StreamBrain

An HPC Framework for Brain-like Neural Networks on CPUs, GPUs, and FPGAs

Artur Podobas[1], Martin Svedin[1], Steven W.D. Chien[1], Ivy B. Peng[3], Naresh Balaji Ravichandran[1], Pawel Herman[1], Anders Lansner[1,2], Stefano Markidis[1]

[1] KTH Royal Institute of Technology, Sweden

[2] Stockholm University, Sweden

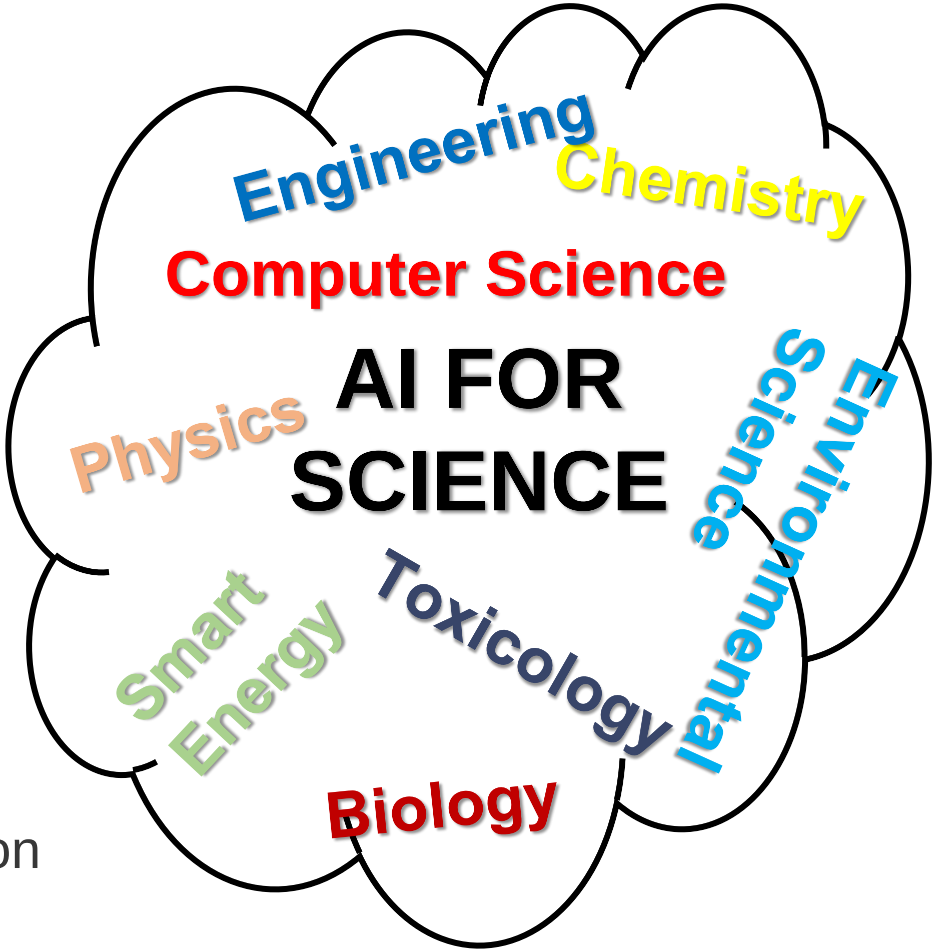
[3] Lawrence Livermore National Laboratory, US

European Commission

Horizon 2020
European Union funding
for Research & Innovation

Introduction

- Surge in Machine Learning (ML) popularity
- Adoption in multiply scientific fields
- Many based on backpropagation
  - Labeled data
  - Supervised learning
    - Minimize loss function
    - Backpropagate errors
    - Update gradients
- There exist other frameworks with mature theory
  - One such example is **B**ayesian **C**onfidence **P**ropagation **N**eural **N**etwork (BCPNN)



Motivation

- Why explore alternative Deep-Learning strategies such as BCPNN?
  1. BCPNN is brain-inspired → Gain insights into how we humans compute?
     - Spiking and Non-Spiking formulations → Map to neuromorphic systems?
  2. BCPNN supports supervised, semi-supervised, and unsupervised learning
     - No need to have a labeled data
     - Excellent for bringing order to seemingly un-labeled data
  3. *Learning rules are local*
     - BCPNN theory allow each unit (think: neuron) to be updated based only on local state
     - Can cater to extreme-scale parallelism
- Challenge: *Currently, no easy-to-use or high-performance implementation of BCPNN ⇔ hard for non-experts to use*

What is BCPNN?

Bayesian Confidence Propagation Neural Network

(Part 1 of 3)

- Brain-like neural network framework
- Based on Hebbian learning and Bayes' theorem[1]
- Two types of units:
  - **H**yper**C**olumn **U**nits (HCUs)
    - Main building block
    - Captures and learns a certain variable/feature
    - Contains multiple MCUs
  - **M**ini**C**olumn **U**nits (MCUs)
    - Models a certain instance of associated HCU's variable

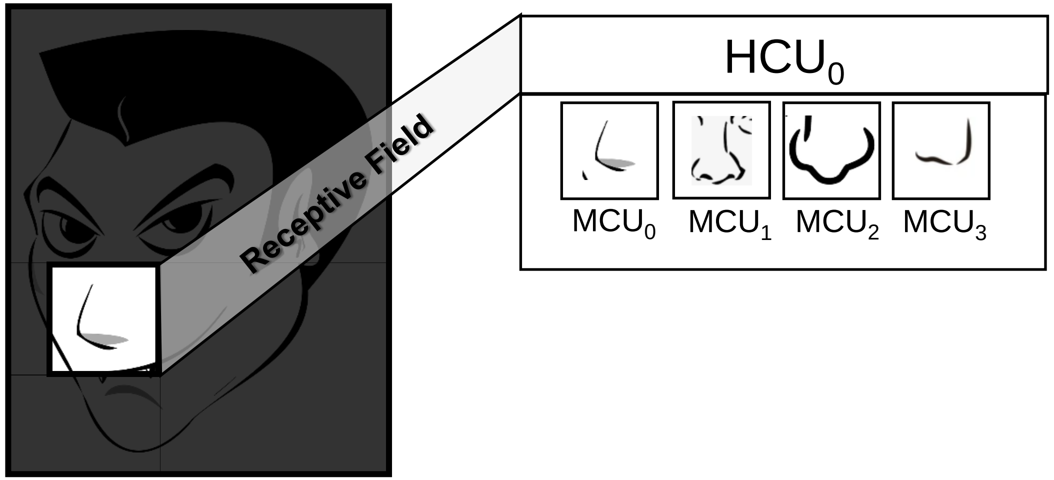


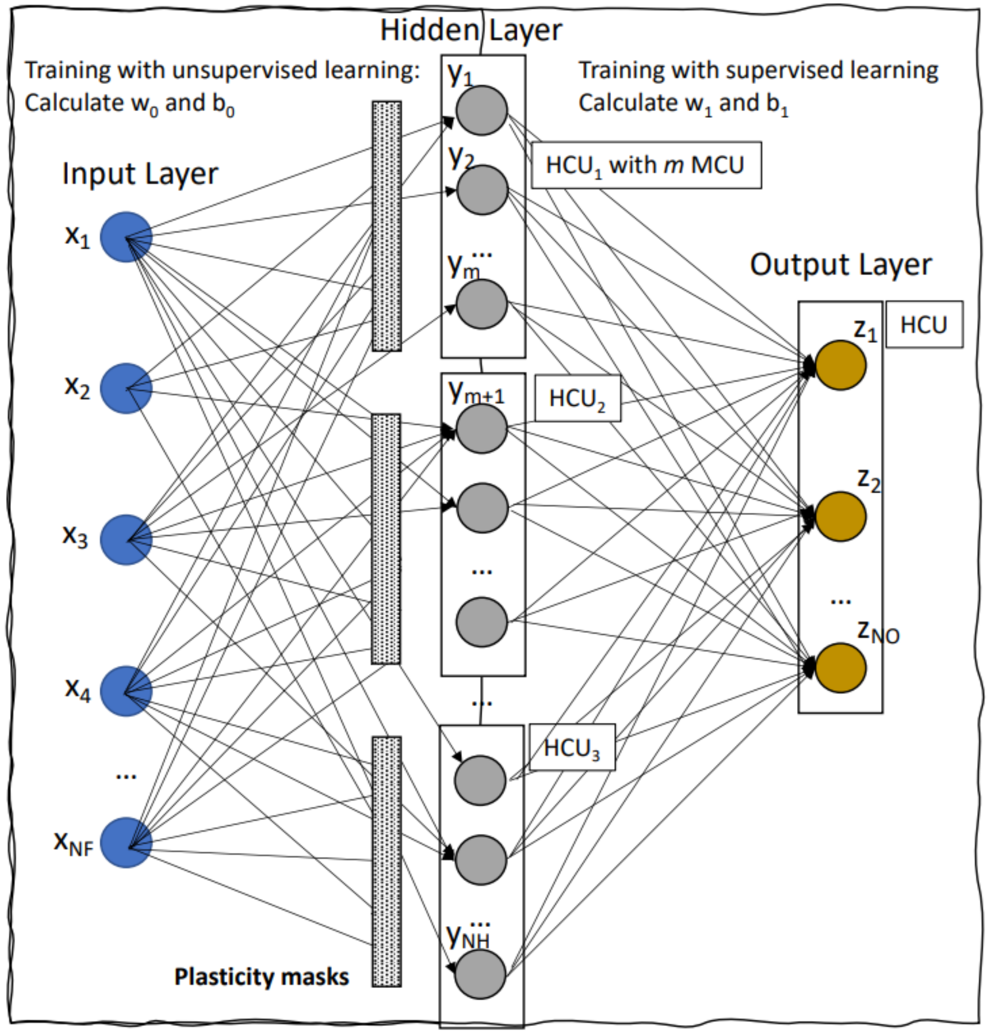
Image Source: Pixabay.com (creative commons)

[1] *Learning representations in Bayesian Confidence Propagation Neural Networks* Ravichandran et al., 2020

Bayesian Confidence Propagation Neural Network

(Part 2 of 3)

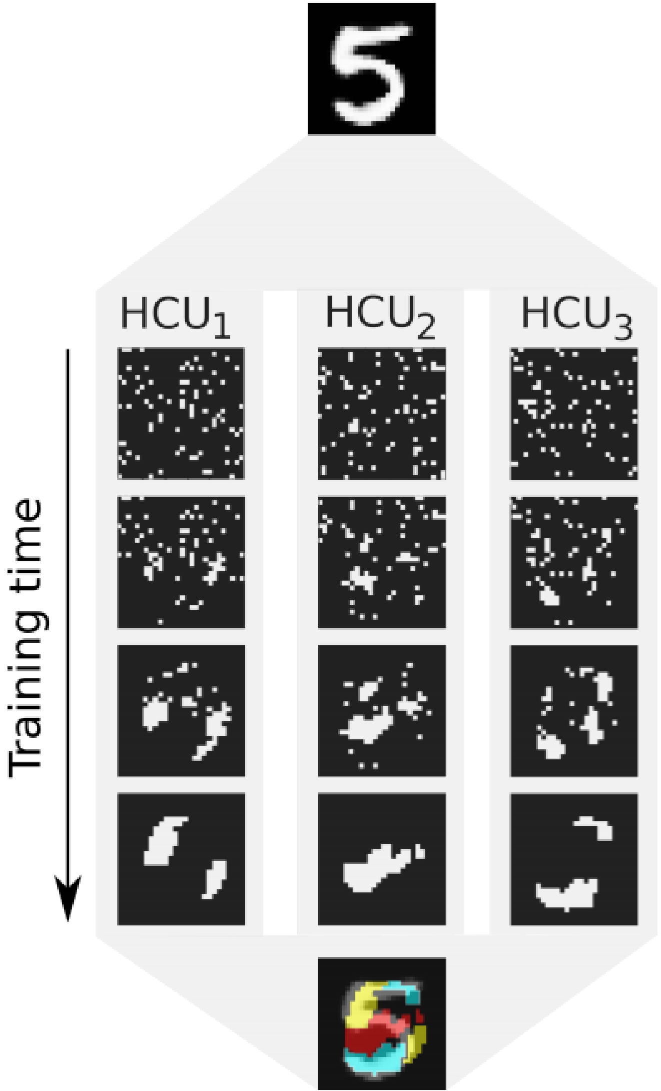
- Sample Network
  1. Input layer
     - Most often images
  2. Hidden Layer
     - Each HCU has a different receptive field
       - “Looks” at different parts of the image
       - Structural Plasticity
       - Sparsity
     - Can learn without supervision
  3. Classification Layer
     1. Classifies the data (e.g., types of image)
     2. Train with supervision (or even with Backprop ⇔ Hybrid BCPNN+SGD networks)



Bayesian Confidence Propagation Neural Network

(Part 3 of 3)

- Example 3 HCUs hidden network
  - The number “5” in the input layer
- Execution:
  1. Initially receptive fields of HCUs randomized (looks at random location in input image)
  2. As training continues, structural plasticity teaches HCUs where in the image to capture most information
  3. Finally, we end up with three HCUs capturing different aspects of the input image.
- This is called *structural plasticity*

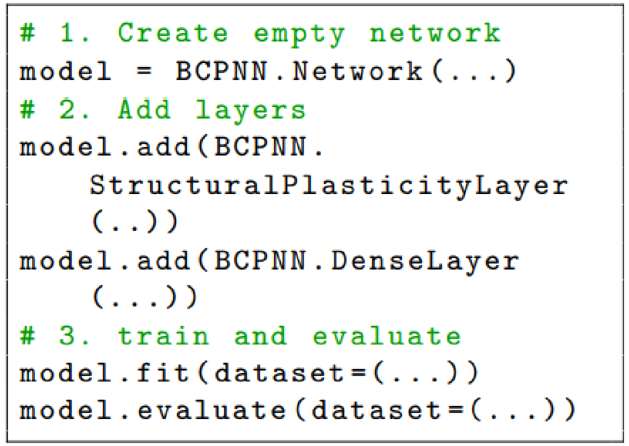


StreamBrain

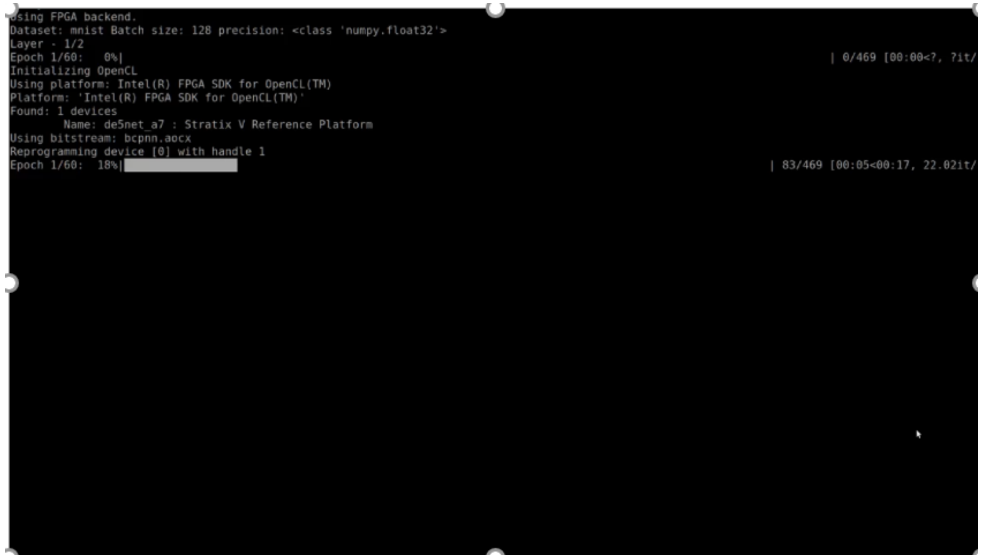
StreamBrain

(Part 1 of 4)

- A high-performance DSL for accessing and using BCPNN
- Simple, intuitive, easy-to-use Keras-like interface
- Based on Python
- Multiple backends for High-Performance Computing
- Supports batching



```
# 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=(...))
```

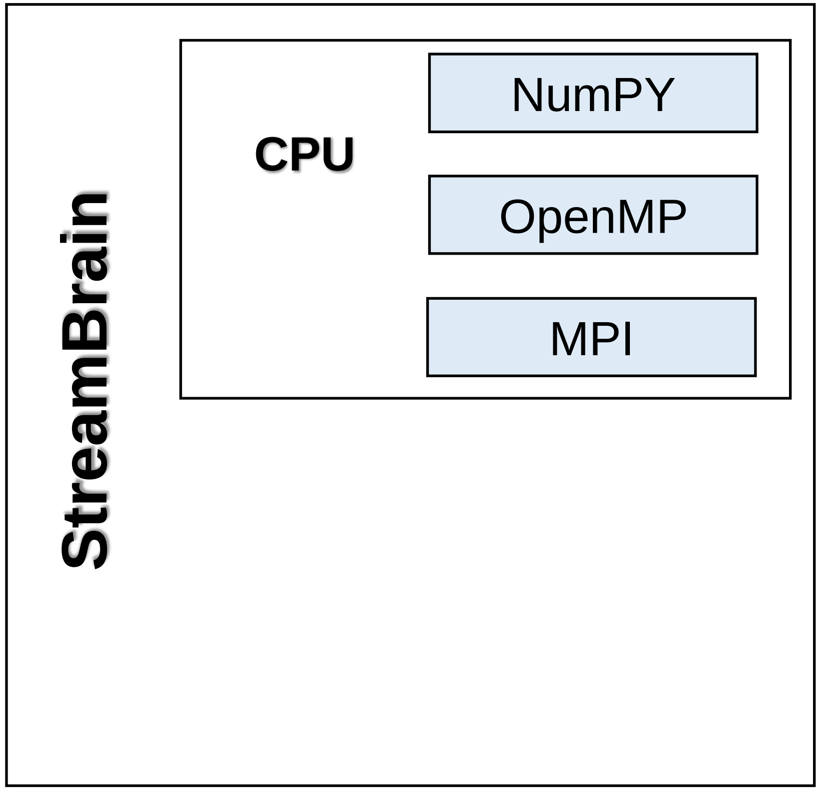


```
Using FPGA backend.
Dataset: mnist Batch size: 128 precision: <class 'numpy.float32'>
Layer - 1/2
Epoch 1/60:   0%|                                                          | 0/469 [00:00<?, ?it/
Initializing OpenCL
Using platform: Intel(R) FPGA SDK for OpenCL(TM)
Platform: 'Intel(R) FPGA SDK for OpenCL(TM)'
Found: 1 devices
        Name: de5net_a7 : Stratix V Reference Platform
Using bitstream: bcpnn.aocx
Reprogramming device [0] with handle 1
Epoch 1/60:  18%|████████                                          | 83/469 [00:05<00:17, 22.02it/
```

Video of training BCPNN using StreamBrain.

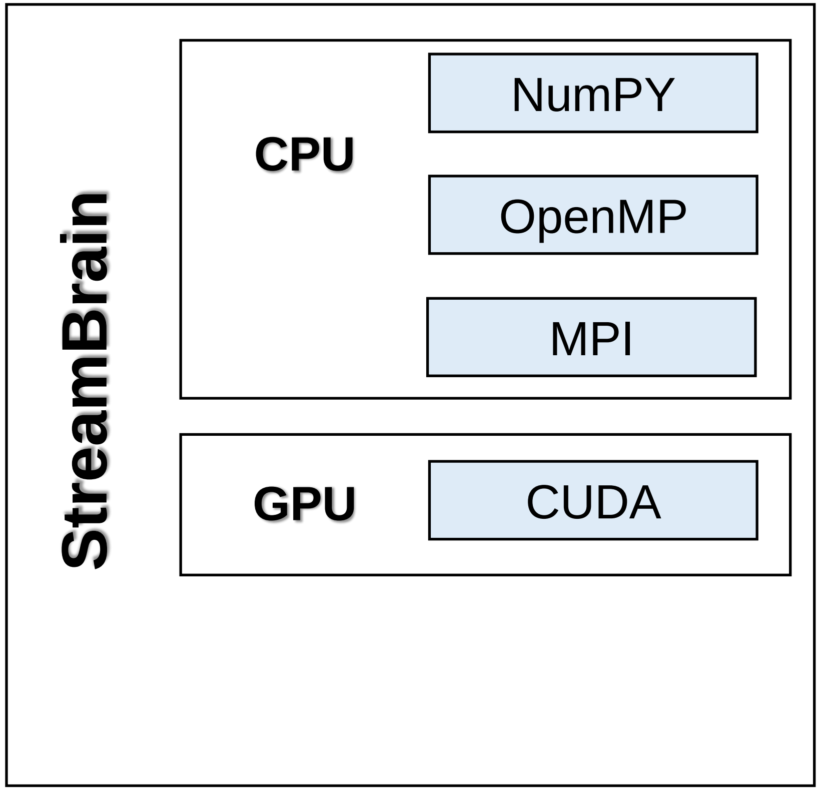
StreamBrain
(Part 2 of 4)

- Python-based backend
    - Leverages NumPy (and thus BLAS libraries where available)
    - Easy to integrate into existing ML pipelines
- OpenMP-based backend
    - Used for performance-critical sections
    - OpenMP 2.0+ directives
        - Primarily data-parallel directives used
    - Partially vectorized for AVX256/512
- MPI-based backend
    - Data-parallelism
        - Sub-batches per MPI process
        - Intra-process parallelism using OpenMP



StreamBrain
(Part 3 of 4)

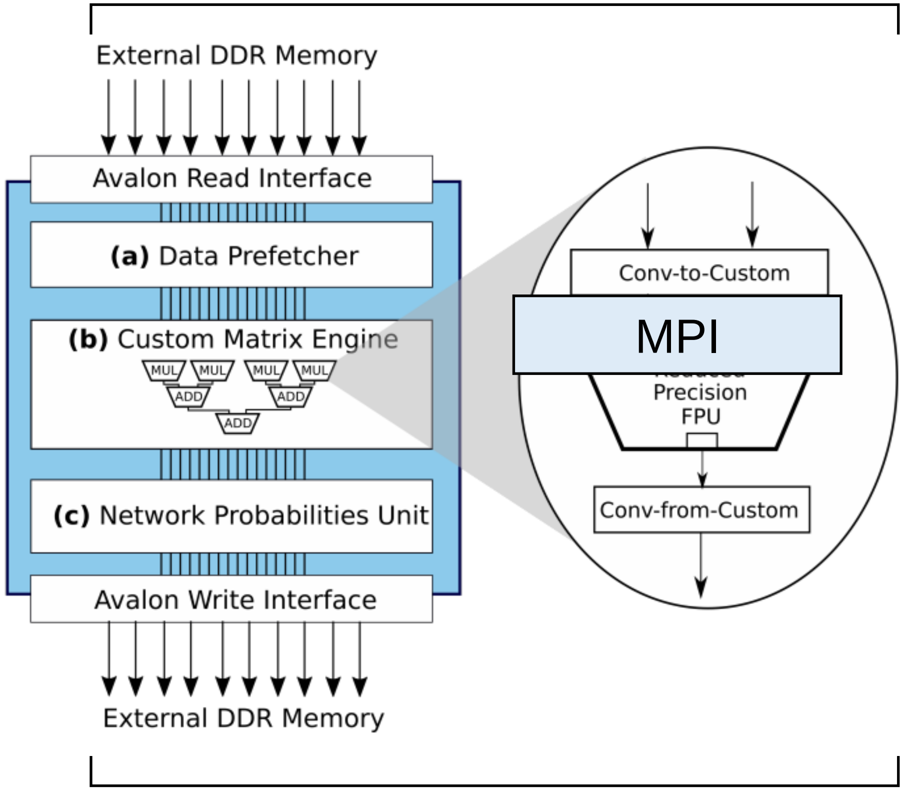
- CUDA-based version of the entire pipeline
  - Very high performance
- Both training and inference loop on the GPU
  - Reduce transfer overheads
- Support for modern Nvidia GPUs
  - Nvidia V100, and A100



StreamBrain

(Part 4 of 4)

- Used to explore architectural design-decisions
- Targets performance-critical StreamBrain sections
- Described using Intel OpenCL HLS
- Support for different numerical representations (through FLoPoCo[1])
  - IEEE-754 single-precision
  - BrainFloat-16 (BF-16)
  - But also BF-14,-15,-20,-24-,28
    - Add/Sub, Multiplication, Division, Log
- Important to understand the impact of numerical precision, especially for future architectures!



[1] *Designing custom arithmetic data paths with FloPoCo,* Dinechi et al., 2011

Results

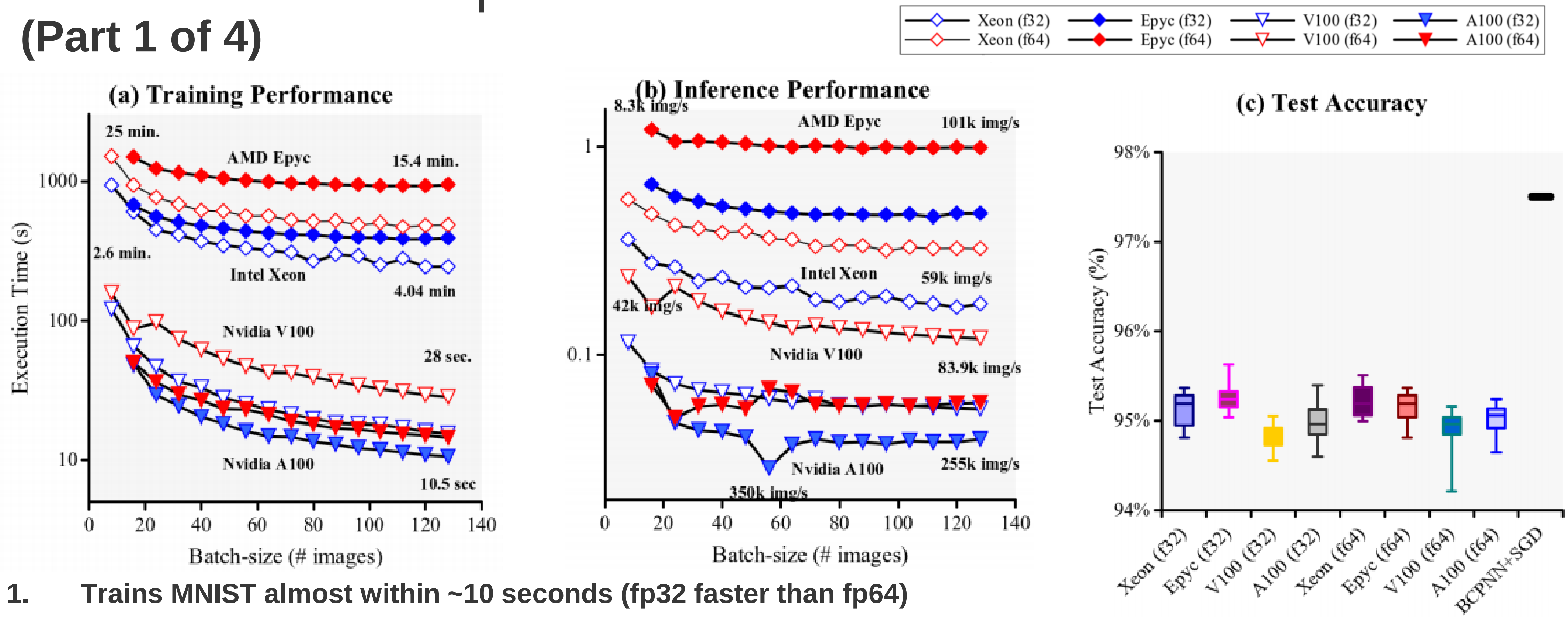
Experimental Platform

- StreamBrain performance on:
- General-Purpose CPUs:
  - Intel Xeon E5-2698e3, E5-2690v4, Gold 6132, Core i5-8400, AMD Epyc
  - Nvidia Volta-100 and Ampere-100
  - Field-Programmable Gate Arrays (FPGAs, Stratix V)
  - High-Performance Computing Systems
    - KTH Beskow Cray XC40 Supercomputer
    - HPC2N Kebnekaise (at Umeå University) Supercomputer
- Datasets used:
  1. MNIST Data-set benchmark (28x28 black-white images)
  2. STL-10 Data-set (96x96 RGB images)
- Network size: 3000 MCUs (# HCUs subject to optimization)

Figure src: http://clipart-library.com/race-car-cartoon-pictures.html (Creative commons)

Results: MNIST performance

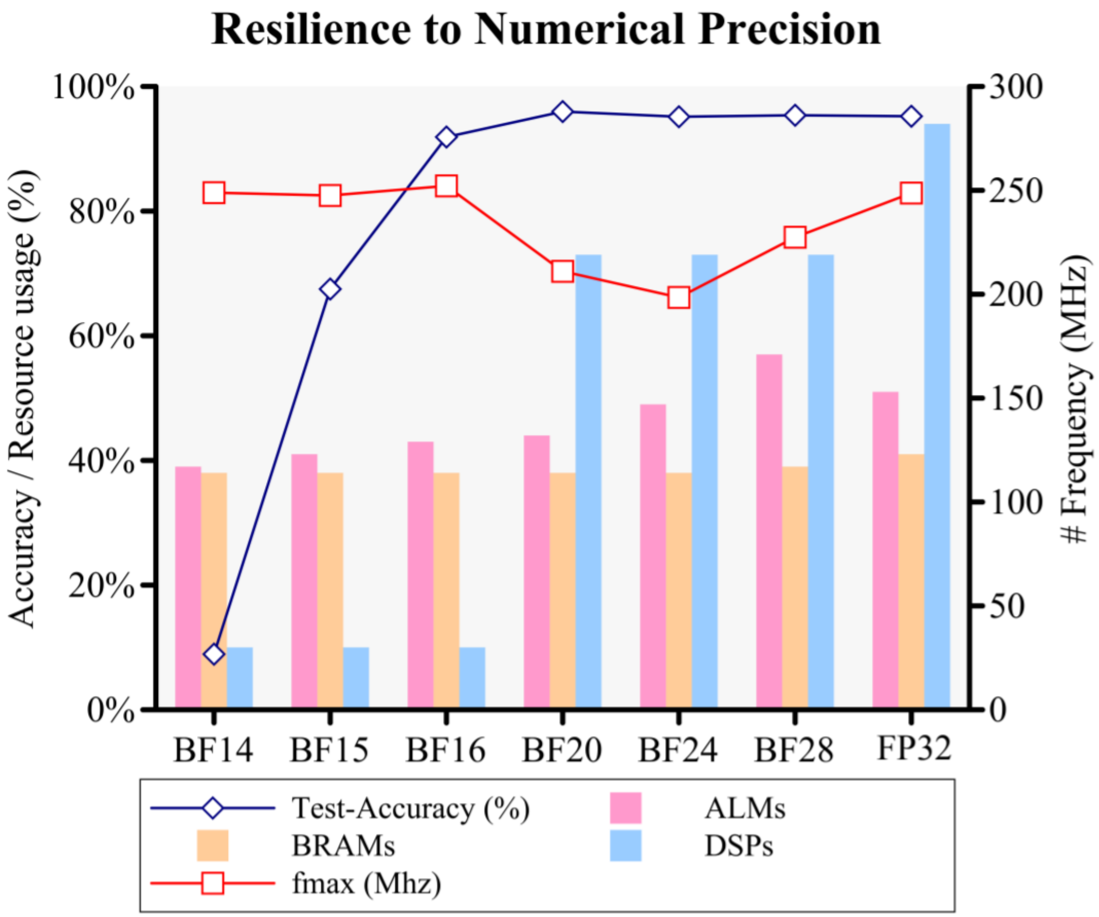
(Part 1 of 4)



1. Trains MNIST almost within ~10 seconds (fp32 faster than fp64)
2. Inference times reaches 350k Img/s.
3. Little difference in single vs. double-precision (both ~95% test accuracy).
4. Hybrid BCPNN + SGD yields correct 97.5%+ accuracy (in line with previous work)
5. Contrast: PyTorch (Backprop) on A100 to reach ~95.5% takes 30+ seconds; StreamBrain for the same accuracy only ~10 seconds!

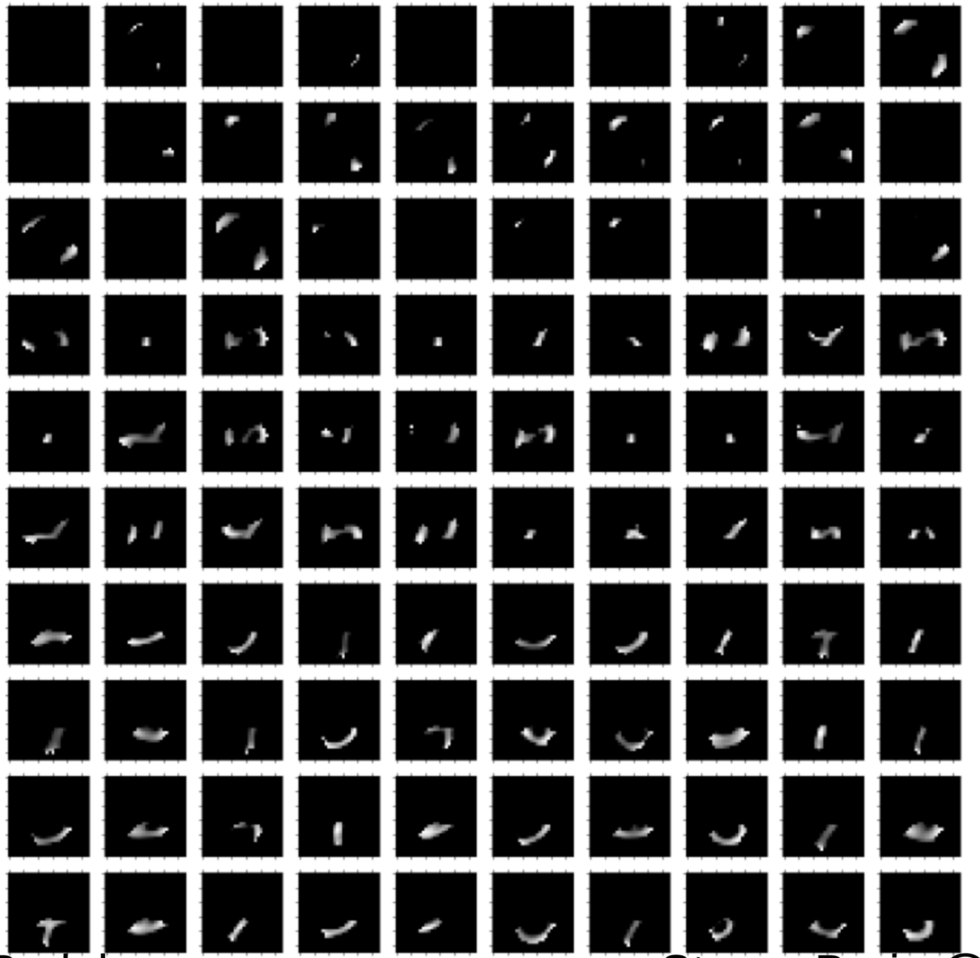
Results: FPGAs and Numerical Representation
(Part 2 of 4)

- Can we go with a lower/smaller numerical representation?
- Yes, we can:
  - All the way to BF16
  - BF15 degrades test accuracy to ~65%
  - BF14 degrades it to chance (~10%)
- What happens?

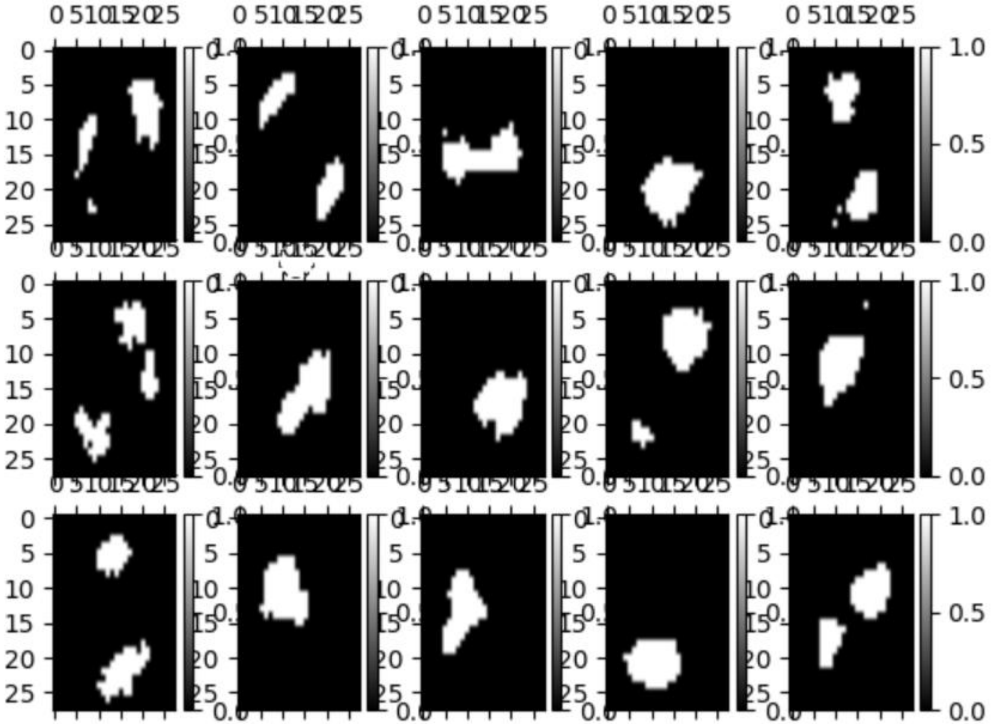


Results: FPGAs and Numerical Representation
(Part 2 of 4)

IEEE 754 (single-precision, 32 bit)



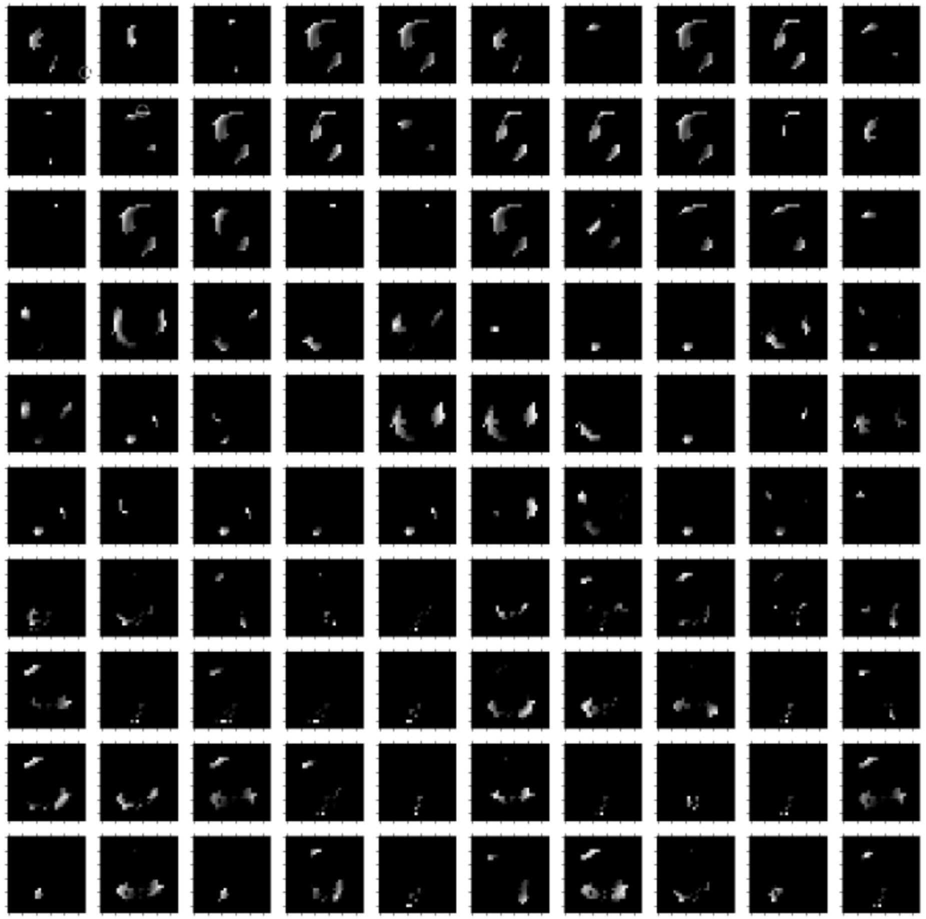
MCUs



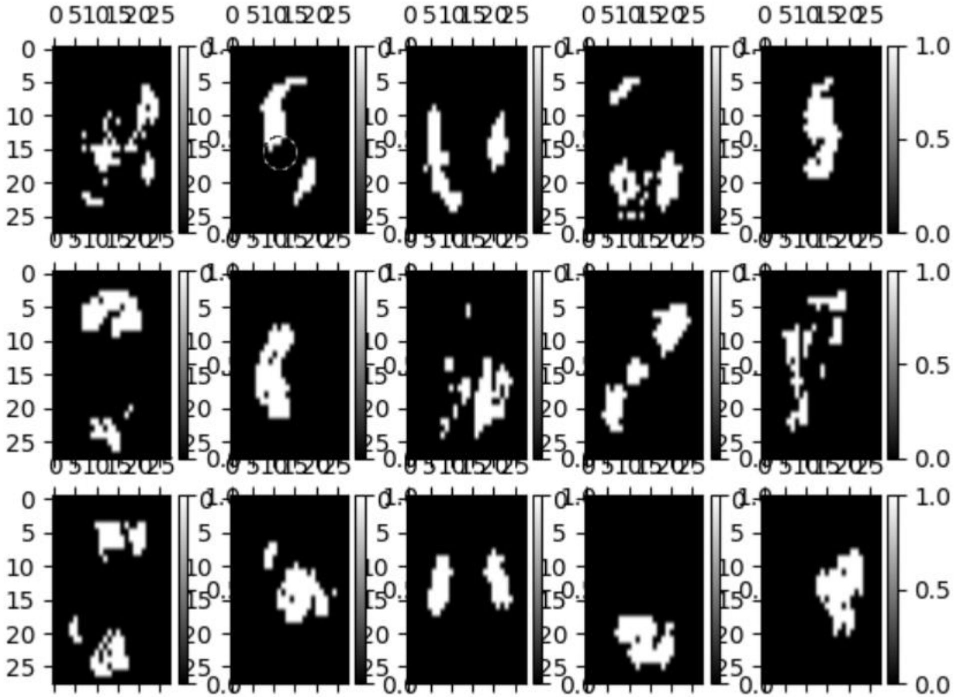
HCUs

Results: FPGAs and Numerical Representation
(Part 2 of 4)

BrainFloat 16 (16-bit)



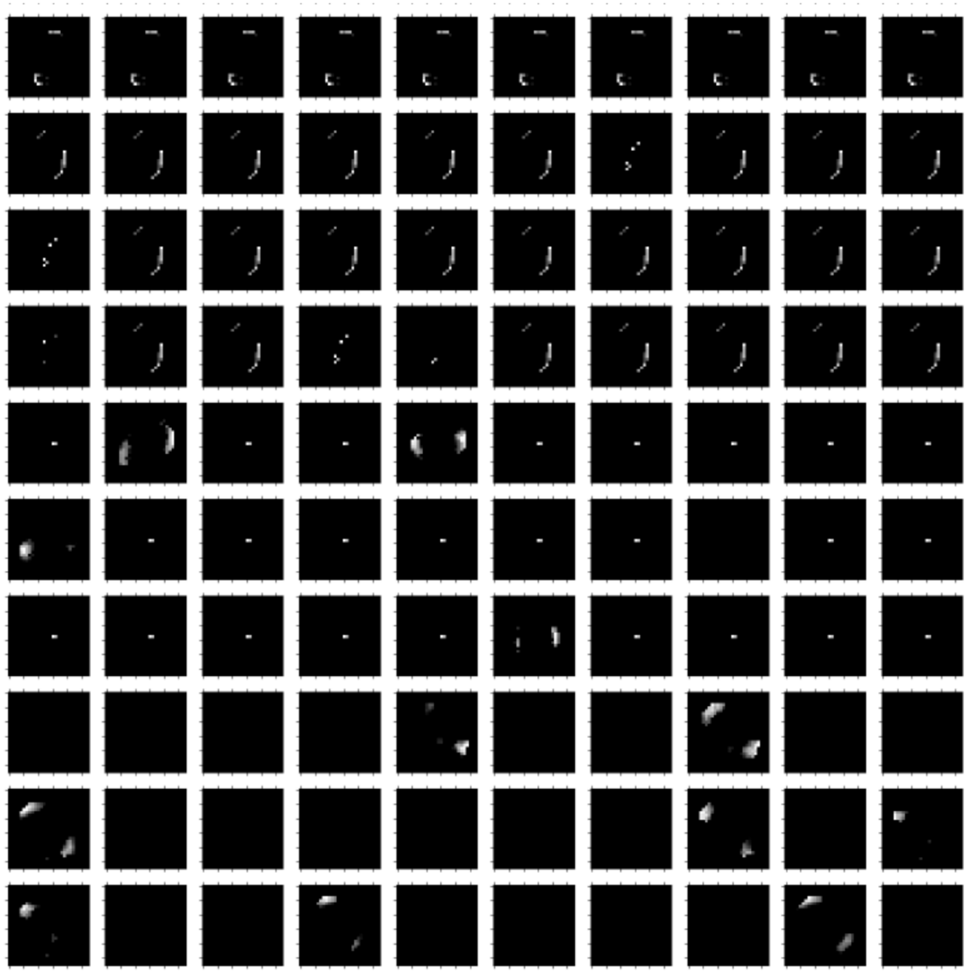
MCUs



HCUs

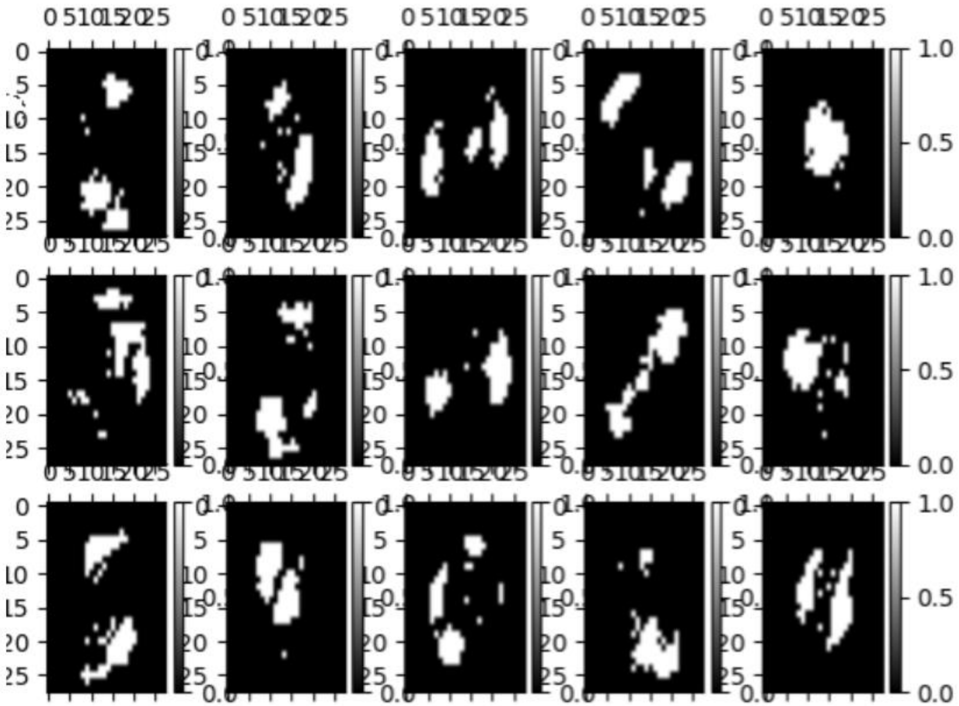
Results: FPGAs and Numerical Representation

(Part 2 of 4)



MCUs

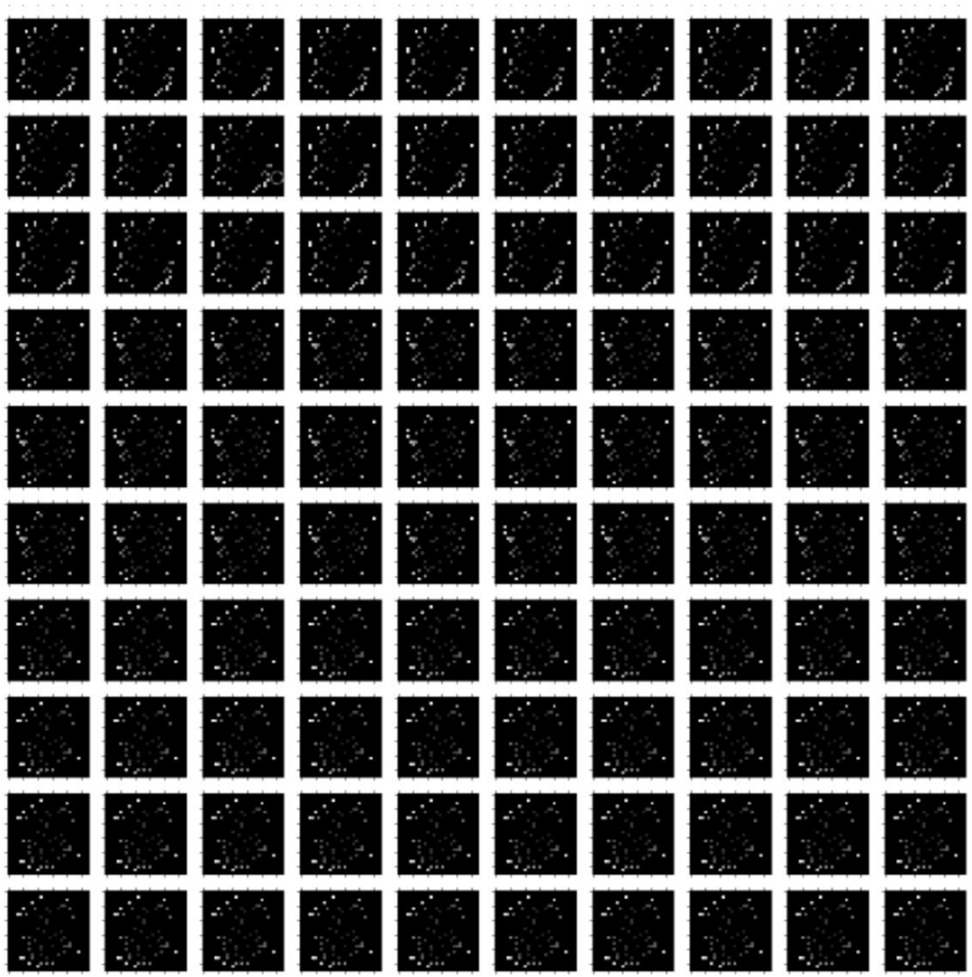
”BrainFloat 15” (15-bit)



HCUs

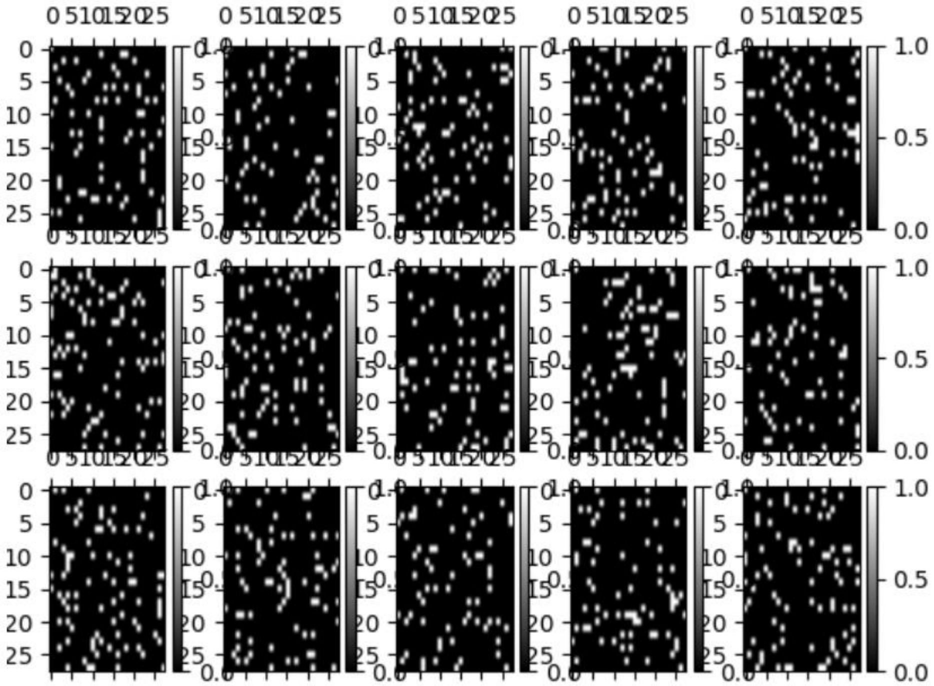
Results: FPGAs and Numerical Representation

(Part 2 of 4)



MCUs

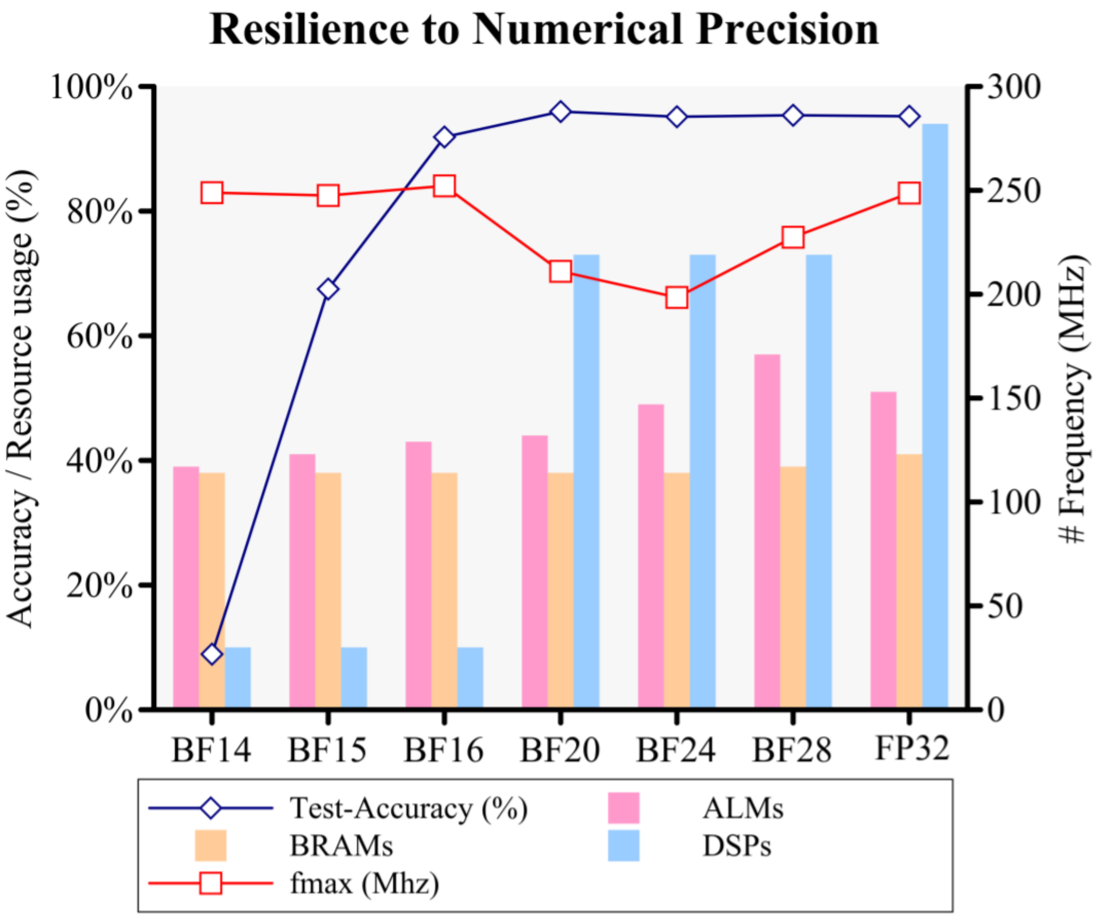
”BrainFloat 14” (14-bit)



HCUs

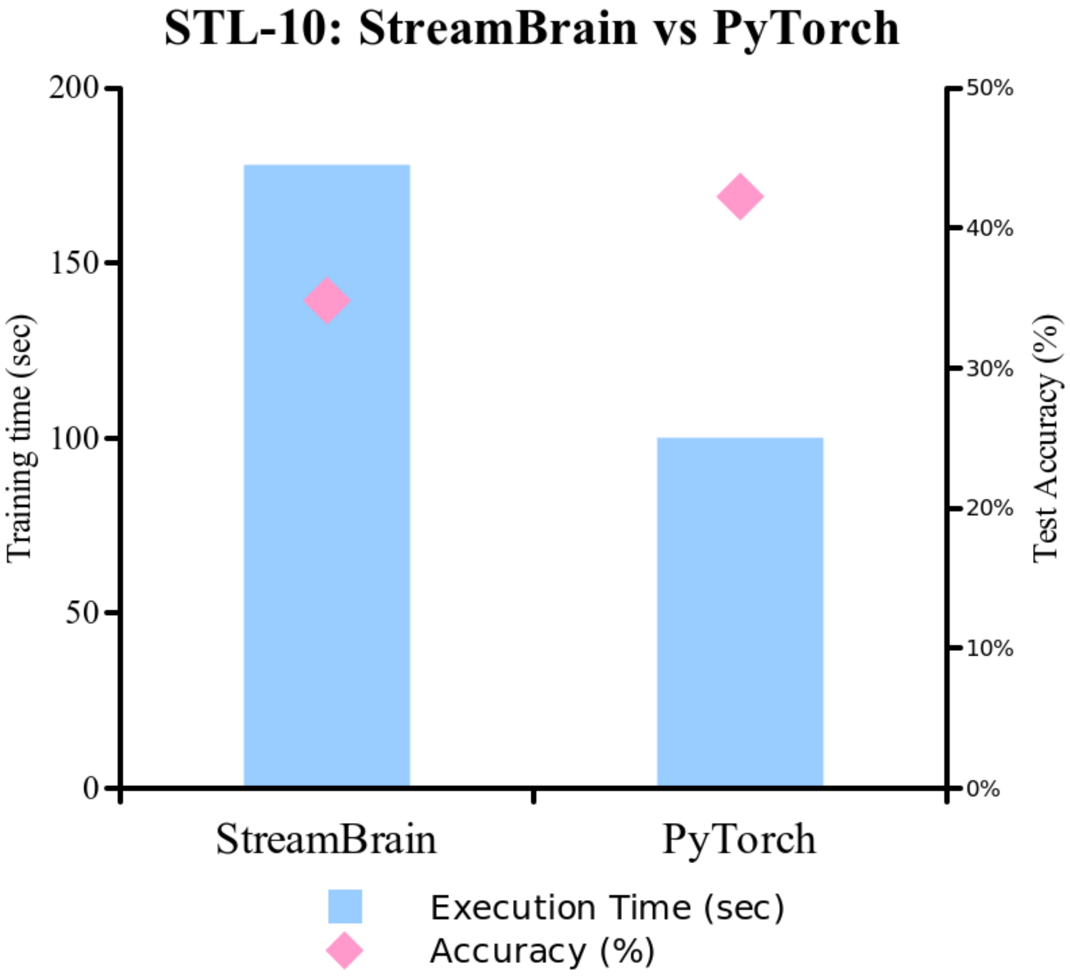
Results: FPGAs and Numerical Representation
(Part 2 of 4)

- Can we go with a lower/smaller numerical representation?
- Yes, we can:
  - All the way to BF16 (small degradation)
  - BF15 degrades test accuracy to ~65%
  - BF14 degrades it to chance (~10%)
- FPGA accelerator characteristics
  - 200+ MHz frequency
  - Smaller representation ⬄ Less area used
- BCPNN capable of training with next-gen hardware that supports reduced precision



Results: Performance on STL-10
(Part 3 of 4)

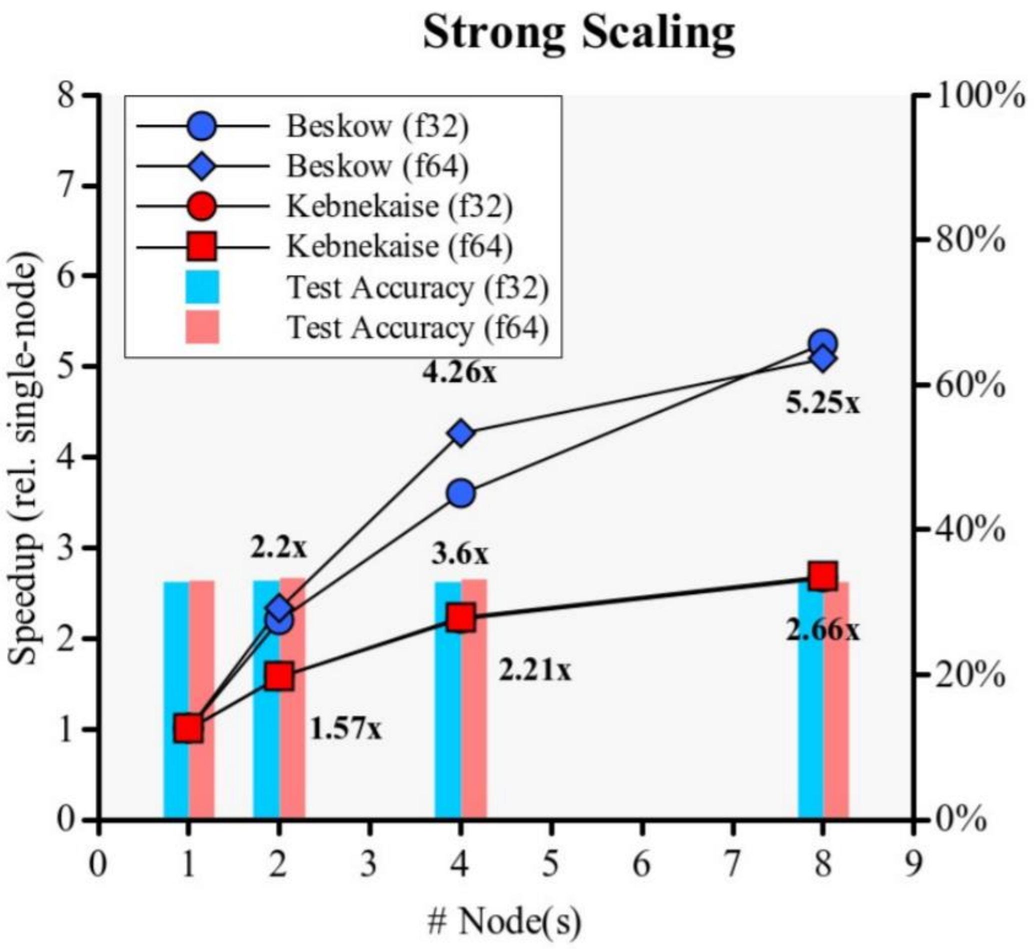
- First time STL-10 tested with BCPNN
- Execution on Nvidia A100
  - Compared to PyTorch using GPU with similar MLP capacity network
- StreamBrain ~**1.7x** slower than PyTorch
- Test accuracy of BCPNN slightly worse than MLP (PyTorch)
  - Requires further investigation
  - Still far from Convolutional Neural networks (90+%)



Results: Strong scaling STL-10

(Part 4 of 4)

- Early results on the MPI-backend
- Strong-scaling result on the two HPC machines
- Beskow computer scales up to 8 nodes (**5.25x** speedup)
- Kebnekaise performs sub-optimally
  - Needs further investigation/scrutiny



Future Work

- BCPNN with alternative number representations (or mixed!)
  - UNUMs, Posits, Elias, etc.
- New types of layers and encodings to better capture, e.g., color images
- Exploiting Sparsity
  - BCPNN ample opportunity to reduce cost of data movement and computation
- Extreme scalability
  - Towards synchronous-free execution at the exascale

Conclusion

- BCPNN an alternative to “traditional” Deep-Learning
  - 95+% on MNIST; 97.5+% on mixed BCPNN+SGD
- StreamBrain a framework that allows high-performance exploration of BCPNN
  - CPU, GPU, and FPGA backend
  - Training and testing on larger input sets than before (STL-10 within minutes!)
  - Trains MNIST within ~10 seconds
- StreamBrain used for architectural studies through FPGA
  - BF16-capable without much performance degradation.
  - How about Posits and other more exotic representations?
- Check out: **https://github.com/KTH-HPC/StreamBrain**
- **Do you have a use-case that might benefit from StreamBrain / BCPNN?**
  - **Contact podobas@kth.se**

Thank you!

EPiGRAM HS

KTH epcc ETH zürich

CRAY THE SUPERCOMPUTER COMPANY Fraunhofer ECMWF

epigram-hs.eu

European Commission

Horizon 2020
European Union funding
for Research & Innovation