Charles Robert Brunstad
TrueSim: Simulating Routing in TrueNorth
https://github.com/robb17/TrueSim
12/18/19

Interest in the use of artificial neural networks (ANNs), a flexible computational tool with a wide range of important applications, has greatly increased over the past few decades. ANNs are attractive for a number of reasons, including prowess in capturing nonlinearity, high tolerance to noise, generalizability, and parallelizability.[1] The basic unit of computation in these networks is the so-called neuron, a biologically inspired node that uses differentiable, non-linear functions to determine whether or not it should contribute to downstream neurons' inputs.[2] The relationship between an artificial neuron and that found *in vivo* is rather simple: connections between nodes model axons and dendrites, and the nonlinear activation function is thought to approximate the threshold-based activation behavior in the soma.[3] Additionally, both ANNs and the nervous system dynamically alter the strength of connections within their networks to carry out learning.

Even more recently, however, researchers have been turning to neuromorphic computing in hopes of achieving energy efficiencies that approach that of the brain. After all, the brain's ability to power roughly 100 billion parallel computational units with only 20W represents a level of efficiency that is orders of magnitude ahead of our current chip-making technology.[4] To this end, spiking neural networks (SNNs) have been the focus of intense study, and have even been designated the third generation of neural networks.[5] Under this schema, artificial neurons communicate dynamically in time by pulsing out trains of spikes, and information is encoded not simply by an activation, but rather by a number of higher-dimensional metrics, including spike rates and timings. Importantly, because spikes are intermittent, unlike activations in their ANN counterparts, SNNs offer far greater energy efficiencies.[6]

Unfortunately, one is unable to realize significant efficiency gains when performing SNN inferences on traditional sequential cache-based architectures—these systems simply cannot exploit the massively parallel event-based nature of spiking neural networks. In other words, performing neuromorphic computation should logically be conducted on neuromorphic hardware. Enter TrueNorth, an architecture boasting 4096 neurosynaptic cores arranged in a 2D mesh. Capable of performing 58 giga-synaptic operations per second (GSOPS) and achieving an efficiency of 400 GSOPS/W, it represents the culmination of implementing several neuromorphic principles in chip form.[7] True to its purpose, the architecture features heavy deployment of asynchronous circuits, which it connects via an asynchronous-synchronous flow when necessary. Unlike in a typical von Neumann computer, program instructions are not stored linearly in memory; instead, its behavior is determined entirely by the programmed character of neurons and the nature of their connections. These neurons are housed in the chip's many cores—256 at a time—and spikes are transmitted across the intra-chip network when generated.

The present work aims to evaluate the efficiency of the network-on-chip, especially as it might compare with a 3D-mesh on the same architecture. A simulator, TrueSim, was implemented in pure Python to model the flow of spike-representing packets through the network-on-chip, and workload policies—both synthetic and quasi-faithful to that of a true SNN—were devised. Parameters associated with the network, such as bandwidth and speed, were loosely approximated.

In an attempt to remain as faithful as possible to the actual design of TrueNorth, its routing subsystem was rigorously studied. In the TrueNorth architecture, each spike packet contains two parameters—*dx* and *dy*. The former specifies exactly how many cores are to be traversed horizontally, and the latter specifies exactly how many cores are to be traversed vertically. These are relative values: they specify distance with respect to the core from which the packet originates. In traveling to its destination, a packet hops from router to router, and each movement results in an appropriate update to either *dx* or *dy*.

Horizontal routing is carried out before vertical routing. More specifically, eastward hops result in decrementing the *dx* parameter contained within a packet, and westward hops result in incrementing said value. With regards to changes in verticality, northward hops are carried out in concert with decrementing *dy*, whereas southward hops are synchronized with incrementing *dy*. A packet is routed into a given core once *dy* becomes 0. This simplistic routing scheme, while prone to generating areas of high traffic, guarantees freedom from deadlock.

Within a router are four distinct substructures of the forwarding variety and two that perform the task of interfacing a given core with the router. Internally, the forwarding units are highly similar, consisting of an initial merge unit and the circuitry required to pick an appropriate output for a given packet. Merges are performed in a nondeterministic manner—that is, packets are accepted on a first-come first-serve basis. Stalled packets are buffered and forwarded as resources become available. As a result of the routing schema's hierarchical nature, packets need to change course in one of only four ways. Those on an eastward or westward pathway may continue traveling as they were, or they may be routed northward or southward. Packets on a north- or southbound journey, however, cannot "turn the corner," as it must be true for packets on these routes that their *dx* parameter is 0. This simple fact reduces routing complexity greatly and was duly exploited in the designing of TrueNorth. For example, once a packet has arrived in the south-forwarding subsystem, that packet can only be routed further south or to the present core. The west-forwarding subsystem, on the other hand, can inject packets to the west, north, or south (routing said packet locally can be achieved in an indirect manner by first routing to either of the south- or north-forwarding subsystems). This modular design is beneficial also because it enables non-conflicting packets to be handled concurrently without an additional latency overhead. For example, packets traveling eastward and westward do not compete for internal resources. Clearly, the router is optimized for asynchronous control.

Although the distance between a packet's source and destination is highly variable, a packet must arrive at its destination in a timely manner. More specifically, TrueNorth's basic unit of time is a synchronization trigger referred to as a tick, where one tick corresponds to a millisecond, and each packet contains four bits to represent the number of ticks that can elapse before reaching its destination. So, each packet must arrive at its destination within 15 ticks of being created. Ticks are global, synchronous events that occur independently of routing pipelines.

In keeping with the principles discussed above, the TrueSim's design is entirely event-based. Just as portions of the real chip consume minimal power when not in use, simulated core and line objects are not accessed by the CPU except as dictated by packet movement. Instead, TrueSim accesses each packet at each timestep and performs the necessary operation to advance it in time. The top-level algorithm by which it performs the simulation is presented below:

```
for each timestep:
      for each packet:
            decrement delay by one timestep
      to_visit = []
      for each packet, selected randomly:
            if packet.parent_component is a line and delay == 0:
                  if packet.dx > 0:
                        add packet to component_2's wait buffer
                  elif packet.dx < 0:
                        add packet to component_1's wait buffer
                  elif packet.dy > 0:
                        add packet to component_1's wait buffer
                  elif packet.dy < 0:
                        add packet to component_2's wait buffer
                  set packet wait time
            if packet.parent_component is a core and packet.parent_component
            is not in to_visit:
                  append packet.parent_component to to_visit
      for each core in to_visit:
                  for each packet in the send buffer:
                        if packet has been sent through each subsystem:
                              add to directional packet_out register if
                              register is empty
                                          else return to send buffer
                  for each packet in the wait buffer:
                        if packet has been sent through each subsystem:
                              add to directional packet_out register if
                              register is empty
                                          else send to send buffer
      for each core in to_visit:
            for each packet in the packet_out_buffer:
                  offload packet to corresponding wire
                  packet.parent_component.packet_out_buffer[packet] = wire
```

Of note is the random selection of packets from lines. Routing between cores is asynchronous in nature, so a given packet's time of arrival is randomized for an extra level of authenticity. Additionally, the algorithm is agnostic of whatever packet-generation policy may be operant. Packets may be spawned in an attempt to simulate input or in response to the arrival of a spike, but—so long as said packets are included in the set of packets on which the algorithm operates—they are handled appropriately.

The individual simulation objects were implemented so as to reflect the timings of the real system to a fair degree of accuracy. Of course, time flows continuously in the real chip, and our algorithm requires some level of quantization. But optimizing for temporal granularity involves a tradeoff: too fine, and unnecessary simulation overhead is incurred; too coarse, and the simulator fails to accurately capture the behavior of the chip. To address this problem, it was assumed that, if a packet were to traverse the maximum number of cores both vertically and horizontally in low-traffic conditions, said packet would arrive at its destination in 15 ticks, the maximum allowed latency in TrueNorth network-on-chip routing. To meet this bitlength-imposed deadline, a packet therefore averages .11 ticks, or 11 centiticks, per core. Because it was desired to capture events that occur within a core, such as forwarding from one forwarding subsystem to the next within the same core, the centitick was selected as the default unit of time within TrueSim.

A packet's journey from one core to another consists of several steps. It is first queued into a wait buffer, sorted into the appropriate forwarding subsystem, and assigned a delay. On each centitick, delay is decremented until zeroed out. Then, it is routed to one of three constructs: another forwarding unit (in the case of turning a corner), an output buffer, or the local core itself. If the corresponding output wire is available, the packet is injected from the buffer into said wire. Otherwise, it is placed back in a blocked queue. In selecting packets for placement into the output buffer, the blocked queue has priority over the wait queue. Importantly, the delay associated with each wait queue is optimized so that, on average, a packet traversing the longest path possible reaches its destination in approximately 15 ticks, including the additional forwarding subsystem-to-subsystem delay sustained by turning the corner. Achieving this timing would likely have been difficult with increased granularity.

Implementing highly event-based operation required inventive commingling of the simulator's objects. For instance, lines and cores engage in bidirectional handshakes. That is, simulated cores maintain pointers to their attending line objects, and simulated lines maintain pointers to their attending core objects. By configuring these structures in this way, the simulation machinery can accurately guide packets along their routes in a hand-over-hand manner.

In order to model a 3D-mesh topology, TrueNorth's architecture was logically extended into a third dimension. Under this regime, planes of cores are instantiated and stacked on top of one another. Routing was generalized into a third dimension, as well, via the creation of down-

forwarding and up-forwarding subsystems. In keeping with the existing schema, routing in the third dimension was given lowest priority. Interestingly, packets traversing the grid in the east or west direction with *dy* values equal to zero and nonzero *dz* values are required to turn two corners, incurring a significant delay.

TrueSim's perhaps greatest asset is its ability to model a number of workloads—even those that could not be executed on its silicon counterpart. To this end, several policies were implemented. In each, the user provides values corresponding to the number of cores and type of topology. First, statically generated toy workloads were implemented. Under this policy, the user must hard-code packets' sites of generation and destination cores, and these packets are distributed to the appropriate cores at the initial timestep. Additionally, a random policy was implemented such that each neuron generates a packet per tick with a user-specified probability. Packets are assigned random, valid destinations such that their distance traveled in each direction averages out to a user-specified value. Of course, this packet-generating policy aligns relatively poorly with that of a true SNN, but it nonetheless provides some key first-order insights.

To accurately model conditions within TrueNorth, a quasi-faithful SNN workload was devised, and neurons were implemented as a simulation object. The user inputs the number of layers, number of neurons, and the probability with which a neuron will begin a spike train. Neuron objects are instantiated with a random, reasonable spiking frequency and fire at said frequency under one of two conditions: if the neuron is in the input layer, it is randomly activated with the user-specified probability; otherwise, it is activated by an incoming spike train with probability equal to the user-specified probability times a large constant. This approximate behavior was formulated to compensate for the fact that both training and performing inference on the variety of deep SNN for which TrueNorth is best suited would cause simulation to be computationally intractable on the available resources. Neurons are mapped to one of the number of layers specified and assigned a target core in an immediately adjacent downstream layer. Finally, neurons are mapped to cores at random to prevent heavy congestion.

In evaluating the various policies, TrueSim records several metrics. These include the total number of hops made by packets during the simulation, the number of instances in which a packet was delayed by congestion, and the number of stale packets—that is, the number of packets that failed to reach their destination in the required 15 ticks. For the random workload, these metrics were recorded as a function of the probability of each neuron's firing and as a function of average distance between source and destination. For the quasi-faithful workload, these metrics were recorded as a function of the probability of each neuron's firing in addition to core count. The results are reproduced in tables 1 through 4.

Topology	Firing Probability	Total Hops (millions)	Stale Packet Count	Packet Delays (millions)
Mesh	0.00001	0.011	0	0.00002
	0.00003	0.034	0	0.00015
	0.0001	0.12	0	0.0018
	0.0003	0.35	0	0.019
	0.001	1.2	0	0.56
	0.00115	1.3	0	1.2
3D-mesh	0.00001	0.0091	0	0.00001
	0.00003	0.029	0	0.00012
	0.0001	0.096	0	0.0012
	0.0003	0.29	0	0.012
	0.001	0.97	0	0.22
	0.0015	1.5	0	0.94
	0.0017	1.6	0	1.7

Table 1 | Performance comparison **as a function of spiking probability** under the **random packet generation policy**. Simulations were conducted over 3000 centiticks. Core count and average hop length per direction were maintained at 256 and 3, respectively.

Topology	Mean Hop Distance per Direction	Total Hops (millions)	Stale Packet Count	Packet Delays (millions)
Mesh	2	0.22	0	0.0033
	4	0.42	0	0.0062
	6	0.59	0	0.0088
3D-mesh	2	0.30	0	0.0043
	4	0.49	0	0.0064
	6	0.51	0	0.0065

Table 2 | Performance comparison **as a function of average hop distance** under the **random packet generation policy**. Simulations were conducted over 2000 centiticks. Core count was maintained at 1024, and firing probability was pegged at 0.0001.

Topology	Spiking Probability	Total Hops (millions)	Stale Packet Count	Packet Delays (millions)
Mesh	0.00001	0.077	0	0.00061
	0.00003	0.22	0	0.0053
	0.0001	0.68	0	0.082
	0.0003	1.7	0	17
	0.001	3.5	18495	182
3D-mesh	0.00001	0.040	0	0.00040
	0.00003	0.12	0	0.0032
	0.0001	0.36	0	0.032
	0.0003	0.93	0	0.37
	0.001	2.0	0	49

Table 3 | Performance comparison **as a function of spiking probability** under the **quasi-faithful SNN packet generation policy**. Simulations were conducted over 3000 centiticks. Core count was maintained at 256, and neuron count was held invariant at 100000.

Topology	Core Count	Total Hops (millions)	Stale Packet Count	Packet Delays (millions)
Mesh	256	0.65	0	0.208
	1024	1.3	0	0.044
	4096	2.7	0	0.016
3D-mesh	256	0.37	0	0.057
	1024	0.63	0	0.017
	4096	1.0	0	0.0065

Table 4 | Performance comparison **as a function of core count** under the **quasi-faithful SNN packet generation policy**. Simulations were conducted over 3000 centiticks. Neuron count was held invariant at 50000, and spiking probability was pegged at 0.0005.

Results of the aforementioned evaluation runs demonstrate that network saturation is attained at lower loads in the 2D-mesh topology than in the 3D-mesh topology. Under the random packet generation policy, the number of packet delays exceeded the number of total hops in the former topology at around 1.15 million hops, whereas these metrics approached equality in the latter topology at around 1.6 million hops. Moreover, it should be noted that the 3D-mesh topology was running at a core deficit of 40, as the cubic root of 256 is not an integer and is internally rounded down. Additionally, varying the mean distance traveled per hop under the random packet generation policy demonstrated that packets in the 3D-mesh topology were allowed to travel for longer distances than they were in its planar counterpart. Furthermore, simulation results under the quasi-faithful SNN packet generation policy revealed that packets could be transmitted to their destinations not only with fewer delays in a 3D-mesh topology, but also with fewer hops. In this run it was also demonstrated that stale packets are produced at lower loads in the 2D mesh topology than in the 3D mesh topology. Finally, the last test, in which

core counts were varied while other parameters were held constant, suggests that the results obtained are general.

The TrueNorth chip represents a unique, state-of-the-art approach to neuromorphic computing. TrueSim was developed to evaluate the efficacy of its 2D-mesh topology and to highlight the benefits of an alternative topology. It was found that, as far as TrueSim is capable of realistically modeling on-chip routing, a 3-D mesh topology can result in better packet throughput. To be fair, it should be noted that, given current fabrication technology, such a topological change would necessarily entail an increase in average wire length, a factor that was not considered in the development of the simulator. Most importantly, however, TrueSim was implemented so as to enable great programmatic flexibility, and it is anticipated that such considerations could be easily. Further, it is anticipated that additional topologies, such as the flattened butterfly, could be investigated for application within the TrueNorthe ecosystem.

References

[1] Basheer, I.A., Hajmeer, M. N., 2000. Artificial neural networks: fundamentals, computing, design, and application. J. Microbiological Methods, 43 (1), 3-31.

[2] Tavanaei, A., Ghodrati, M., Kheradpisheh, S.R., Masquelier, T., Maida, A., 2019. Deep learning in spiking neural networks. J. arXiv, abs/1804.08150.

[3] Basheer, I.A., Hajmeer, M. N., 2000. Artificial neural networks: fundamentals, computing, design, and application. J. Microbiological Methods, 43 (1), 3-31.

[4] Akopyan, F., Alvarez-Icaza, R., Datta, P., Beakes, M., Manohar, R., Jackson, B. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34 (10), 1537-1557.

[5] Tavanaei, A., Ghodrati, M., Kheradpisheh, S.R., Masquelier, T., Maida, A., 2019. Deep learning in spiking neural networks. J. arXiv, abs/1804.08150.

[6] Tavanaei, A., Ghodrati, M., Kheradpisheh, S.R., Masquelier, T., Maida, A., 2019. Deep learning in spiking neural networks. J. arXiv, abs/1804.08150.

[7] Akopyan, F., Alvarez-Icaza, R., Datta, P., Beakes, M., Manohar, R., Jackson, B. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34 (10), 1537-1557.