Active-Routing: Compute on the Way for Near-Data Processing

Jiayi Huang, Ramprakash Reddy Puli, Pritam Majumder
Sungkeun Kim, Rahul Boyapati, Ki Hwan Yum and EJ Kim

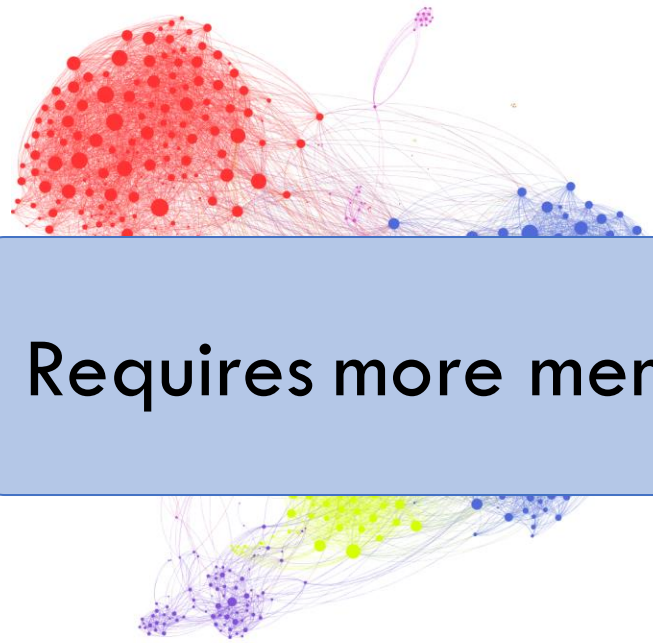
Outline

- Motivation
- Active-Routing Architecture
- Implementation
- Enhancements in Active-Routing
- Evaluation
- Conclusion

Motivation

Data Is Exploding

- Graph processing (social networks)



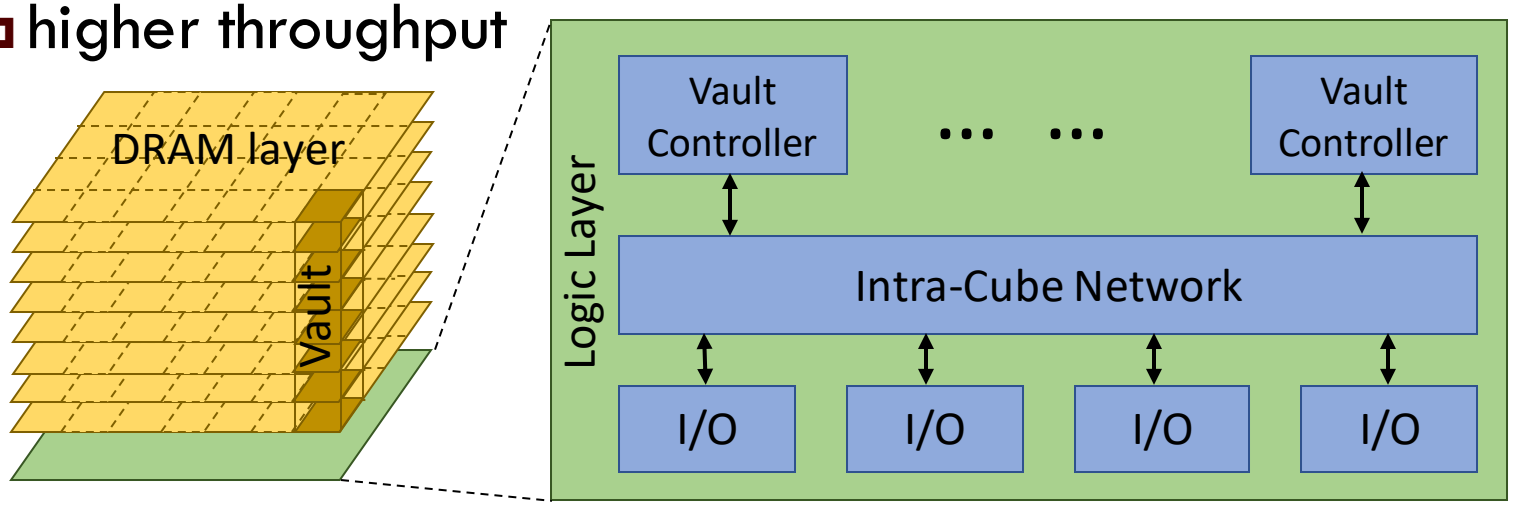
Requires more memory to process big data

- Deep learning (NLP) [Hestess et al. 2019]

Ref: https://griffsgraphs.files.wordpress.com/2012/07/facebook-network.png

Demand More Memory

- 3D die-stacked memory [Loh ISCA'08]
  - HMC and HBM
  - Denser capacity
  - higher throughput



- Memory network [Kim et al. PACT'13, Zhan et al. MICRO'16]
  - Scalable memory capacity
  - Better processor bandwidth

Enormous Data Movement Is Expensive

Active-Routing for dataflow execution in memory network

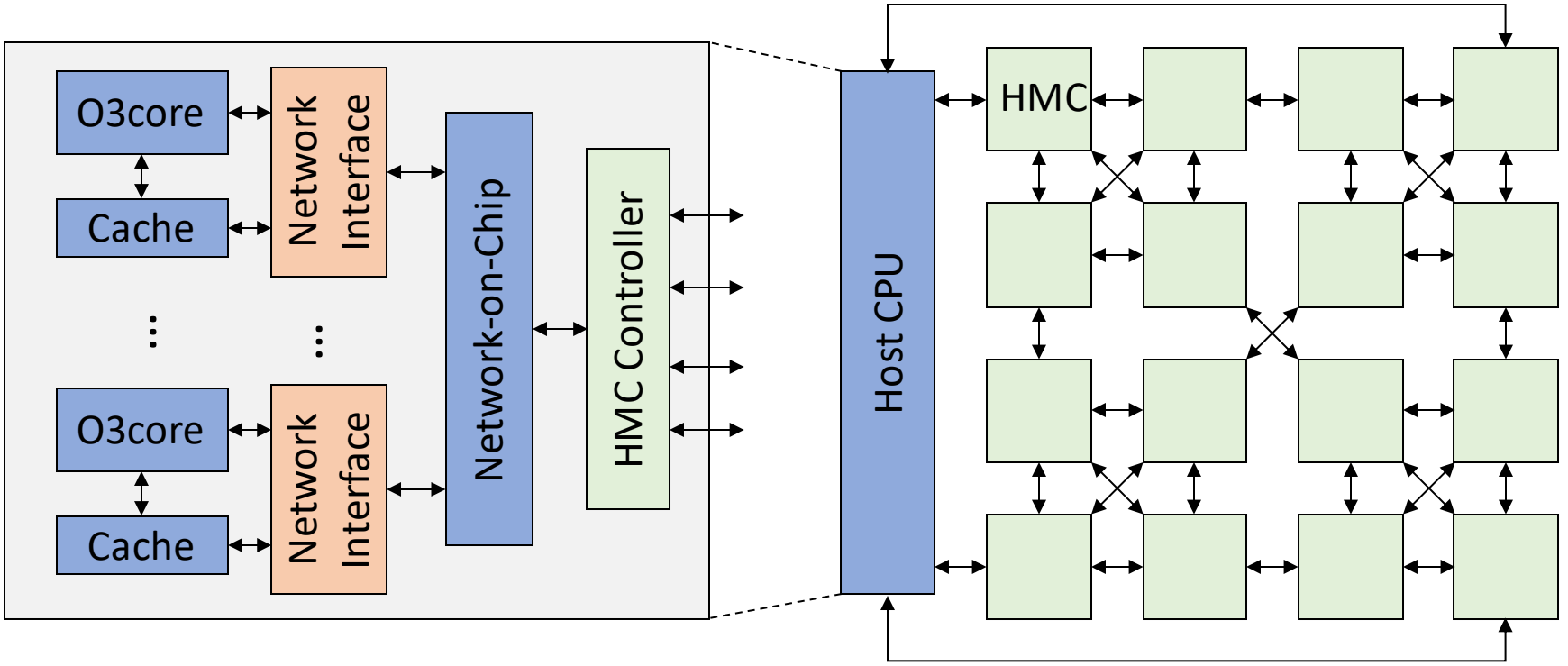
Reduce data movement and more flexible

Exploit **memory throughput** and **network concurrency**

- Processing-in-memory (PIM)
  - PIM-Enabled Instruction (PEI) [Ahn et al. ISCA'2015]
  - C = A x B (Can we bring less data?)
- In-network computing
  - Compute in router switches [Panda IPPS'95, Chen et al. SC'11]
  - MAERI [Kwon et al. ASPLOS'18]

Active-Routing Architecture

System Architecture



Active-Routing Flow

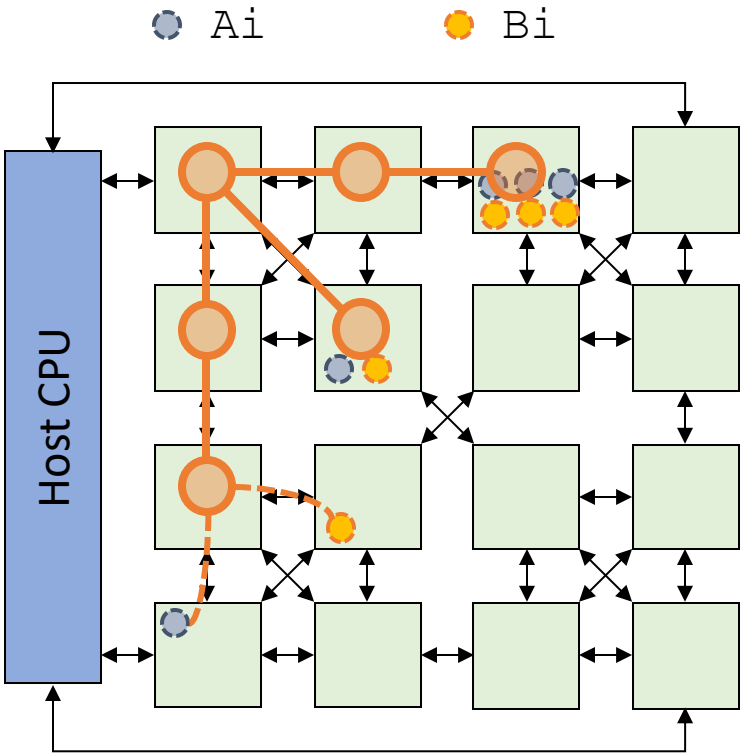
- Active-Routing tree dataflow for compute kernel

- Compute kernel example

```
for (i = 0; i < n; i++)
{
      sum += *Ai × *Bi;
}
```

  - Reduction over intermediate results

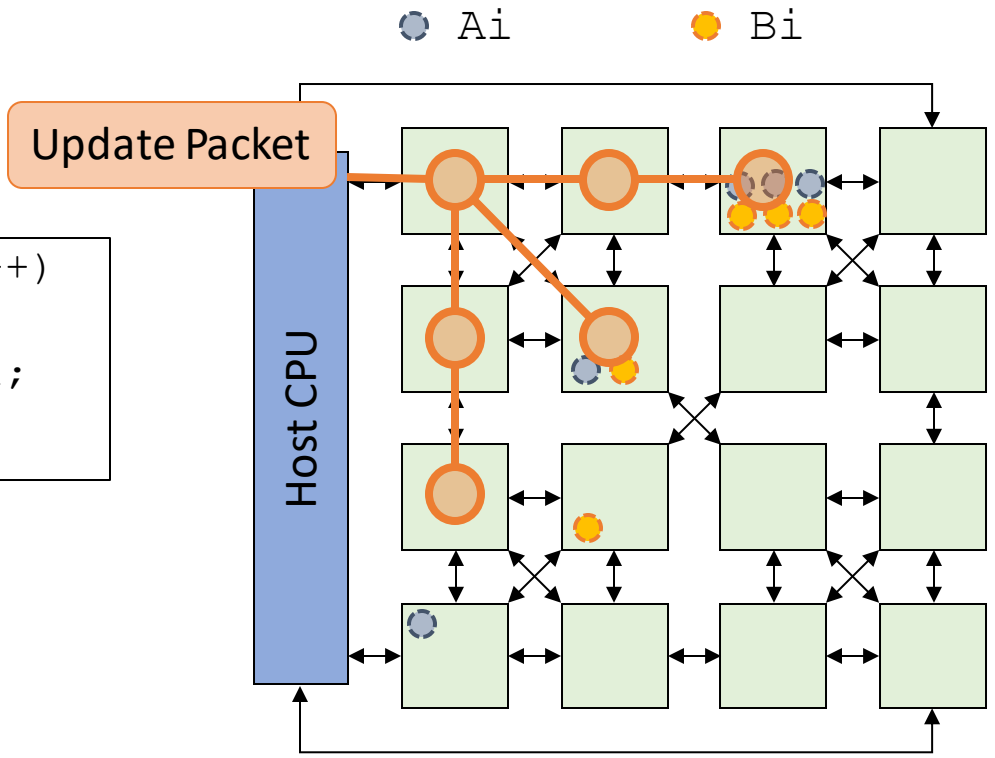
- Active-Routing tree dataflow



Active-Routing Three-Phase Processing

- Active-Routing Tree Construction

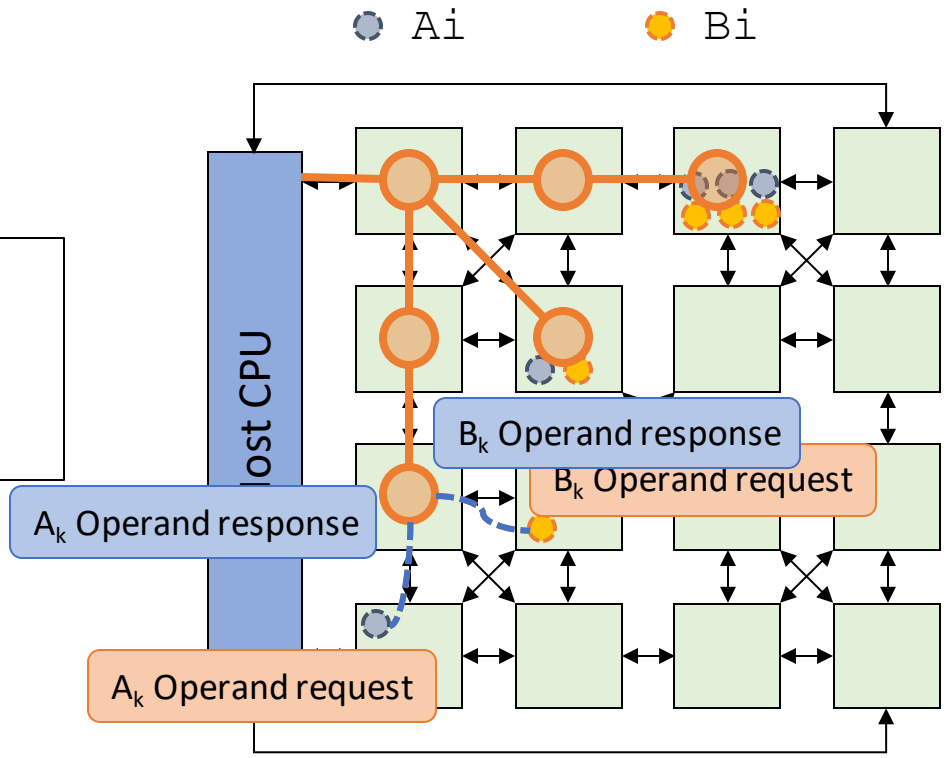
```
for (i = 0; i < n; i++)
{
      sum += *Ai × *Bi;
}
```



Active-Routing Three-Phase Processing

- Active-Routing Tree Construction
- Update Phase for data processing

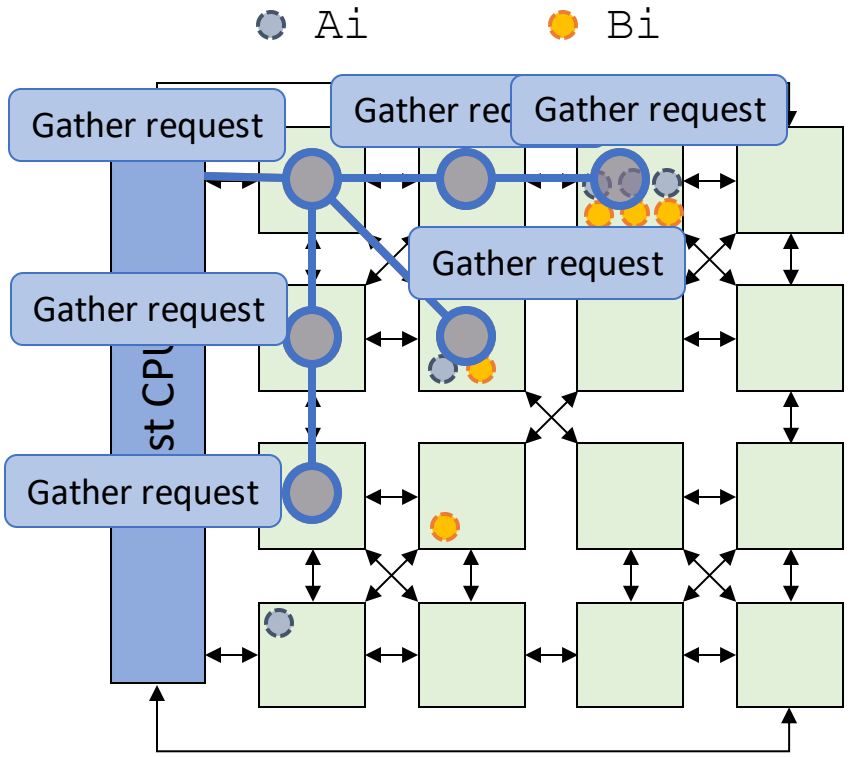
```
for (i = 0; i < n; i++)
{
      sum += *Ai × *Bi;
}
```



Active-Routing Three-Phase Processing

- Active-Routing Tree Construction
- Update Phase for data processing
- Gather Phase for tree reduction

```
for (i = 0; i < n; i++)
{
      sum += *Ai × *Bi;
}
```



Implementation

Programming interface and ISA extension

- `Update(void *src1, void *src2, void *target, int op);`
- `Gather(void *target, int num_threads);`

```
for (i = 0; i < n; i++)
{
      sum += *Ai × *Bi;
}
```

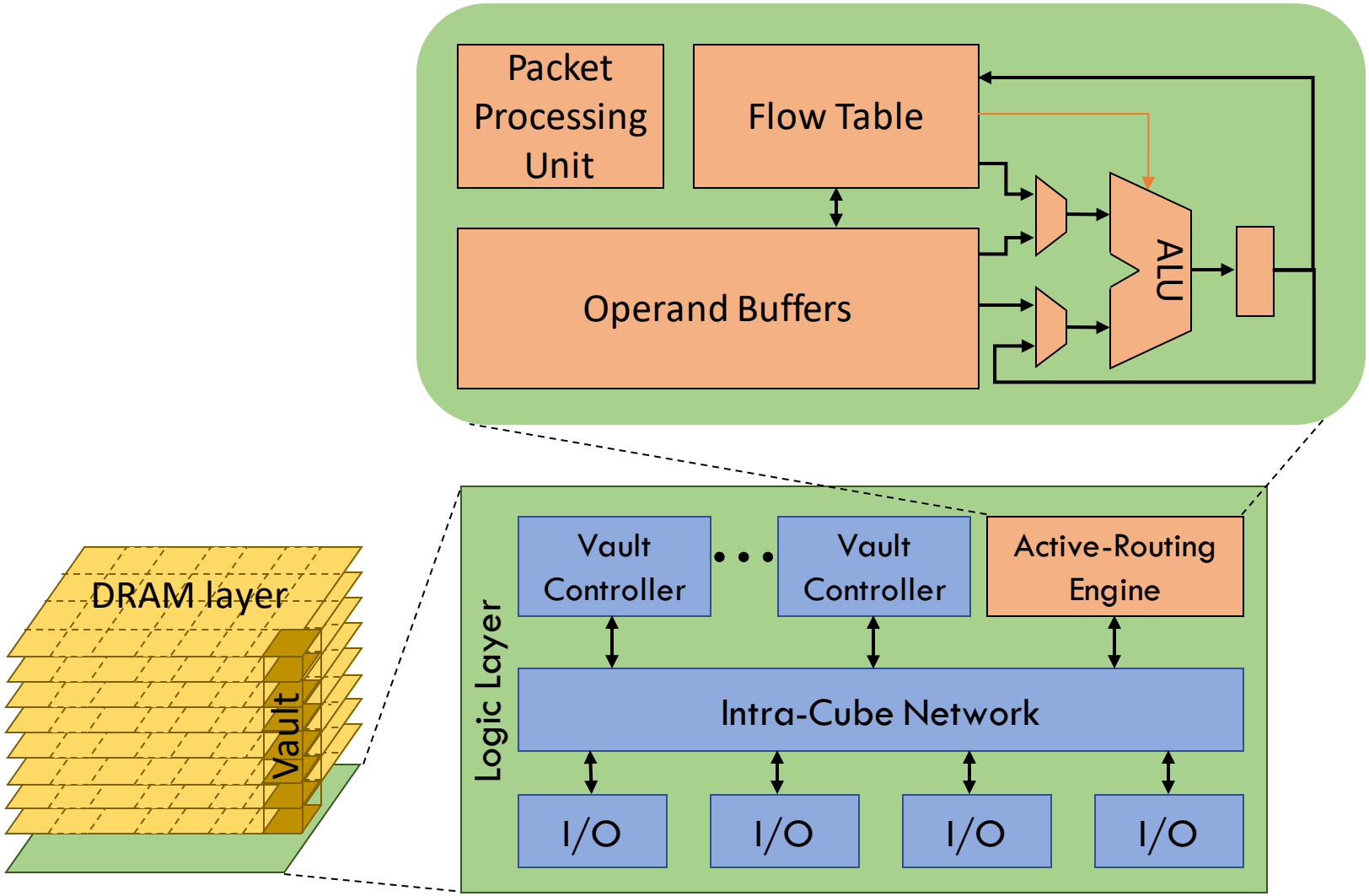
```
for (i = 0; i < n; i++)
{
      Update(Ai, Bi, &sum, MAC);
}
Gather(&sum, 16);
```

Programming interface and ISA extension

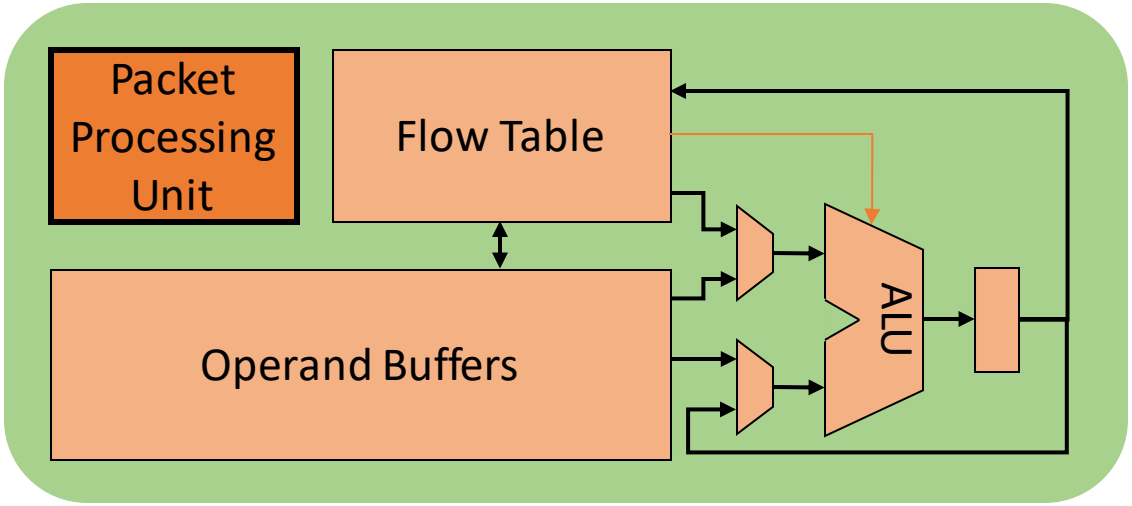
- `Update(void *src1, void *src2, void *target, int op);`
- `Gather(void *target, int num_threads);`

- Offloading logic in network interface
  - Dedicated registers for offloading information
  - Convert to Update/Gather packets

Active-Routing Engine

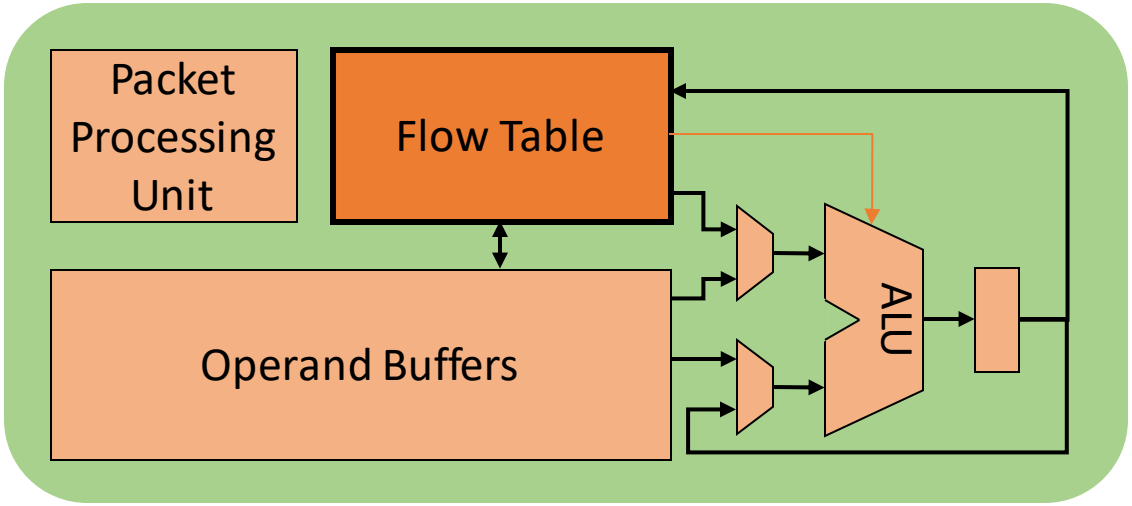


Packet Processing Unit



- Process Update/Gather packets
- Schedule corresponding actions

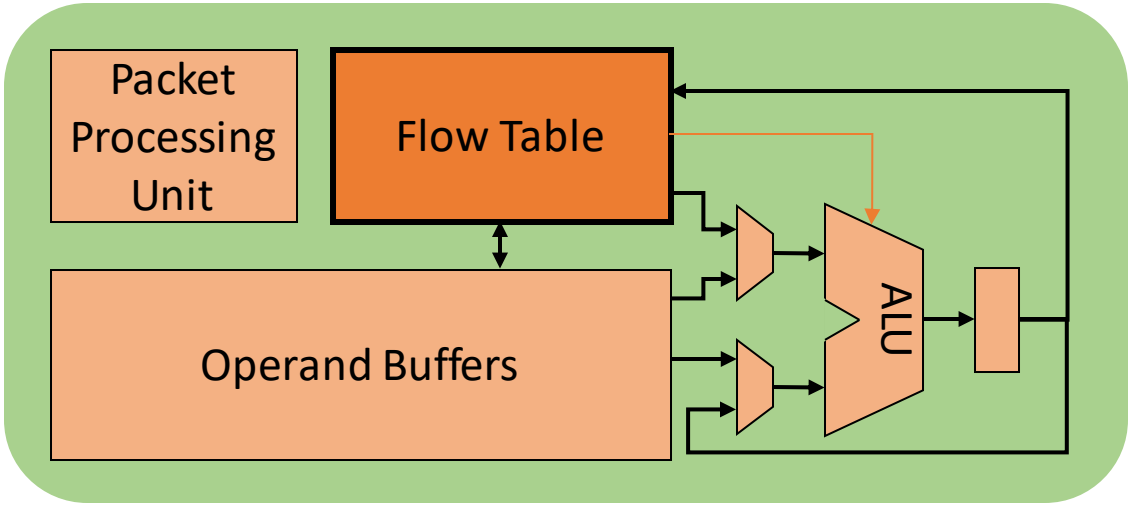
Flow Table



- Flow Table Entry

64-bit	6-bit	64-bit	64-bit	64-bit	2-bit	4-bit	1-bit
flowID	opcode	result	req_counter	resp_counter	parent	children flags	Gflag

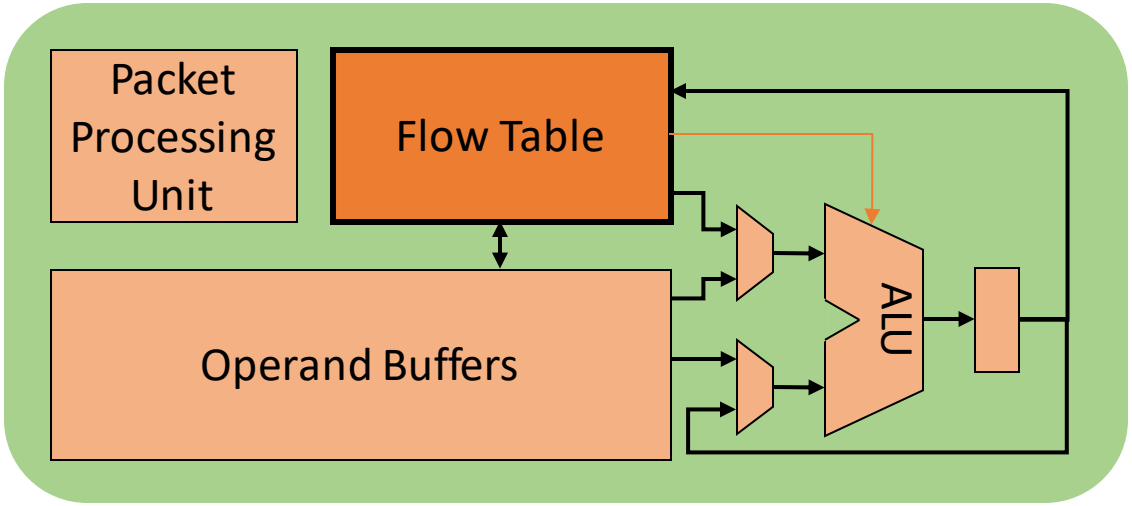
Flow Table



- Flow Table Entry

64-bit	6-bit	64-bit	64-bit	64-bit	2-bit	4-bit	1-bit
flowID	**opcode**	**result**	req_counter	resp_counter	parent	children flags	Gflag

Flow Table

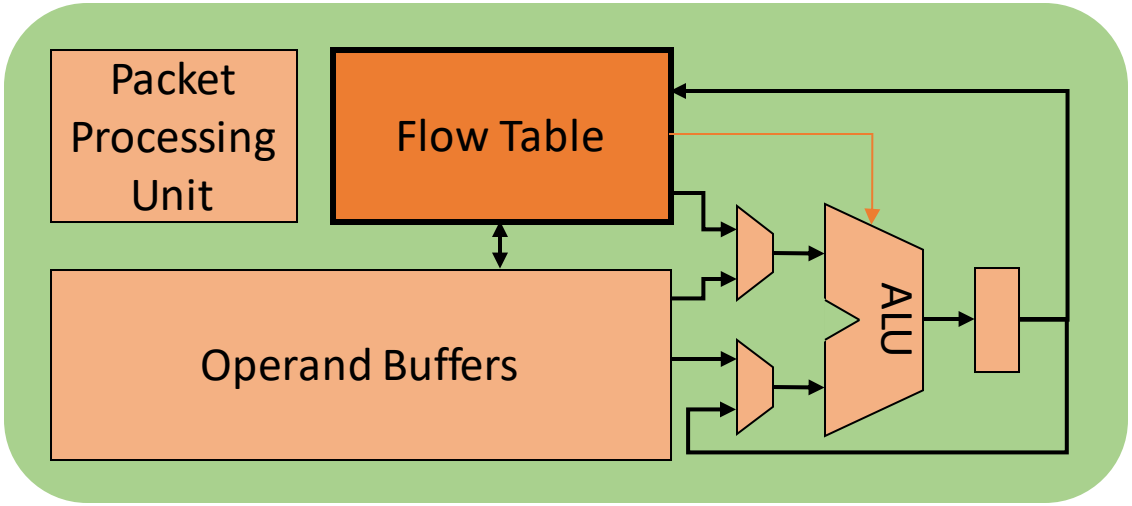


- Flow Table Entry

64-bit	6-bit	64-bit	64-bit	64-bit	2-bit	4-bit	1-bit
flowID	opcode	result	req_counter	resp_counter	**parent**	**children flags**	Gflag

  - Maintain tree structure

Flow Table

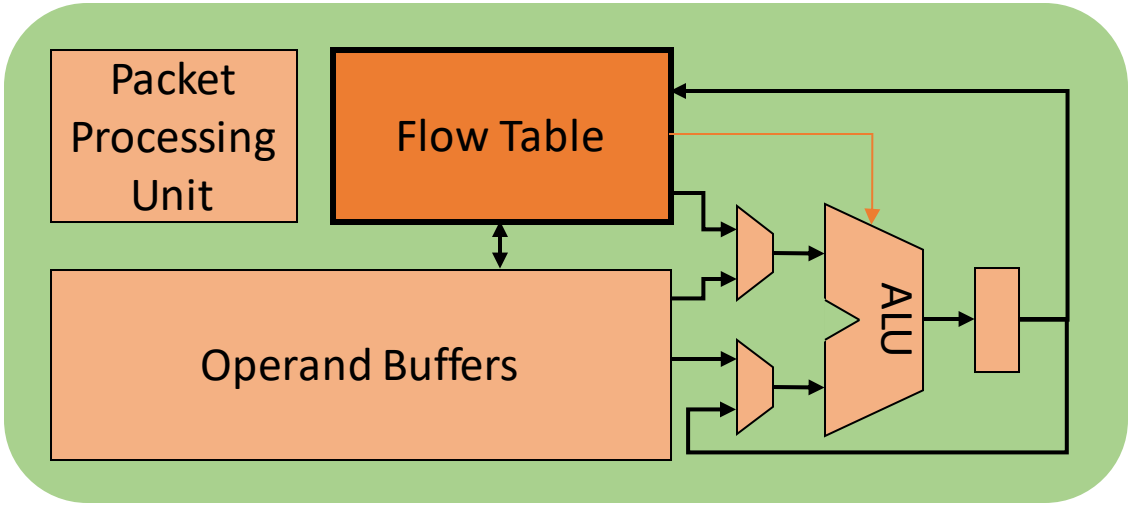


- Flow Table Entry

64-bit	6-bit	64-bit	64-bit	64-bit	2-bit	4-bit	1-bit
flowID	opcode	result	**req_counter**	**resp_counter**	parent	children flags	Gflag

  - Maintain tree structure
  - Keep track of state information of each flow

Flow Table

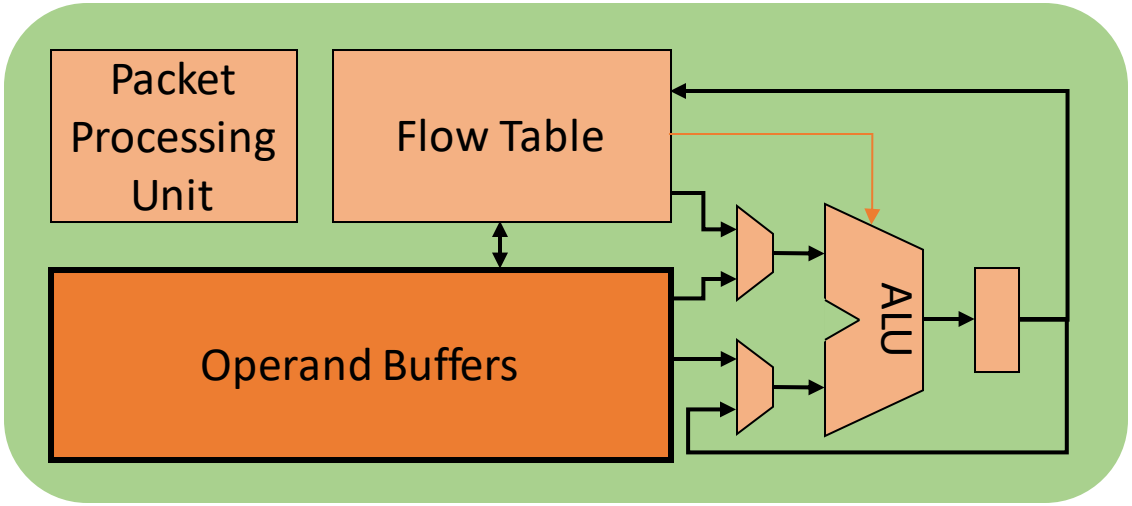


- Flow Table Entry

64-bit	6-bit	64-bit	64-bit	64-bit	2-bit	4-bit	1-bit
flowID	opcode	result	req_counter	resp_counter	parent	children flags	**Gflag**

  - Maintain tree structure
  - Keep track of state information of each flow

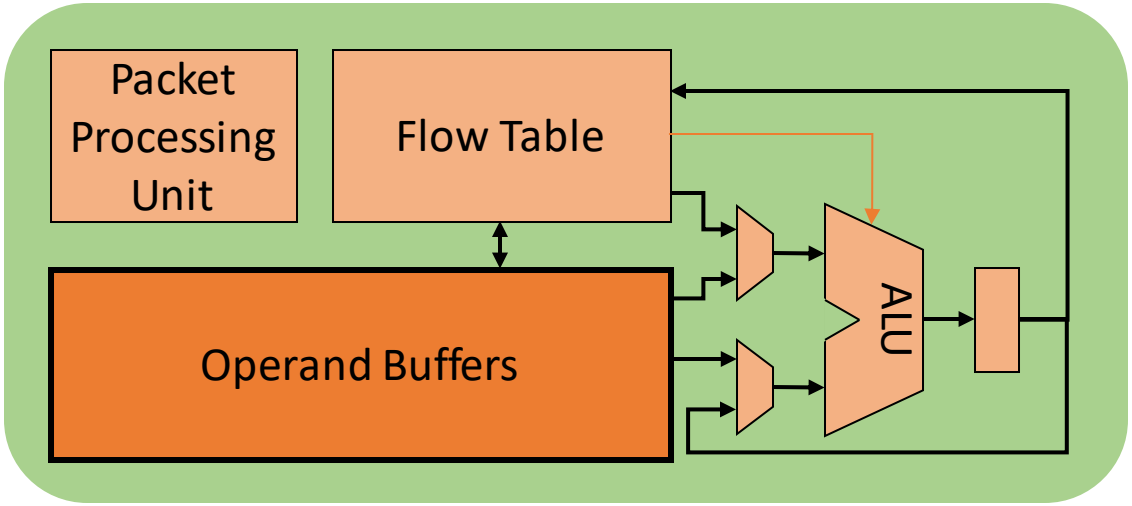
Operand Buffers



- Operand Buffer Entry

64-bit	64-bit	1-bit	64-bit	1-bit
flowID	op_value1	op_ready1	op_value2	op_ready2

Operand Buffers

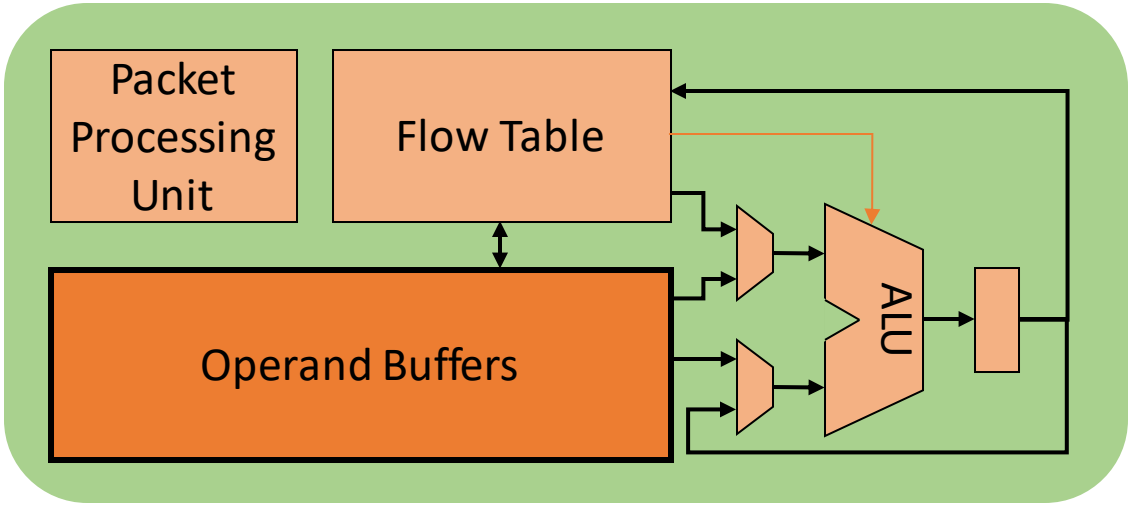


- Operand Buffer Entry

64-bit	64-bit	1-bit	64-bit	1-bit
flowID	**op_value1**	op_ready1	**op_value2**	op_ready2

  - Shared temporal storage

Operand Buffers



- Operand Buffer Entry

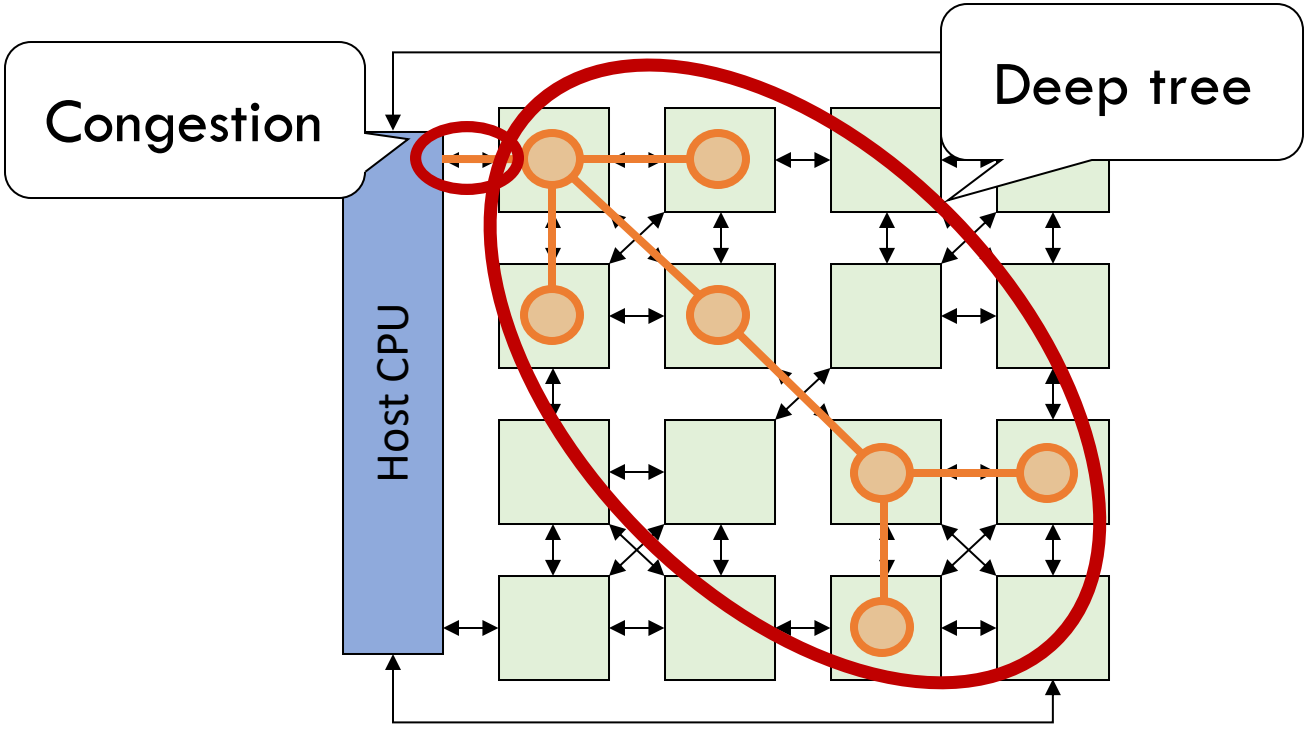
64-bit	64-bit	1-bit	64-bit	1-bit
flowID	op_value1	**op_ready1**	op_value2	**op_ready2**

  - Shared temporal storage
  - Fire for computation in dataflow

Enhancements in Active-Routing

~~One Tree Per Flow~~

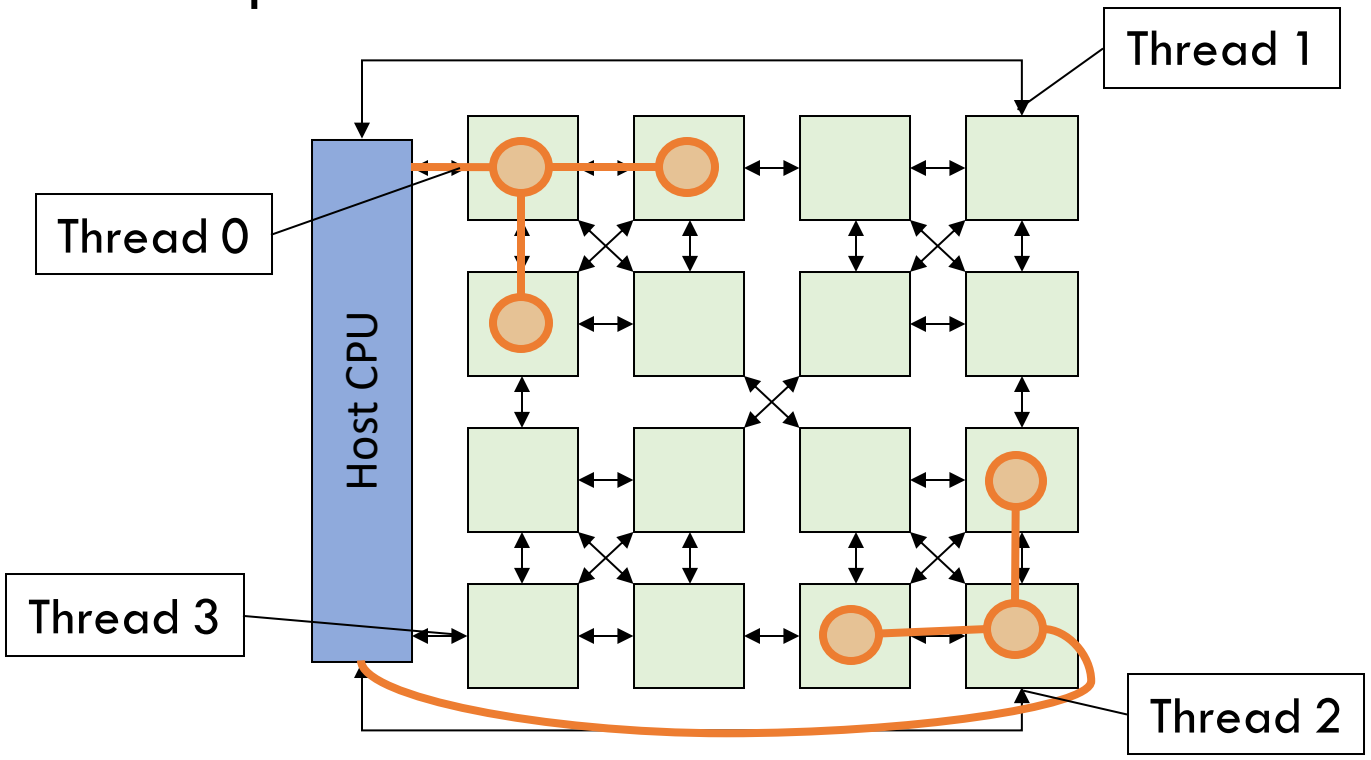
- One tree from one memory port



  - Deep tree
  - Congestion at memory port

Multiple Trees Per Flow

- Build multiple trees



  - ART-tid: interleave the thread ID
  - ART-addr: nearest port based on operands' address

Exploit Memory Access Locality

- Pure reduction
  - Irregular (random)
  - Regular

```
for (i = 0; i < n; i++)
{
      sum += *Ai;
}
```

- Reduction on intermediate results
  - Irregular-Irregular (II)
  - Regular-Irregular (RI)
  - Regular-Regular (RR)

```
for (i = 0; i < n; i++)
{
      sum += *Ai × *Bi;
}
```

- **Offload cache block granularity for regular accesses**

Evaluation

Methodology

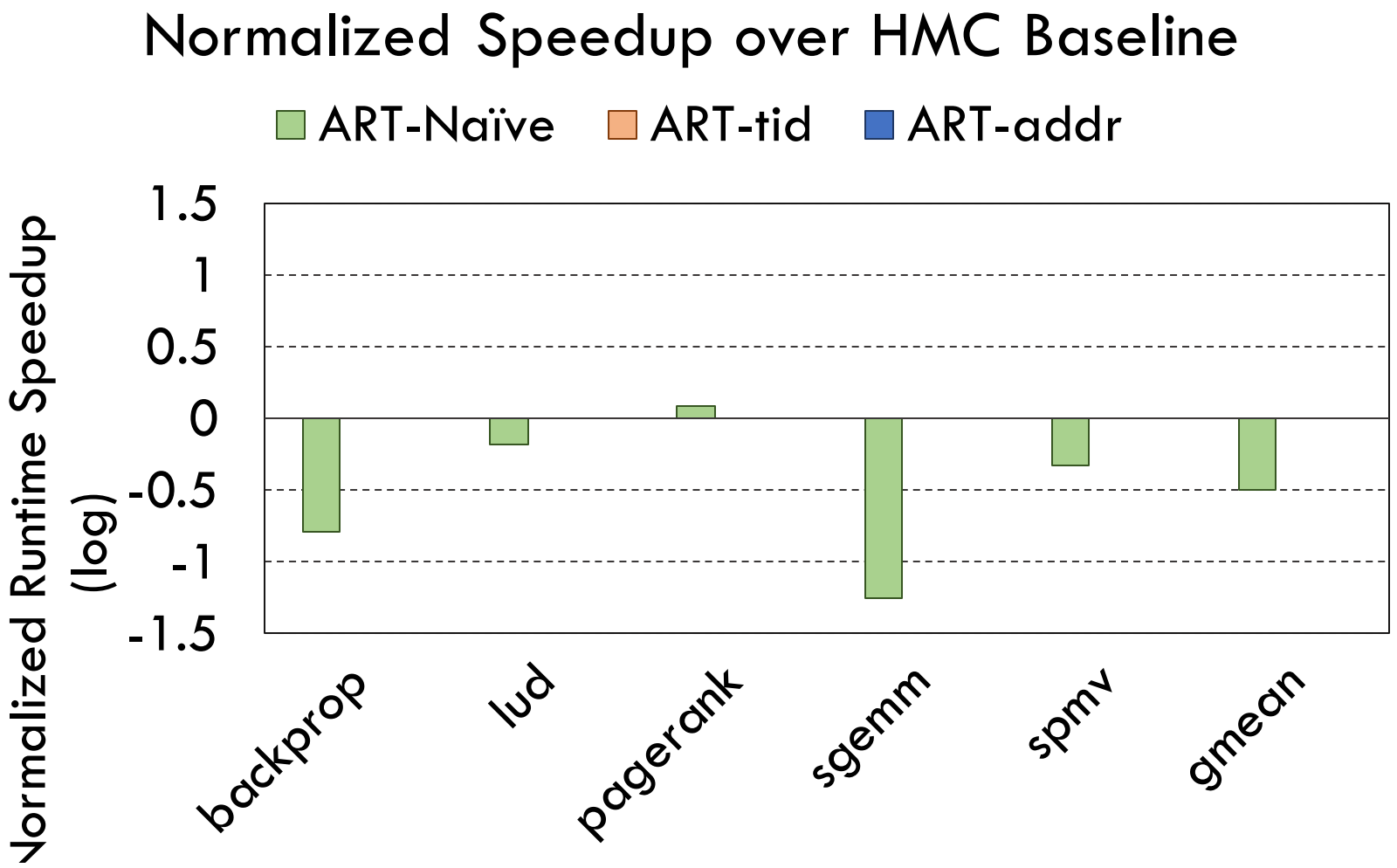
- Compared techniques
  - HMC Baseline
  - PIM-Enabled Instruction (PEI)
  - Active-Routing-threadID (ART-tid)
  - Active-Routing-address (ART-addr)
- Tools: Pin, McSimA+ and CasHMC
- System configurations
  - 16 O3 cores at 2 GHz
  - 16 memory cubes in dragonfly topology
  - Minimum routing with virtual cut-through
  - Active-Routing Engine

    1250 MHz, 16 flow table entries, 128 operand entries

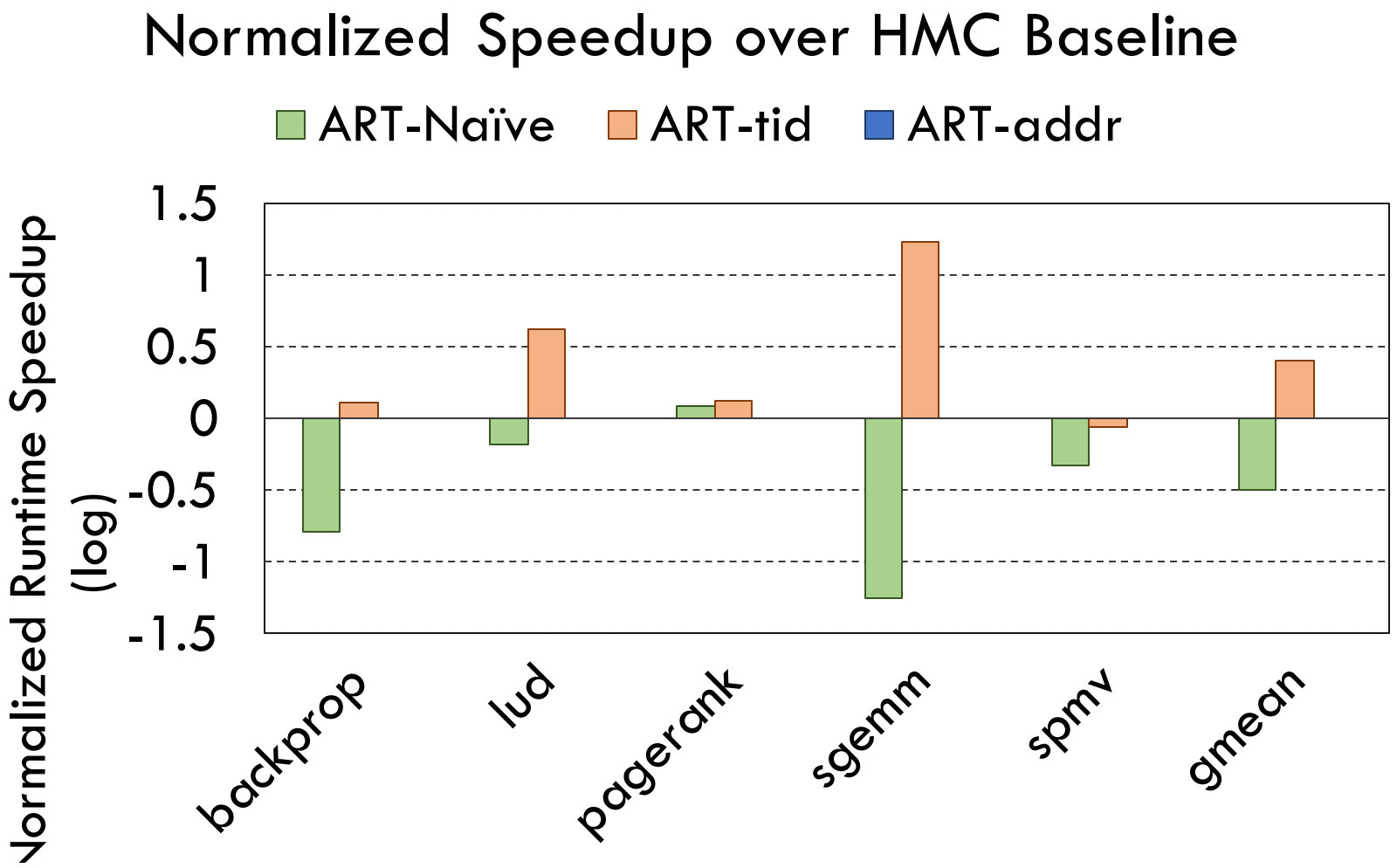
Workloads

- Benchmarks (graph app, ML kernels, etc.)
  - backprop
  - lud
  - pagerank
  - sgemm
  - spmv
- Microbenchmarks
  - reduce (sum reduction)
  - rand_reduce
  - mac (multipy-and-accumulate)
  - rand_mac

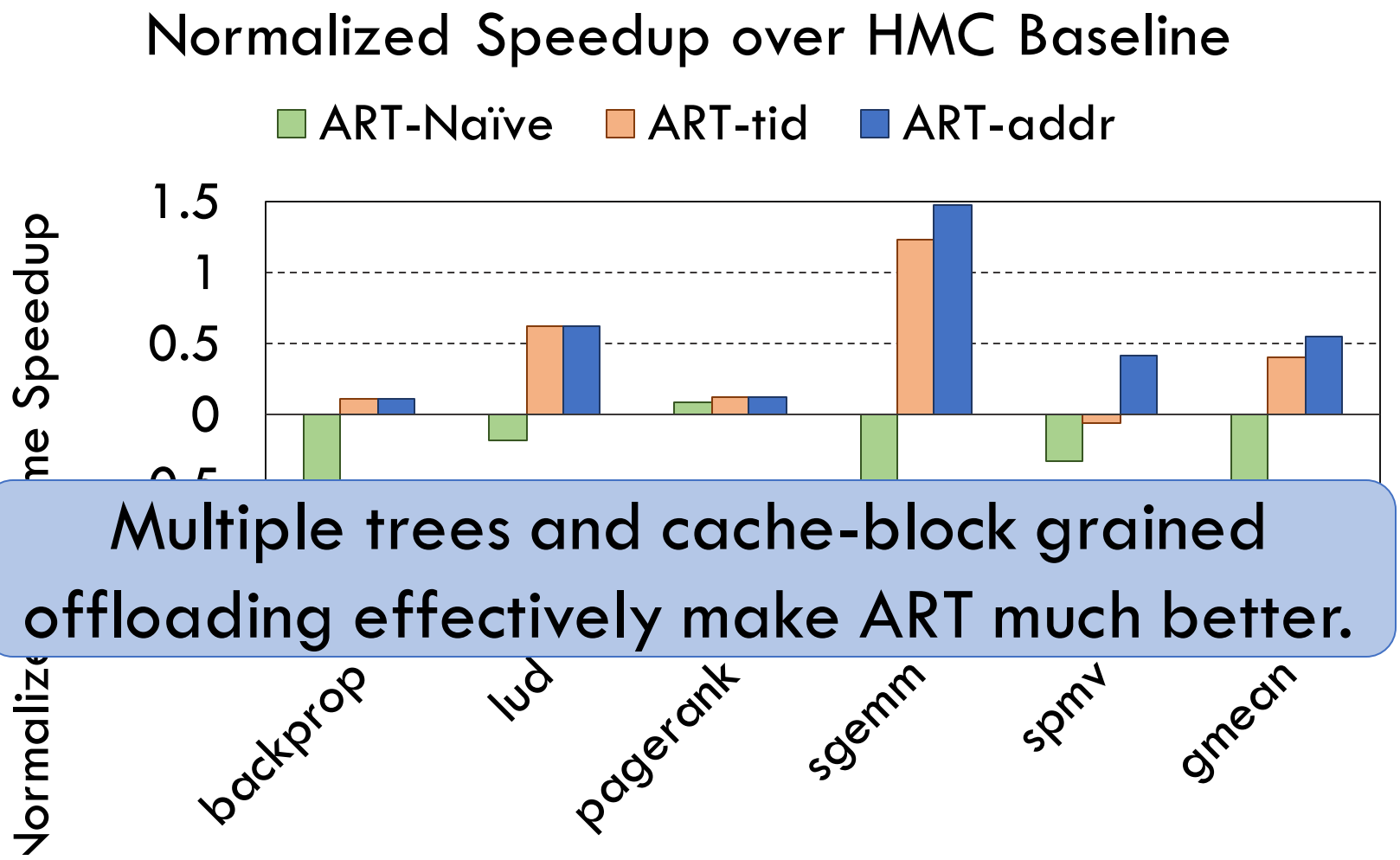
Comparison of Enhancements in Active-Routing



Comparison of Enhancements in Active-Routing

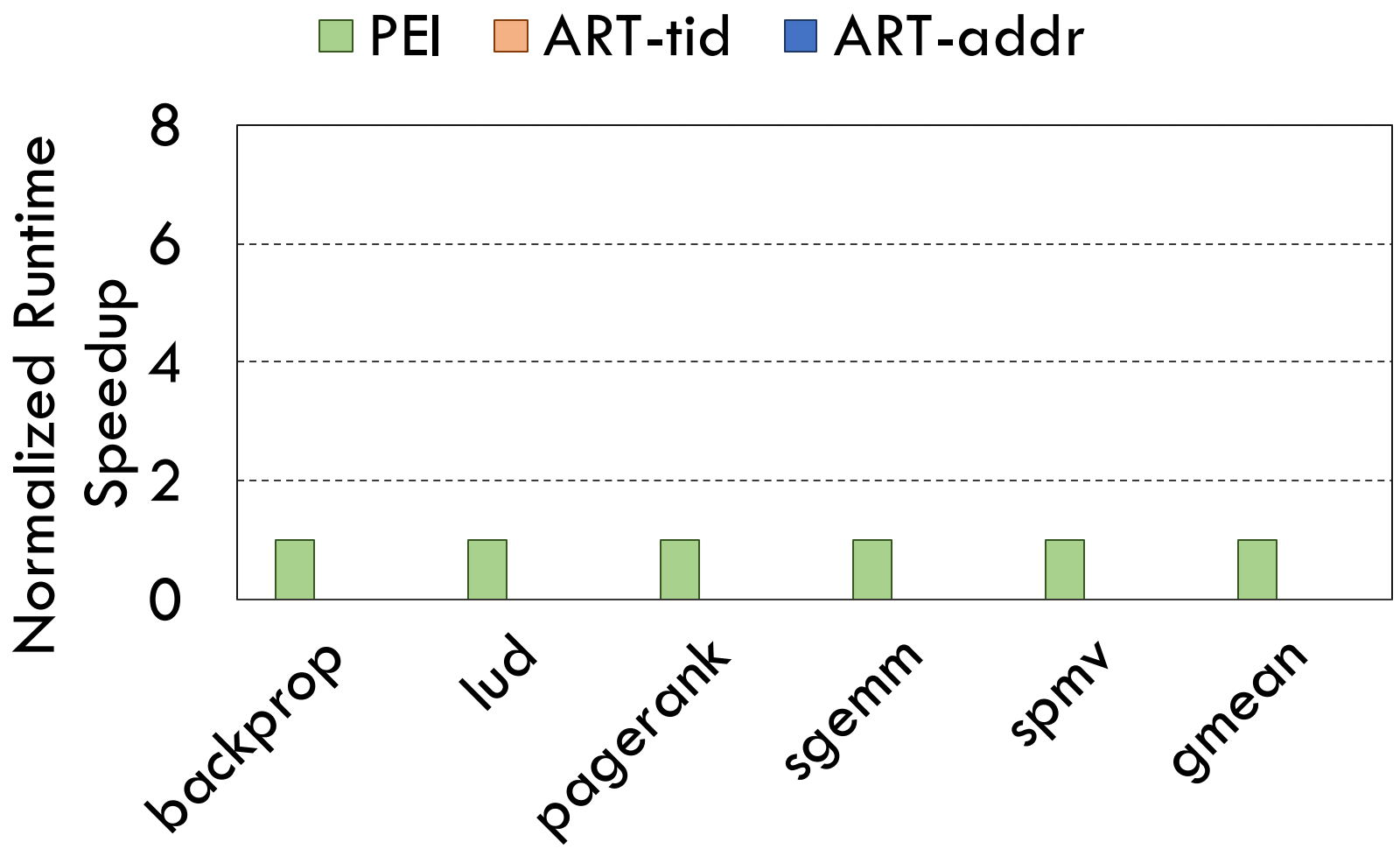


Comparison of Enhancements in Active-Routing

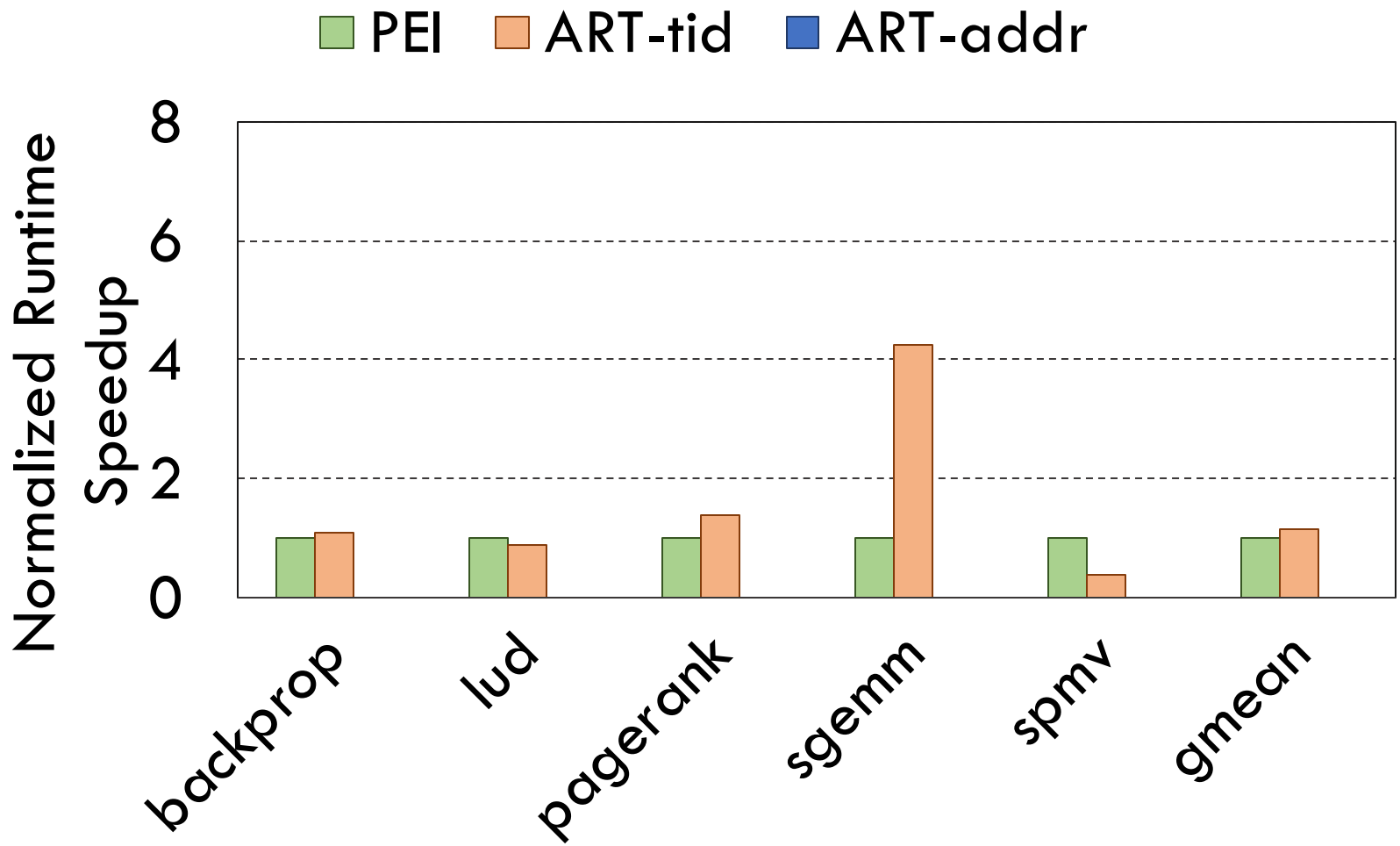


Multiple trees and cache-block grained offloading effectively make ART much better.

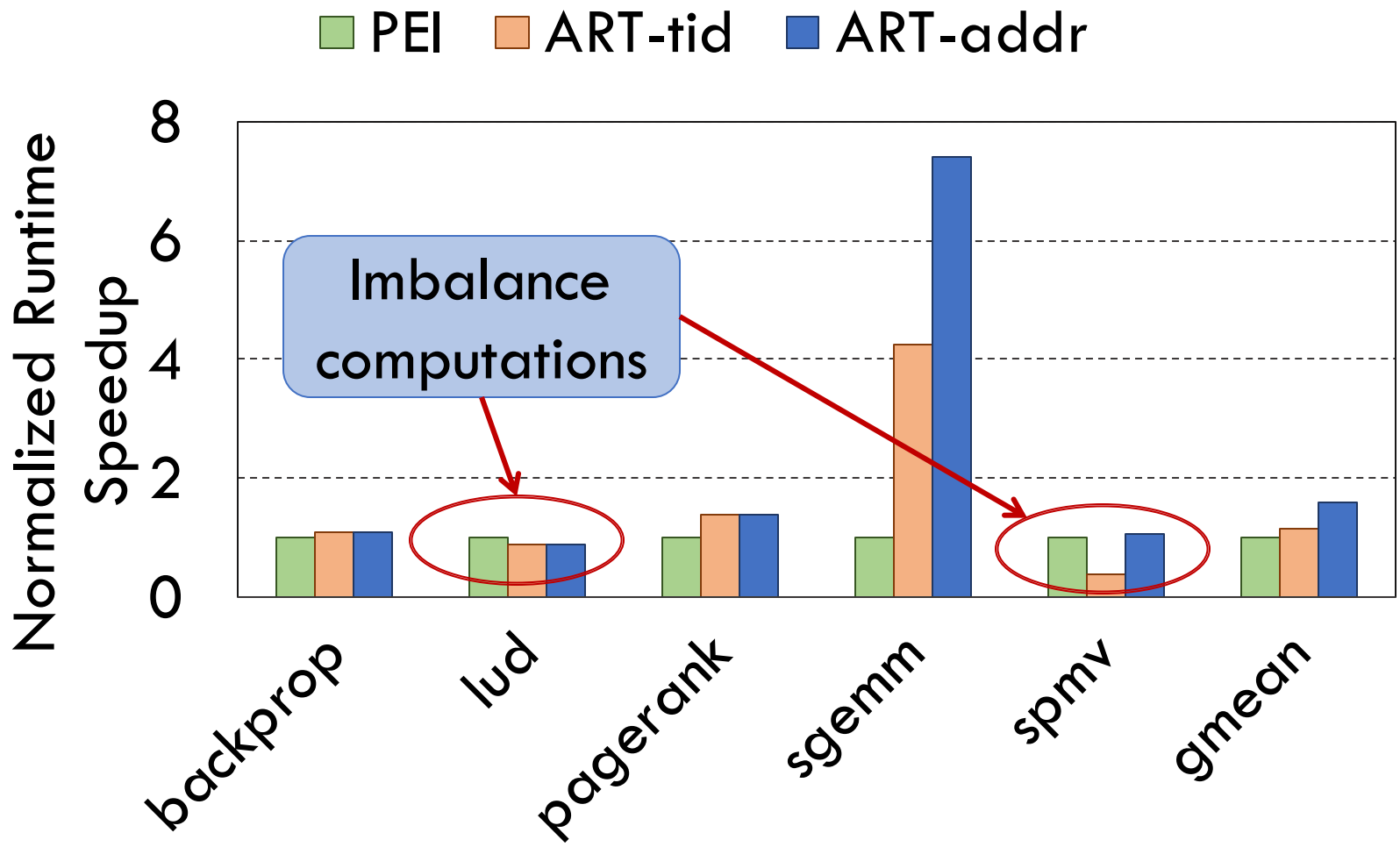
Benchmark Performance



Benchmark Performance

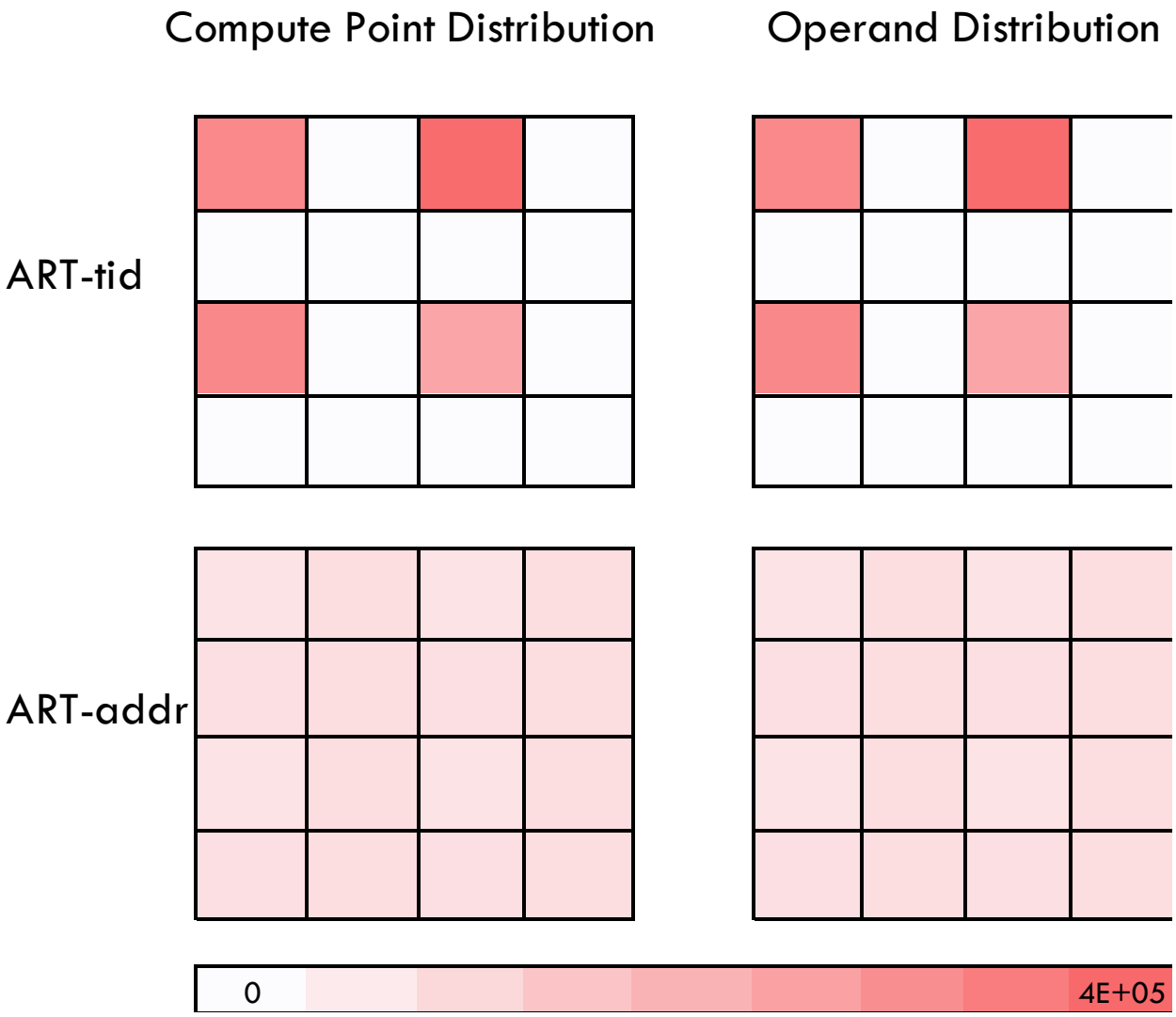


Benchmark Performance

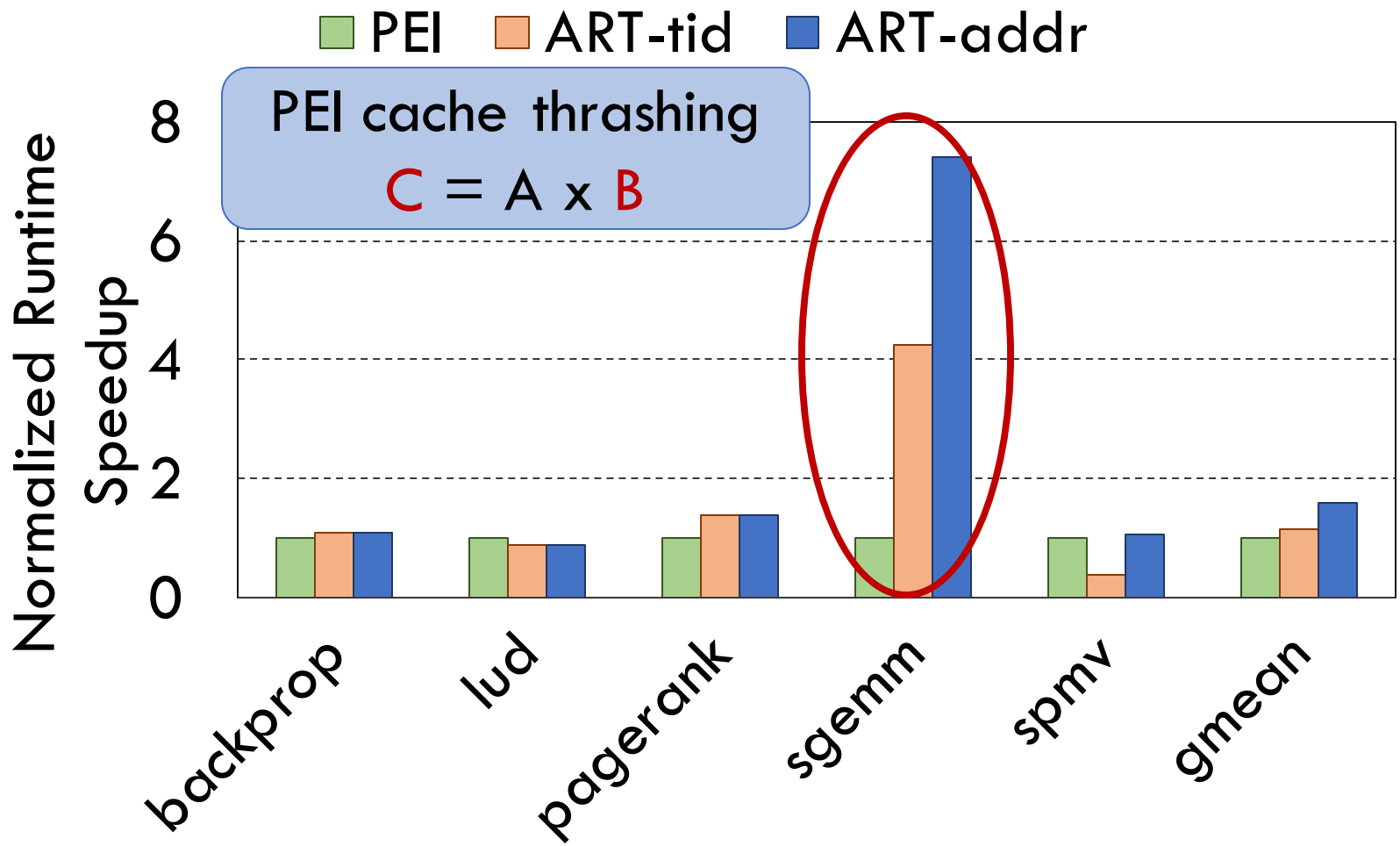


- In general, ART-addr > ART-tid > PEI

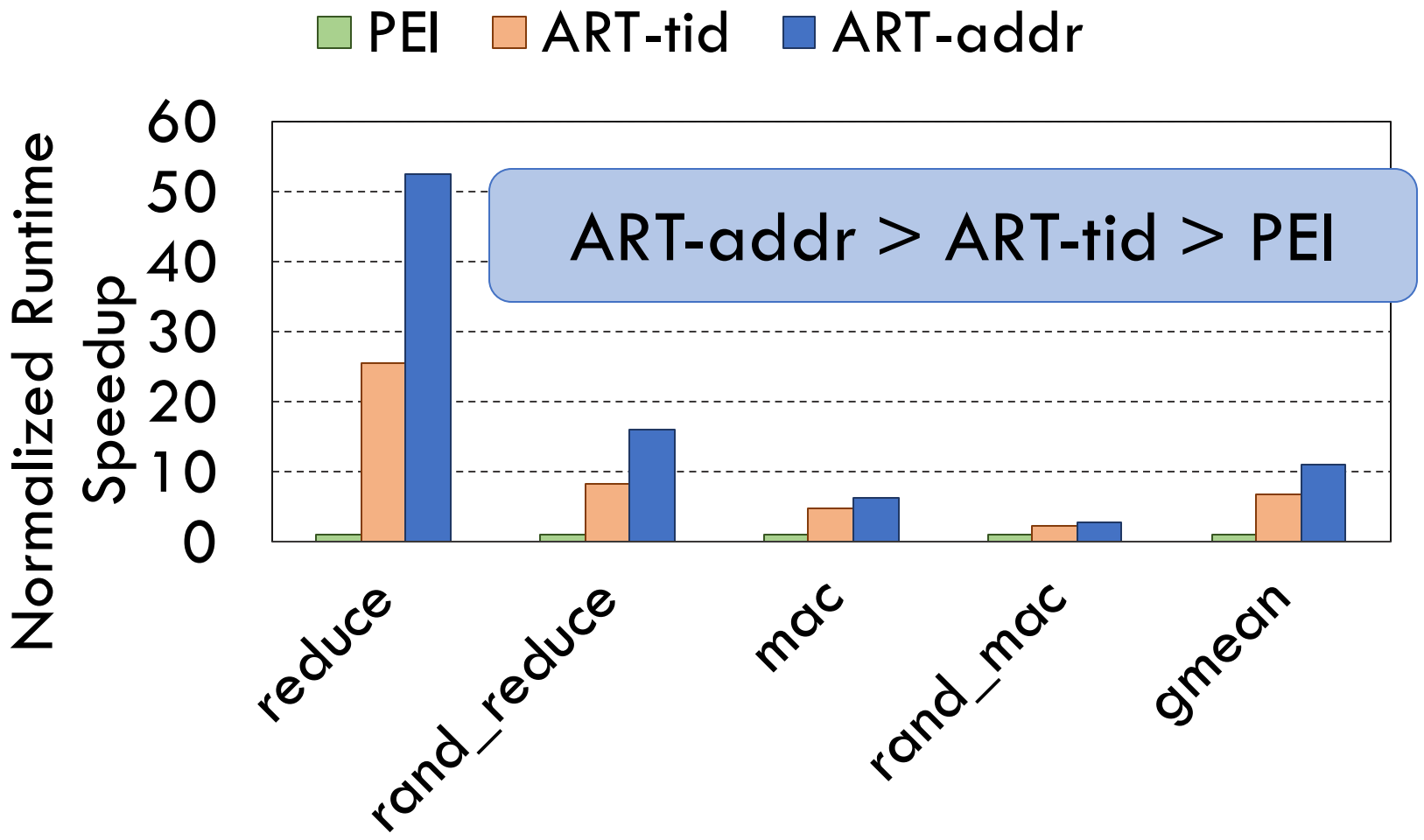
Analysis of spmv



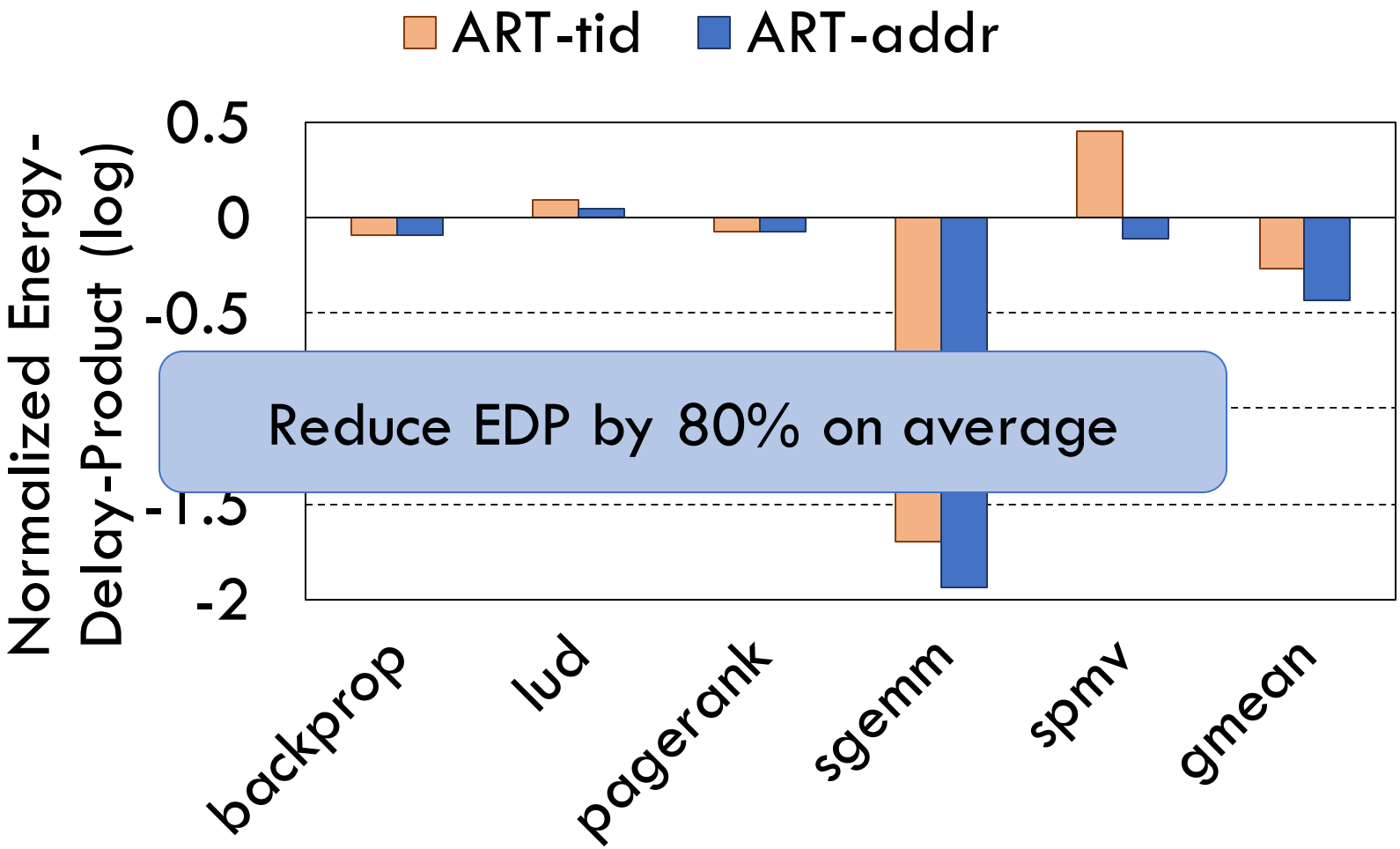
Benchmark Performance



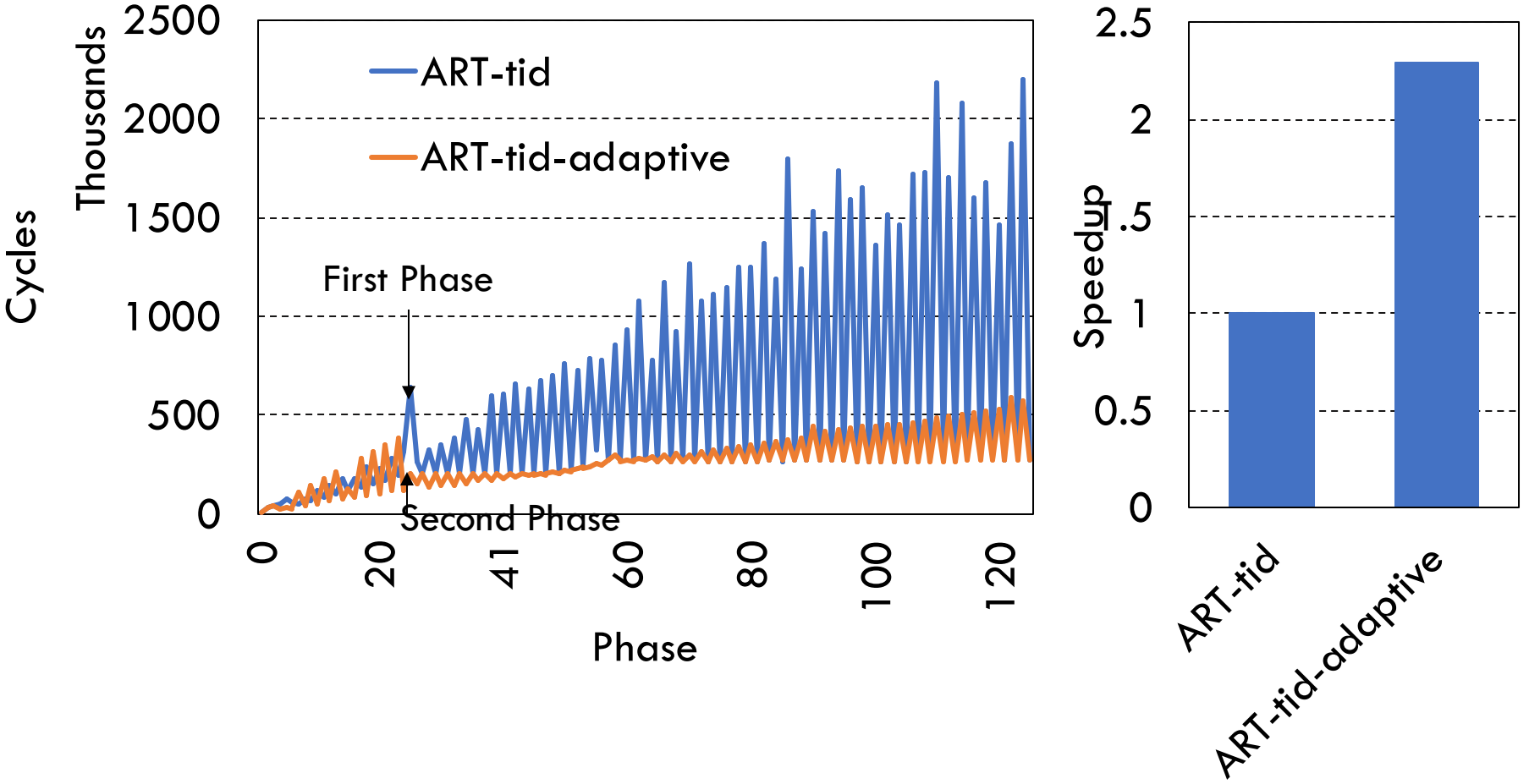
Microbenchmark Performance



Energy-Delay Product



Dynamic Offloading Case Study (lud)



- ART-tid-adaptive: dynamic offloading based on locality and reuse

Conclusion

- Propose Active-Routing in-network computing architecture which computes near-data in the memory network in data-flow style
- Present a three-phase processing procedure for Active-Routing
- Categorize memory access patterns to exploit the locality and offload computations in various granularities
- Active-Routing achieves up to 7x speedup with 60% average performance improvement and reduce energy-delay product by 80% on average

Thank You & Questions

Jiayi Huang

jyhuang@cse.tamu.edu

Active-Routing: Compute on the Way for Near-Data Processing

Jiayi Huang, Ramprakash Reddy Puli, Pritam Majumder
Sungkeun Kim, Rahul Boyapati, Ki Hwan Yum and EJ Kim

Backup

API Extensions

- `UpdateRR(void *src1, void *src2, void *target, int op);`
  - Offload both operands in cache–block granularity
- `UpdateRI(void *src1, void *src2[], void *target, int op);`
  - Fetch irregular data then offload to regular operands location for cache-block granularity processing
- `UpdateII(void *src1, void *src2, void *target, int op);`
  - Offload both operands in single element granularity
- `Update(type src1, type src2, void *target, int op);`
  - Support general PIM
- `Gather(void *target, int num_threads);`
  - As explicit barrier

Workloads (Optimization Region and Dataset)

- Benchmarks
  - **backprop**: activation calculation in feedforward pass (2M hidden units)
  - **lud**: upper and lower triangular matrix decomposition (4K matrix dimension)
  - **pagerank**: ranking score calculation (web-Google graph from SNAP Dataset)
  - **sgemm**: matrix multiplication (4K x 4K matrix)
  - **spmv**: matrix-vector multiplication loop (4K dimension with 0.7 sparsity)

- Microbenchmarks
  - **reduce**: sum reduction over a sequential vector (64K dimension)
  - **rand_reduce**: sum reduction over random elements (64K dimension)
  - **mac**: multipy-and-accumulate over two sequential vectors (2 64K-D vectors)
  - **rand_mac**: multiply-and-accumulate over two random element lists (2 64K-element lists)

On/Off-Chip Data Movement