2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 978-1-6654-3577-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/IPDPSW52791.2021.00054

# Sparse Binary Matrix-Vector Multiplication on Neuromorphic Computers

Catherine D. Schuman, Bill Kay, Prasanna Date, Ramakrishnan Kannan, Piyush Sao, and Thomas E. Potok
*Computer Science and Mathematics Division, Oak Ridge National Laboratory*
Oak Ridge, TN, USA

***Abstract*—Neuromorphic computers offer the opportunity for low-power, efficient computation. Though they have been primarily applied to neural network tasks, there is also the opportunity to leverage the inherent characteristics of neuromorphic computers (low power, massive parallelism, collocated processing and memory) to perform non-neural network tasks. Here, we demonstrate how an approach for performing sparse binary matrix-vector multiplication on neuromorphic computers. We describe the approach, which relies on the connection between binary matrix-vector multiplication and breadth first search, and we introduce the algorithm for performing this calculation in a neuromorphic way. We validate the approach in simulation. Finally, we provide a discussion of the runtime of this algorithm and discuss where neuromorphic computers in the future may have a computational advantage when performing this computation.**

***Index Terms*—neuromorphic computing, graph algorithms, matrix-vector multiplication, spiking neural networks**

## I. Introduction

Neuromorphic computers are a compelling complementary technology to traditional von Neumann computers. Neuromorphic computers are custom hardware systems that are inspired by the brain in both their structure and their functionality. Most neuromorphic system implement a form of spiking neural network computation. The fundamental computational components in a neuromorphic architecture are neurons and synapses, and communication in neuromorphic architectures is typically done via spikes. Neurons receive information in the form of charge either from an external source or from a synapse; they accumulate charge and upon reaching a threshold, they fire and create spikes. Synapses communicate spikes between neurons and have associated weights and delays. The weight value governs how spikes along that synapse affects that charge value of the post-synaptic neuron and delay values govern how long it takes for spike to propagate from the pre-synaptic neuron to the post-synaptic neuron.

Neuromorphic computers are inherently parallel, have collocated processing and memory, and tend to consume less power than their more conventional computing counterparts. Though neuromorphic systems are primarily targeted towards performing neural network tasks [1], there is increasing evidence that they may be useful in a variety of other computing applications and algorithms as well [2], specifically be exploiting the inherent computational characteristics of these systems.

In this work, we describe for the first time how binary matrix-vector multiplication can be computed using networks of spiking neurons and how this approach can be mapped onto neuromorphic computers. We validate the described approach using the NEST neural simulator [3]. We describe a runtime analysis of the neuromorphic computation and discuss where neuromorphic computers in the future may have a computational advantage over traditional CPU, specifically for sparse binary matrices.



## II. Related Work

Sparse matrix-vector multiplication (SPMV) is a performance-critical sparse basic linear algebra subroutine (BLAS). SPMVis applied in various numerical applications such as conjugate gradient (CG), generalized minimum residual (GMRES), and scientific applications such as solving partial differential equations (PDEs), graph and machine learning applications, and scientific simulations. Researchers have studied optimizing SPMVon single-core CPU [4], multicore CPUs [5], GPUs [6], and Xeon-Phi [7]. Optimizing SPMVon Von Neumann architecture reduces to designing new sparse matrix storage formats to exploit cache-hierarchy and SIMD/SIMT units to optimize data movements. Note that accelerators such as GPUs and Xeon-Phi, despite operating on lower frequencies, outperform CPUs both in terms of time and energy by employing a large number of simple compute cores. Neuromorphic hardware can be seen as one extreme of this architecture spectrum, since it use extreme simple compute element operating on low frequency, but it can provide much higher concurrency.

While the typical use-case of neuromorphic systems is neural network centric, spike-based neuromorphic systems are well poised for many application areas (edge computing, autonomous vehicles, internet of things etc.) due to their low energy costs and intrinsic parallelism. Indeed, neuromorphic systems have been used to address several non-neural network-based tasks [2]. In addition, multiple graph algorithms like shortest path, longest shortest path, neighborhood subgraph extraction, minimum spanning trees, graph centrality measures and others have been proposed in the literature [8]–[11].

978-1-6654-3577-2/21/$31.00 ©2021 IEEE
DOI 10.1109/IPDPSW52791.2021.00054

Neuromorphic systems have also been used to model epidemiological simulations [12]. Smith et al. used Intel Loihi [13] and IBM TrueNorth [14] neurosynaptic systems to solve steady state partial differential equations [15]. Further application areas include energy-efficiency in signal processing [16] and resilience in high performance computing [17].

## III. Method

To calculate binary matrix-vector multiplication using a network of spiking neurons, we take the opposite approach of GraphBLAS, which translates the problem of calculating a breadth first search (BFS) in an unweighted graph into a problem of binary matrix vector multiplications [18]. At every step, BFS explores all of the neighbor nodes that have not been visited so-far at the present depth prior to moving on to the nodes at the next depth level. BFS terminates when all the nodes in the graph are visited. Here, we translate the problem of computing a binary matrix vector multiplication into this single step of BFS and utilize a network of spiking neurons.

### A. BFS via Binary Matrix-Vector Multiplication

As described in [18], a BFS calculation on an unweighted graph can be calculated using binary-matrix vector calculations. Suppose we have an undirected graph $G(V, E)$. We can begin the BFS calculation by constructing a binary vector $x$ of length $|V|$ where all elements are zeroes, except for the element corresponding to the source node for the BFS. To perform the BFS, this vector $x$ is multiplied by the transpose of the adjacency matrix, $\mathbf{A}^\intercal$. The resulting $\mathbf{A}^\intercal x$ is a vector where there are ones in all locations that correspond to nodes that are directly connected to the source node.

### B. BFS via Spiking Neural Networks

In contrast, neuromorphic systems can very naturally compute BFS-style computation. In particular, a graph $G(V, E)$ can be used to construct a corresponding network of spiking neurons, where each node becomes a neuron and each edge becomes a synaptic connection between two neurons. As has been described in multiple previous works [8]–[10], a single source shortest path calculation can be performed on this graph by stimulating the source neuron and allowing spikes to propagate throughout the graph. The time that each neuron fires indicates the length of the shortest path to that node. To account for multiple source nodes (i.e., multiple ones in the vector), we will simply stimulate multiple neurons at time 0.

### C. Binary Matrix-Vector Multiplication via Spiking Neural Networks

Using this approach, we now describe in this work how to utilize spiking neuromorphic systems to compute binary matrix-vector multiplication. Given a matrix $\mathbf{A} \in \{0,1\}^{n \times n}$ and a binary vector $\mathbf{x} \in \{0,1\}^n$, we can treat $\mathbf{A}^\intercal$ as the adjacency matrix of a graph $G(V, E)$, where $|V| = n$. We can construct the corresponding spiking neural network (SNN) for a neuromorphic system by creating a neuron for each node in the graph and a synapse for each edge. In order to allow

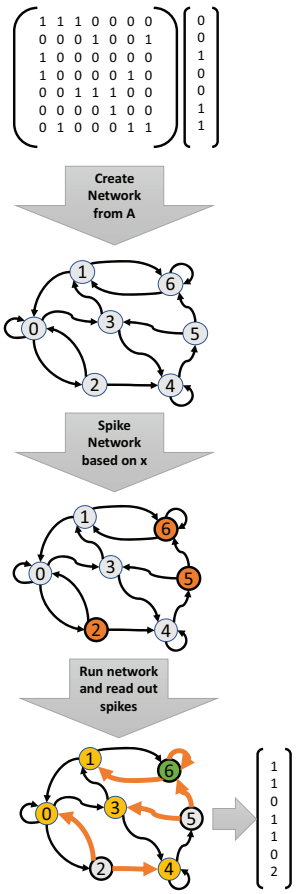


Fig. 1. Example of the process to perform binary matrix vector multiplication with a neuromorphic network. The network created from the matrix $\mathbf{A}$ by using $\mathbf{A}^\intercal$ as the adjacency matrix for the network. Then, all of the neurons that correspond to ones in the vector $\mathbf{x}$ are spiked as input. Spikes travel along all of the outgoing edges and cause corresponding spikes on the neurons they're connected to (shown in the last graph). The neurons that spike once are shown in yellow, and the neuron that spikes twice (6) is shown in green. The number of spikes are shown in the resulting vector, which equals $\mathbf{Ax}$

for multiple ones in the binary vector $\mathbf{x}$ (i.e., multiple source nodes in the VFE), we must guarantee that all edges have a unique value. In [10], we have previously shown that this can be achieved by setting the delay value of each edge to $\frac{1}{2^j}$, where $j \in \{1, ..., |E|\}$. This guarantees that each spike will arrive at a unique time and that the sum will be less than one simulation time step. We discuss the practicality of implementing this on hardware in the following section.

To calculate $\mathbf{Ax}$, we will simply stimulate all of the neurons indicated in the vector $\mathbf{x}$ and run the simulation of the SNN long enough for spikes to propagate along the immediate adjacent synapses. The output nodes that fire in that period correspond to the ones in the resulting $\mathbf{Ax}$ vector. An example of this process is given in Figure 1. It is worth noting that we are only performing a single step of the BFS here to accomplish a single matrix-vector multiplication.

### D. Neuromorphic Implementation Practicalities

Physical neuromorphic hardware implementations have precision limits, for example, on the synaptic delay value. Thus, allowing for $\frac{1}{2^{|E|}}$ is not practical to realize on a neuromorphic implementation. In this work, we utilize the NEST neural simulator and we use the default synaptic precision for NEST. The minimum granularity on the synaptic delay is 0.1. Unlike [10] in which we require that the sum of any combination of synaptic delays also be unique, in this case, we only require that the delays be unique values. Therefore, we can realize unique delays values by setting each value to simply be $0.1j$ for $j \in \{1, ..., |e|\}$. We will then simulate for $0.1|e|$ time steps to allow spikes to propagate along at most one edge. In general, if the minimum synaptic delay value is $\alpha$, then the edge delays should be set to be $\alpha j$ for $j \in \{1, ..., |e|\}$.

### E. Runtime Analysis

A matrix-vector calculation on a conventional computing system require $O(|V|^2)$ operations. Traditional Big-$O$ notation does not make sense for a neuromorphic implementation. There are different ways that computational complexity can be quantified for a neuromorphic system: one is the number of synaptic and/or neuronal operations (e.g., fires) and the other is the amount of simulation time required to produce the result. In this work, we focus on the second, and we use a notation similar to Big-$O$, $O_{\mathcal{N}}$, to denote this neuromorphic quantity.

We assume that the neuromorphic system will be operating as a co-processor alongside a conventional CPU. In this case, the CPU converts the matrix to a network of spiking neurons, loads the network onto the neuromorphic co-processor, simulates the neuromorphic co-processor, and extracts the spikes to calculate the appropriate vector. The network graph construction requires $O(|V| + |E|)$ and occurs on a conventional CPU. The neuromorphic simulation time is $O(\alpha|E|)$ (where $\alpha$ is the minimum value of a synaptic delay), which we simplify as $O_{\mathcal{N}}(|E|)$. There is also cost for loading the network and extracting spikes, though these values will be specific to the neuromorphic hardware and communication hardware between the CPU and neuromorphic system.

## IV. Preliminary Results

### A. Verification via Simulation

To confirm that the approach is valid on a system of spiking neurons, we performed the binary matrix-vector multiplication for all binary matrices $\mathbf{A}$ of up to size $n \times n$ and for all binary vectors $\mathbf{x}$ of size $n$ or $n = 3$. We further verified the performance of the approach by evaluated all binary vectors of size $n$ for each of 1000 randomly selected matrices, for $n \in \{4, ..., 10\}$. The results of each of these binary matrix-vector multiplications via neuromorphic simulation matched the output of numpy's matrix-vector multiplication result.

### B. Runtime Analysis and Neuromorphic Advantage

To understand when a neuromorphic system will have an advantage when performing this type of calculation, we provide a brief analysis of CPU run time when compared with neuromorphic run time. As discussed in Section III-E, if a matrix is size $N \times N$, and the vector is size $N$, the corresponding graph has $N$ vertices ($N = |V|$). Because the neuromorphic algorithm depends on the number of edges or synapses in the network, it depends on the sparsity of the matrix (the adjacency matrix of the network). Additionally, neuromorphic systems are expected to be slower than their CPU counterparts with respect to clock speed or even asynchronous altogether. In this section, we assume that the neuromorphic system performs $|E|$ simulation time steps and that each of those time steps requires $\gamma T$, where $T$ is the amount of time it takes the CPU to perform a computation and $\gamma \leq 1$.

In Figure 2, we illustrate how the neuromorphic implementation will perform with different matrix sparsities. The naïve CPU-only implementation does not depend on the sparsity of the matrix, so only one CPU line is plotted. We show different sparsity values (up to sparsity 0.5) as well as how the speed of the neuromorphic system (with respect to the CPU) affects performance. Note that the neuromorphic implementation of this algorithm has both a CPU component (to construct the graph) and a neuromorphic component (to run the simulation). We can see that when the neuromorphic system is reasonably fast, it is beneficial to perform the calculation on the neuromorphic system. However, when the neuromorphic system becomes substantially slower than the CPU and/or if the matrix is sufficiently dense, it is no longer worthwhile to move the computation to the neuromorphic system. It is also worth noting that time to load the network onto the neuromorphic hardware and communicate spikes to and from the hardware can also affect the performance, though there may still be an advantage in using a neuromorphic co-processor if that communication can be optimized.

### C. Computational Analysis

We can further probe into the analysis of when it will be worthwhile to perform the calculation on a neuromorphic hardware system. The neuromorphic pre-processing step on the CPU requires $|V| + |E|$ time steps, which, with a sparsity of $\rho$ and $|V| = N$, becomes $N + \rho N^2$. The neuromorphic component of the computation requires $|E|$ time steps in the worst case (when minimum synaptic delay values are 1). With a slowdown of $\gamma$, as described in the previous section, we would expect the neuromorphic system to require $\gamma\rho N^2$ time steps. Additionally, we can factor in a time cost for communication to and from the neuromorphic system, which we will call $C$. If working with a particular neuromorphic hardware system, $\gamma$ and $C$ will be known. One can calculate if it will be worthwhile to perform the computation on the neuromorphic system if the following is true:

$$N + \rho N^2 + \gamma(\rho N^2) + C < N^2 \tag{1}$$

## V. Future Work and Conclusions

Since the only walks of length 1 are single edges, the entries of adjacency matrix of a directed graph $A$ satisfy $(A)_{ij}$ is the number of walks of length 1 from $i$ to $j$. In general,

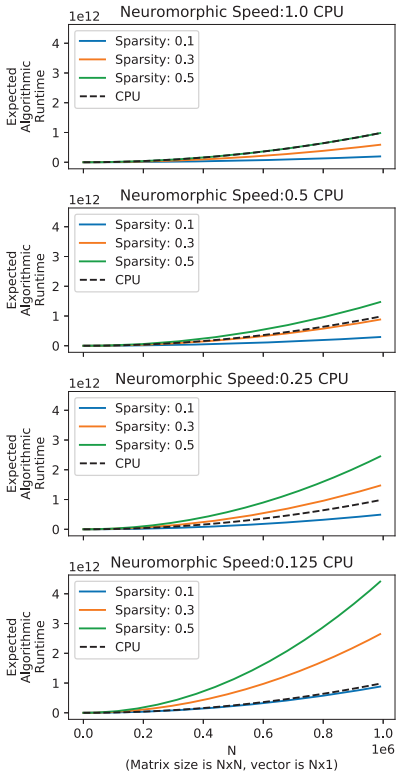


Fig. 2. Effect of sparsity of the matrix (shown as different lines in each plot) and speed of the neuromorphic processor (each of the four plots) on neuromorphic algorithmic performance. The units for the y-axis are the amount of time it takes for one computation on a CPU. We show different neuromorphic speeds in terms of how much slower the neuromorphic system is to perform one simulation time step than the CPU system takes to perform one computation.

it is true that $(A^k)_{ij}$ is the number of walks of length $k$ from $i$ to $j$. However, the methods herein do not translate to computing $A^k\mathbf{x}$, as two length $k$ walks that start at $i$, end at $j$, and use all of the same edges (in a different order) count as two walks but should be regarded as only one walk in our methodologies. Also, our preprocessing step which prevents one neuron from being spiked simultaneously breaks down when dealing with walks (i.e., paths with repeated edges). With a goal of computing $A^k\mathbf{x}$, the former problem will require a deeper analysis of the neuromorphic interpretation of $A^k$ while the latter problem can be resolved with more refined future neuromorphic systems that have a built in mechanism for handling tied spikes. In this way, the future research will highlight the relationships between linear algebra, GraphBLAS, and neuromorphic systems and will also inform more advanced functionality in future neuromorphic hardware.

## ACKNOWLEDGMENT

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

## REFERENCES

[1] 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 preprint arXiv:1705.06963*, 2017.

[2] J. B. Aimone, K. E. Hamilton, S. Mniszewski, L. Reeder, C. D. Schuman, and W. M. Severa, "Non-neural network applications for spiking neuromorphic hardware," in *Proceedings of the Third International Workshop on Post Moores Era Supercomputing*, 2018, pp. 24–26.

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

[4] R. Vuduc, J. W. Demmel, and K. A. Yelick, "Oski: A library of automatically tuned sparse matrix kernels," in *Journal of Physics: Conference Series*, vol. 16, no. 1. IOP Publishing, 2005, p. 071.

[5] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," in *SC'07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. IEEE, 2007, pp. 1–12.

[6] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," in *ACM Sigplan Notices*, vol. 45, no. 5. ACM, 2010, pp. 115–126.

[7] X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey, "Efficient sparse matrix-vector multiplication on x86-based many-core processors," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM, 2013, pp. 273–282.

[8] C. D. Schuman, K. Hamilton, T. Mintz, M. M. Adnan, B. W. Ku, S.-K. Lim, and G. S. Rose, "Shortest path and neighborhood subgraph extraction on a spiking memristive neuromorphic implementation," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, 2019, pp. 1–6.

[9] J. B. Aimone, Y. Ho, O. Parekh, C. A. Phillips, A. Pinar, W. Severa, and Y. Wang, "Provable neuromorphic advantages for computing shortest paths," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, 2020, pp. 497–499.

[10] 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.

[11] 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.

[12] 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.

[13] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[14] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[15] J. D. Smith, W. Severa, A. J. Hill, L. Reeder, B. Franke, R. B. Lehoucq, O. D. Parekh, and J. B. Aimone, "Solving a steady-state pde using spiking networks and neuromorphic hardware," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–8.

[16] 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.

[17] 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.

[18] J. Kepner, P. Aaltonen, D. Bader, A. Buluç, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke *et al.*, "Mathematical foundations of the graphblas," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2016, pp. 1–9.