Hardware/software Co-design for Neuromorphic Systems

Rajit Manohar, Yale University

Introduction

Transistor technology for electronic computer systems is now at the single digit nanometer scale. This enormous advance through sustained efforts over more than sixty years has resulted in computers that are extremely efficient in terms of the energy per unit of computation. This progress in hardware was arguably driven by the demand for computation, as software systems and digital technology became integrated with more and more of our lives. Despite this progress in device technology, general-purpose microprocessors—the heart of a modern computer—can still be viewed as a "von Neumann" computer with control, storage, arithmetic, and input/output devices. As the demand for computation grows unabated while the scaling of transistor technology slows down, alternate approaches to further reducing the energy per unit of computation are required.

There are several "rules of thumb" that have been well-understood since the 1980s when it comes to designing energy-efficient VLSI architectures. The architecture should use short wires and hence make use of local communication, since communication costs often dominate the energy budget. The architecture should consist of a large collection of components that operate in parallel, and each component should operate at a low voltage to save power while leveraging parallelism for performance. While these properties might be very appealing from a chip design perspective, the performance of such a system can be exploited only if users write parallel software, and even a small serial part of the program quickly limits performance [1]. Hence, the improvements in transistor technology were used to build more and more complex execution engines that could be readily leveraged by existing software, until power constraints limited the introduction of additional complexity and led to the shift to multi-core systems.

There is a much less understood system that exhibits all the attributes required for energy-efficient computation—and that is the human brain. The human brain uses roughly 20W of power and can perform tasks in real-time and on noisy inputs that go well beyond the capabilities of modern computers. Neurons, synapses, dendrites, and axons in the brain operate in an asynchronous, massively parallel fashion. Electrical potentials are in the range of tens of mV, and storage and computation are tightly integrated in individual neurons and synapses [2]. Each component is slow, but the overall system exhibits remarkable performance, routinely outperforming traditional computers in certain problem domains. The system is not explicitly programmed in the traditional sense, but instead learns from experience and repetition. Hence, Biology provides an existence proof of a fine-grained, massively parallel computer that can perform important tasks with orders of magnitude less energy compared to the best conventional computers.

Computers today are being asked to perform tasks that have similar characteristics to those human beings excel at such as driving a car or recognizing faces. Drawing inspiration from Biology, neuromorphic computer systems aim to create electronics inspired by the human brain with the goal of replicating its energy-efficiency and computational ability [3]. This paper discusses neuromorphic system design from a computing perspective—namely, the hardware/software co-design trade-offs in the design of such systems.

Early neuromorphic systems

Since even today we have a limited understanding of how biological systems compute, a better description of a neuromorphic computer is one that provides an electronic implementation of *models* of the brain that are used by computational neuroscientists [4].

Neurons communicate with each other via *spikes*. A spike can be viewed as a digital, continuous time signal. Biological spikes have many distinct features in their waveform [2], but these are typically simplified in an electronic implementation to the point where a spike is simply a signal transition, and the arrival time of the spike is determined by the signal transition time. Many neuromorphic systems designed to scale up to millions of neurons including TrueNorth [5], Loihi [6], Braindrop [7], Neurogrid [8], and the recently announced Loihi2 use continuous time, asynchronous or self-timed digital circuits for spike communication. Clocked spike-based systems discretize time using the global clock signal; in this context, the presence or absence of a spike is determined at each clock edge.

Spike outputs from a neuron travel along axons, and these axons can be connected to the input of thousands of other neurons via synapses that are part of the input dendritic tree for a neuron. There are many different models for neurons and synapses, and for what occurs when an input spike arrives. The simplest model and the one that is easiest to implement is the leaky integrate-and-fire model (LIF). In this model, a neuron is modeled by a membrane potential. If a spike arrives at a particular time, the membrane potential is increased by an amount determined by the weight of the synapse along which the spike arrived. If the membrane potential exceeds a threshold, and output spike is generated, and the potential is reset. Finally, there is a constant leak that causes the membrane potential to decrease with time if no input spike arrives. Models that are biophysically more accurate include the Hodgkin-Huxley model [4], where the state of the neuron and synapse are modeled with multiple state variables that interact via coupled non-linear differential equations. Other models of input spike arrival include using a waveform to represent the effect of the input spike on the neuron input over time rather than implicit synaptic dynamics.

Early neuromorphic systems were designed to mimic sophisticated biological models. Analog circuit models were used in early neuromorphic systems for energy-efficient computation of neuron membrane potentials [9]. Continuous-time operation required asynchronous digital communication. The output of a neuron could be connected to thousands of other neurons. To support this level of fanout in a VLSI technology where traditional digital circuits with an average fanout of between two and four were wiring limited, a packet-switched protocol for communicating spikes called the address-event representation (AER) was introduced [10] (Fig. 1). These circuits were designed in the 1980s and early 1990s when the feature size was 0.5µm or larger. Each circuit was used to show electronic versions of specific biological structures such as the retina [11] and cochlea [12].

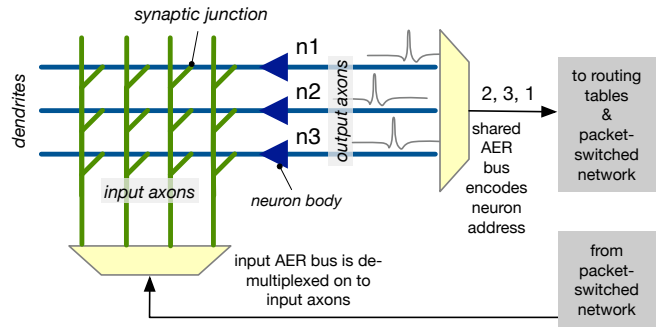

Figure 1. Address-event representation (AER) is used to time-multiplex neuron outputs onto a shared bus. Packets encode the neuron address that spiked, and this is used to determine the set of destination axons to which the spike must be delivered. Incoming spikes are de-multiplexed to the input axons when they are received from the packet-switched network.

Programmable neuromorphic systems

To have a neuromorphic system as a supplement to a conventional computer system requires it to be programmable. This means we need flexibility in the underlying hardware, as well as a mechanism to map computations directly to the neuromorphic substrate. For simplicity, we assume that the "assembly language" for the neuromorphic platform is a graph-based specification of the underlying spike-based computation. This is akin to traditional FPGA-based systems, where a gate-level netlist is mapped to a

programmable hardware substrate consisting of programmable logic and programmable interconnect.

The biggest challenge in the design of large-scale neuromorphic systems is creating an efficient programmable communication infrastructure. Traditional FPGAs are designed to map digital circuits that have an average fan-out of between 2 and 4. Even with this limited typical fanout, the bulk of the die area (over 80%) of an FPGA is devoted to implementing programmable wires using muxes and configuration bits. Neuromorphic systems have typical axon fanouts of over a thousand, and so dedicated point-to-point connectivity akin to an FPGA is prohibitively expensive. Instead, each spike endpoint is given an *address*, and a packet-switched routing fabric is designed that can route a spike to its destination address [13]. While different spike routing architectures have been proposed [13], the most popular approaches include bi-directional mesh networks [5,6] as well as tree-based routing [7,8] (Fig. 2).

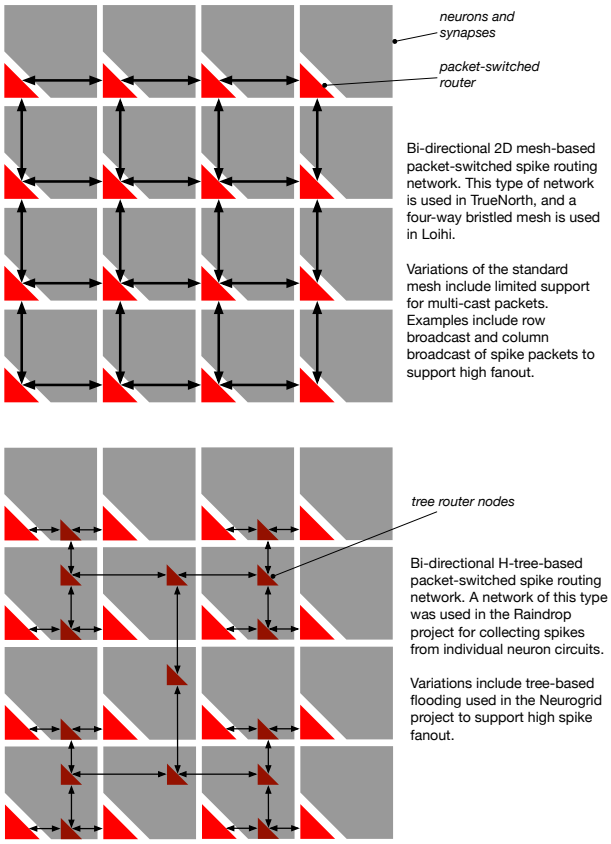

Figure 2. Examples of spike-routing architectures combined with neurons and synapses.

Neuron models

There are many computational models of neurons, and different researchers in the neuroscience field use different models. If we draw an analogy to software, we can view the neuron model has the choice of arithmetic instructions that the hardware should support. In the software domain, we have well-defined requirements, and the choice of arithmetic instructions can be reduced to questions about efficiency and universality. Given a *complete* set of arithmetic primitives, we can emulate any of the others. Hence once we have a complete set, the rest is a question of efficiency: what is the best choice given the cost of hardware implementation v/s the software overhead for a set of benchmarks? Various instruction-set architectures have made different choices, but any of them can be used to implement the same software program by a compilation process.

The same approach is not quite so straightforward in the neuromorphic case, as there is no consensus on the right level of abstraction for biological information processing. However, there are well-understood categories of biological spiking behavior exhibited by a range of neuron models [14]. Even a simple neuron model like LIF can, with some simple enhancements, replicate complex neuron behavior in a local micro-circuit configuration of one, two, or three such simple neurons [15]—the neuromorphic equivalent of technology mapping to a standard cell library.

The crossbar and neuro-synaptic core

Since packet-switched communication networks have significantly higher cost than dedicated wires, a natural question that arises is whether we can reduce the packet traffic in the routing network. The output spike from a neuron must be delivered to many destinations, and each of those destinations is specified by an address. If those addresses are physically near each other on the chip, then we could send a single packet to a group of addresses, reducing the communication cost.

To do so, a collection of synapses and their associated neurons are grouped together into a cluster called a *neurosynaptic* core [16]. Furthermore, this architecture also introduced the now-ubiquitous wiring-efficient crossbar organization of axons and dendrites (Fig. 3). A collection of neurons is grouped together into a single core. Each of those neurons have a set of incoming synapses, organized into the input dendrite for the neuron. The collection of neurons has a set of incoming axons, and each axon can connect to each neuron at a dedicated synapse.

There are two parameters for a neurosynaptic core: the number of neurons (N), and the number of input axons (A). In the case of a crossbar configuration, the number of synapses supported is $A*N$. If a synapse supports a weight of b bits of precision, then the amount of state required is $A*N*b$. The benefit of the crossbar organization is that an N-way fanout of a spike is supported within a core via a simple local wire fanout—about as energy-efficient as it gets. If the If the synapses are fully utilized, then this organization reduces the number of spike packets by a factor of N. Since typical values of N are in the 128 to 1024 range, this is a significant reduction in communication cost. The crossbar architecture also permits experimentation with different synaptic materials that can be deposited as part of back-end-of-line processing in-between the axon and dendrite metal layer steps. This organization was used in TrueNorth [5], as well as most systems that use advanced materials to implement a synaptic device [17].

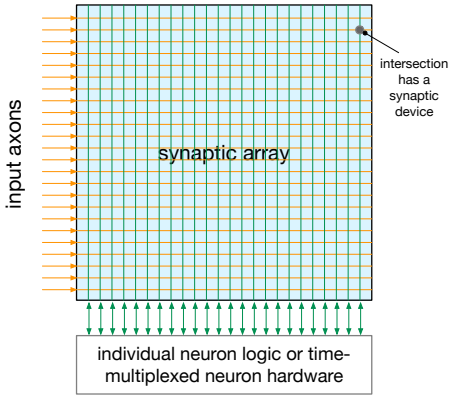

Figure 3. A neurosynaptic core consisting of a dense synaptic array with row/column accesses like a standard memory. The synaptic device could be standard digital logic (e.g. an SRAM) or use a beyond-CMOS material.

To understand the software trade-offs, suppose we had a single neuron in a core ($N=1$). In this case, A would be determined by the maximum fan-in for an individual neuron. To prevent a small number of extremely high fan-in neurons from determining the value of A, we can introduce special *relay neurons* whose sole purpose is to forward their accumulated input to another input axon for a *secondary* neuron. This approach breaks up a large fan-in into a tree of relay neurons, just like high fan-in logic gates are broken down into trees

of smaller fan-in gates. Based on an analysis of typical graphs to be mapped to the architecture, a suitable value of *A* can be selected.

As we increase the value of *N*, we introduce a second software constraint. Any two neurons within an individual neurosynaptic core *share* the same input axons. Hence, we expect two effects: (i) not all neurons will have the same set of input axons—i.e., some of the synapses out of *A*N* will have zero weight, and this number will increase as *N* increases; (ii) the value of *A* itself will increase to accommodate the diversity of inputs, further increasing the number of zero weight synapses in a core. As the number of zero weight synapses in a core increase, the benefits of the crossbar organization diminish.

The mapping of neurons to neurosynaptic cores must take the synaptic constraint imposed by the core organization into account. It is beneficial to group neurons that share input axons into the same core. From the software standpoint, this can be viewed as an input sharing constraint for the core: neurons that share many axons should be grouped into an individual core. Once the maximum number of unique input axons is reached, no additional neurons can be added to a core due to resource constraints.

There is a second challenge with a crossbar organization, and this has to do with array efficiency. Driving an axon in the crossbar requires an address decoder, and reading the dendritic voltage requires readout circuitry. This is akin to an SRAM bank design, with word line drivers and bit line read circuitry. In a small SRAM bank, the overhead of the control circuits can be very large—on par with the area used for the bit-cell array. This intuition holds for the crossbar in a neurosynaptic core as well. Also, for designs with newer materials that require more complex driver and readout logic, the overhead of the access circuitry is even higher.

An alternate architecture for a neurosynaptic core is to replace the crossbar with a more traditional memory organization, where each input axon has a list of weights for each neuron in the core. A direct implementation of this idea would require *A*N*b* bits of state. However, if the connectivity is sparse, then we could represent *S**b weights, where *S* is the total number of synapses in the core. Associated with each weight would be a neuron index, increasing the storage to *S*b*log(N)*, which would be addressed by a *log(S)* bit address. Finally, an additional table would be needed to indicate the start address of the synapses for each axon, another *A*log(S)* bit table. The Loihi architecture [6] uses an addressing mechanism of this type.

Finally, a third choice is to not permit the synaptic weights to be fully specified by the user, but instead provide support for a diversity of weights that can be selected in some fashion. In a digital implementation, this can be via a pseudo-random number generator that uses the axon and neuron indices to select a weight. In an analog implementation, this can be via a programmable analog network [7].

The number of bits needed for each weight, as well as the number of bits needed for the internal membrane potential for the neuron are both parameters that are selected by a co-design process. Typical choices are in the 8-bit range for weights (especially if learning is supported), and 16-bit range for the neuron state. The synaptic bits are much more expensive, since the number of synapses is many orders of magnitude larger than the number of neurons. If a synapse weight is not adaptive and a software analysis shows that a small number of unique weights is sufficient per core, then the synaptic storage requirements can be significantly reduced by storing a weight index rather than the actual weight value per synapse [5].

Learning

One of the key characteristics of biological networks is that they learn and adapt continuously. This adaptation is implemented by changing the weights of the synapses. The most used learning rule is some variation of spike timing-dependent plasticity [18], where the temporal proximity of an input spike to a synapse and the output spike for the associated neuron is used to update the weight of a synapse. In a crossbar organization, this is achieved by applying the appropriate driving signals to the axon and dendrite to update the weight stored at their intersection. In a memory-oriented architecture, the coincidence in spike timing is recorded in a buffer and used to periodically update the weights stored in memory [6].

We don't fully understand how biological systems achieve robust long-term memory while supporting continuous weight adaptation. Hence, neuromorphic systems often permit the learning mechanism to be disabled (or don't support it at all) so that they can be used in a "pre-trained" mode.

Temporal fidelity and determinism

Information in spiking neural networks is carried in spike timing. In biological systems, there are direct, physical connections between the source of a spike and its targets. As discussed earlier, the limited wiring in VLSI forces us to use time-multiplexed wires and a packet-switched routing approach to delivering spikes to their destination. This time-multiplexing can change the spike timing, thereby also impacting the computation.

There are two approaches to this problem. The first, adopted by systems like Neurogrid [8] and Braindrop [7] is to ensure that the system is robust to this perturbation in spike timing. Biological systems have neuron spike rates that are roughly in the 1-10 Hz range, and typical computational neuroscience software models assume a temporal fidelity of 0.1ms. If the neuromorphic system is operating in real-time, then as long as the temporal uncertainty is much less than 0.1ms, the computation is likely to be robust. Since CMOS digital circuits can operate in the GHz throughput regime, even a thousand packet stall would introduce less than a 1µs uncertainly in a well-pipelined routing network. This means that a fast enough routing network would result in low temporal uncertainty.

To examine the implications of this approach, consider a network architecture with *N* total neurons and a bisection bandwidth of *B* spikes/sec, where a router link has a mean occupancy of *o* units of time (Fig. 4). Hence, if a packet must wait for another one, it must wait *o* units of time. If the bisection cuts *L* links, $B = L/o$. If *N*/2 neurons spike at *R*=10 Hz, a traditional network analysis would require that $B \geq R*N/2$. However, suppose we introduce the software constraint that the uncertainty is 0.1ms. If we assume the packets are uniformly spread across the *L* links, then a packet may have to wait for (R**N*/2*L*-1) other packets, i.e., for *o**(R**N*/2*L*-1) seconds, which must be less than 0.1ms = 1/(1000*R). Substituting $B=L/o$, we get $B \geq 1000*(R*R*N/2-L)$—much larger than the traditional network bisection bandwidth requirement of R*N/2 [19].

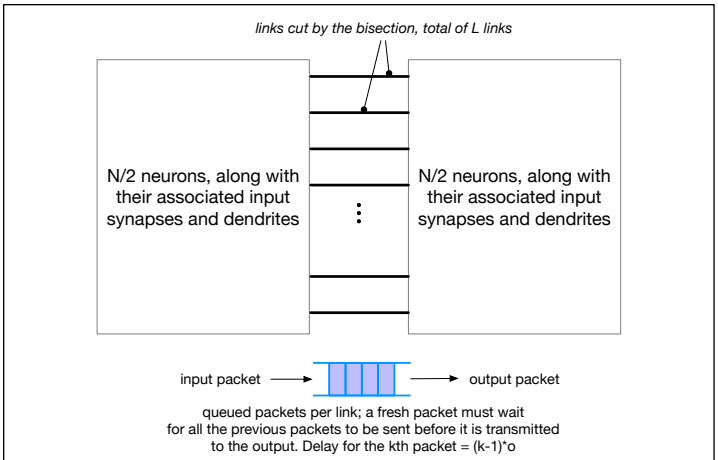

Figure 4. If we partition the architecture into two halves, then the network bisection bandwidth and per link throughput determines how long a packet might have to wait--which introduces timing variability.

Another alternative is to explicitly track time in the hardware. This approach is taken by both TrueNorth and Loihi; TrueNorth has an external global strobe signal that is used to advance time. Loihi uses a global barrier synchronization through its packet routing network to

advance time. In this approach, the temporal fidelity of the neuromorphic system is determined by the discrete timestep that is built into the hardware.

Routing tables

When a neuron in a neurosynaptic core produces an output spike, the spike is delivered to a set of destination cores at specific axon addresses. This routing information is stored in a per-neuron routing table in each core. Since most architectures have a finite size for this routing table, this imposes another software constraint on the neuron fanout and connectivity. A more flexible routing architecture requires a larger routing table. The size of the output routing table coupled with the synapses per input axon in the core determine the total fanout an individual neuron can support. This is coupled with the constraints of the neurosynaptic core itself. Fig. 5 shows two different design points for the overall neurosynaptic connectivity selected by the TrueNorth and the Loihi projects.

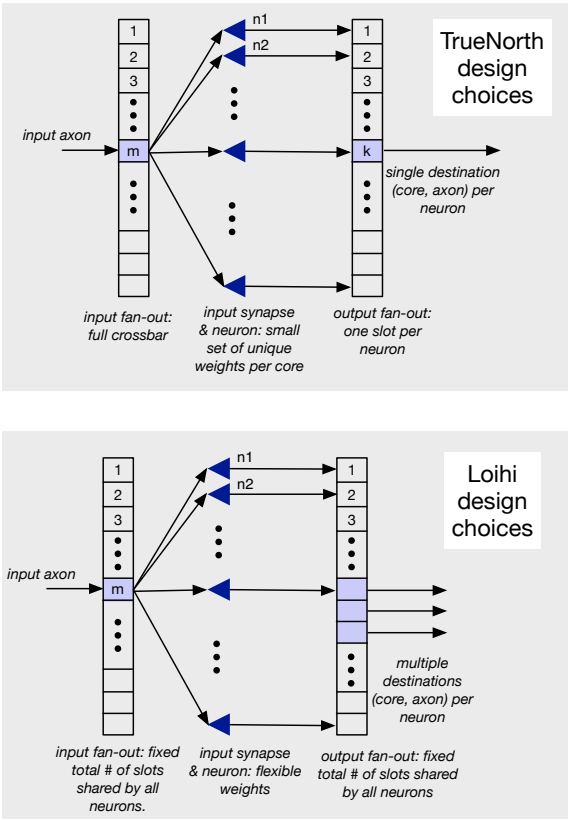

Figure 5. Different choices made by two different large-scale neuromorphic architectures: TrueNorth and Loihi.

Once neurons and synapses have been clustered, the clusters must be placed on-chip to minimize the distance a spike travels, thereby minimizing routing energy. This is like a VLSI or FPGA placement problem, where the wiring length corresponds to the distance traveled by a spike. Applying placement-style optimizations in the software flow to optimize the physical location of neurosynaptic cores can reduce the overall energy cost by a significant amount. Routing tables can be populated once this placement information is known.

Information representation

The biggest impact on energy in large-scale programmable neuromorphic systems is in the cost of spike routing. The best way to reduce the energy cost is to reduce the total number of spikes generated by the network. This must be tackled at the algorithmic level, when mapping a problem into a spiking neural network architecture.

The most direct way weights can be selected for a spiking neural network is to convert a more traditional artificial neural network into a spiking representation (e.g. [20]). Since artificial neural networks have well-established algorithms for training, this approach leverages significant effort in tools and methodology in computing the weights for artificial neural networks and translates it to (non-learning) neuromorphic systems. In this approach, the real valued numbers used in traditional neural networks correspond to the average spike rate. If we make the simplifying assumption that spikes are Poisson (a common assumption in computational neuroscience) and use a slotted model (consistent with the temporal schemes in Loihi and TrueNorth), and the average spike rate corresponds to a real number, then we can determine the amount of time that we must wait in computing the average to get an accurate estimate of the real number to a given precision. Note that since we are computing an estimate of a stochastic process, there is always some probability of error, and hence we also need to select a confidence threshold. Unfortunately, the number of spikes needed grows very quickly; for example, if we wish to correctly represent a real number with Poisson spikes 90% of the time, then we need to wait for 20 time slots to get a one-bit precise representation, and 2582 slots to get a four-bit precise representation. A summary of the number of slots for different choices of precision and confidence are shown in Table 1.

Table 1. When using Poisson spikes, the number of time slots needed to obtain an accurate estimate of a rate with high confidence grows quicky with the precision required.

Precision (bits)	Number of slots needed for different confidence values		
	c = 95%	c = 90%	c = 75%
1	28	20	8
2	176	126	56
3	848	592	288
4	3670	2582	1248
5	15211	10731	5227

In other words, for any computation where value precision is important, the spike cost is extremely high. We can counteract this by permitting spikes to carry values (e.g. in the recently announced Loihi2), and by using *thinning* to scale down the number of spikes sent over the network [7]. In specific situations, we can use other spike representations that are more efficient—for example, information can be represented by the time difference between two spikes rather than a spike rate [4]. The challenge of changing spike encoding schemes is to design the appropriate network of neurons given the physical neurons available in the hardware implementation.

Another commonly used approach is to use populations of neurons to represent a value. When input spikes are delivered to the population, different neurons spike at different times. A winner-take-all circuit is used to select the neuron that spiked first and use the neuron index to represent the value encoded by the population. The combination of populations of neurons and rates has been used extensively in the neural engineering framework [21], which permits the synthesis of neural circuits from a dynamical systems formulation of the computation.

Summary

We have discussed numerous hardware/software co-design issues in neuromorphic systems. The appropriate choices for each of these issues is a strong function of the actual computations being mapped to the neuromorphic substrate. To make progress on this front, what is needed is a collection of accepted and open benchmark suite that is designed to showcase the efficiency of spike-based neuromorphic architectures. We are working toward this goal and developing a flexible neuromorphic architecture that can be used as a reference design by the community.

References:

[1] G.M. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities”, Proc. AFIPS, 1967.

[2] E. Kandel *et al.*, “Principles of Neural Science”, McGraw Hill, 2021.

[3] C.A. Mead, “Neuromorphic electronic systems”, PIEEE, Oct 1990.

[4] P. Dayan and L. Abbott, “Theoretical Neuroscience”, MIT Press, 2005.

[5] P. Merolla *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface”, *Science*, 2014.

[6] M. Davies *et al.,* “Loihi: A neuromorphic manycore processor with on-chip learning”, IEEE Micro, 2018.

[7] A. Neckar *et al.,* “Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model”, PIEEE, Jan 2019.

[8] B.V. Benjamin *et al.*, “Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations”, PIEEE, 2014.

[9] M. Mahowald and R. Douglas, “A silicon neuron”, Nature, 1991.

[10] J. Lazzaro *et al.,* “Silicon auditory processors as computer peripherals”, IEEE Trans. Neural Nets, May 1993.

[11] M. Mahowald and C.A. Mead, “The silicon retina”, Scientific American, 1991.

[12] R.F. Lyon and C.A. Mead, “An analog electronic cochlea,” IEEE Trans. Acoust. Speech and Signal Processing, 1988.

[13] Shih-Chii Liu *et al*., “Event Based Neuromorphic Systems”, Wiley, 2015.

[14] E.M. Izhikevich, “Which model to use for cortical spiking neurons?”, IEEE Trans. Neural Networks, 2004.

[15] A.S. Cassidy *et al.*, “Cognitive Computing Build Block: A versatile and efficient digital neuron model for neurosynaptic cores”, IJCNN 2013.

[16] P. Merolla *et al.*, “A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm”, CICC 2011.

[17] S. Yu, “Neuro-inspired computing with emerging nonvolatile memorys”, PIEEE, Feb 2018.

[18] G. Bi and M. Poo, “Activity induced synaptic modifications in hippocampal culture: Dependence on spike timing, synaptic strength and cell type”, J. Neuroscience, 1999.

[19] S. Moradi *et al.*, “The impact of on-chip communication on memory technologies for neuromorphic systems”, J. Physics D, Oct 2018.

[20] P. Diehl, “Performant spiking systems”, Ph.D. thesis, ETHZ 2016.

[21] C. Eliasmith *et al.*, “Neural engineering: computation, representation, and dynamics in neurobiological systems”, MIT Press, 2003.