Loihi: a Neuromorphic Manycore Processor with On-Chip Learning

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steve McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, Hong Wang
Contact: mike.davies@intel.com
Intel Labs, Intel Corporation

Abstract—Loihi is a 60 mm^2 chip fabricated in Intel's 14nm process that advances the state-of-the-art modeling of spiking neural networks in silicon. It integrates a wide range of novel features for the field, such as hierarchical connectivity, dendritic compartments, synaptic delays, and most importantly programmable synaptic learning rules. Running a spiking convolutional form of the Locally Competitive Algorithm, Loihi can solve LASSO optimization problems with over three orders of magnitude superior energy-delay-product compared to conventional solvers running on a CPU iso-process/voltage/area. This provides an unambiguous example of spike-based computation outperforming all known conventional solutions.

1 Introduction

Neuroscience offers a bountiful source of inspiration for novel hardware architectures and algorithms. Through their complex interactions at large scales, biological neurons exhibit an impressive range of behaviors and properties that we currently struggle to model with modern analytical tools, let alone replicate with our design and manufacturing technology. Some of the magic that we see in the brain undoubtedly stems from exotic device and material properties that will remain out of our fabs' reach for many years to come. Yet highly simplified abstractions of neural networks are now revolutionizing computing by solving difficult and diverse machine learning problems of great practical value. Perhaps other less simplified models may also yield near-term value.

Artificial neural networks (ANNs) are reasonably well served by today's von Neumann CPU architectures and GPU variants, especially when assisted by coprocessors optimized for streaming matrix arithmetic. Spiking neural network models, on the other hand, are exceedingly poorly served by conventional architectures. Just as the value of ANNs was not fully appreciated until the advent of sufficiently fast CPUs and GPUs, the same could be the case for spiking models—except different computing architectures will be required.

The neuromorphic computing field of research spans a range of different neuron models and levels of abstraction. Loihi (pronounced "low-EE-hee") is one stake in the ground motivated by a particular class of algorithmic results and perspectives from our survey of computational neuroscience and recent neuromorphic advances. We approach the field with an eye for mathematical rigor, top-down modeling, rapid architecture iteration, and quantitative benchmarking. Our aim is to develop algorithms and hardware in a principled way as much as possible.

We begin this paper with our definition of the SNN computational model and the features that motivated Loihi's architectural requirements. We then describe the architecture that supports those requirements and provide an overview of the chip's asynchronous design implementation. We conclude with some preliminary 14nm silicon results.

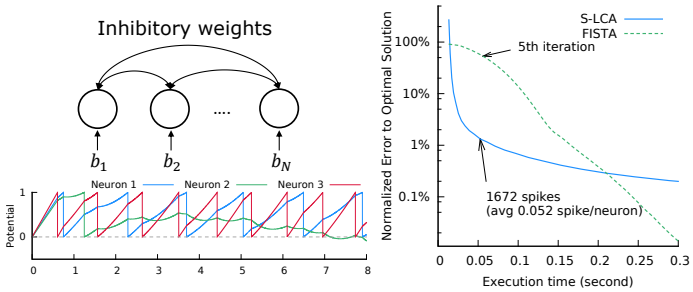
Importantly, Section 2.2 presents a result that unambiguously demonstrates the value of spike-based computation for one foundational problem. We view this as a significant result in light of ongoing debate about the value of spikes as a computational tool in both mainstream and neuromorphic communities. The skepticism towards spikes is well founded, but in our research we have moved on from this question, given the existence of an example that potentially generalizes to a very bload class of neural networks, namely all recurrent networks.

2 Spiking Neural Networks

We consider a spiking neural network (SNN) as a model of computation with neurons as the basic processing elements. Different from artificial neural networks, SNNs incorporate time as an explicit dependency in their computations. At some instant in time, one or more neurons may send out single-bit impulses, the *spike*, to neighbors through directed connections known as *synapses*, with a potentially nonzero traveling time. Neurons have local state variables with rules governing their evolution and timing of spike generation. Hence the network is a dynamical system where individual neurons interact through spikes.

2.1 Spiking Neural Unit

A spiking neuron integrates its spike train input in some fashion, usually by low pass filter, and fires once a state variable exceeds a threshold. Mathematically, each spike



(a) (b)

Fig. 1: (a) The network topology for solving LASSO. Each neuron receives the correlation b_i between the input data and a predefined feature vector as its input. Bottom figure shows the evolution of membrane potential in a 3-neuron example; the spike rates of the neurons stabilizes to fixed values. (b) Algorithmic efficiency comparison of a solution based on spiking network (S-LCA) and conventional optimization methods (FISTA). Both algorithms are implemented on a CPU with single thread. Y-axis is the normalized difference to the optimal objective function value. Figures taken from [1] with detailed information therein.

train is a sum of Dirac delta functions $\sigma(t) = \sum_k \delta(t - t_k)$ where t_k is the time of the k-th spike. We adopt a variation of the well-known CUBA leaky-integrate-and-fire model that has two internal state variables, the *synaptic response current* $u_i(t)$ and the *membrane potential* $v_i(t)$. The synaptic response current is the sum of filtered input spike trains and a constant bias current.

$$u_i(t) = \sum_{j \neq i} w_{ij}(\alpha_u * \sigma_j)(t) + b_i \tag{1}$$

where w_{ij} is the synaptic weight from neuron-j to i, $\alpha_u(t) = \tau_u^{-1} \exp(-t/\tau_u)H(t)$ is the synaptic filter impulse response parameterized by the time constant τ_u with $H(t)$ the unit step function, and b_i is a constant bias. The synaptic current is further integrated as the membrane potential, and the neuron sends out a spike when its membrane potential passes its firing threshold θ_i.

$$\dot{v_i}(t) = -\frac{1}{\tau_v} v_i(t) + u_i(t) - \theta_i \sigma_i(t) \tag{2}$$

Note that the integration is leaky, as captured by the time constant τ_v. v_i is initialized with a value less than θ_i, and is reset to 0 right after a spiking event occurs.

Loihi, a fully digital architecture, approximates the above continuous time dynamics using a fixed-size discrete timestep model. In this model, all neurons need to maintain a consistent understanding of time so their distributed dynamics can evolve in a well-defined, synchronized manner. It is worth clarifying that these fixed-size, synchronized time steps relate to the *algorithmic time* of the computation, and need not have a direct relationship to the hardware execution time.

2.2 Computation with Spikes and Fine-grained Parallelism

Computations in SNNs are carried out through the interacting dynamics of neuron states. An instructive example is the ℓ_1-minimizing sparse coding problem, also known as LASSO, which we can solve with the SNN in Figure 1a using the Spiking Locally Competitive Algorithm [2]. The objective of this problem is to determine a sparse set of coefficients that best represents a given input as the linear combination of features from a feature dictionary. The coefficients can be viewed as the activities of the spiking neurons in Figure 1a that are competing to form an accurate representation of the data. By properly configuring the network, it can be established that as the network dynamics evolve, the *average spike rates* of the neurons will converge to a fixed point, and this fixed point is identical to the solution of the optimization problem.

Such computation exhibits completely different characteristics from conventional linear algebra based approaches. Figure 1b compares the computational efficiency of an SNN with the conventional solver FISTA [3] by having them both solve a sparse coding problem on a single-threaded CPU. The SNN approach (labelled S-LCA) gives a rapid initial drop in error and obtains a good approximate solution faster than FISTA. After this, the S-LCA convergence speed significantly slows down, and FISTA instead finds a much more precise solution quicker. Hence an interesting efficiency-accuracy tradeoff arises that makes the SNN solution particularly attractive for applications that do not require highly precise solutions, e.g., a solution that is 1% within the optimal.

The remarkable algorithmic efficiency of S-LCA can be attributed to its ability to exploit the temporal ordering of spikes, a general property of the SNN computational model. In Figure 1a, the neuron that has the largest external input to win the competition is more likely to spike at the earliest time, causing immediate inhibition of the other neurons. This inhibition happens with only a single one-to-many spike communication, in contrast to the usual need for all-to-all state exchanges with matrix arithmetic based solutions such as FISTA and other conventional solvers. This implies that the SNN solution is communication efficient, and it may solve the optimization problem with a reduced number of arithmetic operations. We point interested readers to [1] for more discussions.

Our CPU-based evaluation has yet to exploit one important advantage of SNN-based algorithms: the inherent abundant parallelism. The dominant part of SNN computations—the evolution of individual neuron states within a timestep—can all be computed concurrently. However, harnessing such speedup can be a nontrivial task especially on a conventional CPU architecture. The parallelizable work for each neuron only consists of a few variable updates. Given that the parallel segment of the work can be executed very quickly, the underlying architecture must support a fine granularity of parallelism with minimal overhead in coordinating the order of computations. These observations motivate fundamental features of the Loihi architecture, described in Section 3.

2.3 Learning with Local Information

Learning in an SNN refers to adapting the synaptic weights and hence varying the SNN dynamics to a desired one. Similar to conventional machine learning, we wish to express

learning as the minimization of a particular loss function over many training samples. In the sparse coding case, learning involves finding the set of synaptic weights that allows the best performing sparse representation, expressed as minimizing the sum of all sparse coding losses. Learning in an SNN naturally proceeds in an online manner, where training samples are sent to the network sequentially.

SNN synaptic weight adaptation rules must satisfy a locality constraint: each weight can only be accessed and modified by the destination neuron, and the rule can only make use of locally available information, such as the spike trains from the presynaptic (source) and postsynaptic (destination) neurons. The locality constraint imposes a significant challenge on the design of learning algorithms, as most conventional optimization procedures do not satisfy it. Although the development of such decentralized learning algorithms is still in active research, some pioneering work exists showing the promise of this approach. They range from the simple Oja's rule for finding principal components, to the Widrow-Hoff rule for supervised learning and its generalization to exploit precise spike timing information [4], to the more complex unsupervised sparse dictionary learning using feedback [5] and contrastive divergence for RBMs [6].

Once a learning rule satisfies the locality constraint, the inherent parallelism offered by SNNs will then allow the adaptive network to be scaled up to large sizes in a way that can be computed efficiently. If the rule also minimizes a loss function, then the system will have well defined dynamics.

To support the development of such scalable learning rules, Loihi offers a variety of local information to a programmable synaptic learning process:

- *Spike traces* corresponding to filtered presynaptic and postsynaptic spike trains with configurable time constants (Section 3.4.4). In particular, a short time constant allows the learning rule to utilize precise spike timing information, while a long time constant captures the information in spike rates.
- Multiple spike traces for a given spike train filtered with different time constants. This provides support for differential Hebbian learning by measuring perturbations in spike patterns and Bienenstock-Cooper-Munro learning using triplet STDP [7], among others.
- Two additional state variables per synapse, besides the normal weight, in order to provide more flexibility for learning. For example, these can be used as *synaptic tags* for reinforcement learning.
- Reward traces that correspond to special *reward spikes* carrying signed impulse values to represent reward or punishment signals for reinforcement learning. Reward spikes are broadcast to defined sets of synapses in the network that may connect to many different source and destination neurons.

Loihi is the first fully integrated digital SNN chip that supports any of the above features. Some small-scale neuromorphic chips with analog synapse and neuron circuits have prototyped synaptic plasticity using spike traces, for example [8], but these prior chips have orders of magnitude lower network capacity compared to Loihi as well as far less programmability.

2.4 Other Computational Primitives

Loihi includes several computational primitives related to other active areas of SNN algorithmic research:

- Stochastic noise. Uniformly distributed pseudorandom numbers may be added to a neuron's synaptic response current, membrane voltage, and refractory delay. This provides support for algorithms such as Neural Sampling [9], which can solve probabilistic inference and constraint satisfaction problems using stochastic dynamics and a form of Markov chain Monte Carlo sampling.
- Configurable and adaptable synaptic, axon, and refractory delays. This provides support for novel forms of temporal computation such as *polychronous dynamics* [10], in which subsets of neurons may synchronize over periods of varying timescales. The number of polychronous groups far exceeds the number of stable attractors in conventional attractor networks, suggesting a productive space for computational development.
- Configurable dendritic tree processing. Neurons in the SNN may be decomposed into a tree of *compartment* units, with the neuron's input synapses distributed over those compartments. Each compartment supports the same state variables as a neuron, but only the root of the tree (soma compartment) generates spike outputs. The compartments' state variables are combined in a configurable manner by programming different *join functions* for each compartment junction.
- Neuron threshold adaptation in support of intrinsic excitability homeostasis.
- Scaling and saturation of synaptic weights in support of "permanence" levels that exceed the range of weights used during inference.

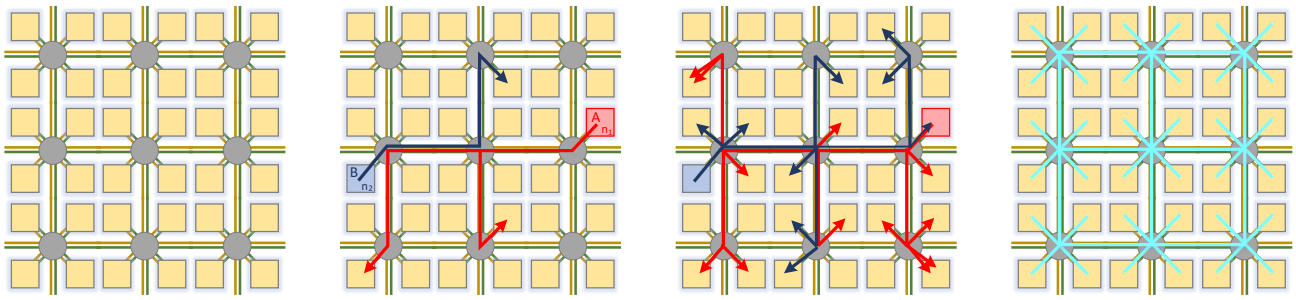
The combination of these features in one device, especially in combination with Loihi's learning capabilities, is novel for the field of SNN silicon implementation.

3 Architecture

3.1 Chip Overview

Loihi features a manycore mesh comprising 128 neuromorphic cores, three embedded x86 processor cores, and off-chip communication interfaces that hierarchically extend the mesh in four planar directions to other chips. An asynchronous network-on-chip (NoC) transports all communication between cores in the form of packetized messages. The NoC supports write, read request, and read response messages for core management and x86-to-x86 messaging, spike messages for SNN computation, and barrier messages for time synchronization between cores. All message types may be sourced externally by a host CPU or on-chip by the x86 cores, and these may be directed to any on-chip core. Messages may be hierarchically encapsulated for off-chip communication over a second-level network. The mesh protocol supports scaling to 4096 on-chip cores and, via hierarchical addressing, up to 16,384 chips.

Each neuromorphic core implements 1,024 primitive spiking neural units (compartments) grouped into sets of



(a) Initial idle state for timestep t. Each square represents a core in the mesh containing multiple neurons

(b) Neurons n_1 and n_2 in cores A and B fire and generate spike messages

(c) Spikes from all other neurons firing on timestep t in cores A and B are distributed to their destination cores

(d) Each core advances its algorithmic timestep to $t + 1$ as it handshakes with its neighbors via barrier synchronization messages

Fig. 2: Mesh Operation

trees constituting neurons. The compartments, along with their fanin and fanout connectivity, share configuration and state variables in ten architectural memories. Their state variables are updated in a time-multiplexed, pipelined manner every algorithmic timestep. When a neuron's activation exceeds some threshold level, it generates a spike message that is routed to a set of fanout compartments contained in some number of destination cores.

Flexible and well provisioned SNN connectivity features are crucial for supporting a broad range of workloads. Some desirable networks may call for dense, all-to-all connectivity while others may call for sparse connectivity; some may have uniform graph degree distributions, others power-law distributions; some may require high precision synaptic weights, *e.g.* to support learning, while others can make do with binary connections. As a rule, algorithmic performance scales with increasing network size, measured not only by neuron counts but especially neuron-to-neuron fanout degrees. We see this rule holding all the way to biological levels (1:10,000). Due to the $O(N^2)$ scaling of connectivity state in the number of fanouts, it becomes an enormous challenge to support networks with high connectivity using today's integrated circuit technology.

To address this challenge, Loihi supports a range of features to relax the sometimes severe constraints that other neuromorphic designs have imposed on the programmer:

1) Sparse network compression. Besides a common dense matrix connectivity model, Loihi supports three sparse matrix compression models in which fanout neuron indices are computed based on index state stored with each synapse's state variables.
2) Core-to-core multicast. Any neuron may direct a single spike to any number of destination cores, as the network connectivity may require.
3) Variable synaptic formats. Loihi supports any weight precision between one and nine bits, signed or unsigned, and weight precisions may be mixed (with scale normalization) even within a single neuron's fanout distribution.
4) Population-based hierarchical connectivity. As a generalized weight sharing mechanism, *e.g.* to support convolutional neural network types, connectivity templates may be defined and mapped to specific population instances during operation. This feature can reduce a network's required connectivity resources by over an order magnitude.

Loihi is the first fully integrated SNN chip that supports any of the above features. All prior chips, for example the previously most synaptically dense chip [11], store their synapses in dense matrix form that significantly constrains the space of networks that may be efficiently supported.

Each Loihi core includes a programmable learning engine that can evolve synaptic state variables over time as a function of historical spike activity. In order to support the broadest possible class of rules, the learning engine operates on filtered spike traces. Learning rules are microcode-programmable and support a rich selection of input terms and output synaptic target variables. Specific sets of these rules are associated with a *learning profile* bound to each synapse to be modified. The profile is mapped by some combination of presynaptic neuron, postsynaptic neuron, or class of synapse. The learning engine supports simple pairwise STDP rules and also much more complicated rules such as triplet STDP, reinforcement learning with synaptic tag assignments, and complex rules that reference both rate-averaged and spike-timing traces.

All logic in the chip is digital, functionally deterministic, and implemented in an asynchronous *bundled data* design style. This allows spikes to be generated, routed, and consumed in an event-driven manner with maximal activity gating during idle periods. This implementation style is well suited for spiking neural networks that fundamentally feature a high degree of sparseness in their activity across both space and time.

3.2 Mesh Operation

Figure 2 shows the operation of the neuromorphic mesh as it executes a spiking neural network model. All cores begin at algorithmic timestep t. Each core independently iterates over its set of neuron compartments, and any neurons that enter a *firing* state generate spike messages that the NoC distributes to all cores that contain their synaptic fanouts. Spike distributions for two such example neurons n_1 and n_2 in cores A and B are illustrated in Figure 2b, with additional

spike distributions from other firing neurons adding to the NoC traffic in Figure 2c.

The NoC distributes spike (and all other) messages according to a dimension-order routing algorithm. The NoC itself only supports unicast distributions. To multicast spikes, the output process of each core iterates over a list of destination cores for a firing neuron's fanout distribution and sends one spike per core. For deadlock protection reasons relating to read and chip-to-chip message transactions, the mesh uses two independent physical router networks. For bandwidth efficiency, the cores alternate sending their spike messages across the two physical networks. This is possible because SNN computation does not depend on the spike sequence ordering within a timestep.

At the end of the timestep, a mechanism is needed to ensure that all spikes have been delivered and that it's safe for the cores proceed to timestep $t + 1$. Rather than using a globally distributed time reference (clock) that must pessimize for the worst-case chip-wide network activity, we use a *barrier synchronization* mechanism, illustrated in Figure 2d. As each core finishes servicing its compartments for timestep t, it exchanges *barrier* messages with its neighboring cores. The barrier messages flush any spikes in flight and, in a second phase, propagate a timestep-advance notification to all cores. As cores receive the second phase of barrier messages, they advance their timestep and proceed to update compartments for time $t + 1$.

As long as management activity is restricted to a specific "preemption" phase of the barrier synchronization process that any embedded x86 core or off-chip host may introduce on demand, the Loihi mesh is provably deadlock free.

3.3 Network Connectivity Architecture

In its most abstract formulation, the neural network mapped to the Loihi architecture is a directed multigraph structure $\mathcal{G} = (\mathcal{N}, \mathcal{S})$, where $\mathcal{N}$ is the set of neurons in the network and $\mathcal{S}$ is a set of synapses (edges) connecting pairs of neurons. Each synapse $s \in \mathcal{S}$ corresponds to a 5-tuple: $(i, j, \texttt{wgt}, \texttt{dly}, \texttt{tag})$, where $i, j \in \mathcal{N}$ identify the source and destination neurons of the synapse, and `wgt`, `dly`, and `tag` are integer-valued properties of the synapse. In general, Loihi will autonomously modify the synaptic variables $(\texttt{wgt}, \texttt{dly}, \texttt{tag})$ according to programmed learning rules. All other network parameters remain constant unless they are modified by x86 core intervention.

An abstract network is mapped to the mesh by assigning neurons to cores, subject to each core's resource constraints. Figure 3 shows an example of a simple seven neuron network mapped to three cores. Given a particular neuron-to-core mapping for $\mathcal{N}$, each neuron's synaptic fanin state (`wgt`, `dly`, and `tag`) must be stored in the core's synaptic memory. These schematically correspond to the synaptic spike markers in Figure 3. Each neuron's fanout edges are projected to a list of core-to-core edges, and each core-to-core edge is assigned an `axon_id` identifier unique to each destination core. The neuron's synaptic fanout contained within each destination core is associated with the corresponding `axon_id` and organized as a list of 4-tuples $(j, \texttt{wgt}, \texttt{dly}, \texttt{tag})$ stored in the synaptic memory in some suitably compressed form. When neuron i spikes, the mesh

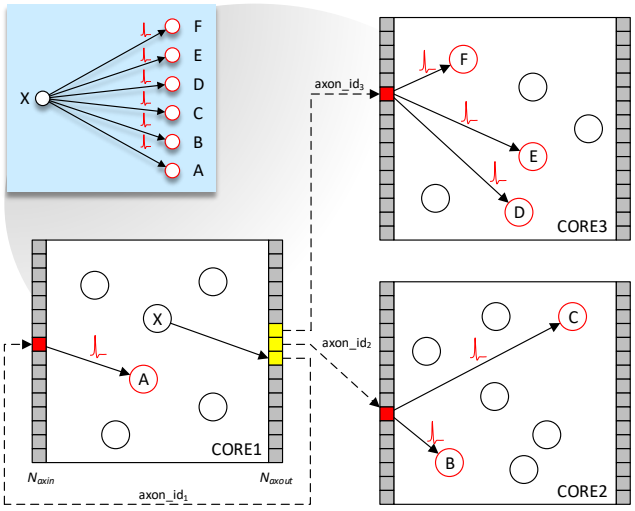


Fig. 3: Neuron-to-neuron mesh routing model

routes each `axon_id` to the appropriate fanout core which then expands it to the corresponding synaptic list

This connectivity architecture can support arbitrary multigraph networks subject to the cores' resource constraints:

1) The total number of neurons assigned to any core may not exceed 1,024 (N_{cx}).
2) The total synaptic fanin state mapped to any core must not exceed 128KB ($N_{syn} \times 64b$, subject to compression and list alignment considerations.)
3) The total number of core-to-core fanout edges mapped to any given core must not exceed 4,096 (N_{axout}). This corresponds to the number of output-side routing slots highlighted in yellow in Figure 3.
4) The total number of distribution lists, associated by `axon_id`, in any core must not exceed 4,096 (N_{axin}). This is the number of input-side `axon_id` routing slots highlighted in red in Figure 3.

In practice, constraints 2 and 4 tend to be the most limiting.

In order to exploit structure that may exist in the network $\mathcal{G}$, Loihi supports a hierarchical network model. This feature can significantly reduce the chip-wide connectivity and synaptic resources needed to map convolutional-style networks in which a template of synaptic connections is applied to many neurons in a uniform way.

Formally, we represent the hierarchical template network as a directed multigraph $\mathcal{H} = (\mathcal{T}, \mathcal{E})$ where $\mathcal{T}$ is a set of disjoint *neuron population types* and $\mathcal{E}$ defines a set of edges connecting between pairs $T_{src}, T_{dst} \in \mathcal{T}$. An edge $E \in \mathcal{E}$ associated with the (T_{src}, T_{dst}) population type pair is a set of synapses where each $s \in E$ connects a neuron $i \in T_{src}$ to to a neuron $j \in T_{dst}$.

In order to hierarchically compress the resource mapping of the desired flat network $\mathcal{G} = (\mathcal{N}, \mathcal{S})$, a set of disjoint neuron populations instances $\mathcal{P}$ must be defined where each $P \in \mathcal{P}$ is a subset of neurons $P \subset \mathcal{N}$. Each population instance is associated with a population type $T \in \mathcal{T}$ from the hierarchical template network $\mathcal{H}$. Neurons $n \in \mathcal{N}$ belonging to some population instance $P \in \mathcal{P}$ are said to be *population-mapped*. By configuring the $\mathcal{H}$ connectivity in

hardware, the redundant connectivity in $\mathcal{G}$ is implied and doesn't consume resources, beyond what it takes to map the population-level connectivity of $\mathcal{H}$ as if it were a flat network.

Population-mapped neurons produce *population spike* messages whose `axon_id` fields identify (1) the destination population P_{dst}, (2) the source neuron index $i \in P_{src}$ within the source population, and (3) the particular edge connecting between T_{src} and T_{dst} when there is more than one. One population spike must be sent per destination population rather than per destination core, as in the flat case. This marginally higher level of spike traffic is more than offset by the savings in network mapping resources.

Convolutional artificial neural networks (ConvNets), in which a single kernel of weights is repeatedly applied to different patches of input pixels, is an example class of network that greatly benefits from hierarchy. By treating such a weight kernel as the template connectivity that is applied to the different image patches (population instances), Loihi can support a spiking form of such networks. The S-LCA network discussed in Section 5.2 features a similar kernel-style convolutional network topology which additionally includes lateral inhibitory connections between the feature neurons of each population instance.

3.4 Learning Engine

3.4.1 Baseline STDP

A number of neuromorphic chip architectures to date have incorporated the most basic form of pairwise, nearest-neightbor spike time dependent plasticity (STDP). Pairwise STDP is simple, event-driven, and highly amenable to hardware implementation. For a given synapse connecting presynaptic neuron j to postsynaptic neuron i, an implementation needs only maintain the most recent spike times for the two neurons (t_j^{pre} and t_i^{post}). Given a spike arrival at time t, one local nonlinear computation needs to be evaluated in order to update the synaptic weight:

$$\Delta w_{i,j} = \begin{cases} A_- \mathcal{F}(t - t_i^{post}), & \text{On presynaptic spike} \\ A_+ \mathcal{F}(t - t_j^{pre}), & \text{On postsynaptic spike} \end{cases} \quad (3)$$

where $\mathcal{F}(x)$ is some approximation of $e^{-x/\tau} \cdot H(x)$, for constants $A_- < 0$, $A_+ > 0$, and $\tau > 0$. Since a design must already perform a lookup of weight $w_{i,j}$ on any presynaptic spike arrival, the first case above matches the natural dataflow present in any neuromorphic implementation. To support this depressive half of the STDP learning rule, the handling of a presynaptic spike arrival simply turns a read of the weight state into a read-modify-write operation, assuming availability of the t^{post} spike time.

The potentiating half of Equation 3 is the only significant challenge that pairwise STDP introduces. To handle this weight update in an event-driven manner, symmetric to the depressive case, the implementation needs to perform a backwards routing table lookup, obtaining $w_{i,j}$ from the firing postsynaptic neuron i. This is at odds with the algorithmic impetus for more complex and diverse network routing functions $R : j \rightarrow Y$, where $i \in Y$. The more complex R becomes, the more expensive, in general, it becomes to implement an inverse lookup R^{-1} efficiently in hardware. Some implementations have explored creative solutions to this problem [12], but in general these approaches constrain network topologies and are not scalable.

For Loihi, we adopt a less event-driven *epoch-based* synaptic modification architecture in the interest of supporting arbitrarily complex R and extending the architecture to more advanced learning rules. This architecture delays the updating of all synaptic state to the end of a periodic learning epoch time T_{epoch}.

An epoch-based architecture fundamentally requires iteration over each core's active input axons, which Loihi does sequentially. In theory this is a disadvantage that a direct implementation of the R^{-1} reverse lookup may avoid. However, in practice, any pipelined digital core implementation still requires iteration over active input axons in order to maintain spike timestamp or trace state. Even the fully transposable synaptic crossbar architecture used in [12] includes an iteration over all input axons per timestep for this reason.

3.4.2 Advancing Beyond Pairwise STDP

A number of architectural challenges arise in the pursuit of supporting more advanced learning rules. First, the functional forms describing $\Delta w_{i,j}$ become more complex and seemingly arbitrary. These rules are at the frontier of algorithm research and therefore require a high degree of configurability. Second, the rules involve multiple synaptic variables, not just weights. Finally, advanced learning rules rely on temporal correlations in spiking activity over a range of timescales, which means more than just the most recent spike times must be maintained. These challenges motivate the central features of Loihi's learning architecture, described below.

3.4.3 Learning Rule Functional Form

On every learning epoch, a synapse will be updated whenever the appropriate pre- or post-synaptic conditions are satisfied. A set of microcode operations associated with the synapse determines the functional form of one or more transformations to apply to the synapse's state variables. The rules are specified in sum-of-products form:

$$z := z + \underbrace{\sum_{i=1}^{N_P} S_i \prod_{j=1}^{n_i} \underbrace{(V_{i,j} + C_{i,j})}_{T_{i,j}}}_{P_i} \quad (4)$$

where z is the transformed synaptic variable (either `wgt`, `dly`, or `tag`), $V_{i,j}$ refers to some choice of input variable available to the learning engine, and $C_{i,j}$ and S_i are microcode-specified signed constants.

Table 1 provides a comprehensive list of product terms as encoded by a 4-bit field in each microcode op. The multiplications and summations of Equation 4 are computed iteratively by the hardware and accumulated in 16-bit registers. The epoch period is globally configured per core up to a maximum value of 63, with typical values in the 2 to 8 range. To avoid receiving more than one spike in a given epoch, the epoch period is normally set to the minimum refractory delay of all neurons in the network.

Encoding	Term ($T_{i,j}$)	Bits	Description
0	$x_0 + C$	5b (U)	Presynaptic spike count
1	$x_1 + C$	7b (U)	1st presynaptic trace
2	$x_2 + C$	7b (U)	2nd presynaptic trace
3	$y_0 + C$	5b (U)	Postsynaptic spike count
4	$y_1 + C$	7b (U)	1st postsynaptic trace
5	$y_2 + C$	7b (U)	2nd postsynaptic trace
6	$y_3 + C$	7b (U)	3rd postsynaptic trace
7	$r_0 + C$	1b (U)	Reward spike
8	$r_1 + C$	8b (S)	Reward trace
9	$\texttt{wgt} + C$	9b (S)	Synaptic weight
10	$\texttt{dly} + C$	6b (U)	Synaptic delay
11	$\texttt{tag} + C$	9b (S)	Synaptic tag
12	$\text{sgn}(\texttt{wgt} + C)$	1b (S)	Sign of case 9 (± 1)
13	$\text{sgn}(\texttt{dly} + C)$	1b (S)	Sign of case 10 (± 1)
14	$\text{sgn}(\texttt{tag} + C)$	1b (S)	Sign of case 11 (± 1)
15	C	8b (S)	Constant term. (Variant 1)
15	$S_m \cdot 2^{S_e}$	4b (S)	Scaling term. 4b mantissa, 4b exponent. (Variant 2)

TABLE 1: Learning rule product terms

The basic pairwise STDP rule only requires two products involving four of these terms (0, 1, 3, and 4) and two constants. The Loihi microcode format can specify this rule in a single 32-bit word. With an encoding capacity of up to sixteen 32-bit words and the full range of terms in Table 1, the learning engine provides considerable headroom for far more complex rules.

3.4.4 Trace Evaluation

The trace variables ($x_1, x_2, y_1, y_2, y_3, r_1$) in Table 1 refer to filtered spike trains associated with each synapse that the learning engine modifies. The filtering function associated with each trace is defined by two configurable quantities: an impulse amount δ added on every spike event and a decay factor α. Given a spike arrival sequence $s[t] \in \{0, 1\}$, an ideal trace sequence $x[t]$ over time is defined as follows:

$$x[t] = \alpha \cdot x[t-1] + \delta \cdot s[t].$$

The Loihi hardware computes a low-precision (seven bit) approximation of this first-order filter using stochastic rounding.

By setting δ to 1 (typically with relatively small α), $x[t]$ saturates on each spike and its decay measures elapsed time since the most recent spike. Such trace configurations exactly implement the baseline STDP rules dependent only on nearest-neighbor pre/post spike time separations described in Section 3.4.1. On the other hand, setting δ to a value less than 1, specifically $1 - \alpha^{T_{\min}}$, where $T_{\min}$ is the minimum spike period, causes sufficiently closely spaced spike impulses to accumulate over time and $x[t]$ reflects the average *spike rate* over a timescale of $\tau = -1/\log \alpha$.

4 Design Implementation

4.1 Core Microarchitecture

Figure 4 shows the internal structure of the Loihi neuromorphic core. Colored blocks in this diagram represent the major memories that store the connectivity, configuration, and dynamic state of all neurons mapped to the core. The core's total SRAM capacity is 2Mb including ECC overhead. The coloring of memories and dataflow arcs illustrates the core's four primary operating modes: input spike handling (green), neuron compartment updates (purple), output spike generation (blue), and synaptic updates (red). Each of these modes operates independently with minimal synchronization at a variety of frequencies, based on the state and configuration of the core. The black structure marked UCODE represents the configurable learning engine.

The values annotated by each memory indicate its number of logical addresses, which correspond to the core's major resource constraints. The number of input and output axons (N_{axin} and N_{axout}), the synaptic memory size (N_{syn}), and the total number of neuron compartments (N_{cx}) impose network connectivity constraints as described in Section 3.3. The parameter N_{sdelay} indicates the minimum number of synaptic delay units supported, eight in Loihi. Larger synaptic delay values, up to 62, may be supported when fewer neuron compartments are needed by a particular mapped network.

Figure 4 shows the internal structure of the Loihi neuromorphic core. Colored blocks in this diagram represent the major memories that store the connectivity, configuration, and dynamic state of all neurons mapped to the core. The total SRAM capacity over all of these memories is 2Mb including ECC overhead. The coloring of memories and dataflow arcs illustrate the core's four primary operating modes: input spike handling (green), neuron compartment updates (purple), output spike generation (blue), and synaptic updates (red). Each of these modes, described below, operates independently based on the state of the core with minimal mutual synchronization.

The IN and OUT processes interface the core to the NoC. They parse incoming messages, direct management accesses to the appropriate sub-block, formulate read responses, implement the barrier synchronization protocol, and control power to the core's SRAM arrays when instructed to do so to manage leakage power.

4.1.1 Input Spike Handling

When a spike arrives at the core, the `axon_id` field is extracted and, by mapping through the `SYNAPSE_MAP` and `SYNAPSE_MEM` pair of memories, it is expanded into a list of 4-tuples: $(\texttt{c}_i, \texttt{wgt}_i, \texttt{dly}_i, \texttt{tag}_i)$. Each 4-tuple defines a synaptic accumulation operation to apply to compartment $\texttt{c}_i$ at timestep $t + \texttt{dly}_i + 1$. The `DENDRITE_ACCUM` memory serves as a temporal circular buffer accumulating the total amount of synaptic weight activation received for a particular $(\texttt{c}_i, t + \Delta t)$ pair of compartment and timestep. The synaptic accumulation operation is handled by a read-modify-write access:

$$\texttt{DENDRITE\_ACCUM}[\texttt{c}_i \times 2^{\texttt{DlyBits}} + (t + \texttt{dly}_i + 1) \% 2^{\texttt{DlyBits}}] \mathrel{+}= S(\texttt{wgt}_i).$$

`DlyBits` is a configuration parameter that allocates the 8,192 addresses in `DENDRITE_ACCUM` between delay resolution and compartment indices. With 1,024 compartments in use, synaptic delay ranges of 0 to 5 are supported. A maximum

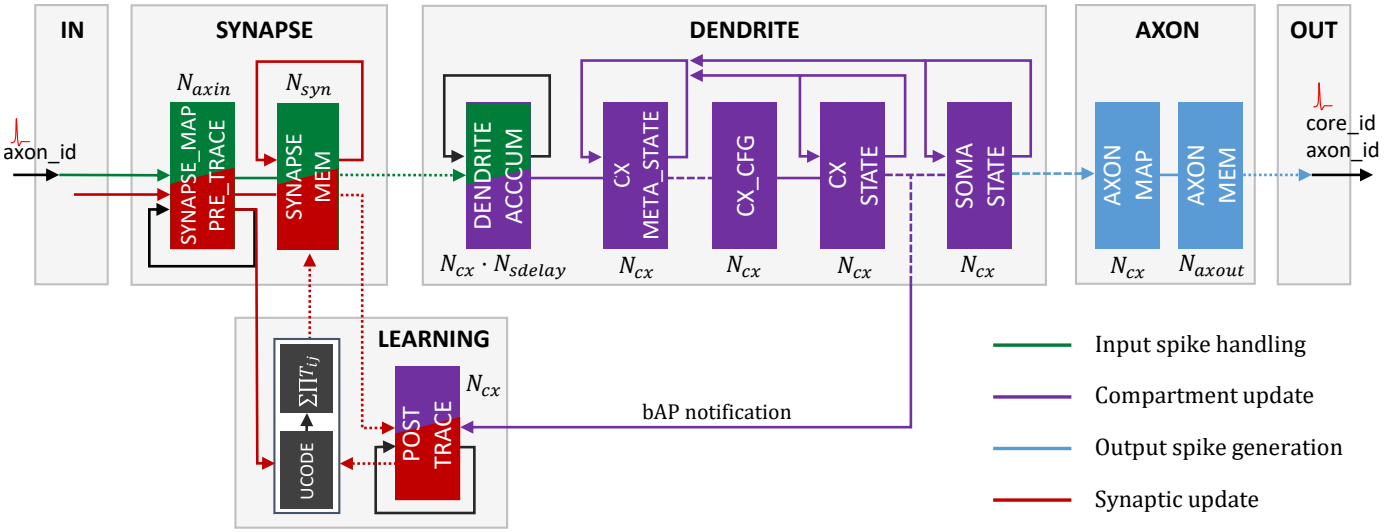


Fig. 4: Core Top-Level Microarchitecture. The SYNAPSE unit processes all incoming spikes and reads out the associated synaptic weights from the memory. The DENDRITE unit updates the state variables u and v of all neurons in the core. The AXON unit generates spike messages for all fanout cores of each firing neuron. The LEARNING unit updates synaptic weights using the programmed learning rules at epoch boundaries.

range of 0 to 61 can be supported when only 128 compartments are mapped to the core.

The function $S(w)$ scales and optionally saturates the weight value as stored in the synaptic memory. $S(w)$ is configured by the particular synaptic format used to unpack the synapse from SYNAPSE_MEM and can be used to normalize ranges of weights across different bit widths. This allows, for example, learning-enabled 8-bit weights and static 1-bit weights to coexist in the same network with the same range of syanptic strengths, allowing for better network compression. Saturation introduces nonlinearity in the weight scale that can be used to provide a range of synaptic "permanence" values that extend beyond the range of synaptic strengths used for synaptic accumulation at the destination neuron.

4.1.2 Neuron Compartment Updates

Concurrent to input spike handling, the core iterates over its mapped compartments $\mathtt{c} = 0 \ldots \mathtt{NumCx} - 1$, servicing the accumulated synaptic input for the current timestep t and evolving each compartment's dynamic state variables. This involves reading state and configuration memories associated with c (DENDRITE_ACCUM, CX_META_STATE, CX_STATE, and CX_CFG) and applying the neuron model dynamics from Section 2.1. Note that the accumulated value read from DENDRITE_ACCUM provides the summed value over all past synaptic input fanning into compartment c in Equation 1.

If the compartment c is configured as a soma type and the compartment exceeds its spiking threshold, then it generates a firing event (marked "bAP" in Figure 4, for backwards action potential) that is propataged backwards through the dendritic compartment tree to the learning engine, where the spike time t is recorded for each post-synaptic trace state entry in the POST_TRACE memory. The firing event is scheduled in an axon delay buffer maintained by the SOMA_STATE memory such that the outgoing spike can be serviced after $D_{\mathtt{c}}^{axon}$ timesteps have elapsed.

Updated state is written back to CX_META_STATE, CX_STATE, and, if applicable, SOMA_STATE.

The CX_META_STATE memory maintains the activity status of each compartment for activity gating purposes. An inactive compartment that receives no synaptic input on a particular timestep will not be updated, saving both processing time and energy. For example, this can occur when both u and v state variables are zero, the compartment's bias b is zero, and no previously generated spike is waiting for its axon delay to elapse.

4.1.3 Output Spike Handling

When a soma compartment's spike is ready for transmission, the AXON stage of the pipeline expands the spike to a list of NoC spike messages for delivery to the neuron's fanout cores and populations. The expansion process is similar to the one that occurs for ingress spikes in Synapse between SYNAPSE_MAP and SYNAPSE_MEM. The compartment number indexes the AXON_MAP memory, returning a pointer to AXON_MEM and a list length. Typically each 32-bit entry in the AXON_MEM list specifies a single spike message, although certain spike types can consume as many as three entries (*e.g.* long-format spike messages destined off-chip). The AXON_MEM memory contains 4,096 entries shared among all distribution lists, so a fully populated core with 1,024 neurons will typically support an average fanout factor of four cores per neuron. Higher fanout factors are possible when fewer neurons are mapped to the core or when the list lengths are nonuniform.

With the hierarchical connectivity model, all neurons belonging to the same population type share a single distribution list in AXON_MEM. This provides two levels of compression in AXON_MEM resources: one from neuron to population, the second from population to population type. Somewhat offsetting this savings is the need to route one

population spike for each fanout population, rather than one spike per fanout core, as with discrete spikes.

4.1.4 Synaptic Updates

Every T_{epoch} timesteps, when learning is enabled, all cores in the mesh enter a synaptic update phase of operation in which their synaptic variables are updated according to microcode-specified learning rules. During this phase, only the memories and dataflow shown in red in Figure 4 are active. The core iterates over all learning-enabled axons in the SYNAPSE_MAP memory. These axons have associated trace variables stored with them, conceptually as an interleaved PRE_TRACE memory, and at minimum the presynaptic trace variables (x_1, x_2) are updated to the current end-of-epoch time according to the spike filtering model described in Section 3.4.4. Similar updates are applied to the postsynaptic trace variables (y_1, y_2, y_3) stored in the POST_TRACE memory.

For each updated axon in SYNAPSE_MAP that either has active (nonzero) traces or is configured for unconditional update, the associated synaptic fanout list in SYNAPSE_MEM is dereferenced and all learning-enabled synapses are directed to the learning engine for processing. The learning engine associates each synapse to be updated with its postsynaptic traces, and as long as they are also active (or unconditionally enabled), the appropriate microcode-specified learning rule transformations are applied to the synaptic variables (wgt, dly, tag) and written back to SYNAPSE_MEM.

The list of microcode operations to apply to a given synapse is determined by the sum of three profile fields derived from the synapse's associated POST_TRACE entry, SYNAPSE_MEM format entry, and POST_TRACE entry. Typically all but one of these profile values will be zero. The profile determines a starting address in a 16-entry, 32-bit UCODE_MEM table. A basic pairwise STDP rule can be encoded in a single 32-bit word. More complex rules such as those assigning to both wgt and tag with fairly complicated expressions (*e.g.* for reinforcement learning) may require as many as four or five words to encode. The microcode datapath is iterative and can service two product terms from independent product sequences (P_i, P_j) per iteration.

Support for synaptic delays leads to considerable hidden complexity in the design implementation. When modifying a synapse with a nonzero delay dly, appropriate delay-adjusted presynaptic trace values $x_k[t - \texttt{dly}]$ must be computed for use in the learning rule evaluations. This requires a buffer of $\lceil \max(\texttt{dly})/T_{epoch} \rceil$ historical trace variables to be maintained over multiple PRE_TRACE entries. Presynaptic trace sampling for a particular t_{event} time required by a learning rule then involves dereferencing the appropriate historical x_k snapshot and exponentially decaying it by an adjusted dly-dependent time interval.

4.1.5 Performance Challenges and Optimizations

Varying degrees of parallelism and serialization are applied to sections of the core's pipeline in order to balance the throughput bottlenecks that typical workloads will encounter. Dataflow drawn with finely dotted arrows in Figure 4 indicate parts of the design where single events are expanded into a potentially large number of dependent events. In these areas, we generally parallelize the hardware.

For example, synapses are extracted from SYNAPSE_MEM's 64-bit words with up to four-way parallelism, depending on the synaptic encoding format, and that parallelism is extended to DENDRITE_ACCUM and throughout the synaptic modification pipeline in the learning engine. Conversely, the presynaptic trace state is stored together with SYNAPSE_MEM pointer entries in the SYNAPSE_MAP memory, which then may result in multiple serial accesses per ingress spike. This balances pipeline throughputs for ingress learning-enabled axons when their synaptic fanout factor within the core is on the order of 10:1 while maintaining the best possible area efficiency.

Read-modify-write (RMW) memory accesses, shown as loops around the relevant memories in Figure 4, are fundamental to the neuromorphic computational model and unusually pervasive compared to many other microarchitecture domains. Such loops can introduce significant design challenges, particularly for performance. We manage this challenge with an asynchronous design pattern that encapsulates and distributes the memory's state over a collection of single-ported SRAM banks. The encapsulation wrapper presents a simple dual-ported interface to the environment logic and avoids severely stalling the pipeline except for statistically rare address conflicts.

4.2 Asynchronous Design Methodology

Biological neural networks are fundamentally asynchronous, as reflected by the absence of an explicit synchronization assumption in the continuous time SNN model given in Section 2. Accordingly, asynchronous design methods have long been seen as the appropriate tool for prototyping spiking neural networks in silicon, and most published chips to date use this methodology [11] [13] [14] [15]. Loihi is no different and in fact the asynchronous design methodology developed for Loihi is the most advanced of its kind.

For rapid neuromorphic design prototyping, we extended and improved on an earlier asynchronous design methodology used to develop several generations of commercial Ethernet switches [16]. In this methodology, designs are entered according to a top-down decomposition process using the CAST and CSP languages. Modules in each level of design hierarchy communicate over message-passing channels that are later mapped to a circuit-level implementation, which in this case is a *bundled data* implementation comprising a data payload with request and acknowledge handshaking signals that mediate the propagation of data tokens through the system. Figure 5 shows a template pipeline example. Each pipeline stage has at least one pulse generator, such as the one shown in Figure 6, that implements the two-phase handshake and latch sequencing.

Fine-grain flow control is an important property of asynchronous design that offers several benefits for neuromorphic applications. First, since the activity in SNNs is highly sparse in both space and time, the activity gating that comes automatically with asynchronous flow control eliminates the power that would often be wasted by a continuously running clock. Second, local flow control allows

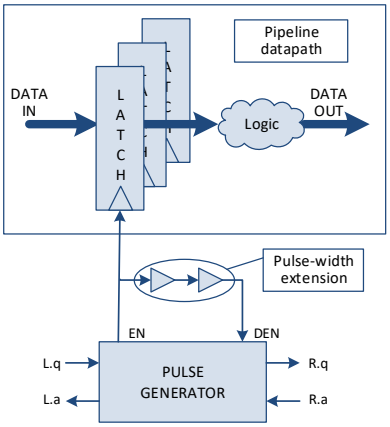


Fig. 5: Bundled data pipeline stage

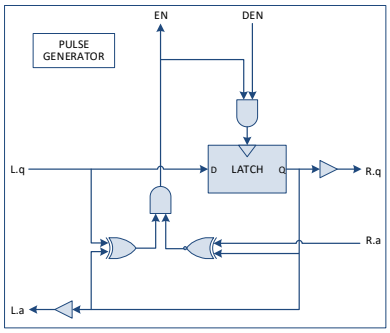


Fig. 6: Bundled Data Pulse Generator Circuit

different modules in the same design to run at their natural microarchitectural frequencies. This properly complements the need for spiking neuron processes to run at a variety of timescales dependent on workload and can significantly simplifiy back-end timing closure. Finally, asynchronous techniques can reduce or eliminate timing margin. In Loihi, the mesh-level barrier synchronization mechanism is the best example of asynchronous handshaking providing a globally significant performance advantage by eliminating needless mesh-wide idle time.

Given a hierarchical design decomposition written in CSP, a pipeline synthesis tool converts the CSP module descriptions to Verilog representations that are compatible with standard EDA tools. The initial Verilog representation supports logic synthesis to both synchronous and asynchronous implementations with full functional equivalence, providing support for synchronous FPGA emulation of the design.

The asynchronous back-end layout flow uses standard tools with an almost fully standard cell library. Here, the asynchronous methodology simplifies the layout closure problem. At every level of layout hierarchy, all timing constraints apply only to neighboring, physically proximate pipeline stages. This greatly facilitates convergent timing closure, especially at the chip level. For example, the Loihi mesh assembles by physical abutment without needing any unique clock distribution layout or timing analysis for different mesh dimensions or core types.

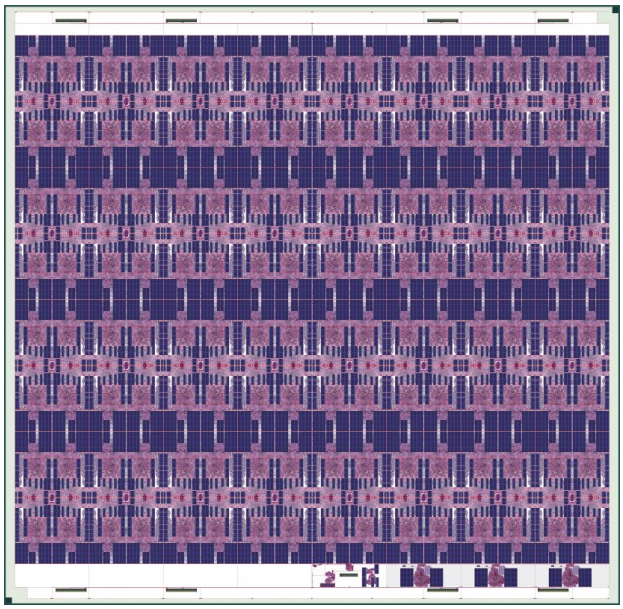


Fig. 7: Loihi chip plot

5 Results

5.1 Silicon Realization

Loihi was fabbed in Intel's 14nm FinFET process. The chip instantiates a total of 2.07 billion transistors and 33 MB of SRAM over its 128 neuromorphic cores and three x86 cores, with a die area of 60 mm^2. The device is functional over a supply voltage range of 0.50V to 1.25V. Table 2 provides a selection of energy and performance measurements from pre-silicon SDF and SPICE simulations, consistent with early post-silicon characterization.

Loihi includes a total of 16MB of synaptic memory. With its densest 1-bit synapse format, this provides a total of 2.1 million unique synaptic variables per mm^2, over three times higher than TrueNorth, the previously most dense SNN chip [11]. This does not consider Loihi's hierarchical network support that can significantly boost its effective synaptic density. On the other hand, Loihi's maximum neuron density of 2,184 per mm^2 is marginally worse than TrueNorth's. Process normalized, this represents a 2× reduction in the design's neuron density, which may be interpreted as the cost of Loihi's greatly expanded feature set, an intentional design choice.

5.2 Algorithmic Results

On an earlier iteration of the Loihi architecture, we quantitatively assessed the efficiency of Spiking LCA to solve LASSO, as described in Section 2.2. We used a 1.67 GHz Atom CPU running both LARS and FISTA [3] numerical solvers as a reference architecture for benchmarking. These solvers are among the best known for this problem. Both chips were fabbed in 14nm technology, were evaluated at a 0.75V supply voltage, and required similar active silicon areas (5 mm^2).

The largest problem we evaluated is a convolutional sparse coding problem on a 52×52 image with a 224-atom dictionary, a patch size of 8×8, and a patch stride of 4

Measured parameter	Value at 0.75V
Cross-sectional spike bandwidth per tile	3.44 Gspike/s
Within-tile spike energy	1.7 pJ
Within-tile spike latency	2.1 ns
Energy per tile hop (E-W / N-S)	3.0 pJ / 4.0 pJ
Latency per tile hop (E-W / N-S)	4.1 ns / 6.5 ns
Energy per synaptic spike op (min)	23.6 pJ
Time per synaptic spike op (max)	3.5 ns
Energy per synaptic update (pairwise STDP)	120 pJ
Time per synaptic update (pairwise STDP)	6.1 ns
Energy per neuron update (active / inactive)	81 pJ / 52 pJ
Time per neuron update (active / inactive)	8.4 ns / 5.3 ns
Mesh-wide barrier sync time (1-32 tiles)	113-465ns

TABLE 2: Loihi pre-silicon performance and energy measurements

(a) Original (b) Reconstruction

Fig. 8: Image reconstruction from the sparse coefficients computed using the Loihi predecessor.

pixels. Loihi's hierarchical connectivity provided a factor of 18 compression in synaptic resources for this network. We solved the sparse coding problem to a solution within 1% of the optimal solution. Figure 8 compares the original and the reconstructed image using the computed sparse coefficients.

Table 3 shows the comparison in computational efficiency between these two architectures, as measured by EDP. It is not surprising to see that the conventional LARS solver can handle problems of small sizes and very sparse solutions quite efficiently. On the other hand, the conventional solvers do not scale well for the large problem and the Loihi predecessor achieves the target objective value with over 5,000 times lower EDP.

Loihi's flexible learning engine allows one to explore and experiment with various learning methods. We have developed and validated the following networks in pre-silicon FPGA emulation with all learning taking place on chip:

- A single-layer classifier using a supervised variant of STDP similar to [4] as the learning method. This network, when trained with local-intensity-change based temporally spike-coded image samples, can achieve 96% accuracy on the MNIST dataset using ten neurons, in line with a reference ANN of the same structure.
- Solving the shortest path problem of a weighted graph. Vertices and edges are represented as neurons and synapses respectively. The algorithm is based on the effects of STDP on a propagating wavefront of spikes [17].
- Solving a one-dimensional, non-Markovian sequential decision making problem. The network learns the decision making policy in response to delayed reward and punishment feedback similar to [18].

No. Unknowns	400	1,700	32,256
No. nonzeros in solutions	≈10	≈30	≈420
Energy	2.58x	8.08x	48.74x
Delay	0.27x	2.76x	118.18x
EDP	0.7x	22.33x	5760x

TABLE 3: Comparison of solving ℓ_1 minimization on Loihi and Atom. Results are expressed as improvement ratios Atom/Loihi. The Atom numbers are chosen from using the more efficient solver between LARS and FISTA.

The algorithmic development and characterization of Loihi is just beginning. These proof-of-concept examples use only a fraction of the resources and features available in the chip. With Loihi now in hand, our focus turns to scaling and further evaluating these networks.

6 Conclusion

Loihi is Intel's fifth and most complex fabricated chip in a family of devices that explore different points in the neuromorphic design space spanning architectural variations, circuit methodologies, and process technology. In some respects, its flexibility may go too far, while in others, not far enough. Further optimizations of the architecture and implementation are planned. The pursuit of commercially viable neuromorphic architectures and algorithms may well end at design points far from what we have described in this paper, but we hope Loihi provides a step in the right direction. We offer it as a vehicle for collaborative exploration with the broader research community.

REFERENCES

[1] P. T. P. Tang, T.-H. Lin, and M. Davies, "Sparse coding by spiking neural networks: Convergence theory and computational results," *arXiv e-prints*, 2017.

[2] S. Shapero, M. Zhu, J. Hasler, and C. Rozell, "Optimal sparse approximation with integrate and fire neurons," *International journal of neural systems*, vol. 24, no. 05, p. 1440001, 2014.

[3] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[4] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[5] T.-H. Lin, "Local Information with Feedback Perturbation Suffices for Dictionary Learning in Neural Circuits," *arXiv e-prints*, 2017.

[6] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers in neuroscience*, vol. 7, p. 272, 2014.

[7] J. Gjorgjieva, C. Clopath, J. Audet, and J.-P. Pfister, "A triplet spike-timingdependent plasticity model generalizes the BienenstockCooperMunro rule to higher-order spatiotemporal correlations," *Proceedings of the National Academy of Sciences*, vol. 108, no. 48, pp. 19 383–19 388, 2011. [Online]. Available: http://www.pnas.org/content/108/48/19383.abstract

[8] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Frontiers in Neuroscience*, vol. 9, p. 141, 2015.

[9] L. Buesing, J. Bill, B. Nessler, and W. Maass, "Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons," *PLoS computational biology*, vol. 7, no. 11, p. e1002211, 2011.

[10] E. M. Izhikevich, "Polychronization: Computation with Spikes," *Neural Computation*, vol. 18, no. 2, pp. 245–282, 2006, pMID: 16378515. [Online]. Available: https://doi.org/10.1162/089976606775093882

[11] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. [Online]. Available: http://science.sciencemag.org/content/345/6197/668

[12] J. s. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and D. J. Friedman, "A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, Sept 2011, pp. 1–4.

[13] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[14] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[15] J. Binas, G. Indiveri, and M. Pfeiffer, "Spiking analog VLSI neuron assemblies as constraint satisfaction problem solvers," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 2094–2097.

[16] M. Davies, A. Lines, J. Dama, A. Gravel, R. Southworth, G. Dimou, and P. Beerel, "A 72-Port 10G Ethernet Switch/Router Using Quasi-Delay-Insensitive Asynchronous Design," in *2014 20th IEEE International Symposium on Asynchronous Circuits and Systems*, May 2014, pp. 103–104.

[17] F. Ponulak and J. J. Hopfield, "Rapid, parallel path planning by propagating wavefronts of spiking neural activity," *Frontiers in Computational Neuroscience*, vol. 7, p. 98, 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3714542/

[18] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural Computation*, vol. 19, no. 6, pp. 1468–1502, 2007.