that includes the three-dimensional (3D) location of every point describing each neuron in the column, as well as information about the identity of each neuron and its parts (see Figure 2).

Data in the neuron database are derived from three sources, collected in the following order: 1) electrophysiological measurements of the neurons, 2) morphological reconstructions of the neurons, and 3) morphological measurements of the reconstructions. First, an experimenter accesses a neuron in the actual cortical tissue using a microelectrode. This electrode allows the experimenter to record and assign the electrical dynamics of the neuron to a category such as "classical

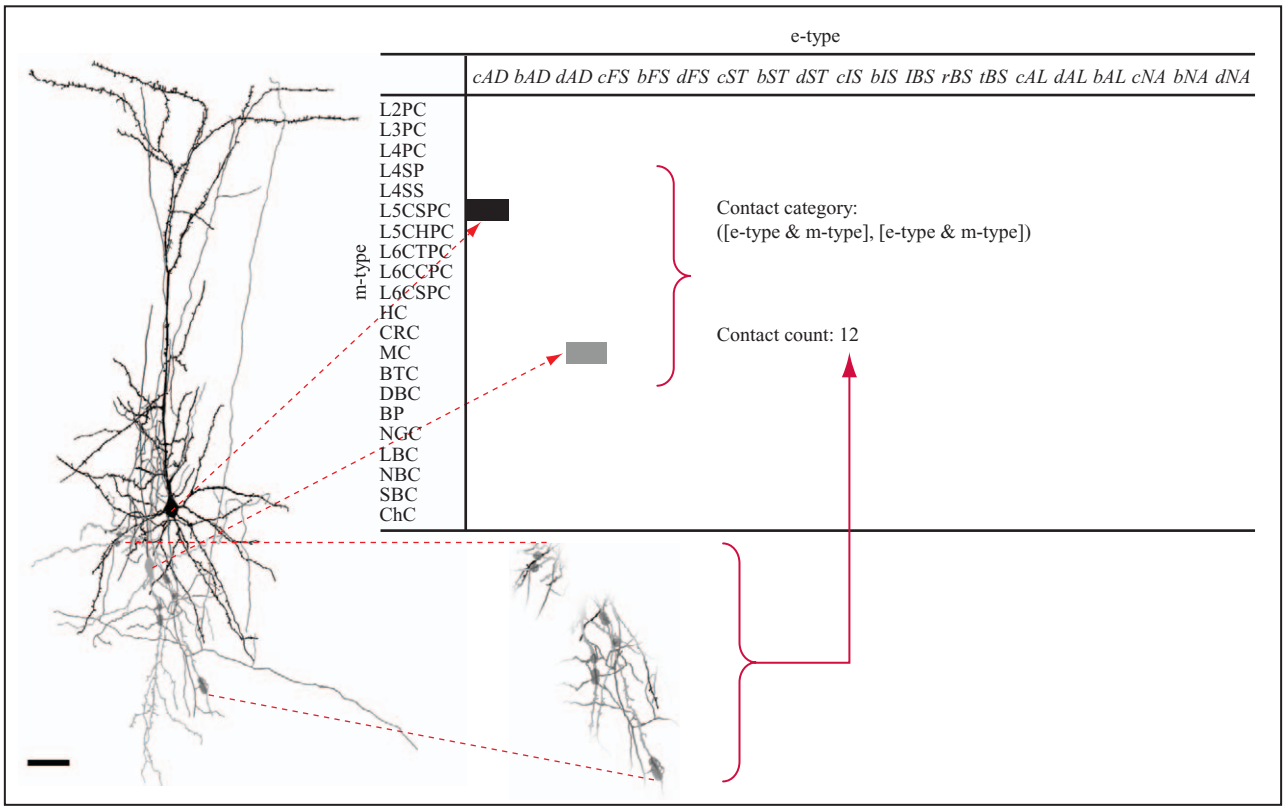

Figure 2

A pair of touching neurons are categorized by their e-type (electrophysiological category) and m-type (morphological category), resulting in a unique identifier in a "neuron-space" (scale bar is 100 μm). These identifiers are concatenated to give rise to a unique contact identity key for the tabulation of detected contacts. The expression "([e-type & m-type], [e-type & m-type])" refers to a category of touches formed by considering contacts between neurons that fall into the intersection of two neuron categories denoted "e-type & m-type."

accommodating" (cAD) or "classical fast spiking" (cFS). (The terms cAD and cFS describe the sequence of action potentials, or "spikes"—e.g., for cAD neurons, spike frequency decreases during the sequence, whereas for cFS neurons, spike frequency is high and constant.) This electrophysiological category (or "e-type") is associated with the neuron in the neuron database. The electrode contains a biological agent that darkly stains the neuron. The anatomist views the stained neuron through a microscope and uses an apparatus to digitize the morphology of the neuron by recording the 3D location of points representing branch points or transitions in the direction or diameter of a neural fiber (or neurite) [13]. The anatomist identifies each neurite as either an axon (i.e., a presynaptic output fiber) or a dendrite (i.e., a postsynaptic input fiber) and estimates each neurite diameter at each point collected. A "reconstruction" in the database comprises all of the approximately 4,000 points from a single neuron, and each reconstruction is analyzed and assigned to a morphological category, or "m-type," such as "layer 5 cortico-subcortical pyramidal cell" (L5CSPC) or "Martinotti cell" (MC), which is then associated with the neuron in the neuron database (see Figure 2).

Circuit building: Problem description and requirements

The column specification file contains approximately 40 million points, each comprising three coordinates, a diameter, and a pre-computed distance between each point and each subsequent point in a segment. The problem then is to calculate the location of all contacts between segments in this model of the tissue. Points comprising neurons form sets, to which each contact must ultimately be related. Thus, neuron identity (i.e., layer, e-type, and m-type), branch identity (axon or dendrite, branch order), and the identity of branch points are also present for each relevant set. Set data are derived from individual measurements of individual neurons in the database, usually taken from different animals and often on different days. The Blue Brain Project has limited its

studies to a particular area of the neocortex (the somatosensory area) of the albino Wistar rat. Circuit building is challenging because sets typically span the entire data space (i.e., the entire volume of the column), while the principal calculation of contacts between segments must occur over all points within a local space.

Circuit statistics are derived from many experimental sources. Anatomical studies at a variety of scales (e.g., light microscopic and electron microscopic scales) yield distributions of neurons and synapses in the column. Electrophysiological studies using simultaneous recordings from more than one neuron [14] yield probabilities of connectivity between neurons of particular categories (e.g., e-type to e-type, m-type to m-type). Optical methods are also used to probe connections in parallel [15]. Because circuit tracing and measurement techniques have undergone rapid development only recently, these data at present are heterogeneous, incomplete, and changing. Nevertheless, they represent the target for any validation of the structural simulation of the column, described above.

We attempted to create a system for rapidly identifying contacts between all 40,000,000 neurite segments (defined by two points) present in the column specification file. The geometrical problem of calculating the distance between cylindrical objects is well understood [16]. The scale of our calculation, however, motivated us to target our application for the massively parallel architecture of the IBM Blue Gene/L* (BG/L) supercomputer [17]. We require the maximum possible speed for calculating these contacts because the observed contact statistics in the column must fit experimental observations from any and all sources. We expected that the initial column specified in BlueBuilder would not fit these statistics. Thus, requirements emerged that 1) the column specification must be revisable within the current application such that the precise neuron position and rotation about the radial axis are modifiable, and 2) column revision must occur iteratively on the basis of some comparison between observed and simulated circuit statistics. Because we anticipated that many thousands of iterations would be necessary to fit the simulated tissue to observed circuit statistics, we recognized that contact detection and tabulation for the full column must be accomplished in approximately 1 minute by the full 8,192-processor BG/L supercomputer used for the Blue Brain Project.

Parallel algorithm overview

Iterative architecture

Design philosophy: Collective equality

The parallel software architecture that we pursued avoids master/slave relationships between the processors of the BG/L system. We use the phrase *master/slave* to refer to a communication protocol in which one process (the master) controls one or more other processes (the slaves), such that the direction of control is always from the master to the slaves. Our goal was to minimize I/O and communication and create a compute-bound application whose performance would, therefore, scale linearly with respect to the number of processors. We recognized that the column specification file could be read once into memory, given its size of 1.73 GB and given that the amount of memory available was 2.1 TB. Because the computational work involved with contact detection is inherently local (the relationship between two neurites is localized to a point in the 3D column), we also recognized that the work of each processor could be derived from a specific location in the column. However, assigning the work from these locations could not be performed by a single master processor without multiple accesses to a file, since each processor has access to only 256 MB of memory. Therefore, we recognized that the calculation needed for distributing work for contact detection itself had be distributed. For this reason, processes in the current architecture are collectively equal: Each performs precisely the same operations on data read from a single file. This property has many additional benefits that we address below, including 1) a logical data structure for neuron data read from a file, 2) exclusive use of the collective communication functions of the Message Passing Interface (MPI), the functions of which have been optimized on the BG/L system for rapid data exchange among all processors [18], and 3) a simple interface to design and extend the parallel algorithm.

Initialization

The work performed by each process in the MPI application we describe here differs only in its starting data and is, thus, a single-program, multiple-data (SPMD) application. Initialization allows processor rank to determine which data is read from a file and ensures that the amount of data read is nearly equal for each processor. In this way, distributing the work of contact detection is nearly load balanced with respect to computation and communication loads. The scheme for distributing the work of contact detection is also calculated during initialization and ensures adequate load balancing during this more computationally intensive second phase of the algorithm.

The role of the Director

We observed a need to incorporate arbitrary circuit analysis steps into the algorithm in order to allow for new refinements to the measurement and fitting of circuit statistics. We anticipated, for example, the need to correct for the effect of arbitrary spatial sampling resolution in

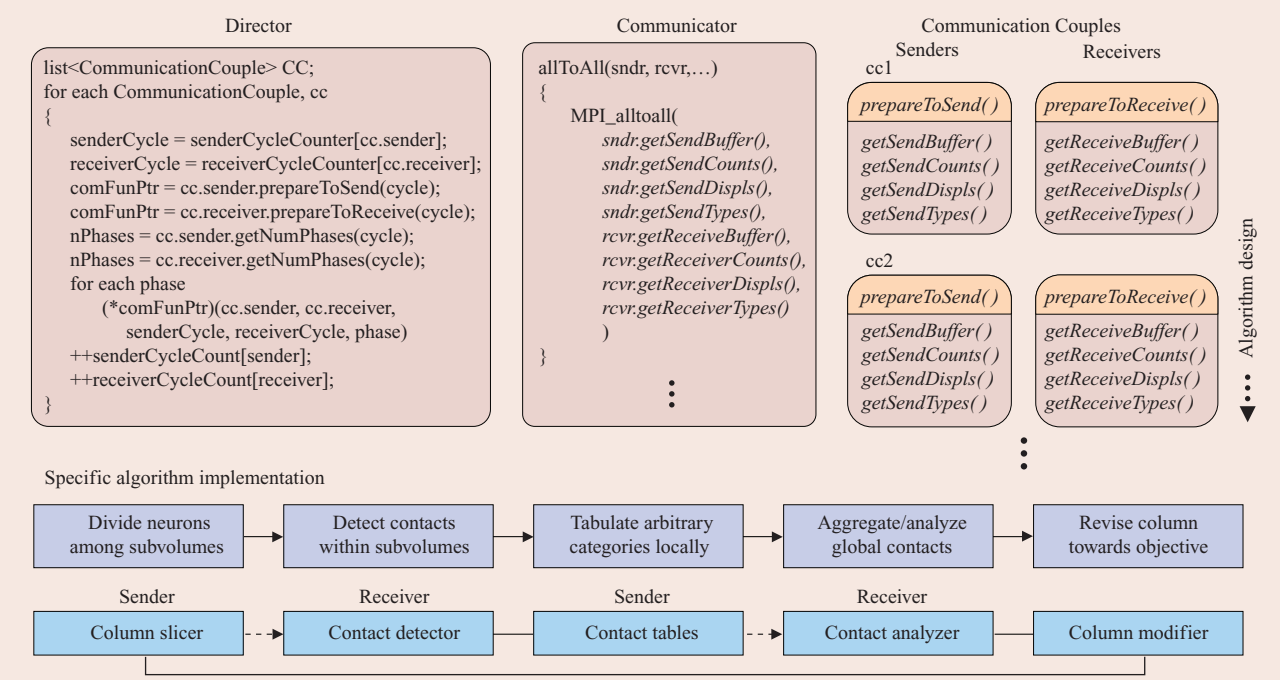

Figure 3

The role of the Director in the iterative algorithm. Using interfaces on algorithm objects that implement the abstract Sender and Receiver interfaces, an algorithm designer composes these objects into an algorithm by adding Sender–Receiver couples to the Director object. Algorithm execution then consists of the Director initiating computation (by executing *prepareToSend* and *prepareToReceive* methods) and communication (by executing function pointers from the Communicator object) for each couple. The specific algorithm described in this paper is depicted (bottom). (cc1: Communication Couple 1; dashed arrows: collective communication between Senders and Receivers.)

the neuron digitization method [13] on contact distributions, as well as the need, at times, to transform individual contacts between neurites into multiple synapses (e.g., in cases in which the neurites are parallel). While these issues have been addressed using the current iterative architecture, the specifics of these technical solutions are beyond the scope of this paper. Initially, we designed a simple two-phase algorithm that distributes the work of contact detection (by means of the process of slicing neurons) and performs contact detection within a column subvolume (i.e., between neuron fragments in the subvolume). These phases are separated by a single communication boundary using MPI collective communication. In addition, in order to realize an iterative algorithm, one more communication boundary is traversed to complete the process loop and aggregate circuit statistics. In addition, in order to implement these steps, the initial software architecture also permits an arbitrary number of additional steps with arbitrary collective communication on step boundaries, using generic interfaces to incorporate and specify each step. The only assumption in our design is that communication on the boundaries is performed using a communication function wrapped in a standard interface.

The central agent in this architecture is the *Director*. (Note that programming objects, such as Director, are capitalized by convention.) A Director generically implements a single iteration of the algorithm as a list of pairs of generic object interfaces known as *Senders* and *Receivers*. The list defines the algorithm and is constructed during the parameterization of the Director (**Figure 3**). Parameterization is performed during compile time and involves constructing the Sender and Receiver objects necessary to implement a desired algorithm and then adding Sender–Receiver pairs (or *Communication Couples*) to the Director. The Director also has a reference to a Communicator object that implements all of the communication functions necessary for communication between Senders and Receivers. A single iteration then occurs as the Director iterates over its list of Communication Couples.

Collective communication

Senders and Receivers are implemented to support a generic interface that specifies a set of communications.

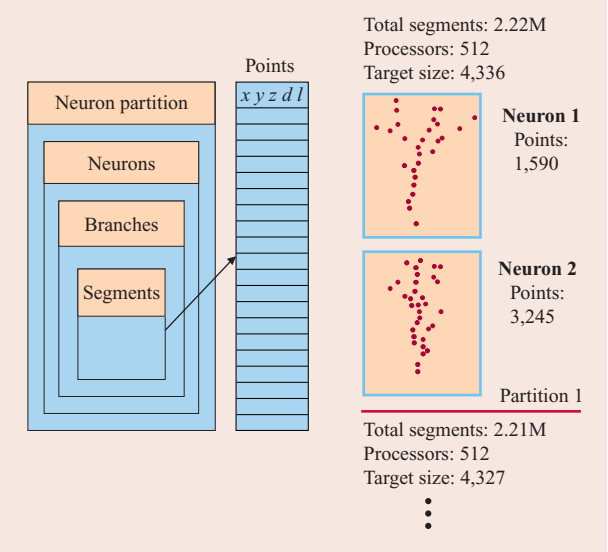

Figure 4

Neuron representations. The hierarchical Neuron object data structure (left). Assignment of neurons to the Neuron Partition object is performed by using an adaptive algorithm (right) to ensure adequate load balancing of the work of neuron slicing. (*Slicing* is discussed further in the section "Data decomposition: The 3D slicing metaphor." Variable *d* refers to the diameter of a segment. Variable *l* refers to the length of a segment.)

The specified communications are decomposed into cycles, and within a cycle, phases. Cycles allow a single Sender or Receiver object to specify different communication events and, thus, participate in an arbitrary number of independent Communication Couples. A user composes an algorithm by placing Communication Couples in an order (see Figure 3, top right). For any given Sender or Receiver that appears multiple times in the order, an internal cycle mapping will specify what communication is attempted and in what order for any single iteration. For this reason, Senders and Receivers provide to the Director the number of cycles they implement, and the Director maintains for each iteration independent cycle counts for each Sender and Receiver. These counts allow the Director to access cycle-dependent interfaces on each Sender and Receiver (see Figure 3, top left).

The use of phases allows an implementer to divide a single cycle into a series of independent collective communications. One important use of phases derives from the MPI requirement that collectives (i.e., collective communications) have access on both the send and the receive side to the amount of data communicated. Therefore, for certain communication cycles, the amount of data to be sent must be communicated in one phase, followed by the data itself in a second phase. Senders and Receivers provide the Director with the number of phases per cycle that they implement, and the Director then loops through the execution of each phase, thus executing a single cycle of a Sender and Receiver, and thus completing communication for a single Communication Couple in the Director algorithm list (see Figure 3, bottom and upper right). The Director also checks for compatibility within a Communication Couple by ensuring that the number of phases specified by its Sender and Receiver is identical for each Sender and Receiver.

Senders and Receivers map cycle and phase numbers to data references necessary for MPI collective execution. These references may include pointers to a buffer, an array of data counts, an array of data displacements, and an array of MPI data types. In addition, Senders and Receivers use abstract interfaces to map cycle and phase numbers to function pointers on the Communicator object, thus specifying for each cycle and each phase which Communicator function is necessary to execute communication between Senders and Receivers (upper middle part of Figure 3). The Director checks for compatibility within a Communication Couple by ensuring that the specified function pointers are identical. Communicator functions provide wrappers for MPI collectives so that they can be referenced using the same function pointer type. Wrapping of MPI collective functionality is naturally accomplished by requiring that these functions take only Sender and Receiver references, Sender and Receiver cycle numbers, and a phase number as arguments. The Communicator implements these functions by requesting the appropriate data references from the Sender and Receiver, and then composing these data into arguments for MPI collectives. Because of isomorphism in the design of all MPI collectives, in some cases, this amounts to passing the references directly to the MPI, while in others, passing the de-referenced pointers to the MPI.

Computation: Preparing to communicate

The Director has a list of Communication Couples that allows a developer to conceive of and implement an algorithm as a series of MPI collective communication events. These communication events occur after each cycle of distributed computation performed over the data in distributed memory. Computation is initiated prior to communication for each Communication Couple by means of abstract Sender and Receiver interfaces. The names of these interfaces, *prepareToSend* and *prepareToReceive*, are intended to remind a developer that in this architecture, all computation is viewed as a preparation for communicating. Thus, all computation required to implement the parallel algorithm, as well as all computation required to support communication

interfaces described above, is initiated by the Director executing these two interfaces. Just as cycle and phase numbers are mapped to the communication interfaces described above, they are mapped to the appropriate communication preparation interface for each Sender or Receiver (see Figure 3, left). Each phase of a Communication Couple then consists of a preparation to send, a preparation to receive, and an execution of the Communicator function pointer returned by these two interfaces.

Parallel algorithm implementation

BlueBuilder tissue data format

Neuron data

The tissue specification file produced by BlueBuilder provides the number of neurons in the tissue, followed by an array representing the number of segments per neuron, and an array of offsets into the file from which each neuron can be read. To conserve memory, specific neuron data is accessed by reading an offset from the tissue specification file, seeking a new position in the file, and then reading that neuron data alone. The anatomical location of the center of the neuron, the layer of neocortex from which it was collected, its e-type, and its m-type are each read at this location. Neurons are then described by a number of branches, for which each branch is a contiguous set of points (**Figure 4**). (The terms "target" and "partition" in this figure are explained in the "Partitioning algorithm" section.)

Segment data

Branches are identified by a branch type (e.g., axon or dendrite) and branch order, and they are described by a number of segments, each of which consists of two points. Because segments in a branch are contiguous, points of a branch are represented uniquely in the file. Branches, however, have independent endpoints such that each branch is represented independently in the column specification file.

Column histogram

Because the work of contact detection is localized to specific regions within the tissue, a means of estimating the amount of contact-detection work that is present in an arbitrary tissue volume is required. Therefore, in addition to neuron description and structural data, BlueBuilder also exports a representation of the composition of the tissue per unit volume. This representation is read from files as three histograms representing the number of points projected onto each axis of the column per unit length (see **Figure 5** for a depiction of the axes). The bin width of the histogram is set by BlueBuilder and equals the mean segment length

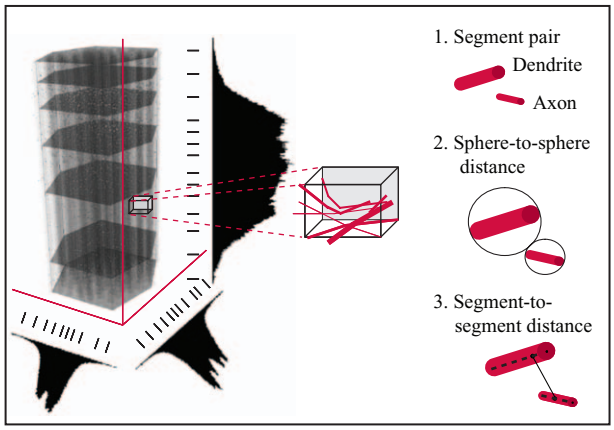

Figure 5

Column slice planes are determined by histogram equalization in each of three dimensions (left). Resulting voxels (middle) receive messages containing segments that pass through the voxel, as determined by the slicing algorithm. Contact detection proceeds on the basis of three criteria that are schematically illustrated (right).

projected onto each of the three axes, thus ensuring that the point count is a reasonable estimate of segment density along each axis. Bin width, therefore, represents the minimum granularity possible for dividing the work of contact detection.

Column partitioning

Trade-off: Load balancing vs. simple local data structures

The initial problem of load balancing involves dividing the work of data distribution among the processors of the system. Ideally, each processor would read an equal number of segments from file. Several additional factors caused us to consider a less precise load-balancing scheme for this stage of computation. A primary factor was the observation that the logical neuron data structure is useful for performing many neuro-scientific analyses and is best maintained in a single memory space. For example, modifications to the tissue for the purpose of circuit fitting are performed on a whole neuron in our current tissue revision scheme and, therefore, would require additional communication steps if neurons were divided among processors when loaded from a file. Also, for certain modifications, whole branches must be examined together, making the task of marshalling data for these calculations much simpler if the neuron data structure is maintained locally.

Partitioning algorithm

Partitioning the work of data distribution for contact detection requires that each processor load unique

neurons from file such that each processor has roughly an equal number of total segments to distribute. The array of segment counts for each neuron found at the beginning of the column specification file provides a means of partitioning the system without loading data from every neuron. We devised an adaptive partitioning algorithm that computes the total number of points in the column divided by the number of processors (see Figure 4). This value becomes a target number of segments for the partition of the first processor. Every partition is calculated in every processor. The partitioner traverses the segment count array until its running total of points exceeds the target minus one-half the number of points in the next neuron in the file. The partitioner then stores the first and last neuron in the partition, recalculates the optimum number of remaining segments for each remaining processor, and resets its running total to zero. If at any point the number of remaining neurons equals the number of remaining processors, one remaining neuron is assigned to each remaining processor. This continues in parallel for each processor until all neuron segment counts have been traversed and all partitions calculated.

Dividing neurons among column subvolumes

Logical structure: Neurons, branches, segments

On the basis of the calculated partition for each processor rank, neurons are loaded from file into a hierarchical data structure. The Neuron Partition object provides access to an array of Neurons. Neurons provide information concerning e-type, m-type, number of branches, and a reference back to the Neuron Partition. Branches are allocated and maintained by the Neuron, which provides access to its array of Branches. Branches provide access to the branch type, number of segments, a reference back to the parent Neuron object, and a pointer into an array of Segments. Segments are represented by the x, y, and z coordinates of two points, as well as the diameter of the segment, a segment descriptor, the segment index within the branch, and the offset of the segment into the partition. Thus, the hierarchical data structure provides access to all aspects of a neuron, regardless of which level of the hierarchy a piece of code can access (see Figure 4).

Segment data allocation

In our architecture, Segments could easily be allocated by Branches, just as Branches are allocated by Neurons. However, we identified an additional requirement for Segment allocation based on our strategy for data distribution using MPI collectives. This requirement states that whenever possible, all data to be communicated must be allocated within a contiguous region of memory in order to simplify the use of MPI collectives and MPI data types, and to avoid wasting time and memory copying data into a user-defined message buffer. Since the number of Segments in a partition is known before it is loaded from file, we were able to satisfy this requirement while maintaining the logical relationship between Neurons, Branches, and Segments. The contiguity of Segments in memory is implemented so as not to affect the logical data access hierarchy described above, thus keeping the system both simple to use and memory efficient.

Data decomposition: The 3D slicing metaphor

We have named the task of data distribution for contact detection "3D slicing," since the column itself is sliced into volumes as segments are sorted and sent to processors responsible for contact detection in each resulting volume. All processors in the system are responsible for both slicing neurons in a partition and performing contact detection in a volume. A user parameterizes a slicing scheme by setting the number of slices per dimension used to create volumes. The slices in each dimension are then equalized over the corresponding histogram so that slice boundaries create slices containing equal numbers of points. Linear interpolation allows slice planes to fall within a bin. By performing this equalization for each dimension, the volumes are themselves equalized to include approximately the same number of segments (see Figure 5).

Slicing consists of iterating through all segments in a partition of a processor and associating each partition with each of the volumes through which it passes. The association occurs by adding a segment offset within the partition to the send list of a destination processor. The method for determining which volumes receive a segment involves first finding all slice planes through which a segment passes in each dimension, and then solving a set of linear equations to determine the points of intersection between the segment and each of these slice planes. These points, together with the endpoints of a segment, provide the centers for a set of bounding cubes whose sides equal the diameter of the segment. If the opportunity for contacts between segments extends beyond the segment radius (e.g., if variable dendrite spine lengths are included in the tissue model [11]), the sides of these bounding cubes are extended accordingly. The volumes in which the bounding cube resides are then computed, and the indices of the volumes added to a temporary send list. This list is finally sorted, and its unique elements are used as indices into an array of send lists of the destination processors. The offset of the segment into the partition is then added to each list indexed in this way. Thus, the slicing algorithm ensures that each segment is associated with every volume in

which it might have contacts with another segment in the system.

Sender/slicer

The Column Slicer object implements the above algorithm and the Sender interface. It participates in two phases of communication. The first phase uses *MPI_alltoall* to communicate the number of segments to be sent to each processor. The second phase uses *MPI_alltoallw* to communicate the actual segment data. Segment counts are easily derived from the size of the send list for each destination processor. Segment data are read from the contiguous allocation in memory for the data using derived MPI data types. We chose *MPI_alltoallw* so that a different data type could be defined for each destination processor, representing the irregular pattern of segments in memory that must be communicated. Hence, the count and offset of the data sent are precisely one and zero for every processor sending segments. A dynamically constructed data type incorporates only data that needs sending from a Segment object (i.e., five double-precision floating-point values: x, y, z, diameter, and a descriptor) and composes multiple instances of this type into a single type using the offsets stored during slicing for each destination processor. The Receiver then receives segment data in a contiguous block of packed segment data, which can easily be traversed for the purpose of contact detection.

Detecting contacts in a subvolume

Receiver/detector

The detection of contacts between segments is performed by a Contact Detector object, which implements the Receiver interface. Data is received from the Column Slicer in a different format from the one in which it was sent. Because the Segment object data structure is useful only in the context of a complete neuron, the data received for contact detection within a column subvolume is received and stored in its primitive type form (in the context of object-oriented languages) as a contiguous allocation of double-precision floating-point values.

Segment descriptor, segment spaces

As noted already, segments include an 8-byte value that constitutes a segment descriptor. We have implemented a bit mapping for this value and interfaces that allow a user to access it as either a long integer (for the purpose of indexing) or a double (for the purpose of simple MPI communication with other doubles). The bit mapping allows a segment to maintain a compressed key that includes its layer (3 bits), m-type (5 bits), e-type (5 bits), branch type (2 bits), branch order (7 bits), segment index within its branch (10 bits), branch index within its neuron (13 bits), and neuron index within its class (19 bits). This mapping will suffice for tissues up to 500,000,000 neurons. The descriptor facilitates communication of user-defined contact categories and, thus, is the basis for user-defined contact tabulation categories.

Contact categories

Because contact detection is costly, we aimed to minimize unnecessary comparisons between segments that cannot constitute a logical contact. Thus, we allowed a user to specify an arbitrary contact category specification. For example, in most cases a user wants to detect only unidirectional contacts from axons to dendrites. By parameterizing the Contact Detector object with a contact category (axon or dendrite), each segment received by the Contact Detector will be considered with another only if it is part of an axon, and then only if the other segment is part of a dendrite. The application of contact categories at this stage minimizes total computation but also degrades the load-balancing scheme for the contact detection described above. The bit patterns of the segment descriptor are logically masked to extract its branch-type bits in order for this comparison to occur.

Geometry of segment capsule distances

We also minimize computation of contacts by testing whether candidate segments are within range for a contact to occur. Again, while total computation is decreased, load balancing is also degraded. Each segment is first enclosed in a sphere, and pairs of spheres are tested for overlap. If they intersect, the final calculation of segment-to-segment distance is made (see Figure 5). If this distance is less than the sum of the radii of segments, we accept this pair as having a contact. Additionally, we may relax this criterion by some amount specified for a particular neuron or branch combination, as may be useful for modeling specific axons contacting specific dendrites having spines of a particular length. This method of distance comparison treats segments as a cylinder with equal-diameter half-spheres on each end (which together are called "capsules") [16]. We accept this geometrical representation of a segment because it creates a reasonable approximation of the real neurite, and because use of it considerably simplifies the segment-to-segment distance calculation.

Local elimination of globally redundant contacts

Once all contacts within a volume are calculated, we identify those contacts that may also be detected in another volume on another processor. This situation occurs when both segments traverse a volume boundary together. We created a consistent method that examines

in which volume the contact point actually resides in order to prevent globally redundant contacts.

Tabulation, aggregation, analysis, revision

Global tables

Recall that circuit statistics, to which we aim to fit the contact data collected from our simulated tissue, derive from a variety of data collection methods that measure various rates of connectivity in the circuit. We have captured the current categories of these circuit statistics in the segment descriptors that are sent with each segment. By masking bit patterns in these 8-byte values for each pair of touching segments, we dynamically create generic keys into a local database where contact counts are accumulated. In this way, tables in the database can represent, for example, the number of contacts between particular e-types, particular m-types, or particular layers. Furthermore, descriptor categories can be masked together, for example, to yield a table comprising all unique touches by combining segment, branch, and neuron index bit-fields into a single key. The tables, thus, provide a powerful means for analyzing the tissue. In order for analysis and revision to proceed, however, these tables must be combined through another collective communication so that each processor aggregates global statistics from tables created locally.

Analyses and required tables

In addition to user-specified tables, a particular Analysis object may itself require specific tables of particular contact categories. Because tables are costly in terms of memory usage, a global table of all unique contacts, for example, often cannot be aggregated into a single memory space once a tissue reaches a particular size. Thus, we have allowed several unique tables to exist simultaneously in the system in order to provide a variety of descriptions of the contacts. If a user or Analysis object requests identical tables, only one is constructed and then shared between them. Users must carefully choose which tables to construct to ensure that memory usage does not exceed a maximum value after tables are aggregated.

Analysis objects derive from a generic interface. They implement an analysis generically using the tables that they require. They also signal when they no longer require iteration. In this way, circuit refinement can proceed until one or several Analysis objects meet some set of user-specified circuit target conditions (which are specified as a parameterization of an Analysis).

Analyzer implementation of Sender and Receiver interfaces

The Analyzer object comprises Analysis instances associated by the user and manages the aggregation of tables on each processor. The Analyzer implements both the Sender and the Receiver interface, since after communication, it replaces locally constructed contact tables with global aggregations of these tables from all processors. The tables are packed into a contiguous memory buffer after their dynamic creation during contact detection, and they are sent in a two-phase collection communication. In the first phase, the table sizes are communicated using *MPI_allgather*, and in the second, the actual tables are communicated using *MPI_allgatherv*. The Analyzer object also signals the main execution loop when all analyses are complete and iteration can be terminated.

Revision

Because analyses are performed on global tables aggregated on all processors in the system, the results of analysis are immediately available globally for the revision of the circuit. The revision currently involves individually rotating and translating each neuron. As the Analyzer traverses each Analysis that it contains, it aggregates rotations and translations produced by each Analysis for each neuron in the Neuron Partition object on each processor. In this way, the results of analyses are directly mapped onto circuit revision. The final rotations and translations are then used by the Neuron Partition to perform a rigid-body rotation/translation transformation on the points that represent each neuron. In addition to rigid-body transformations, the current architecture also supports neuron deformation to revise tissue composition and achieve a target set of circuit statistics. Neuron deformation could be used, for example, to recreate the specific relationships between different neuron branches observed in neurons that develop near one another in the same tissue, since the current model is composed of neurons that developed in different animals.

Example: Monte Carlo

Consider a circuit-fitting task that requires m-types for synapses based on a matrix of m-type-to-m-type connection probabilities. The difference between current contact statistics and the minimum target probability can easily be calculated by an Analysis object. This Analysis would require a table that globally aggregates the contact counts between each m-type-to-m-type pair. Recall that contacts provide the opportunity for a synapse to form; thus, in this example, touch counts within a touch category (divided by the total number of possible touches within the category) need only equal or exceed the target probability. Then, given some parameter that controls the rate of change (i.e., a "temperature") and the previous difference measurement, the Analysis could calculate how much each neuron in the tissue should be rotated or translated. This calculation

would result in tissue revision, as described above. When the tissue no longer requires revision, it has achieved some optimum by means of a Monte Carol simulation.

Performance

The performance of the calculation is measured on the basis of memory usage, computation time, and communication time for each step in the algorithm. Our target tissue for these measurements was a column comprising 10,000 neurons and 40 million segments. The average segment radius was 0.23 ± 0.24 μm, and average segment length was 3.65 ± 3.2 μm. The cylindrical column was constructed in a volume $550 \times 1{,}200 \times 550$ μm, and the neurites of its neurons extended in a volume $4{,}400 \times 3{,}300 \times 4{,}300$ μm. The most complex cell in the simulation included 19,000 segments. The total time required to compute 28,456,789 contacts using 8,000 processors of the BG/L system was 55 seconds.

Performance showed supralinear scaling from 216 to 3,375 processors running in the virtual node mode of the BG/L system [17] [**Figure 6(a)**]. (When we use the term *supralinear*, we refer to the fact that as we added more processors, we observed a greater proportional increase in performance than predicted by a linear relationship.) We compiled the application using the blrts_xlc++ compiler with `-O4` and `-qhot` optimization flags. Good performance scaling derives from both the amount of work and the memory footprint decreasing as column subvolumes become smaller, resulting in fewer operations and better cache utilization on each node. Each dimension of the column was divided into an equal number of slices. We observed that load imbalance also grew as a function of the number of processors. In particular, load imbalance grew for the most demanding step of the algorithm, contact detection [**Figure 6(b)**], in which greater than 95% of the computation time of the algorithm occurs, for reasons discussed above. Finally, we observed that the maximum percentage of time spent executing MPI collective communication (minus wait times) was less than 1% for all numbers of processors less than 2,744. This percentage grew to 3.5% for 3,375 processors, then rapidly to 20.8% for 8,000 processors, indicating that the application remains compute bound at the current scaling.

Conclusion

The architecture described in this paper succeeds in rapidly detecting contacts between branched neuron morphologies using the BG/L supercomputer. Understanding the microcircuitry of neural tissue, such as the neocortical column, is a necessary step toward understanding the computation it performs and the contribution it makes to global brain function. In addition, detailed models of neural tissue can one day help to model diseases that target the physical integrity of these tissues.

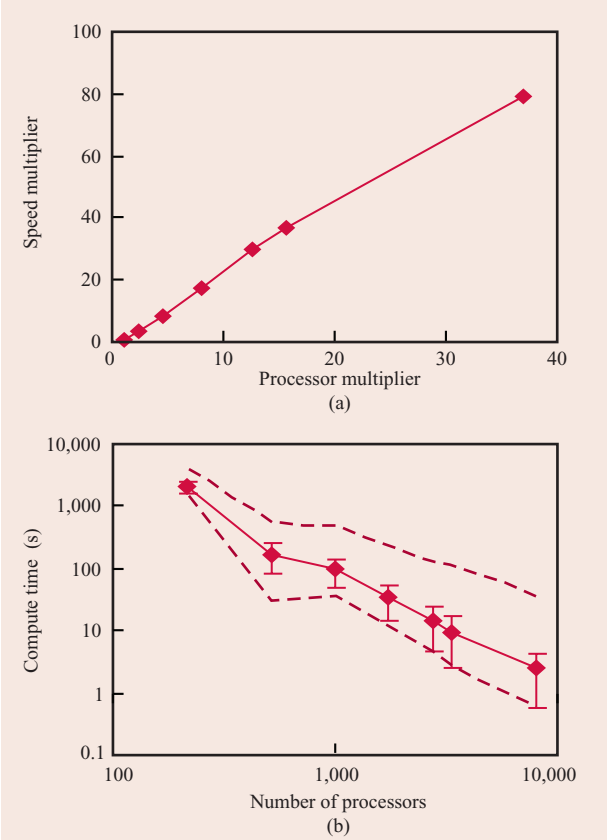

Figure 6

Scaling and load balancing. The iterative algorithm shows (a) supralinear speedup from 216 to 3,375 processors despite growing load imbalance in the (b) compute-bound step of the iterative algorithm involving contact detection. (In the bottom graph, the solid line indicates the mean. Bars indicate standard deviation. The dashed lines show maximum and minimum values. A total of 10,000 neurons were used for both graphs.)

The architecture discussed here was specifically designed for the communications and memory architecture of the BG/L platform and, thus, made extensive use of MPI collectives, since the BG/L supercomputer is optimized for efficient use of these collectives [18]. The use of a generic set of interfaces for specifying steps in the algorithm and communication on step boundaries has allowed rapid development and extension of the current algorithm to accommodate additional circuit-fitting exercises.

We note that additional uses for this architecture include memory-intensive applications that require local analysis of globally distributed data structures. One such application is 3D image segmentation. Stacks of

images are generated in many areas of biology, including automated microscopy of neural tissue [19]. Often, structures in the tissue must be traced through many images in a stack. The process of identifying, aligning, and reconstructing these structures is computationally intensive [20]. Using the BG/L platform and the current generic application interfaces to process image stack data in parallel for rapid reconstruction of 3D structures observed in microscopic images of neural tissue is, therefore, another domain for application of this architecture.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

References

1. S. Ramón y Cajal, *Recuerdos de Mi Vida: Historia de Mi Labor Científica*, Alianza Editorial, Madrid, 1923.
2. S. Ramón y Cajal, *La Textura del Sistema Nerviosa del Hombre y los Vertebrados*, Moya, Madrid, 1904.
3. A. L. Hodgkin, A. F. Huxley, and B. Katz, "Measurement of Current–Voltage Relations in the Membrane of Giant Axons of Loligo," *J. Physiol.* **116**, 424–448 (1952).
4. A. Gupta, Y. Wang, and H. Markram, "Organizing Principles for a Diversity of GABAergic Interneurons and Synapses in the Neocortex," *Science* **287**, No. 5451, 273–278 (2000).
5. A. L. Hodgkin and A. F. Huxley, "A Quantitative Description of Membrane Current and Its Application to Conduction and Excitation in Nerve," *J. Physiol.* **117**, 500–544 (1952).
6. F. A. Dodge, Jr. and J. W. Cooley, "Action Potential of the Motorneuron," *IBM J. Res. & Dev.* **17**, No. 3, 219–229 (1973).
7. M. Hines, "Efficient Computation of Branched Nerve Equations," *Int. J. Biomed. Comput.* **15**, No. 1, 69–76 (1984).
8. C. Koch and I. Segev, Eds., *Methods in Neuronal Modeling*, MIT Press, Cambridge, MA, 2001.
9. R. D. Traub and R. K. Wong, "Synchronized Burst Discharge in Disinhibited Hippocampal Slice. II. Model of Cellular Mechanism," *J. Neurophysiol.* **49**, 459–471 (1983).
10. L. Alonso-Nanclares, S. Anderson, G. Ascoli, R. Benavides-Piccione, A. Burkhalter, G. Buzsaki, B. Cauli, et al., "Petilla 2005: Nomenclature of Features of GABAergic Interneurons of the Cerebral Cortex"; see *http://www.columbia.edu/cu/biology/faculty/yuste/petilla/petilla-webpages/Nomenclature/PetillaNomenclaturefinal.pdf*.
11. A. Stepanyants and D. B. Chklovskii, "Neurogeometry and Potential Synaptic Connectivity," *Trends Neurosci.* **28**, 387–394 (2005).
12. K. Sfyrakis, F. Schuermann, A. Jan, and H. Markram, "BlueBuilder: Building the Neocortical Column According to Recipe," *FENS Forum 2006—Abstracts*, A037.16, 2006.
13. J. R. Glaser and E. M. Glaser, "Neuron Imaging with Neurolucida—A PC-Based System for Image Combining Microscopy," *Comput. Med. Imaging. Graph.* **14**, No. 5, 307–317 (1990).
14. H. Markram, "A Network of Tufted Layer 5 Pyramidal Neurons," *Cerebral Cortex* **7**, 523–533 (1997).
15. J. Kozloski, F. Hamzei-Sichani, and R. Yuste, "Stereotyped Position of Local Synaptic Targets in Neocortex," *Science* **293**, No. 5531, 868–872 (2001).
16. D. Eberly, Intersection of Cylinders, Geometric Tools, Inc. (2000); see *http://www.geometrictools.com*.
17. N. R. Adiga, G. Almasi, G. S. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, et al., "An Overview of the Blue Gene/L Supercomputer," *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2002, pp. 1–22.
18. G. Almási, C. J. Archer, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng, "Optimization of MPI Collective Communication on Blue Gene/L Systems," *Proceedings of the 19th Annual International conference on Supercomputing*, ACM Press, 2005, pp. 253–262.
19. A. Can, H. Shen, J. N. Turner, H. L. Tanenbaum, and B. Roysam, "Rapid Automated Tracing and Feature Extraction from Retinal Fundus Images Using Direct Exploratory Algorithm," *IEEE Trans. Inform. Technol. Biomed.* **3**, No. 2, 125–138 (1999).
20. K. A. Al-Kofahi, S. Lasek, D. H. Szarowski, C. J. Pace, G. Nagy, J. N. Turner, and B. Roysam, "Rapid Automated Three-Dimensional Tracing of Neurons from Confocal Image Stacks," *IEEE Trans. Inform. Technol. Biomed.* **6**, No. 2, 171–187 (2002).

Received March 15, 2007; accepted for publication April 10, 2007; Internet publication December 11, 2007

James Kozloski *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (kozloski@us.ibm.com).* In 1999, Dr. Kozloski received his Ph.D. degree in neuroscience from the University of Pennsylvania. He subsequently held a research position at Columbia University in the lab of Dr. Rafael Yuste, where he discovered stereotyped positions of local synaptic targets in neocortex. He joined the research staff of IBM in 2001, and in 2006, he was also named Adjunct Assistant Professor at Columbia. Dr. Kozloski's research interests, primarily in computational biology, include structural biology, neural system modeling, functional simulations of neocortex, and molecular biology. He invents in the area of neurotechnology, and designs parallel computing software architectures and interfaces for both simulation and data analysis problems in neuroscience.

Konstantinos Sfyrakis *Blue Brain Laboratory, Brain Mind Institute, Faculté des Sciences de la Vie, Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland (konstantinos.sfyrakis@epfl.ch).* In 2001, Dr. Sfyrakis received his Ph.D. degree in computational chemistry at Surrey University, School of Biomedical and Molecular Science, United Kingdom. He subsequently worked as a postdoctoral fellow at the Bernoulli Institute of Mathematics, at EPFL, Switzerland, before joining the Research and Development Group of the Brain Mind Institute at EPFL. Currently, he designs and writes computing software applications and interfaces for scientific problems in neuroscience.

Sean Hill *IBM Research Division, Thomas J. Watson Research Center and the Blue Brain Project, Brain Mind Institute, Ecole Polytechnique Fédérale de Lausanne (EPFL), Station 15, 1015 Lausanne, Switzerland (seh@zurich.ibm.com).* In 2000, Dr. Hill received his Ph.D. degree in computational neuroscience from the University of Lausanne. He subsequently joined the Research Group of Dr. Giulio Tononi at the Neurosciences Institute in La Jolla, California, and then moved with Dr. Tononi to the University of Wisconsin, Madison, in 2001. He has developed numerous large-scale models of neural systems and is the designer and developer of the general-purpose neural simulator, Synthesis. As part of his postdoctoral research, he developed the first large-scale thalamocortical model that replicates neural activity during wakefulness and sleep. He joined IBM Research and the Blue Brain Project in May 2006 and now serves as Project Manager for computational neuroscience area. His research interests include the use of large-scale biologically realistic computer models to understand information processing, network connectivity, and synaptic plasticity in the brain.

Felix Schürmann *Blue Brain Laboratory, Brain Mind Institute, Faculté des Sciences de la Vie, Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland (felix.schuermann@epfl.ch).* Dr. Schürmann is the General Project Manager of the Blue Brain Project and a postdoctoral fellow at the Brain Mind Institute at the EPFL. He started his studies of physics at the University of Heidelberg, Germany, supported by the German National Academic Foundation. He obtained his M.S. degree in physics from the State University of New York, Buffalo, under the supervision of Richard Gonsalves. During this time, he was a Fulbright Scholar. His master's thesis dealt with the foundations of computing, including the simulation of quantum computing. In 2005, he received his Ph.D. degree in physics from the University of Heidelberg, Germany, under the supervision of Karlheinz Meier. His work focused on alternative approaches to computing. Using mixed-signal very-large-scale integration (VLSI), he co-designed an efficient implementation of a neural network in hardware and was the first to adopt the theory of liquid computing in hardware.

Charles Peck *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (cpeck@us.ibm.com).* In 1994, Dr. Peck received his Ph.D. degree in electrical engineering from the University of Cincinnati. He currently leads the Biometaphorical Computing Research Group at IBM, dedicated to analyzing and modeling the brain for scientific, medical, and technology applications. This work includes data-driven modeling via the Blue Brain collaboration with Ecole Polytechnique Fédérale de Lausanne, as well as theory-driven modeling of global brain function and individual structures, such as the cortex, cerebellum, and basal ganglia. In 1998, while at the Lockheed Martin Corporation, Dr. Peck was awarded the NOVA Award for Technical Excellence, the corporation's highest honor. He was also selected by the National Academy of Engineering as one of America's top young engineers.

Henry Markram *Blue Brain Laboratory, Brain Mind Institute, Faculaté des Sciences de la Vie, Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland (henry.markram@epfl.ch).* Project Director of the Blue Brain Project, Director of the Center for Neuroscience and Technology, and Co-director of the Brain Mind Institute at EPFL, Dr. Markram received his Ph.D. degree from the Weizmann Institute of Science, was a Fulbright Scholar at the National Institutes of Health, and a Minerva Fellow in the Laboratory of Bert Sakmann at the Max Planck Institute, Heidelberg, Germany. Dr. Markram's many discoveries include being the first to alter the precise relative timing of single presynaptic and postsynaptic action-potentials to reveal spike timing-dependent synaptic plasticity. As an assistant professor at the Weizmann Institute for Science, Israel, he began systematically analyzing the neocortical column, discovering novel synaptic learning mechanisms and a spectrum of new principles governing neocortical microcircuit structure and function. Together with Wolfgang Maass, he developed the theory of liquid computing. In 2002, he moved to EPFL as full professor, founder, and director of the Brain Mind Institute as well as director of the Center for Neuroscience and Technology.