A Unified Hardware/Software Co-Design Framework for Neuromorphic Computing Devices and Applications

James S. Plank, Garrett S. Rose, Mark E. Dean
EECS Department
University of Tennessee
Knoxville, TN 37996
[jplank,garose,markdean]@utk.edu

Catherine D. Schuman
Computational Data Analytics
Oak Ridge National Laboratory
Oak Ridge, TN 37830
schumancd@ornl.gov

Natheniel C. Cady
CNSE
SUNY Polytechnic Institute
Albany, NY 12203
ncady@sunypoly.edu

***Abstract*—With the death of Moore's law, the computing community is in a period of exploration, focusing on novel computing devices, paradigms, and techniques for programming. The TENN-Lab group has developed a hardware/software co-design framework for this exploration, on which we perform research with three thrusts:**

1) **Devices for computing, such as memristors and biomimetic membranes.**
2) **Applications that employ spiking neural networks for processing.**
3) **Machine learning techniques to program.**

The design framework is unified, because it allows all three thrusts to work in concert, so that, for example, new results on device design can apply instantly to the current results of applications and learning. In this paper, we detail the interweaving components of the design framework. We then describe case studies on each of the research thrusts above, highlighting how the unified framework is enabling to each case study.

I. Introduction

The computing community is currently in a period of exploration. Moore's law has died, and as such, researchers are exploring alternatives to the von Neumann computing paradigm to achieve improved computational performance, lower power consumption, and novel functionalities. Notable products from this exploration are novel new architectures for computation [1], [2], [3], [4], [5], theories on biologically inspired computation [6], [7], and algorithmic approaches to problems that are more successful at, for example, image classification, than previous approaches [8], [9].

At the University of Tennessee and Oak Ridge National Laboratory, we have developed a research group called TENN-Lab (Laboratory of Tennesseans Exploring Neuromorphic Networks). We focus on Neuromorphic computing, specifically spiking neural networks, as the basic vehicle for our research. Our goal is to push the research boundaries of three important areas in the search for novel computing paradigms and implementations:

1) Designing new architectures for computing, based on conventional and novel computing devices.
2) Developing new applications for these architectures.
3) Using machine learning techniques to program.

In this paper, we describe our approach, specifically the hardware/software co-design framework that allows us to explore these three areas simultaneously. Our framework is based on a general model of spiking neural networks, so that we may develop applications generally, without having the applications concern themselves with the specifics of a neural network or neuromorphic implementation. Our neural network architectures export this general interface, but each implements functionality specific to the devices on which it is implemented. For example, our mrDANNA architecture (see Section IV-A) has been designed to leverage implementation by novel memristive devices, and it has been implemented within our general model, so that our applications may execute upon it without modification. We program the applications with a combination of supervised learning (via genetic algorithms) and unsupervised learning (via Spike Timing Dependent Plasticity: STDP), and are performing active research in these areas, so that we may widen the scope, applicability and effectiveness of our architectures and applications.

In this paper, we present three case studies of our approach, showcasing each of the three research thrusts. The first of these is the development of mrDANNA as a neural network architecture inspired by memristive computing devices (Section IV-A). The second presents an application called RoboNAV, a self-contained, self-navigating vehicle that uses neuromorphic architectures to direct its exploration of spaces while avoiding obstacles (Section IV-B). The third presents a structure-based enhancement to learning, called *Output Redun-*

This research was supported in part by an Air Force Research Laboraty Information Directorate grant (FA8750-16-1-0065). This manuscript has been authored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

dancy, to improve the effectiveness of the genetic component of learning (Section IV-C). These case studies highlight the interoperability of our environment, and how each research thrust improves the other.

II. Related Work

An overview of the field of neuromorphic computing is available in [10]. Design and software frameworks have been developed for some fully fleshed out neuromorphic systems, including SpiNNaker [1], IBM's TrueNorth [2], and FACETS/BrainScaleS [3], but beyond programming languages such as PyNN [11], there is little that enables the general co-design of hardware and software for neuromorphic systems.

In this work, we describe how we utilize our co-design framework for device research, application research, and learning research. For device research, we show how our co-design framework has been used to develop a hardware implementation based on hafnium-oxide memristors. Memristors have emerged in recent years as a common circuit component included in neuromorphic systems [12], [13], [14], [15]. Although metal-oxide based memristors have been common, including those based on hafnium oxide as in our implementation [16], titanium oxide [17], and tungsten oxide [18], a variety of other types of materials are also being used to implement memristors, including chalcogenide memristors [19] and polymer and organic memristors [20]. Though we limit ourselves to hafnium oxide memristors in this work, this co-design framework can easily be applied to other memristive implementations.

For our applications research example, we demonstrate the use of our co-design framework to develop an application in autonomous robot navigation with obstacle avoidance. Robotics have been a common application area for neuromorphic systems, because they often require very small and very power efficient controls systems. Navigation has been a popular choice among other neuromorphic systems, including autonomous driving [21], maze navigation [22], and obstacle-avoidance [23], [24].

For our learning research example, we employ a genetic algorithm to train neuromorphic systems, and we explore approaches in which we can utilize a single simulation run to perform a complex fitness evaluation by utilizing "Output Redundancy." Genetic algorithms and other evolutionary approaches have been utilized to train a variety of both neural networks and neuromorphic systems [25]. The application of genetic algorithms to neural networks (sometimes called neuroevolution) has been used to determine parameters of networks (such as the weight values of synapses) and/or network topology [26], [27], [28], [29]. For neuromorphic systems, genetic algorithms and other evolutionary approaches are attractive because they can work within the constraints of the neuromorphic system and train directly to a particular implementation or device type. They have been employed for a variety of neuromorphic implementations, including programmable devices [30], [31], custom devices [32], and memristor-based implementations [33].

III. Description of the Co-Design Framework

The basic components of our co-design framework are in Figure 1. At the center is a software core, which defines interfaces that allow the other three components to interoperate, and implements functionalities common across components. Examples of the interfaces are primitives for neural network creation, modification and serialization. The network creation primitive allows an application to express its complexity, in terms of number of input/output neurons and hidden neurons, and have each device/architecture implement network creation in accordance with those parameters in a manner that is consistent with the device and architecture. The network modification primitives allow the genetic algorithms to direct optimization by utilizing reproductive operators such as mutation and crossover, each of which is implemented by the device/architecture in a manner that makes sense to the specifics of the device and architecture. The serialization primitive allows the genetic algorithms to store populations of networks, so that the applications may utilize the best ones once the supervised learning phase is complete.

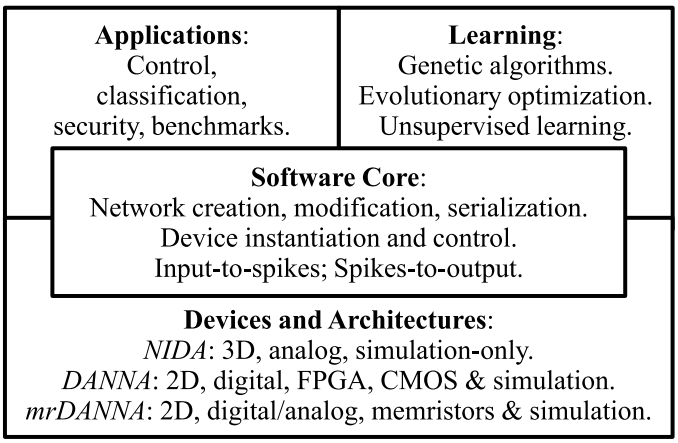

Fig. 1. The components of our hardware/software co-design framework.

A second set of interfaces governs the operation of the neuromorphic devices. For example, an application may allocate an instance of a device and load a network from a serialization. Then, the application drives the device, specifying input (see the next paragraph for a description), running the device, and then interpreting output. The same interface is used for both device simulations and physical implementations of the devices. The interface allows for multiple simultaneous device instantiations, so that the genetic algorithms may evaluate large populations of networks in parallel.

Finally, the core implements functionality to facilitate input to and output from the neuromorphic devices. An application may wish to express its input as a numeric value. The core allows the application to define the range of its input values, and how each input will be encoded for the device. For example, the input may be *rate coded* so that large values are converted to many pulses and small values are converted to few pulses; or the input may be *binned*, so that specific

input neurons are associated with specific ranges of values; or the input values may be converted to specific amounts of charge, when applied to the device. Our core supports these techniques and their combinations. After defining the input coding, the application simply calls a core procedure when it wants to apply input, and the proper encoding is applied. Interpreting output is similar. The core implements voting, counting and binning as techniques to extract boolean, integer and set outputs from the neuromorphic device.

A. Devices and Architectures

We have developed three neuromorphic architectures within this framework. They share the following inspirations from the brain: *Neurons* that accumulate charge and fire upon reaching a threshold, *synapses* that transmit charge temporally from neuron to neuron, and *plasticity* that allows the strengths of synapses to grow and shrink according to their influence on neuron firing. They differ in how they store and transmit charge, their connectivity, and how they are implemented.

The first of these is named NIDA (Neuroscience-Inspired Dynamic Architecture) [25] and features analog neurons that are laid out in three dimensions. Synapses may connect any pair of neurons, and their delays are defined by the distance between the neurons they connect. Multiple models of leak, potentiation and depression are supported by NIDA [34]. It is implemented in simulation with support for visualization.

The second architecture is called DANNA (Dynamic Adaptive Neural Network Array) [35] and is much more constrained than NIDA. It is composed of a two-dimensional array of programmable elements, where each element may be a neuron or a synapse. Neuron thresholds, synaptic weights and synaptic delays are all programmable, digital quantities, and any element (a neuron or synapse) may only connect to its 16 nearest neighbors (in the 8 compass directions). DANNA also supports multiple models of potentiation and depression for unsupervised learning. We have implemented DANNA in simulation and on FPGA's. The largest DANNA FPGA grid that we have tested is 75 X 75, which we have verified to be cycle-accurate with the simulator. We have a CMOS design for DANNA, and are fabricating a 5 X 5 test chip in late 2017.

The third architecture is mrDANNA, described in detail in Section IV-A. We are currently performing the experiments to develop a fourth architecture based on biomimetic membranes for the computing fabric [36].

B. Applications

Our applications fall into four categories: Control, classification, security and benchmarks. We describe a control application in Section IV-B. Other control applications are a simulation of an inverted pendulum on a cart, the cellphone game "Flappy Bird," and multiple navigation engines similar to the one described in Section IV-B. Our classification applications train on labeled data from multi-dimensional data sets such as those found in the UCI Machine Learning Repository. Their performance is as good as, or superior to other machine learning projects on popular labeled data sets such as Iris flower identification and Wisconsin Breast Cancer diagnosis [37]. The security applications are in the domains of packet rate determination and low-power sensors, and the benchmark applications perform simple and composable tasks such as digital operations, counting and voting.

C. Learning

Our machine learning methodology is a combination of supervised and unsupervised learning. The supervised learning relies on a genetic algorithm approach, where both parameters (e.g., neuron thresholds, synapse weights) and neuron/synapse structures are modified in evolutionary optimization. In particular, the crossover operation can take two parent networks, split each in half, and combine the two sets of halves, structurally, into two child networks.

When a network is generated from the genetic algorithm, the plasticity of the architecture allows it to undergo unsupervised learning. In this phase, repetition of input may be employed to allow synapses to strengthen as they cause neurons to fire, or to weaken when their firings are inconsequential. We have demonstrated the effectiveness of unsupervised learning on a benchmark application (XOR) on mrDANNA [38].

IV. Case Studies

In this section, we detail three case studies of research projects in each of the three areas of our co-design framework.

A. Device Research: Development of mrDANNA

Memristors are two-terminal devices whose resistance depends the current/voltage operational history of the device. Memristors encompass a broad category of devices, including phase change memory and a variety of conductive filament based devices which have also been termed resistive memory, RRAM, or ReRAM. While the mechanism of switching varies between the various types of memristors, the fundamental properties of these devices include non-volatility and repeatable modulation of resistance state based on current and/or voltage changes. The majority of memristors are based on a metal-insulator-metal (MIM) structure, where the insulator is a transition metal-oxide. Although many transition metal-oxide MIM devices exhibit resistive switching behavior, their manufacturability, endurance, data retention, and multi-level programmability vary widely.

Our application of memristors is to emulate synaptic functionality in neuromorphic circuits. For example, during the learning phase of neural network generation, the relative strength of connections (synapses) in the network must be encoded in some form of memory. Memristors can encode these synaptic weights with much higher density than conventional SRAM or DRAM. Further, some memristive devices have exhibited multi-level, analog switching behavior.

One major challenge is to integrate memristors seamlessly with CMOS-based integrated circuits. To meet this need, we have developed CMOS-integrated RRAM that can function as memristors using a standard 65nm process technology. RRAM elements are implemented at the interface between the

front-end transistor contacts and back-end metalization layers. This creates RRAM in a 1-transistor/1-memristor (1T1R) configuration, where each memristor is connected in series to a transistor. This transistor serves as a current limiting device, as well as an addressing element.

These CMOS-integrated RRAM have displayed excellent switching characteristics (2V and lower operation), high endurance (over 1 billion cycles), and multi-level operation (up to 10 resistance levels per RRAM cell). By using short pulse-based biasing ($<$ 10 ns pulses), the resistance of these devices can be incrementally increased or decreased, allowing for the multi-level operation necessary for neuromorphic computation [39].

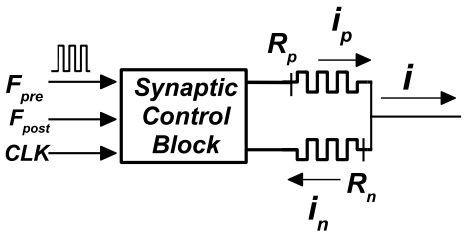

Fig. 2. Schematic of a twin-memristor structure for synapses with positive and negative weights.

We are currently implementing CMOS/memristor hybrid circuits to construct a memristive dynamic adaptive neural network array, or *mrDANNA* [40]. MrDANNA traces its lineage from NIDA and DANNA. A schematic of one of our synapse structures is shown in Figure 2. This synapse exploits two memristors for positive and negative weights. Using the multi-level capability of our memristor-based RRAM cells, the weights of synapses can be incrementally adjusted online and in real-time, following learning rules for long term potentiation and depression (LTP/LTP). More specifically, if a post-synaptic firing event occurs just after a corresponding pre-synaptic firing event, then LTP is triggered and the synaptic weight increases. Similarly, if a post-synaptic firing event happens to occur simultaneous to a pre-synaptic fire, then the synaptic weight decreases due to LTD.

Figure 3 shows Cadence Spectre simulation results for LTP and LTD events for an example 2-input mrDANNA network. When the input synapses F_{pre1} and F_{pre2} fire right before causing the post-synaptic neuron (F_{post}) to fire, their weights (G_{eff1} and G_{eff2}) potentiate. When they fire simultaneously, or just afterwards, they depress. There are two events in this picture (roughly times 0.2 and 0.6) where multiple firings both potentiate and depress the synapses' weights.

Using electrical characteristics measured from the CMOS-integrated RRAM, we have developed a memristor device model for circuit-level SPICE simulations. This device-level model and corresponding circuit designs were used to guide the creation of a virtual model for simulating full mrDANNA networks using memristive synapse structures. Like DANNA, this model has a 2D grid of elements; however, in mrDANNA, each major element is a core consisting of a single neuron

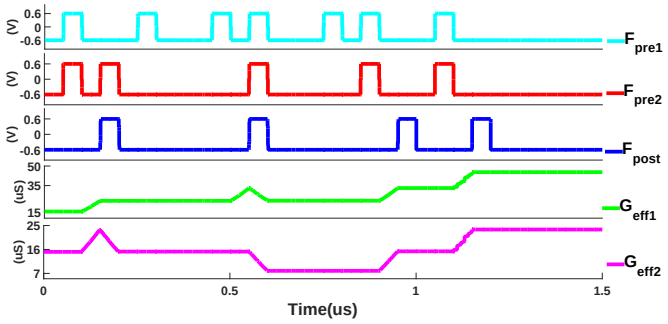

Fig. 3. Cadence Spectre simulation results showing LTP and LTD events for a 2-input mrDANNA network.

and eight memristive synapses. Connectivity follows that of the DANNA system and is limited to nearest neighbor connections. From the mrDANNA model, we have developed a high-level simulator that fits within the structure described in Section III. We have used the simulator to generate networks that solve our various applications. We show further work with mrDANNA in the two subsections below. Our first mrDANNA test chip is scheduled for fabrication at SUNY-PI in late 2017.

B. Application Research: Development of RoboNav

RoboNav started as a senior design project in the EECS department at the University of Tennessee. The goal was to design and build a self-contained robot, under neuromorphic control, that can avoid obstacles while exploring its environment. Once the robot was designed, the work split into two sub-tasks: physically building the robot, and programming a robot simulator that could be used to train a neuromorphic network to drive it. When each of these tasks was finished, a neuromorphic (DANNA) FPGA was loaded on the robot and attached to the inputs, the FPGA was configured with both the neuromorphic network and communication programming so that the physical inputs were translated to DANNA inputs. The final project was successful. RoboNav is pictured in the left side of Figure 4, and can explore spaces, while avoiding obstacles. We describe each of the tasks in RoboNav's development in greater detail below.

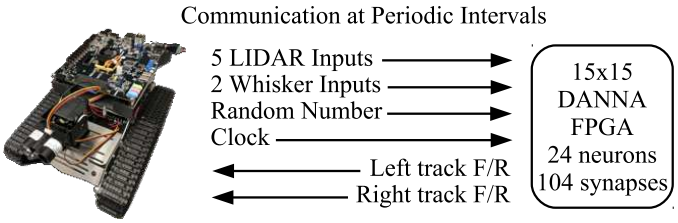

Fig. 4. Picture of RoboNav, with schematics on how it communicates with the DANNA FPGA.

RoboNav uses a track-based Kuman Sm5 with 12V motors. To sense obstacles, it employs a sweeping Garmin LIDAR-Lite v3 sensor on a servo. It sweeps in a 120 degree field of vision,

providing 8-bit sampling input at five locations (i.e., a 30 degree separation between sampling). Additionally, it has two "whiskers" on limit switches to detect drop-offs in the terrain (e.g., stairs, curbs, holes, etc). The neuromorphic processor is a Kintex-7 FPGA on a Digilent Genesys 2 development board. The FPGA implements the DANNA neuromorphic model, described briefly above in Section III. The information from the LIDAR and whisker sensors is converted to neuromorphic inputs, and the neuromorphic outputs are converted to track inputs via a MicroBlaze core. The core and FPGA are battery powered, and installed on the robot so that it is self-contained.

The simulator for RoboNav fits into the structure of our neuromorphic co-design framework. The simulator is written in C++ with a visualizer in Open-GL. The interaction between the robot and the neuromorphic navigation application is depicted in Figure 4. At regular intervals, the five LIDAR inputs, two whisker inputs, a random number generator, and a clock signal are all communicated as neuromorphic input, using the input-to-spike functionality described in Section III. The random numbers are intended to help explore random spaces. There are four output synapses, one for each direction of movement for each of the two tracks. At each interval, we count the firings of each pair of synapses for each track, and for each pair, the synapse that fires the most determines the direction.

We chose DANNA for RoboNav because of its FPGA implementation, which allows for self-contained operation. The size of the DANNA array is 15 X 15. To generate a DANNA network, we performed a parallel version of our genetic algorithm [41], utilizing 18,000 processing nodes for 24 hours on Oak Ridge National Laboratory's Titan supercomputer. The fitness function combines several goals of RoboNav: obstacle avoidance is the highest priority, followed by coverage of a simulated room, and the utilization of all track directions. The last goal was added, because the first successful RoboNav network only turned right! The best network achieves a fitness score of 72%, and is able to avoid all obstacles, but not achieve total room coverage from all starting positions. Since there are obstacles in the space, it is impossible to achieve 100% coverage.

The resulting network has 24 neurons and 104 synapses. We have successfully demonstrated the self-contained, battery-powered RoboNav exploring halls and offices at the University of Tennessee, for which it has not been trained, while avoiding all obstacles.

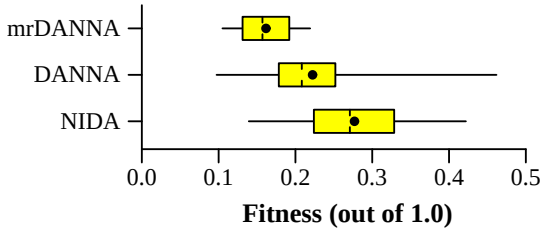

Fig. 5. Fitness results of performing 100 independent optimizations of RoboNav on each of three different models, for one hour each.

Because it was developed within our co-design framework, RoboNav may be used for benchmarking neuromorphic architectures and machine learning. As a brief example, Figure 5 displays the results of performing a small evolutionary optimization experiment on Titan. In the experiment, for each of the three models (NIDA, DANNA and mrDANNA), 100 independent optimizations were performed on 7 cores each, for exactly one hour. The graph shows Tukey plots of the 100 fitness values for each model. The fitness values compare poorly to the 72% network above, because each test is using 0.000016 of the computing power of the optimization that produced the network with 72% fitness.

The results show that NIDA generates better networks on the whole than DANNA, which generates better networks than mrDANNA. Drawing conclusions about the models, however, requires more care than simply comparing numbers, because the recently-written mrDANNA simulator is much slower than the more mature NIDA and DANNA simulators, and therefore performed less optimization. We include this graph to demonstrate that once an application like RoboNav is implemented within the co-design framework, it may be used to perform research on all of the models, and as we demonstrate in the next subsection, on the learning methodology.

C. Learning Research: Output Redundancy

Like all heuristic search techniques, evolutionary optimization (EO) suffers when the search space becomes too large. One potential way to improve EO is to add *diversity* to the population, to prevent the search from staying in local minima [42], [27]. In this section, we present a brief technique that we call *Output Redundancy*, to add diversity to populations.

We explain the technique with reference to one of our benchmark applications: XOR. In this application, we present a series of w-bit XOR problems to the neuromorphic device. We show an example, where $w = 1$, in Figure 6(a). For each problem, two bits, A and B, are pulsed into the neuromorphic device. There are separate input neurons for 0 and 1, which we label A0/A1/B0/B1. The device then runs for a set interval of time. During that interval, we count output pulses on two neurons, one for an output of 0 and one for an output of 1. Whichever pulses more is the answer (ties go to zero). We repeat the process for subsequent problems. When we train networks using EO, we affix a seed and generate 200 random XOR problems. The fitness of a network is the percentage (or factor) of correct answers.

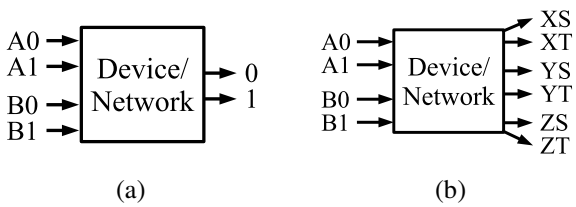

Fig. 6. Structuring a one-bit XOR application without Output Redundancy (a), and with Output Redundancy (b).

With Output Redundancy, we choose a number, $R >= w$ of outputs. In Figure 6(b), that number is three, and there are three pairs of output neurons, which we label XS/XT, YS/YT and ZS/ZY. This gives us a great deal of flexibility in how we interpret output. We may select any of the pairs X, Y or Z, and then we can assign 0 to either S or T, and 1 to the other. That yields 6 different ways to interpret output. When $w > 1$, there are $R!/(R-w)!$ ways to assign the output pairs to bits, and 2^w ways to assign each output in each pair to a value. Therefore, the number of ways to interpret output is:

$$N = 2^w \left(\frac{R!}{(R-w)!} \right).$$

When we perform EO to train networks using Output Redundancy, we can evaluate each of the N ways of interpreting output with one execution of the neural network. The reason is that the interpretation of the output in XOR does not affect the input. This is not true for all applications. The control applications (like RoboNav), for example, have the output affect the state of the system, and therefore affect the input. The classification and security applications are like XOR, and the output does not affect the input.

At the end of an execution during EO, there are then N fitnesses from which to choose. We choose the best of these, and report it as the overall fitness of a network. This adds diversity. To give an example from Figure 6(b), the EO may select a network whose (0/1) outputs are XS/XT, and a network whose outputs are ZT/ZS, to reproduce. In Figure 6(a), all networks have the same outputs.

Herein, we present a brief experiment with Output Redundancy, one and two-bit XOR, and all three of our neuromorphic architectures. For each value of w, we performed over 100 EO runs without Output Redundancy, and then with Output Redundancy for $R = \{w, w+1, w+2, w+3\}$. For each run, we stopped either when the fitness was one, meaning the network solved the problem, or we had run for 100 epochs. Each epoch ran fitness calculations on a population of 1000 networks, after which a round of genetic mutations and reproductions generated the next epoch. We present the results in Figures 7 and 8.

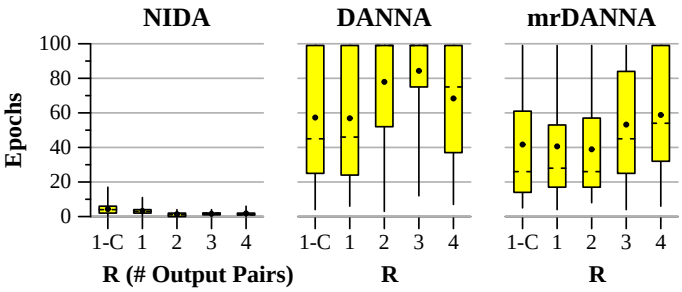

Fig. 7. Output Redundancy experiment for one-bit XOR.

Figure 7 shows the number of epochs for each run, when $w = 1$. We label the run without Output Redundancy "1-C" to signify that there is one pair of outputs, and this is the control. For each architecture and value of R, we show a Tukey plot of all of the runs. The tukey plot has a line from minimum to maximum, a box from first quartile to third quartile, a circle for the mean and two hash marks for the median. When a run reaches 100 epochs, that signifies that no network with a fitness of one was found.

For this application, NIDA evolves quickly and effectively, and the performance of EO also improves for $R \in \{1, 2\}$. The larger values of R find networks faster on average than when there is no Output Redundancy. However, they don't find them faster than when $R \in \{1, 2\}$. MrDANNA also improves slightly when $R \in \{1, 2\}$, but then it performs worse for higher values of R. DANNA, on the other hand, does not exhibit any improvement from Output Redundancy. It also takes the longest to find networks, failing in over 25% of the runs in all cases. We surmise that this is a result of DANNA's constrained connectivity. Although mrDANNA is also constrained to nearest neighbor connectivity, each element in mrDANNA is richer, being composed of a neuron and multiple synapses, rather than being constrained to a single neuron or synapse as in DANNA.

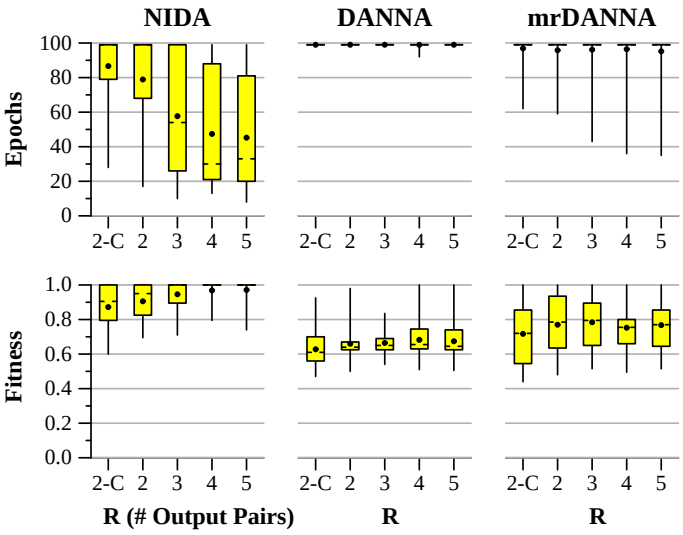

Fig. 8. Output Redundancy experiment for two-bit XOR.

In Figure 8, we present the results for $w = 2$. The control is now labeled "2-C." In this figure, in addition to showing the number of epochs, we show the fitnesses achieved. The reason is that when a majority of runs reach 100 epochs, the final fitness values measure of how well the EO performs. When $w = 2$, Output Redundancy shows marked effectiveness in NIDA. Without Output Redundancy, the majority of runs reach 100 epochs; when $R = 5$, nearly all of the runs achieve a fitness of 1, with more than half of the runs finding a network in under 40 epochs. DANNA and mrDANNA also show some effectiveness with Output Redundancy. This can be seen in DANNA by the increasing average fitness values, and in mrDANNA by the fact that maximum fitness networks are found with fewer epochs as R increases. Once again, for

this problem, NIDA evolves the best, followed by mrDANNA and DANNA.

Because this is a brief experiment, we do not draw wide conclusions from it. Output Redundancy is clearly an effective technique in some instances, but further study is warranted. Moreover, the difficulty of DANNA to find fully fit networks suggests that we should explore modifications to DANNA (perhaps trading programmable synapse length for greater connectivity), or strive to improve its reproduction operations. The important feature of this case study is that the unified environment allows us to explore a learning technique, and how it applies to multiple neuromrophic architectures, which are implemented on a variety of hardware devices.

V. Future Work and Conclusion

In this work, we describe a hardware/software framework for the co-design of neuromorphic computing devices, learning/training algorithms, and applications. We present case studies on each of three factors to demonstrate how this framework enables development on all three components and that the development completed on any one of the three components enables further research in the other two components. We describe how this framework enables research on a particular device implementation based on memristive synapses, on the development of an application for autonomous robot navigation, and on the development of an enhancement to the genetic algorithm based on Output Redundancy.

This framework will continue to enable exploration in each of these three facets of neuromorphic computing co-design. We are already actively pursuing the development of new neuromorphic device implementations, including an implementation utilizing biomimetic membranes. We intend also to contribute to research on existing neuromorphic implementations by developing the appropriate interfaces to our software core. For application research, we are actively developing applications in a variety of domains, including scientific data classification, anomaly detection in network traffic, and other autonomous vehicle navigation tasks (including helicopters and drones). Through the inclusion of multiple neuromorphic implementations with significantly varying characteristics, and a wide variety of application types, we can systematically explore learning and training methods for neuromorphic systems. We intend to implement and test a variety of unsupervised learning mechanisms based on spike-timing dependent plasticity (STDP), and we intend to explore other optimization methods beyond genetic algorithms and evolutionary optimization.

As described in previous sections, we are actively developing and deploying *real* neuromorphic devices, in both our FPGA implementation of DANNA and our test chips for both DANNA and mrDANNA. The learning and training paradigms in our framework provide us with networks for each of these devices that can solve real problems, such as the robotic navigation task described above. We are encouraged by the results enabled from this software/hardware co-design framework, and we intend to continue to utilize this framework to demonstrate how neuromorphic systems can and will be successfully deployed.

Acknowledgment

This research was supported in part by an Air Force Research Laboraty Information Directorate grant (FA8750-16-1-0065). It is also supported in part by the National Science Foundation grant 1631472. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. The allocation on OLCF's Titan was made possible through a Department of Energy Office of Science's ASCR Leadership Computing Challenge (ALCC) award. Research sponsored in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy.

References

[1] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the SpiNNaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[2] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza *et al.*, "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2013, pp. 1–10.

[3] D. Brüderle, M. A. Petrovici, B. Vogginger, M. Ehrlich, T. Pfeil, S. Millner, A. Grübl, K. Wendt, E. Müller, M.-O. Schwartz *et al.*, "A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems," *Biological cybernetics*, vol. 104, no. 4-5, pp. 263–296, 2011.

[4] E. Dauler, D. Rosenberg, J. Sage, J. Chiaverini, and W. Oliver, "3D integration of trapped ion and superconducting qubit technologies," in *42nd Annual GOMACTech Conference*, Reno, NV, March 2017.

[5] E. D. Dahl, "An introduction to quantum computing using map coloring," in *42nd Annual GOMACTech Conference*, Reno, NV, March 2017.

[6] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers in Neural Circuits*, vol. 10, pp. 1–13, 2016.

[7] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "STDP and STDP variations with memristors for spiking neuromorphic learning systems," *Frontiers in neuroscience*, vol. 7, p. 2, 2013.

[8] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[9] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, p. 201604850, 2016.

[10] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," arXiv:1705.06963, May 2017. [Online]. Available: https://arxiv.org/abs/1705.06963

[11] A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in neuroinformatics*, vol. 2, 2008.

[12] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[13] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications," *Nano letters*, vol. 12, no. 1, pp. 389–395, 2011.

[14] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[15] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis, "Integration of nanoscale memristor synapses in neuromorphic computing architectures," *Nanotechnology*, vol. 24, no. 38, p. 384010, 2013.

[16] E. Covi, S. Brivio, A. Serb, T. Prodromakis, M. Fanciulli, and S. Spiga, "HfO2-based memristors for neuromorphic applications," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 393–396.

[17] M. Hu, Y. Wang, Q. Qiu, Y. Chen, and H. Li, "The stochastic modeling of TiO2 memristor and its usage in neuromorphic system design," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 2014, pp. 831–836.

[18] T. Chang, P. Sheridan, and W. Lu, "Modeling and implementation of oxide memristors for neuromorphic applications," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, 2012.

[19] Y. Li, Y. Zhong, L. Xu, J. Zhang, X. Xu, H. Sun, and X. Miao, "Ultrafast synaptic events in a chalcogenide memristor," *Scientific reports*, vol. 3, 2013.

[20] V. Erokhin, "Organic memristive devices: Architecture, properties and applications in neuromorphic networks," in *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 305–308.

[21] J. V. Arthur, P. A. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. K. Esser, N. Imam, W. Risk, D. B. D. Rubin *et al.*, "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE, 2012, pp. 1–8.

[22] H. Ames, M. Versace, A. Gorchetchnikov, B. Chandler, G. Livitz, J. Léveillé, E. Mingolla, D. Carter, H. Abdalla, and G. Snider, "Persuading computers to act more like brains," in *Advances in Neuromorphic Memristor Science and Applications*. Springer, 2012, pp. 37–61.

[23] M. Azhar and K. R. Dimond, "Design of an FPGA based adaptive neural controller for intelligent robot navigation," in *Digital System Design, 2002. Proceedings. Euromicro Symposium on*. IEEE, 2002, pp. 283–290.

[24] S. Pande, F. Morgan, S. Cawley, T. Bruintjes, G. Smit, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, "Modular neural tile architecture for compact embedded hardware spiking neural network," *Neural processing letters*, vol. 38, no. 2, pp. 131–153, 2013.

[25] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 145–154.

[26] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[27] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[28] D. B. D'Ambrosio, J. Gauci, and K. O. Stanley, "HyperNEAT: The first five years," in *Growing adaptive machines*. Springer, 2014, pp. 159–185.

[29] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[30] P. Rocke, B. McGinley, F. Morgan, and J. Maher, "Reconfigurable hardware evolution platform for a spiking neural network robotics controller," in *International Workshop on Applied Reconfigurable Computing*. Springer, 2007, pp. 373–378.

[31] A. Zuppicich and S. Soltic, "FPGA implementation of an evolving spiking neural network," *Advances in Neuro-Information Processing*, pp. 1129–1136, 2009.

[32] J. Schemmel, K. Meier, and F. Schürmann, "A VLSI implementation of an analog neural network suited for genetic algorithms," in *International Conference on Evolvable Systems*. Springer, 2001, pp. 50–61.

[33] G. Howard, E. Gale, L. Bull, B. de Lacy Costello, and A. Adamatzky, "Towards evolving spiking networks with memristive synapses," in *Artificial Life (ALIFE), 2011 IEEE Symposium on*. IEEE, 2011, pp. 14–21.

[34] C. D. Schuman, "The effect of biologically-inspired mechanisms in spiking neural networks for neuromorphic implementation," in *IJCNN: The International Joint Conference on Neural Networks*, Anchorage, May 2017.

[35] M. E. Dean, J. Chan, C. Daffron, A. Disney, J. Reynolds, G. S. Rose, J. S. Plank, J. Birdwell, and C. D. Schuman, "An application development platform for neuromorphic computing," in *International Joint Conference on Neural Networks*, Vancouver, July 2016.

[36] Y. Shen, P. O. Saboe, I. T. Sines, M. Erbakan, and M. Kumar, "Biomimetic membranes: A review," *Journal of Membrane Science*, vol. 454, pp. 359–381, March 2014.

[37] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *International Joint Conference on Neural Networks*, Vancouver, July 2016.

[38] M. M. Adnan, S. Sayyaparaju, B. W. Ku, C. D. Schuman, S. K. Lim, R. C. Pooser, and G. S. Rose, "A digital pulse width modulation technique for spike-timing-dependent on-chip learning in memristive neuromorphic systems," Submitted for publication, 2017.

[39] K. Beckmann, J. Holt, H. Manem, J. Van Nostrand, and N. C. Cady., "Nanoscale hafnium oxide RRAM devices exhibit pulse dependent behavior and multi-level resistance capability," *MRS Advances*, vol. 1, no. 49, pp. 3355–3360, 2016.

[40] G. Chakma, S. Sayyaparaju, R. Weiss, and G. S. R. and, "A mixed-signal approach to memristive neuromorphic system design," in *Midwest Symposium on Circuits and Systems (MWSCAS)*, Boston, MA, August 2017.

[41] C. D. Schuman, A. Disney, S. P. Singh, G. Bruer, J. P. Mitchell, A. Klibisz, and J. S. Plank, "Parallel evolutionary optimization for neuromorphic network training," in *Machine Learning in HPC Environments, Supercomputing 2016*, Salt Lake City, November 2016.

[42] D. F. P. Durr and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.