StreamBrain: An HPC DSL for Brain-like Neural Networks on Heterogeneous Systems

Artur Podobas*, Martin Svedin*, Steven W. D. Chien*, Ivy B. Peng†, Naresh Balaji Ravichandran*,
Pawel Herman*, Anders Lansner*‡, Stefano Markidis*
*KTH Royal Institute of Technology, Sweden
†Lawrence Livermore National Laboratory, USA
‡Stockholm University, Sweden

***Abstract*—We introduce *StreamBrain* – a high-performance DSL for Brain-like Neural Networks. StreamBrain supports multiple backends such as FPGAs, GPUs, and CPUs on heterogeneous HPC systems while providing a convenient Keras-like interface to users. We show that training MNIST dataset on the BCPNN model only takes 15 seconds. We empirically show that batching is critical for the BCPNN model as it allows the computational intensity to be controlled. Finally, we explored the resilience of the BCPNN model to reduced width of the numerical representation and showed that the mantissa of the double-precision (DP) computation could be reduced to 9-bit, yielding nearly twice the performance of the original double-precision implementation.**

I. Introduction

The brain-like Bayesian Confidence Propagation Neural Network (BCPNN [1]) is a machine learning algorithm with a mature and solid theoretical foundation. The BCPNN is a probabilistic graphical model that employs graphs representing a problem by combining probability and graph theories. BCPNN models problem with a collection of random variables (x_1, x_2, ..., x_n, y_1, ..., y_m, z_1, ..., z_l) as joint distribution p(x_1, x_2, ..., x_n, y_1, ..., y_m, z_1, ..., z_l). Each node of the graph represents a random variable, while the edge represents the dependence or correlation between the variables. The training phase determines the weights (w) and biases (b) that characterize the connection, which is later used for prediction. Currently, the performance of BCPNN on modern heterogeneous systems remains unexplored. We introduce *StreamBrain* – a high-performance DSL for deploying the BCPNN model on heterogeneous HPC systems that are accelerated by FPGAs or GPUs. This work introduces the design and implementation of StreamBrain. Our main contributions are as follows:

1) StreamBrain, a Keras-like library for implementation, evaluation, and deployment of BCPNN for use in future high-performance computers and data-centers,
2) Analysis, implementation, validation, and empirical evaluation of three different BCPNN backends for CPUs, GPUs, and FPGAs, on the MNIST data-set,
3) Empirical evaluation on both batching (a well-known deep-learning optimization) and variable-precision arithmetics on the BCPNN model.

II. StreamBrain Framework

We design the StreamBrain framework for deploying emerging brain-like machine learning models on HPC and data-center systems. The StreamBrain framework provides a Python interface that is familiar to ML programmers, albeit the core computational-intensive parts are refactored out and optimized for OpenMP, CUDA, or FPGA backends that are common on existing heterogeneous systems. We aspire to support two types of operations in StreamBrain. The first operation is *streaming*, where a third party (e.g., a network card or camera) delivers input to the application at variable (and unpredictable) latencies. The second mode is called *batched* mode and is similar to existing DL frameworks. This paper focuses on exploring the *batched* mode for both training and inference.

A. StreamBrain DSL

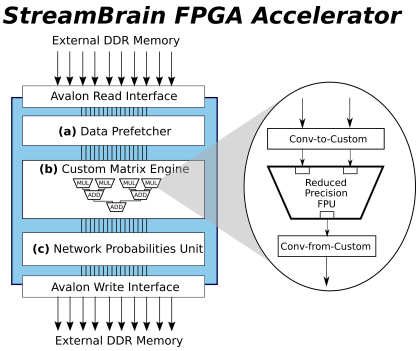
We created a Keras-like interface for StreamBrain to capture the simplicity so that users can focus on their algorithmic optimizations other than code optimizations for different hardware backend. A BCPNN model can be viewed as a stack of layers. Each layer can individually take in an input array and give an output array. Although it is still unclear how multi-layer BCPNNs will be implemented and used, as the current BCPNNs are shallow networks, our interface design is extensible for future complex network architecture. In this work, we use a simple two-layer BCPNN that combines supervised and unsupervised training. Listing 1 show that such a BCPNN model can be implemented with few lines of code.

```
# 1. Create empty network
model = BCPNN.Network(...)
# 2. Add layers
model.add(BCPNN.StructuralPlasticityLayer(..))
model.add(BCPNN.DenseLayer(...))
# 3. train and evaluate
model.fit(dataset=(...))
model.evaluate(dataset=(...))
```

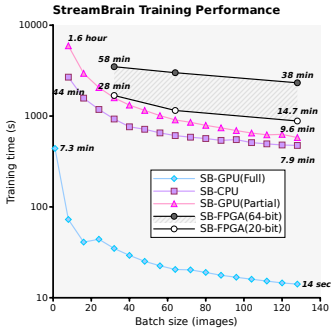
Listing 1: A simple BCPNN Network described using the StreamBrain framework

III. Implementation

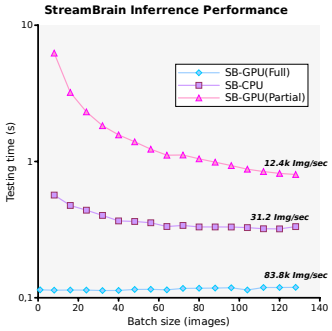
The StreamBrain implementation is based on the Python Numpy module and expressing the operation in BCPNN Algorithms as arrays and tensor operations. We implemented



(a) StreamBrain FPGA Accelerator



(b) The training performance of MNIST.



(c) The inference performance.

multiple backends to support accelerators on existing HPC systems. The backends can be selected through environment variables and are interfaced with the StreamBrain framework through PyBind11. We offer four different backends operating at two different granularities. Three of the backends operate at a level that is tightly integrated with the Python framework, while one backend is fully ported to the GPUs (and is the one with the highest performance).

FPGA Backend Figure 1a conceptually illustrates the StreamBrain FPGA backend called SB-FPGA that targets the Stratix V DE5-Net board. The main components were implemented using Intel's OpenCL SDK [2] for FPGAs. Custom, reduced precision floating-point units were generated using the FloPoCo [3] library and manually included as custom OpenCL libraries. This allows us empirically evaluate the impact of non-standard (other than IEEE-754) floating-point representations at speed, which is hard to do in software (floating-point emulation is slow). Overall, the system includes: **(a)** a data prefetcher that fetches neural network data from external DDR3 memory, **(b)** a custom matrix-multiplication engine capable of different floating-point representations, and **(c)** a custom network unit for update network weights. The accelerator run between 150 (64-bit FPUs) and 223 MHz (20-bit FPUs) and consumes more than half of the DSP (computational) blocks of the FPGA.

GPU Backend We ported two versions of StreamBrain to the GPU using CUDA: **(i)** only the most compute-intensive parts called SB-GPU(Partial), and **(ii)** the entire StreamBrain framework called SB-GPU(Full).

CPU Backend We parallelized compute-intensive components using OpenMP. We call this version SB-CPU.

IV. Evaluation

We evaluated StreamBrain using the Intel DE5-Net FPGA board as well as a HPC node (Kebnekaise at HPC2N in Umeå) with NVIDIA V100 GPUs. The training and testing uses MNIST dataset for character recognition.

Training Performance Figure 1b reports the training time at various batch sizes. Both the SB-CPU and SB-GPU(Partial) backend exhibit a large span in execution time, ranging from 45 minutes down to 8 minutes. This significant reduction in runtime highlights the importance of tuning the batch size in BCPNN models.

The SB-GPU (Full) backend achieved 26x-38x speedup over the SB-CPU backend. It also reached 40x-45x speedup over the SB-GPU(Partial) backend. Interestingly, the SB-GPU (Full) backend can finish training the entire MNIST dataset on the BCPNN model in less than 15 seconds, which is comparable to the performance of modern deep-learning frameworks.

The shaded area reports the FPGA backend (SB-FPGA) when varying the number representation in bits. There is 2.07x-2.63x performance gain from using a smaller numerical representation (20-bit) compared to the larger one (64-bit) while the accuracy sustains.

Inference Performance Figure 1c reports the inference performance of the BCPNN using the MNIST images dataset. StreamBrain achieves the highest throughput (88k img/s) using the SB-GPU(Full) backed. Interestingly, the performance degraded at large batch-size on SB-GPU(Full), where moving from a single to 128 images reduces the peak throughput by 5%. Both SB-CPU and SB-GPU(Partial) implementations have performance improvement monotonically as a function of larger batch-size likely due to fewer transfers.

V. Conclusion

We designed and developed *StreamBrain* framework, a DSL with support for heterogeneous backends, including FPGAs, GPUs, and CPUs. We evaluated the training and inference performance of the BCPNN model in the StreamBrain framework on multiple systems.

Acknowledgment

The work is supported by the European Commission H2020 program, Grant Agreement No. 801039 (EPiGRAM-HS).

References

[1] N. B. Ravichandran, A. Lansner, and P. Herman, "Learning representations in bayesian confidence propagation neural networks," *arXiv preprint arXiv:2003.12415*, 2020.

[2] U. Czajkowski, Tomasz S et al.and Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras, and D. P. Singh, "From opencl to high-performance hardware on fpgas," in *22nd international conference on field programmable logic and applications (FPL)*. IEEE, 2012, pp. 531–534.

[3] F. De Dinechin and B. Pasca, "Designing custom arithmetic data paths with flopoco," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.