Event-Related Simulation of Neural Processing in Complex Visual Scenes

Stefan Mihalas, Yi Dong, Rüdiger von der Heydt, Ernst Niebur
Zanvyl Krieger Mind/Brain Institute
and Department of Neuroscience
Johns Hopkins University
Baltimore, MD 21218

***Abstract*—Understanding the neural processing of complex visual scenes is facilitated by tools that allow efficient simulation of large neural systems. We here present an environment for the implementation of large networks of generalized integrate-and-fire neurons which uses an asynchronous event-based algorithm. It allows accurate (up to machine precision) calculation of neuronal spike times. Its efficiency arises from its ability to solve the subthreshold dynamics of the model analytically. The simulator is implemented in JAVA and it can run in parallel on a large number of computers with different architectures without the need for compilation. The neuronal network to be simulated and all parameters are defined in extendible markup language. A model of the primate early visual system is implemented. The use of the tool is illustrated by simulating the processing of both simple and complex visual scenes through retina, thalamus and primary visual cortex.**

I. Introduction

Primates use their complex nervous systems to process the large amounts of information that is generated in the rich perceptual environments they inhabit. To understand the underlying mechanisms, it is necessary to develop quantitative models that complement heuristic approaches. While analytical insight is precious, the complexity of these models usually requires numerical solutions, *i.e.,* computer simulations. We here introduce a software package that is suitable for the simulation of large networks of interacting neuron. The underlying neuronal model is analytically solvable between spikes, allowing an event-driven implementation that determines spike times at machine precision. The package is designed to run on heterogeneous clusters of commodity computers. A preliminary version of the simulator was presented in abstract form [1].

A. Single neuron model

The native single neuron model implemented in our simulator is the generalized integrate-and-fire neuron [2]. It is capable of producing spiking/bursting, tonic, phasic or adapting responses, depolarizing and/or hyperpolarizing afterpotentials *etc.* The richness of the model behavior is due to a variable threshold in combination with an arbitrary number of firing-induced currents [3]. The equations governing the time evolution of the membrane potential are solved analytically between spiking; this can be done very efficiently since the model dynamics can be writtnen as a diagonalizable set of linear differential equations [2].

B. Implementation

All of our simulations consist of two stages. The first is the construction of the network to be simulated. The importance of this task, and the effort required, is frequently underestimated; constructing a network of thousands (or more) of neurons with realistically structured connectivity is a substantial task. The major effort in this stage is the definition of the network connectivity since even a medium-sized network (a few thousand neurons) can have hundreds of thousands or more synaptic connections. Just as in nature (where connections are not individually "hard-coded" in the genome), it would be very inefficient to specify each individual connection. Instead, pathways are defined in the form of mappings from one network part to another. As a principle, we always use the highest possible level of description to define a connectivity pattern. For instance, rather than constructing thousands of connections between two populations on an individual basis, we instead define the generic properties of the connections in this pathway parametrically, and the simulator then generates the actual connectivity pattern autonomously. As a practical matter, the simulator can be instructed to subdivide the network in subnetworks and assigns them to the available machines, with the objective of keeping as many connected neurons as possible on the same machine, and thus minimizing communication between machines (Fig. 1A).

The second stage is the simulation of network function. All communications between model neurons are done by exchange of spikes (action potentials). A neuron whose voltage exceeds its threshold generates a spike event that is propagated with appropriate parameters (mainly synaptic weight and delay) to all its postsynaptic partners. The simulation is usually structured into "trials" and these trials are distributed in an efficient way among the available machines (Fig. 1B). The rationale is that a realistic simulation of an electrophysiological experiment should reflect that an experiment is usually repeated many times and the final results are then obtained from averaging. This is simulated by running the same model with different random seeds.

The code is machine-independent (written in JAVA) and the description language is non-proprietary (written in XML).

C. Simulator structure

The simulator is composed of two parts, the machine-level layer and the user interface. The machine-level layer organizes the events and the communication between them. The user interface translates the network description (which is in both human-readable and machine-readable XML format) and the setup of the experiment(s) into the code for the machine-level layer, and provides visualization tools to present the output of the simulation.

Mapping of the network onto host machines At the level of the physical hardware, the simulator has a three-level pyramid structure as shown in Figure 1A. At the top of the pyramid, a single "MasterHost" assigns trials to different "TrialHosts" and collects information from the simulation. We define one trial as one realization of the stochastic network simulation, with all trials identical except for a different random number seed. The number of trials simulated simultaneously is equal to the number of TrialHosts. If, as is usually the case, the number of trials surpasses that of TrialHosts, a new trial is assigned to a TrialHost as soon as it has completed the previous trial.

At the second layer of the pyramid, TrialHosts distribute trials over "NetHosts." The simulator is designed for networks that are much larger than can be implemented on one commodity (off-the-shelf) computer. Therefore, the network is typically simulated on a cluster of machines and each TrialHost assigns different parts of the network to different NetHosts. Each TrialHost is responsible for the simulation of one instance of the complete network and an adequate number of NetHosts must be assigned to each Trialhost by the user on the Masterhost. Each TrialHost together with its associated NetHosts carries out a simulation of the entire network. As soon as it finishes the simulation of the trial, a TrialHost is assigned a new simulation by the MasterHost, and this continues until all trials have been completed.

In the simple example shown in Fig. 1A, we assume that four simulation machines are available as NetHosts (in addition to the MasterHost and the two TrialHosts), and that two of them are sufficient to simulate the network. Therefore, two trials will be run simultaneously. Fig. 1B shows how a network of 12 neurons is divided into two parts, and how each of these parts is simulated on one NetHost. In the example network, as in most biologically realistic networks, neurons send most of their synapses to their neighbors. Therefore, to the extent possible, the network is partitioned such that neighboring neurons are mapped to the same NetHost. In the example, all spikes except those from neurons 3 and 4 stay within one NetHost.

We adopted the synaptic structure proposed by ref. [4]. As shown in Fig.1B, each NetHost stores an array of target hosts and a hashmap of synapses. The array of target NetHosts contains a given NetHost ID if the neuron has synapses that are presynaptic to neurons implemented on that NetHost. If, on the other hand, a post-synaptic neuron is implemented on the same NetHost as the presynaptic neuron, synapses are stored on this NetHost in a hashmap, with the neuron index as the key and the value as an axonal branch (an axonal branch, discussed in the next section, contains several synapses). We use a hashmap to minimize RAM usage. In the example, neuron 9 is implemented on NetHost 1 and it receives synapses from neurons 2, 3, and 4. Therefore synapses $2 \rightarrow 9, 3 \rightarrow 9, 4 \rightarrow 9$ are stored on NetHost 1. Since neuron 4, implemented on NetHost 2, has a synapse targeting neuron 9, the list of target NetHosts of neuron 4 includes NetHost 1. By using this synaptic structure, all synaptic weights and delays are stored on the same NetHost as the neuron they synapse onto. This significantly reduces communication needs between hosts and it is easy to implement synaptic plasticity since synaptic weights can be changed locally.

Input queues The most frequent events in the simulation are neurons receiving input (on their dendrites) since each firing of an action potential is one event in the queue of neuronal output events but it results in many (one for each postsynaptic neuron) in the queue of neuronal input events. For the sake of efficiency, we reduce the size of the input queues by splitting them into two parts each, one for network-internal spikes and one for network-external spikes. The internal input spike queue only contains input spikes generated in the simulation, while the external input spike queue handles spikes from external sources (e.g. background spikes, or spikes resulting from simulated sensory input). The size of the input queues is further reduced by organizing them by axonal branches. As in biology, each simulated axonal branch consists of the synapses onto all neurons that are implemented on the same NetHost and that, furthermore, have the same synaptic delay. When a neuron fires, the simulator inserts the events from the axonal branches into the corresponding input queues, in the order of the time when the post-synaptic neurons receive the synaptic input (as given by the spike time and the delay). All synaptic inputs from one axonal branch thus arrive at the same time and those inputs are processed sequentially by the postsynaptic neurons.

Synchronization between hosts Communication between TrialHosts and MasterHost is simple. After completing the simulation of a trial, the TrialHost notifies the MasterHost which then assigns it a new trial number, and this procedure continues until all trials are finished.

The role of the TrialHosts is to organize trials for the NetHosts. At the beginning of the trial, each TrialHost broadcasts the trial information to the NetHosts and sends a go-signal to the NetHosts to begin the simulation. When a NetHost finishes its task, it notifies its TrialHost. Once all NetHosts are finished, the trial is completed. It should be obvious that most of the work is performed by the NetHosts; indeed, the structure of the simulator is such that NetHosts are utilized to their maximum while TrialHosts and the MasterHost serve to facilitate and organize the computation/communication needs of the NetHosts. As such, the computational demands on MasterHosts and TrialHosts are usually minor. Frequently, the MasterHost is simply the workstation that the user is submitting the job from and the role of Trialhosts is performed by one of the machines whose main

role is that of a NetHost.

D. User interface

Users define the model in an XML description file which completely describes the network to be simulated, all its inputs, the choice of which variable values will be read out, the number of repetitions ("trials"), noise levels,*etc*. The syntactical correctness of the XML file is verified by a validator. In this section, we discuss user choices.

Connectivity Much of the complexity of a neural network, biological or artificial, results from the patterns of the connections between its components. In the simulator, the components of the connectivity can be specified using several complementary methods.

The most flexible is the explicit definition of each connection. While this is useful in some cases, for networks of the size the simulator is designed for it would be very inefficient to use this method for all connections, and it would also be biologically unrealistic. Biological neural networks typically do not consist of individually grown connections but, instead, of pathways. The details of how these pathways develop are presently insufficiently understood but developments likely requires much less *a priori* information (e.g. in the genome) than would independent specification of each single connection. In the simulator, the functional equivalent of this collective control strategy is to always use the description of the connection structure at the highest possible conceptual level. Of particular importance are two-dimensional layers of neurons and the connections between them. All connections pathways are between two-dimensional layers of neurons, including the already mentioned individually specified connection pattern which is defined as a matrix of connection parameters (see next paragraph) between the originating and target layers. A more parsimonious connection specification scheme is a convolution-based pattern in which a small matrix defines the connection strength. This is then repeated akin to a two-dimensional convolution operation. Furthermore, both the individually specified and the convolution-based connection patterns can be either deterministic (i.e., completely specified by the connection or convolution matrix) or probabilistic, and each pattern can be either "convergent" (many-to-one from the source layer to the target layer) or "divergent" (one-to-many).

Each synaptic connection is characterized by several parameters: the neurotransmitter (glutamate, GABA, or other) which determines the dynamical postsynaptic conductance changes effectuated by the activation of the synapse, the synaptic delay, and the "strength" of the synapse. The latter is defined as the maximal transmembrane current controlled by the synapse.

Experiments Simulations are organized analogously to electrophysiological experiments. Each simulated Experiment consists of SubExperiments with different external inputs ("stimuli") presented in each of them. Stimuli are either simple geometrical objects (e.g., "Line", "Square") or they are read in from external bitmap images (in gif, jpg, and bmp formats). Stimuli impinge onto the specified neuronal layer(s) through activation of synapses whose input type, strength, rate, and rate decay are all user-defined. For each SubExperiment, the length of the simulation and the number of repetitions to be performed with different random number seeds are defined.

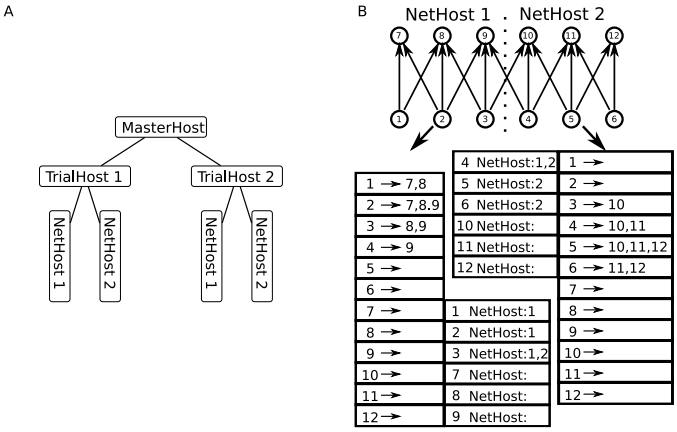

Fig. 1. Example of subnetwork structure at the machine layer of the simulator. (A) Overall structure of the simulator (see text for details). In the example, two trials are simulated simultaneously, on TrialHosts 0 and 1, respectively. (B) Distribution of neurons of the simulated network by one of the TrialHosts onto its two NetHosts. In the example 2-layer feed-forward net shown at the top, an arrow indicates a synaptic connection and a circle indicates a neuron, with its index inside the circle. This network is divided into two parts and distributed onto the two NetHosts. The short array associated with each NetHost (bottom) lists the neurons simulated on this NetHost as well as to on which NetHosts the neurons they project to are located: NetHost 1 simulates neurons 1,2,3,7,8,9, NetHost 2 simulates neurons 4,5,6,10,11,12. The right-hand side of the long array in a NetHost contains a list of all neurons that receive synaptic input and are simulated on this NetHost, with the presynaptic neuron on the left-hand side of the arrow. Spikes from neurons 3 and 4 are exchanged between NetHosts, all others stay on the same NetHost.

Recorder For large network simulations, it is impractical to monitor in detail the states of all, or even most, neurons. In our simulator, we chose to provide output in a format similar to that recorded by experimental neurophysiologists. *Single-unit* electrodes record the spiking times of individual neurons in a format suitable for presentation in a raster plot. The *multi-unit* electrodes record the firing activity of a small pool of neurons. *Intracellular* electrodes record neuronal membrane voltage and the currents through different transmembrane channels. An example of the latter would be the current through the glutamate receptors in an excitatory synapse. We also define a *Field* represetation that shows the firing activity in a whole layer of neurons, akin to data provided by optical imaging methods. The data can be visually represented by slides of images, each slide representing 1 time bin (of length chosen by the user) and each neuron 1 pixel of the image.

II. Results

A. A realistic simulation example

To demonstrate the usefulness and performance of the simulator, we show an example of a model of the primate early visual system including retina, lateral geniculate nucleus (LGN) and primary visual cortex (V1). The goal of this system is to extract contour information at the level of complex cells in primary visual cortex. The neuronal network is a feed forward network composed of leaky integrate-and-fire neurons.

The first spiking layer in the primate visual system is comprised of retinal ganglion (RG) cells. In the model, the firing rate of these cells is determined by center-surround filters consisting of a difference of Gaussians (center: standard deviation 1 pixel, weight 1; surround: standard deviation 3 pixels, weight 0.7). On-center ganglion cells (neuron layer has prefix RG and suffix ON) respond to high-intensity centers and low intensity surrounds, and the opposite is the case for off-center ganglion cells (prefix RG, suffix OFF). LGN neurons have two types of center surround-receptive fields, on-center off-surround (the layer name has a prefix ONLGN) and off-center on-surround (prefix OFFLGN), and each comes in two types, excitatory (suffix "E") and inhibitory (suffix "I"). All LGN neurons receive specific connections from retinal ganglion cells. Excitatory cells receive additional input from the inhibitory LGN neurons. Receptive fields of some LGN cells are illustrated in Figure 2A (retinal cells are included in the simulation but their receptive field properties are not illustrated for the sake of brevity).

Orientation selectivity is generated in our cortical model based on the mechanism proposed by Hubel and Wiesel [5]. Simple cells in primary visual cortex (V1) receive synapses from LGN neurons whose receptive fields are linearly organized and thus respond best to oriented lines. We consider simple cells in V1 at 3 spatial scales. Scale 1 is the fine scale (full pixel resolution), scale 2 is averaged over 2×2 pixels, and scale 4 is averaged over 4×4 pixels. There are two major types of simple cells, with receptive fields having odd (prefix "SO") and even (prefix "SE") symmetry relative to their centers, respectively. We only consider 2 preferred orientations, horizontal and vertical. As shown in Figure 2B, there are 4 different combinations for the even and odd simple cells for each of the 3 scales (suffix "L" and "R" for preferred tuning orientation vertical, "U" and "D" for preferred tuning orientation horizontal). "SO" simple cells receive input from one line of "ONLGN,E" neurons at one side and from another line of "OFFLGN,E" neurons at the other side, resulting in a receptive field with orientation tuning (Fig. 2B). Without loss of generality, we only describe the connectivity of "SO,L" neurons. The "SO,L" simple cells of scales 1, 2 and 4 at coordinate of (x, y) receive connections from"ONLGN,E" neurons at (x', y') with synaptic weights

$$\frac{1}{n} e^{-\frac{(x'-x+\mu)^2}{2\sigma^2}} H(a-y)H(x+a)$$

and from "OFFLGN,E" neurons with weights

$$\frac{1}{n} e^{-\frac{(x-x'+\mu)^2}{2\sigma^2}} H(a-y)H(x+a)$$

where $H(x)$ is the Heaviside step function, $\mu = 1, 2, 4$, $\sigma = 0.5, 1, 2$, $a = 2.5, 5, 10$ for the 3 scales, and n is a normalization factor.

Similarly, simple cells with even symmetry receive input from one line of excitatory LGN neurons in the center and from two lines of LGN neurons of the opposite on/off selectivity inputs in the flank (see Fig. 2B). The "SE,L" simple cells

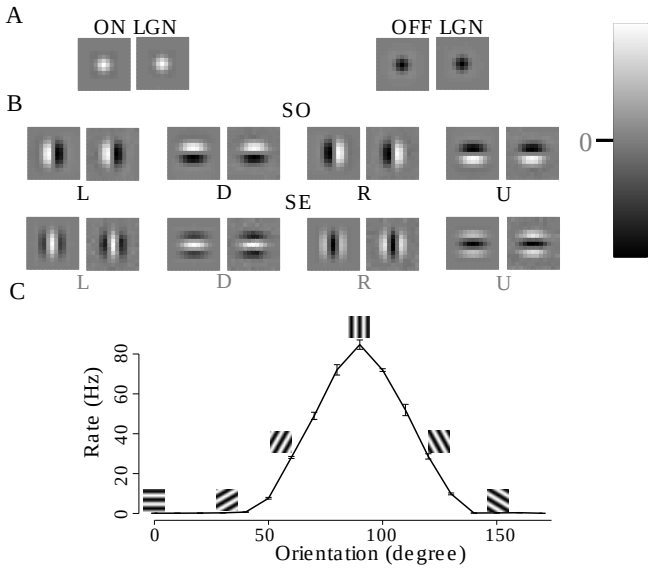

Fig. 2. Receptive fields of LGN and cortical simple cells calculated by the convolution of the connection matrices (receptive field shown on the left of each pair) and by reverse correlation using simulated neural responses (right of each pair). Scale (right) shows excitation (bright) and inhibition (dark), with neutral marked by "0". **A,** Receptive fields of "ONLGN,E" and "OFFLGN,E" thalamic neurons. **B,** Receptive fields of simple cells in V1. Top, odd symmetry; bottom, even symmetry. **C,** Observed (simulated) orientation tuning curve for a complex cell. This neuron is optimally responsive to a vertical sinusoidal grating. The curve is a Gaussian fit ($83.61 \exp(-(x - 90.34)^2/772.245)$, where x is orientation of the grating in units of degree) and the half width of the orientation tuning curve is found to be $46.28°$.

of scale 1, 2, and 4 at location (x, y) receive connections from "ONLGN,E" neuron at (x', y') with synaptic weightss of

$$\frac{1}{n} e^{-\frac{(x'+\mu)^2}{2\sigma^2}} H(a-y)H(x+a)$$

and from "OFFLGN,E" neuron at (x', y') with weights

$$\frac{1}{n} (e^{-\frac{(x'-\mu)^2}{2\sigma^2}} + e^{-\frac{(x'+\mu)^2}{2\sigma^2}}) H(a-y)H(x+a)$$

where μ, σ, a and n are the same as for SO neurons.

For fairly strong feed-forward connections and for essentially linear neuronal responses, receptive fields can be obtained from the patterns of input connections. To verify that these assumptions are satisfied (and also to demonstrate how our simulator very naturally allows to apply well-known data-analysis methods from experimental neurophysiology), we characterized receptive fields using the reverse correlation method. We presented the simulated neuronal network with sequences of stimuli comprised of random dots, each frame lasting $10ms$. The spike-triggered average of the responses to these stimuli approaches the expected linear kernel of the response and is shown for several thalamic and cortical neurons in Figure 2A,B (left: linear kernel, right: receptive field from spike-triggered average). The receptive fields obtained from reverse correlation match the fields computed from the convolution of the connection matrices very well.

In Figure 3, we show simulated population activity of large populations of neurons. Shown are the (gray-level coded) mean

firing rates of retinal, thalamic and cortical populations in our simulator. The activity is in response to the presentation of the gray-scale input image shown in Figure 3A. The picture has a resolution of 144×216, and we accordingly use this as the size of the network layers.

After processing in retina and thalamus (not illustrated), results are passed on to primary visual cortex (our model neglects the substantial feedback from cortex to thalamus). After processing by simple cells, whose receptive field properties are shown in Figure 2 but whose population activity is not illustrated for lack of space, the next stage consists of complex cells. They are characterized by largely phase-invariant responses which are are obtained by convergent input from simple cells. Examples of complex cell responses (prefix "C") are shown in Figure 3B,C for horizontal and vertical preferred orientations, respectively. In Figure 3D we show the combination (*i.e.,* the sum) of horizontal and vertical complex cells, highlighting the edges of all orientations represented in the simulation.

Simple cells in cat visual cortex receive spatially opponent excitation and inhibition input [6]. This inhibitory input most probably originates in other simple cells and may serve to reduce noise [7] or to sharpen orientation selectivity [8]. In our model, inhibitory simple cell layers receive the same connections as excitatory simple cells but are again not shown.

B. Performance and Scaling

The network described in section II-A consists of 921,456 neurons with 61,990,272 synapses. The simulated experiment illustrated in Figure 3 consists of 16 repetitions and all neurons in the simulated 14 layers are recorded simultaneously, Running on a cluster of 8 dual-core machines (AMD Athlon 64 4400) with 16 NetHosts and 1 TrialHost, the full simulation takes $58m\ 38s$, including $34s$ for the construction of the network.

Performance of the simulation for this network is illustrated in Figure 4, separately for the construction of the network (triangles), updating of network states (crosses) and total (circles). Shown is the number of cores times computation time, a quantity which is constant if scaling is linear. When using up to 8 cores, the simulator does scale linearly with the number of cores. When the number of cores exceeds 8, the performance becomes sub-linear for both network construction and running the network. This behavior is to be expected since up to 8 cores, each CPU has full access to hardware resources like memory, network assets, cache, *etc* on this machine. Once the second core on a given machine needs to be included in the simulation, the two cores need to share these resources and performance scales (slightly) sub-linearly. In addition, increasing network size increases communication needs between machines.

An interesting effect is that scaled performance does not vary monotonically with the number of cores. Notably, network construction time takes slightly longer for 7, 11, and 13 cores than for adjacent numbers. We attribute this effect to the

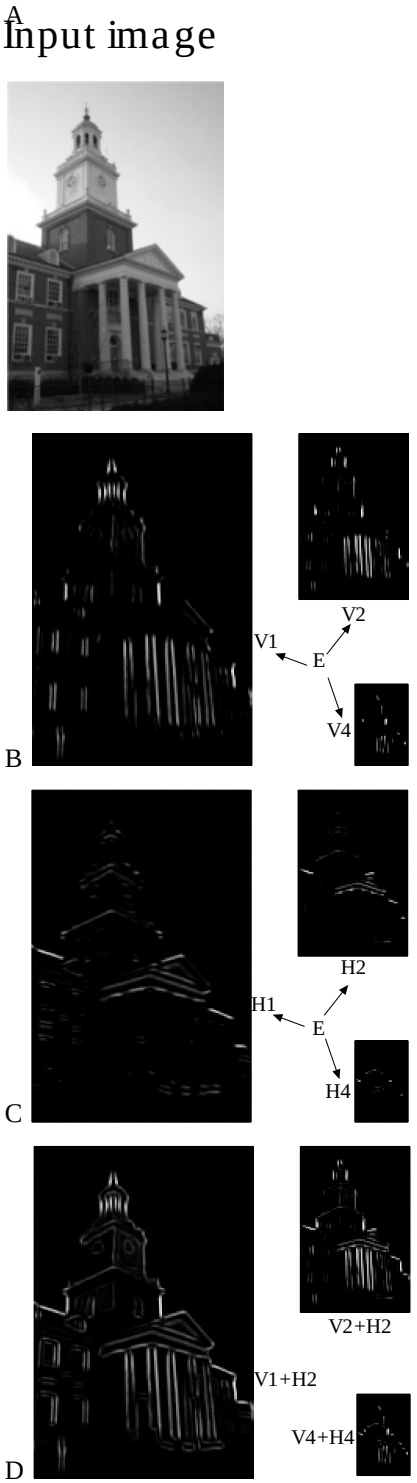

Fig. 3. Responses of some neuronal populations in the simulated primary visual cortex. **A,** Input image. **B,** Activity of complex cells with horizontal preferred orientation. **C,** Same, but for complex cells with vertical preferred orientation. **D** Sum of horizontal and vertical complex cell activities.

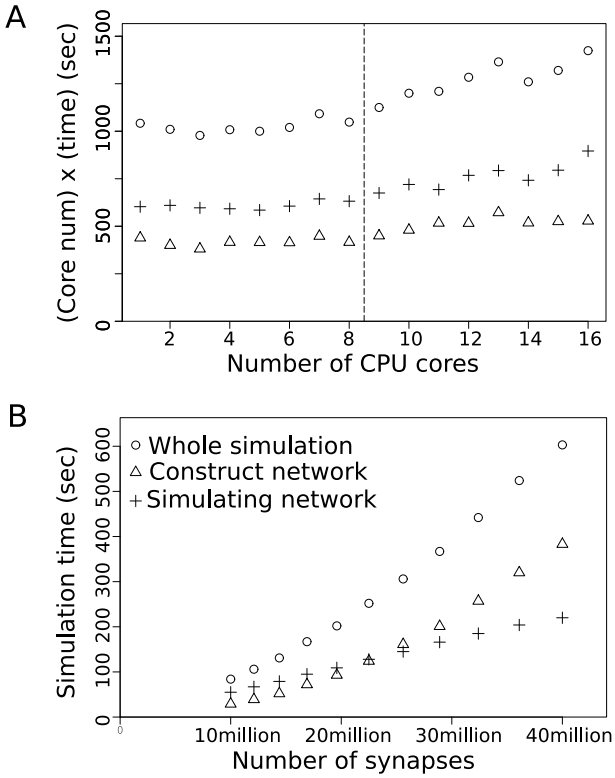

Fig. 4. Scaling performance of the simulator. Circle symbol indicates the whole simulation time, triangle symbol indicates the network construction time and the cross indicates the network simulation time. Panel A, Keeping the size of the network same ($2.88e4$ neurons, $1.44e7$ synapses and average firing $10Hz$) and increasing the number of cores of the CPU (each cpu core corresponds to one NetHost) , the simulation time weighted by the core numbers are plotted. The dotted vertical line indicates the number of computers used in the simulation. At the right of the dotted line, the number of computers used for simulation are fixed at 8. Panel B, keeping the number of cores (8 cores) and the firing rate of the network the same, increasing the size of the network, the simulation times are plotted as a function of simulating network size.

fact that these numbers are prime which leads to uneven load distribution on the NetHosts.

In a second experiment, we kept the number of cores constant at 8 and, again adjusting the mean firing rate of the network to about $10Hz$, we increased the size of the network. As shown in Figure 4 B, simulation time scales linearly with the number of synapses in the network. However, the network construction time scales super-linearly. A likely reason is that the memory needs of the simulation approach the available resources and becomes the performance-limiting factor.

III. Discussion

Like other scientific disciplines, systems neuroscience and, in particular, neurophysiology started out in an exploratory phase when little was known, and then matured to the current more systematic approach in which specific hypotheses can be formulated and their predictions tested in experiments. The complexity of the systems studied requires that these hypotheses are of substantial complexity, too, and generating quantitative predictions requires the development and solution of quantitative models. Implementation of such models, once a sub-speciality of computational neuroscience, is thus becoming an important part of mainstream systems neuroscience.

The neural simulation tool presented in this report addresses several concerns with existing simulation environments. It combines a highly efficient implementation of a single-neuron model with a state-of-the-art event-based neural communication protocol. The protocol allows fully parallel computation and is designed to minimize communication events between machines. Furthermore, it makes near-optimal use of either already existing or of readily available, low-cost computational resources by virtue of the fact that it is implemented in a computer language (JAVA) that is specifically designed to be available on a large number of platforms and potentially heterogeneous networks. While other simulators written in programming languages like C or C++ require compilation and linking of source codes and libraries, JAVA code is run on any supported architecture "out of the box." We note that, although we use the simulator here for a simulation of the early visual system, it can be employed for a large class of problems, *e.g.* to study dynamical properties of simplified, abstract neuronal nets [9]. These objectives are achieved without the end users having a need to learn proprietary scripts or similar constructs or having to compile any source code; the definition of the network and the simulation is entirely contained in extensible mark-up language (XML) files which are organized following the experimental design logic of typical neurophysiological experiments.

Acknowledgment

We thank Eric Carlson for allowing us to use Figure 3A. Work supported by Office of Naval Research grant N000141010278.

References

[1] *An efficient, parallel event driven simulator of realistic neuronal networks*, vol. Abstract 798.1. Washington DE: Society for Neuroscience, 2008.

[2] S. Mihalas and E. Niebur, "A generalized linear integrate-and-fire neural model produces diverse spiking behavior," *Neural Computation*, vol. 21, no. 3, pp. 704–18, March 2009.

[3] B. Hille, *Ionic channels of excitable membranes*. Sunderland, MA: Sinauer, 1992.

[4] A. Morrison, C. Mehring, T. Geisel, A. D. Aertsen, and M. Diesmann, "Advancing the boundaries of high-connectivity network simulation with distributed computing." *Neural Comput*, vol. 17, no. 8, pp. 1776–1801, Aug 2005. [Online]. Available: http://dx.doi.org/10.1162/0899766054026648

[5] D. Hubel and T. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *J. Physiol.*, vol. 148, pp. 574–591, 1959.

[6] D. Ferster, "Spatially opponent excitation and inhibition in simple cells of the cat visual cortex," *J. Neurosci.*, vol. 8, pp. 1172–1180, 1988.

[7] T. Hansen and H. Neumann, "A simple cell model with dominating opponent inhibition for robust image processing." *Neural Netw*, vol. 17, no. 5-6, pp. 647–662, 2004. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2004.04.002

[8] E. Niebur and F. Wörgötter, "Circular inhibition: a new concept in long-range interactions in the mammalian visual cortex," in *Proc. of the International Joint Conference on Neural Networks-San Diego*. Piscataway, NJ: IEEE, 1990, pp. II–367 – II–372.

[9] D. Millman, S. Mihalas, A. Kirkwood, and E. Niebur, "Self-organized criticality occurs in non-conservative neuronal networks during/up/'states," *Nature Physics*, vol. 6, no. 10, pp. 801–805, 2010.