GPU-RANC

A CUDA-Accelerated Simulation Framework for Neuromorphic Architectures

Sahil Hassan, Michael Inouye, Miguel Gonzalez, Ilkin Aliyev, **Joshua Mack**, Maisha Hafiz, Ali Akoglu

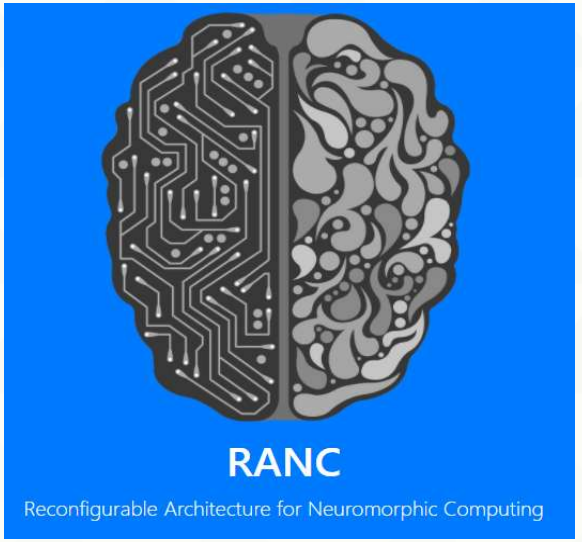
{sahilhassan, mikesinouye, migor2d2, ilkina, **jmack2545**, mhafiz1, akoglu}@arizona.edu

COLLEGE OF ENGINEERING
Electrical & Computer Engineering

Reconfigurable Computing Lab

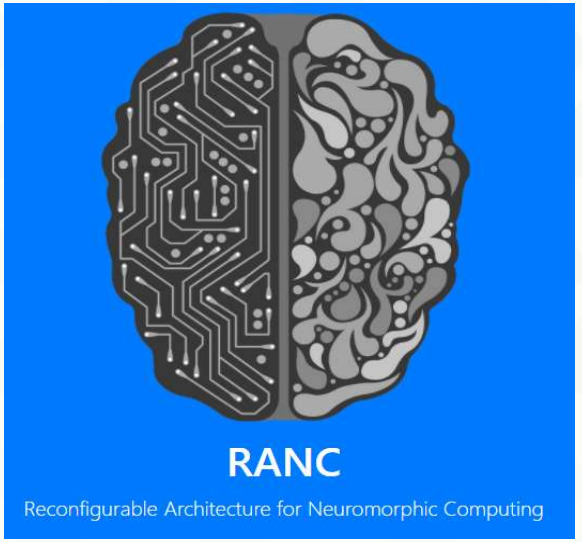
Motivation

- Neuromorphic architectures are at a very early stage
  - The foundational principles are in place, but we believe there is a lot of room to explore and guide future architectures
  - Outside of machine learning algorithms, promising applications have emerged in several other "conventional" areas
  - Open-source, academic simulation & emulation frameworks are needed to enable algorithmic/architectural co-design within the research community
- We have been developing an environment that seeks to meet these needs
  - RANC: A Reconfigurable Architecture for Neuromorphic Computing
  - Software & hardware ecosystem for designing and deploying algorithms to neuromorphic hardware



RANC: A Reconfigurable Architecture for Neuromorphic Computing

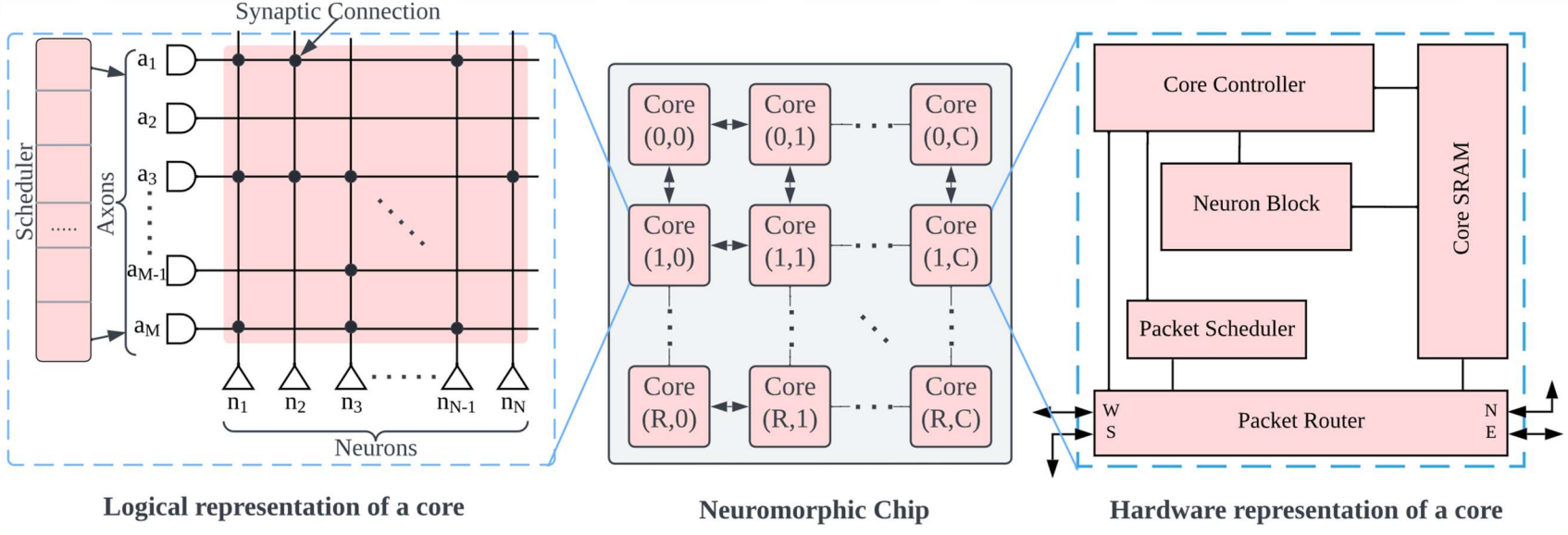
- Composed of three core components:
  - Tensorflow SNN frontend
  - C++-based CPU/GPU software simulator
  - FPGA-based emulation platform
- Allows for full hardware/software architectural customization across both simulation and emulation
- Having full architectural flexibility is important
  - We have previously demonstrated architecture-specific optimizations to improve mapping efficiency of
    - Vector Matrix Multiplication (~50% neuron reduction)
    - Convolutional layer mapping (~98.9% neuron reduction)
    - LDPC decoding (~44% neuron reduction)
- Recent users have expanded it for use as a generally programmable accelerator [1]



[1] Judicael Clair, Guy Eichler, and Luca P. Carloni. 2023. SpikeHard: Efficiency-Driven Neuromorphic Hardware for Heterogeneous Systems-on-Chip. ACM Trans. Embed. Comput. Syst. 22, 5s, Article 106 (October 2023), 22 pages. https://doi.org/10.1145/3609101

RANC: A Reconfigurable Architecture for Neuromorphic Computing

- Both the C++ simulator & FPGA emulator model the same underlying architecture
  - Enables "TrueNorth-Equivalent" behavior when mapping certain classes of applications to serve as a strong baseline
  - Highly parameterized architecture allows rapid specialization for other application scenarios



Logical representation of a core | Neuromorphic Chip | Hardware representation of a core

RANC Simulator

- Flexible simulation environment that requires as input
  - A global configuration file that defines NoC dimensions, global architectural parameters, debug logging
  - An application-specific configuration file that defines the contents of each core SRAM, synaptic connection crossbars, and other per-core configuration settings
- Simulation units measured in "ticks"
- Goal in this work: exploit the large opportunities for SIMD parallelism present in this simulation flow

Algorithm 1: RANC Simulator Algorithm

```
1  Initial setup and input decode
2  for tick in num_ticks do
3      for core in num_cores do
4          clear old contents of scheduler SRAM
5          shift in current tick input to scheduler SRAM
6      for packet in num_packets[tick] do
7          calculate destination / offset
8          route packet
9          write into SRAM
10     for core in num_cores do
11         for neuron in num_neurons do
12             for axon in num_axons do
13                 accumulate neuron potential
14             additional computation / checks
15     for core in num_cores do
16         for neuron in num_neurons do
17             if neuron→ spike == 1 then
18                 calculate destination / offset
19                 route packet
20                 write into SRAM
```

GPU-RANC

- Per-component profiling costs
  - **Scheduler: 2.2%**
  - **Router: 5.8%**
  - **Neuron Block: 91.7%**
- Neuron block is the key kernel to accelerate, but the others are unrolled and replicated to keep the neuron block instances fed
- Neuron block parallelization
  - Launch all neuron computations within each core in parallel (lines 11-14)
  - Expand to grid-level parallelism across all cores (line 10)
  - Reduction-tree accumulation of final neuron potentials (line 13)

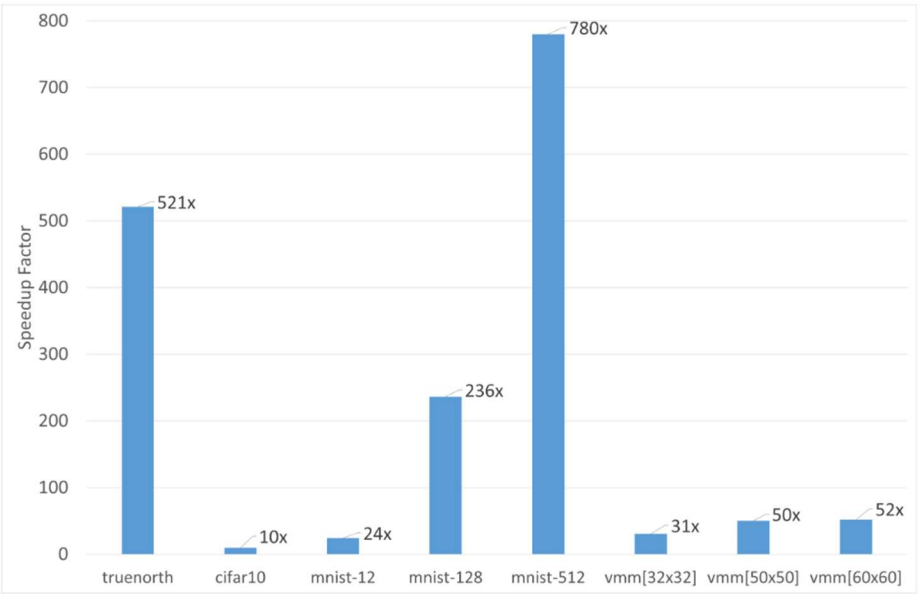
Algorithm 1: RANC Simulator Algorithm

```
1  Initial setup and input decode
2  for tick in num_ticks do
3      for core in num_cores do
4          clear old contents of scheduler SRAM
5          shift in current tick input to scheduler SRAM
6      for packet in num_packets[tick] do
7          calculate destination / offset
8          route packet
9          write into SRAM
10     for core in num_cores do
11         for neuron in num_neurons do
12             for axon in num_axons do
13                 accumulate neuron potential
14             additional computation / checks
15     for core in num_cores do
16         for neuron in num_neurons do
17             if neuron→ spike == 1 then
18                 calculate destination / offset
19                 route packet
20                 write into SRAM
```

GPU-RANC: Results

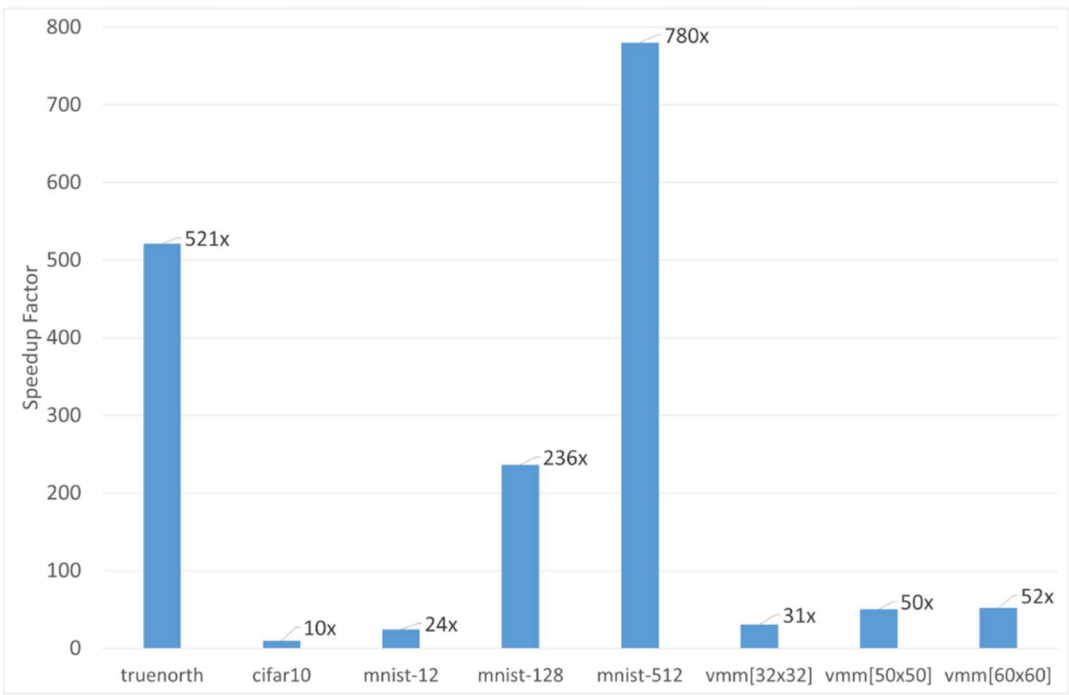
- Applications:
  - MNIST with 12, 128, and 512 core configurations
  - VMMs of 32x32, 50x50, and 60x60
  - CIFAR10
- TrueNorth Ref: an idle TN-equivalent architecture
- Hardware configuration:
  - AMD EPYC 7552 48-core processor @ 2.9GHz
  - Nvidia Tesla V100S 32GB
- Longest application (MNIST-512c) went from 5.6 hours on CPU to 26 seconds on GPU (780x speedup)

Application	Ticks	# of Cores	Axons /Core	Neurons /Core	CPU exec. time (s)
MNIST-12c	10010	12	256	256	539
MNIST-128c	10010	128	256	256	5230
MNIST-512c	10010	512	256	256	20344
VMM 32x32	788	21	256	256	65
VMM 50x50	889	45	512	256	311
VMM 60x60	1095	51	512	256	434
CIFAR10	10010	36	256	256	1693
TrueNorth Ref.	500	4096	256	256	8295



GPU-RANC: Results

- Why are certain configurations faster than others?
  - Shorter-lived applications have less time to amortize GPU overheads
  - "TrueNorth" reference application has essentially no data transfer overheads, but experiences limited serialization due to the high number of threads spawned (134 million)
  - MNIST-512 requires both a large number of cores (512) and exploits a large degree of data parallelism
- Kernel-level analysis left to the paper



Conclusions & future work

Conclusions

- The grid-level parallelism available here enable simulation of much larger kernels than were previously possible
- Enables new degrees of rapid software-based architecture exploration & convergence

Future work

- Explore whether we can integrate any architectural lessons from architectures like Loihi 2
  - Programmable neuron ISAs, graded spikes
- Investigate building compatibility with other SNN training methodologies & software frameworks (PyTorch frontends, etc)
- Continue exploring the applicability of neuromorphic architectures to additional classical computing applications

Thank you!

Contact:
{sahilhassan, jmack2545, akoglu}@arizona.edu