Building Block of a Programmable Neuromorphic Substrate: A Digital Neurosynaptic Core

John V. Arthur*‡, Paul A. Merolla*, Filipp Akopyan*, Rodrigo Alvarez*,
Andrew Cassidy*, Shyamal Chandra*, Steve Esser*, Nabil Imam†,
William Risk*, Daniel Rubin*, Rajit Manohar†, and Dharmendra Modha*
*IBM Almaden Research Center, Almaden, CA, USA
†Cornell University, Ithaca, NY, USA
‡Email: arthurjo@us.ibm.com

***Abstract*—The grand challenge of neuromorphic computation is to develop a flexible brain-like architecture capable of a wide array of real-time applications, while striving towards the ultra-low power consumption and compact size of biological neural systems. To this end, we fabricated a key building block of a modular neuromorphic architecture, a *neurosynaptic core*. Our implementation consists of 256 integrate-and-fire neurons and a 1,024×256 SRAM crossbar memory for synapses that fits in 4.2mm^2 using a 45nm SOI process and consumes just 45pJ per spike. The core is fully configurable in terms of neuron parameters, axon types, and synapse states and its fully digital implementation achieves one-to-one correspondence with software simulation models. One-to-one correspondence allows us to introduce an *abstract neural programming model* for our chip, a contract guaranteeing that any application developed in software functions identically in hardware. This contract allows us to rapidly test and map applications from control, machine vision, and classification. To demonstrate, we present four test cases (i) a robot driving in a virtual environment, (ii) the classic game of pong, (iii) visual digit recognition and (iv) an autoassociative memory.**

I. Introduction

Custom neuromorphic chips have steadily marched towards the goal of matching the incredible power efficiency and scale of biological neural systems [1], [2], [3]. Despite advances on the hardware front, there has been little progress to using these chips for real-world applications. One of the main obstacles holding back the wide spread utility of low-power neuromorphic chips is the lack of a consistent software–hardware neural programming model, where neuron parameters and connections can be learned off-line to perform a task in software with a guarantee that the same task will run on power-efficient hardware.

In this paper, we present a digital *neurosynaptic core* that by design has a functional specification that can be simulated in any software environment. Our core incorporates central elements from neuroscience, including 256 leaky integrate-and-fire neurons, 1024 axons, and 256×1024 synapses. During operation, the core runs a discrete event simulation, and for each time step the activity in terms of spikes per neuron, neuron state, etc. is identical to the equivalent property of a corresponding software simulation. This one-to-one correspondence allows algorithm development to proceed without hardware in hand, enabling parallel algorithm and hardware progress. In contrast, approaches that use dense analog circuits to model neural components do not maintain one-to-one correspondence between hardware and software, due to device mismatch and statistical fluctuations (e.g., changes in ambient temperature) [4].

Given that our digital hardware is equivalent to a software model, one can ask: why not take the software model itself and translate it into hardware directly using either conventional or commodity computational substrates? Running the software specification on conventional supercomputers is out of the question due to high power consumption [5]. Commodity chips, which include DSP [6], GPU [7], and FPGA [8] solutions, also lead to relatively high power that limits scaling. Specifically, these solutions require high-bandwidth to communicate spikes from separately located processing and memory, and to meet real-time performance clock speeds are typically run in the gigahertz range. In contrast, we implement fanout by integrating crossbar memory with neurons to keep data movement local, and use an asynchronous event-driven design where each circuit evaluates in parallel and without any clock, dissipating power only when performing useful computation. These architectural choices result in a core that implements a family of parameterized spatiotemporal functions in a naturally power constrained manner, where power grows linearly with activity. In practice, our prototype chip fabricated in a 45nm SOI process consumes just 45pJ of active power per spike [9].

To demonstrate our approach, we implement four neural applications that were first developed in software and subsequently mapped to the neurosynaptic core.

II. Architecture

The basic building blocks of our neurosynaptic core are axons, synapses, and neurons (Fig. 1(a)). Within the core, information from the axons is modulated by the synapses as it flows to the neurons. The structure of the core consists of $\mathsf{K} = 1024$ axons that connect via $\mathsf{K}\times\mathsf{N}$ binary-valued synapses to $\mathsf{N} = 256$ neurons. We denote the connection between axon j and neuron i as W_{ji}.

The dynamics of the core are updated at a discrete time step, Δt. Let t denote the index of the current time step, where t has units of milliseconds (i.e., $\Delta t = 1$ms). Each axon corresponds to a neuron's output that could either reside on the core or somewhere else in a larger system with many cores. At each time step t, each axon j is presented with an activity bit $A_j(t)$ that represents whether its corresponding neuron fired in the previous time step ($A_j(t) = 1$) or not ($A_j(t) = 0$). Axon j is statically designated as one of three types G_j, which assumes a value of 0, 1, or 2; these types are used to differentiate connections (e.g., excitatory or inhibitory) with different efficacies. Correspondingly, neuron i weighs synaptic input from axon j of type $G_j \in \{0, 1, 2\}$ as $S_i^{G_j}$. Thus, neuron i receives the following input from axon j:

$$A_j(t) \times W_{ji} \times S_i^{G_j}. \tag{1}$$

For our neurons, we use a single compartment, leaky integrate-and-fire model parameterized by its leak L, threshold θ, and three synaptic values S^0, S^1, S^2 that correspond to the different axon types. The membrane potential $V(t)$ of neuron i is updated in each time step as

$$V_i(t+1) = V_i(t) + L_i + \sum_{j=1}^{\mathsf{K}} \left[A_j(t) \times W_{ji} \times S_i^{G_j} \right]. \tag{2}$$

When $V_i(t)$ exceeds its threshold θ_i, the neuron produces a spike and its potential is reset to 0. Negative membrane potentials are also clipped back to 0 at the end of each time step.

III. Implementation

The block-level implementation of our neurosynaptic core consists of an input decoder with 1024 axon circuits, a 1024×256 SRAM crossbar, 256 neurons, and an output encoder (Fig. 1(b)). Communication at the input and output of the core uses an *address-event representation* (AER), which encodes binary activity, such as $A(t)$, by sending the locations of active elements via a multiplexed channel [10]. In order to minimize active power consumption, we perform neural updates in an event-driven manner. Specifically, we followed an asynchronous design style, where every block performs request-acknowledge handshakes to perform quasi-delay insensitive communication (without a clock) [11].

The detailed operation of the core in a discrete time step commences in two phases: the first phase implements the axon-driven component, and the second phase implements a time step synchronization. In the first phase, address-events are sent to the core one at a time, and these events are sequentially decoded to the appropriate axon block (e.g., axon 3 from Fig. 1(b)). On receiving an event, the axon activates the SRAM's row, which reads out all of the axon's connections as well as its type. All the connections that exist (all the `1`'s) are then sent to their respective neurons, which perform the appropriate membrane potential updates; the `0`'s are ignored. After the completion of all the neuron updates, the axon block de-asserts its read, and is ready to process the next incoming address event; this sequence repeats until all address events for the current time step are serviced. In the second phase, which occurs once every millisecond, a synchronization event (Sync) is sent to all the neurons. On receiving this synchronization, each neuron checks to see if its membrane potential is above threshold, and if so, it produces a spike and resets the membrane potential to 0; these spikes are encoded and sent off as address events in a sequential fashion. After checking for a spike, the leak is applied.

The purpose of breaking neural updates into two phases is to ensure that the hardware and software are always in lock step at the end of each time step. Specifically, the order in which address events arrive to the core or exit the core can vary from chip to chip due to resource arbitration—especially when events are sent through a non-deterministic routing network. To remain one-to-one, the different orderings must not influence the spiking dynamics. We achieve one-to-one equivalence by first accounting for all the axon inputs, and then checking for spikes after these inputs have been processed. This also gives us a precise bound for operating in real time: all address events must be accounted for before the synchronization event, which we trigger once per millisecond.

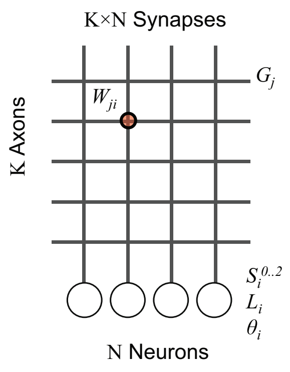
Our core was fabricated in a 45nm SOI process, and has 3.8 million transistors in 4.2mm^2 of silicon (Fig. 1(c), *left*), and consumes just 45pJ per spike at $\mathsf{V_{dd}} = 0.85$V. To test our chip, we built a custom printed circuit board that interfaces with a PC through a USB link (Fig. 1(c), *right*). Through this link, we can interface our chip to virtual and real environments via address event communication, as well as configure neuron–synapse parameters via a shift-register scanner (not shown).

IV. Neural Programming Model

A bedrock of modern computing is the idea that a software program will execute repeatably and deterministically on any compatible hardware platform. This basic contract between software and hardware developers, referred to as an *abstract programming model*, has paved the way for specialization in both domains, allowing software engineers to focus on algorithmic development while hardware engineers tackle performance. We apply this principle and introduce a neural programming model for our neurosynaptic core.

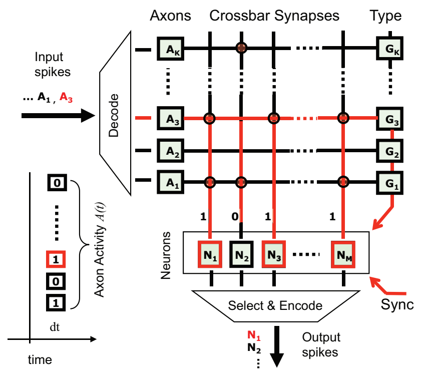
The closest equivalent to a program for our core is the state of neural parameters, which determines the core's dynamics in response to inputs given its initial conditions. Specifically, each core has $2^{256\times1024}$ synapse configurations (W), 3^{1024} axon type configurations (G), and $2^{9\times5\times256}$ neuron state configurations (S^0, S^1, S^2, L, θ). The challenge is to find a configuration in this enormous space that results in a desired function.

Our process for programming the neurosynaptic core is as follows: Given a particular task, such as a recognition or sensory-motor control problem, the task must first be framed in the context of a spiking neural network. The first step is to assign each neuron to a particular function; where some possible functions include feature detection, memory, sensory input, and motor output. Next, connections and parameters for each of the different neuron types need to be determined. In the simplest case, these parameters can be hand-specified when

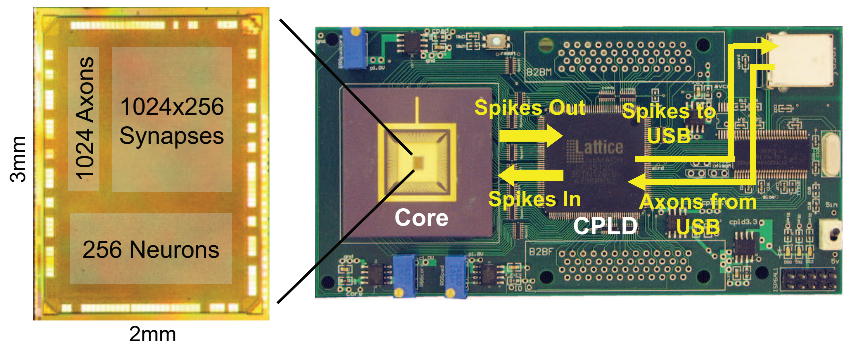


Name	Description	Range
W_{ji}	Connection between axon j and neuron i	0,1
G_j	Axon type	0,1, 2
$S_i^{0..2}$	Synapse values	-256 to 255
L_i	Leak	-256 to 255
θ_i	Threshold	1 to 256

(a)



(b)



(c)

Fig. 1. (a) The core consists of axons, represented as rows; dendrites, represented as columns; synapses, represented as row–column junctions; and neurons that receive inputs from dendrites. The parameters that describe the core have integer ranges as indicated. (b) Internal blocks of the core include axons (A), crossbar synapses implemented with SRAM, axon types (G), and neurons (N). An incoming address event activates axon 3 (A_3), which reads out that axon's connections, and results in updates for neurons N_1, N_3 and N_M. (c) *left* Neurosynaptic die measure 2mm × 3mm including the I/O pads. *right* Test board that interfaces with the chip via a USB 2.0 link. Spike events are sent to a PC for data collection, and are also routed back to the chip to implement recurrent connectivity.

the function is straightforward. For more complex functions, the parameters need to be learned offline, typically through an optimization procedure on a representative dataset from a real or virtual environment. Finally, these parameters are transformed into a hardware-compatible format, where they can be downloaded to the hardware platform and tested in real-time.

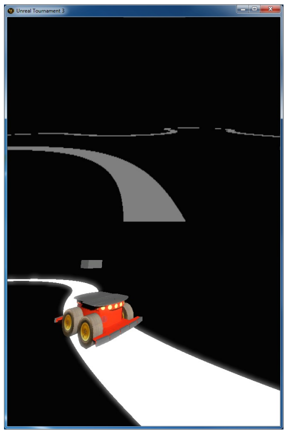
To demonstrate our neural programming framework, we describe four example applications that have been mapped to our hardware in the following subsections. Results were collected from our custom chip via the PC-USB interface, while using the host PC for graphics display, user interface, and supporting software as necessary. We also verified for a number of test cases that each application operated exactly as predicted by the software model. Detailed information on the specific parameter values used for each example can be found in the appendix.

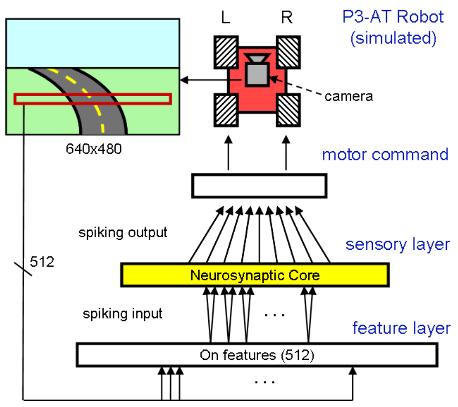
A. Autonomous Virtual Robot Driver

For our first application, we programmed the neurosynaptic core to steer a simulated robot around a virtual racetrack. The goal of the robot driver application is to keep the robot on the racetrack using an autonomous, closed-loop control system. The virtual robot is a physics based emulation of the real-life MobileRobots Pioneer 3-AT (P3AT), a four wheel drive robotic platform with a vision sensor. We defined a clover-shaped racetrack in the Unreal Tournament 3 virtual environment. The autonomous control system receives visual input from its camera, and controls the robot by issuing differential wheel speed commands to the left and right wheels. Fig. 2(a) shows screenshots of the view seen by the robot camera and the robot on the racetrack in the virtual environment.

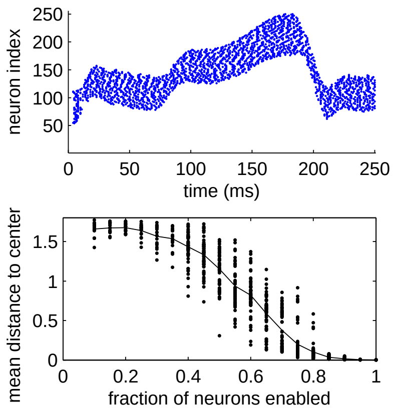
Controlling the P3AT robot in the virtual environment requires two key components: vision to see the racetrack and a control system to direct the robot's actions. In the virtual environment, we simplified the vision task of detecting the road by performing preprocessing on the host computer, and focused on implementing the control system on the hardware.

We program the neurosynaptic core to implement a negative proportional feedback controller in which difference in power applied to the left and right wheels is proportional to the difference in the P3AT's direction of travel and the direction to the center of the road (in visual space). Therefore, when the P3AT drives down the center of the road both wheels receive


(a) Virtual Environment



(b) System Block Diagram



(c) Results

Fig. 2. (a) Screenshots of the robot-eye view in the virtual environment at top and the simulated P3AT robot on the racetrack at bottom. (b) Autonomous robot control system block diagram. (c) *top* Spike activity of network during driving task. *bottom* Average distance from center of driving track, as a function of the number of neurons used where distance saturates at 2.0 in failed trials.

equal power. The system is a self-contained closed loop, and a block diagram is shown in Fig. 2(b). Vision processing consists of a layer of "On" feature detectors that spike when the road is within their receptive field. Note that this feature layer could be augmented with many other features (e.g., "Off" features, edges, road color statistics, etc.) for a more complex visual processing system, which would be necessary in more realistic environments. The neurons in the sensory layer, which reside in the neurosynaptic core, are driven from these feature layer detectors to essentially form road detectors that represent the position of the road in the visual field. The motor command processing block integrates the spiking output from the sensory layer and produces two values, one for both of the left and right wheels to steer the robot. The power to each wheel is proportional to the number of spikes on each side of the visual field; therefore, if the road is left of center the left wheel receives greater power, causing a turn to the left, centering the position of the road in the visual scene.

We program the neurosynaptic core as follows, mapping the sensory neuron layer to the hardware. There are 512 "On"-features, which are mapped to 1024 axons, half excitatory and half inhibitory. The 256 neurons are then divided up across the visual field, centered at uniform intervals and each taking input from 32 excitatory and 16 inhibitory axons. The road detectors are excitatory in the center ($S_0 = +2$), inhibitory on the edges ($S_1 = -2$), and zero elsewhere, forming a matched filter that fires maximally when the road is centered in the filter's receptive field.

We measured the robustness of our neural controller by varying the fraction of neurons enabled(from 10 to 100%) and tracking the average distance the robot deviated from the center of the track (Fig. 2c). Using 100% of the neurons, the performance is near perfect (low mean-squared error), and degrades smoothly with decreasing percentage of neurons.

B. Pong Player

For our second example, we simulated the classic video game of Pong and implemented an autonomous player that is controlled by the neurosynaptic core. The goal of Pong is to keep a moving ball contained in a 2D box, where one wall of the box is open such that the ball must be hit by a paddle controlled by a player. To make the game more challenging the direction of the ball is randomly perturbed when it bounces off the wall opposite to the paddle. Because the ball moves faster than the paddle, the challenge for the player is to predict the correct paddle position far enough in advance.

To implement an autonomous player, the neural network interacts with the game the same way a normal player would, through visual access to the ball's position and a 1D paddle controller but without access to precise ball position, velocity, and kinematics that virtual players typically have. The playing area is divided into a 16×14 grid corresponding to visual receptive fields of 224 neurons that are direction selective, spiking only when the ball moves towards the virtual player's side of the field (loosely based on the fly visual system [12]). For paddle-control, we assign 14 motor neurons to control a joystick (inertial controller), which moves the paddle's location toward the position of the most active motor neuron.

We implement the player with a spiking neural network using three subsystems: sensory, reset, and motor. The function of the sensory neurons is to store the ball's trajectory as a trace of neural activity (Fig. 3(b-c)). Specifically, sensory neurons become active when the ball moves across the game area, and remain active without external input via excitatory self-connections (autapses). To reset the persistent activity between volleys, the sensory neurons receive strong inhibition from an interneuron when the ball reaches the paddle, and the sensory neurons remain inhibited until the ball reaches the bottom row of the playing area (shutoff by a second inhibitory interneuron).

Finally, to associate ball trajectories with the appropriate

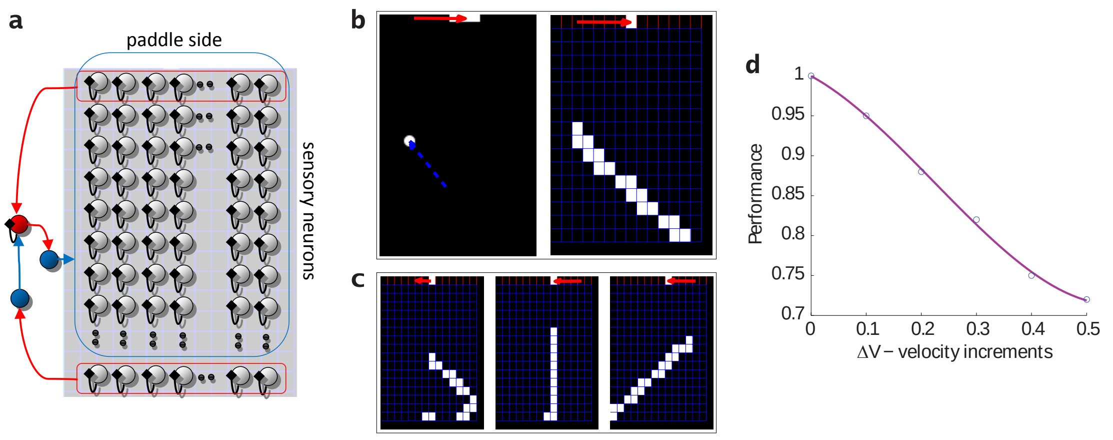


Fig. 3. (a) Schema of the underlying architecture of the network. Excitatory connections are red, inhibitory connections are blue, and autapses are black. Areas of projecting or projected excitatory/inhibitory connections are red/blue circled. (b) *left* Representation of the Pong game area and *right* example traces of active neurons left by the ball trajectory (white squares), and the motor neuron corresponding to the position of the paddle center (upper grid in red). The red arrow indicates the paddle's initial and final position. (c) Three examples of the neural player after training. (d) Performance of the system measured as the percentage of hits as a function of increasing ball velocity. The data points are fit by a cubic line.

motor neuron activations, we used an offline supervised learning algorithm to learn the sensory to motor neuron connections. Specifically, the connections were updated according to a perceptron update algorithm conditioned to maximal stability [13] based on training examples of ball trajectories and desired motor outputs (a teacher signal). Through this training, the network learned to recognize trajectories and their associated endpoints rather than computing the ball's kinematics. To transform the real-valued weights into a hardware-compatible format, the synapse strengths for each motor neuron were truncated to the closest of the means of a trivariate Gaussian fit (since neurons are restricted to three synapse types). With this mapping, the player performance is perfect for slow ball velocities, and degrades as the speed increases (Fig. 3(d)). Qualitatively, the learned strategy resembles a human strategy, since the motor neurons move the paddle in the vicinity of the correct position early in the ball's trajectory (i.e., coarse prediction), and continually refines its prediction as the ball approaches.

C. Visual Digit Recognition

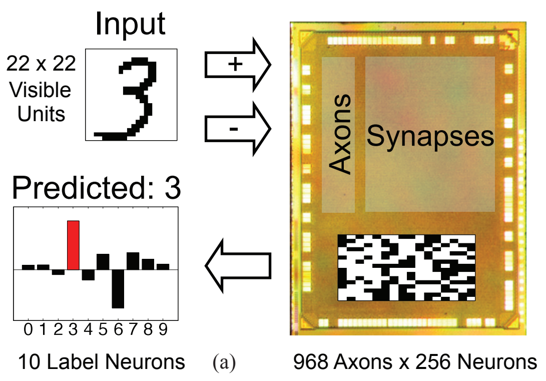
Our third application was to classify handwritten numeric digits from the MNIST dataset [14]. The MNIST dataset contains 70,000 example images of the handwritten digits 0 through 9. Each 22x22 pixel image has been segmented into binary (black or white) pixels. The challenge of this task is to accurately recognize the digits in spite of the significant variation between writing styles and handwritten instances.

We approached this task using a Restricted Boltzmann Machine (RBM), a well-known algorithm for classification and inference tasks. Specifically, we trained a two-layer RBM offline with 484 visible units and 256 hidden units on handwritten digits from the MNIST dataset. Our learning procedure followed directly from [15]; briefly, we use contrastive divergence to learn 484×256 real-valued weights to capture the probability distribution of pixel correlations in the digits (60,000 images). After learning these weights, we trained 10 linear classifiers on the outputs of the hidden units using supervised learning. Finally, we test how well the network classifies digits on out-of-sample data (10,000 images), and achieved 94% accuracy.

To map the RBM onto our neurosynaptic core, we make the following choices: First, we represent the 256 hidden units with our integrate-and-fire neurons. Next, we represent each visible unit using two axons, one for positive (excitatory) connections and the other for negative (inhibitory) connections, accounting for 968 of 1024 axons. Then we cast the 484×256 real-valued weight matrix into two 484×256 binary matrices, one representing the positive connections (taking the highest 15% of the positive weights), and the other representing the negative connections (taking the lowest 15% of the weights). Finally, the synaptic values and thresholds of each neuron were adjusted to normalize the sum total input in the real-valued case with the sum total input of the binary case.

Following the example from [16], we are able to implement the RBM using spiking neurons by imposing a global inhibitory rhythm that clocks network dynamics. In the first phase of the rhythm (no inhibition), hidden units accumulate synaptic inputs driven by the pixels, and spike when they detect a relevant feature; these spikes correspond to binary activity of a conventional (non-spiking) RBM in a single update. In the second phase of the rhythm, the strong inhibition resets all membrane potentials to 0. We sent the outputs of the hidden units to the same linear classifier as before (note that these classifiers are not implemented in the neural hardware).

We achieve 89% accuracy for out-of-sample data (see Fig. 4 for one trial). Our simple mapping from real-valued to binary weights shows that the performance of the RBM degrades gracefully, and suggests that more sophisticated algorithms, such as deep Boltzmann machines, will also perform well in



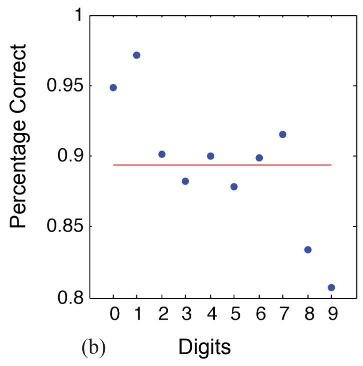


Fig. 4. (a) Pixels that represent visible units drive spike activity on excitatory (+) and inhibitory (-) axons to stimulate a 16×16 grid of neurons on the chip. Here, spikes are indicated as black squares, and encode the digit as a set of features. The spikes are sent to an off-chip linear classifier that predicts 3 as the most likely digit, whereas 6 is the least likely. (b) Measured performance for each digit for out-of-sample data (points), and the average performance (red line) is 89%.

hardware despite low precision synaptic weights.

D. Autoassociation

Our fourth application was storing and recalling patterns in a manner that allows a subset of a pattern to retrieve the whole. The challenge of this autoassociation task is storing patterns implicitly in synaptic weights such that they can be recalled with fidelity. We approach autoassociation in two distinct ways, one with a sparse version of the classic Hopfield Network and the other with a two-layer spiking network (Fig. 5).

The Hopfield Network is a classic neural network, which realizes autoassocative memory in a recurrently connected network by increasing connection strengths between correlated neurons while decreasing strengths between anticorrelated neurons across stored memory patterns. We select a sparse Hopfield implementation, storing m binary patterns of length $n = 256$ each with 8 `1`s (sparsity $\gamma = 8/256$) [17], [18]. Such a sparse implementation is less sensitive to low precision (binary) weights while maintaining a high memory capacity. We trained the network offline, computing real-valued connection strengths from neuron j to neuron i with the Hopfield Rule given by:

$$W_{ji} = \sum_{k=1}^{m} \left(\frac{v_j^k}{\gamma} - 1 \right) \left(\frac{v_i^k}{\gamma} - 1 \right)$$

where v_j^k and v_i^k are the stored states of the jth and ith neurons in pattern k.

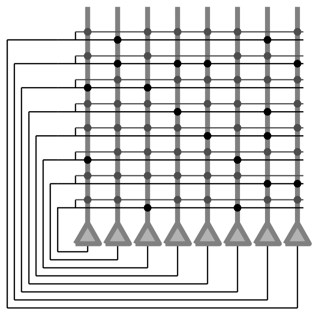
We mapped the real-valued synaptic strengths to the binary neurosynaptic core using the same procedure that we used for the RBM (Section 4.3): We imposed a global inhibitory rhythm that clocks neuron dynamics; we binarized the weight matrix (W_{ij}) by using two axons for each neuron's recurrent connections, setting postive weights to one on a positive axon ($S_i^0 > 0$) and the rest to one on a negative axon ($S_i^1 < 0$) and setting S_i^0, S_i^1 such that the sum total of excitation and inhibition were preserved. Then, we set all neurons' thresholds ($\theta_i = 1$) so that any net excitation caused a neuron to spike, signalling the recall of a `1`.

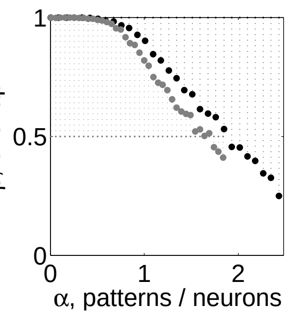
We tested the Hopfield Network's capacity, the number of patterns it can store, as well as its completion, how well it recalls stored patterns. To test capacity, we activated each pattern in its entirety (10 sets of patterns) and after 10 time steps observed similarity between the final state and stored pattern, given by the overlap:

$$\beta = \frac{1}{n} \sum_{i=1}^{n} (p_i - \gamma)(v_i - \gamma)/\gamma/(1-\gamma)$$

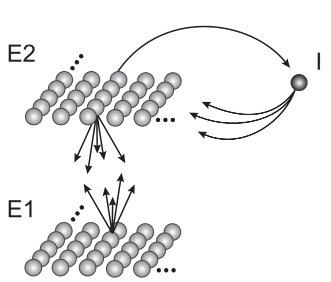
where p is the original pattern, and v is the network state. We found that for few patterns stored ($\alpha = m/n < 0.6$) patterns were well maintained, with $\beta \approx 1$; as more patterns were stored, storage degraded, with with β decreasing further below 1 as the number of stored patterns increased (Fig. 5(b)). We tested the network's completion by activating half the `1`s in each pattern ($\beta = 0.5$) and observing how the pattern overlap β changed. For few patterns stored ($\alpha < 0.6$), patterns were faithfully recalled; β increased to ≈ 1. As the number of pattern stored increased ($0.6 < \alpha < 1.0$), the network state moved closer to stored patterns but did not consistently recall full patterns; β increased above 0.5 but did not reach 1. As the number of pattern stored was further increased ($\alpha > 1.0$), the network state moved further from the stored patterns; β decreased below 0.5, degrading from the partially activated pattern.

In a similar manner to the Hopfield network, we implemented autoassociative pattern recall in a two-layer spiking network. The two-layer spiking network stores patterns in the connections between two excitatory layers of neurons [19]. The first layer, E1, acts as both the input and output. When activated, E1 neurons drive the second layer, E2, activating only the neuron(s) most selective to the input, ensured by winner-take-all inhibition mediated by a neuron in layer I (Fig. 5(c)). Through reciprocal connections, activity in E2

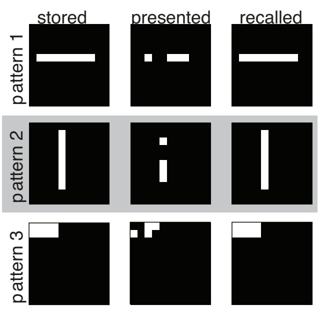

(a)



(b)



(c)



(d)

Fig. 5. (a) The Hopfield Network stores patterns in a recurrent weight matrix. (b) When the ratio of patterns to neurons (α) increases, when initialized to the stored patterns (black), overlap (β) decreases. When initialized to stored patterns with half of the ones dropped (gray), the overlap increases. (c) The two-layer network (E1 and E2) stores patterns in reciprocal connections between the excitatory layers. (d) When presented with stored patterns with half of the ones dropped, the network restores the patterns.

drives recall of all E1 neurons in the original pattern. In a system with 121 neurons in both E1 and E2, we stored 121 patterns consisting of 8 randomly selected E1 neurons. The patterns were learned offline by initializing weights between E1 and E2 to `0`, then assigning each pattern to a unique E2 neuron and setting the weights between that E2 neuron and the pattern in E1 to `1`. To test completion, we activated half the E1 neurons in a pattern each with probability 0.1 per step for 50 time steps (20 trials per pattern). The two-layer network showed 0.995 correct hit rate, and 0.011 false positive rate (Fig. 5(d)).

V. Discussion

A long standing goal of the neuromorphic community is to build compact, scalable, and power-efficient neural hardware that is supported by a corresponding software environment. Two recent notable projects with similar aims are PyNN [20] and the SpiNNaker project [21]. Our breakthrough neurosynaptic core is the first of its kind in working silicon to integrate neurons, dense synapses, and communication on chip, leading to ultra-low active power in a dense technology, while simultaneously demonstrating one-to-one correspondence with a determinstic neural programming model.

As we look forward to building multicore chips with tens of thousands of neurons and millions of synapses, the challenge becomes searching the astronomic parameter space to find a configuration that achieves a desired neural function. In the current neural programming model, our approach is to learn parameters offline where any optimization technique can be used (i.e., the algorithm may or may not be neural inspired). An alternative approach is to integrate learning into the neural hardware itself (e.g., using spike timing-dependent plasticity), allowing connections to be updated during run time (see [22], [23] for examples). This approach, however, is limited to learning rules that are efficient to implement in hardware. We believe that a hybrid of these two approaches is the best, similar to evolutionary biology: nature provides a hardwired scaffold, while nurture allows for online adaptation.

Appendix A
Neural Programs (Source)

The neurosynaptic core is defined as the set of parameters:

$$\{W_{ji}, G_j, S_i^{G_j}, L_i, \theta_i\}$$

For each of the four demos, the neural parameter values used to program each application, are listed in Table I. The synaptic connectivity matrix W_{ji}, is shown in Fig. 6.

Acknowledgment

This research was sponsored by DARPA under contract No. HR0011-09-C-0002. The views and conclusions contained herein are those of the authors should not be interpreted as representing the official policies, either expressly or implied, of DARPA or the U.S. Government. Authors thank Sue Gilson, Greg Corrado, Scott Hall, Kavita Prasad, Scott Fairbanks, Jon Tse, Ben Hill, Robert Karmazin, and Carlos Ortega for their contributions. Approved for Public Release, Distribution Unlimited.

References

[1] C. Mead, *Analog VLSI and neural systems*. Addison-Wesley, 1989.

[2] K. Boahen, "Neurogrid: emulating a million neurons in the cortex," in *IEEE international conference of the engineering in medicine and biology society*, 2006.

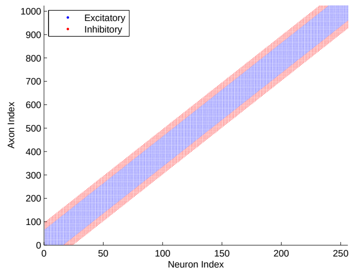
[3] G. Indiveri, E. Chicca, and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *IEEE Transactions on Neural Networks*, vol. 17, no. 1, pp. 211–221, 2006.

[4] A. Montalvo, R. Gyurcsik, and J. Paulos, "Building blocks for a temperature-compensated analog vlsi neural network with on-chip learning," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 6, 1994, pp. 363–366.

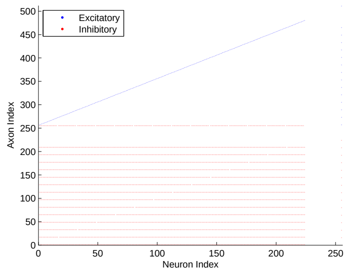
[5] R. Ananthanarayanan, S. Esser, H. Simon, and D. Modha, "The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 63.

[6] B. Shi, E. Tsang, S. Lam, and Y. Meng, "Expandable hardware for computing cortical feature maps," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2006.

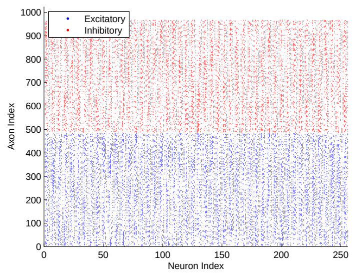
[7] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. Veidenbaum, "Efficient simulation of large-scale spiking neural networks using cuda graphics processors," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2009, pp. 3201–3208.



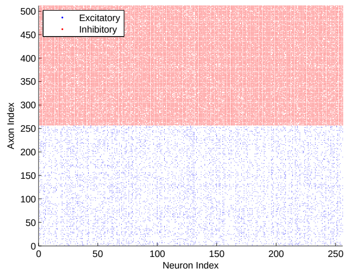
(a) Driver



(b) Pong



(c) RBM



(d) Hopfield

Fig. 6. Synapse connectivity matricies, W_{ji}. Excitatory synapses are blue, inhibitory synapses are red.

TABLE I
NEURAL PARAMETER SUMMARY

PARAMETER	**DRIVER**	**PONG**	**RBM**	**Hopfield**
W_{ji}	$\in \{0,1\}$	$\in \{0,1\}$	$\in \{0,1\}$	$\in \{0,1\}$
G_j	j=1:2:1023 : 0 (exc.)	j=0:255 : 0 (inh.)	j=0:483 : 0 (exc.)	j=0:255 : 0 (exc.)
G_j	j=0:2:1022 : 1 (inh.)	j=256:511 : 1 (exc.)	j=484:967 : 1 (inh.)	j=256:511 : 1 (inh.)
S_i^0	$+2$	-10	$[+2,+6]$	$[+1,+4]$
S_i^1	-2	9	$[-6,-2]$	$[-4,-1]$
L_i	0	-5	0	0
θ_i	$64+\text{rand}([0,64])$	5	$[1,80]$	1

[8] L. Maguire, T. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, no. 1-3, 2007.

[9] A. Anonymous, "A Digital Neurosynaptic Core Using Embedded Crossbar Memory with 45pJ per Spike in 45nm," in *In Submission to CICC*. IEEE, 2011.

[10] N. Imam and R. Manohar, "Address-event communication using token-ring mutual exclusion," *Proceedings of the IEEE International Symposium on Asynchronous Circuits*, 2011.

[11] R. Manohar, "A case for asynchronous computer architecture," *Proceedings of the ISCA Workshop on Complexity-Effective Design*, 2000.

[12] A. Borst, M. Egelhaaf, and J. Haag, "Mechanisms of dendritic integration underlying gain control in fly motion-sensitive interneurons," *Journal of Computational Neuroscience*, vol. 2, pp. 5–18, 1995, 10.1007/BF00962705. [Online]. Available: http://dx.doi.org/10.1007/BF00962705

[13] S. Shalev-shwartz and Y. Singer, "A new perspective on an old perceptron algorithm," in *Computational Learning Theory*, 2005, pp. 264–278.

[14] Y. LeCun and C. Cortes. The MNIST Database of Handwritten Digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[15] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, p. 504, 2006.

[16] P. Merolla, T. Ursell, and J. Arthur, "The thermodynamic temperature of a rhythmic spiking network," *CoRR*, vol. abs/1009.5473, 2010.

[17] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Science*, vol. 79, 1982.

[18] G. Palm, F. Schwenker, and F. Sommer, "Neural associative memories," *Biological Cybernetics*, vol. 36, 1993.

[19] S. Esser, A. Ndirango, and D. Modha, "Binding sparse spatiotemporal patterns in spiking computation," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2010, pp. 1–9.

[20] A. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "PyNN: a common interface for neuronal network simulators," *Frontiers in neuroinformatics*, vol. 2, 2008.

[21] X. Jin, F. Galluppi, C. Patterson, A. Rast, S. Davies, S. Temple, and S. Furber, "Algorithm and software for simulation of spiking neural networks on the multi-chip spinnaker system," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2010, pp. 1–8.

[22] A. Anonymous, "A 45nm CMOS Neuromorphic Chip with a Scalable Architecture for Learning in Networks of Spiking Neurons," in *In Submission to CICC*. IEEE, 2011.

[23] J. Arthur and K. Boahen, "Recurrently connected silicon neurons with active dendrites for one-shot learning," *Neural Networks, Proceedings*, vol. 3, pp. 1699 – 1704, 2004.