Shortest Path and Neighborhood Subgraph Extraction on a Spiking Memristive Neuromorphic Implementation

Catherine D. Schuman, Kathleen E. Hamilton, Tiffany Mintz, Md Musabbir Adnan, Bon Woong Ku, Sung-Kyu Lim, and Garrett S. Rose

Abstract—Spiking neuromorphic computers (SNCs) are promising as a post Moore's law technology because of their potential for very low power computation. SNCs have primarily been demonstrated on machine learning applications, but they can also be used for applications beyond machine learning. Here, we demonstrate two graph problems (shortest path and neighborhood subgraph extraction) that can be solved using SNCs. We estimate the performance of a memristive SNC for these applications on three real-world graphs.

1 Introduction

Spiking neuromorphic computing systems (SNCs) [13] are compelling as a post Moore's law technology for a variety of reasons, including the potential for significantly lower power consumption than traditional computing architectures for certain workloads. Much of the work in neuromorphic computing applications thus far has been targeted towards neural network computation [13]. However, an SNC is simply a specialized hardware system with certain characteristics, namely, massively parallel systems with very simple computational units and communication, and collocated processing and memory. In this work, we seek to exploit these properties on two graph algorithms and present results for a shortest path finding algorithm and a subgraph extraction algorithm as mapped to a simulated memristive SNC.

A number of recent studies have shown that SNCs can be applied to a wider class of problems beyond spiking neural networks. In [2], Aimone, et al., proposes the spiking neural threshold gate model as a framework for realizing non-neural network applications for spiking neuromorphic systems, but there are many different approaches that have been implemented: integer factorization [10], particle image velocimetry [15], the generation of Markovian random walks [14], or the simulation of flocking behavior [3]. Constraint satisfaction problems have also been solved using spiking neural networks [7] using spike-based annealing, or by mapping Ising models to spiking neural networks [5].

We focus on graph algorithms; an application we believe is likely well suited to SNCs. It was shown in [15] that deciding how to map a problem to a SNC is non-trivial. For our algorithms, the inherent structure of graphs simplifies the embedding task while preserving the original graph sparsity, in contrast to other methods that rely on dense SNCs [8]. In [6], Davies, et al., note that there are methods (specifically [11]) for calculating shortest weighted path using SNCs; we discuss a variation of this approach further in Section 2.

2 Approach

We construct an SNC similar to those described in [12]: defined by a set of leaky integrate-and-fire neurons $\{n(v_{th}, t_R)\}$ with programmable thresholds and refractory periods, and a set of synapses $\{s(\delta, s_w)\}$ with programmable delays and weights. We require that the synapses can realize a form of spike timing dependent plasticity (STDP); in this work we use 1-step STDP.

An undirected graph $G(V,E)$ is defined by a vertex set $V(G)$ and an edge set $E(G)$. A directed graph $D(V,E)$ is defined by a vertex set $V(D)$ and a set of directed arcs $E(D)$. We use the notation E to refer to either a set of undirected edges or directed arcs, but directed edges will explicitly label the arc direction $e_{i\to j} \neq e_{j\to i}$ while undirected edges will only label the two terminal nodes $e_{ij} = e_{ji}$. The graphs in this paper are simple and we convert unweighted graphs into weighted graphs by assigning each $e \in E$ a length of 1. To avoid confusion with synaptic weights, we refer to graph edges as having lengths rather than weights.

Finally, we assume that the graphs on which we are computing can be embedded directly into a neuromorphic hardware system. A graph (G or D) is directly mapped to a SNC: each $v_i \in V$ defines a neuron $n_i \in n$ and each edge $e \in E$ defines a synapse $s \in s$. Directed arcs are mapped to directed synapses, and undirected edges are mapped to symmetric pairs of synapses $e_{ij} \to (s_{i\to j}, s_{j\to i})$. The parameters of n_i and s are defined by the specific application.

2.1 Shortest Path

We implemented a single-source multiple-destination shortest path finding algorithm on a simulated SNC using an approach similar to that described in [11]. We map a given graph G or D onto a neuromorphic system for shortest path finding by determining the system parameters $(v_{th}, t_R, \delta, s_w)$ of the SNC defined in above. If the graph is unweighted, we first convert it into a weighted graph.

For each $v \in V$, we create a neuron $n(v_{th} = 0, t_R = \alpha)$ where α is defined as:

$$\alpha = \left(\sum_{e \in E} len(e) + 1 \right) + 1. \tag{1}$$

This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy (DOE). The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. 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 the U.S. Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan.

- *C. Schuman, K. Hamilton, T. Mintz are with Oak Ridge National Laboratory, Oak Ridge, TN. (E-mail: schumancd@ornl.gov)*
- *S. Lim, B. Ku are with the Georgia Institute of Technology, Atlanta, GA.*
- *M. Adnan, G. Rose are with the University of Tennessee, Knoxville, TN.*

TABLE 1
Energy Estimate per Event Type

	Accumulation	Fire	Learning	Idle
Neuron	9.81 pJ	12.5 pJ	-	7.2 pJ
Synapse	1.45 pJ	-	2.58 pJ	0.07 pJ

By setting the refractory period to be greater than the sum of the edge lengths in the graph, we force each neuron in the network to fire at most once. For each $e \in E$ we create synapses $s(\delta = len(e) + 1, s_w = 1)$.

Once the SNC is configured, we choose source vertex v_s and stimulate its corresponding neuron n_s in the network at network time 0. We then simulate the activity in the network for α time steps. After, using the spike times of the various neurons in the network, we can determine the length of the shortest path from n_s to that neuron. The synapses that have potentiated (increased weights) are those that caused their post-synaptic neuron to fire, and thus, are part of the shortest path to that neuron from n_s. It is worth noting that if there are multiple shortest paths (i.e., the shortest paths are not unique), this approach will find all of the shortest paths.

2.2 Neighborhood Subgraph Extraction

A neighborhood of vertex v_i in a graph G, $N_G(v_i)$, is defined as the subgraph made up of all vertices $v_j \in V(G)$ such that $e_{i,j} \in E(G)$ and the set of all edges that connect the vertices in this set. To determine a neighborhood subgraph in a graph G, we configure the corresponding SRC with the following parameters: $(v_{th} = 1, t_R = 1)$ for all neurons, and $(\delta = 2, s_w = 1)$ for all synapses.

Extracting the neighborhood subgraph of a vertex v_s can be executed in two steps: first the vertices of the subgraph are identified, then the edges are identified. To identify the vertices of the subgraph, we stimulate the corresponding neuron n_s at time step 0 and simulate the network for exactly two time steps. The neurons that fire in those two time steps will be $\{n' \cup n_s\}$, the set of neurons that are adjacent to n_s and n_s, which correspond directly to the vertices in our desired subgraph. To identify the subgraph edges, we increase the spike threshold for $\{n\} \setminus \{n' \cup n_s\}$ to $|E(G)| + 1$. We reset the simulation, load the new graph with updated thresholds and $s_w = 1$, and simultaneously stimulate the neuron set $\{n' \cup n_s\}$. After two time steps we read all out all synapses that have $s_w > 1$ (the original weight) to obtain the edges that have potentiated and thus are edges of the subgraph.

2.3 Estimating Energy Usage

To determine approximate energy usage of a memristive SNC on these tasks, we collect event information from the simulation of the network, including the number of times a neuron accumulates, the number of times a neuron fires, the number of times a synapse receives a fire, and the number of times a synapse potentiates or depresses. We have previously discussed this simulation code [12] and how we estimated energy usage in [1], [4], [12]. A summary of the approximate energy usage of those events is given in Table 4, where the neuron estimates come from [4] and the synapse estimates come from [1].

3 Results

We use three graphs from the Stanford Large Network Dataset Collection [9] to evaluate performance on a simulated SNC: a California road network graph, a collaboration network graph from high energy physics on Arxiv, and a co-purchasing network from Amazon for June 1, 2003 (Table 2). To obtain performance for both tasks, we select the source vertex to be the vertex with the highest degree. We implemented naive shortest path and neighborhood subgraph extraction implementations, and confirmed that the output of the CPU implementations and the simulated SNC are identical. In particular, for shortest path, the neuromorphic implementation output in both the lengths of the paths found and the paths themselves, and for neighborhood subgraph extraction, the neighborhoods matched exactly.

TABLE 2
Real-World Graphs [9]

Graph	Type	Vertices	Edges
roadNet-CA	Undirected	1,965,206	2,766,607
ca-HepPh	Undirected	12,008	118,521
amazon0601	Directed	403,394	3,387,388

TABLE 3
SNC/CPU Co-Processor Breakdown

	Shortest Path	Neighborhood
SNC Runtime	O($\|E(G)\|$)	O(1)
CPU Runtime	O($\|E(G)\|$)	O($\|V(G)\| + \|E(G)\|$)
CPU$-$ $>$SNC Net Loads	1	2
SNC$-$ $>$CPU Net Reads	1	2

TABLE 4
Estimated SNC Energy Usage Results

	roadNet-CA	ca-HepPh	amazon0601
Shortest Path	161.35 J	48.85 mJ	21.28 J
Neighborhood	58.32 μJ	999.14 nJ	12.56 μJ

The number of clock cycles required on the SNC for shortest path relies on the size of the graph, while the number of clock cycles required for neighborhood subgraph extraction is fixed for any graph size. However, the shortest path method requires significantly less traditional processor computation for extraction. Table 3 summarizes the differences between the two approaches in terms of SNC usage and traditional processor usage, as well as communication between the two. Table 4 gives a breakdown of estimated energy usage per graph per task. As can be seen, because of the O($|E|$) clock cycles required for shortest path, significantly more time is spent on the SNC and thus, more energy usage is required. Also, the vast majority of the energy cost for both tasks comes from neuron and synapse idle states. If the idle costs could be reduced, significantly less energy would be required. These energy estimates do not capture the co-processor computation cost or communication cost associated with implementing these algorithms, but do give a rough estimate on what performance to expect.

4 Conclusion

There is significant opportunity to utilize SNCs for non-machine learning tasks in future HPC systems, though mapping these tasks to SNCs may be non-trivial. Here, we show how two graph problems can be mapped to a memristive SNC and give energy estimates on performance for three real-world graphs. These results illustrate how one might map a non-standard task onto an SNC and they also show that the two tasks have radically different performance and ways of utilizing the characteristics of the hardware, despite both being graph-related tasks.

ACKNOWLEDGMENTS

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] M. M. Adnan, S. Sayyaparaju *et al.*, "A twin memristor synapse for spike timing dependent learning in neuromorphic systems," in *Proceedings of the 31st IEEE International System-On-Chip Conference (SOCC)*. IEEE, 2018, p. To appear.

[2] J. B. Aimone, O. Parekh, and W. Severa, "Neural computing for scientific computing applications: more than just machine learning," in *Proceedings of the Neuromorphic Computing Symposium*. ACM, 2017, p. 7.

[3] R. Araújo, N. Waniek, and J. Conradt, "Development of a dynamically extendable spinnaker chip computing module," in *International Conference on Artificial Neural Networks*. Springer, 2014, pp. 821–828.

[4] G. Chakma, M. M. Adnan *et al.*, "Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the circuit-level," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 125–136, 2018.

[5] K. Corder, J. V. Monaco, and M. M. Vindiola, "Solving vertex cover via ising model on a neuromorphic processor," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.

[6] M. Davies, N. Srinivasa *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[7] G. A. Fonseca Guerra and S. B. Furber, "Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems," *Frontiers in neuroscience*, vol. 11, p. 714, 2017.

[8] K. E. Hamilton, N. Imam, and T. S. Humble, "Community detection with spiking neural networks for neuromorphic hardware," in *Proceedings of the Neuromorphic Computing Symposium*. ACM, 2017, p. 9.

[9] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[10] J. V. Monaco and M. M. Vindiola, "Integer factorization with a neuromorphic sieve," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–4.

[11] F. J. Ponulak and J. J. Hopfield, "Rapid, parallel path planning by propagating wavefronts of spiking neural activity," *Frontiers in computational neuroscience*, vol. 7, p. 98, 2013.

[12] C. D. Schuman, R. Pooser *et al.*, "Simulating and estimating the behavior of a neuromorphic co-processor," in *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*. ACM, 2017, pp. 8–14.

[13] C. D. Schuman, T. E. Potok *et al.*, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[14] W. Severa, R. Lehoucq *et al.*, "Spiking neural algorithms for markov process random walk," *arXiv preprint arXiv:1805.00509*, 2018.

[15] W. Severa, O. Parekh *et al.*, "Spiking network algorithms for scientific computing," in *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 2016, pp. 1–8.