Towards a Design Framework for TNN-Based Neuromorphic Sensory Processing Units

Prabhu Vellaisamy and John Paul Shen
Neuromorphic Computer Architecture Lab, Carnegie Mellon University

***Abstract*—Temporal Neural Networks (TNNs) are spiking neural networks that exhibit brain-like sensory processing with high energy efficiency. This work presents the ongoing research towards developing a custom design framework for designing efficient application-specific TNN-based *Neuromorphic Sensory Processing Units* (NSPUs). This paper examines previous works on NSPU designs for UCR time-series clustering and MNIST image classification applications. Current ideas for a custom design framework and tools that enable efficient software-to-hardware design flow for rapid design space exploration of application-specific NSPUs while leveraging EDA tools to obtain post-layout netlist and power-performance-area (PPA) metrics are described. Future research directions are also outlined.**

***Index Terms*—temporal neural networks, neuromorphic sensory processing units**

I. Introduction

Deep Neural Networks (DNNs) have achieved impressive performance on various sensory applications such as computer vision, speech, and object recognition [4]. However, their computing demands have increased exponentially [6] and are unsustainable given their computational and economic costs [12]. Temporal Neural Networks (TNNs) [8]–[10] are a special class of Spiking Neural Networks (SNNs) that use precise spike timings to represent and process information, mimicking the brain's *temporal* computational paradigm.

TNNs employ simple feed-forward networks operating on spike-timing relationships, unlike DNNs that use high-dimensional tensor processing. One distinguishing feature of TNNs is temporal encoding, in which information is represented by the relative timing of the spikes, requiring just a single spike to encode one value. Furthermore, TNNs use biologically plausible Spike Timing Dependent Plasticity (STDP) local learning rules, enabling them to perform online continuous learning, while DNNs use compute-intensive backpropagation, which separates training and testing phases. These two distinctive attributes of TNNs make them the most "neuromorphic" or brain like.

[1] demonstrates the efficacy of using TNNs for unsupervised time-series clustering, which can be employed for edge-native applications such as anomaly detection and healthcare monitoring. A microarchitectural framework for *direct* CMOS implementation of TNNs has been proposed in [5], and successfully used to implement key TNN building blocks of: neurons, columns, synapses, and the STDP learning rules.

This paper presents the current research in developing an automated software-to-hardware design framework for designing compelling sensory processing units based on TNNs, namely *Neuromorphic Sensory Processing Units* (NSPUs). Targeted applications for NSPUs include diverse sensory signal processing for edge, mobile, and IoT devices. Due to the potential of extreme energy efficiency, NSPUs can enable always-on, on-device, autonomous, and highly distributed processing. We envision a design framework for NSPUs that incorporates: (1) a scalable microarchitecture model for implementing application-specific NSPUs, and (2) a suite of supporting tools, including a PyTorch-based software simulator to perform rapid design space explorations, and automated tools for RTL-to-GDSII flow to produce post-layout power-performance-area (PPA) metrics for NSPUs.

II. Background & Initial Results

Due to their neuromorphic attributes, NSPUs exhibit extreme energy efficiency and are ideal for always-on edge-native processing devices. Initial results in [5] reports post-synthesis evaluation of the TNN column implementations in 45 nm and demonstrates the efficacy of using TNNs for MNIST digit classification using supervised learning. A *large* column configuration of 1024 synapses and 16 output neurons consists of 1.7M gates, with the area and power footprint of 1.65 mm^2 and 7.96 mW, which is less than 1% of the area and power budget of mobile SoCs [5].

[1] illustrates the efficacy of using TNNs for unsupervised time-series clustering of 36 UCR benchmark datasets and demonstrates that TNN designs either outperform or are competitive to the state-of-the-art algorithms. Furthermore, hardware complexity analysis is performed by doing standard technology scaling to achieve predictive 7nm results from 45 nm post-synthesis results for the designs employed for the 36 datasets. Ongoing work on developing and incorporating custom TNN column-based macro cells in the ASAP7 library shows promising PPA improvements ($\sim$17% less power, $\sim$16% faster, and $\sim$27% less area) - the largest TNN column used in [1] with 6,750 synapses consumes just 0.054 mm^2 area, 39 μW power, and 28.14 ns computation time. These results establish that developing NSPU

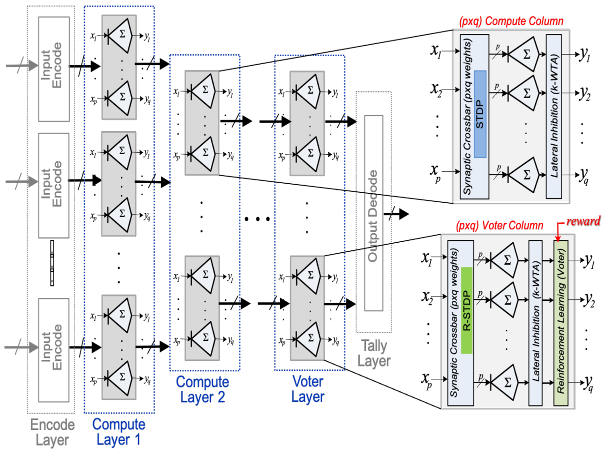

Figure 1: Hierarchical TNN Organization

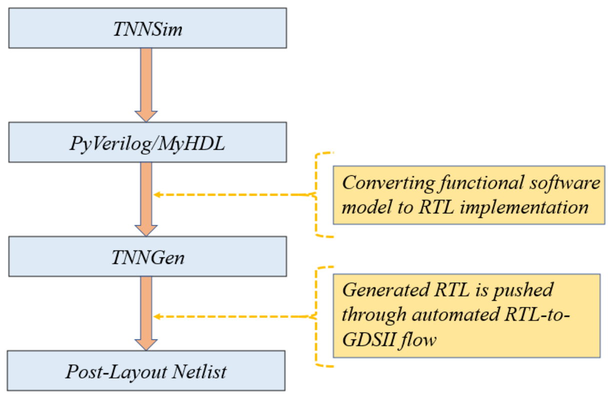
TNNSim
PyVerilog/MyHDL
TNNGen
Post-Layout Netlist
Converting functional software model to RTL implementation
Generated RTL is pushed through automated RTL-to-GDSII flow

Figure 2: Custom NSPU Design Framework Flow

designs capable of online unsupervised and supervised learning has enormous potential in edge-native sensory processing applications.

III. NSPU Microarchitecture Framework

Fig. 1 depicts a high-level hierarchical architecture of TNNs, where each neuron has p synaptic inputs and one output, and each synapse carries a local synaptic weight that is updated based on the relative timing of incoming spike to that synapse and the outgoing spike from the neuron body. The STDP learning rule determines the synaptic weight update, and through it, a neuron *learns* an input feature by adapting the weight to match with the corresponding input pattern. A detailed description of TNN microarchitecture is provided in [5].

The fundamental building block for NSPUs is a *column*; we denote a column configuration of pxq as containing q excitatory neurons and a synaptic crossbar connecting p inputs to the q neurons via pxq synapses. Larger TNN designs can be constructed by stacking multiple columns to form a multi-column layer, and also by cascading multiple layers into a large multi-layer TNN. Multi-layer TNNs have been utilized for MNIST digit recognition. [10] has shown a 4-layer TNN with 3.096M synapses can achieve 1% error rate on MNIST.

IV. Automated Toolsuite for NSPUs

The ongoing research aims to develop a custom design framework for efficient and automated mapping of NSPU designs from software to hardware, facilitating rapid design space exploration and yielding post-layout PPA values from place-&-route. The entire framework will consist of the following - (i) *TNNSim*, a functional TNN simulator implemented in PyTorch [7], (ii) *TNNGen* - a script suite for the Cadence toolchain for performing automated RTL-to-GDSII flow to generate predictive 7 nm PPA values, (iii) the ASAP7 [2] standard cell library, (iv) a custom macro library containing TNN functional units developed as an extension to the ASAP7 library that leads to optimized TNN column design, and (v) a Python library function, such as PyVerilog [11] or MyHDL [3], to convert Python codes to RTL.

The design framework, depicted in Fig. 2, will allow high-level users to experiment with constructing application-specific NSPU designs while getting accurate hardware complexity analysis results in predictive 7 nm. A typical user will implement PyTorch NSPU models for specific sensory processing applications by invoking TNNSim's built-in library of TNN functional modules, finetune the architectures to achieve the best performing models and then choose to initiate the hardware flow. Once initiated, PyVerilog or MyHDL is leveraged to map the NSPU functional models to RTL. This flow is more optimal than high-level synthesis (HLS) flows as it allows direct Verilog or VHDL scripting and outputs desired optimal RTL implementations. Consequently, TNNGen is invoked, and automated RTL-to-GDSII flow occurs, involving the Cadence tools like Genus for synthesis, Innovus for place-&-route, and Voltus for accurate signoff power numbers.

V. Future Research Directions

A complete NSPU design framework, incorporating TNNSim and TNNGen, is being developed. Using this framework and its suite of tools, researchers and developers can explore the design and implementation of a wide range of application-specific NSPUs, targeting IoT, edge, and mobile devices. This framework can also facilitate the design of specialized NSPUs for edge-native acceleration of AI/ML workloads that consume less than 10 mW of power. Furthermore, this framework can serve as a foundation for implementing arbitrary space-time functional units [8] in the near future.

References

[1] S. Chaudhari, H. Nair, J. M. Moura, and J. P. Shen, "Unsupervised clustering of time series signals using neuromorphic energy-efficient temporal neural networks," in *2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021.

[2] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[3] J. Decaluwe, "Myhdl: a python-based hardware description language." *Linux journal*, no. 127, pp. 84–87, 2004.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, 2015.

[5] H. Nair, J. P. Shen, and J. E. Smith, "A microarchitecture implementation framework for online learning with temporal neural networks," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021.

[6] OpenAI, "AI and Compute," https://openai.com/blog/ai-and-compute.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[8] J. Smith, "Space-time algebra: A model for neocortical computation," in *International Symposium on Computer Architecture (ISCA)*, 2018.

[9] J. E. Smith, "Space-time computing with temporal neural networks," *Synthesis Lectures on Computer Architecture*, vol. 12, 2017.

[10] J. E. Smith, "A temporal neural network architecture for online learning," *arXiv preprint arXiv:2011.13844*, 2020.

[11] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2015, pp. 451–460.

[12] N. Thompson, K. Greenewald, K. Lee, and G. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2020.