ActiveN: A Scalable and Flexibly-programmable Event-driven Neuromorphic Processor

Xiaoyi Liu[1], Zhongzhu Pu[1], Peng Qu[1], Weimin Zheng[1], Youhui Zhang[1,2]

[1] Beijing National Research Center for Information Science and Technology, Tsinghua University, China

[2] Zhongguancun Laboratory, China

Overview & Motivation

Energy efficiency of neuromorphic computing comes from its event-driven nature.

Overview & Motivation

Energy efficiency of neuromorphic computing comes from its event-driven nature.

- $\Rightarrow$ Sparsity in both spatial and temporal domain

Overview & Motivation

Energy efficiency of neuromorphic computing comes from its event-driven nature.

- $\Rightarrow$ Sparsity in both spatial and temporal domain

Trend 1: We're having more diverse neuron models (update model, postsynaptic accumulation model).

- LIF and its variants, Izhikevich, HH, Biexponential

Overview & Motivation

Energy efficiency of neuromorphic computing comes from its event-driven nature.

- $\Rightarrow$ Sparsity in both spatial and temporal domain

Trend 1: We're having more diverse neuron models (update model, postsynaptic accumulation model).

- LIF and its variants, Izhikevich, HH, Biexponential

Conflict: Contemporary general-purpose processor architectures cannot efficiently deals with this sparsity.

Overview & Motivation

Energy efficiency of neuromorphic computing comes from its event-driven nature.

- $\Rightarrow$ Sparsity in both spatial and temporal domain

Trend 2: We're having larger models.

- Schmidt et al(2018): 4.13M neurons, 24.2B synapses

Overview & Motivation

Energy efficiency of neuromorphic computing comes from its event-driven nature.

- $\Rightarrow$ Sparsity in both spatial and temporal domain

Trend 2: We're having larger models.

- Schmidt et al(2018): 4.13M neurons, 24.2B synapses

Conflict: High density storages (e.g. DRAMs) also cannot efficiently deals with sparsity.

Overview & Motivation

Energy efficiency of neuromorphic computing comes from its event-driven nature.

- $\Rightarrow$ Sparsity in both spatial and temporal domain

Is it possible to introduce architectural / ISA extensions onto a general purpose architecture, to efficiently and performantly execute neuromorphic computing payloads?

Overview & Motivation

Energy efficiency of neuromorphic computing comes from its event-driven nature.

- $\Rightarrow$ Sparsity in both spatial and temporal domain

Is it possible to introduce architectural / ISA extensions onto a general purpose architecture, to efficiently and performantly execute neuromorphic computing payloads?

YES

Overview & Motivation

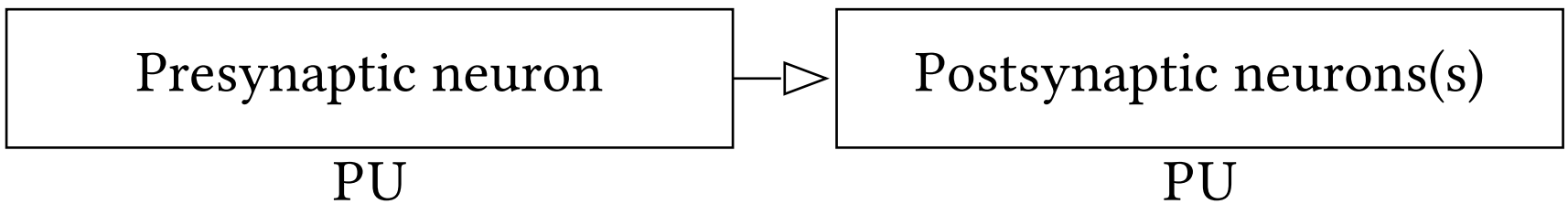
Energy efficiency of neuromorphic computing comes from its event-driven nature.

- $\Rightarrow$ Sparsity in both spatial and temporal domain

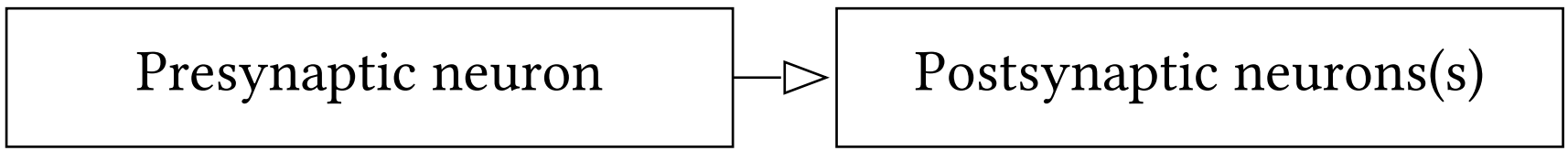
Is it possible to introduce architectural / ISA extensions onto a general purpose architecture, to efficiently and performantly execute neuromorphic computing payloads?

YES *Terms and conditions may apply

Categorizing memory accesses

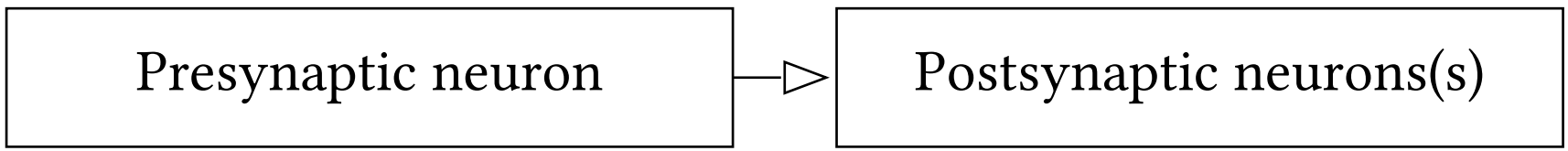


Categorizing memory accesses



Data	Size	Sparse Access?	Lifetime
Neuron state	$\mathcal{O}(n)$	Y/N	Persistent
Synapse data	$\mathcal{O}(n^2)$	Y	Persistent
Spikes	$\mathcal{O}(n^2)$	Y	Temporary
Neuron inputs	$\mathcal{O}(n)$	Y	Persistent

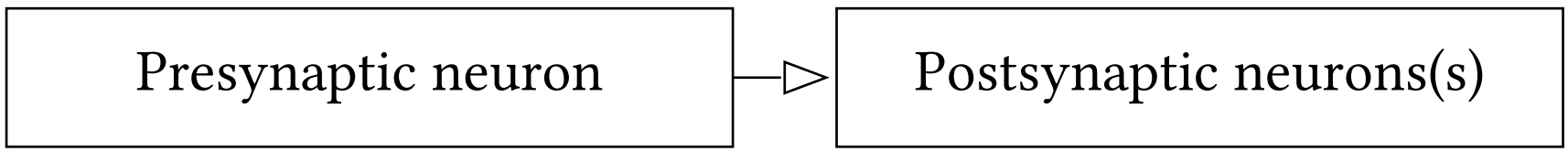
Categorizing memory accesses



Data	Size	Sparse Access?	Lifetime
Neuron state	$\mathcal{O}(n)$	Y/N	Persistent
Synapse data	$\mathcal{O}(n^2)$	Y	Persistent
Spikes	$\mathcal{O}(n^2)$	Y	Temporary
Neuron inputs	$\mathcal{O}(n)$	Y	Persistent

- Make synapse data accesses dense
- Eliminate spike data storage
- Move neuron states / inputs into scratchpads

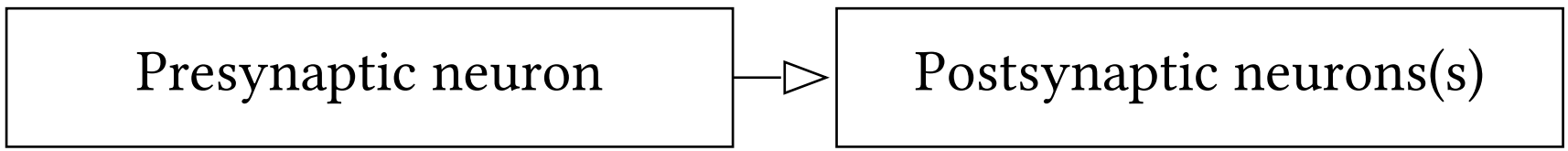
Categorizing memory accesses



Data	Size	Sparse Access?	Lifetime
Neuron state	$\mathcal{O}(n)$	Y/N	Persistent
Synapse data	$\mathcal{O}(n^2)$	Y	Persistent
Spikes	$\mathcal{O}(n^2)$	Y	Temporary
Neuron inputs	$\mathcal{O}(n)$	Y	Persistent

- Make synapse data accesses dense ⇒ Store as CSR-format
- Eliminate spike data storage
- Move neuron states / inputs into scratchpads

Categorizing memory accesses

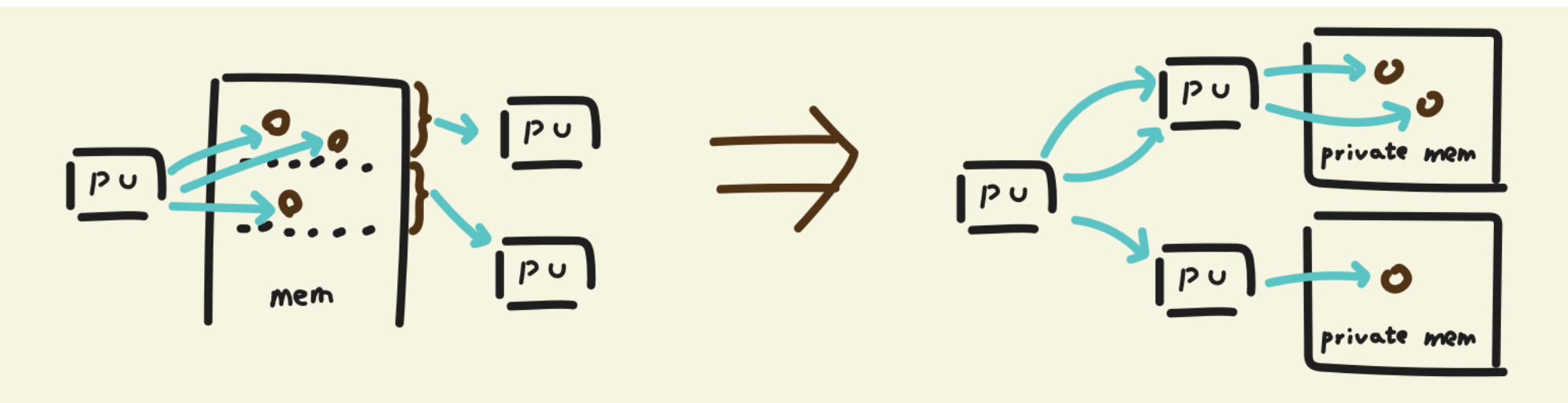


Data	Size	Sparse Access?	Lifetime
Neuron state	$\mathcal{O}(n)$	Y/N	Persistent
Synapse data	$\mathcal{O}(n^2)$	Y	Persistent
Spikes	$\mathcal{O}(n^2)$	Y	Temporary
Neuron inputs	$\mathcal{O}(n)$	Y	Persistent

- Make synapse data accesses dense ⇒ Store as CSR-format
- Eliminate spike data storage ⇒ How to consume timely?
- Move neuron states / inputs into scratchpads ⇒ How to write remotely?

Active messaging

PUs can send messages to PUs. Incoming events would trigger event handlers, which is scheduled according to priorities by the hardware.



Access latencies for global memory (synapse matrix)

Simple optimization: put CSR row pointers into scratchpad: saves one roundtrip.

Access latencies for global memory (synapse matrix)

Simple optimization: put CSR row pointers into scratchpad: saves one roundtrip.

Long latency accesses still unavoidable: requires blocking / MSHRs & registers.

“Context-free” memory accesses

Access to synapse connection matrix is “Context-free”

```
const neuronId;
update(state[neuronId], input[t][neuronId]);
const neighboors = await read(synapses[neuronId]);
for(const [neighId, weight, delay] of neighboors)
  input[t+delay][neigh] += weight;
```

“Context-free” memory accesses

Access to synapse connection matrix is “Context-free”

```
// const neuronId;
// update(state[neuronId], input[t][neuronId]);
const neighboors; // = await read(synapses[neuronId]);
for(const [neighId, weight, delay] of neighboors)
  input[t+delay][neigh] += weight;
```

“Context-free” memory accesses

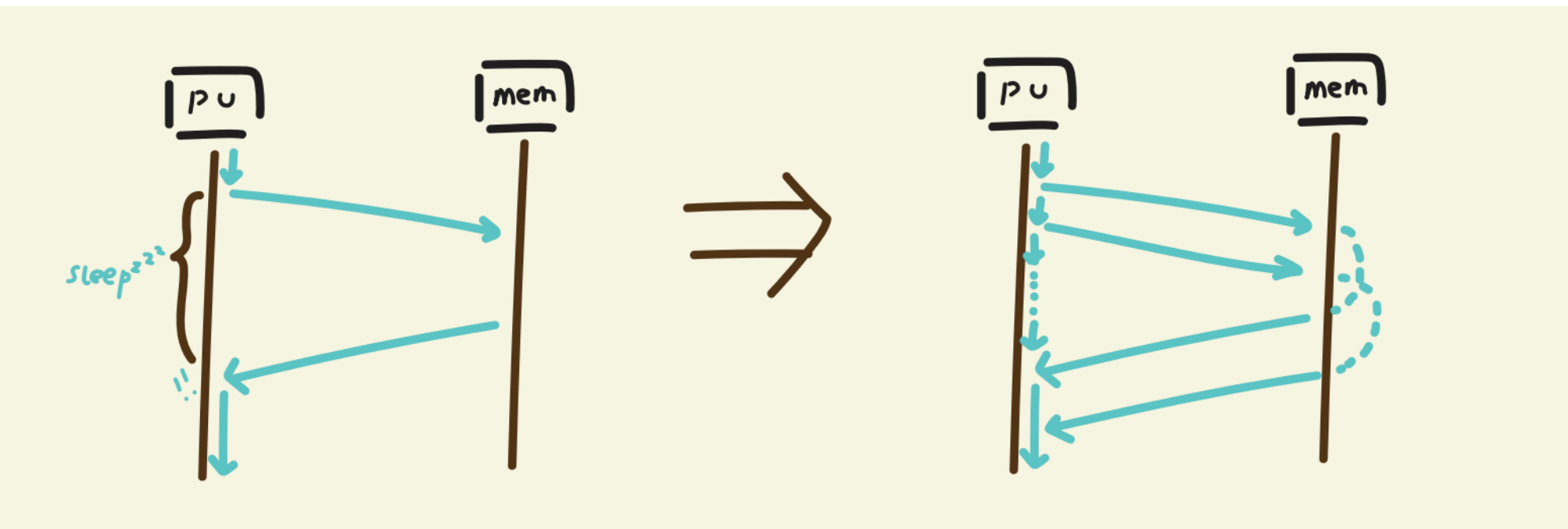
Access to synapse connection matrix is “Context-free”

```
// const neuronId;
// update(state[neuronId], input[t][neuronId]);
const neighboors; // = await read(synapses[neuronId]);
for(const [neighId, weight, delay] of neighboors)
  input[t+delay][neigh] += weight;
```

“Fire-and-forget” asynchronous memory accesses

Asynchronous memory access through Active Messaging

PUs can send messages indicative of a load / store to memory controller.



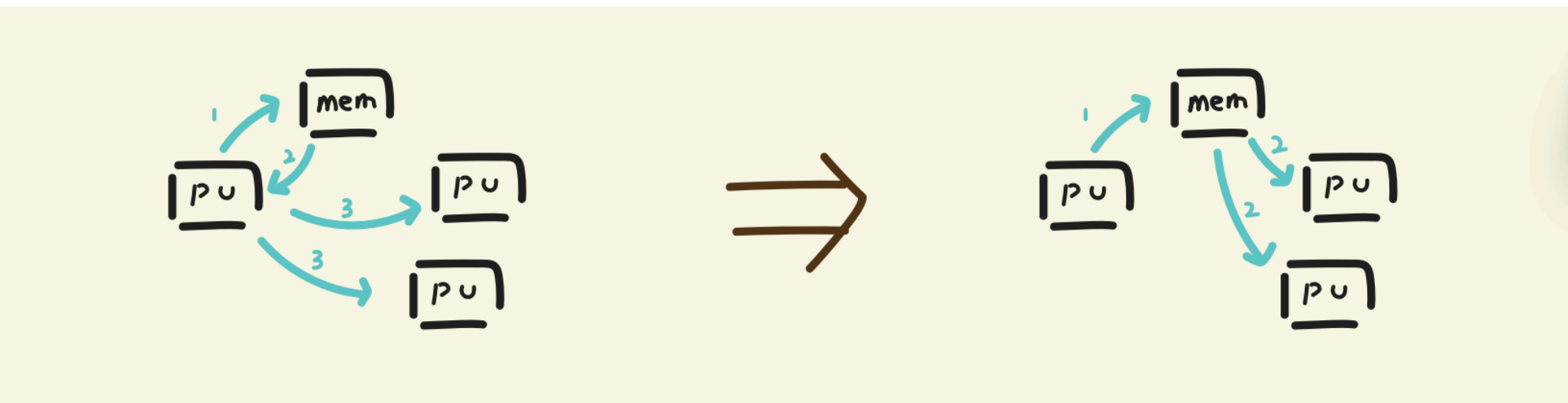
Asynchronous memory access through Active Messaging

PUs can send messages indicative of a load / store to memory controller.

- More concurrent accesses ⇒ Higher achievable bandwidth
- Saves on architectural “context tracking” structures: MSHRs, warp queue, physical registers…

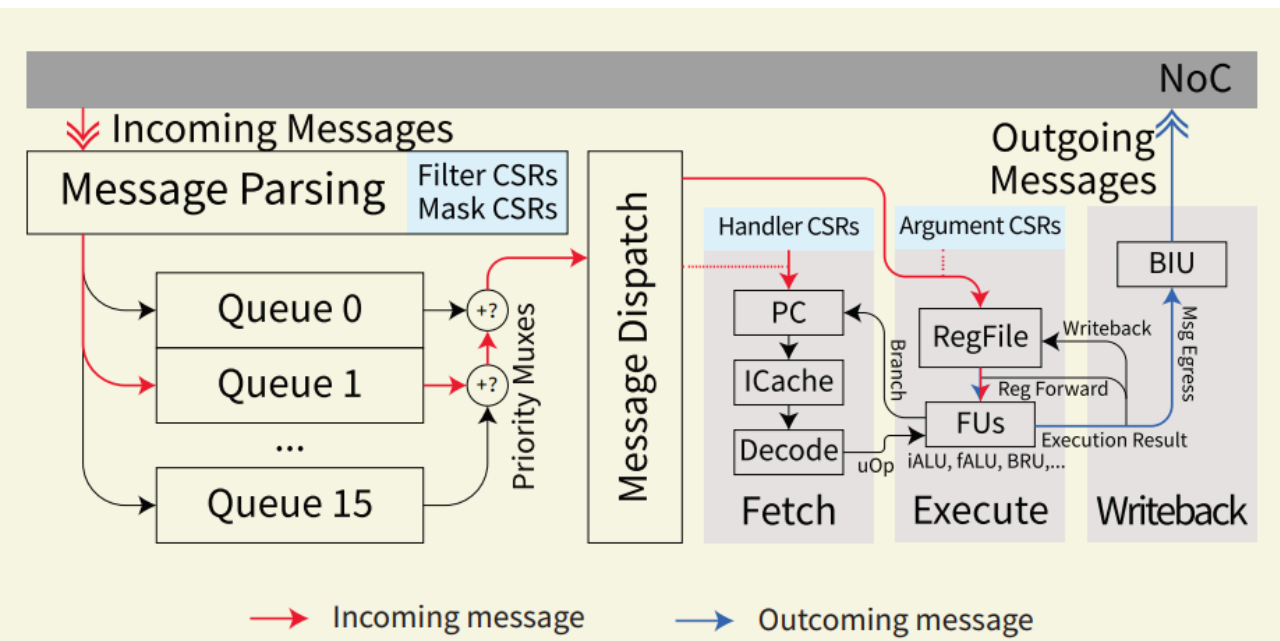
CSR-aware memory controller

Only for “context-free” synapse matrix loads: memory controller directly forwards the data to postsynaptic PU.

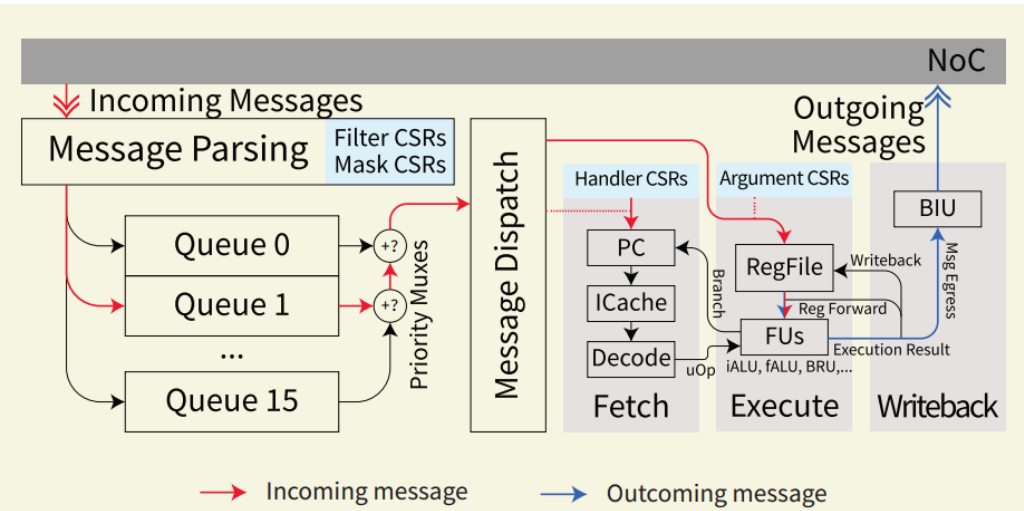


Implementation / Core and active messaging

Based on a in-order 3-stage RISC-V core.



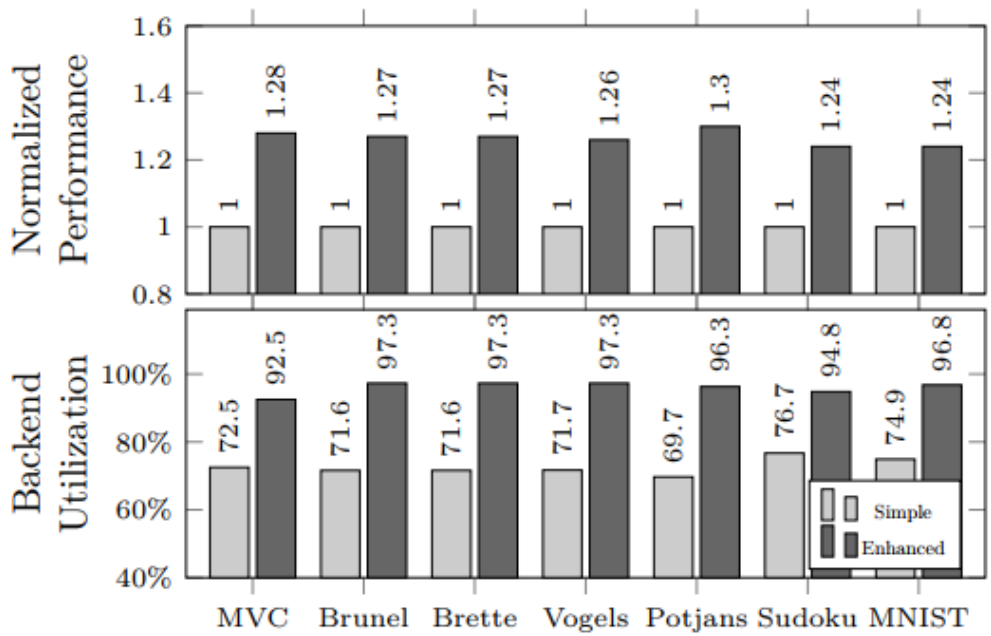
Implementation / Core and active messaging



- Base ISA RV32IF, 32-bit XLEN
- Arguments are directly written into register file.
- FPU 2-cycle delay, other FUs 1 cycle delay.

Implementation / SMT-like event parallelism

FPU and branches introduces bubbles into the pipeline. SMT-like parallelism introduces ~25% performance gain.



Implementation / SMT-like event parallelism

FPU and branches introduces bubbles into the pipeline. SMT-like parallelism introduces ~25% performance gain.

Hardware cost:

- Register file
- PC register
- **Issue width is not increased**

Implementation / PPA

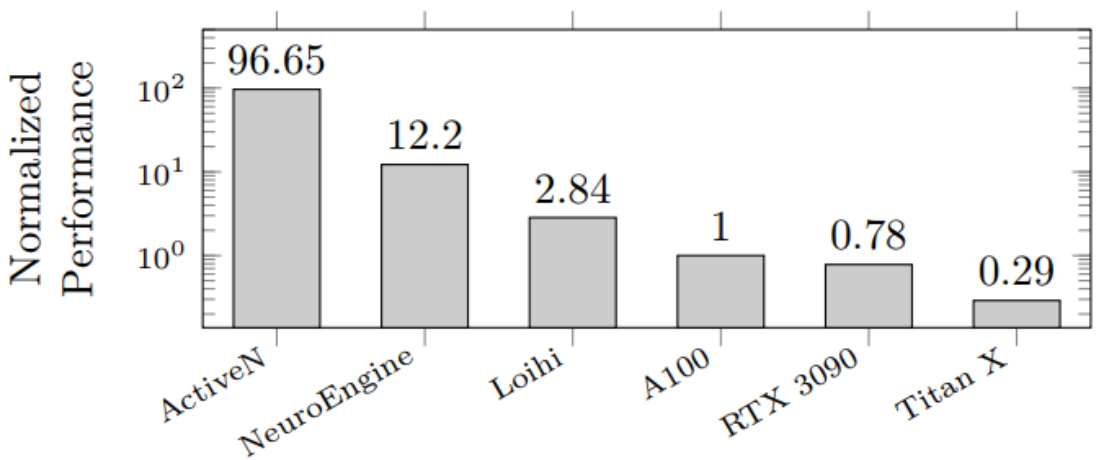
512 PU, interconnected with a two-layer ring bus.

Synthesised with 28nm process node. Frequency **1GHz**.

PU modules		Area (um^2)	Peak Power (mW)
Core	Baseline (fixed)	69315	44.31
	Baseline (floating)	85497	49.64
	Enhanced	92267	55.81
SRAM	ICache	5753	0.874
	Scratchpad	94187	4.012
NoC Router (per cluster)		127795	64.97
Entire System (512 PUs)		Area (mm^2)	Peak Power (W)
DRAM Controller		≈ 3	≈ 0.4
$\sum$	Baseline (fixed)	90.74	27.27
	Baseline (floating)	99.03	30.00
	Enhanced	102.50	33.16

Evaluation / Performance & Energy efficiency

Booksim + DRAMsim + Core RTL model online co-simulation

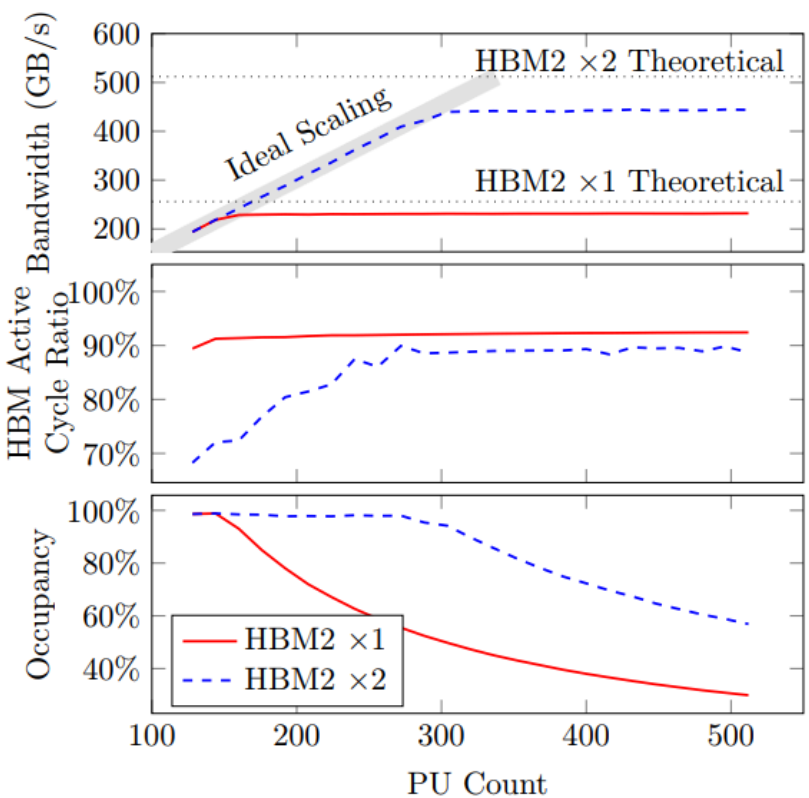


- ~96.6x end-to-end performance against A100, ~151.9x energy efficiency.
- Comparing with other neuromorphic processors, we don't have to store synapse data into on-die SRAM, meaning more computation power.

Evaluation / Scalability

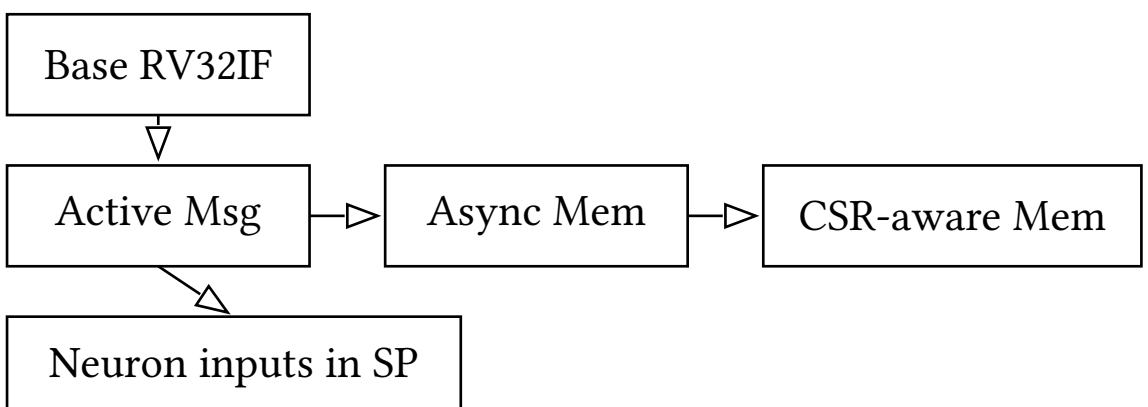
Same amount of neurons, scaling working PU count.

- Can saturate computation and (most) memory bandwidth, depending on which is bounding.
- Close to ideal scaling when computational-bounded.



Conclusion

ActiveN is a event-driven neuromorphic architecture, achieved by adding ISA extensions such as **active messanging, asynchronous memory access and CSR-aware memory controller** to a base RISC-V manycore architecture, which can be performant, efficient and flexible.



Thank you!

- GitHub: `CRAFT-THU/ActiveN`