SANDIA REPORT
SAND2020-10632
Printed September 2020

Sandia National Laboratories

Generic Spiking Architecture (GenSA)

Fred Rothganger
Arun Rodrigues

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@osti.gov
Online ordering:	http://www.osti.gov/scitech

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.gov
Online order:	https://classic.ntis.gov/help/order-methods/

ABSTRACT

Neuromorphic devices are a rapidly growing area of interest in industry, which machines in production by IBM and Intel, among others. These devices promise to reduce size, weight and power (SWaP) costs while increasing resilience and facilitating high-performance computing (HPC).

Each device will favor some set of algorithms, but this relationship has not been thoroughly studied. The field of neuromorphic computing is so new that existing devices were designed with merely estimated use-cases in mind. To better understand the fit between neuromorphic algorithms and machines, a simulated machine can be configured to any point in the design space. This will identify better choices of devices, and perhaps guide the market in new directions.

The design of a generic spiking machine generalizes existing examples while also looking forward to devices that haven't been built yet. Each parameter is specified, along the approach/mechanism by which the relevant component is implemented in the simulator.

CONTENTS

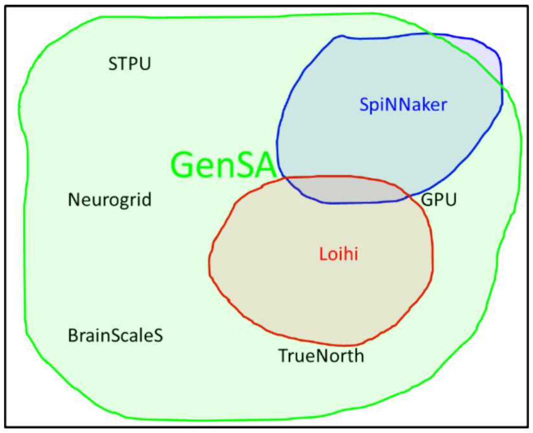
This page left blank

1. INTRODUCTION

Several neural-inspired computing machines have reached the level of maturity to be manufactured and put into service over the last decade. Well known examples include IBM's TrueNorth, University of Manchester's SpiNNaker, and Intel's Loihi. Here we focus on spiking architectures rather than tensor-flow accelerators. A spiking neuromorphic machine typically implements some form of the leaky-integrate-and-fire (LIF) neuron model, combined with communication of "spike" events between units. (A "unit" is one instance of the LIF model.) Spike events typically have an expected delivery time or delay value which is carried as payload. Each combination of source and destination units also has a weight that indicates how much effect the spike has on the downstream unit.

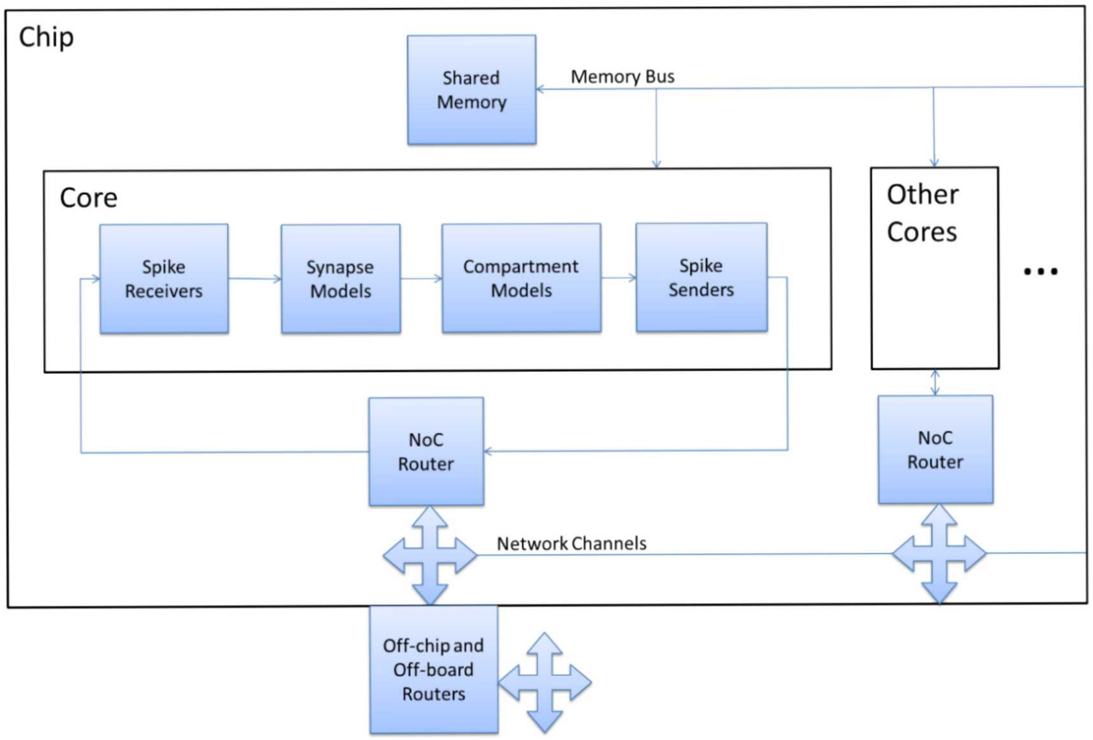
The design of each spiking architecture contains assumptions about the nature of neural computation and what hardware would best support that. Unfortunately, practical neural algorithms have taken longer to mature than the machines themselves. The designs were made in a vacuum, and it is not clear which of them is the best fit for a given application. The goal of the Generic Spiking Architecture (GenSA) is explore the design space of neuromorphic processors, seeking optimal hardware for selected spiking algorithms and the use-cases they represent. This may help in the selection of the best existing architecture for a given problem, or it may lead to the discovery of a new region of design space that should be pursued in future machines.

A "generic" architecture must encompass the whole design space, such that all the existing architectures can be represented. The primary machines we consider are IBM's TrueNorth [5], Intel's Loihi [2], and University of Manchester's SpiNNaker [6]. To a lesser extent, we consider Sandia's own STPU [4], as well as the mixed-signal systems BrainScaleS [3] and Neurogrid [1]. The following is a notional diagram to illustrate the idea of a design space.



GenSA does not need to replicate each architecture exactly. It is sufficient that the generic machine be configured to provide similar performance characteristics under similar inputs (code + data). It is implemented by a simulator rather than actual hardware. This allows the flexibility to enable/disable various resources and functions. We use two software tools to implement GenSA. Simulations are done with Sandia's Structural Simulation Toolkit (SST), a parallel discrete-event simulation framework designed specifically for modeling computer architectures [7]. Algorithms are specified in Neurons to Algorithms (N2A), a tool for modeling large-scale neural systems [8]. N2A contains an optimizing compiler that outputs to multiple backends. The logic to construct an SST configuration for a given set of parameters will be implemented as a GenSA backend.

The diagram below illustrates the major components. A neuromorphic machine must scale to a very large cluster, possibly with enough processing units to represent an entire brain. There may be many levels to the machine organization to allow such an enormous system. Starting at the board level and working downward, there are several chips, each of which contains several cores. A network fabric crosses all the machine levels in order to move event messages between neural units. Each core will process several compartments (typically LIF units). It must accept incoming event messages from the fabric, store them during their delay period and integrate them as they become active. The core must compute the dynamics of each compartment and send event messages out to the network fabric.



The remainder of this document details each component in the form of parameter and implementation considerations. At the end we report on initial implementation efforts and their results.

2. ARCHITECTURE DESCRIPTION

Each section begins with a general discussion of the issues involved with the given portion of the machine and a survey of solutions found in existing hardware. Then it lists the relevant parameters. Parameter values can be interpreted in three ways. They could be the requirements of a given algorithm. Or they could be the upper limits that a particular instance of the GenSA machine can support. Or they could be treated as dimensions of a search space, as described in the "Parameter Search" section at the end.

2.1. Compartment Model

Existing machines can be divided in two categories: restricted versus general. Restricted machines always execute some variant of the leaky-integrate-and-fire (LIF) model. While the available model may have many parameters and interesting extra features, the user does not have the option to choose an entirely different model. A minimal LIF model would be:

```
V ← V + sum (voltage from each spike that comes in)
if V > threshold
    fire spike
    V ← reset voltage
else
    V ← V * decay
```

Remarkably, this simple model is not universal to all current neuromorphic machines. Specifically, the IBM TrueNorth (TN) does not multiply the decay rate, but instead subtracts it. Effectively, this produces linear rather than exponential decay. Perhaps this design choice was motivated by the desire to reduce hardware costs (adders cost O(b), while multipliers cost O(b^2), where b is the number of bits in the operands). IBM suggests that it is possible to get similar dynamics from the TN model by using some of its stochastic features. For the purposes of GenSA, we assume that all restricted machines are capable of basic LIF dynamics, and that any deficiencies in their model result in only a constant factor increase in time or space complexity.

An open question we hope to answer with simulation studies is how much difference "extra" LIF features actually make. In addition to the type of decay in the model, several architectures add random numbers at various points. Typical places include voltage update, spike threshold, and amount of voltage or current delivered by the spike. Here is a more general LIF model that captures some of these options:

```
V ← V + sum (voltage from each spike + per-spike random voltage) + random voltage
if V > threshold and p > random
    fire spike
    V ← reset voltage
else if exponential decay
    V ← V * decay
else if linear decay
    V ← V + decay
```

A general machine is capable of executing an arbitrary dynamical system. In this case, the model is not hard-coded into the machine, but rather represented by a set of general-purpose operations. The SpiNNaker is in this category. The cost for running a LIF model on a general machine may be higher than for a restricted machine, because it does not take advantage of hardware optimizations.

The added costs can be described in terms of the number of digital operations (memory movements, arithmetic) actually done to implement the model.

On the other hand, a general machine can execute models that may solve the overall problem more efficiently. The added complexity of reducing the computation to LIF-only could overwhelm the gains made by executing LIF more efficiently.

In order to take all these factors into account, GenSA compares every machine in terms of digital operations, as if it were a general machine, but also restricts some machines to work only with the LIF model that is actually available. These constraints are encoded into the parameter set by making some values constant that might otherwise be free variables. In some cases, a reduction will be included in the process of converting an algorithm to run on GenSA, adding the extra network structures to implement features missing from the LIF model.

A final consideration is how to compare analog machines with digital. In both cases, there will be a bit equivalence for an operation carried out, and an associated energy cost for the overall operation. For example, an analog adder might have 8-bit resolution, so it can be treated as an 8-bit digital operation. Likewise the energy cost of the operation can be accounted for in both the digital and analog machines and expressed in Joules.

The following are the key parameters for compartments. As noted above, for restricted machines these will be exactly the number of bits and operations involved. For general machines, these can either be the upper limits in general, of the values for a specific model being tested.

Compartments per Core – The maximum number of compartments a core can support.

Compartment FLOPs – The total number of operations done during one update cycle. We count floating-point add as one operation and floating-point multiply as one operation. Likewise, integer add, integer multiply, analog add, and analog multiply all count as one operation. The energy and gate costs for these operations will differ between adds and multiplies, but the purpose of this parameter is to give an abstract work cost. Some of this work may be done in parallel, so FLOPs is not exactly the same as time cost.

Compartment Bits per Variable – The basic size of storage for one value in the model. This is both a numeric resolution and a measure of space cost.

Compartment State Variables – Values that must be stored between update cycles. This is a crude measure of the complexity of the dynamical system being modeled.

Compartment Temp Variables – Values that are computed and used each cycle, then forgotten. Like FLOPs, the total number of variables (state + temp) is a measure of the overall complexity of the model.

Compartment Max Temp Variables – The number of values that must be held simultaneously in temporary memory. This is another measure of the model complexity.

Compartment Bits = Compartment State Variables * Compartment Bits per Variable – The total size of long-term storage per compartment.

Compartment Bits per Core = Compartment Bits * Compartments per Core – Total amount of memory per core allocated to long-term storage of compartment state.

Approach: GenSA will simulate compartment model by directly executing the set of equations that describe it, without any concern for special hardware structure. In particular, no operations are parallelized. The compartments in a single core will execute serially, so the time cost is Compartments per Core * Compartment FLOPS.

2.2. Synapse Model

From a generic architecture perspective, each synapse is a dynamical system, just as each compartment is a dynamical system. However, there are potentially many more synapses, O(n^2) where n is the number of compartments. In an algorithm which reflects the network characteristics of real brains, the number of synapses is more like O(kn) where k is a constant on the order of 10^4.

Because there are many more synapses, most current architectures assume that they are very simple objects and allocate correspondingly fewer resources to each one. It remains to be seen whether this assumption will hold in the long-term as more is learned about neural computation. Loihi introduces non-vacuous synapse dynamics the update the weight based on traces of recent activity.

While compartment models are self-contained, synapse models make changes to state variables outside themselves, specifically state variables in a compartment model which accumulate (acts as a reduction for) the effects of spike events. The LIF compartment model above includes the connection model used by a typical architecture.

A typical synapse on a restricted machine consists of a source->destination mapping, along with a weight and a delay. The details of addressing are covered in the section on network fabric. Storage for these items are included there, while the model of synapse dynamics includes all other values. To accommodate a future with much more sophisticated dynamical models in each synapse, GenSA describes them with essentially the same set of parameters as for compartments.

Synapse FLOPs per Update

Synapse State Variables

Synapse Temp Variables

Synapse Max Temp Variables

Synapse Bits = Synapse State Variables * Synapse Bits per Variable

Synapse Bits per Core = Synapse Bits * Synapse per Core

Approach: GenSA will execute and synapse dynamics in a manner similar to compartments, with similar cost accounting.

2.3. Network Fabric

The means by which event messages move around in a system is arguably the most important part of a neuromorphic machine, more important than the compartment model. The main driver of cost in conventional computers is communication, and the key claim of neuromorphic is an alternate approach to communication. The system diagram shown above has four major blocks dedicated to spike processing: senders, receivers, and two levels of network routing. We will elaborate on these below, after some general description of event communication.

Typically, a connection between two compartments includes a weight, a delay, and a source→destination mapping. The mapping can be expressed in several ways. One approach is source-address routing (SAR, for example SpiNNaker). A source ID, such as the ID of the originating compartment, travels with the network packet. Routing tables are configured to distribute the event to all interested destinations. Another approach is destination-address routing

(DAR, for example TN and Loihi). A target ID, such as the address of a compartment, travels with the packet. On the receiving side there is a mapping between target ID and specific compartments. Notice that a final mapping between ID and compartment happens for both SAR and DAR, so this stage of the process may be considered equivalent. The main distinction is in the method of intermediate routing.

Each event has an associated weight. This can be carried as a payload in the event packet itself, or it can be retrieved from the final mapping table when the event is processed. It is also possible for an event to carry a non-weight payload while using table-lookup for the weight. However, payloads are usually kept to a minimum in neuromorphic systems, not much more than a single float.

Each event has an associated delivery time. This is often expressed as an integer number of delay cycles relative to the current cycle, but can also be a fine-grained time relative to start of simulation. The delivery time is generally carried as payload (Loihi, TN), but could also be retrieved from a table on the receiving side (STPU).

The notion of delivery time implies some method of keeping the entire system synchronized, and several methods appear in practice. The network could operate on a global clock. This clock could be set slow enough the all events are guaranteed to deliver before the next cycle, at least within some travel radius (TN). The system could run on scaled wall-time and communicate asynchronously. The system could also ignore time completely. This approach is viable if all processing takes constant time. The alternative to clocking is to use a barrier signal to maintain sim-time synchronization (TN, SpiNNaker).

In some cases, such as a convolutional network, weights and delays follow a repeating pattern. The memory for redundant weights can be saved by using an indirect table lookup. In this case, the address is mapped to a group of values and a relative offset within the group (Loihi). One consequence of this is that the storage cost can vary significantly with network structure.

Network Chips – Total number of chips in system.

Network Cores per Chip

Network Address – {SAR, DAR}

Network Synchronization Method – {barrier, clock, none}

Network Cycle Time – For "clock" synchronization method, this is wall-time for one cycle. Used mainly for estimating cost, and has nothing to do with the amount of sim-time that transpires per cycle.

Approach: GenSA will simulate each core of the system as requiring execution time proportional to its own load. Thus, the cores will require synchronization at the network level, and GenSA will simulate those mechanisms as well.

2.3.1. Spike Senders

This is the hardware required to queue and inject messages into the network fabric. It is possible for a machine to implement delayed delivery at this stage by buffering events until the target time, then transmitting. A drawback of that approach is the loss of slack time to get a message through a congested network. It makes more sense to transmit immediately and buffer messages on the receiving side. All existing architectures take this approach. We assume that no temporal buffering occurs on the sending side.

The plural term "senders" refers to the fact that a limited number of events may be handled per cycle. One such limit is the rate at which messages can be injected onto the network. The core may buffer events, and some could be lost if too many are generated.

Buffering could be as simple as a single-bit flag on each of a fixed set of event configurations. An architecture that uses DAR must hold target addresses in a table on the sending side. In that case, the number of senders is limited by the capacity of the table. An example of this approach is TN, which has exactly one event configuration per compartment. The only way to get greater fan out is to send the event to an intermediate core which has multiple compartments listening to the same input.

Senders Total – The maximum number of events that may be injected per cycle by one core. Could be unlimited. If truly unlimited, the machine must ensure all messages are delivered before stepping to the next cycle. Otherwise, this parameter should not exceed network capacity.

Senders per Compartment – The maximum number of events that may be configured for a single compartment. Could be unlimited. Implicitly, the quantity configured on one compartment may reduce the number available to another, as they will both be competing for Senders Total. Notice that this parameter has roughly the same meaning as fan out.

Approach: The constraints on senders will be taken into account when the test algorithm is compiled. The GenSA simulator will simply assume these constraints have been followed.

2.3.2. Spike Receivers

Most existing architectures do the work of buffering delayed events on the receiving side. Approaches to buffering include associative memory, event queues, ring buffers and shift registers. A receiver may do something as simple as set a single bit indicating that an event arrived.

An event address (whether SAR or DAR) is often not the same as a specific compartment. Instead, there is a mapping between the address and compartments that subscribe to it. For example, the TN models 256 "axons" which feed a connection matrix with 256 "neurons". The weights for a given event target are stored as elements in the matrix. Thus, TN support 256 event targets which can be mapped in a fully arbitrary way to the compartments in a core. Loihi also has the concept of mapping a single event to multiple compartments.

Spike Receivers – The number of addressable event targets in a core. This may be different than Compartments per Core.

Spike Buffer Delay – The maximum time an event can be delayed, expressed as a number of cycles.

Spike Buffer Depth – The maximum number of events that can be held in the delay buffer.

Spike Buffer Clobber – The maximum number of events that can be distinguished if they have exactly the same address and delivery time. For example, TN sets a bit in a shift register when an event arrives with a given delay. It cannot distinguish any additional events, so its value for this parameter is 1. On the other hand, the STPU architecture keeps an intermediate sum of weights from incoming events. Even though there is a single entry in the ring buffer, each additional event can change the value, so they are effectively distinguished. The STPU value for this parameter is unbounded.

Spike Weight Bits – Numeric resolution of an individual weight. Note that TN uses a configurable 4-entry lookup table for each event target. Perhaps a fair way to account for this is to add up the total bits used to represent weights (256 * 2 + 4 * 8 = 544) and divide that by the number of weights (544 / 256 = 2.125).

Spike Fan Out – Maximum number of compartments than can subscribe to one event target.

Approach: Some of these constraints, such as Spike Fan Out, will be applied when the test algorithm is compiled. The GenSA simulator will construct weight tables and spike buffers based on the parameters. GenSA will switch buffering strategies based on the values of Spike Buffer Depth, Delay and Clobber}. If Spike Buffer Delay is greater than Spike Buffer Depth * Network Cycle Time, then GenSA will use an event queue. Otherwise it will use a ring buffer or shift register. If clobber is 1, then a shift register with 1 bit per delay position is sufficient. If clobber is greater than 1, a ring buffer with log2(Spike Buffer Clobber) bits is needed.

2.3.3. On-Chip Routers

If the communication fabric is viewed as a graph, then routers are the nodes, and point-to-point connections between them are the edges. It is possible for multiple routers to share a bus, in which case they must contend for bandwidth. A bus can be modeled as a re-broadcast node with edges to each router on it, with the bandwidth per link divided down by the number of edges. With this simplification, we treat all communication as point-to-point.

Routers have point-to-point connections with cores. One approach is for each router to be closely associated with a single core. Routers could be placed at the “corners” of cores in a flat 2D mesh (TN, Loihi). In that case they could be associated with either 1 or 4 cores. Routers can also have a larger number of links and be connected in more complex topologies, such as a toroidal mesh with diagonal edges (SpiNNaker).

Each link carries one or more channels. This is a measure of how much parallelism is in the link. Two channels means that two events can move across the link at exactly the same time. Most links will be serial, and only move one event at a time. Having multiple parallel data lines does not necessarily equate to multiple channels, unless each data line can carry a different event from the others. An example of multiple channels would be an optical bus that uses multiple frequency bands to distinguish signal sources [NICE architecture proposed by Murat Okandan, unpublished].

Routing is based on the address embedded in the event packet. There are several approaches to interpreting address. In absolute addressing, every core in the system has a unique ID (SpiNNaker). In relative addressing, the address field contains a number steps to take along each dimension of a grid (TN).

Router Topology – A function that generates the edge list between routers.

Router Links to Routers – The number of direct connections with other routers.

Router Links to Cores – The number of cores a router directly connects.

Router Channels per Link to Router – The number of parallel pathways along a link between two routers. In simple terms, this is the number of events that can be moved at exactly the same moment. Usually links are serial, so this number is 1.

Router Channels per Link to Core – The number of parallel pathways between a given router and a core it is serving. This limits the number of events that can be injected at exactly the same moment. Typically 1.

Router Rate to Core – Maximum number of events that can be routed through a single channel in 1 second. This assumes event packets are constant size, so it is equivalent to bandwidth / size.

Router Rate to Router – Generally the same as Router Rate to Core.

Router Buffer – Maximum number of event packets that can be held in a router at one time. For example, a value of 1 means that if a router is currently transmitting a packet, any incoming packet gets dropped.

Approach: GenSA will explicitly simulate the movement of event packets through the network. Delivery will be absolutely reliable unless incoming packets exceed a router's buffer capacity. At the point, all late-comers will be dropped.

2.3.4. *Off-Chip Routers*

This is the mechanism that assembles neuromorphic devices into larger systems. In a system with relative addressing, it is possible to simply expose the on-chip fabric. The physical package has electrical connections for every link that reaches the edge of the internal mesh. A square package would have pin on all 4 sides which connect directly to their neighbors on the board (TN, Loihi). This approach has limited scaling, since the number of physical connections grows with module size. Eventually, large systems must resort to some form of serial links and routing.

These routers must deal with multiple levels of physical organization: board, rack, and so on. They must extend the addressing scheme. Here we see a possible advantage for SAR. As an event gets routed out of its local subsystem, it can acquire more address bits to indicate its origin. DAR, on the other hand, must explicitly add the extra bits at the point of origin, so the maximum scale of the system must be built into the design.

The SpiNNaker has fairly sophisticated routing, so it is worth a brief discussion. Each chip has one router shared by 18 cores. The router contains a limited-size lookup table. Addresses are matched, partially or fully, then sent along any of 6 links. Since SpiNNaker uses SAR, this allows for added fan-out at each stage of routing.

Router Off-Chip Topology – A function that generates the edge list between routers.

Router Off-Chip Links to Internal Routers – The number of direct connections between off-chip routers and their on-chip counterparts.

Router Off-Chip Links to Internal Cores – The number of internal cores that an off-chip router directly connects.

Router Off-Chip Table Size – For configurable routers, the number of address patterns that can be stored.

Routers Off-Chip – Total number of routers exposed off-chip.

Router Off-Chip Links – The number of links per off-chip router.

Router Off-Chip Channels per Link – The number of parallel pathways along a link between two routers.

Router Off-Chip Buffer

Approach: Same as for on-chip routers.

2.4. Memory

A neuromorphic core may have several levels of memory. If it uses digital processing, there will likely be registers associated with an arithmetic logic unit (ALU). The registers will only hold values temporarily. Model state and parameters will be held in a tightly-coupled memory analogous to L1 in a conventional processor. There may be another level of memory on the chip which serves all the cores on a shared bus, analogous to L2 memory (SpiNNaker). We will borrow the terms L1 and L2 for ease of reference.

For devices that have an L2, we assume the memory runs at the same speed as L1. However, since multiple cores contend for it, the effective speed must be divided by the number of cores. The moment that the total memory utilization crosses the L1 size, speed can slow significantly. This suggests that in such machines, it is better to spread the computation over more cores than to exceed the capacity of L1. The reason not to is if communication costs/complexity are even higher than memory access time.

Memory L1 – Number of bits in L1 associated with one core.

Memory L2 – Number of bits in L2 shared across all cores on a chip.

Memory Bandwidth – Bits that can be read or written in 1 second.

Approach: Restrictions on memory size will be applied at compile time, so the generated GenSA machine will always operate within the specified memory budget. This may include distributing the test algorithm over a greater number of cores or chips in order to fit. GenSA will flag each data item (such as weight tables, event buffers, compartment state variables, etc.) as currently being in L1 or L2. Any items that are in L2 when accessed will be tagged as in L1, and a corresponding time penalty added. Items in L1 will be ejected (marked as in L2) to make room, and a time penalty added for the expected write. GenSA will select a reasonable cache management policy. In addition, the compiler may mark some memory items as locked in L1 so they can't be ejected.

3. PARAMETER SEARCH

The GenSA simulator will account for time, space and energy consumed while running a given algorithm. Some combination of these will act as the objective function in an optimization procedure. In order to search the design space, it would be best if all the parameters were continuous. However, many of the parameters describe above are discrete, and some are choices between only 2 or 3 options. One method to search this space is a form of relaxation where the parameters are treated as continuous, then converted to the nearest valid value when actually generating the simulated machine.

Some of the exemplar machines impose constraints between their parameters as a consequence of optimizations chosen during their design. GenSA will not try to impose these constraints during parameter sweeps. Instead, it will choose a reasonably optimal implementation for each given set of parameters. That may involve constructing somewhat different simulated machines, even along the range of a single parameter.

Due to the compiled nature of the GenSA machine, some parameters can be varied at run time, while others can only be varied by recompiling. A search strategy while know which parameters are in which category and put the run-time parameters in the inner loop.

REFERENCES

[1] Benjamin BV, Gao P, McQuinn E, Choudhary S, Chandrasekaran AR, Bussat JM, Alvarez-Icaza R, Arthur JV, Merolla PA, Boahen K (2014). Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. Proceedings of the IEEE, vol 102, no 5, pp 699-716.

[2] Davies M, Srinivasa N, Lin TH, Chinya G, Joshi P, Lines A, Wild A, Wang H (2018). Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. IEEE Micro.

[3] Grübl A, Billaudelle S, Cramer B, Karasenko V, Schemmel J (2019). Verification and Design Methods for the BrainScaleS Neuromorphic Hardware System. arXiv.

[4] Hill AJ, Donaldson JW, Rothganger FH, Vineyard CM, Follett DR, Follett PL, Smith MR, Verzi SJ, Severa W, Wang F et al. (2017) A Spike-Timing Neuromorphic Architecture. IEEE International Conference on Rebooting Computing (ICRC). pages 1-8.

[5] Merolla PA, Arthur JV, Alvarez-Icaza R, Cassidy AS, Sawada J, Akopyan F, Jackson BL, Imam N, Guo C, Nakamura Y, Brezzo B, Vo I, Esser SK, Appuswamy R, Taba B, Amir A, Flickner MD, Risk WP, Manohar R, Modha DS (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. Science 345 (6197): 668–73.

[6] Painkras E, Plana LA, Garside J, Temple S, Davidson S, Pepper J, Clark D, Patterson C and Furber S (2012). SpiNNaker: A multi-core System-on-Chip for massively-parallel neural net simulation. Proceedings of the IEEE 2012 Custom Integrated Circuits Conference, San Jose, CA.

[7] Rodrigues AF, Voskuilen GR, Hammond SD and Hemmert KS (2016) Structural Simulation Toolkit (SST). SAND Report, Sandia National Lab, Albuquerque, NM.

[8] Rothganger F, Warrender C, Trumbo D, Aimone J (2014) N2A: a computational tool for modeling from neurons to algorithms. Frontiers in Neural Circuits, volume 8.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
John Wagner	1421	jswagne@sandia.gov
Technical Library	01177	libref@sandia.gov