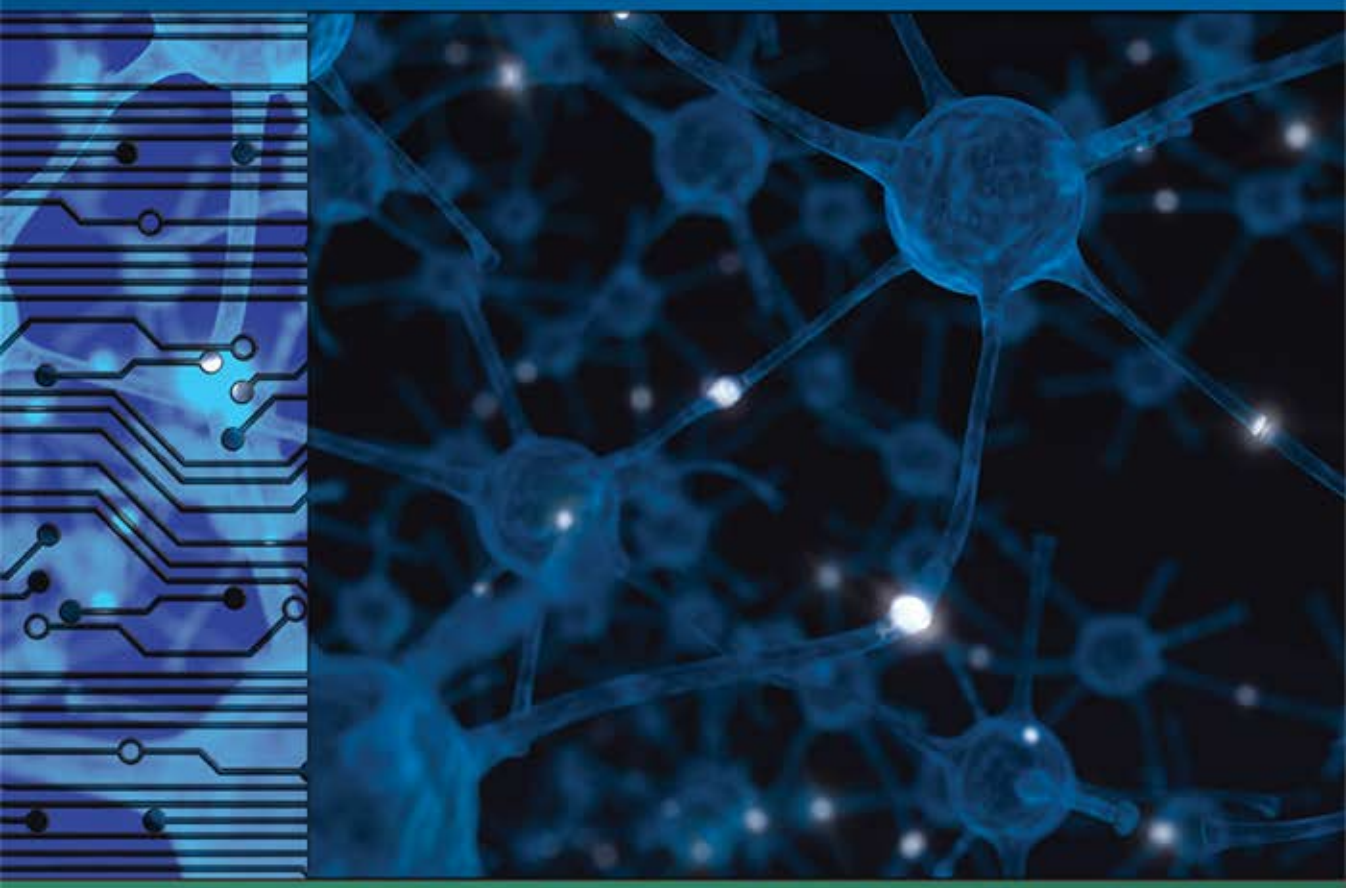
Neuromorphic Computing

Architectures, Models, and Applications

A Beyond-CMOS Approach to Future Computing



DOE Workshop Report

June 29–July 1, 2016
Oak Ridge, TN

U.S. DEPARTMENT OF ENERGY
Office of Science

OAK RIDGE National Laboratory

Neuromorphic Computing Architectures, Models, and Applications

A Beyond-CMOS Approach to Future Computing

June 29–July 1, 2016

Oak Ridge National Laboratory
Oak Ridge, Tennessee

Organizing Committee

Thomas Potok, Oak Ridge National Laboratory
Catherine Schuman, Oak Ridge National Laboratory
Robert Patton, Oak Ridge National Laboratory
Todd Hylton, Brain Corporation
Hai Li, University of Pittsburgh
Robinson Pino, U.S. Department of Energy

Contents

List of Figures

Neuromorphic Computing

Architectures, Models, and Applications

A Beyond-CMOS Approach to Future Computing

Executive Summary

The White House[1] and Department of Energy[2] have been instrumental in driving the development of a neuromorphic computing program to help the United States continue its lead in basic research into (1) Beyond Exascale—high performance computing beyond Moore's Law and von Neumann architectures, (2) Scientific Discovery—new paradigms for understanding increasingly large and complex scientific data, and (3) Emerging Architectures—assessing the potential of neuromorphic and quantum architectures.

Neuromorphic computing spans a broad range of scientific disciplines from materials science to devices, to computer science, to neuroscience, all of which are required to solve the neuromorphic computing grand challenge. In our workshop we focus on the computer science aspects, specifically from a neuromorphic device through an application. Neuromorphic devices present a very different paradigm to the computer science community from traditional von Neumann architectures, which raises six major questions about building a neuromorphic application from the device level. We used these fundamental questions to organize the workshop program and to direct the workshop panels and discussions. From the white papers, presentations, panels, and discussions, there emerged several recommendations on how to proceed.

(1) **Architecture Building Blocks**—What are the simplest computational building blocks? What should the neuromorphic architecture look like and how should we evaluate and compare different architectures? Current CMOS-based devices and emerging devices (e.g., memristor, spintronic, magnetic, etc.) and their associated algorithms could potentially emulate the functionality of small regions of neuroscience-inspired neurons and synapses ranging from great details to very simple abstracts. Just as biological neural systems are composed of networks of neurons and synapses that learn and evolve integrated, interdependent responses to their environments, so must the computational building blocks of neuromorphic computing systems learn and evolve network organization to address the problems presented to them. Scalability and generalization need to be provided by neuromorphic architecture. Potentially, the effective data representation, communication, and information storage and processing could be the key considerations.

[1] The White House has announced a Nanotechnology-Inspired Grand Challenge for Future Computing, which seeks to create a new type of computer that can "proactively interpret and learn from data, solve unfamiliar problems using what it has learned, and operate with the energy efficiency of the human brain." This grand challenge will leverage three other national research and development initiatives: National Nanotechnology Initiative (NNI), National Strategic Computing Initiative (NSCI), and the BRAIN initiative.

[2] In October 2015 the DOE Office of Science conducted a roundtable on neuromorphic computing with leading computer scientists, device engineers, and materials scientists that emphasized the importance of an interdisciplinary approach to neuromorphic computing. http://science.energy.gov/~/media/ascr/pdf/programdocuments/docs/Neuromorphic-Computing-Report_FNLBLP.pdf

Research Challenge: Invest and guide effective collaborations and connections between theory of computation, neuroscience, and nonlinear device physics with machine learning and large-scale simulations to discover the new materials and devices, the building blocks, and therefore novel architecture of a practical neuromorphic computing system.

(2) **Configurations**—*What should we expect from reconfigurable devices?* Traditionally, devices for the part most have been static (with gradual evolutionary modifications to architecture and materials, primarily based on CMOS), and software development dependent on the system architecture, instruction set, and software stack has not changed significantly. A reconfigurable device-enabled circuit architecture requires a tight connection between the hardware and software, blurring their boundary.

Research Challenge: Computational models need to be able to run at extreme scales (+exaflops) and leverage the performance of fully reconfigurable hardware that may be analog in nature and computing operations that are highly concurrent, as well as account for nonlinear behavior, energy, and physical time-dependent plasticity.

(3) **Learning Models**—*How is the system trained/programmed?* Computing in general will need to move away from the stored programming model to a more dynamic, event-driven learning model that requires a broad understanding of theory behind learning and how best to apply it to a neuromorphic system.

Research Challenge: Understand and apply advanced and emerging theoretical concepts from neuroscience, biology, physics, engineering, and artificial intelligence and the overall relationship to learning, understanding, and discovery to build models that will accelerate scientific progress.

(4) **Development System**—*What application development environment is needed?* A neuromorphic system must be easy to teach and easy to apply to a broad set of tasks, and there should be a suitable research community and investment to do so.

Research Challenge: Develop system software, algorithms, and applications to program/teach/train.

(5) **Applications**—*How can we best study and demonstrate application suitability?* The type of applications that seem best suited for neuromorphic systems are yet to be well defined, but complex spatio-temporal problems that are not effectively addressed using traditional computing are a potentially large class of applications.

Research Challenge: Connect theoretical formalisms, architectures, and development systems with application developers in areas that are poorly served by existing computing technologies.

(6) **Hardware Development**—*How do we build and/or integrate the necessary computing hardware?* As compared to conventional computing systems, neuromorphic computing systems and algorithms need higher densities of typically lower precision memories operating at lower frequencies. Also, multistate/analog memories offer the potential to support learning and adaptation in an efficient and natural manner. Without efficient hardware implementations that leverage new materials and devices, the real growth of neuromorphic applications will be substantially hindered.

Research Challenge: Enable a national fabrication capability to support the development and technology transition of neuromorphic materials, devices, and circuitry that can be integrated with state-of-the-art CMOS into complete and functional computing systems.

We propose that DOE Office of Science, Advanced Scientific Computing Research (ASCR) program office, in particular, develop and execute a program in neuromorphic computing focused on DOE's priorities of leading innovation to deliver a beyond-exascale vision and strategy, revolutionize scientific discovery, and answer challenging questions about the future of computing. The program should be based on exploring neuromorphic computing from fundamental applied physics and materials science, device, circuitry, componentry, and hardware architecture through an application level with strong ties to new research in materials, devices, and biology. High performance computing–enabled simulations should be central to ensuring the success of leading potential prototyping, testing, and evaluation of the building blocks, configurations, learning models, development systems, hardware, and applications for future neuromorphic computers.

I. Introduction and Motivation

In October 2015, the White House Office of Science and Technology Policy released A Nanotechnology-Inspired Grand Challenge for Future Computing, which states the following:

"Create a new type of computer that can proactively interpret and learn from data, solve unfamiliar problems using what it has learned, and operate with the energy efficiency of the human brain."

As a result, various federal agencies (DOE, NSF, DOD, NIST, and IC) collaborated to deliver "A Federal Vision for Future Computing: A Nanotechnology-Inspired Grand Challenge" white paper presenting a collective vision with respect to the emerging and innovative solutions needed to realize the Nanotechnology-Inspired Grand Challenge for Future Computing. The white paper describes the technical priorities shared by multiple federal agencies, highlights the challenges and opportunities associated with these priorities, and presents a guiding vision for the R&D needed to achieve key near-, mid-, and long-term technical goals.

This challenge falls in line and is very synergistic with the goals and vision of the neuromorphic computing community, which is to build an intelligent, energy efficient system, where the inspiration and technological baseline for how to design and build such a device comes from our recent progress in understanding of new and exciting material physics, machine intelligence and understanding, biology, and the human brain as an important example.

Investment in current neuromorphic computing projects has come from a variety of sources, including industry, foreign governments (e.g., the European Union's Human Brain Projects), and other government agencies (e.g., DARPA's SyNAPSE, Physical Intelligence, UPSIDE, and other related programs). However, DOE within its mission should make neuromorphic computing a priority for following important reasons:

1. The likelihood of fundamental scientific breakthroughs is real and driven by the quest for neuromorphic computing and its ultimate realization. Fields that may be impacted include neuroscience, machine intelligence, and materials science.

2. The commercial sector may not invest in the required high-risk/payoff research of emerging technologies due to the long lead times for practical and effective product development and marketing.

3. Government applications for the most part are different from commercial applications; therefore, government needs will not be met if they rely on technology derived from commercial products. Moreover, DOE's applications in particular are also fundamentally different from other government agency applications.

4. The long-term economic return of government investment in neuromorphic computing will likely dwarf other investments that the government might make.

5. The government's long history of successful investment in computing technology (probably the most valuable investment in history) is a proven case study that is relevant to the opportunity in neuromorphic computing.

6. The massive, ongoing accumulation of data everywhere is an untapped source of wealth and well-being for the nation.

What is neuromorphic computing?

Neuromorphic computing combines computing fields such as machine learning and artificial intelligence with cutting-edge hardware development and materials science, as well as ideas from neuroscience. In its original incarnation, "neuromorphic" was used to refer to custom devices/chips that included analog components and mimicked biological neural activity [Mead1990]. Today, neuromorphic computing has broadened to include a wide variety of software and hardware components, as well as materials science, neuroscience, and computational neuroscience research. To accommodate the expansion of the field, we propose the following definition to describe the current state of neuromorphic computing:

Neural-inspired systems for non–von Neumann computational architectures

In most instances, however, neuromorphic computing systems refer to devices with the following properties:

- two basic components: neurons and synapses,
- co-located memory and computation,
- simple communication between components, and
- learning in the components.

Additional characteristics that some (though not all) neuromorphic systems include are

- nonlinear dynamics,
- high fan-in/fan-out components,
- spiking behavior,
- the ability to adapt and learn through plasticity of both parameters, events, and structure,
- robustness, and
- the ability to handle noisy or incomplete input.

Neuromorphic systems also have tended to emphasize temporal interactions; the operation of these systems tend to be event driven. Several properties of neuromorphic systems (including event-driven behavior) allow for low-power implementations, even in digital systems. The wide variety of characteristics of neuromorphic systems indicates that there are a large number of design choices that must be addressed by the community with input from neurophysiologists, computational neuroscientists, biologists, computer scientists, device engineers, circuit designers, and material scientists.

Why now?

In 1978, Backus described the von Neumann bottleneck [Backus1978]:

> Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck. Not only is this tube a literal bottleneck for the data traffic of a problem, but, more importantly, it is an intellectual bottleneck that has kept us tied to word- at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand. Thus programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck, and much of that traffic concerns not significant data itself but where to find it.

In the von Neumann architecture, memory and computation are separated by a bus, and both the data for the program at hand as well as the program itself has to be transferred from memory to a central processing unit (CPU). As CPUs have grown faster, memory access and transfer speeds have not improved at the same scale [Hennessy2011]. Moreover, even CPU performance increases are slowing, as Moore's law, which states that the number of transistors on a chip doubles roughly every 2 years, is beginning to slow (if not plateau). Though there is some argument as to whether Moore's law has actually come to an end, there is a consensus that Dennard scaling, which says that as transistors get smaller that power density stays constant, ended around 2004 [Shalf2015]. As a consequence, energy consumption on chips has increased as we continue to add transistors.

While we are simultaneously experiencing issues associated with the von Neumann bottleneck, the computation-memory gap, the plateau of Moore's law, and the end of Dennard scaling, we are gathering data in greater quantities than ever before. Data comes in a variety of forms and is gathered in vast quantities through a plethora of mechanisms, including sensors in real environments, by companies, organizations, and governments and from scientific instruments or simulations. Much of this data sits idle in storage, is summarized for a researcher using statistical techniques, or is thrown away completely because current computing resources and associated algorithms cannot handle the scale of data that is being gathered. Moreover, beyond intelligent data analysis needs, as computing has developed, the types of problems we as users want computers to solve have expanded. In particular, we are expecting more and more intelligent behavior from our systems.

These issues and others have spurred the development of non–von Neumann architectures. In particular, the goal of pursuing new architectures is not to find a replacement for the traditional von Neumann paradigm but to find architectures and devices that can complement the existing paradigm and help to address some of its weaknesses. Neuromorphic architectures are one of the proposed complement architectures for several reasons:

1. Co-located memory and computation, as well as simple communication between components, can provide a reduction in communication costs.

2. Neuromorphic architectures often result in lower power consumption (which can be a result of analog or mixed analog-digital devices or due to the event-driven nature of the systems).

3. Common data analysis techniques, such as neural networks, have natural implementations on neuromorphic devices and thus are applicable to many "big data" problems.

4. By building the architecture using brain-inspired components, there is potential that a type of intelligent behavior will emerge.[3]

Overall, neuromorphic computing offers the potential for enormous increases in computational efficiency as compared to existing architecture in domains like big data analysis, sensory fusion and processing, real-world/real-time controls (e.g., robots), cyber security, etc. Without neuromorphic computing as part of the future landscape of computing, these applications will be very poorly served.

The goal of this report is to discuss some of the major open research questions associated with the ***computing*** aspect of neuromorphic computing and to identify a roadmap of efforts that can be made by the computing community to address those research questions. By **computing** we mean those aspects of neuromorphic computing having to do with architecture, software, and applications, as opposed to more device- and materials-related aspects of neuromorphic computing. This workshop was preceded by a DOE-convened roundtable on Neuromorphic Computing: From Materials to Systems Architecture, held in October 2015. The roundtable, which included computer scientists, device engineers, and materials scientists, was co-sponsored by the Office of Science's ASCR and BES offices and emphasized the importance of an interdisciplinary approach to neuromorphic computing. Though this report is written specifically from the computing perspective, the importance of collaboration with device engineers, circuit designers, neuroscientists, and materials scientists in addressing these major research questions is emphasized.

II. Current State of Neuromorphic Computing Research

Because of the broad definition of neuromorphic computing and since the community spans a large number of fields (neuroscience, computer science, engineering, and materials science), it can be difficult to capture a full picture of the current state of neuromorphic computing research. The goals and motivations for pursuing neuromorphic computing research vary widely from project to project, resulting in a very diverse set of work.

One view of neuromorphic systems is that they represent one pole of a spectrum of repurposable computing platforms (Figure 1). On one end of that spectrum is the synchronous von Neumann architecture. The number of cores or computational units increases in moving across this spectrum, as does the asynchrony of the system.

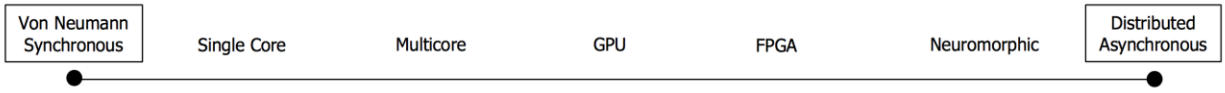


Figure 1: Spectrum of repurposable computing platforms [WSP:Hylton].

One group of neuromorphic computing research is motivated by computational neuroscience and thus is interested in building hardware and software systems capable of completing large-scale, high-accuracy

[3] It is worth noting that this intelligent behavior may be radically different from the intelligent behavior observed in biological brains, but since we do not have a good understanding of intelligence in biological brains, we cannot currently rely on replicating that behavior in neuromorphic systems.

simulations of biological neural systems in order to better understand the biological brain and how these systems function. Though the primary motivation for these types of projects is to perform large-scale computational neuroscience, many of the projects are also being studied as computing platforms, such as SpiNNaker [Furber2013] and BrainScaleS [Brüderle2011].

A second group of neuromorphic computing research is motivated by accelerating existing deep learning networks and training and thus is interested in building hardware that is customized specifically for certain types of neural networks (e.g., convolutional neural networks) and certain types of training algorithms (e.g., back-propagation). Deep learning achieves state-of-the-art results for a certain set of tasks (such as image recognition and classification), but depending on the task, training on traditional CPUs can take up to weeks and months. Most state-of-the-art results on deep learning have been obtained by utilizing graphics processing units (GPUs) to perform the training process. Much of the deep learning research in recent years has been motivated by commercial interests, and as such the custom deep learning–based neuromorphic systems have been primarily created by industry (e.g., Google's Tensor Processing Unit [Jouppi2016] and the Nervana Engine [Nervana2016]). These systems fit the broad definition of neuromorphic computing in that they are neural-inspired systems on non–von Neumann hardware. However, there are several characteristics of deep learning–based systems that are undesirable in other neuromorphic systems, such as the reliance on a very large number of labeled training examples.

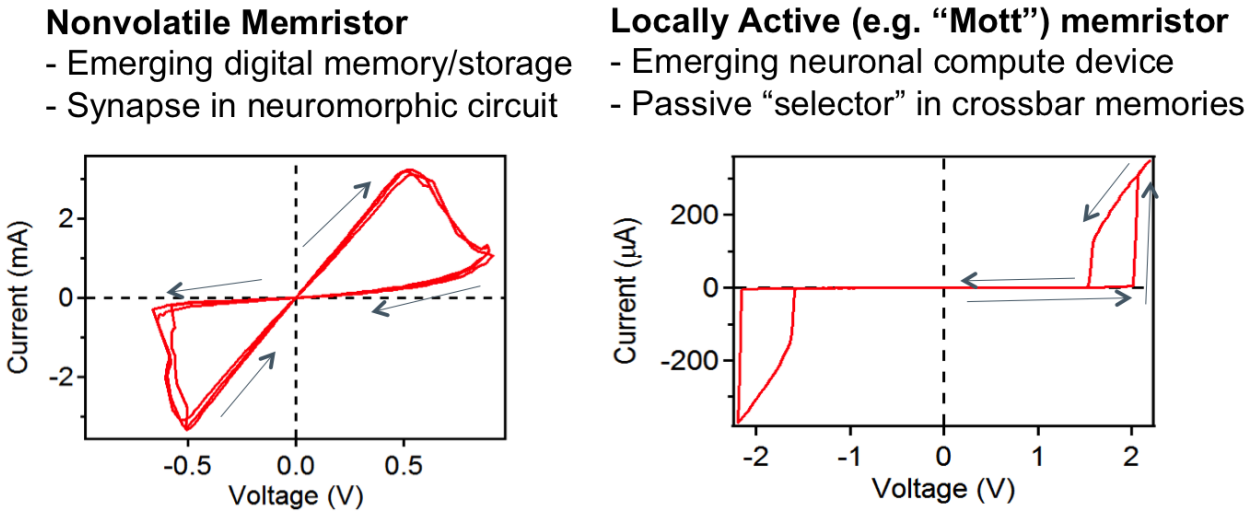


Figure 2: Two types of memristors that could be used in neuromorphic systems [Chua1971, WSP:Williams].

The third and perhaps most common set of neuromorphic systems is motivated by developing efficient neurally inspired computational hardware systems, usually based on spiking and non-spiking neural networks. These systems may include digital or analog implementations of neurons, synapses, and perhaps other biologically inspired components. Example systems in this category include the TrueNorth system [Merolla2014], HRL's Latigo chip [WSP:Stepp], Neurogrid [Benjamin2014], and Darwin [Shen2016]. It is also worth noting that there are neuromorphic implementations using off-the-shelf commodities, such as field programmable gate arrays (FPGAs), which are useful as both prototypes systems and, because of their relative cost, have real-world value as well.

One of the most popular technologies associated with building neuromorphic systems is the memristor (also known as ReRAM). There are two general types of memristors: nonvolatile, which is typically used to implement synapses, and locally active, which could be used to represent a neuron or axon (Figure 2). Nonvolatile memristors are also used to demonstrate activation functions and other logical computations. Memristors used to implement synapses are often used in a crossbar (Figure 3). The crossbar can operate in either a current mode or a voltage mode, depending on the energy optimization constraints. The use of spiking neural networks as a model in some neuromorphic systems allows these types of systems to have asynchronous, event-driven computation, reducing energy overhead. Another use of these devices is to realize a co-located dense memory for storing intermediate results within a network (such as weights). Also, the use of technologies such as memristors have the potential to build large-scale systems with relatively small footprints and low energy usage.

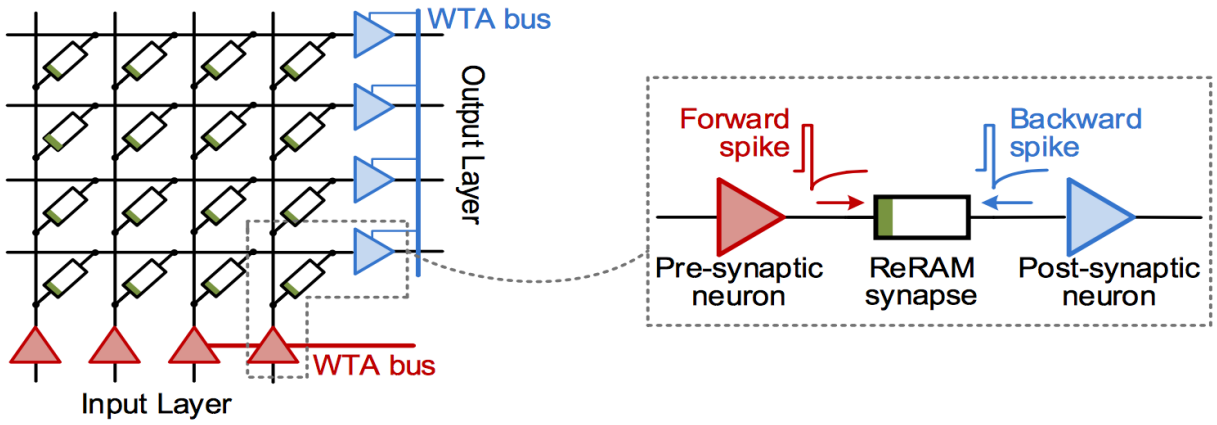


Figure 3: ReRAM used as synapses in a crossbar array [WSP:Saxena].

It is worth noting that other emerging architectures (beyond neuromorphic computers) can implement neuromorphic or neuromorphic-like algorithms. For example, there is evidence that at least some neural-inspired algorithms can be implemented on quantum computers [WSP:Humble]. As multiple architectures are being considered and developed, it is worthwhile to consider how those architectures overlap and how each of the architectures can be used to benefit one another, rather than developing those architectures in isolation of each other.

When considering neuromorphic computing as compared with other emerging computer architectures, such as quantum computing, there is a clear advantage in hardware development in that hardware prototypes are appearing regularly, both from industry and academia. One of the key issues is that projects generating hardware prototypes are not very well connected, and most systems are either not made available for external use or are limited to a relatively small number of users. Even most neuromorphic products developed by industry have not been made commercially available. Thus, communities built around an individual project are relatively small, and there is very little overlap from project to project. Moreover, because the communities are so limited, there has been little focus on making the end-products accessible to new users with the development of supporting software systems.

When considering a von Neumann architecture from the computer science perspective, the following picture emerges (Figure 4).

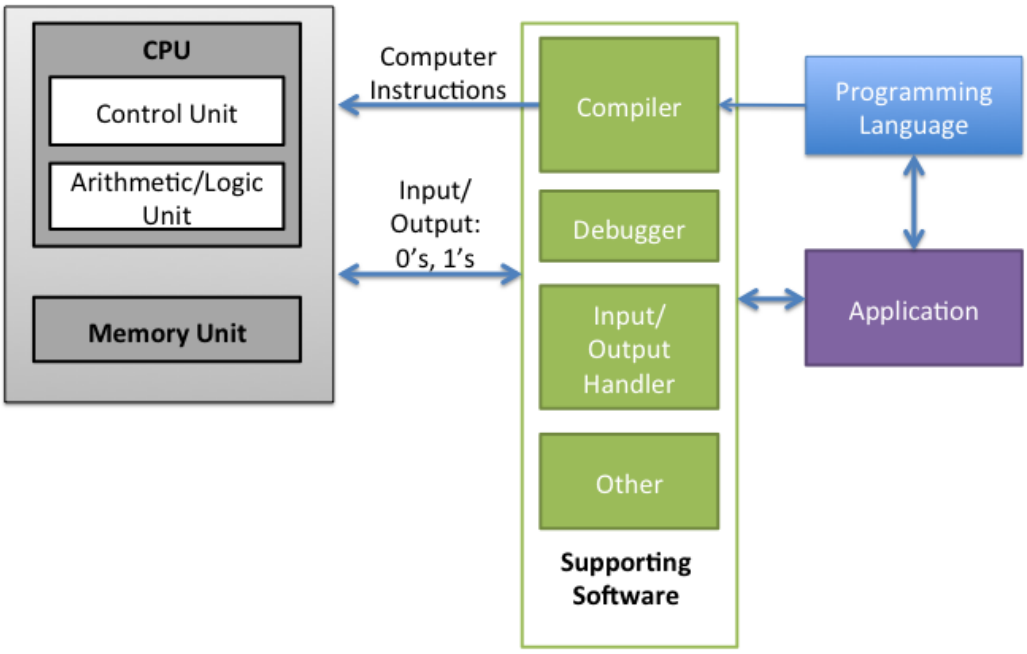


Figure 4: A von Neumann or traditional architecture from the computer science perspective.

In this view of the von Neumann architecture, we see the clear structure of the architecture itself, which is made up of a CPU and a memory unit. The programmer for a von Neumann system typically writes a program for a particular application in a high-level programming language. The program is then translated by the compiler into computer instructions, which are stored in the memory unit and executed by the CPU. The application or user then interacts with the "program" on the CPU through input and output.

We contrast this with a neuromorphic architecture from the computer science perspective (Figure 5).

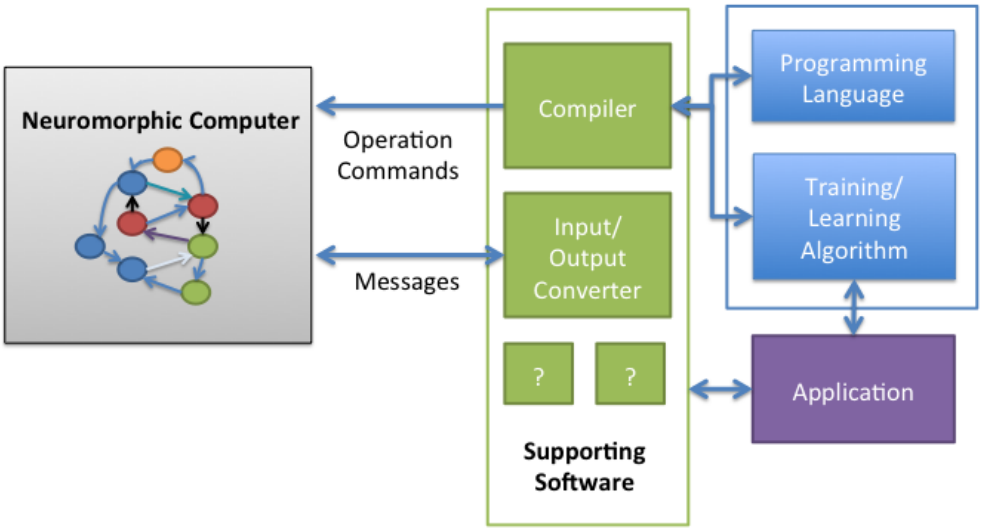


Figure 5: A potential neuromorphic architecture from the computer science perspective.

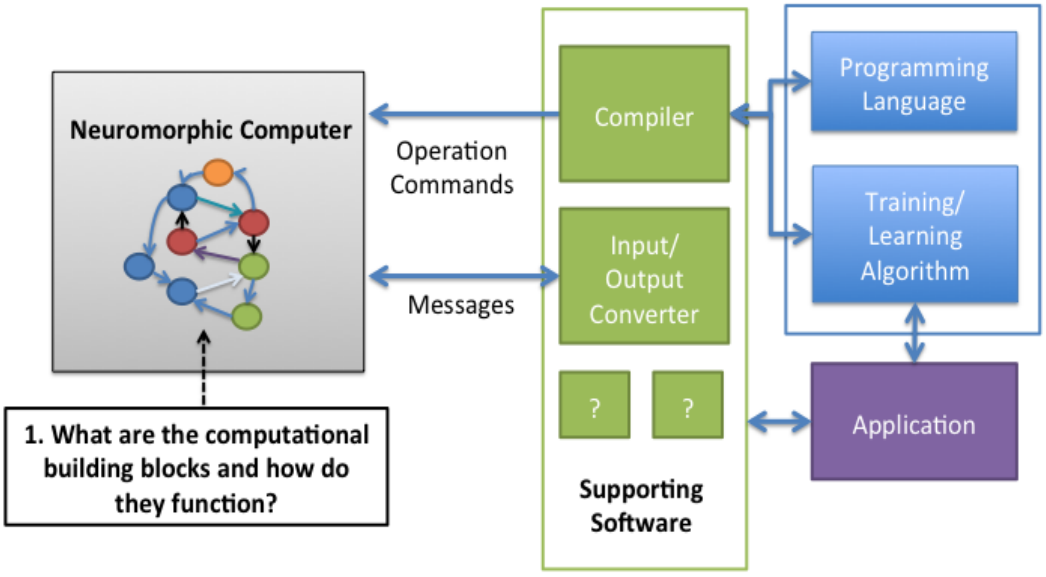
First, the neuromorphic computer itself is no longer made up of a separate memory unit and CPU; it is instead made of different computational building blocks, each of which is likely contain some combination of memory and processing. Although it is possible for there to be a programming language that is used by a programmer, it is more likely that the neuromorphic computer configuration for a particular application is determined by a training/learning algorithm that may be run either off-line and off-chip, or run on the chip itself. There could still be a compiler to turn the programming language or output from the training algorithm into configuration commands for the neuromorphic computer. Finally, the application is likely communicating with the neuromorphic computer using spikes (assuming the neuromorphic computer is representing a spiking neural network) or some other simple message type, in which case an input/output converter will likely be necessary.

III. Open Issues

Neuromorphic computing includes researchers in fields such as neuroscience, computing, computer and electrical engineering, device physics, and materials science. Each research area has a set of open research questions that must be addressed in order for the field to move forward. The focus of the workshop was to identify the major questions from a computing perspective of neuromorphic computing or questions that can be addressed primarily by computational scientists, computer scientists, and mathematicians and whose solutions can benefit from the use of high performance computing (HPC) resources.

Based on the submissions to the workshop and the presentations and discussions during the workshop, six major questions emerged, each of which is described in the following sections and framed in the bigger picture of a neuromorphic architecture (as shown in Figure 5). It is important to note that none of these questions can be answered by computing alone and will require close collaboration with other disciplines across the field of neuromorphic computing.

A. What are the basic computational building blocks and general architectures of neuromorphic systems?



The basic computational building blocks of most neuromorphic systems are neurons and synapses. Some neuromorphic systems go further and include notions of axons, dendrites, and other neural structures, but in general, neurons and synapses are the key components of the majority of neuromorphic systems. The information propagation usually is conducted through spikes: whenever enough charge has flowed in at synapses, a neuron generates outgoing spikes, which causes charge to be injected into the post-synaptic neuron.

What neuron and synapse models are appropriate?

When defining a neuromorphic architecture or model, the associated neuron and synapse models must be chosen from among a large variety of neuron and synapse models. Example neuron models range from very simple (e.g., McCulloch Pitts [McCulloch1943]) to very complex (e.g., Hodgkin-Huxley [Hodgkin1952]).

Figure 6 shows a spectrum of neuron models ranging from more complex and biologically accurate to simpler and more biologically inspired.

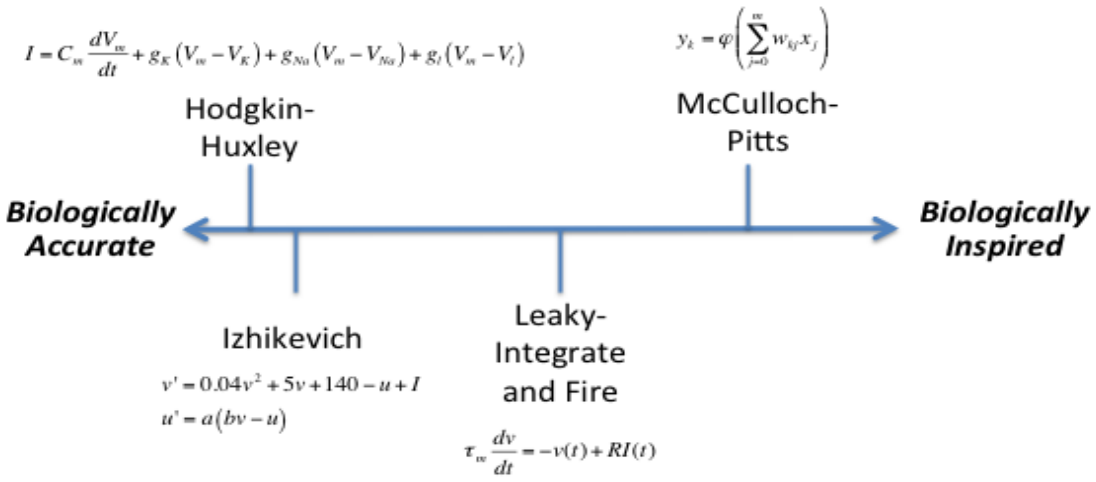


Figure 6: Example neuron models [Hodgkin1952, Izhikevich2003, Gerstner2002, McCulloch1943].

In addition to complicated neuron models, there are a variety of synapse implementations, ranging from very simple synapses that are only represented by weight values (such as those used with McCulloch-Pitts neurons) to extremely complex, biologically realistic synapses that include notions of learning or adaptation. Artificial synapses may or may not have a notion of delay, depending on the implementation. Some synapse models represent synapse activity using a conductance-based model [Vogelstein2007] rather than a weight value. Synaptic weights may be adapted over the course of network simulation via learning mechanisms such as Hebbian-learning-like rules (e.g., spike-timing-dependent plasticity (STDP) [Caporale2008] or long-term and short-term potentiation and depression [Rabinovich2006]). In general, we do not believe that detailed bio-mimicry is necessary or feasible because neurobiology is extremely complicated and still not well understood. The task instead should be to create an understanding of what matters in the biology (that is, a theory) and to use that theory to effectively constrain and integrate the component models and to transfer certain intuition about the brain to a different technological substrate (such as a neuromorphic chip or software model).

What other biological components may be necessary for a working neuromorphic system?

Beyond neurons and synapses, there are a variety of other biologically inspired mechanisms that may be worth considering as computational primitives. Astrocytes are glial cells in biological brains that act as regulatory systems; in particular, they can stimulate, calm, synchronize, and repair neurons. Certainly we will want to consider what computational effects these regulatory-type systems will have on neuromorphic models. Neurotransmitters and neuromodulatory systems also have a significant effect on the behavior of biological brains. It is not clear how these systems influence the capabilities of biological brains, such as learning, adaptation, and fault tolerance, but it is worthwhile to consider their inclusion in neuromorphic models. In general, the goal should be to make minimalistic systems first and then to grow their complexity as we understand how the systems operate together. If we start with the proposition that it must be very, very complex to work, then we will surely fail.

There are many different types of neurons and synapses within biological neural systems, along with other biological components such as glial cells; different areas of the brain have different neuron types, synapse types, connectivity patterns, and supporting systems. Artificial neural networks have taken inspiration from different areas of the brain for different types of neural networks. For example, the structure of convolutional neural networks is inspired by the organization and layout of the visual cortex [LeCun2015], whereas hierarchical temporal memory (HTM) takes its inspiration from the organization of the neocortex [Hawkins2016, WSP:Kudithipudi] (Figure 7). When considering the model selection, it may be worthwhile to target a specific functionality or set of applications and take inspiration from a particular part of the brain that performs that functionality well in determining the characteristics of the model.

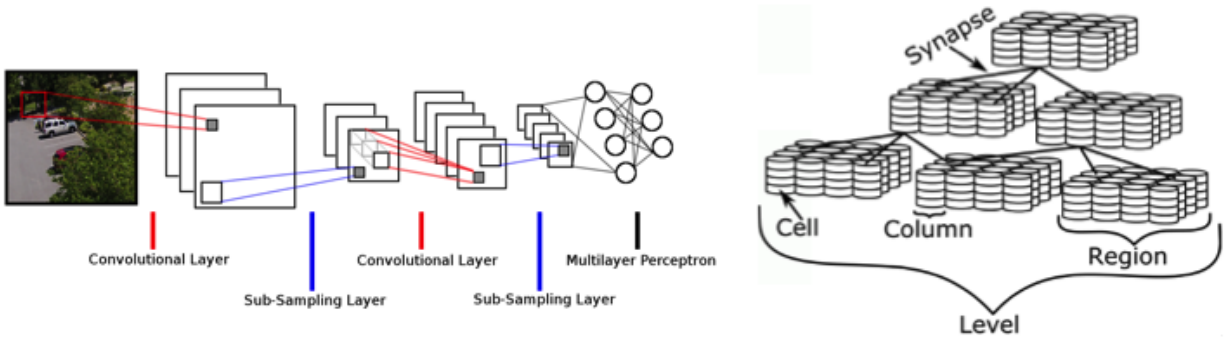


Figure 7: Convolutional neural network example (left) as compared with hierarchical temporal memory example (right) [WSP:Kudithipudi].

How do device and materials characteristics influence computational building block selection?

When selecting the appropriate computational primitives, a variety of questions must be addressed. One major issue associated with selecting the neuron and synapse models is what adaptations need to be made in the model as a result of the chosen device or implementation. For example, certain device implementations may restrict a parameter space or functionality of a certain component of either the neuron or synapse model. These adaptations will almost certainly have an effect on the theoretical

capabilities of the model. An open research challenge is to create a theory that incorporates the constraints of the implementation rather than to create an implementation that perfectly matches a particular model. In this vein, there are really two challenges that, given the uncertain state of both device and neural research, need to be addressed in parallel. First, there exists a theoretical space where we can formally show that certain algorithmic computations can produce certain functions. Understanding that computation abstractly (perhaps in terms of computational building blocks) is critical. The second challenge is to understand how to reformulate those building blocks to best leverage the specialized devices that have unique abilities to accelerate or compress certain operations. That is, the underlying theory is very important, but considering that theory in the context of physical devices is also important.

How should neurons and synapses be organized for effective information propagation and communication?

Connectivity and plasticity in synaptic weighting dominate the complexity of neuromorphic computing. Offering the compatible connection density as human brain usually results in unaffordable complexity and unsalable system design. Therefore, how to organize neurons and synapses to obtain the greatest density of interconnect at the local level while offering scalable long-range connectivity and balancing the traffic in routing of neural events remains an open research question.

How does computational building block selection influence data representation?

The choice of models will also necessarily affect the way data is represented in the system and how data will be encoded and decoded between a neuromorphic device. For example, a spiking neural network model requires inputs to be encoded as spikes and spiking events to be decoded as outputs. When selecting the model, we should consider what impact this encoding and decoding process will have on the overall performance of the system and determine whether any benefits we gain from the model selection in theoretical capabilities and efficiency may be outweighed by the cost of communication to and from the device.

What are the criteria in evaluating neuromorphic system architectures?

Our goal is to develop practical computational systems inspired by biological mechanisms. When evaluating and selecting the architecture, we shall consider if the system is good for things that brains are good at and provides features such as high energy efficiency, learning ability, adaptive to environment, etc. The impact from physical design and the feasibility of the hardware implementation are also important. A suite of benchmarks/tasks for evaluation and comparisons might be necessary.

What level of biological detail is necessary to capture the desired functionality?

Another major research question associated with selection of the models is what level of complexity and/or biological realism is necessary to achieve desired functionality. This question is inextricably linked with the desired end-goal of the neuromorphic system. If the goal of the neuromorphic device is to produce biologically realistic simulations, then the models will necessarily be as biologically realistic as possible. However, this report is written from the standpoint that the goal of a neuromorphic architecture should be to produce computationally useful systems, not biologically realistic behavior.

Some conjecture that in order to achieve intelligent behavior, we must emulate biological systems as closely as possible. However, even with the leaps that have been made in neuroscience, neurobiology, neuroanatomy, and computational neuroscience in recent years, we still do not have a full understanding of the functionality of biological neural systems or how their functionality ultimately leads to intelligent behavior. As such, there is no guarantee that more biologically realistic systems will indeed lead to intelligent neuromorphic systems. To address this question, we must rigorously examine how each biologically inspired model or mechanism influences the performance of the system and only move forward with those that clearly affect performance (Figure 8). That is, we should never include a mechanism or property in a neuromorphic system just because it is present in the biological brain; it must have a justifiable purpose in the performance of the neuromorphic system. In general, while practical systems will almost certainly draw inspiration and ideas from biology, it is very unlikely that detailed simulation of biological brains will be practical or necessary in building computationally useful neuromorphic architectures.

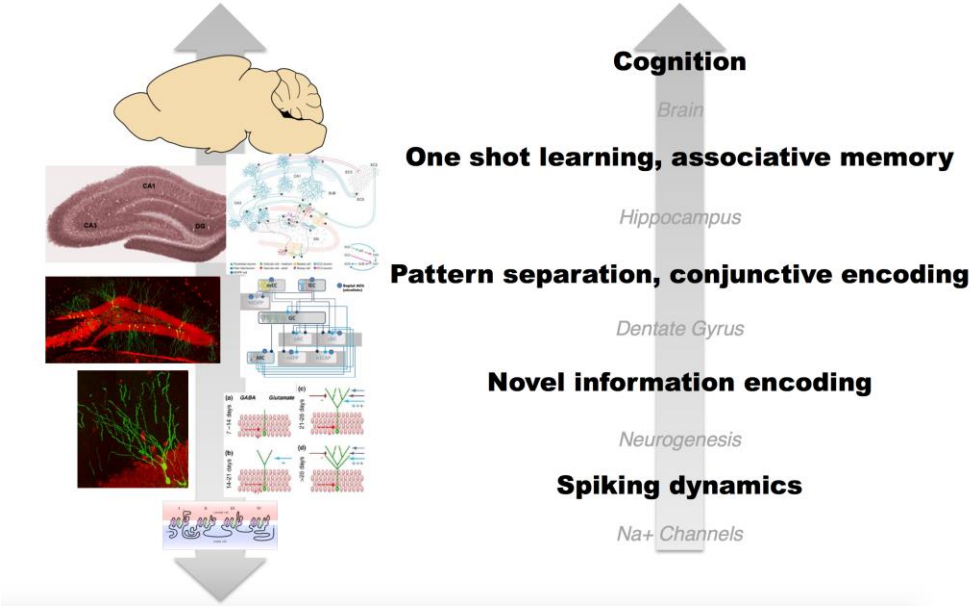


Figure 8: Levels of abstraction in biological brains and what functionality they may allow [WSP:Aimone].

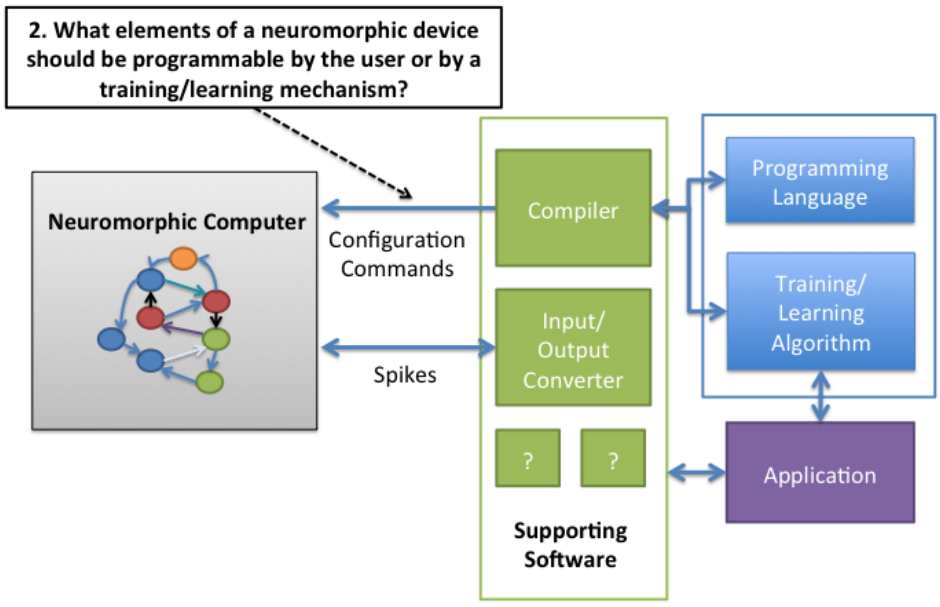
It is of vital importance that we are able to build and test systems with varying levels of complexity in order to understand the relative strengths and weaknesses of each system. For example, we speculate that the more complex the neuron/synapse model is, the more difficult it will be to efficiently program the system (or, the more difficult it will be for the system to learn its parameters). On the other hand, by choosing simpler models, we may be significantly reducing the capabilities of the neuromorphic system. Building hardware is often an expensive endeavor, in both time and resources. Software simulation of models is going to be a key step in understanding the models and their capabilities, and it will be necessary for these software simulations to be capable of handling very large neuromorphic networks to show utility on real applications. Further, we believe that model reduction techniques, such as uncertainty quantification and sensitivity analysis (UQ/SA) that are widely used in other simulation-heavy sciences, could be invaluable in helping guide our abstraction of complex biological systems into more abstract computing-friendly models of neural computation [WSP:Aimone].

There is almost certainly no one right answer to the selection of computational primitives that will be the most appropriate for all neuromorphic systems. Neuromorphic systems will be used in a wide variety of ways, and it is likely that each of these systems will make use of different types of models. However, it's highly important that we are able to analyze these models, to compare them, and to evaluate the strengths and weaknesses of each approach in real use-cases.

Recommendation: Connect the theory of computation with neuroscience and the nonlinear physics of devices using machine learning and large-scale simulation

1. Look to both neuroscience and devices to develop potential computational building blocks and models.
2. Develop software simulation modules of potential computational building blocks that can be combined interchangeably to build full (potentially large-scale) neuromorphic systems that can be evaluated on real applications.
3. Use the software simulator study and evaluate the neuromorphic system on a variety of performance characteristics and highlight the strengths, weaknesses, and overall capabilities of each resulting system. Make use of existing neuromorphic implementations (such as TrueNorth, Neurogrid, and Latigo) as appropriate.
4. Develop hardware of computational building blocks and architecture in small-scale (and potentially large-scale) systems to evaluate system performance and scalability and verify the software simulators.
5. Use the software simulator study to inform the development of the computational theory of neuromorphic systems.

B. What elements of a neuromorphic device should be configurable by the user or by a training/learning mechanism?



One of the issues with developing neuromorphic systems is defining the level of reconfigurability of the device. Neuromorphic devices are usually made of two fundamental component types (neurons and synapses), but even once a model or models are chosen for device implementation, there is still a question of what elements of the device will be programmable or flexible for implementing new applications (without changing the fundamental device structure). For example, elements that may be determined as part of programming or that may be adaptable as a part of learning include

1. which models will be chosen if multiple neuron or synapse models are available,
2. activating or deactivating neurons and synapses to influence the structure of the network,
3. setting connectivity between neurons and synapses, and
4. setting parameter values.

What effect does device programmability have on device performance?

There will likely be major trade-offs between device performance/footprint and programmability. Programmability will not be a major factor in many specific applications, such as devices on autonomous vehicles or in brain-computer interfaces. In these cases, the goal of the device is to typically perform a very small set of tasks in an extremely efficient way and often with a much smaller device than would be required when performing the same task on a programmable chip. For other uses of neuromorphic computers, such as a co-processor in a larger, heterogeneous computing system, more flexibility in the configuration of the device is a desired quality in order to ensure that the device may be used for a wide variety of applications.

The level of configurability at the device level is tied closely to the selection of models and computational primitives. The complexity of the selected models may influence the type of configurability that is required. Given enough flexibility in the connectivity and structure, it is likely possible to replicate the behavior of more complex models. For example, the behavior of a single neuron in a complex neuron model may be replicated by multiple neurons with simpler neuron models.

This research question is also inextricably linked with research questions in device development and materials science research. The level of configurability in a physical hardware system is going to be heavily dependent upon what flexibility is actually allowed by the type of device and materials used. For example, available connectivity is likely to be restricted, limiting the types of structural selections that will be made on the device. The effects of these restrictions in flexibility/configurability can be studied in simulation and provide insights to the hardware and materials researchers, while insights and innovations from hardware and materials research will drive developments in simulation.

What desired characteristics of neuromorphic systems rely on flexibility and configurability of the device?

The level of configurability or flexibility at the device level will also determine how the device is programmed, what algorithms can be evaluated on-chip, and what types of learning are possible on-line. There is a trade-off in making the device more flexible or configurable because with more programmable options at the device level, there are more variables to be determined by a programmer or an algorithm. However, by restricting the device reconfigurability, we are also potentially restricting the device's adaptability. One of the key features of biological neural systems is their ability to repair or heal

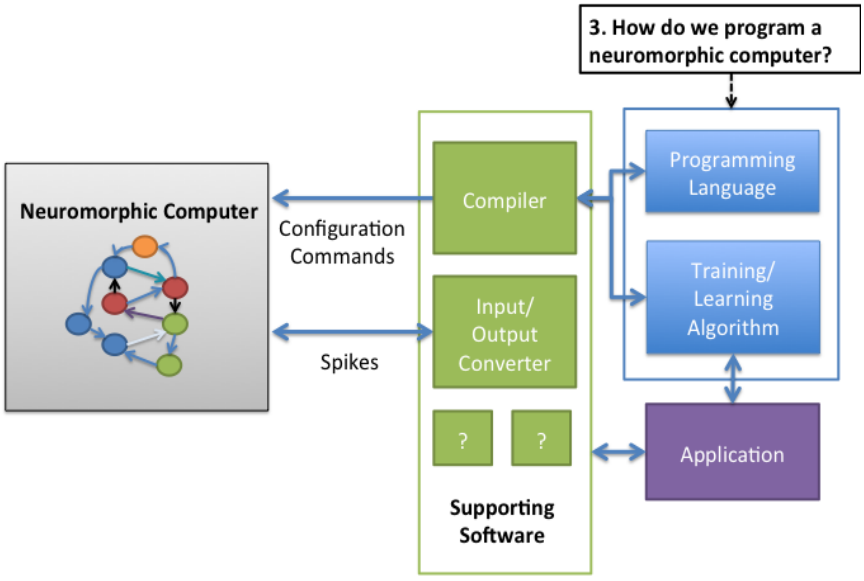
themselves or to develop new structures in order to compensate for lost functionality. By including the reconfigurability as part of the device's capabilities, we are allowing for the inclusion of mechanisms that will allow the neuromorphic system to have similar adaptability features.

Once again, there is certainly no single correct answer for what level of configurability is appropriate for neuromorphic systems. We suspect that there will be a spectrum of resulting neuromorphic devices, ranging from custom neuromorphic application-specific integrated circuits (ASICs) for specific applications to neuromorphic computers which have significant flexibility and can be used for a broad range of applications.

Recommendation: An integrated approach of computational models and reconfigurable hardware

1. Define a list of desired characteristics of neuromorphic systems and determine which characteristics rely on configurability and reconfigurability at the device level.
2. Augment the computational building blocks simulators to allow for different levels of configurability of the neuromorphic system.
3. Study and evaluate the effect of device configurability/adaptability on programming/training difficulty and application space on the software simulator; inform device and materials research in reconfigurable hardware.
4. Implement the reconfigurable hardware design and provide feedback to enhance the software simulator.

C. How do we train/program a neuromorphic computer?



A key question for neuromorphic computers is how to use the device for real applications. In answering this question, we must define what it means to "program" a neuromorphic device, or perhaps more

appropriately, how neuromorphic devices will learn or be trained. As such, we define three terms to describe the operation of neuromorphic computers on real applications.

- **Programmed**: The user explicitly setting all parameters and potentially the structure of the networks to perform a particular task.

- **Trained**: A training algorithm is defined. Example situations from the application are presented as part of the algorithm. The user provides feedback (as part of the algorithm) as to how well the neuromorphic device is performing the task, and the algorithm updates parameters and potentially structure based on that feedback. In machine learning this is often referred to as "supervised learning." An example of a training algorithm for certain types of neuromorphic implementations and models is back-propagation. We also categorize reinforcement learning as a training method because the algorithm is receiving feedback. In this case, feedback is either "good" or "bad" as opposed to the correct answer when a wrong answer is given.

- **Learning**: A learning algorithm is defined. Example situations from the applications are presented as part of the algorithm, and the algorithm defines ways in which the structure and parameters are updated based on the input it receives. In this case, the user does not provide feedback, but the environment may provide some sort of inherent "reward" or "punishment" (in the case of reinforcement learning). In machine learning, this is often referred to as "unsupervised learning." Unsupervised learning algorithms typically create a compressed "representation" of the input structure. An example of a learning algorithm for certain types of neuromorphic implementations and models is spike-timing-dependent plasticity.

For neuromorphic computing in the real world, there are roles for all three use-cases, and it is likely that some combination of the three will be used. The role of a software developer for neuromorphic computers is going to be radically different depending on the selected programming paradigm. The developer for an explicitly programmed neuromorphic computer will need to explicitly set all parameters and structure and understand the implications of each selection and how they interact. Developers for trained neuromorphic computers will need to consider what examples should be presented and what feedback to provide as part of the training process. They will also likely define parameters for the algorithm itself. For learning neuromorphic computers, the "developer" may be defined as the person who is presenting examples, or there may be no developer at all. For trained and learning neuromorphic computers, the term developer may also be used for the person who *defines* the training or learning algorithm. Another role of a programmer for a neuromorphic computer will be to determine what inputs are given, how those inputs are represented, and how they are presented to the device.

Highly redundant, high-volume inputs are the target of unsupervised learning algorithms. The challenge for the unsupervised learning algorithm is to substantially compress and abstract that input so that it can be separated and compared with other low-volume inputs. Low-volume inputs that are examples of what (a human thinks) the machine should do/output are targets of supervised learning algorithms, preferably used in conjunction with the compressed representations created by the unsupervised algorithms. The challenge for the supervised learning algorithm is generalize from these examples to similar cases that it has never seen before. End-to-end training methodologies like convolutional deep nets bypass the need for unsupervised learning because they work on data sets where the unsupervised learning is not

needed/helpful (like collections of labeled images where there is no correlation among successive images).

Very low-volume inputs that provide feedback about the overall state of the system are the target of reinforcement learning algorithms. The challenges for reinforcement learning algorithms are (1) to construct a cost function (which is easy for tasks like games but can be very difficult to define for other use-cases) and (2) to give enough information so that a large network can effectively organize itself. Presumably the reinforcement learning is much easier if the state has been compressed by something like an unsupervised learning algorithm.

There is absolutely no reason to believe that there is a simple "learning rule" that might be applied at a synapse (e.g., a Hebbian rule or STDP) that will somehow create a general-purpose learning system. Very little work has been done on learning systems that close the loop with the real world—systems in which the output feeds back to the input by changing the environment.

There are strengths and weaknesses for each of the three programming paradigms. The inclusion of manual programming allows for the neuromorphic device to be used for applications other than its native purpose (running a neural-inspired system). By allowing for manual programming, we open the door for the possibility of using the underlying chip structure for non-neurally inspired computational problems that would still benefit from features such as event-driven computation and co-located memory and computation. However, manual programming is not likely to be a scalable approach for very large neuromorphic systems; moreover, it will require an expert in the underlying architecture in order to program it in a meaningful way.

The key advantage for training methods is that many supervised learning methods have already been developed in artificial neural network theory that may be able to be mapped to neuromorphic systems. There are several issues moving forward with training methods, however. One is that many of the training algorithms and methods were devised with the von Neumann architecture in mind and thus will not necessarily take advantage of the native processing capabilities of a neuromorphic system. It will require a fundamental shift in the way we think about algorithm development and programming in order to adapt these training methods to neuromorphic systems. Moreover, we should not limit ourselves to simply adapting existing techniques to neuromorphic systems. We should also figure out how to do supervised learning in truly neuromorphic algorithms/systems.

Another major issue with training methods or supervised learning is that problems that involve data require labeled examples to work properly. As noted, one of the key driving forces behind the need for a new computing paradigm is the tremendous amount of data that is being gathered. It will likely be impossible for there to be sufficient manpower to provide enough labeled examples in these complex data sets to make use of existing training methods, for several reasons:

1. Existing methods tend to require a large number of labeled examples that span the set of “interesting” characteristics in the data set.

2. Many of the complex data sets are domain specific, where there are few who have the expertise required to label the data in a meaningful way.

3. The sheer complexity of much of the data being gathered is difficult to comprehend, even for the human experts.

4. In many cases, it is not clear what characteristics of the data should be labeled in order to produce scientific discovery.

As such, existing supervised learning methods are clearly not going to solve the types of problems we would like neuromorphic systems to address. An ideal neuromorphic system will incorporate a notion of learning in which the system itself is intelligently analyzing the raw data and indicating which features may be important. The primary issue with developing a learning algorithm is that it is not clear how learning is done in the brain or how it should be done in a neuromorphic system. Basic learning mechanisms such as Hebbian learning or spike-timing-dependent plasticity [Caporale2008] provide examples of how a neural network may learn in an unsupervised way, but those methods are also somewhat limited. Though this has been a common approach in the literature, there is no reason to believe that there is a simple "learning rule" that might be applied at a synapse (such as a Hebbian rule or STDP) that will somehow create a general-purpose learning system. One of the biggest goals of the neuromorphic computing community is to determine learning methods that enable one-shot learning, or at the very least, learning from a small set of examples, of the type that occurs in biological brains. One speculation is that one-shot learning implies that the system has experience in a closely related task such that it already has an efficient representation. In this case, learning is then some small refinement on an existing network. Pursuing this approach is one potential path to achieving one-shot, or near one-shot, learning in neuromorphic systems.

We should also not think of neuromorphic learning mechanisms as existing in isolation. For many applications, a neuromorphic system will likely not only be taking input from the environment and processing that input in some way but will also be making decisions that will, in turn, affect the environment. Very little work has been done on learning systems that close the loop with the real world. Because one of the major advantages of neuromorphic systems is their potential for extremely low power, it is extremely likely that one of their major use-cases will be in real-world environments (e.g., on sensors or autonomous vehicles). As such, it is important that we consider how to develop neuromorphic learning algorithms that close the loop with the real world.

Overall, it is clear that an entirely new way of thinking about algorithm development will be required for neuromorphic systems (Figure 9). The community will have to break itself out of the von Neumann way of thinking in order to do so. In order to develop new learning methods with the characteristics of biological brains, we will need to learn from cutting edge research in neuroscience. As part of that process, we will need to build a theoretical understanding of “intelligence.” Without the theoretical underpinnings, we will not be able to implement truly intelligent neuromorphic systems.

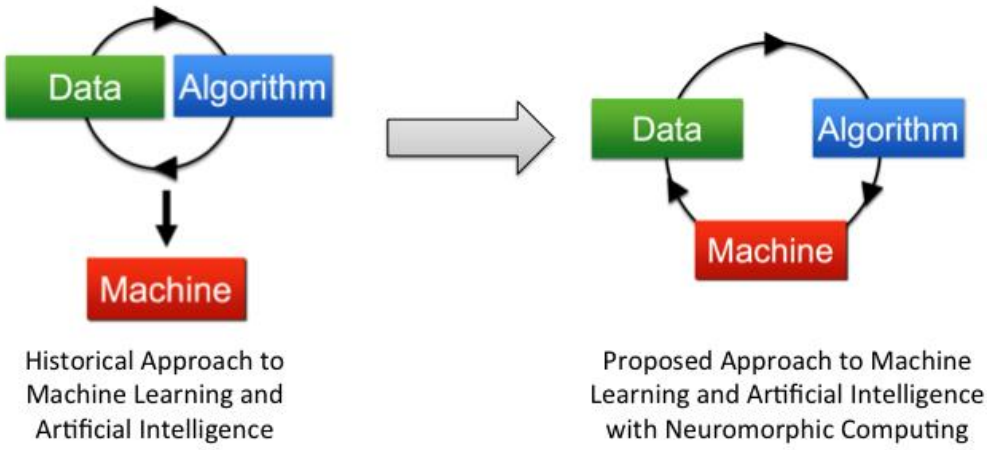


Figure 9: Incorporation of machine/device design into algorithm development [WSP: Mohsenin-Koushanfar].

Where and when should training/learning for a neuromorphic system take place?

A major technical and detailed consideration, especially for trained and learning neuromorphic computers, is where training/learning takes place and when training/learning takes place. We define the location of learning as either "on-chip" or "off-chip." We use these terms for convenience even though the actual device or computer acting as a neuromorphic computer may not be a chip. On-chip training/learning is when the training/learning algorithm is implemented as part of the operation of the neuromorphic computer. Many existing neuromorphic computers do not include any notion of on-chip training or learning. Off-chip training/learning is when the training/learning algorithm does not take place on the neuromorphic computer (though the neuromorphic computer may take part of the algorithm or a simulation may be used in its place). We define the time of learning as either "on-line" or "off-line."

On-line training/learning occurs when the algorithm makes real-time updates in a real situation. Usually on-line refers to a learning instance, which the neuromorphic chip is making decisions on how to update itself dynamically without feedback. However, training algorithms may also fit into this paradigm as part of a framework such as reinforcement learning, where the user may be providing minimal feedback. Off-line training/learning usually occurs when the training knowledge is available earlier than when the device is used in the environment. One possibility for implementing hybrid on-line/off-line learning system is a system that can operate in real-time using an off-line trained model but store sampled input statistics, network outputs, and implications and update itself off-line at a later point. This approach is arguably what the cortex/hippocampus interaction provides to brains; our cortical "models" update very slowly (potentially even off-line during sleep), whereas the hippocampus is continuously learning and storing what occurs in the world and eventually using this info to update the cortical model.

Neuromorphic devices can contain combinations of on-line, off-line, on-chip, and off-chip training/learning. On-line training/learning methods are usually by necessity on-chip. Off-line training/learning mechanisms may be on-chip or off-chip. Even neuromorphic computers that rely heavily on on-line learning will almost certainly include off-line pre-training and/or programming by the user in which some structure and parameters are defined and then refined as part of the on-line learning process.

How does the application choice affect the programming/training/learning paradigm?

Clearly the choice of applications for neuromorphic computers is going to have a radical effect on the selected programming/training/learning paradigm. Beyond defining the programming/training/learning paradigms, it is of vital importance that we understand how these paradigms interact and what effect the selection of each algorithm has on the overall performance of resulting neuromorphic systems. The development of on-chip algorithms in particular will require significant interaction with the hardware developers and material scientists as to what operations are possible given the selected hardware and materials.

Which biological learning mechanisms should be considered as part of training/learning?

Another major question associated with programming/learning/training method is what that method defines. Many traditional neural network training or learning methods refer to algorithms that set the weight values of synapses in the neural network. The selection of the model and the flexibility/programmability of the device will affect what the programming method is capable of defining. Clearly biological learning systems encompass not only synaptic learning but also neurogenesis and neuronal-death, axonal growth and pruning, and neuromodulation. One of the key questions associated with developing training and learning mechanisms is how important structural evolution and change is as part of learning in biological brains. For computer scientists and computational scientists, this is clearly a question that can be addressed through simulation. We may examine the effect of changing structure and incorporating in complex neuromodulatory systems inspired by biological learning on the overall performance of a particular neuromorphic model. Using this information, we may inform the requirements of device developers and materials scientists in the selection and development of new devices and materials for neuromorphic systems.

One of the key features of biological brains that likely enables very fast learning from limited examples or trials is the structural features that are present in biological brains as a result of evolution that are then "customized" through learning processes. It may be the case that a neuromorphic system includes a longer-term off-line training or learning component that creates a gross network structures or modules that are then refined and tuned by shorter-term on-line training or learning component.

Can we define a set of neuromorphic computational primitives that can be used to build a large set of neuromorphic algorithms?

One of the key challenges with the design of neuromorphic hardware is preventing obsolescence in the face of new neuromorphic algorithms. This is particularly problematic for learning algorithms, as better training approaches are constantly being developed, thus making any existing hardware learning structures much less effective and attractive. Identifying a set of neuromorphic computational primitives from which any neuromorphic algorithm, including learning algorithms, can be "assembled" would make be beneficial. If these computational primitives can then be implemented through efficient circuits into a neuromorphic device, the device would be able to implement most new neuromorphic computational algorithms. This would minimize the possibility of device obsolescence.

Can we define a high-level abstraction of neuromorphic systems for general neuromorphic algorithm development?

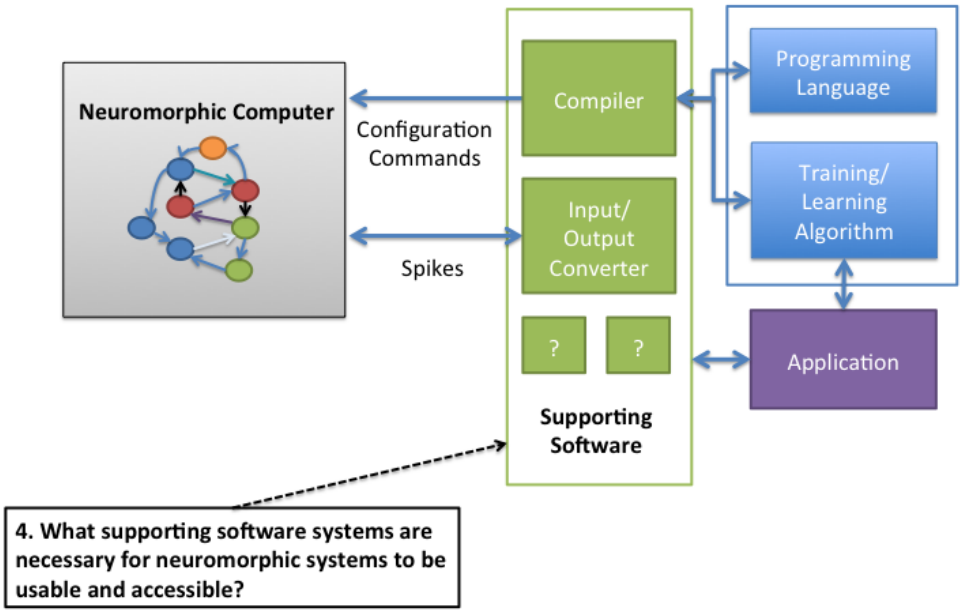
Clearly the selection of computational primitives will have a serious impact on algorithm development. However, there is also the need to abstract away details about a particular architecture when developing algorithms. Ideally, a single algorithm developed for neuromorphic computing should be able to be implemented on most, if not all, neuromorphic architectures or devices. It is worthwhile to consider defining the high-level properties of neuromorphic systems that can be used as part of a given algorithm. Then these high-level properties can be used to define a commonly accepted pseudo-programming language, where each "compiler" to convert the algorithm pseudo-code may be then be developed for each individual neuromorphic architecture or device.

Though we have a proof of concept that complex learning algorithms exist (in the human brain itself), it is not clear that we will be able to replicate that behavior in a neuromorphic computer. However, it does seem clear that if any computing platform is capable of replicating that behavior, a neuromorphic computer seems the most natural to do so.

Recommendation: Develop theories of training and learning informed by neuroscience to be applied to neuromorphic computing.

1. Step outside of the von Neumann way of thinking about algorithm development. Take inspiration from existing training and learning methods to develop new algorithms specifically for neuromorphic devices.
2. Formalize computer science–compatible models of biological neural learning processes, such as spike-timing plasticity, synaptogenesis, and neurogenesis, to allow appropriate identification of both algorithmic utility and hardware feasibility.
3. Create theories of the mind and theories of intelligence that can be used to inform the development of entirely new algorithms and learning methods.
4. Define a set of neuromorphic computational primitives that can be used to "assemble" a large variety of existing neuromorphic algorithms and implement new learning algorithms. This will minimize the possibility of hardware obsolescence.

D. What supporting software systems are necessary for neuromorphic systems to be usable and accessible?



In order for neuromorphic systems to be a valid complementary architecture for the future computing landscape, it is vital that we consider the accessibility and usability of the systems during the early stages of development. One of the major questions associated with usability and accessibility is how neuromorphic systems will actually be integrated into environments and what supporting software is necessary. We describe two example use-cases of neuromorphic computers or processors and the supporting software that will be required for each use-case.

In one use-case, a neuromorphic system may be directly connected to sensors and/or control mechanisms as an embedded system on autonomous vehicles, sensors, or robots that are deployed in real environments. In this case, it will be necessary to build custom protocols and schemes for communication among custom devices within the deployed system itself, as well as the functionality to communicate results and/or data with a centralized server (Figure 10). For this case, it is almost certain that most training/learning will be done off-line (and perhaps also off-chip) and loaded onto the neuromorphic device, so training/learning software will be required. The neuromorphic processor will probably be customized for the particular application with very little programmability, adaptability, and on-line learning, in order to reduce the complexity and, as a consequence, the size and the energy consumption of the device.

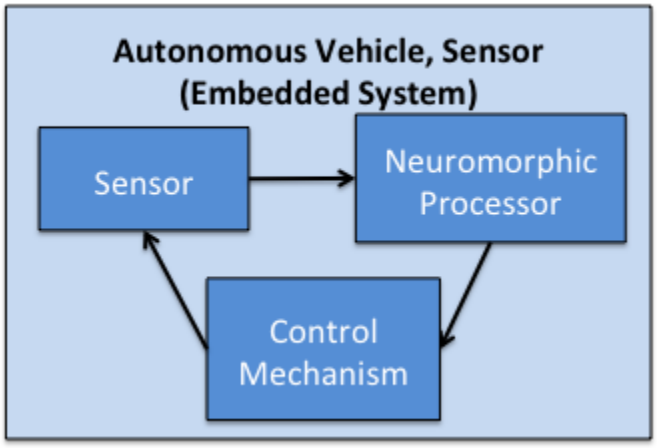


Figure 10: Embedded system example.

A second use-case for neuromorphic computers is as a co-processor in a future heterogeneous node. Figure 11 shows an example of a heterogeneous node, which includes traditional CPUs, GPUs, a neuromorphic processor, a quantum processor, and potentially other emerging architectures. For this use-case, the supporting software will be extensive, and the device itself will likely be more programmable than neuromorphic devices for other use-cases in order to enable the device to be as useful as possible in the heterogeneous node. Communication protocols will be required.

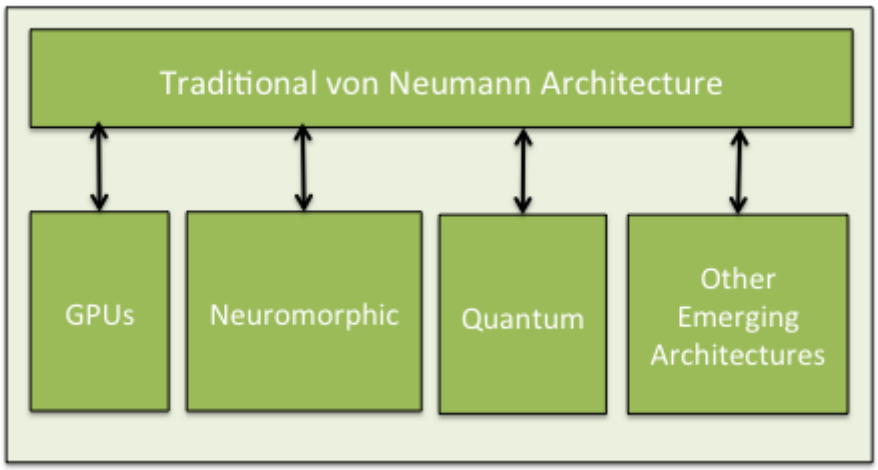


Figure 11: Co-processor example.

Particularly for the final use-case, but also for all other use-cases of neuromorphic devices, supporting software for the neuromorphic devices and an associated software community will be necessary in making neuromorphic computing accessible enough for wide release. For instance, it will almost certainly be important to define abstract network representations (such as PyNN [Davison2009] or N2A [Rothganger2014]) that are common among different implementations and devices but also flexible enough to accommodate the capabilities and characteristics of each unique device and application. To be effective, these tools should be capable of both creating simulations of a neural circuit algorithm or model on conventional systems as well as eventually providing abilities to compile onto specialized

neuromorphic hardware or systems. The community is a vital component in defining these representations, just as the developer community was vital in defining standards for various aspects of traditional computing.

What does a "program" for a neuromorphic computer look like?

Instances of abstract network representations can be thought of as high-level "programs" for neuromorphic devices, which will then need to be "compiled" for each individual device. For some devices, the abstract representation will likely have a direct conversion process, resulting in a simple compiler. However, depending on the implementation and its associated restrictions in parameter representations or connectivity, an extensive mapping process may be required. Thus, the development of basic compilers for neuromorphic systems to perform the conversion from abstract network representation to machine code for the neuromorphic device will be an important software component (Figure 12).

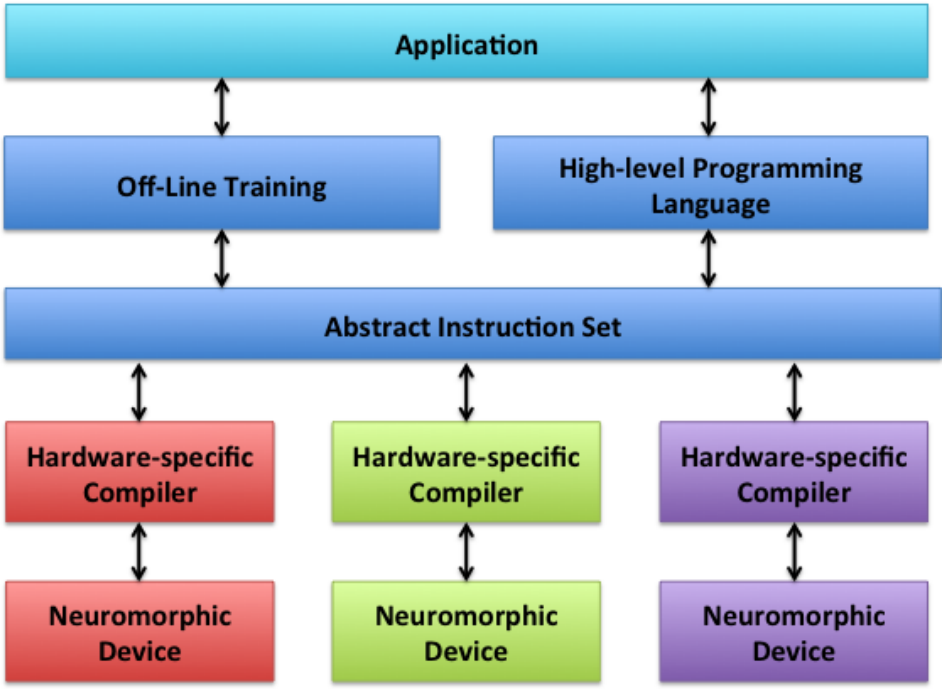


Figure 12: Example software-stack, that includes hardware-specific compilers, an abstract instruction set, high-level programming languages, and off-line training mechanisms.

We make the analogy that the abstract network representation is similar to assembly language. For neuromorphic computers, the development of this assembly language may be performed in several different ways (see Section III.C): the user may manually set the parameters (programming) or the network may be designed using supervised (training) or unsupervised (learning) processes. Software that performs these algorithms and interact with either simulators or the devices themselves may be required in developing the appropriate abstract network instance for any given application.

What other types of software are required for using neuromorphic systems?

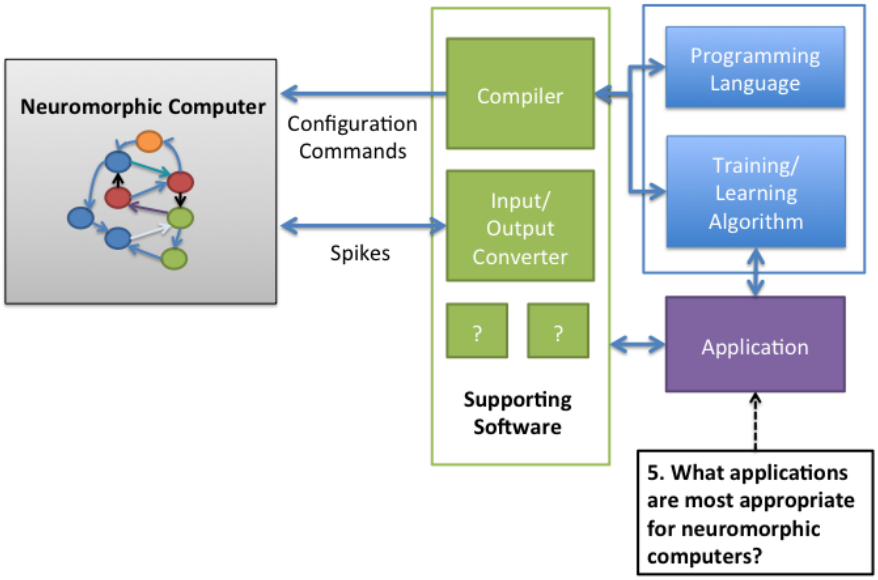
Because of this new way of thinking about developing software, visualization tools will be critical to understanding the performance of neuromorphic computers. Visualization tools will be key in helping familiarize a novel user with the new computational structure. For developers, visualization tools for the abstract network representation as well as visualizations of each device itself will prove to be invaluable sources for identifying issues at the system level and, depending on how detailed they are, at the component level. Even for device developers, detailed visualizations will provide significant insight in verifying and debugging hardware implementations. While building these architectures from the ground up, we will also have the opportunity to introduce user interface experts into the development process early and, as a consequence, develop intuitive user interfaces for interacting with, training, and programming the neuromorphic devices.

Overall, it is clear that there are many potential use-cases for neuromorphic systems, and we must be prepared to accommodate a variety of uses-cases during the development of associated software tools. The development of modular software stacks will be an extremely important component to allow for the usability and accessibility of neuromorphic systems in the future.

Recommendation: Define and develop required supporting software systems for using neuromorphic computers.

1. Define use-cases for neuromorphic systems and what supporting software systems will be required for each use-case.
2. Define a common abstract network representation and instruction sets that the community can adopt and that can be used to build higher-level supporting software systems, such as visualization tools and application-specific software.

E. What applications are most appropriate for neuromorphic computers?



Another major ongoing research question is what applications are most appropriate for neuromorphic computers. There are really two underlying fundamental questions for neuromorphic computing (and perhaps for all non-von Neumann) architectures in terms of applications: what applications ***can*** they solve and what applications ***should*** they solve?

What are the theoretical computational capabilities of neuromorphic systems?

The first question can only be answered with extensive research into theoretical results. For neuromorphic computing architectures, this may be achieved by examining computational complexity classes and looking at Turing computability properties. However, it is likely that each neuromorphic model will have to be examined separately in terms of its computational abilities. It is also likely that an entirely new computational theory paradigm will need to be defined in order to encompass the computational abilities of neuromorphic systems, as they have radically different properties from the traditional von Neumann architecture.

What applications are natural for neuromorphic systems?

Though the theoretical results are certainly important, a perhaps more important (and certainly more immediate) question is what applications are most ***appropriate*** for neuromorphic systems. For example, though it may be possible for a neuromorphic computer to perform precise floating point calculations, it is not a natural application. One obvious set of applications for neuromorphic computers is the set of applications for which artificial neural networks perform well. However, it is important to note that artificial neural networks have undergone 60 years of research for which the architectures and algorithms were adapted and tailored to von Neumann systems. In fully understanding what applications are most appropriate for neuromorphic computers, a fundamental paradigm shift in the way we think about what computers are capable of solving is required. It is tempting to categorize the set of applications that are appropriate for neuromorphic computers as the "applications" that are appropriate for biological neural systems. Though this may be true to some extent, our lack of understanding of biological neural systems

and all of their underlying mechanisms restricts our ability to fully define the capabilities of a neurally inspired computer.

One of the key aspects of most neuromorphic systems is the inclusion of temporal processing capabilities; this is one of the key differentiating features between spiking neural networks (the most commonly implemented neural network model in neuromorphic systems) and deep learning. In determining the set of applications for neuromorphic systems, we should consider this strength and choose applications that will leverage that strength. For traditional artificial neural network type applications, this will likely include processing spatiotemporal data (e.g., clustering or classifying spatiotemporal data) and control problems where a sense of memory and timing in the system is important.

Another key characteristic of biological systems that neuromorphic systems attempt to capture is the ability to process noisy data and the ability to learn from a relatively small set of data (as compared with deep learning systems, which require a large amount of data). For applications that require the ability to make decisions in the face of inadequate or noisy data sources, a neuromorphic system is likely a good candidate.

What are the characteristics of applications for which neuromorphic systems may be appropriate?

To better understand neuromorphic computing systems as a whole, it will be beneficial to define a set of benchmark applications. However, it is of vital importance that these applications represent a diverse problem space. Neuromorphic computing, especially as applied to general applications, is still in its infancy. There is a danger in selecting a small, relatively homogeneous set of benchmark applications because, by defining a "goal set," we may restrict the way the community thinks about neuromorphic systems and their capabilities, as well as how we select models, build devices, and choose training/learning/programming paradigms. It is of vital importance that we, as a community, use benchmarks as tools for comparison purposes but not as driving forces for research. Figure 13 gives some examples of application characteristics for which neuromorphic systems are suitable.

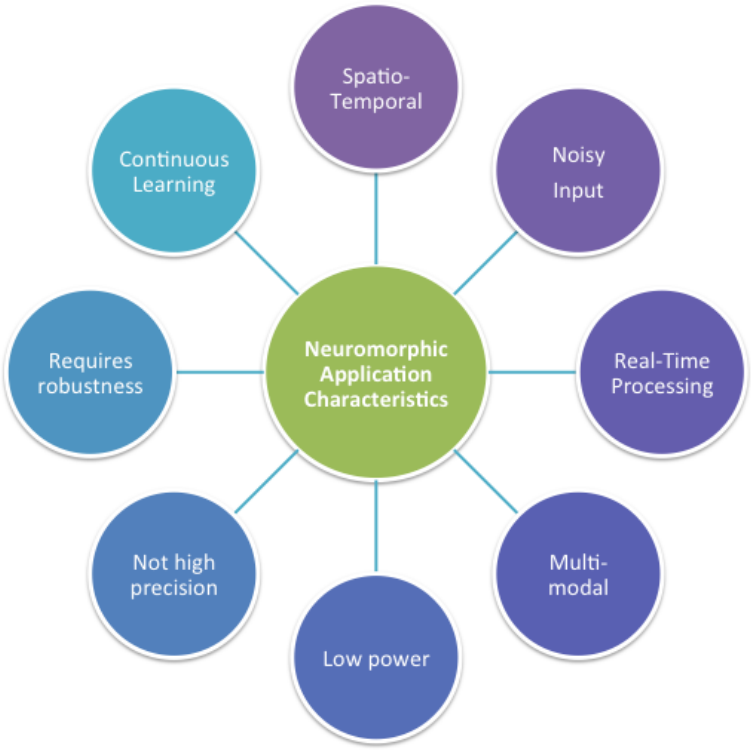


Figure 13: Applications that require devices with one or more of these properties may be well suited for neuromorphic systems.

Defining the set of applications that are most appropriate for neuromorphic systems will also, necessarily, affect device design. For example, we defined several potential use-cases in the introduction: simulation engines for computational neuroscience, co-processors for supercomputers and/or personal computers, custom neuromorphic supercomputers for predefined applications, and in situ computers on sensors or autonomous vehicles in real environments. For each of these use-cases, the particular application will drive the model selection, programming/learning/training paradigm, and device design. As a result, we should also consider the possibility that neural computing should not be a "one-size-fits-all" technology. For example, there is increasing evidence that one can build neural algorithms that leverage neuromorphic hardware at known precisions suitable for accelerating many numerical and scientific computing applications [Severa2016]; however, this type of application may well require a distinct implementation from more biologically inspired applications that may benefit from stochastic implementations or real-time approximate learning capabilities. Perhaps the most interesting use-case (from a computer science perspective), is the use of a neuromorphic computer as a co-processor. In this case, the desired properties of a neuromorphic device include programmable, computationally fast (relative to a traditional CPU or GPU or other co-processor) on a particular problem subset, and relatively low power consumption. Figure 14 gives an overview of some potential uses of neuromorphic devices.

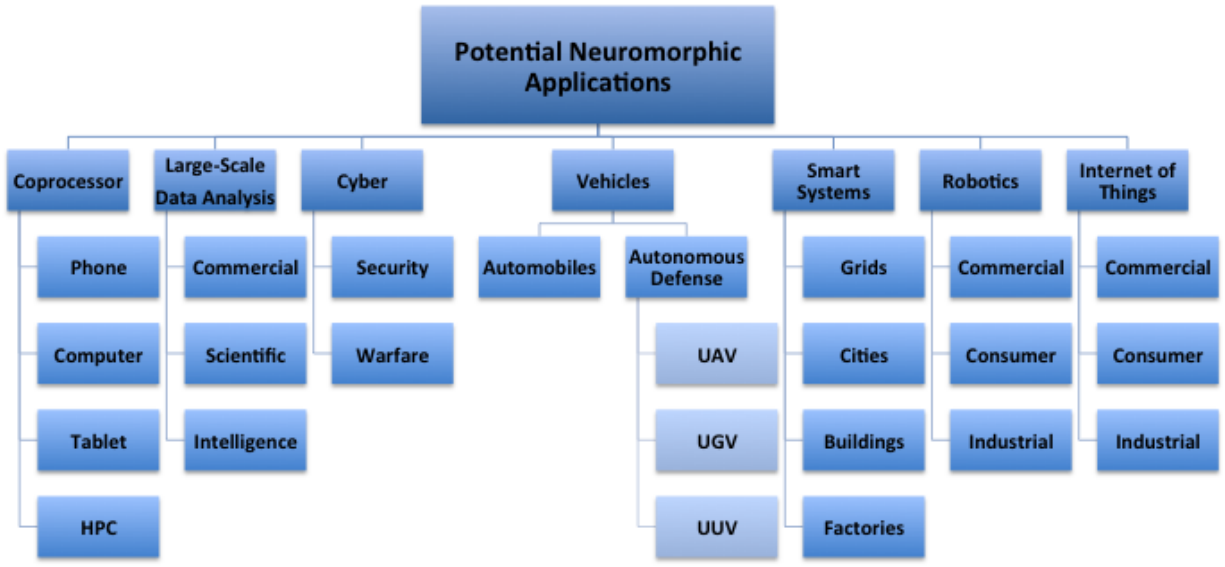


Figure 14: Potential applications for neuromorphic computers [WSP:Hylton].

How can neuromorphic hardware be leveraged in high performance computing clusters?

With the increase of device unreliability, scaling existing computational systems to exascale levels is extremely challenging. Neuromorphic computers have been shown to be more power efficient than traditional computers by several orders of magnitude [Hasan2016]. Using a neuromorphic system as a processing component could make the design of exascale systems more feasible.

The key use of the neuromorphic computing system would be likely be in analyzing and processing large volumes of data. Several studies have shown that neuromorphic algorithms can be used in approximating non-neuromorphic applications to gain significant power savings at minimal loss in accuracy [Chen2012]. Thus when approximation is acceptable, computation could be off-loaded to the neuromorphic hardware in an HPC system to save power. Additionally, neuromorphic hardware could be used to monitor system health at minimal power overhead to predict component failures in an HPC. This could be used to assist schedulers in avoiding potentially problematic computation nodes.

Recommendation: Define a diverse set of benchmark applications for neuromorphic computers.

1. Define a relatively large set (20 or more) of benchmark or target applications for neuromorphic computers that require one or more of the desired characteristics of neuromorphic systems.
2. Make associated data sets and/or application simulation software available to the community so that real comparisons may be between different neuromorphic platforms.

F. How do we build and/or integrate the necessary computing hardware?

The primary focus of this report has been on computing aspects of neuromorphic computing. However, in order to have a truly successful neuromorphic computing program, it will be necessary that we consider the hardware and device components as well. As compared to conventional computing systems, neuromorphic computing systems and algorithms need higher densities of typically lower precision memories operating at lower frequencies. Also, multistate/analog memories offer the potential to support learning and adaptation in an efficient and natural manner. Without efficient hardware implementations that leverage new materials and devices, the real growth of neuromorphic applications will be substantially hindered.

As noted throughout this report, significant effort will be needed on the part of the computing community to work with researchers in the areas neuromorphic materials, devices, and circuitry. Neuromorphic systems will not and cannot be developed by a single field in isolation. The decisions we make at the model and algorithm level should both inform and be informed by what occurs at the materials, device, and circuitry levels. DOE has capabilities at all levels of neuromorphic computing (models, algorithms, circuitry, devices, and materials) that can and should be leveraged together.

Recommendation: **Enable a national fabrication capability to support the development and technology transition of neuromorphic materials, devices, and circuitry that can be integrated with state-of-the-art CMOS into complete and functional computing systems.**

1. Determine the appropriate neuromorphic materials, devices, and circuitry that need to be supported at such a foundry.
2. Make the fabrication foundry available to a wide user base of researchers and developers of neuromorphic processors as an effective method to develop complete neuromorphic systems by leveraging and sharing common infrastructure and interfaces.
3. Ensure that software for effective emulation of new computing architecture components is readily available to enable the scientific community to explore new computing approaches.

IV. Intermediate Steps

There are several intermediate steps that the community can take to begin to address these challenges.

1. **Large-Scale Simulators:** The first short-term goal is the development of large-scale simulations that are capable of supporting different neuromorphic architectures at different levels of abstraction. High-level simulations will be important for examining and testing different computational building blocks, as well as connectivity patterns and structural programmability requirements. High-level simulations can be used to develop and test theoretical results, as well as to help develop algorithms for training and learning. Insights gained from these simulations can also inform device design. Lower-level simulations are also likely to be important. Large-scale simulations at the device level, circuit level, or even materials level can be used in concert with

small-scale prototypes of devices to provide insight as to the capabilities and characteristics of the system, including energy efficiency estimates.

2. **Algorithms:** A second intermediate goal is to delve deeply into the development of algorithms specifically for neuromorphic system. In doing so, we need to understand the fundamental characteristics of each system, as well as how they operate. As a first step, we may continue to adapt existing training/learning methods so that they are appropriate for neuromorphic computers. However, we cannot continue to rely on off-line and off-chip training and learning; we must be willing to allow for suboptimal performance of the resulting algorithms, at least in the initial development stages. It is also worthwhile to begin to form some theoretical foundations of what it means for a system to be intelligent. In doing so, we may look to neuroscience, nonlinear dynamics, and theories of the mind.

3. **Hardware Development:** A third intermediate goal is to develop neuromorphic hardware, including both computation building blocks and system architecture. It requires an integrated effort with a good understanding of not only the material science, device process, and circuit design but also the neuroscience and software engineering. We expect complete and functional computing systems by leveraging new materials, devices, and novel circuitry and architecture. The investigation and prototype of system integration may be necessary. It is also worthwhile to enable a fabrication foundry available to a wide user base of researchers and developers of neuromorphic processors by leveraging and sharing common infrastructure and interfaces.

4. **Supporting Software:** A fourth intermediate goal is the development of supporting software for both simulation software and neuromorphic hardware. It is of vital importance that this software is developed in conjunction with the simulation software, rather than waiting for the devices to be made available. Tools such as visualizations and user interfaces can be used to debug the software and hardware itself throughout development and ease the interaction for algorithm developers.

5. **Benchmark Suite:** A fifth intermediate goal is to define a set of benchmarks for neuromorphic systems in order to test the relative strengths and weaknesses of various neuromorphic computing approaches. It is of vital importance that this set of benchmarks accurately reflects the range of capabilities and characteristics of neuromorphic systems (Figure 15). Benchmarks for neural network systems, such as classification, control, and anomaly detection, should be included in the set of benchmarks. At least one neuroscience-specific benchmark (such as generating particular spiking patterns) should also be included, to encompass neuromorphic systems built specifically to accurately simulate biological neural systems. It is also likely worthwhile to include at least one non-neural network and non-neural-inspired benchmark, such as running a traditional graph algorithm. One goal in defining the set of benchmarks is that they should be diverse enough that they will not directly drive, and thus possibly restrict, the creativity in the development of neuromorphic systems. As part of defining a set of benchmarks, relevant data sets and simulation codes should be made available for the community, so that fair comparisons may be made between neuromorphic systems. As such, building the data sets and simulation software will require effort.

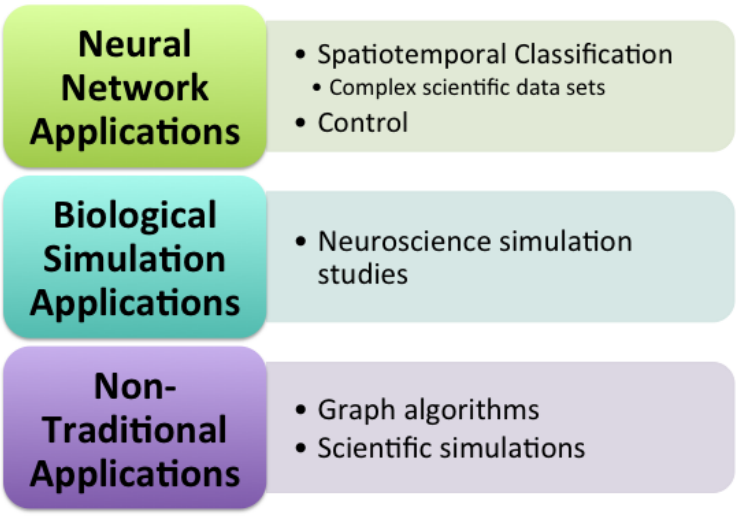


Figure 15: Example benchmark application areas.

6. **Metrics:** A sixth intermediate goal is the development and selection of appropriate metrics for measuring performance and comparing different neuromorphic systems. This step will need to be accomplished simultaneously with benchmark selection. Neuromorphic systems are almost always measured in terms of application-specific metrics (such as accuracy for classification tasks). Though application-specific metrics are important (and should be defined alongside the set of benchmarks), it is equally important that other metrics be defined as well. A power efficiency metric will also be important to define. Most reported results for neuromorphic computing devices include some power metric, but it is also important to include energy consumption and power efficiency of supporting software running on traditional architectures as well as communication costs when considering the true power cost of a neuromorphic system. Another major metric to consider is the computation metric similar to FLOPS for traditional architectures; this will likely be the hardest metric to define because different models will have significantly different operating characteristics. It may also be worthwhile to consider other metrics, such as size of device (footprint), scalability, and programmability/flexibility. By defining a common set of metrics and benchmarks, we may begin to quantify which neuromorphic models and devices are best for a particular type of application.

For each of these intermediate steps, it is important that the computing community develop strong interactions with other key fields in the neuromorphic computing: the neuroscience community, the hardware/device development community, and the materials community.

V. Long-Term Goals

Four major areas within neuromorphic computing emerged as the most important to focus on for long-term success.

1. **Learning:** The development of general purpose learning and training methods for neuromorphic systems is and will continue to be a major goal for neuromorphic computing, and it will likely require long-term investment in order to produce successful systems. It will require a coordinated

effort between computational neuroscientists, researchers with specialties in learning and intelligence, machine learning and artificial intelligence researchers, and algorithm designers in order to produce them.

2. **Theory:** The development of computational theory and theories of intelligence within neuromorphic computing will also be key in neuromorphic systems being accepted by the computing community at large. The development of theoretical results associated with abstract models and learning or intelligence will also drive algorithm development, device design, and materials research of the future. With a handle on the theoretical capabilities of the system, we may inspire the development and innovation of new neuromorphic systems for the next several decades.

3. **Large-scale coordinated effort:** One of the major conclusions reached during the workshop was that neuromorphic computing research in the United States as a whole and in DOE in particular would benefit from a large-scale, coordinated program across all of the associated disciplines. Such a neuromorphic computing program for the United States was proposed by Stan Williams during our workshop (Figure 16). The first four proposed elements of this large-scale neuromorphic computing program fall neatly in line with the major research questions proposed in this report, as well as the proposed short-term and long-term goals we have identified for neuromorphic computing moving forward. It is vital that we consider the broader picture of the community when addressing these goals.

4. **Applications:** As data sets have continued to grow in recent years, the long-term prognosis for data analysis across a broad range of applications appears problematic. In particular, scientific data sets are typically very large with numerous, dense, and highly interrelated measurements. One primary characteristic of many of these scientific data sets is spatial and/or temporal locality in the observation space. In other words, it is rare that any two observations will be independent in both time and space, and proximity of any two measurements in time and/or space is highly relevant to an analysis of the data. Furthermore, the scale of the data is typically so immense that many separate measurements are almost inevitably represented as an average, sacrificing detail for tractability. A neuromorphic computing paradigm may be ideally suited to address these challenges through the ability to leverage both spatial and temporal locality. It is imperative that we investigate this promising opportunity.

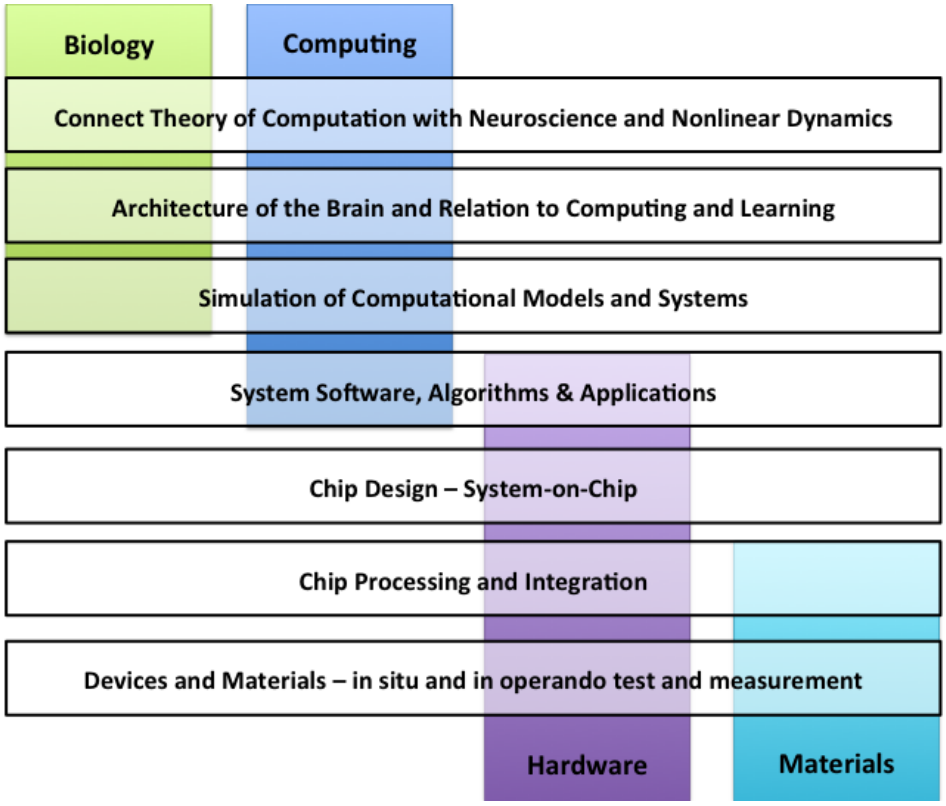


Figure 16: Large-scale neuromorphic computing program, as proposed by Stan Williams [WSP:Williams].

VI. Conclusions

Overall, there are several major conclusions of our workshop:

1. **Importance of simulators, benchmarks, and metrics:** There are many different approaches to neuromorphic models, architectures, devices, and algorithms. We need a way to study (via simulation) and evaluate (via metrics and benchmark definitions) these systems in a meaningful way.

2. **Paradigm shift for programming:** A fundamental shift in the way we think about programming and computing algorithm development is going to be required. We need to develop theories of learning and intelligence and understand how they will be applied to neuromorphic computing systems.

3. **Focus on innovation, not applications:** Applications are important, but particular applications should not be the driving focus in the development of a neuromorphic computer. Though benchmarks are important for evaluation, they should not limit or restrict development.

4. **Move away from brain emulation:** The goal of a neuromorphic computer should not be to emulate the brain. We should instead take inspiration from biology but not limit ourselves to particular models or algorithms, just because they work differently from their corresponding

biological systems, nor should we include mechanisms just because they are present in biological systems.

5. **Large-scale coordination:** The community would greatly benefit from a large-scale coordinated effort, as well as a neuromorphic "hub" through which we may openly share successes and failures, supporting software, data sets and application simulations, and hardware designs.

DOE Office of Science and ASCR are well positioned to address the major challenges in neuromorphic computing for the following three important reasons.

1. **World-class scientific user facilities:** We have access to and experience using world-class scientific user facilities, such as HPC systems, which will be invaluable in studying large-scale simulations of neuromorphic systems, and materials science user facilities, which will be invaluable in providing insight into the properties of materials to build neuromorphic systems.

2. **World-class researchers:** We have researchers in each of the major focus areas of the community, including biology, computing, devices, and materials, who have experience in building cross-disciplinary collaborations.

3. **World-class science problems:** We have world-class science problems for which neuromorphic computing may apply. Without DOE involvement, it is likely that neuromorphic computing will be limited to commercial applications.

It is clear that neuromorphic computing will play a critical role in the landscape of future computing. The feasibility of such systems has been demonstrated, but there are still major research questions that need to be resolved in the development and application of these systems. DOE can fill an important national role in enabling solutions that address these important basic research and development challenges.

References

[Backus1978] Backus, John. "Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs." Communications of the ACM21.8 (1978): 613–641.

[Benjamin2014] Benjamin, Ben Varkey, et al. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations." Proceedings of the IEEE 102.5 (2014): 699–716.

[Brüderle2011] Brüderle, Daniel, et al. "A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems." Biological cybernetics 104.4-5 (2011): 263–296.

[Caporale2008] Caporale, N. and Y. Dan. "Spike timing-dependent plasticity: a Hebbian learning rule." Annu. Rev. Neurosci. 31 (2008): 25–46.

[Chen2012] Tianshi Chen, Yunji Chen, Marc Duranton, Qi Guo, Atif Hashmi, Mikko Lipasti, Andrew Nere, Shi Qiu, Michele Sebag, Olivier Temam. "BenchNN: On the Broad Potential Application Scope of Hardware Neural Network Accelerators," Proceedings of the IEEE International Symposium on Workload Characterization, November 2012.

[Chua1971] Chua, Leon. "Memristor-the missing circuit element." IEEE Transactions on circuit theory 18.5 (1971): 507–519.

[Davison2009] Davison, Andrew, et al. "PyNN: a common interface for neuronal network simulators" (2009).

[Furber2013] Furber, Steve B., David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple, and Andrew D. Brown. "Overview of the SpiNNaker system architecture." IEEE Transactions on Computers 62.12 (2013): 2454–2467.

[Gerstner2002] Gerstner, Wulfram, and Werner M. Kistler. Spiking neuron models: Single neurons, populations, plasticity. Cambridge University Press, 2002.

[Hasan2016] Hasan, R., T. M. Taha, C. Yakopcic, and D. Mountain, "High Throughput Neural Network based Embedded Streaming Multicore Processors," Proceedings of the IEEE International Conference on Rebooting Computing, October 2016.

[Hawkins2016] Hawkins, J. Biological and Machine Intelligence. Release 0.4, 2016. Accessed at http://numenta.com/biological-and-machine-intelligence/.

[Hennessy2011] Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.

[Hodgkin1952] Hodgkin, A.L. and Huxley, A.F. "A quantitative description of membrane current and its application to conduction and excitation in nerve." The Journal of physiology 117.4 (1952): 500.

[Izhikevich2003] Izhikevich, E.M. "Simple model of spiking neurons." IEEE Transactions on neural networks 14.6 (2003): 1569–1572.

[Jouppi2016] Jouppi, Norm. Google supercharges machine learning tasks with tpu custom chip, May 2016. URL: https://cloudplatform.googleblog.com/2016/05/Google- supercharges-machine-learning-tasks-with-custom-chip.html.

[LeCun2015] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature521.7553 (2015): 436–444.

[McCulloch1943] McCulloch, W.S. and W. Pitts. "A logical calculus of the ideas immanent in nervous activity." The bulletin of mathematical biophysics 5.4 (1943): 115–133.

[Mead1990] Mead, Carver. "Neuromorphic electronic systems." Proceedings of the IEEE78.10 (1990): 1629–1636.

[Merolla2014] Merolla, Paul A., et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface." Science 345.6197 (2014): 668–673.

[Nervana2016] Nervana. Nervana engine, 2016. URL: https://www.nervanasys.com/technology/engine/.

[Rabinovich2006] Rabinovich, M.I., P. Varona, A. I. Selverston, and H. D. Abarbanel. "Dynamical principles in neuroscience." Reviews of modern physics 78(4) (2006): 1213.

[Rothganger2014] Rothganger, F., C.E. Warrender, D. Trumbo, and J.B. Aimone. "N2A: a computational tool for modeling neurons to algorithms." Frontiers in Neural Circuits 8.1 (2014)

[Severa2016] Severa, W., O. Parekh, C.D. James, and J.B. Aimone. “Spiking Network Algorithms for Scientific Computing” – Proceedings of the IEEE International Conference on Rebooting Computing, October 2016.

[Shalf2015] Shalf, John M., and Robert Leland. "Computing beyond Moore's Law." Computer 48.12 (2015): 14–23.

[Shen2016] Shen, Juncheng, et al. "Darwin: a neuromorphic hardware co-processor based on Spiking Neural Networks." Science China Information Sciences59.2 (2016): 1–5.

[Vogelstein2007] Vogelstein, R.J., U. Mallik, J.T. Vogelstein, and G. Cauwenberghs. “Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses.” Neural Networks, IEEE Transactions on 18(1) (2007): 253–265.

Workshop Presentation References

[WSP:Aimone] James Aimone, Kristofor Carlson, and Fredrick Rothganger. “Neural Computing: What Scale and Complexity is Needed?”

[WSP:Cao] Yu Cao, Steven Skorheim, Ming Tu, Pai-Yu Chen, Shimeng Yu, Jae-Sun Seo, Visar Berisha, Maxim Bazhenov, and Zihan Xu. “Efficient Neuromorphic Learning with Motifs of Feedforward Inhibition.”

[WSP:Chen] Yiran Chen. “Algorithm Innovations of Enhancing Scalability and Adaptability of Learning Systems.”

[WSP:Humble] Kathleen Hamilton, Alexander McCaskey, Jonathan Schrock, Neena Imam, and Travis Humble. “Associative Memory Models with Adiabatic Quantum Optimization.”

[WSP:Hylton] Todd Hylton. “Perspectives on Neuromorphic Computing.”

[WSP: Kudithipudi] Dhireesha Kudithipudi, James Mnatzaganian, Anvesh Polepalli, Nicholas Soures, and Cory Merkel. “Energy Efficient and Scalable Neuromemristive Computing Substrates.”

[WSP:Marinella] Matthew J. Marinella, Sapan Agarwal, A. Alec Talin, Conrad D. James, and F. Rick McCormick. “Device to System Modeling Framework to Enable a 10 fJ per Instruction Neuromorphic Computer.”

[WSP:Mohsenin-Koushanfar] Tinoosh Mohsenin and Farinaz Koushanfar. “Bringing Physical Dimensions to the Deep Networks for Neuromorphic Computing.”

[WSP:Panda] Priyadarshini Panda and Kaushik Roy. “Enabling on-chip intelligence with low-power neuromorphic computing.”

[WSP: Parker] Alice Parker. “Object Recognition and Learning using the BioRC Biomimetic Real-Time Cortex Neurons.”

[WSP:Rose] Gangotree Chakma, Elvis Offor, Mark Dean, and Garrett Rose. “A Reconfigurable Memristive DANNA Circuit with Implementations in Pattern Recognition.”

[WSP:Saxena] Vishal Saxena and Xinyu Wu. “Addressing Challenges in Neuromorphic Computing with Memristive Synapses.”

[WSP:Schuman] Catherine Schuman. “Roadmap for Neuromorphic Computing from the Computer Science Perspective.”

[WSP:Stepp] Praveen Pilly, Nigel Stepp and Jose Cruz-Albrecht. “Exploiting Criticality in HRL's Latigo Neuromorphic Device.”

[WSP:Taha] Tarek Taha, Raqibul Hasan, and Chris Yakopcic. “Energy Efficiency and Throughput of Multicore Memristor Crossbar Based Neuromorphic Architectures.”

[WSP:Williams] R. Stanley Williams. “A Nanotechnology-Inspired Grand Challenge for Future Computing.”

[WSP:Wolfe] Chris Carothers, Noah Wolfe, Prasanna Date, Mark Plagge, and Jim Hendler. “Large-Scale Hybrid Neuromorphic HPC Simulations, Algorithms and Applications.”

[WSP:Xie] Yuan Xie. “Architecture, ISA support, and Software Toolchain for Neuromorphic Computing in ReRAM Based Main Memory.”

[WSP:Yanguas-Gil] Angel Yanguas-Gil. “Beyond the crossbar: materials based design and emulation of neuromemristive devices and architectures.”

Workshop Participants

Organizing Committee

Thomas Potok, Oak Ridge National Laboratory
Catherine Schuman, Oak Ridge National Laboratory
Robert Patton, Oak Ridge National Laboratory
Todd Hylton, Brain Corporation
Hai Li, University of Pittsburgh
Robinson Pino, U.S. Department of Energy

Program Committee

Brad Aimone, Sandia National Laboratories
Frank Alexander, Los Alamos National Laboratory
Mike Berry, University of Tennessee
Jim Brase, Lawrence Livermore National Laboratory
Yiran Chen, University of Pittsburgh
Mark Dean, University of Tennessee
Hal Finkel, Argonne National Laboratory
Mark Greaves, Pacific Northwest National Laboratory
Giacomo Indiveri, University of Zurich
Garrett Kenyon, Los Alamos National Laboratory
Kerstin Kleese van Dam, Brookhaven National Laboratory
Dhireesha Kudithipudi, Rochester Institute of Technology
Emre Neftci, University of California, Irvine
Garrett Rose, University of Tennessee

Justin Shumaker, Army Research Laboratory
Tarek Taha, University of Dayton
Brian Van Essen, Lawrence Livermore National Laboratory
Jeff Vetter, Oak Ridge National Laboratory

Attendees

Sapan Agarwal, Sandia National Laboratories
Brad Aimone, Sandia National Laboratories
Esam El-Araby, University of Kansas
Joseph Bebel, University of Southern California
John Douglas Birdwell, University of Tennessee
Grant Bruer, University of Tennessee
Yu Cao, Arizona State University
Joseph Caroli, Air Force Research Laboratory
Gangotree Chakma, University of Tennessee
Xiang Chen, University of Pittsburgh
Yiran Chen, University of Pittsburgh
Son Dao, HRL Laboratories, LLC
Mark Dean, University of Tennessee
Adam Disney, University of Tennessee
Patricia Eckhart, University of Tennessee
Arthur Edwards, Air Force Research Laboratory
William Harrod, U.S. Department of Energy
Travis Humble, Oak Ridge National Laboratory
Todd Hylton, Brain Corporation
Neena Iman, Oak Ridge National Laboratory
Conrad James, Sandia National Laboratories
Ramakrishnan Kannan, Oak Ridge National Laboratory
Garrett Kenyon, Los Alamos National Laboratory
Scott Kolodziej, Texas A&M University
Farinaz Koushanfar, University of California, San Diego
Dhireesha Kudithipudi, Rochester Institute of Technology
Carolyn Lauzon, U.S. Department of Energy
Cindy Leiton, Stony Brook University
Hai Li, University of Pittsburgh
Jeremy Liu, Oak Ridge National Laboratory
Arthur Barney Maccabe, Oak Ridge National Laboratory
Matthew Marinella, Sandia National Laboratories
Cory Merkel, Air Force Research Laboratory
John Mitchell, University of Tennessee
Jamaludin Mohd Yusof, Los Alamos National Laboratory
Tinoosh Mohsenin, University of Maryland, Baltimore County
Priyadarshini Panda, Purdue University

Alice Parker, University of Southern California
Robert Patton, Oak Ridge National Laboratory
Praveen Pilly, HRL Laboratories, LLC
Robinson Pino, U.S. Department of Energy
Raphael Pooser, Oak Ridge National Laboratory
Thomas Potok, Oak Ridge National Laboratory
John Reynolds, University of Tennessee
Garrett Rose, University of Tennessee
Nandakishore Santhi, Los Alamos National Laboratory
Vishal Saxena, Boise State University
Catherine Schuman, Oak Ridge National Laboratory
Angie Scott, Oak Ridge National Laboratory
Devanand Shenoy, U.S. Department of Energy
Susheela Singh, Oak Ridge National Laboratory
Nigel Stepp, HRL Laboratories, LLC
Tarek Taha, University of Dayton
Alec Talin, Sandia National Laboratories
Andrew Valesky, Oak Ridge National Laboratory
Brian Van Essen, Lawrence Livermore National Laboratory
Wuije Wen, Florida International University
Lloyd Whitman, White House Office of Science and Technology Policy
R. Stanley Williams, Hewlett Packard Enterprise
Noah Wolfe, Rensselaer Polytechnic Institute
Austin Wyer, Oak Ridge National Laboratory
Yuan Xie, University of California, Santa Barbara
Angel Yanguas-Gil, Argonne National Laboratory
Aaron Young, University of Tennessee

Workshop Agenda

Wednesday, June 29, 2016

Time	Event	Location
10:30–11:00	Check-In/Badging	Visitor's Center
11:00–12:30	Welcome/Working Lunch: • Tom Potok, ORNL • Shaun Gleason, ORNL • Robinson Pino, DOE	Tennessee A&B
12:30–1:15	Keynote: Todd Hylton, Brain Corporation	Tennessee A&B
1:15–2:00	Keynote: Catherine Schuman, ORNL	Tennessee A&B
2:00–2:45	Keynote: Cindy Leiton, Stony Brook University	Tennessee A&B
2:45–3:30	Break	Lobby & Tennessee C
3:30–4:00	Presentation: Lloyd Whitman, White Office of Science and Technology Policy	Tennessee A&B
4:00–5:00	Break-Out Discussions: Architectures, Algorithms, Applications	Architectures: Tennessee A&B, Algorithms: Cumberland Applications: Emory
5:00–5:15	Summary	Tennessee A&B

Thursday, June 30, 2016

Time	Event	Location
8:00–8:30	Coffee/Light Breakfast	Lobby & Tennessee C
8:30–8:40	Welcome/Recap	Tennessee A&B
8:40–10:00	Short Presentations: • James Aimone, Kristofor Carlson and Fredrick Rothganger: Neural Computing: What Scale and Complexity is Needed? • Alice Parker: Object Recognition and Learning using the BioRC Biomimetic Real-Time Cortex Neurons • Kathleen Hamilton, Alexander McCaskey, Jonathan Schrock, Neena Imam and Travis Humble: Associative Memory Models with Adiabatic Quantum Optimization • Tinoosh Mohsenin and Farinaz Koushanfar: Bringing Physical Dimensions to the Deep Networks for Neuromorphic Computing	Tennessee A&B
10:00–10:30	Break	Lobby & Tennessee C

Time	Event	Location
10:30–12:00	Short Presentations: • Yuan Xie: Architecture, ISA support, and Software Toolchain for Neuromorphic Computing in ReRAM Based Main Memory • Angel Yanguas-Gil: Beyond the crossbar: materials based design and emulation of neuromemristive devices and architectures • Matthew J. Marinella, Sapan Agarwal, A. Alec Talin, Conrad D. James and F. Rick McCormick: Device to System Modeling Framework to Enable a 10 fJ per Instruction Neuromorphic Computer • Chris Carothers: Large-Scale Hybrid Neuromorphic HPC Simulations, Algorithms and Applications	Tennessee A&B
12:00–1:00	Working Lunch and Plenary: Stan Williams, HP	Tennessee A&B
1:00–2:30	Short Presentations: • Priyadarshini Panda and Kaushik Roy: Enabling on-chip intelligence with low-power neuromorphic computing • Yu Cao, Steven Skorheim, Ming Tu, Pai-Yu Chen, Shimeng Yu, Jae-Sun Seo, Visar Berisha, Maxim Bazhenov and Zihan Xu: Efficient Neuromorphic Learning with Motifs of Feedforward Inhibition • Praveen Pilly, Nigel Stepp and Jose Cruz-Albrecht: Exploiting Criticality in HRL's Latigo Neuromorphic Device • Yiran Chen: Algorithm Innovations of Enhancing Scalability and Adaptability of Learning Systems	Tennessee A&B
2:30–3:00	Break	Lobby & Tennessee C
3:00–3:45	Lightning Talks: • Vishal Saxena and Xinyu Wu: Addressing Challenges in Neuromorphic Computing with Memristive Synapses • Tarek Taha, Raqibul Hasan and Chris Yakopcic: Energy Efficiency and Throughput of Multicore Memristor Crossbar Based Neuromorphic Architectures • Dhireesha Kudithipudi, James Mnatzaganian, Anvesh Polepalli, Nicholas Soures, and Cory Merkel: Energy Efficient and Scalable Neuromemristive Computing Substrates • Gangotree Chakma, Elvis Offor, Mark Dean and Garrett Rose: A Reconfigurable Memristive DANNA Circuit with Implementations in Pattern Recognition	Tennessee A&B
3:45–5:00	Presenter Panel Discussion, moderated by Mark Dean, University of Tennessee	Tennessee A&B
5:00–5:15	Summary	Tennessee A&B

Friday, July 1, 2016

Time	Event	Location
7:30–8:00	Coffee/Light Breakfast	Lobby & Tennessee C
8:00–10:00	Breakout Discussions	Tennessee A&B, Tennessee C, Cumberland, Emory
10:00–10:30	Break	Lobby & Tennessee C
10:30–11:45	Breakout Discussions	Tennessee A&B, Tennessee C, Cumberland, Emory
11:45–12:00	Final Wrap-Up	Tennessee A&B