Virtual Neuron: A Neuromorphic Approach for Encoding Numbers

Prasanna Date
Oak Ridge National Laboratory
Oak Ridge, TN 37830
Email: datepa@ornl.gov

Shruti Kulkarni
Oak Ridge National Laboratory
Oak Ridge, TN 37830
Email: kulkarnisr@ornl.gov

Aaron Young
Oak Ridge National Laboratory
Oak Ridge, TN 37830
Email: youngar@ornl.gov

Catherine Schuman
University of Tennessee
Knoxville, TN 37996
Email: cschuman@utk.edu

Thomas Potok
Oak Ridge National Laboratory
Oak Ridge, TN 37830
Email: potokte@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory
Oak Ridge, TN 37830
Email: vetter@ornl.gov

Abstract—**Neuromorphic computers perform computations by emulating the human brain and are expected to be indispensable for energy-efficient computing in the future. They are primarily used in spiking neural network–based machine learning applications. However, neuromorphic computers are unable to preprocess data for these applications. Currently, data is preprocessed on a CPU or a GPU—this incurs a significant cost of transferring data from the CPU/GPU to the neuromorphic processor and vice versa. This cost can be avoided if preprocessing is done on the neuromorphic processor. To efficiently preprocess data on a neuromorphic processor, we first need an efficient mechanism for encoding data that can lend itself to all general-purpose preprocessing operations. Current encoding approaches have limited applicability and may not be suitable for all preprocessing operations. In this paper, we present the virtual neuron as a mechanism for encoding integers and rational numbers on neuromorphic processors. We evaluate the performance of the virtual neuron on physical and simulated neuromorphic hardware and show that it can perform an addition operation using 23 nJ of energy on average using a mixed-signal, memristor-based neuromorphic processor. The virtual neuron encoding approach is the first step in preprocessing data on a neuromorphic processor.**

I. Introduction

Neuromorphic computers perform computations by emulating the human brain [1], and like the human brain, they are extremely energy efficient [2]. For instance, while CPUs/GPUs consume around 70–250 W of power, a neuromorphic computer consumes around 65 mW of power (i.e., 4–5 orders of magnitude less power than CPUs and GPUs [3]). The structural and functional units of neuromorphic computation are neurons and synapses, which can be implemented on digital or analog hardware [4]. They impart critical characteristics to neuromorphic computing such as colocated processing and memory, event-driven computation, massively parallel operation, and inherent scalability [5]. These characteristics are crucial for the energy efficiency of neuromorphic computers. Neuromorphic computing is also Turing-complete—that is, capable of general-purpose computation [6]. This ability to perform general-purpose computations and potentially use orders of magnitude less energy in doing so is why neuromorphic computing is poised to be an indispensable part of the energy-efficient computing landscape in the future.

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, world-wide 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). This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725.

Neuromorphic computers are seen as accelerators that can perform machine learning operations by using spiking neural networks (SNNs). For performing any other operation (e.g., arithmetic, logical, relational), we still resort to CPUs and GPUs. These general-purpose operations are important for preprocessing data before it is transferred to a neuromorphic processor. In the current neuromorphic workflow (i.e., preprocessing on the CPU/GPU and inferencing on the neuromorphic processor) more than 99% of the time is spent on data transfer (Table IV). This is highly inefficient and can be avoided by preprocessing on the neuromorphic processor. To develop efficient mechanisms for preprocessing data on a neuromorphic processor, we must first have an efficient mechanism for encoding the data on the neuromorphic processor. The virtual neuron proposed in this paper is such an encoding mechanism.

A few methods to encode numbers on neuromorphic processors exist in the literature [7], but their scope is restricted to the specific application they were designed for, and they are not suitable for general-purpose operations. Furthermore, no good mechanism exists for encoding negative integers and positive and negative rational numbers exactly on neuromorphic computers. To this extent, our main contributions in this work are as follows:

1) We introduce the Virtual Neuron as a spatial encoding mechanism for encoding integers and rational numbers

on neuromorphic processors. We also analyze its computational complexity (Section IV).
2) We implement the virtual neuron in the Neural Simulation Technology (NEST) simulator [8] and test its performance on 16-bit rational numbers (Section V).
3) We analyze the virtual neuron's run time on the Caspian digital neuromorphic hardware [9] and estimate its energy usage on the mrDANNA mixed-signal, memristor-based neuromorphic hardware [10] (Section VI).

II. Related Work

Neuromorphic computing is primarily used for SNN-based machine learning applications, including computer vision [11], natural language [12], and speech recognition [13]. The latest additions to neuromorphic computing applications include graph algorithms [14]–[16], autonomous racing [17], epidemiological simulations [18], classifying supercomputer failures [19], μ-recursive functions [6], [20], and boolean matrix-vector multiplication [21].

Most of the above applications are based on binary numbers and Boolean arithmetic. This is due to the spiking behavior of the neuron—the spikes can be interpreted as a 1, and lack of spike can be interpreted as a 0. Leveraging this behavior, several mechanisms for encoding numbers (mainly positive integers) have been proposed in the literature. Iaroshenko and Sornborger propose neuromorphic mechanisms for encoding binary numbers and use it for binary two's complement operations and binary matrix multiplication [22]. However, their approach uses numbers of neurons and synapses that are of the quadratic and cubic order, respectively. Lawrence et al. perform neuromorphic matrix multiplication by using an intermediate transformation matrix, flattened into a neural node, for encoding [23]. Schuman et al. propose three mechanisms for encoding positive integers on neuromorphic computers; these mechanisms are used in many applications [7]. Zhao et al. develop a compact, low power and robust spiking-time-dependent encoder designed with a leaky integrate-and-fire (LIF) neuron cluster and a chaotic circuit with ring oscillators [24]. Zhao et al. develop a method for representing data by using spike–time dependent encoding that efficiently maps a signal's amplitude to a spike time sequence that represents the input data [25]. Zhao et al. propose an analog temporal encoder for making neuromorphic computing robust and energy efficient [26]. Wang et al. use radix encoding of spikes to realize SNNs more efficiently and improve the speedup by reducing the overall latency for machine learning applications [27]. Other efforts to realize basic computations on neuromorphic platforms that leverage the inherent structure and parameters of SNNs for logic operations (e.g., AND, OR, XOR) were demonstrated in [28]. George et al. perform IEEE 754 compliant addition using SNNs by designing a system based on the Neural Engineering Framework and implemented, simulated and tested the design by using Nengo [29]. Dubey et al. extend this work to perform IEEE 754 compliant multiplication [30].

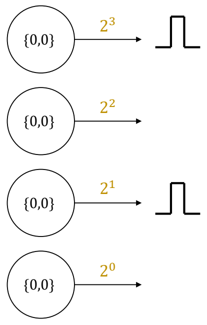

Fig. 1. Encoding rationale of a 4-bit virtual neuron. Numbers can be encoded by selecting synaptic weights as powers of two. Here, we use four neurons to encode the number 10.

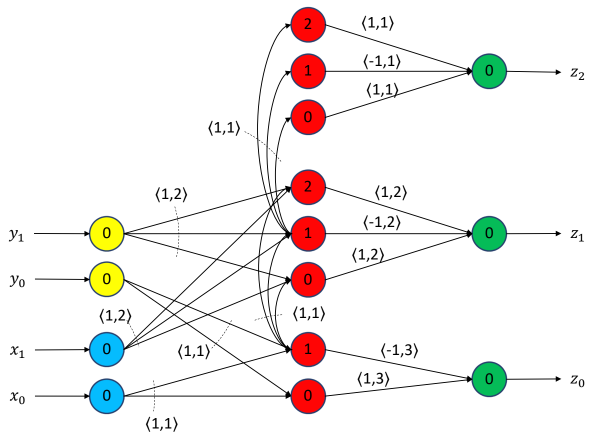

Fig. 2. A 2-bit virtual neuron. Takes two 2-bit numbers as input on the left: X and Y, represented as x_1, x_0, and y_1, y_0 respectively. Adds the two numbers and generates their sum on the right. The sum of two 2-bit numbers can at most be a 3-bit number.

Most of these encoding mechanisms have the ability to encode binary numbers. They are designed with specific applications in mind (e.g., control applications), and it is unclear if they can be used for general-purpose neuromorphic computation, for which arithmetic operations must be performed on positive and negative integers/rationals. Moreover, some encoding mechanisms (e.g., binning) tend to lose information by virtue of discretization. To the best of our knowledge, an efficient mechanism for encoding positive and negative rational numbers does not yet exist in the literature.

III. Neuromorphic Computing Model

We leverage the neuromorphic computing model described by Date et al. [6], [20]. This model is based on the LIF neuron with two parameters (threshold, ν and leak, λ) and two synapse parameters (weight, ω and delay, δ).

IV. The Virtual Neuron

Structurally, the virtual neuron is composed of LIF neurons and synapses that are connected in a particular way. Functionally, the virtual neuron mimics the behavior of an artificial neuron with identity activation. It leverages the binary encoding to encode numbers and performs addition operations similar to a ripple carry adder. Figure 1 shows a way of encoding 4-bit numbers on a neuromorphic computer. Each neuron in the figure represents a bit. The synapse coming out of the neuron assigns a value to the binary spike of the neuron by multiplying its weight. We can encode rational numbers by having powers of two as the weights. The synapses coming out of the four neurons in Figure 1 have weights 2^0, 2^1, 2^2, and 2^3. When the second and fourth neurons (from the bottom) spike, the result gets multiplied by 2 and 8, respectively, in the outgoing synapses. This is interpreted as the number 10.

A 2-bit virtual neuron is shown in Figure 2. As input, it takes two 2-bit numbers, X and Y, which are shown in the figure as $[x_1, x_0]$ (blue neurons) and $[y_1, y_0]$ (yellow neurons). It then adds X and Y in the three groups of bit neurons, which are shown in red. We call them *bit neurons* because they are responsible for the bit-level operations in the circuit (e.g., bitwise addition, propagating the carry bit). Finally, it produces a 3-bit number, Z, as output, which is shown in the figure as $[z_2, z_1, z_0]$ (green neurons).

The default internal states of all neurons are set to -1. Furthermore, all neurons have a leak of 0, which means they reset to their default internal state instantaneously if they do not spike. The reset state of all neurons is set to -1 (i.e., all neurons reset to -1 after they spike). The numbers on the neurons indicate their thresholds. For example, the top set of bit neurons (red neurons) has thresholds 0, 1, and 2, respectively. The synapse parameters are indicated in angular brackets on the top or bottom of the synapses. The first parameter is the weight, and the second parameter is the delay. If a group of synapses has the same parameters, then it is indicated with a dotted arc. The delays are adjusted so that bit operations of red neurons are synchronized.

We now describe the inner workings of the virtual neuron shown in Figure 2 by using the following example: $[x_1, x_0] = [1, 1]$ and $[y_1, y_0] = [0, 1]$. We start our analysis when the inputs X and Y have been received in the blue and yellow neurons—let us call this the zeroth time step. In the first time step, the bottom set of bit neurons in red receive an input of 1 along each of their incoming synapses. Thus, the total incoming signal at both of these neurons is 2, which changes their internal state from -1 to 1. As a result, both the bottom red neurons spike. Their spikes are sent along their outgoing synapses, which delay the signal for 3 time steps.

In the second time step, the bit neurons in the middle set receive the following inputs: 1 from the blue incoming neuron that represents x_1, 0 from the yellow neuron that represents y_1, and 1 from the red bit neuron with a threshold of 1 in the bottom group. Thus, the sum of their incoming signals is 2, and their internal states reach a value of 1. As a result, neurons with thresholds 0 and 1 in the middle set spike, whereas the one with threshold 2 does not spike. The spikes from the middle red neurons with thresholds 0 and 1 are sent to the green output neuron that represents z_1 along their outgoing synapses, which are delayed for 2 time steps.

In the third time step, the three bit neurons in the top group of red neurons receive an input of 1 along each of their incoming synapses. As a result, their internal states are incremented by 1 to the value of 0. The neuron with 0 threshold spikes as a result sends its spike along its outgoing synapse to the green neuron that represents z_2.

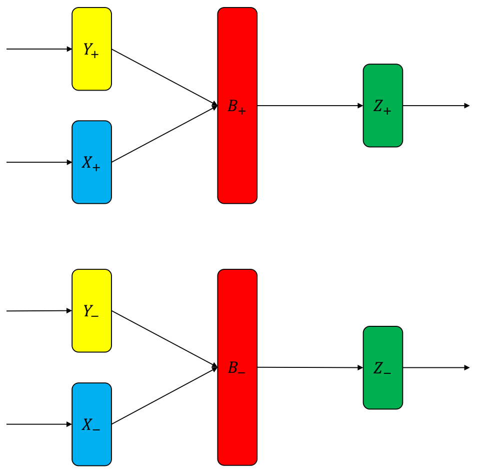

Fig. 3. Encoding P_+-bit positive rationals and P_--bit negative rationals using a virtual neuron.

In the fourth time step, the green neurons that represent z_0, z_1, and z_2 receive their inputs: z_0 receives a 1 and -1 from the bit neurons with the thresholds 0 and 1, respectively, in the bottom group of red neurons. Its total input is thus $1 - 1 = 0$, which keeps its internal state at -1, and it does not spike. Similar operations happen at the green neuron that represents z_1. It, too, does not spike. The green neuron that represents z_2 receives a signal of 1 from the bit neuron with the threshold of 0 in the top red set. As a result, its internal state is incremented by 1 to the value of 0, and it spikes.

The net output $[z_2, z_1, z_0]$ from the circuit is $[1, 0, 0]$, which can be interpreted as the number 4 in binary. Given that our inputs were $[x_1, x_0] = [1, 1]$, and $[y_1, y_0] = [0, 1]$ (i.e., $X = 3$ and $Y = 1$), we have received the correct output of 4 from the virtual neuron circuit. Note that we did not use powers of two in the synapses inside the virtual neuron. Depending on the application, powers of two as synaptic weights may be used on the incoming or outgoing synapses of a virtual neuron.

Although we restricted ourselves to 2-bit positive integers in this example, this approach can be extended to encode positive and negative rational numbers by using positive and negative powers of two. We let P_+ and P_- denote the number of bits used to represent positive and negative numbers, respectively. We call them *positive precision* and *negative precision*. In general, the positive precision, P_+, will be distributed among bits used to represent positive integers $(2^0, 2^1, 2^2, \ldots)$ and positive fractionals $(2^{-1}, 2^{-2}, 2^{-3}, \ldots)$. Similarly, the negative precision, P_-, will be distributed among bits used to represent negative integers $(-2^0, -2^1, -2^2, \ldots)$ and negative fractionals $(-2^{-1}, -2^{-2}, -2^{-3}, \ldots)$.

In Figure 3, the virtual neuron operates on two $(P_+ + P_-)$-bit rational numbers as inputs: X (blue rectangles) and Y

TABLE I
COMPARING THE VIRTUAL NEURON TO OTHER NEUROMORPHIC ENCODING MECHANISMS THAT REPRESENT TWO N-BIT NUMBERS EXACTLY

Metrics	**Binning [7]**	**Rate [7]**	**Time [7]**	**Virtual neuron**	**IEEE-754 [29], [30]**
Time Required (big-$\mathcal{O}$)	$\mathcal{O}(1)$	$\mathcal{O}(2^N)$	$\mathcal{O}(2^N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
No. of neurons (big-$\mathcal{O}$)	$\mathcal{O}(2^N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}$(N)	Ensembles of N neurons with dimension and radius properties[1] $\mathcal{O}$(N)[2].
Accuracy	100%	100%	100%	100%	>90% at 500 neurons per bit. Encoded error at 0 with >300 neurons per bit.
Energy Efficiency (no. of spikes in the worst case)	$\mathcal{O}(1)$ spikes	$\mathcal{O}(2^N)$ spikes	$\mathcal{O}(1)$ spikes	$\mathcal{O}$(N) spikes; N spikes	Energy results unpublished.
Energy Efficiency (no. of spikes in the average case)	$\mathcal{O}(1)$ spikes	$\mathcal{O}(2^N)$ spikes	$\mathcal{O}(1)$ spikes	$\mathcal{O}$(N) spikes; N/2 spikes	Energy results unpublished.

[1]Dimension refers to the number of values represented by the ensemble (this is 1 for a scalar quantity). Radius defines the range of values that can be represented by the ensemble. For the cited work, the dimension is 1, and the radius is set to 2.
[2]Authors only looked at IEEE floating point. How the representation scales with numerical precision is unclear.

TABLE II
COMPARING A VIRTUAL NEURON TO OTHER NEUROMORPHIC ENCODING MECHANISMS FOR ADDING TWO N-BIT NUMBERS

Metrics	**Binning [7]**	**Rate encoding [7]**	**Virtual neuron**	**IEEE 754 [29]**
Time to solution	$\mathcal{O}(1)$	$\mathcal{O}(2^N)$	$\mathcal{O}$(N)	Constant
No. of neurons	$\mathcal{O}(2^N)$	$\mathcal{O}(1)$	$\mathcal{O}$(N)	3075
No. of synapses	$\mathcal{O}(2^N)$	$\mathcal{O}(1)$	$\mathcal{O}$(N)	N/A
Energy Efficiency (no. of spikes in the worst case)	$\mathcal{O}(2^N)$	$\mathcal{O}(2^N)$	$\mathcal{O}$(N)	N/A
Energy Efficiency (no. of spikes in the average case)	$\mathcal{O}(2^N)$	$\mathcal{O}(2^N)$	$\mathcal{O}$(N); N/2	N/A
Accuracy	100%[1]	Depends on model type	100%	100% with 300 neurons per bit

[1]Accuracy is bound by the synapse weight and accumulation accuracy.

(yellow rectangles). Notice that the positive part of the circuit (upper half) is completely independent of the negative part of the circuit (lower half).

A. Computational Complexity

For P_+-bit positive operations, we use $\mathcal{O}(P_+)$ neurons and synapses and perform the virtual neuron operations in $\mathcal{O}(P_+)$ time steps. Specifically, we use $6P_+ + 3$ neurons, $12P_+$ synapses, and $P_+ + 2$ time steps for virtual neuron operations. Similarly, for P_--bit negative operations, we use $\mathcal{O}(P_-)$ neurons and synapses and perform the virtual neuron operations in $\mathcal{O}(P_-)$ time steps. All in all, we use $\mathcal{O}(P_+ + P_-)$ neurons and synapses and consume $\mathcal{O}(\max\{P_+, P_-\})$ time steps for the virtual neuron operations.

In computing the above space and time complexities, our inherent assumption is that the positive and negative precision is variable. However, we envision using the virtual neuron in settings where a neuromorphic computer has a fixed, predetermined positive and negative precision—just like the fixed precision on our laptops and desktops today, i.e., 32, 64, or 128 bits. In such a scenario, P_+ and P_- can be treated as constants. Thus, the resulting space and time complexities for a virtual neuron would all be $\mathcal{O}(1)$.

Table I compares different neuromorphic encoding approaches found in the literature with our approach, the virtual neuron. Because a neuromorphic computer consumes energy that is proportional to the number of spikes, we use the number of spikes in the worst case and the average case as an estimate for the energy usage of different neuromorphic approaches. Across different comparison metrics (e.g., network size, number of spikes), the virtual neuron scales linearly with the bit-precision, N, while providing the exact representation of the input number. Other approaches take exponential space (binning) or exponential time (rate encoding) or are unable to represent rational numbers exactly (IEEE 754).

Table II compares the computational complexity of performing addition with two N-bit numbers under different neuromorphic encoding schemes. Here, we do not include the temporal encoding scheme because under such a simple approach, binary spikes that occur at different time instances cannot be added exactly by spiking neurons. While the virtual neuron performs addition in linear time and by using a linear number of neurons, synapses, and energy, other approaches use either exponential time (rate encoding), exponential space (binning), exponential energy (rate encoding), or significantly large number of neurons per bit (IEEE 754).

V. Implementation Details

We implemented the virtual neuron in Python by using the NEST simulator. We ran the simulations on an Apple MacBook Pro equipped with a quad-core Intel Core i7 CPU running at 2.3 GHz paired with 32 GB of LPDDR4X memory running at 3,733 MHz. We wrote a `VirtualNeuron` class, whose constructor takes a list-like object of length 4 as the precision vector. The elements of this vector correspond to the number of bits for positive integers, positive fractionals, negative integers, and negative fractionals.

We used the `iaf_psc_delta` neuron model in NEST. All neurons had an internal state of -1.0 and a leak of 10^{-6}, which is a good approximation to a 0 leak. All neurons except the bit neurons had a neuron threshold of 0. The set of bit neurons that correspond to the least significant bit in both the positive and negative parts of the circuit had only two neurons with thresholds 0 and 1. All other groups of bit neurons had three neurons with thresholds 0, 1, and 2, respectively.

The synapses between positive (negative) incoming neurons and positive (negative) bit neurons had weights as 1.0 and delays of $i+1$, where i ranges from 0 to P_+ (P_-). The carry synapses go from the bit neuron having a threshold of 1 in the i^{th} group to all neurons in the $(i+1)^{\text{th}}$ group, where i

TABLE III
TESTING VIRTUAL NEURON ON 16-BIT RATIONAL NUMBERS. PRECISION IS [4, 4, 4, 4] FOR POSITIVE INTEGER, POSITIVE FRACTION, NEGATIVE INTEGER, AND NEGATIVE FRACTION, RESPECTIVELY.

X_+		X_-		Y_+		Y_-		Z_+		Z_-	
Decimal	Binary	Decimal	Binary	Decimal	Binary	Decimal	Binary	Decimal	Binary	Decimal	Binary
2.5625	00101001	-11.375	10110110	13.3125	11010101	-6.75	01101100	15.875	011111110	-18.125	100100010
2.3125	00100101	-13.9375	11011111	11.375	10110110	-9.3125	10010101	13.6875	011011011	-23.25	101110100
15.875	11111110	-2.9375	00101111	1.5625	00011001	-4.6875	01001011	17.4375	100010111	-7.625	001111010
8.625	10001010	-10.1875	10100011	8.9375	10001111	-1.625	00011010	17.5625	100011001	-11.8125	010111101
14.6875	11101011	-10.625	10101010	11.625	10111010	-11.875	10111110	26.3125	110100101	-22.5	101101000

goes from 0 to P_+ (P_-). The carry synapses had weights and delays of 1.0. The outgoing synapses from bit neurons with thresholds 0 and 2 had weights of 1.0, whereas those from bit neurons with thresholds 1 had weights of -1.0. These synapses in the positive (negative) part of the circuit had a delay $\max\{P_+, P_-\} - i + 1$ for i that ranged from 0 to P_+ (P_-).

VI. TESTING RESULTS

We tested our implementation of the virtual neuron on 16-bit rational numbers. The precision vector fed to the class constructors was $[4, 4, 4, 4]$. We tested $100,000$ permutations of inputs that were uniformly generated at random. Table III shows a sample of the results. The virtual neuron can encode and correctly add positive and negative rational numbers on a neuromorphic computer.

A. Caspian and Hardware Testing

We also implemented and tested the 16-bit virtual neuron by using the Caspian simulator and μCaspian digital field-programmable gate array (FPGA) hardware [9]. Because the μCaspian run time depends on activity, we ran 1,000 permutations of inputs selected uniformly at random on the μCaspian simulator and hardware and monitored the total number of spikes and the number of cycles used by the processor. Over the 1,000 runs, the simulator reported 73,159 total spikes for an average of 73 spikes per test case. Using Verilator, the 1,000 test cases finished in ∼5,000,000 clock cycles. Around 7,000 cycles were used to load the virtual neuron network, and ∼5,000 cycles were used per test case. Because the processor runs at 25 MHz, the total run time without the overhead from data transfer with the host computer was ∼0.21 s for all the test cases. When we ran the test using the UPduino FPGA, the total time was ∼400 s. One culprit for this slowdown is the 3 MBaud UART connection between the host and the FPGA. While running on hardware, over 99.9% of the execution time was spent in overhead and data transfer. This result highlights the great benefit of using the virtual neuron to perform addition on the neuromorphic processor instead of the CPU/GPU. The results from the hardware evaluation are tabulated in Table IV.

B. mrDANNA Power Estimate

The power required for a particular network execution can be estimated based on the energy required for active and idle neurons and synapses during execution. To estimate the power of the virtual neuron design, we used the same method and energy-per-spike values as reported in [10] for the mrDANNA mixed-signal, memristor-based neuromorphic processor. Using the same number of spikes, neurons, and synapses as reported in the μCaspian simulation, we estimate that a mrDANNA hardware implementation would use ∼23 nJ for the average test case run and around ∼23 mW for continuous operation.

TABLE IV
SUMMARY OF HARDWARE EVALUATION

Method	Execution time of processor	Wall time of evaluation	Power
μCaspian hardware	0.21 s	400 s	
μCaspian simulator	N/A	747 ms	
mrDANNA	1 μs @ 20MHz	N/A	23.04 mW

VII. DISCUSSION

In this paper, we proposed the virtual neuron as a mechanism for encoding and adding rational numbers. Our work is a stepping stone toward a broader class of general-purpose neuromorphic computing algorithms. In addition to the low operational power required to operate a neuromorphic computer, performing the general-purpose operations within the spiking array—without the need to send data to a CPU or GPU—will significantly reduce the communication overhead. Moreover, the virtual neuron is a vital component for composing subnetworks to scale up neuromorphic algorithms.

Many applications that use neuromorphic processors (e.g., classification, anomaly detection, control) may require general-purpose operations as a pre- or post-processing step. For a continually operating neuromorphic processor (e.g., in control applications), these operations may be required between neuromorphic computations. Performing them on the neuromorphic processor will alleviate the costs associated with moving data to and from the neuromorphic processor. Even if the neuromorphic computations described above may not be as fast as those on a traditional processor, it is likely that the slowdown incurred during data transfer will negate any speedup obtained on the traditional processor.

VIII. CONCLUSION

In this work, we presented the virtual neuron as a mechanism for encoding positive and negative integers and rational numbers. We implemented the virtual neuron in the NEST simulator and tested it on 16-bit rational numbers. We compared the computational complexity of the virtual neuron to other neuromorphic encoding mechanisms. We also tested the virtual neuron on neuromorphic hardware and presented its time, space, and power metrics. In our future work, we would like to explore general-purpose neuromorphic algorithms and applications that use virtual neurons.

References

[1] A. Calimera, E. Macii, and M. Poncino, "The human brain project and neuromorphic computing," *Functional neurology*, vol. 28, no. 3, p. 191, 2013.

[2] J. Grollier, D. Querlioz, K. Camsari, K. Everschor-Sitte, S. Fukami, and M. D. Stiles, "Neuromorphic spintronics," *Nature electronics*, vol. 3, no. 7, pp. 360–370, 2020.

[3] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[4] C. D. Schuman, J. P. Mitchell, J. T. Johnston, M. Parsa, B. Kay, P. Date, and R. M. Patton, "Resilience and robustness of spiking neural networks for neuromorphic systems," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–10.

[5] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, B. Kay *et al.*, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.

[6] P. Date, T. Potok, C. Schuman, and B. Kay, "Neuromorphic computing is Turing-complete," in *Proceedings of the International Conference on Neuromorphic Systems 2022*, 2022, pp. 1–10.

[7] C. D. Schuman, J. S. Plank, G. Bruer, and J. Anantharaj, "Non-traditional input encoding schemes for spiking neuromorphic systems," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–10.

[8] M.-O. Gewaltig and M. Diesmann, "Nest (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[9] J. P. Mitchell, C. D. Schuman, R. M. Patton, and T. E. Potok, "Caspian: A neuromorphic development platform," in *Proceedings of the Neuro-inspired Computational Elements Workshop*, 2020, pp. 1–6.

[10] G. Chakma, N. D. Skuda, C. D. Schuman, J. S. Plank, M. E. Dean, and G. S. Rose, "Energy and area efficiency in neuromorphic computing for resource constrained devices," in *Proceedings of ACM Great Lake Symposium on VLSI (GLSVLSI)*, 2018, pp. 379–383.

[11] T. Serre and T. Poggio, "A neuromorphic approach to computer vision," *Communications of the ACM*, vol. 53, no. 10, pp. 54–61, 2010.

[12] S. H. Sung, T. J. Kim, H. Shin, H. Namkung, T. H. Im, H. S. Wang, and K. J. Lee, "Memory-centric neuromorphic computing for unstructured data processing," *Nano Research*, vol. 14, no. 9, pp. 3126–3142, 2021.

[13] P. Blouw and C. Eliasmith, "Event-driven signal processing with neuromorphic computing systems," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8534–8538.

[14] B. Kay, P. Date, and C. Schuman, "Neuromorphic graph algorithms: Extracting longest shortest paths and minimum spanning trees," in *Proceedings of the Neuro-inspired Computational Elements Workshop*, 2020, pp. 1–6.

[15] B. Kay, C. Schuman, J. O'Connor, P. Date, and T. Potok, "Neuromorphic graph algorithms: Cycle detection, odd cycle detection, and max flow," in *International Conference on Neuromorphic Systems 2021*, 2021, pp. 1–7.

[16] K. Hamilton, T. Mintz, P. Date, and C. D. Schuman, "Spike-based graph centrality measures," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–8.

[17] R. Patton, C. Schuman, S. Kulkarni, M. Parsa, J. P. Mitchell, N. Q. Haas, C. Stahl, S. Paulissen, P. Date, T. Potok *et al.*, "Neuromorphic computing for autonomous racing," in *International Conference on Neuromorphic Systems 2021*, 2021, pp. 1–5.

[18] K. Hamilton, P. Date, B. Kay, and C. Schuman D, "Modeling epidemic spread with spike-based models," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–5.

[19] P. Date, C. D. Carothers, J. A. Hendler, and M. Magdon-Ismail, "Efficient classification of supercomputer failures using neuromorphic computing," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2018, pp. 242–249.

[20] P. Date, B. Kay, C. Schuman, R. Patton, and T. Potok, "Computational complexity of neuromorphic algorithms," in *International Conference on Neuromorphic Systems 2021*, 2021, pp. 1–7.

[21] C. D. Schuman, B. Kay, P. Date, R. Kannan, P. Sao, and T. E. Potok, "Sparse binary matrix-vector multiplication on neuromorphic computers," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 308–311.

[22] O. Iaroshenko and A. T. Sornborger, "Binary operations on neuromorphic hardware with application to linear algebraic operations and stochastic equations," 2021. [Online]. Available: https://arxiv.org/abs/2103.09198

[23] S. Lawrence, A. Yandapalli, and S. Rao, "Matrix multiplication by neuromorphic computing," *Neurocomputing*, vol. 431, pp. 179–187, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220316416

[24] C. Zhao, W. Danesh, B. T. Wysocki, and Y. Yi, "Neuromorphic encoding system design with chaos based cmos analog neuron," in *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2015, pp. 1–6.

[25] C. Zhao, B. T. Wysocki, Y. Liu, C. D. Thiem, N. R. McDonald, and Y. Yi, "Spike-time-dependent encoding for neuromorphic processors," *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 3, sep 2015. [Online]. Available: https://doi.org/10.1145/2738040

[26] C. Zhao, J. Li, and Y. Yi, "Making neural encoding robust and energy efficient: An advanced analog temporal encoder for brain-inspired computing systems," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2966986.2967052

[27] Z. Wang, X. Gu, R. Goh, J. T. Zhou, and T. Luo, "Efficient spiking neural networks with radix encoding," *arXiv preprint arXiv:2105.06943*, 2021.

[28] J. Plank, C. Zheng, C. Schuman, and C. Dean, "Spiking neuromorphic networks for binary tasks," in *International Conference on Neuromorphic Systems 2021*, 2021, pp. 1–9.

[29] A. M. George, R. Sharma, and S. Rao, "Ieee 754 floating-point addition for neuromorphic architecture," *Neurocomputing*, vol. 366, pp. 74–85, 2019.

[30] K. Dubey, U. Kothari, and S. Rao, "Floating-point multiplication using neuromorphic computing." [Online]. Available: https://arxiv.org/abs/2008.13245