Real-Time FPGA Simulation of Surrogate Models of Large Spiking Networks

Murphy Berzish, Chris Eliasmith, Bryan Tripp

Centre for Theoretical Neuroscience, University of Waterloo,
Waterloo, Canada

Abstract. Models of neural systems often use idealized inputs and outputs, but there is also much to learn by forcing a neural model to interact with a complex simulated or physical environment. Unfortunately, sophisticated interactions require models of large neural systems, which are difficult to run in real time. We have prototyped a system that can simulate efficient surrogate models of a wide range of neural circuits in real time, with a field programmable gate array (FPGA). The scale of the simulations is increased by avoiding simulation of individual neurons, and instead simulating approximations of the collective activity of groups of neurons. The system can approximate roughly a million spiking neurons in a wide range of configurations.

Keywords: FPGA; Neural Engineering Framework; neuromorphic engineering

1 Introduction

Large-scale neural models have recently become central to several large research investments. For instance, the Human Brain Project (HBP) has a central goal of developing a human-scale neural simulation in the next 10 years. Similarly, the Brain Initiative in the US is making a heavy investment in both computational and experimental neuroscience. Both projects have identified massive increases in computational performance as a critical for achieving their goals.

One approach to large-scale neural simulations is to employ standard supercomputers. Another is to build specialized "neuromorphic" hardware [12, 3]. Compared to supercomputer simulations, neuromorphic simulations generally have less biological detail, but are faster and more power-efficient. However, neither of these solutions are currently available to the majority of researchers.

We recently showed that the activity of a neural population model can often be closely approximated by a much simpler surrogate model [14]. The surrogate model consists of feedback dynamics in a space of latent variables that account for correlated population activity, and a model of spike-related fluctuations in synaptic current. Here, we employ this approach to roughly approximate large neural systems with modest hardware.

We present a prototype implementation of this approach for field-programmable gate arrays (FPGAs). We simulate compressed surrogate models of roughly a

million neurons with arbitrary connectivity, in real time, on a single off-the-shelf chip. We describe general considerations for implementing these models on FPGAs, key design elements of the hardware prototype, and preliminary simulation results from the hardware.

There have been other attempts to build neuromorphic systems on FPGAs [2, 11, 15]. The present approach is distinct in simulating simplified models of groups of neurons, rather than individual neurons, increasing scale at the expense of fidelity. Large scale simulation is important because even the simplest behaviours in mammals involve many millions of neurons.

2 Methods

Our surrogate modelling approach builds on the Neural Engineering Framework [5]. We briefly describe this framework below, and then describe the surrogate modelling approach, with particular considerations for FPGA hardware.

2.1 The Neural Engineering Framework

The Neural Engineering Framework (NEF) is method for constructing biologically realistic neural models [5]. The NEF is a general-purpose approach to implementing high-level algorithms using spiking neurons [6]. Importantly, the high-level description is expressed in terms of vectors and functions on those vectors (including differential equations). Several overviews of the NEF are available [8]. Here we outline the NEF's three main principles.

Principle 1 - Representation Groups of neurons are taken to represent vectors, and connections between groups of neurons compute functions on those vectors. The first NEF principle shows how the activity of a group of neurons can be said to represent a vector. The NEF identifies the preferred stimulus for a neuron with a "preferred direction vector" associated with each neuron [9]. That vector determines the neuron's tuning curve, which can be written for any neuron i as:

$$\delta_i(\mathbf{x}) = G_i[\alpha_i \mathbf{e}_i \mathbf{x} + J_i^{bias}] \tag{1}$$

where δ_i is the spiking output of the neuron, G_i is the neuron model, α_i is a randomly chosen gain term, $\mathbf{x}$ is the input space driving the neuron, $\mathbf{e}_i$ is the preferred direction vector, and J_i^{bias} is a randomly chosen fixed background current.

Given an "encoding" of this type, we can define a *decoding* operation, to characterize the information processing characteristics of the neural population. A biologically plausible, continuous, and time-varying measure of the neuron's response is generated by the reception of a spike at a synapse, which can be written:

$$a_i(\mathbf{x}) = \sum_j h_i(t) * \delta_i(t - t_j(\mathbf{x}))$$

where $h_i(t)$ is the synaptic response (e.g., a decaying exponential with a time constant, τ_{PSC}, whose temporal properties are determined by the neurotransmitter type at the synapse), '$*$' is the convolution operator, and $\delta_i(t - t_j(\mathbf{x}))$ is the spike train produced by neuron i, with spike times indexed by j.

Having defined this continuous variable, we can specify a decoding operation for estimating the input $\mathbf{x}$:

$$\hat{\mathbf{x}} = \sum_{i}^{N} a_i(\mathbf{x})\mathbf{d}_i \tag{2}$$

where N is the number of neurons in the group, $\mathbf{d}_i$ are the linear decoders, and $\hat{\mathbf{x}}$ is the estimate of the original $\mathbf{x}$ value that produced the neural activity (1). We can use least-squares optimization to find these decoders:

$$\arg\min_{\mathbf{d}_i} \int [\mathbf{x} - \sum_{i}^{N} a_i(\mathbf{x})\mathbf{d}_i]^2 d\mathbf{x} \tag{3}$$

where the integral is over all $\mathbf{x}$ values.

Note that employing linear decoding allows us to directly compute connection weights. For example, if a connection between neural groups is meant to compute the identity function $\mathbf{y} = \mathbf{x}$, the connections between individual neurons are given by

$$\omega_{ji} = \alpha_j \mathbf{e}_j^T \mathbf{d}_i \tag{4}$$

where i indexes the neurons in group A and j indexes the neurons in B, and T indicates the transpose.

Principle 2 - Transformation Connections between groups of neurons can also approximate arbitrary functions: $\mathbf{y} = f(\mathbf{x})$. In the NEF this is accomplished by finding decoders $\mathbf{d}_i^f$ that produce the approximation $\hat{f}(\mathbf{x}) \approx f(\mathbf{x})$. This requires the same optimization as in (3), substituting $\mathbf{d}_i^f$ for $\mathbf{d}_i$.

The connection weights can then be computed using (4). In general, the neural connection weights needed to approximate the function $\mathbf{y} = \mathbf{L}f(\mathbf{x})$ are:

$$\omega_{ji} = \alpha_j \mathbf{e}_j^T \mathbf{L}\mathbf{d}_i^f \tag{5}$$

Principle 3 - Dynamics The first two principles can be used to build neural implementations of any feedforward function of $\mathbf{x}$. The NEF also provides a method for computing functions of the form

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{dt}} = f(\mathbf{x}, \mathbf{u}) \tag{6}$$

where $\mathbf{u}$ is the input from some other population.

The NEF exploits the fact that the post-synaptic current induced by a spike is well-approximated by $h(t) = u(t)e^{-t/\tau}$, where $u(t)$ is the step function and τ is the time constant of the neurotransmitter used. This time constant varies

throughout the brain, e.g., from 2-5 ms (AMPA; [10]) up to ∼100ms (NMDA; [13]). Explicitly identifying this aspect of the neural response demonstrates that any connection actually computes $\mathbf{y}(t) = f(\mathbf{x}(t)) * h(t)$.

Given a neural population representing $\mathbf{x}$, an input $\mathbf{u}(t)$, and a connection from $\mathbf{x}$ back to itself computing $g(\mathbf{x}(t))$, we can show

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{dt}} = \frac{g(\mathbf{x}(t)) - \mathbf{x}(t)}{\tau} + \frac{\mathbf{u}(t)}{\tau}. \tag{7}$$

Thus, if we desire the dynamics

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{dt}} = f(\mathbf{x}(t)) + \mathbf{u}(t), \tag{8}$$

we introduce a feedback connection that uses the previous two NEF principles to find connection weights that compute $g(\mathbf{x}(t)) = \tau f(\mathbf{x}) + \mathbf{x}$ and we scale the input $\mathbf{u}(t)$ by τ.

Our exploitation of the inherent first-order low-pass filter found in synaptic connections allows for the implementation of a very wide variety of systems, including linear and nonlinear oscillators, integrators, and arbitrary attractor networks [4]. In short, the NEF approach allows for the construction of neural models that correspond to a very large family of functions, including those typically employed by modern control theory and dynamic systems theory. The NEF was recently used to build the large-scale Spaun model [7].

2.2 Surrogate Population Models

As described in the previous section, the NEF allows construction of neural models that optimally approximate idealized dynamics (Eq. 8), within constraints that are grounded in physiology (these include sensitivity to spike-related fluctuations, saturation of spike rates, etc.). One implication of this approach is that the idealized dynamics serve as a rough approximation of the neural model. NEF simulators allow direct simulation of the idealized dynamics as an aid to debugging. We call such simulations "direct mode" simulations, as opposed to "default mode" spiking simulations.

Beginning with the idealized dynamic model (Eq. 8), we showed recently [14] that the dynamics of the full spiking model can often be largely recovered via efficient approximations of the difference $\hat{\mathbf{f}}(\mathbf{x}) - \mathbf{f}(\mathbf{x})$. This difference consists of static distortion components that can be approximated by interpolation, and spike-related fluctuating components that can be approximated with an autoregressive moving average (ARMA). We refer to this new type of simulation as "population mode" simulation, because it models population-level dynamics. In practice, we actually model $\hat{\mathbf{f}}(\mathbf{x})$ directly, rather than $\hat{\mathbf{f}}(\mathbf{x}) - \mathbf{f}(\mathbf{x})$.

3 Surrogate Models on FPGAs

The ARMA components of the surrogate models are suitable for FPGA simulation. However, interpolation of the static distortions is a performance bottleneck.

The bottleneck arises due to a mismatch between processing capacity and on-chip memory. The hardware can simulate a population much faster than real time, so we multiplex simulation of many populations in the same hardware component (a "population unit"). However, on-chip memory is insufficient to store the model parameters of all these populations. For this reason, the population parameters must be stored in off-chip RAM and loaded for each population, each simulation step. We therefore sought to approximate $\hat{\mathbf{f}}(\mathbf{x})$ using as few unique parameters per population as possible.

The functions that a population can approximate well belong to the space of the first few principal components (PCs) of the population's tuning curves $a_i(\mathbf{x})$ [6]. For this reason, static population output can be approximated efficiently by linear regression with these first few principal components. This requires only as many coefficients as there are important principal components. The advantage (vs. direct interpolation of outputs) is greatest with multidimensional populations. Accuracy depends on regularization of the decoders (eq. 2), in that stronger regularization reduces contributions from minor PCs.

Parameter distributions are often shared by several populations. Moreover, we find empirically (Figure 1) that the principal components of populations with quite diverse parameter distributions are often quite similar. Our approach is therefore to group populations by clustering their principal components, and use the cluster-averaged principal components as basis functions for regression of each required $\hat{\mathbf{f}}(\mathbf{x})$. With this approach, it is necessary to load only a small number of unique regression parameters per population.

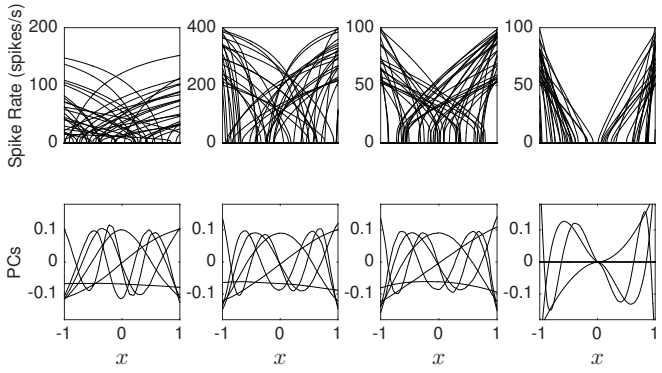

Fig. 1. Tuning curves of four different example populations and their principal components. Each column corresponds to a population of spiking LIF neurons with different parameters. The top panels are 50 tuning curves drawn from populations of 1000 neurons. The bottom panels are the first six principal components of each population. Despite differences between the three leftmost populations, their principal components are nearly the same. As a counter-example, the population on the right has very different principal components (due to a very distinct intercept distribution) so it should go in a different cluster.

3.1 Hardware Prototype

A block diagram of a prototype hardware platform is shown in Figure 2. The design connects fourteen population units, 7 for 1D populations, and 7 for 2D populations (D is the dimension of $\mathbf{x}$; eq. 1). Each population unit consists of a fixed bank of principal components, an encoder unit, and a decoder unit. The encoder and decoder units perform the encoding and decoding operations described in Section 2.1. Both are implemented as circular buffers containing parameters for each population. The encoder unit is also responsible for saving the states of low-pass filters that model synaptic dynamics. The principal components are implemented as interpolating lookup tables with a 12-bit fixed-point representation, and are the same across every population on the same population unit. Both 1D and 2D principal component tables have been implemented with linear and bilinear interpolation. An additional decoder is used to scale pseudo-random Gaussian noise. The interconnect is designed around an all-to-all shared bus architecture that allows any encoder unit to read decoded values from any population unit, or from any external input source. At the end of each timestep, outputs from each population are written to RAM buffers and can be read by encoders in the following timestep. The population units are otherwise independent from each other and use internal RAM buffers to save and load the state of each population as it is time-multiplexed on and off the hardware.

As the simulation time for one population is on the microsecond scale, we time-multiplex 1024 populations on each population unit and run a maximum of 14x1024 population models in real-time simulation, at a rate of 1000 updates per second. In software simulations, 1D populations often have about 10 to 100 neurons, and 2D populations often have 100 to 1000 neurons, so we consider this system to model an approximation of roughly 0.8 to 8 million neurons.

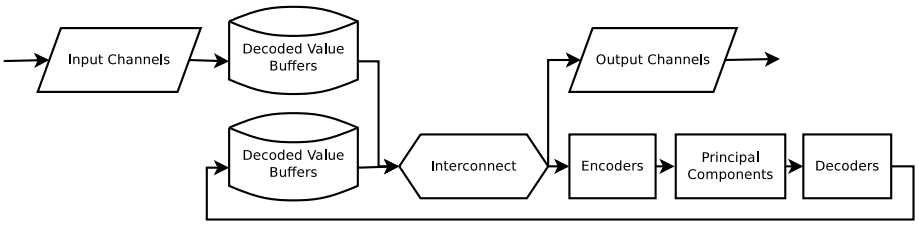

Fig. 2. Block diagram of the hardware design. Of note, concurrent access to the interconnect is coordinated via distinct fixed delays for each encoder unit.

We extended the Nengo neural simulator [1] with a custom backend that translates neural models to FPGA configuration data. The backend determines which PCs can be clustered onto the same population units with the smallest absolute error. It transmits the model parameters to the hardware, and controls the simulation in real time, transferring external input values to the board and reading population outputs.

4 Results

We ran simulations on a Xilinx VC707 development board. The design used a large fraction of the device resources, including 44% of its 37080 Kb of block RAM and 61% of its 2800 customizable DSP operator slices. The design was estimated to use at most 5.675 W of power.

As a demonstration of the hardware, we simulated a neural model consisting of 4096 recurrent networks that approximate van der Pol oscillators. Output from an example population simulated in software (with spiking neurons) is shown in Figure 3 (left). Output from the same population simulated in our hardware is also shown in Figure 3 (centre). The oscillator requires a brief non-zero "startup" initial input in order to begin oscillating. Once this input disappears, at 0.1 s, both dimensions of the decoded output can be seen to oscillate.

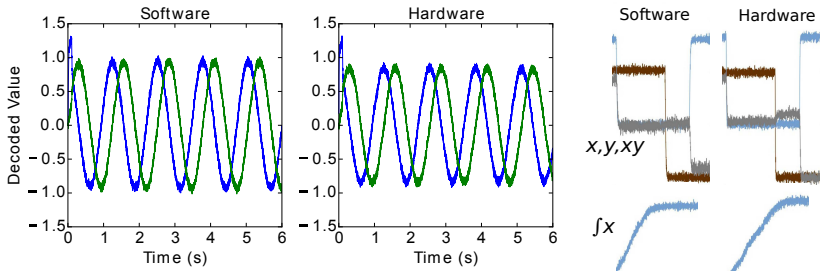

Fig. 3. The left plot shows a software simulation of an NEF spiking recurrent network that implements a van der Pol nonlinear oscillator. The two plotted values correspond to the two state variables of the system, decoded from filtered spike trains. The centre plot is a hardware simulation of the same recurrent network. The plotted values are decoded from principal components of the population, with a model of spike noise. The recurrent connection in hardware is also a function decoded from principal components, with added noise. On the right are two additional examples: a product decoded from a 2D population, and an integrator recurrent network.

5 Discussion

We have described the design and prototype of a new approach to large-scale, real-time neural model simulation using off-the-shelf hardware. This approach allows us to approximately simulate NEF networks of about a million neurons in real time on a single chip. This approach appears promising for three kinds of applications. First, it allows power-efficient real-time simulation of fairly large neural systems. Second, it is well-suited for embedding sophisticated network models in robots. Third, interconnection of multiple FPGAs would potentially allow very large real-time simulations, although such a system would scale sub-linearly, depending on the communication required between devices.

Although population mode does not simulate individual neurons, approximations of network dynamics are often quite close [14]. Some of the missing details are undoubtedly important. However scale is also important. As well, our approach could be used as part of a multi-scale approach, with a small detailed network embedded in an approximate simulation of a larger system.

A more specific limitation of our prototype is that we have only implemented principal component lookup tables of one and two dimensions. The system can simulate populations that encode higher-dimensional vectors, but only if the decoded functions are either linear, or nonlinear only in groups of one or two dimensions. More generally, we emphasize that our implementation is a proof of concept, and that further development and validation is needed.

Acknowledgments. Funded by Discovery Grants (261453 and 296878) and a USRA from NSERC, Canada. The VC707 was donated by RTDS Technologies.

References

1. Trevor Bekolay and et al. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7(January):48, jan 2014.
2. A Cassidy and et al. Design of a one million neuron single fpga neuromorphic system for real-time multimodal scene analysis. In *Information Sciences and Systems (CISS), 2011 45th Annual Conference on*, pages 1–6. IEEE, 2011.
3. S. Choudhary and et al. Silicon neurons that compute. In *International Conference on Artificial Neural Networks*, pages 121–28, 2012.
4. C Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural computation*, 17(6):1276–1314, 2005.
5. C Eliasmith and C H Anderson. Developing and appl a toolkit from a general neurocomputational framework. *Neurocomputing*, 26:1013–1018, 1999.
6. C Eliasmith and C H Anderson. *Neural engineering: Computation, representation and dynamics in neurobiological systems*. MIT Press, Cambridge, MA, 2003.
7. C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, C. Tang, and D. Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, nov 2012.
8. Chris Eliasmith and et al. A large-scale model of the functioning brain. *Science*, 338:1202–1205, 2012.
9. A Georgopoulos and et al. Mental rotation of the neuronal population vector. *Science*, 243:234–236, 1989.
10. P Jonas and et al. Quantal components of unitary EPSCs at the mossy fibre synapse on CA3 pyramidal cells of rat hippo. *J Physio*, 472(1):615–663, 1993.
11. Jing Li and et al. An FPGA-based silicon neuronal network with selectable excitability silicon neurons. *Frontiers in Neuroscience*, 6(183), 2012.
12. P Merolla and et al. Artificial brains: A million spiking-neuron IC with a scalable communication network and interface. *Science*, 345(6197):668–73, 2014.
13. P Sah and et al. Properties of excitatory postsynaptic currents recorded in vitro from rat hippocampal interneurones. *J Physio*, 430(1):605–616, 1990.
14. Bryan P. Tripp. Surrogate Population Models for Large-Scale Neural Simulations. *Neural Computation*, 27(6):1186–1222, 2015.
15. R. Wang and et al. An FPGA implementation of a polychronous spiking neural network with delay adaptation. *Frontiers in Neuroscience*, 7(14), 2013.