NeuroEngine: A Hardware-based Event-driven Simulation System for Advanced Brain-inspired Computing

Extended Abstract

Hunjun Lee*, Chanmyeong Kim*, Yujin Chung, and Jangwoo Kim
Department of Electrical and Computer Engineering, Seoul National University

1. Motivation

Brain-inspired computing aims to understand the mechanisms of the brain and reproduce its computational capabilities to advance various areas in computer science. Deep learning is a successful example to greatly improve the field of pattern recognition and classification by utilizing a simplified artificial neural network (ANN). To further exploit the computational capabilities of the brain and thus make more great advances, various studies rely on spiking neural networks (SNNs) which closely mimic the computations of the brain [2, 10, 12, 14].

SNNs enable brain-like computations by adopting neuron models that change their internal states with respect to both incoming spikes and time. Specifically, various studies identify the rich temporal dynamics of neuron models whose internal states gradually change as a significant computational trait [8, 11]. Therefore, emerging studies actively investigate the potential benefits of SNNs based on such complex models. For example, studies by Smith propose a new computational paradigm based on complex neuron models [12, 13]. Meanwhile, Ponulak et al. reproduce the brain's navigating function [9], and other works adopt SNNs for feature extraction [16] or satisfaction problem [4].

To deploy the emerging SNN workloads, researchers rely on SNN simulation systems simulating the complex neuronal dynamics. Unfortunately, existing SNN simulation systems suffer from high computational overhead and **thus, it is highly demanding to design a system that enables fast and energy-efficient SNN simulation.**

2. Limitations of the State of the Art

Prior works have proposed multiple solutions to simulate the complex neuronal dynamics in SNNs; however, existing methodologies are too slow and energy-inefficient due to their software-based frameworks or hardware-based but time-driven execution mechanisms.

There are some SW-based solutions using CPUs or GPUs to simulate SNNs; however, the inefficiencies of the general purpose processors make them unsuitable for large scale simulations. Our profiling results on existing SW-based solutions [5, 17] indicate that they suffer from orders of magnitude slower simulation compared to HW-based simulators.

Other systems deploy SNN simulations on hardware substrates to accelerate the computations [3, 7]. Although they relieve the computational overhead to some extent, they still suffer from the intense overhead to update the internal states due to their time-driven execution mechanisms. To describe the complex neuronal dynamics, the simulator should keep multiple internal states dedicated to each neuron and continuously update the states every time quantum (i.e., timestep) of the simulation. To that end, the time to update the internal states (i.e., neuron computation) becomes a critical performance bottleneck, which takes up to 99.3% of the total simulation latency.

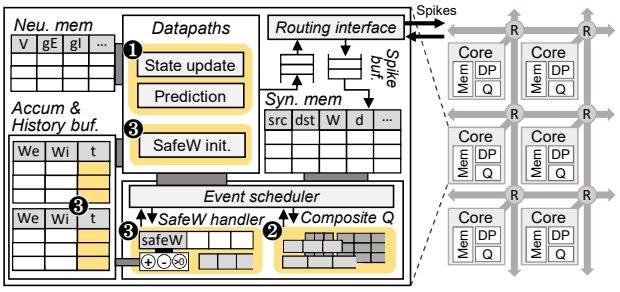

Figure 1: Overview of the NeuroEngine architecture

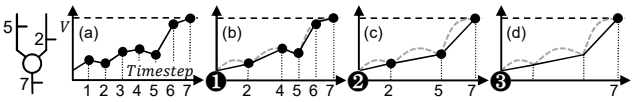

Figure 2: Illustration of the simulation processes with different mechanisms where the dots indicate scheduling and the lines indicate computations ((a) time-driven, (b) baseline event-driven, (c) event-driven with a composite queue, and (d) event-driven with a composite queue and lazy update)

3. Key Insights

In this paper, we present NeuroEngine, a fast and energy-efficient hardware simulator to simulate complex neuronal dynamics efficiently. The key idea is to significantly reduce the number of computations by implementing an event-driven execution mechanism instead of compute-intensive time-driven mechanisms. To achieve the goal, we first identify three architectural challenges in implementing an event-driven simulator. Then, we propose three novel solutions to tackle each challenge. **Our work is very different from existing state-of-the-art SNN simulators [1, 3] as NeuroEngine is the first hardware-based simulator to enable the faithful event-driven simulation for complex neuronal dynamics.**

The event-driven simulator reduces the overall latency and increases energy efficiency by operating with only a small number of computations. Instead of updating the internal states of all neurons every timestep, event-driven mechanisms

* These authors contributed equally to this work

compute the change over several timesteps only when a neuron receives or fires a spike. Considering the highly sparse nature of SNNs, the event-driven mechanism is an efficient candidate to replace the existing simulation methodology.

However, implementing a hardware-based event-driven simulator for SNN simulation introduces three challenges. First, the system requires expensive datapaths to calculate the aggregated change over an arbitrary time interval and to predict the firing time of the neurons. Second, it requires proper data structures to efficiently schedule computations for the neurons that receive or fire a spike. Lastly, the scheduling incurs resource contentions, which limit the parallelization opportunities in SNN simulation. Therefore, we design our simulator, NeuroEngine, using novel schemes to tackle the identified challenges (Figure 1). The key ideas are as follows:

3.1. Optimized Computation Datapath - ❶

First, we design optimized datapaths that support neuron computation for event-driven simulations. We reduce the required bit precision for representing the arbitrary time interval by splitting the interval into multiple fixed sub-intervals. Also, we pipeline the datapaths and add forwarding paths for further optimizations. Then, we replace the power-hungry divider with an approximate shift-based divider for further optimizations. Using our optimized datapaths, we implement our baseline event-driven simulator that efficiently calculates the internal state changes and reduces the number of computations compared to the time-driven simulator (Figure 2-(a)/(b)). However, the simulator suffers from redundant computations in the absence of proper event scheduling units.

3.2. Composite Event Queue Structure - ❷

Next, we devise a composite queue structure to prevent the simulator from scheduling redundant computations. The composite queue consists of a FIFO queue and a priority queue with two auxiliary data structures supporting each queue. The auxiliary data structures detect redundant computations and remove the computations from the queues. By combining our composite queue structure with the baseline simulator, the simulator reduces the number of required computations (Figure 2-(c)). But, the simulator can still suffer from resource contentions making it difficult to handle multiple operations in parallel.

3.3. Lazy Update - ❸

Lastly, we accelerate the overall simulation by parallelizing different operations of SNN simulation. To achieve the goal, we design a light-weight method to predict whether a neuron will fire a spike or not before updating the internal states. The simulator defers the neuron computations until the neuron is expected to fire a spike and then schedules the computations at once (Figure 2-(d)). We call the deferred computations as lazy update, and implement it to complete the design of NeuroEngine. The lazy update bypasses the event queue; therefore, NeuroEngine can parallelize the computations.

4. Main Artifacts

The two main artifacts of this paper are as follows:

NeuroEngine hardware: We implement 32 NeuroEngine cores on top of the scalable NoC in Verilog and synthesize it using Samsung 65nm technology. Among the cores, there is one interface core which acts as programming and reporting terminal between the NeuroEngine chip and onboard CPU.

SNN compilation toolchain: We develop a toolchain to program and deploy SNNs on NeuroEngine. The toolchain provides a programming interface for describing target SNNs using Brian2 [15]. Then, we develop a compiler to generate a connection matrix with parameters and a linker based on a partitioning program [6] to distribute neurons to physical cores. Next, the toolchain sends the configuration data (e.g., mapping data and simulation parameters) to the onboard DRAM. Finally, the toolchain transfers the data stored in DRAM to the NeuroEngine's interface core using onboard CPU. We demonstrate the toolchain by deploying it on the Xilinx VCU1525 FPGA board and use MicroBlaze as an onboard CPU.

5. Key Results and Contributions

In summary, our work makes the following contributions:

- **Challenges in Event-driven SNN Simulation System:** We are the first to identify the three challenges in designing an architecture supporting the event-driven simulation of SNNs with complex neuronal dynamics.
- **Three Novel Architectural Optimizations:** We propose three ideas to solve the identified challenges and implement a fast and energy-efficient event-driven simulator.
- **End-to-End SNN Simulation System:** Our NeuroEngine hardware along with the compilation toolchain enables an end-to-end SNN simulation framework.
- **High Performance Improvement:** NeuroEngine achieves 4.30× speedup and 2.60× energy efficiency over the state-of-the-art hardware-based time-driven SNN simulators.

6. Potential Impact

Our work provides valuable architectural and systemic insights for brain-inspired computing. Brain-inspired computing has been and will be one of the key research topics in both architecture and system communities due to its significant potential benefits. Unfortunately, only a few works focus on designing a system supporting brain-inspired computing using SNNs. Therefore, we believe that our work will provide guidelines for future system designs. First, the three key ideas show how we exploit the SNNs' unique characteristics for optimized architectural support. Second, our end-to-end system implementation with the toolchain provides an example operating model.

References

[1] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, Brian Taba, Michael Beakes, Bernard Brezzo, Jente B. Kuang, Rajit Manohar, William P. Risk, Bryan Jackson, and Dharmendra S. Modha. 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*, 34(10), 2015.

[2] Iulia M Comsa, Thomas Fischbacher, Krzysztof Potempa, Andrea Gesmundo, Luca Versari, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[3] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steve McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan, Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1), 2018.

[4] Gabriel A Fonseca Guerra and Steve B Furber. Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems. *Frontiers in neuroscience*, 11:714, 2017.

[5] Marc-Oliver Gewaltig and Markus Diesmann. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4), 2007.

[6] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[7] Dayeol Lee, Gwangmu Lee, Dongup Kwon, Sunghwa Lee, Youngsok Kim, and Jangwoo Kim. Flexon: A Flexible Digital Neuron for Efficient Spiking Neural Network Simulations. In *Proc. 45th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2018.

[8] Wolfgang Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Networks*, 10(9), 1997.

[9] Filip Jan Ponulak and John J Hopfield. Rapid, parallel path planning by propagating wavefronts of spiking neural activity. *Frontiers in computational neuroscience*, 7:98, 2013.

[10] Saunak Saha, Henry Duwe, and Joseph Zambreno. CyNAPSE: A Low-power Reconfigurable Neural Inference Accelerator for Spiking Neural Networks. *Journal of Signal Processing Systems*, 92, 2020.

[11] James E Smith. Efficient digital neurons for large scale cortical architectures. In *Proc. 41st ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2014.

[12] James E. Smith. *Space-Time Computing with Temporal Neural Networks*. Morgan & Claypool, 2017.

[13] James E Smith. Space-time algebra: a model for neocortical computation. In *Proc. 45th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2018.

[14] James E. Smith. A Neuromorphic Paradigm for Online Unsupervised Clustering. arXiv preprint arXiv:2005.04170, 2020.

[15] Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *Elife*, 8, 2019.

[16] Ping Tak Peter Tang, Tsung-Han Lin, and Mike Davies. Sparse coding by spiking neural networks: Convergence theory and computational results. arXiv preprint arXiv:1705.05475, 2017.

[17] Esin Yavuz, James Turner, and Thomas Nowotny. GeNN: a code generation framework for accelerated brain simulations. *Scientific Reports*, 6, 2016.