Fafnir: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction

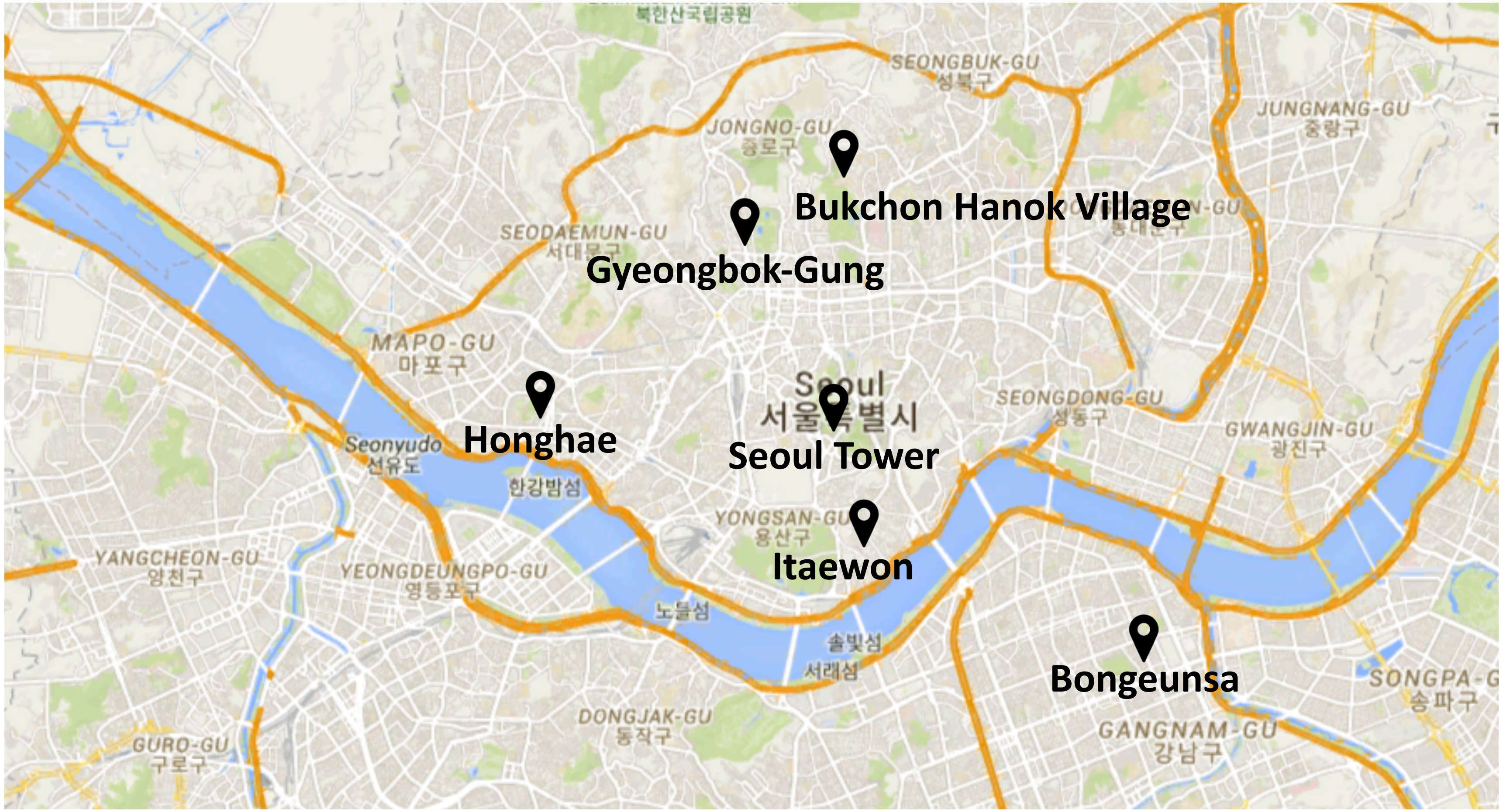
Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Da Eun Shim, Sung-Kyu Lim, and Hyesoon Kim

Georgia Tech

comparch

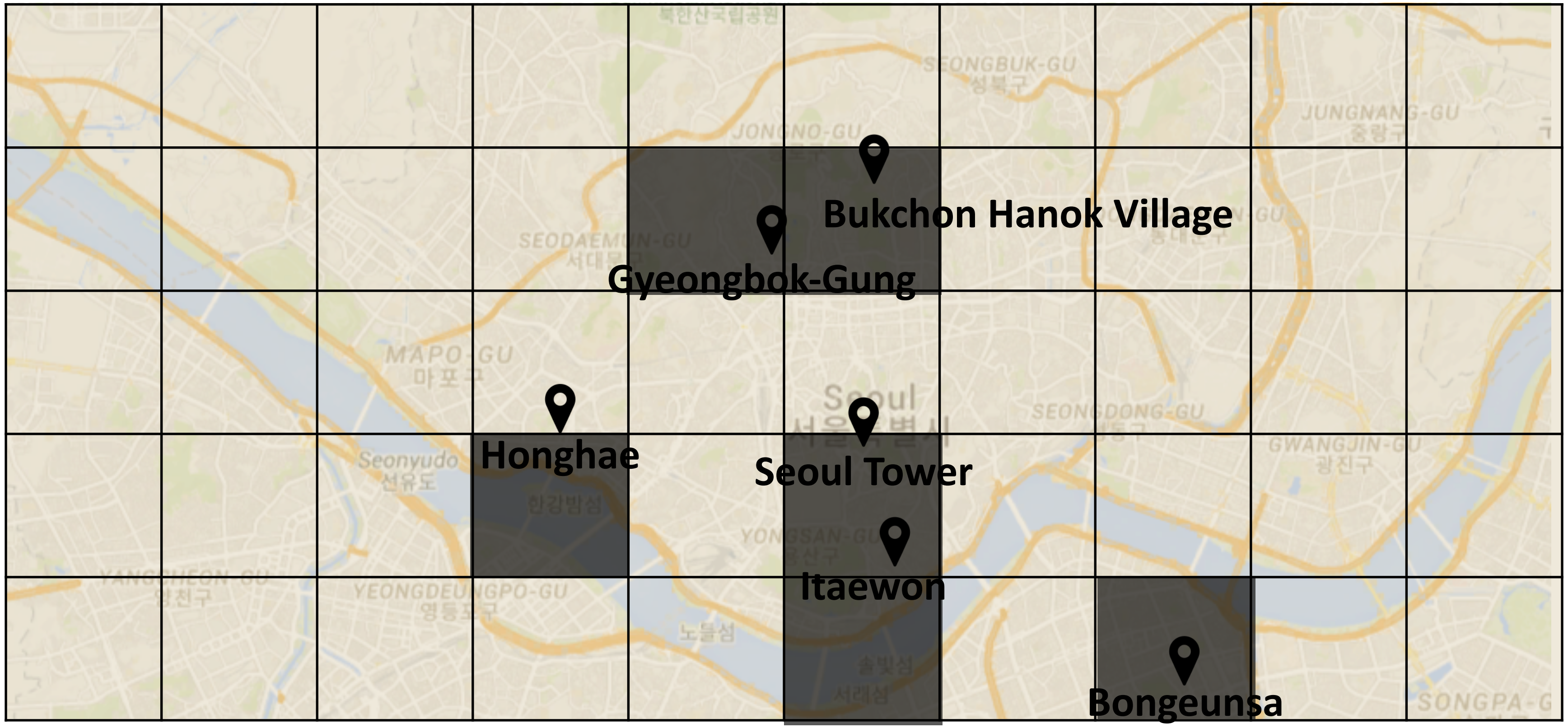
GTCAD Lab

Imagine we Were in Seoul for HPCA'21!



GTCAD Lab

Recommendation Systems Are Similar

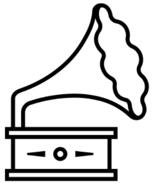


All users' data and features of movies in memory

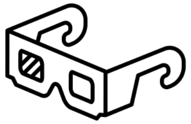
Accessed data

Recommendation Systems Suggest us…

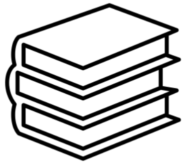
What music to listen



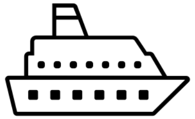
What movie to watch



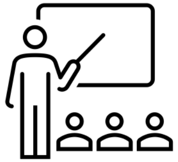
What books to read



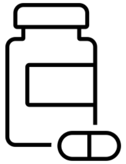
Where to go



What to learn



What medicine to take



Outline

- Main components and sparsity in recommendation system
- Prior near-memory processing approaches and their challenges
- Fafnir: our proposed efficient near-memory intelligent reduction tree
  - Main contributions
  - Architecture and implementation
- Experimental setup
- Performance evaluation
  - Latency
  - End-to-end inference speedup
  - Scalability
  - Power consumption
- Conclusions

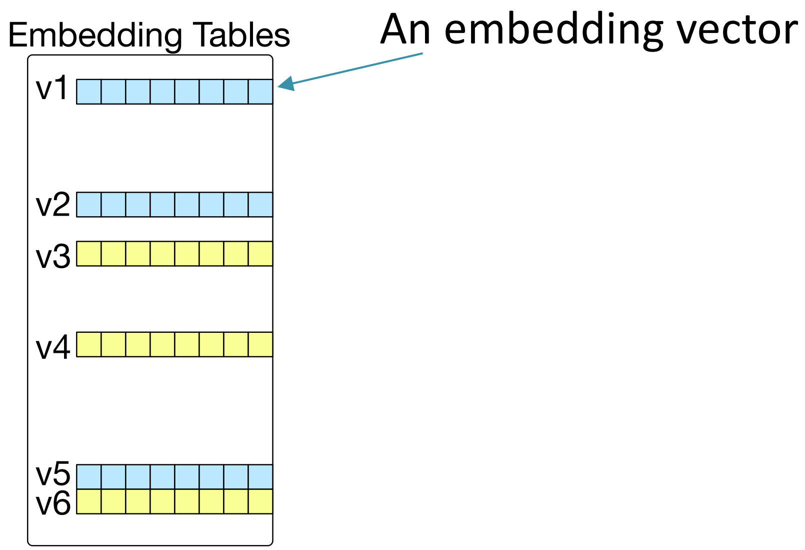
Outline

- **Main components and sparsity in recommendation system**
- Prior near-memory processing approaches and their challenges
- Fafnir: our proposed efficient near-memory intelligent reduction tree
  - Main contributions
  - Architecture and implementation
- Experimental setup
- Performance evaluation
  - Latency
  - End-to-end inference speedup
  - Scalability
  - Power consumption
- Conclusions

Main Components and Sparsity

Recommendation systems consist of

- Embedding tables, accessing to which is **sparse!**

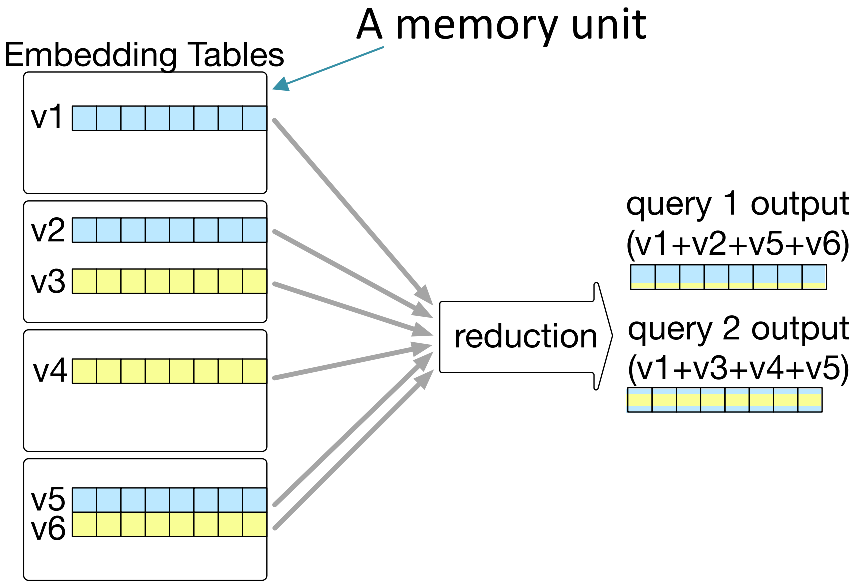


v1 to v6 are some embedding vectors we use in our example throughout this presentation.
We randomly color them in blue and yellow to distinguish them when we apply an operation on them.

Main Components and Sparsity

Recommendation systems consist of

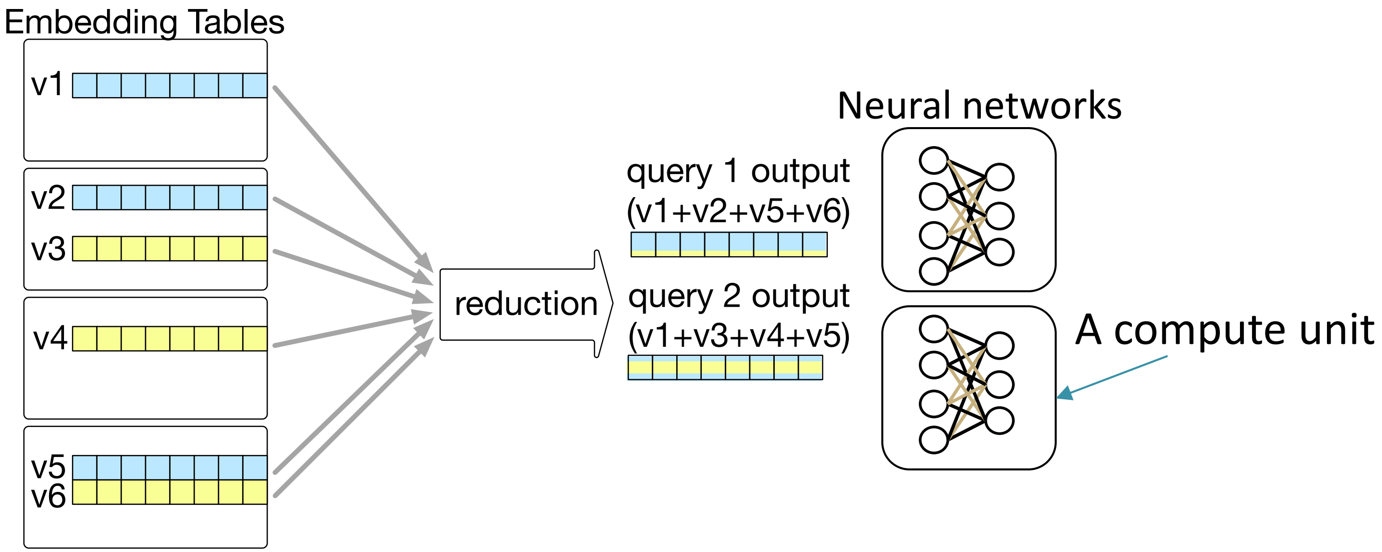
- Embedding tables, accessing to which is **sparse!**



Main Components and Sparsity

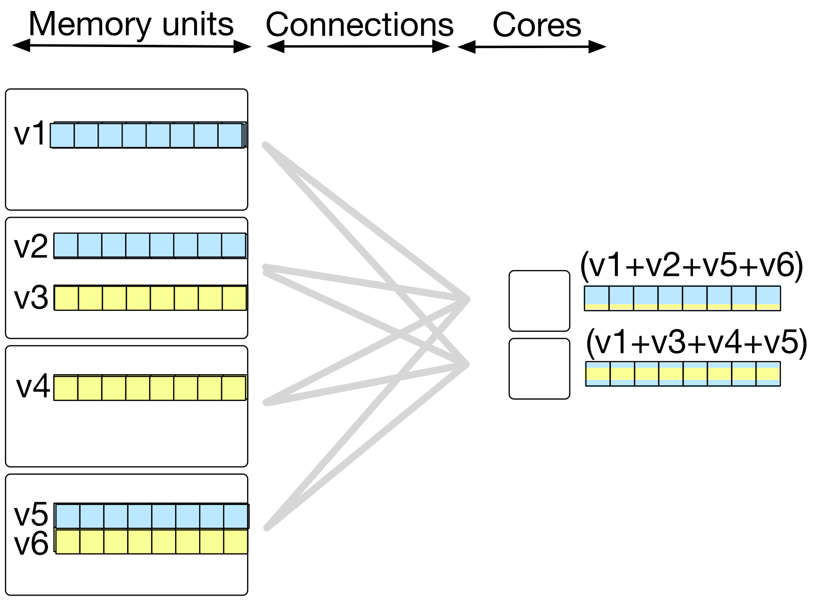
Recommendation systems consist of

- Embedding tables, accessing to which is **sparse!**
- Neural networks



Data Movement Is a Big Challenge

Embedding vectors need to be constantly transferred from memory units to the cores

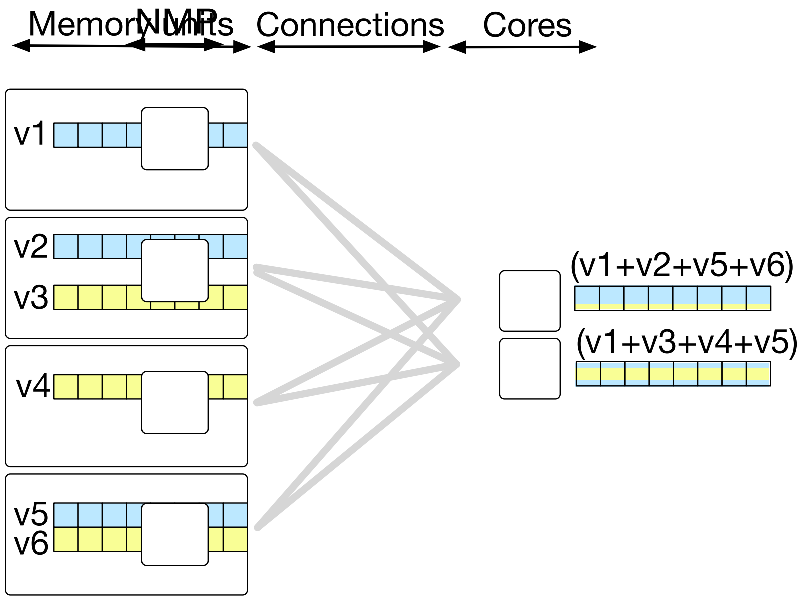


Outline

- Sparsity in recommendation system
- **Prior near-memory processing approaches and their challenges**
- Fafnir: our proposed efficient near-memory intelligent reduction tree
  - Main contributions
  - Architecture and implementation
- Experimental setup
- Performance evaluation
  - Latency
  - End-to-end inference speedup
  - Scalability
  - Power consumption
- Conclusions

Near-Memory Processing (NMP)

Prior proposals suggest performing reduction near memory to transfer less data from memory units to the cores

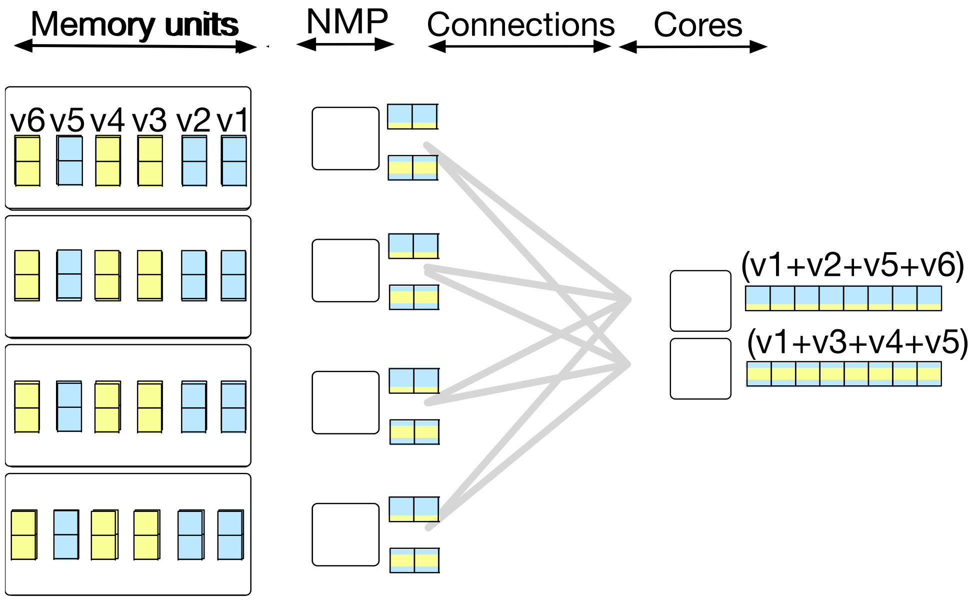


Prior NMP Solutions: TensorDIMM

Guarantees data movement reduction

- Example: transfers only two vectors instead of six

Challenge: Does not fully utilize row buffer locality



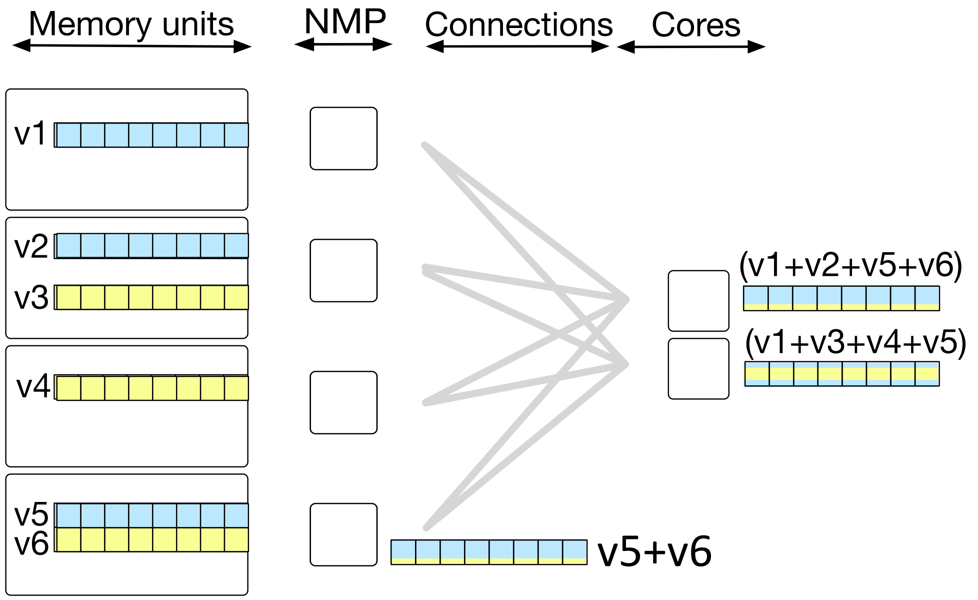
Y. Kwon, et al. “Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning,” in *MICRO*, 2019.

Prior NMP Solutions: RecNMP

Fully utilizes row buffer locality

Challenge: Does not guarantee data movement reduction

- Example: still transfers six vectors (v1, v2, v3, v4, v5, v5+v6)



L.Ke, et al. "Recnmp: Accelerating personalized recommendation with near-memory processing," *ISCA*, 2020.

Key Insight

We cannot process embedding vectors where they reside

- Because they are not co-located in memory!

We do not want to process embedding vectors in the processing cores

- Because it causes huge amount of data movement

We **process** embedding vectors **while we gather** them from random locations of memory

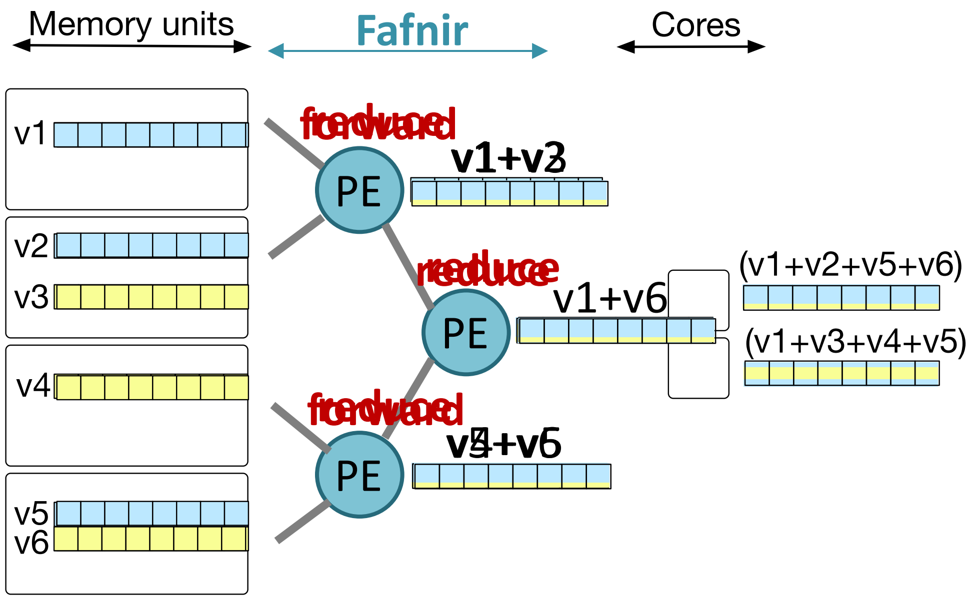
Outline

- Sparsity in recommendation system
- Prior near-memory processing approaches and their challenges
- **Fafnir: our proposed efficient near-memory intelligent reduction tree**
  - **Main contributions**
  - **Architecture and implementation**
- Experimental setup
- Performance evaluation
  - Latency
  - End-to-end inference speedup
  - Scalability
  - Power consumption
- Conclusions

Fafnir – Main Contributions

Guarantees to reduce embedding vectors before sending them to cores

- Sooner (in the leaves) or later (in the root), the corresponding embedding vectors meet within the tree and get reduced



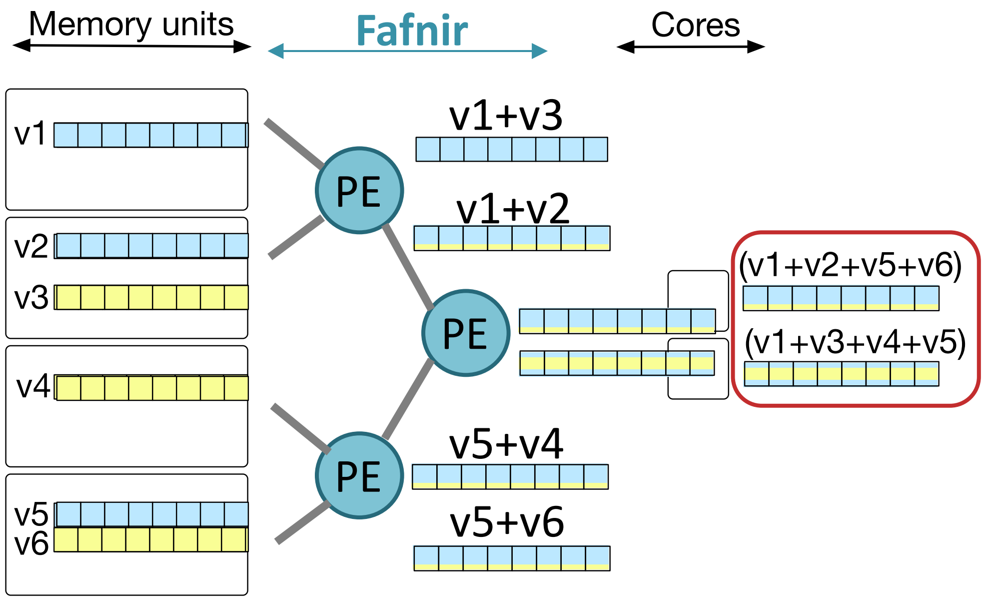
Fafnir – Main Contributions

Does not require a caching mechanism

- Reads all the unique vectors in a batch of query and use them within the tree as many times as required
- Takes advantage of embedding vector locality across multiple queries and that locality is exploited in the PE buffers through streaming operations

q1: v1,v2,v5,v6
q2: v1,v3,v4,v5

v1: q1,q2
v2: q1
v3: q2
v4: q2
v5: q1, q2
v6: q1

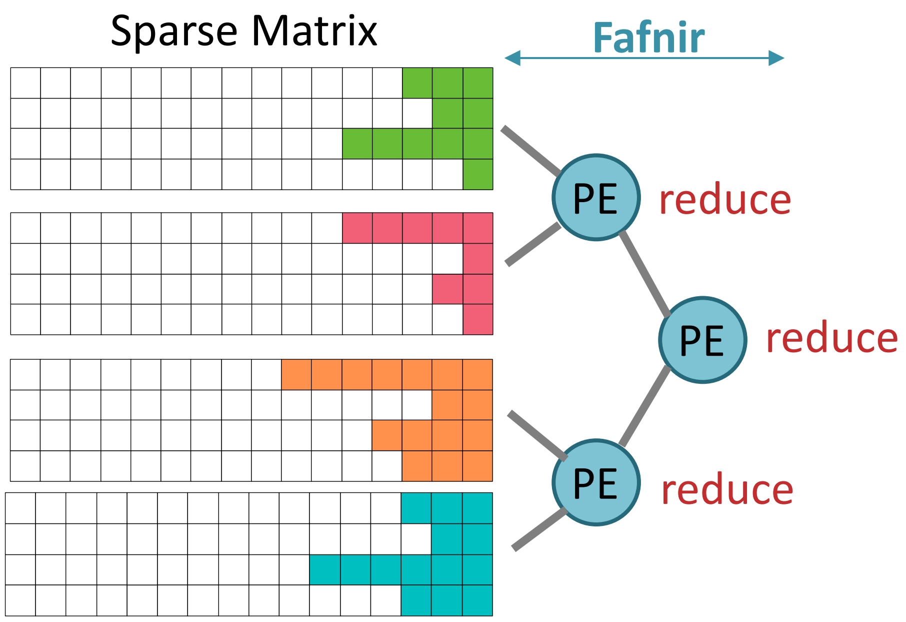


v1 and v5 are used in creating both vectors

Fafnir – Main Contributions

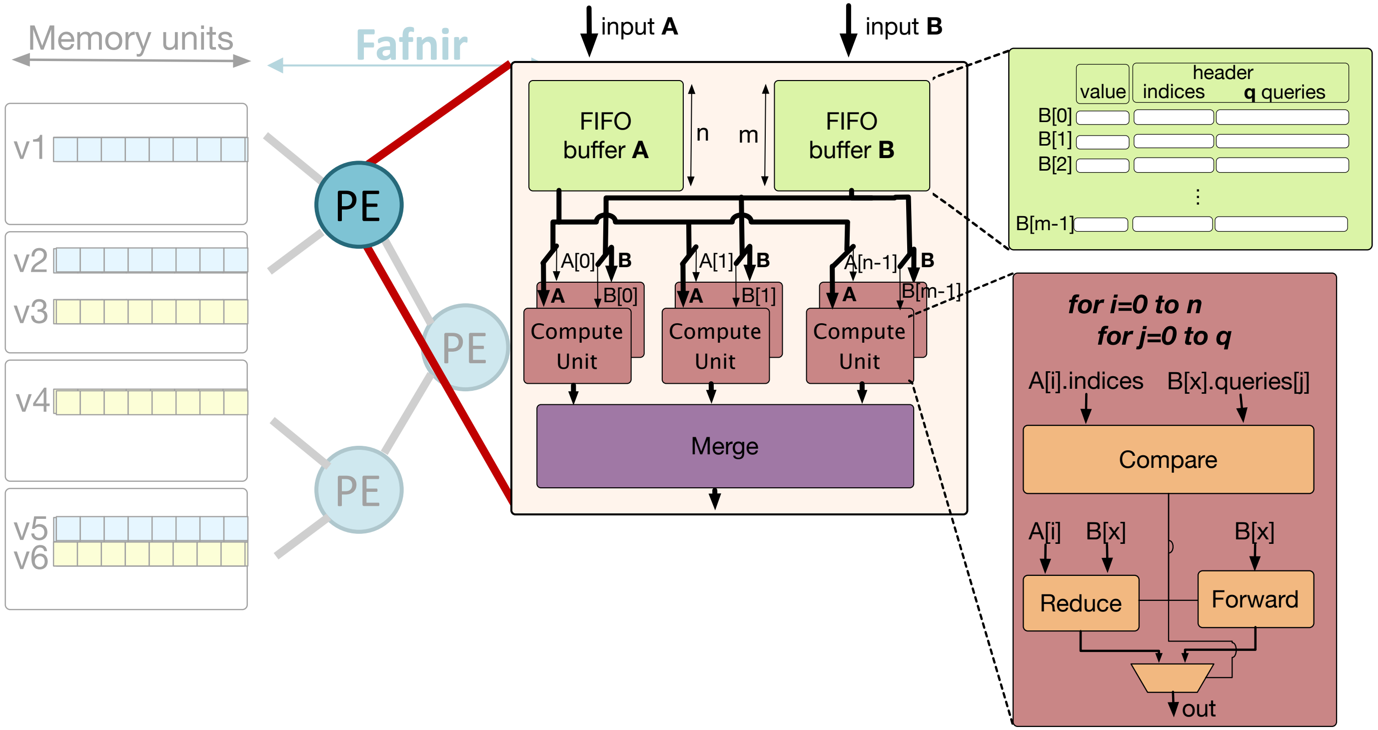
Runs sparse matrix-vector multiplication (SpMV) as well

- If all PEs always perform reduction and leaf PEs first apply multiplication



Fafnir – Architecture

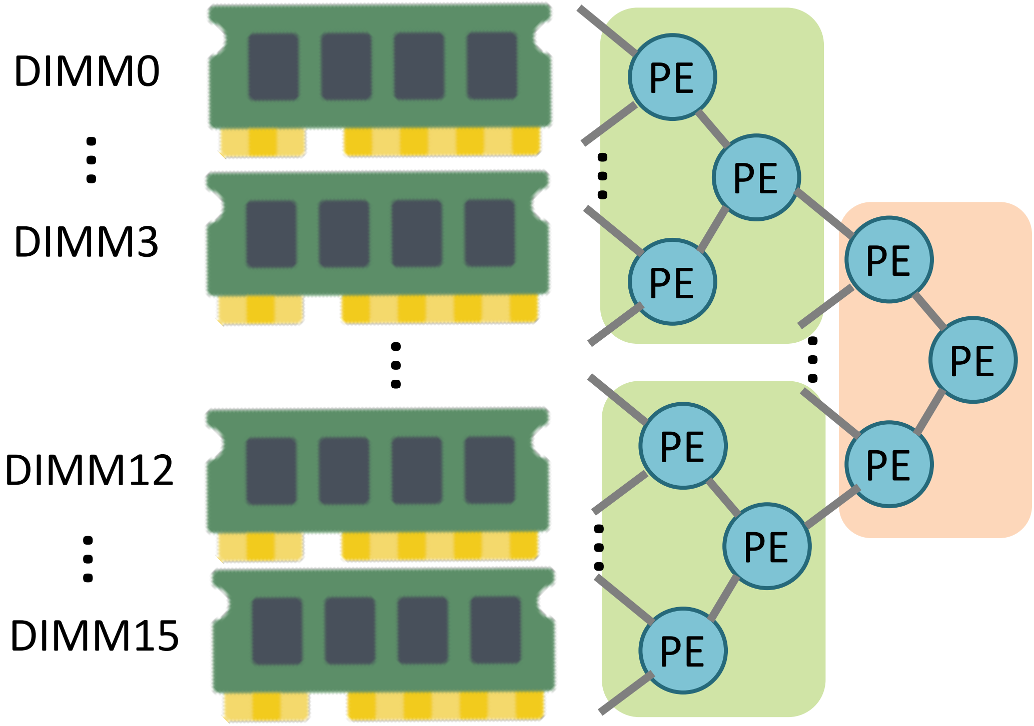
Based on their inputs, PEs decide whether to reduce or forward

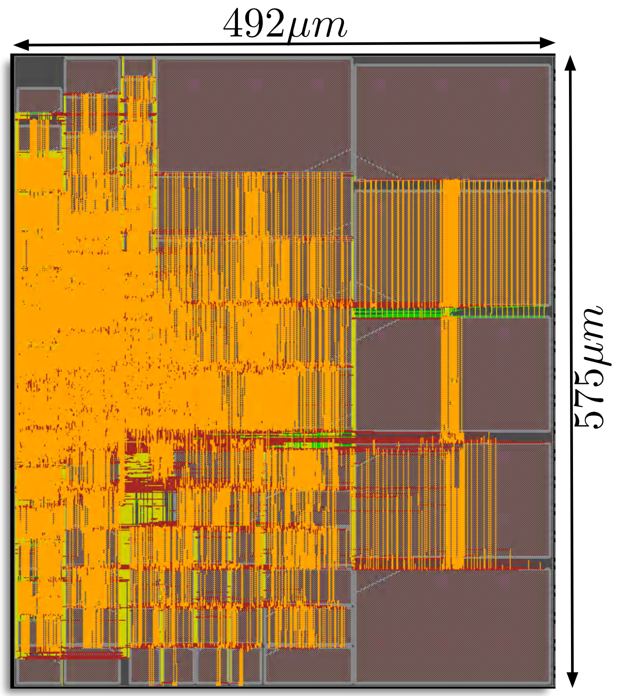


Fafnir – Implementation

We connect 32 ranks with 31 PEs and implement them at 7nm ASAP as

- Four DIMM/rank chips: $0.282\ mm^2$, $23.82\ \ mW$
- One channel chip: $0.121\ mm^2$, $16.37\ mW$





One DIMM/rank Chip

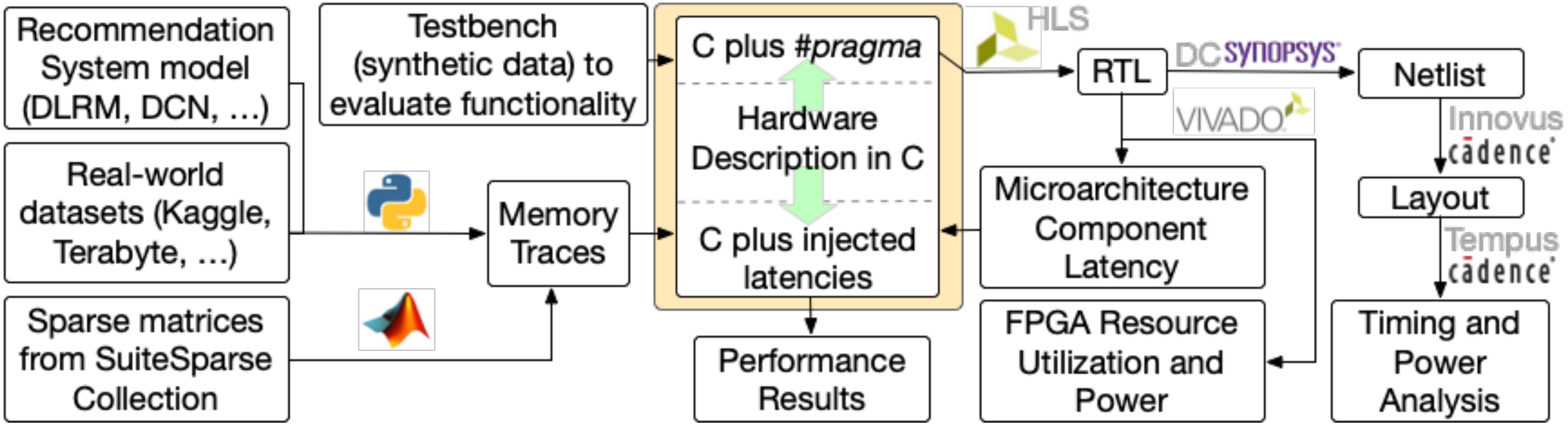
Georgia Tech comparch GTCAD Lab

Outline

- Sparsity in recommendation system
- Prior near-memory processing approaches and their challenges
- Fafnir: our proposed efficient near-memory intelligent reduction tree
  - Main contributions
  - Architecture and implementation
- **Experimental setup**
- Performance evaluation
  - Latency
  - End-to-end inference speedup
  - Scalability
  - Power consumption
- Conclusions

Experimental Setup

We implement Fafnir on XCVU9P Xilinx FPGA and ASIC design at 7nm ASAP



Georgia Tech comparch GTCAD Lab

Experimental Setup

We implement Fafnir on XCVU9P Xilinx FPGA and ASIC design at 7nm ASAP

We evaluate Fafnir for

- Recommendation systems
  - Models: DLRM and DCN
  - Data sets: Criteo Ad Kaggle and Terabyte
- SpMV on matrices from SuiteSparse data set

We compare with

- TensorDIMM (MICRO'19) and RecNMP(ISCA'20) for recommendation systems
- Two-Step (MICRO'19) approach for SpMV

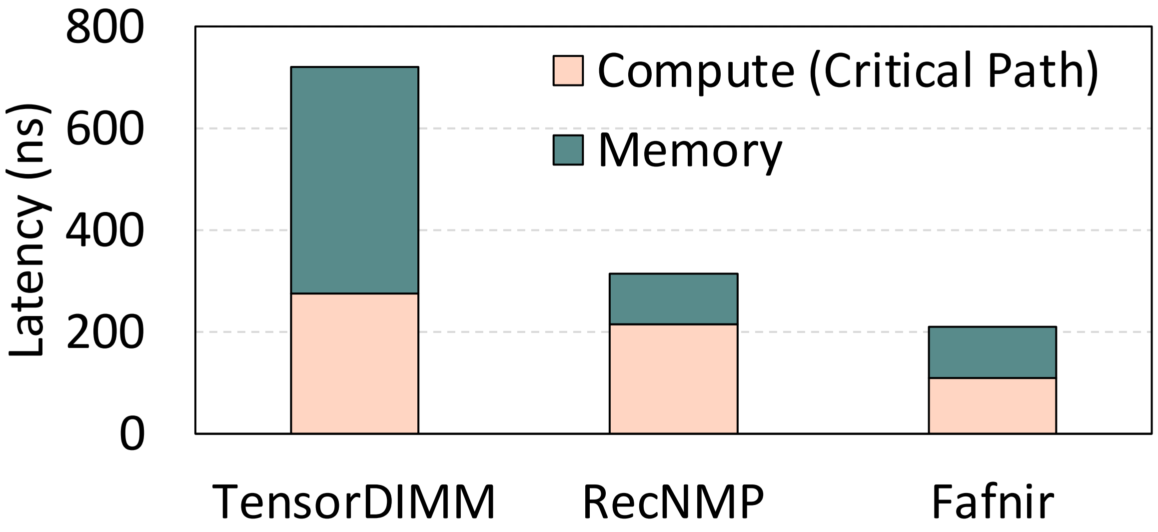
Outline

- Sparsity in recommendation system
- Prior near-memory processing approaches and their challenges
- Fafnir: our proposed efficient near-memory intelligent reduction tree
  - Main contributions
  - Architecture and implementation
- Experimental setup
- **Performance evaluation**
  - **Latency**
  - **End-to-end inference speedup**
  - **Scalability**
  - **Power consumption**
- Conclusions

Evaluation – Latency

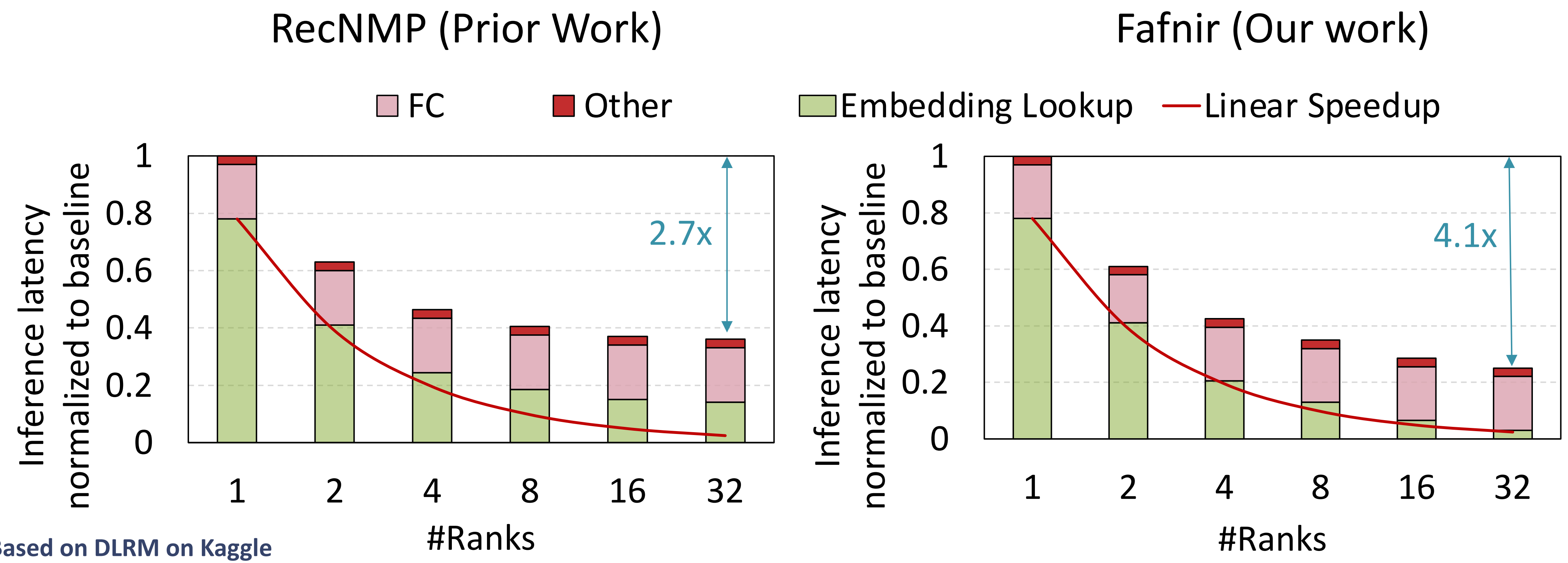
Time to respond to a single query including random accesses to 16 512-byte vectors distributed over 32 ranks.

- Computation of Fafnir is 2.5x faster than prior work
- Memory access of Fafnir is 4.45x faster than TensorDIMM



Evaluation – End-to-End Inference Speedup

The impact of accelerating the embedding lookup on the overall inference time

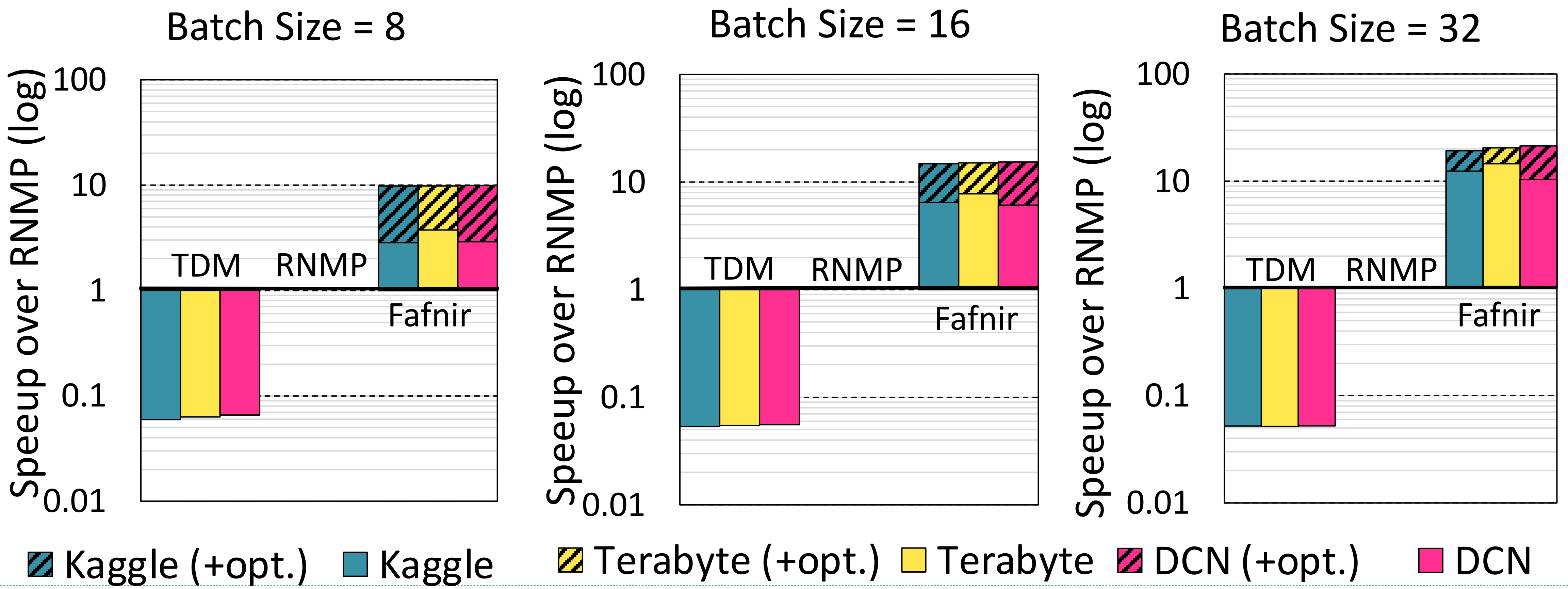


Based on DLRM on Kaggle

Prior work: L.Ke, et al. "Recnmp: Accelerating personalized recommendation with near-memory processing," *ISCA*, 2020.

Evaluation – Scalability

The impact of concurrent batch processing on scalability

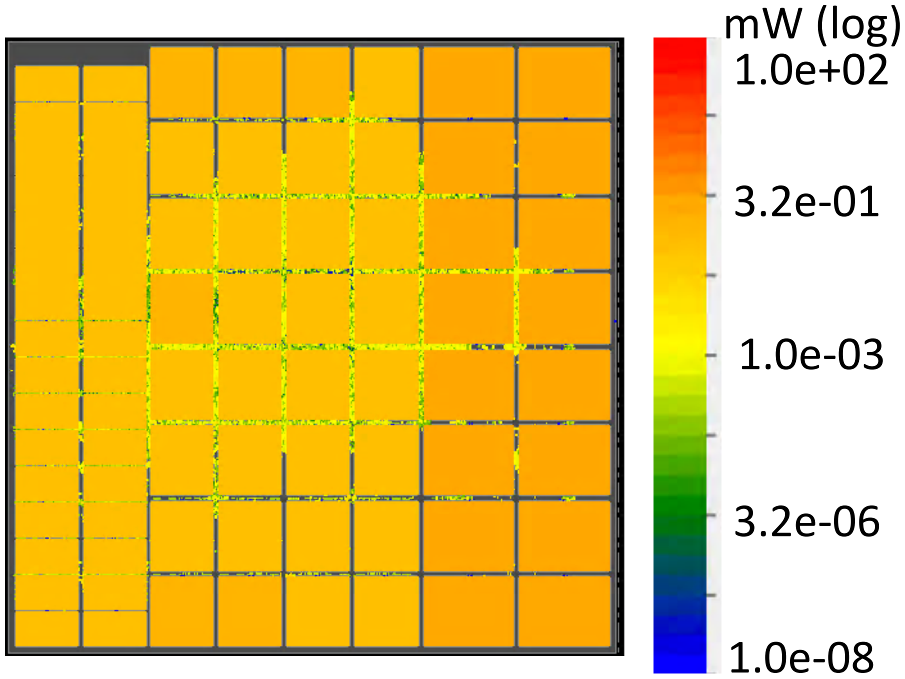


Georgia Tech comparch GTCAD Lab

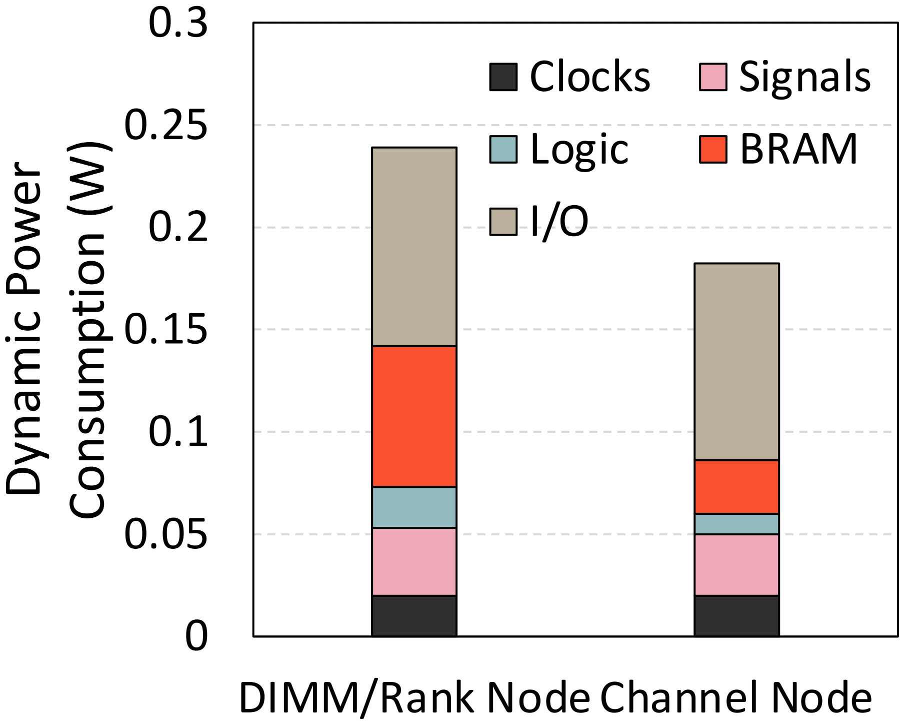
Evaluation –Power Consumption

For a four-channel memory system

- ASIC implementation: 111.64mW
- FPGA implementation: 1.1W



Power distribution of a PE



Georgia Tech comparch GTCAD Lab

Outline

- Sparsity in recommendation system
- Prior near-memory processing approaches and their challenges
- Fafnir: our proposed efficient near-memory intelligent reduction tree
  - Main contributions
  - Architecture and implementation
- Experimental setup
- Performance evaluation
  - Latency
  - End-to-end inference speedup
  - Scalability
  - Power consumption
- **Conclusions**

Conclusions

Fafnir…

- Does not rely on spatial locality
- Minimizes data movement from memory to cores
- Fully utilizes row buffer locality
- Requires fewer connections
- Does not require costly caching mechanisms
- Is application to other application domains (e.g., SpMV)

Backup Slides

Statistics of Workloads for Recommendation Systems

- **The size of embedding vectors:**
  - 64 x 8 bytes = 512 bytes
- **The number of summations:**
  - 64 summations to reduce two vectors
- **An approximate compute intensity:**
  - 0.15 Flops/byte
- **The size of data sets:**
  - Kaggle and Terabyte include 26 tables that we mapped to different ranks utilizing 208 GB
  - DCN includes 400GB data, we report results based on 256GB of it that fits in our 32 ranks
- **Memory size (our configuration):**
  - 4 x 16-GB DDR4 DIMM = 64 GB per a DIMM/Rank Node
  - 4 x 64 GB = 256 GB total for the 32-rank system
- **The number of queries in a batch:**
  - 16 queries per batch, each containing maximum 16 indices

Sparse Matrices from SuiteSparse

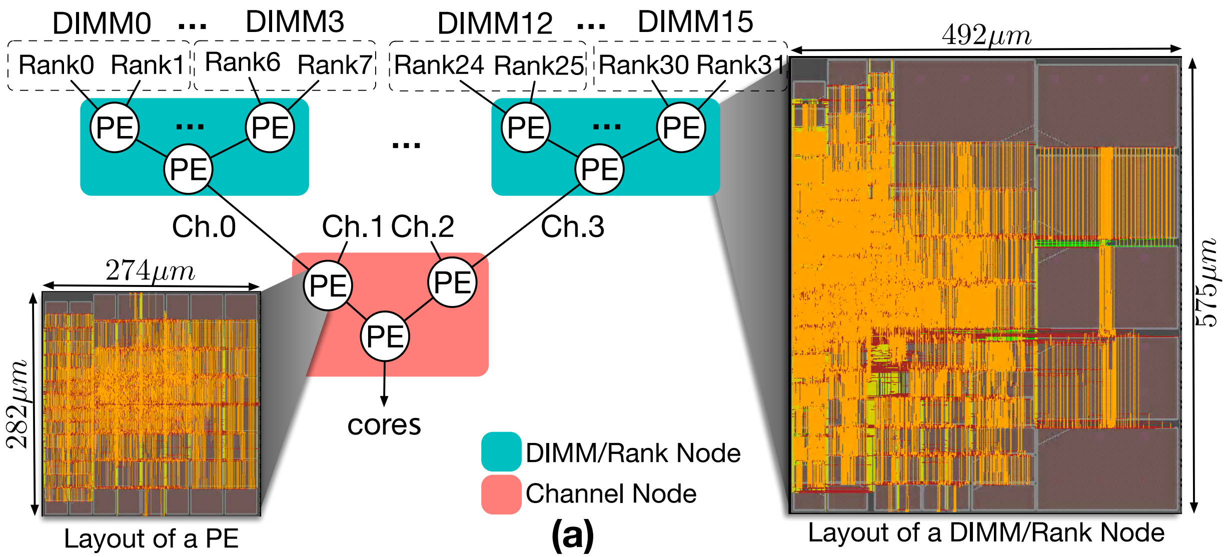
How sparse are the matrices we used for SpMV?

ID	Name	Dim.(M)[1]	Density (%)	Application
RE	N_reactome	0.016	0.025	Biochemical
RI	rail582	0.056	1.2	Linear Prog.
HC	hcircuit	0.1	0.004	Circuit Sim.
2C	2cubes_sphere	0.101	0.016	Electromagnetic
TH	thermomech_dK	0.2	0.006	Thermal
FR	Freescale2	2.9	0.0001	Circuit Sim.
AM	amazon0601	0.4	0.002	Dir. Graph
WG	web-Google	0.91	0.0006	Dir. Graph
RO	roadNet-TX	1.3	0.0001	Unidir. Graph
KR	kron_g500-logn21	2	0.004	Unidir. Multiraph
WI	wikipedia-20070206	3.5	0.0003	Dir. Graph
LJ	soc-LiveJournal1	4.8	0.0002	Dir. Graph

[1] Dim.: dimension or the number of columns/rows of a square matrix.

Mapping Embedding Tables to Memory Addresses

The architecture of Fafnir tree, consisting of DIMM/rank and channel nodes and ASIC designs at 7 nm for a PE and a DIMM/rank node.



(a)

The mapping of embedding tables to memory addresses.

Pointer to Leaves

...	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
row			column			bank				DIMM		R.	CH.		column								
embedding table index										table ID					embedding vector								

(b)

PE configuration

- **The size of PE**
  - The size of input buffers and the number of compute units is defined by the batch size
  - The number of outputs of each PE is limited by the batch size
  - The maximum number of outputs for a PE is min(nm, n+m, B) – n,m: input sizes, B: batch size
  - Each entry of input buffer contains 512B value + 10B header
    - 10B header: 16 queries x 5-bit indices for identifying 32 tables = 16x5/8 = 10B
  - Each PE (at any level of tree) includes 16 compute units
  - Buffer sizes that are sum of all buffers (B: batch size)

<table>
<tr><th rowspan="2">Node</th><th colspan="3">PE buffer (KB)</th><th colspan="3">Node buffer (KB)</th></tr>
<tr><th>B=8</th><th>B=16</th><th>B=32</th><th>B=8</th><th>B=16</th><th>B=32</th></tr>
<tr><td>DIMM/Rank</td><td rowspan="2">4.6</td><td rowspan="2">9.3</td><td rowspan="2">18.5</td><td>32.4</td><td>64.8</td><td>129.5</td></tr>
<tr><td>Channel</td><td>13.9</td><td>27.8</td><td>55.5</td></tr>
</table>

PE Latency

Cycles @200MHz for the components of the compute units of Fafnir based on FPGA implementation:

	Compare	**Parallel paths (reduce or forward)**			
		Reduce (generating the **value**)	Reduce (generating the **header**)		Forward
			Indices generation	Queries generation	
Per item (iteration)	12	3	4	3	16
Batch size = 8/16/32	N/A		32/64/128	29/53/101	N/A

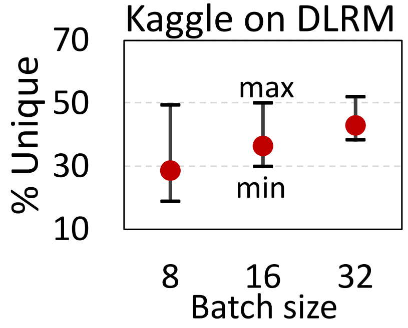
FPGA Resource Utilization

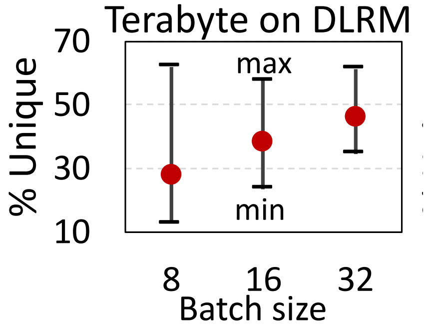
The number of units and the utilization for batch size of 16:

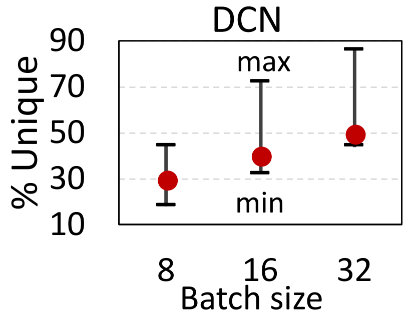
Node	DIMM/Rank Node		Channel Node	
	Units	Utilization (%)	Units	Utilization (%)
LUT	11800	1	7214	0.61
LUTRAM	192	0.03	96	0.02
FF	4646	0.2	3295	0.14
BRAM	68	3.15	26	1.2

Locality in Embedding Accesses

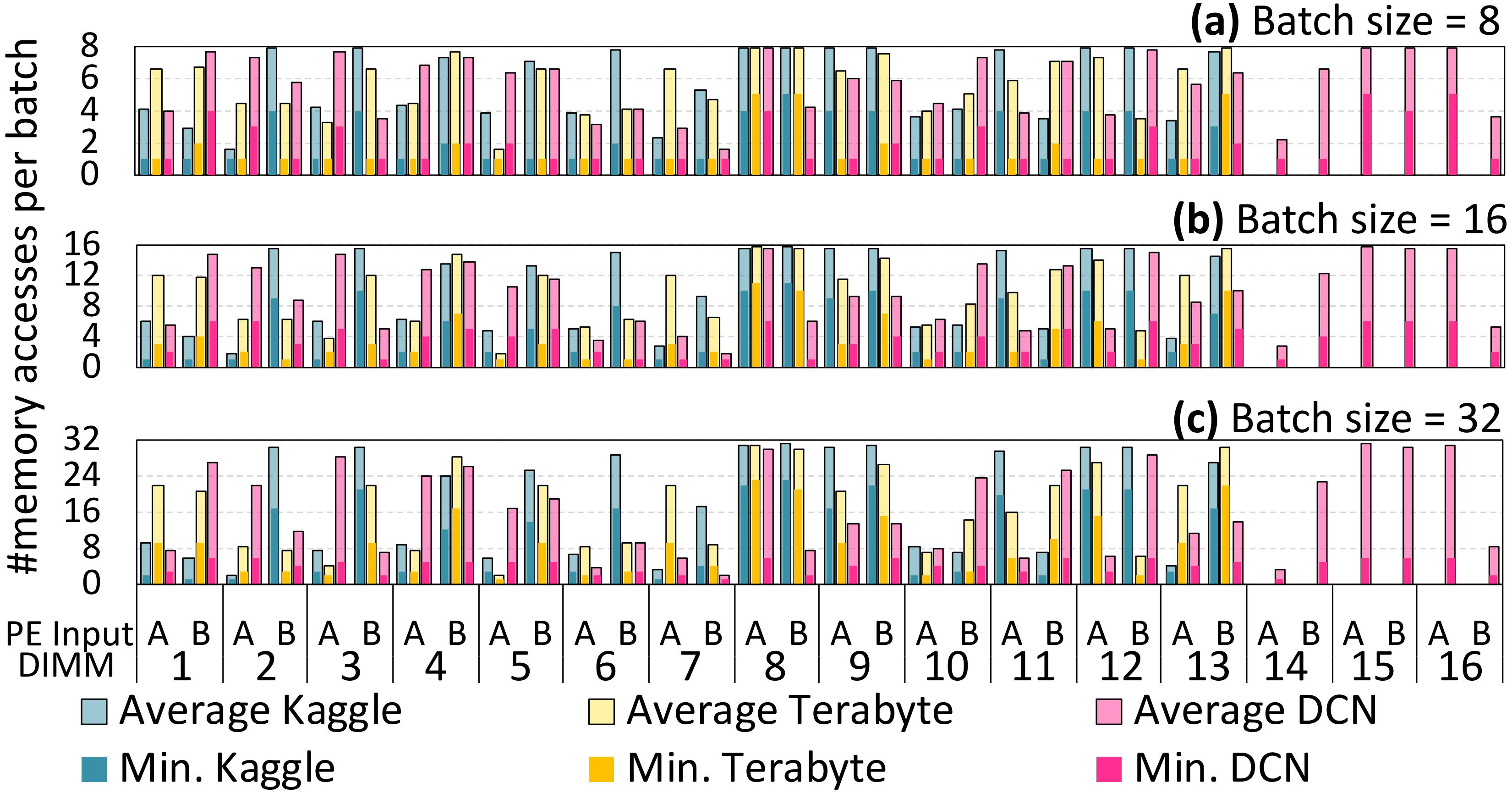
The percentage of unique indices in batches of queries:





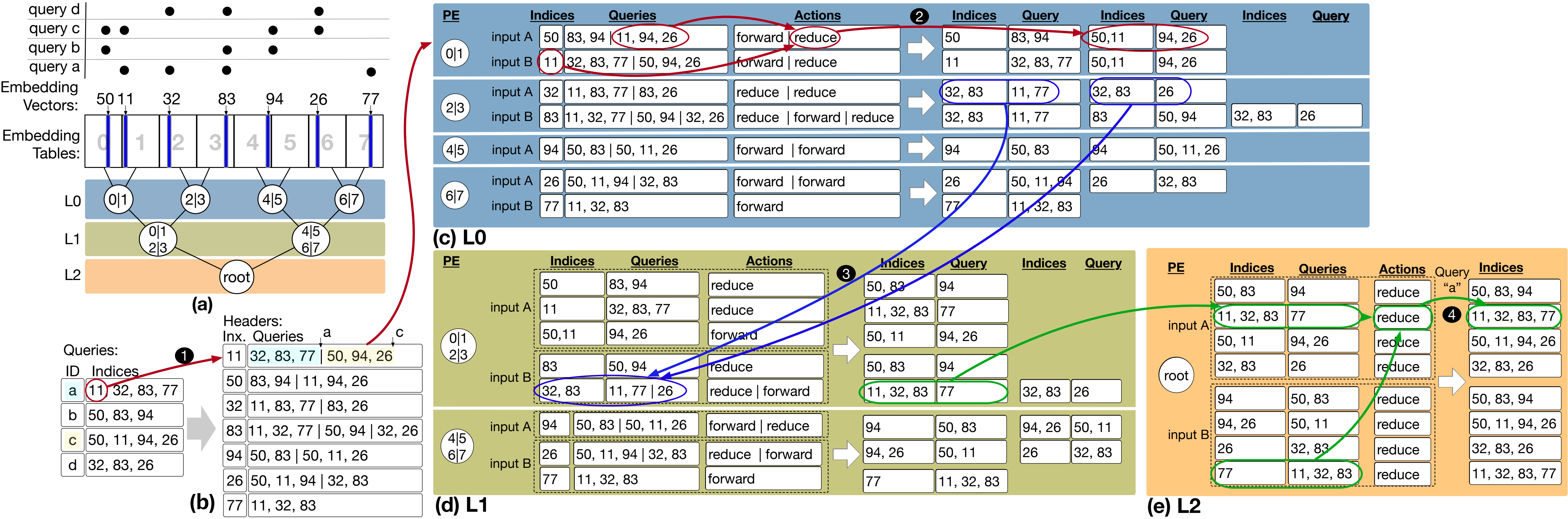


The number of memory accesses at different DIMMs:



Back

Detailed Mechanism of Fafnir



(a) A batch of four queries that access random embedding vectors from eight embedding tables and a three-level Fafnir tree **(b)** Extracting the unique indices of four queries and creating the headers of requests to be forwarded to Fafnir. The steps of processing the four queries through the PEs at three levels of tree: **(c)** L0, **(d)** L1, and **(e)** L2.

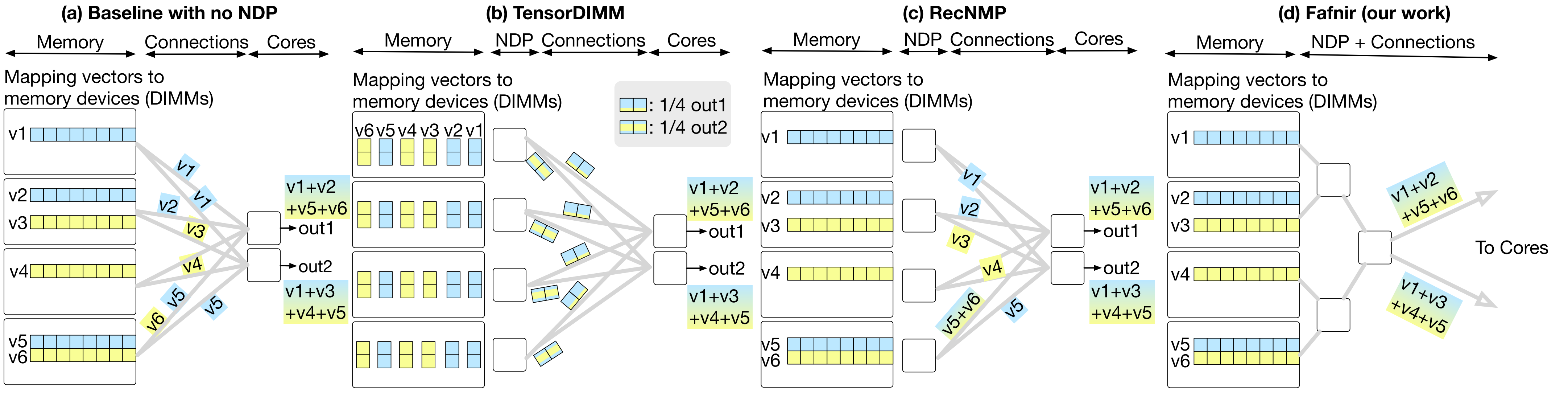
Comparing Prior NMP Solutions and Fafnir

Legend

- ☐ a computation unit for reduction or concatenation

Parameters
- m: # memory devices
- c: number of cores
- v: vector size
- q: # vectors in a query
- n: # queries

Example
- m = 4
- c = n = 2
- v = 8
- q = 4



Section		General	This example	General	This example	General	This example	General	This example
III.A,C	Transferred data (from memory/NDP to cores)	n x q x v	2 x 4 x 8 = 64	n x v	2 x 8 = 16	min: n x v max: n x q x v	6 x 8 = 48 (counting v1 once)	n x v	2 x 8 = 16
III.B	Reading different vectors Reading a vector	parallel ranks sequential columns		random rows parallel ranks		parallel ranks sequential columns		parallel ranks sequential columns	
III.B	Parallel compute at NDP	N/A	N/A	v	8	n x (q-1) x v (in theory)	2 x (4-1) x 8 = 48	n x (q-1) x v	2 x (4-1) x 8 = 48
III.B,C	Scalar operations NDP cores	0 n x (q-1) x v	0 2 x (4-1) x 8 = 48	n x (q-1) x v (m-1) x n concat.	2 x (4-1) x 8 = 48 (4-1) x 2 = 6 concat.	min:0 / max: n x (q-1) x v min: 0 / max: n x (q-1) x v	1 x 8 = 8 5 x 8 = 40	n x (q-1) x v 0	2 x (4-1) x 8 = 48 0
III.C	DIMM-level parallelism	No		No		No		Yes	
III.D	#Connections (excluding connections to memory)	c x m	2 x 4 = 8	c x m	2 x 4 = 8	c x m	2 x 4 = 8	(2m - 2) + c	(2 x 4 - 2) + 2 = 8

Sparsity in recommendation systems

- **Compression of embedding vectors**
  - Particular embedding vector's dimension can scale with its query frequency[1]
- **Compression of embedding tables**
  - Hashing techniques or complementary partitions are used to reduce embedding table size
- **Distribution of random accesses**
  - In the 4-channel system, the probability of having a query with indices on the same channel: ~25%
- **Level of sparsity in the accesses to embedding tables**

DLRM	number of embedding tables	embedding size	min number of indices	max number of indices	batch size	Density of accesses (max)	Density of accesses (min)	**Sparsity (min)**	**Sparsity (max)**
RM1-small	8	1000000	20	80	256	0.256%	0.064%	99.744%	99.936%
RM1-large	12	1000000	20	80	256	0.171%	0.043%	99.829%	99.957%
RM2-small	24	1000000	20	80	256	0.085%	0.021%	99.915%	99.979%
RM2-large	64	1000000	20	80	256	0.032%	0.008%	99.968%	99.992%

[1] A.A. Ginart, et al. "Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems," arXiv:1909.11810v3
[2] H.M. Shi, et al. "Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems," arXiv:1909.02107v2

SpMV vs. Embedding Lookup

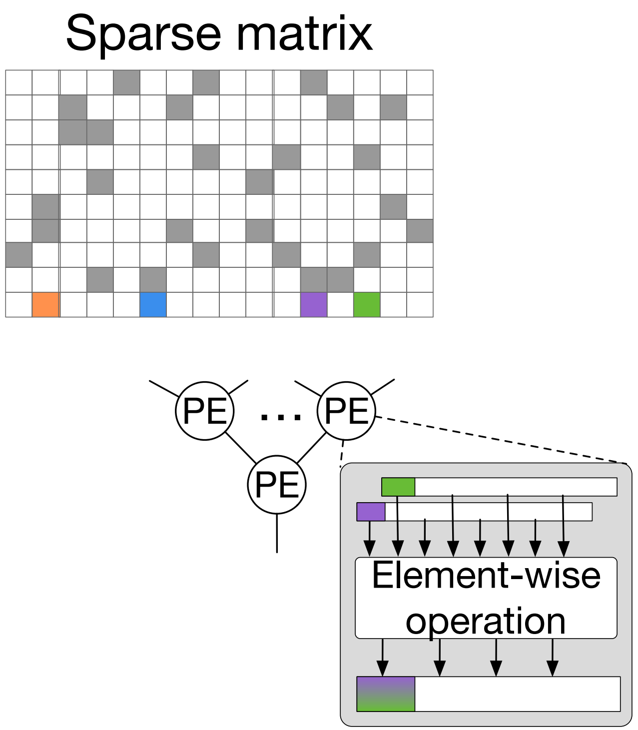
For SpMV, we do not know where the non-zero values of the sparse matrix are located:

- the indices of the elements to be reduced are *unknown* -- indices themselves are read from memory.
- we stream both data and indices through the tree.

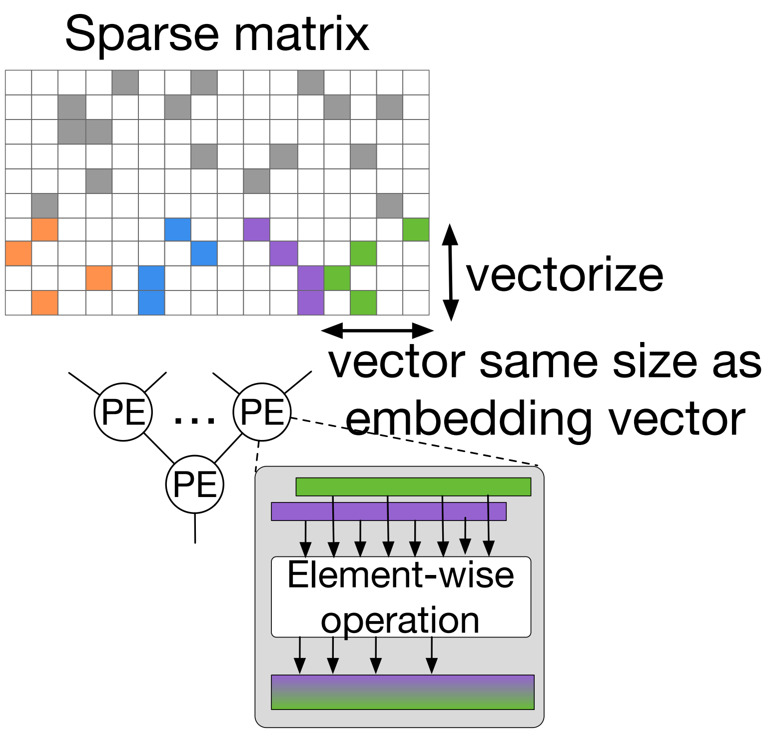
	SpMV	**Embedding lookup**
Indices	Unknown	Known
The type of memory accesses	Stream data and indices	Stream data only
The function of Leaf PEs	Multiplication with the vector operand	Skip multiplications

SpMV using Fafnir -- vectorization

No vectorization (compute units are underutilized):

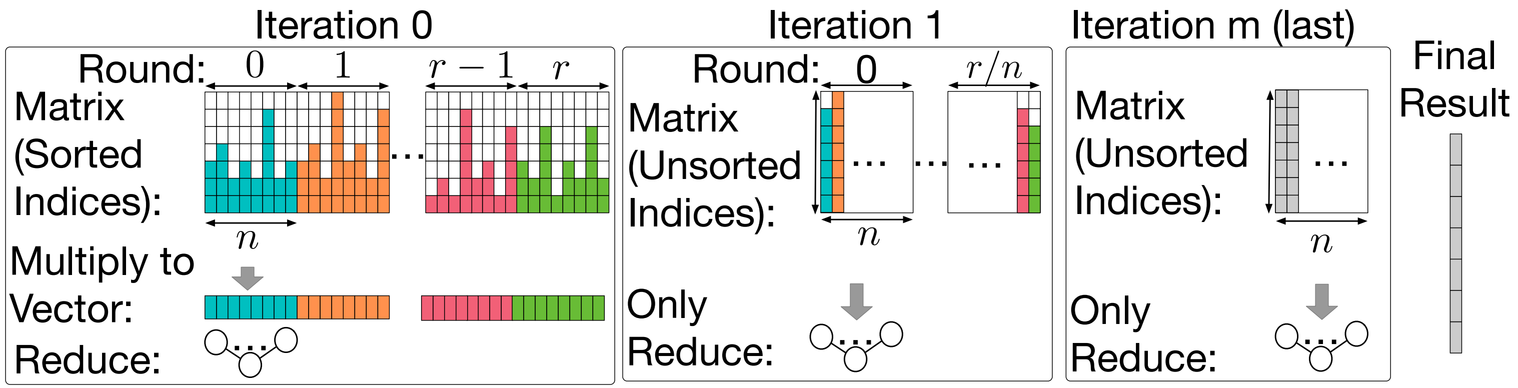


With vectorization:

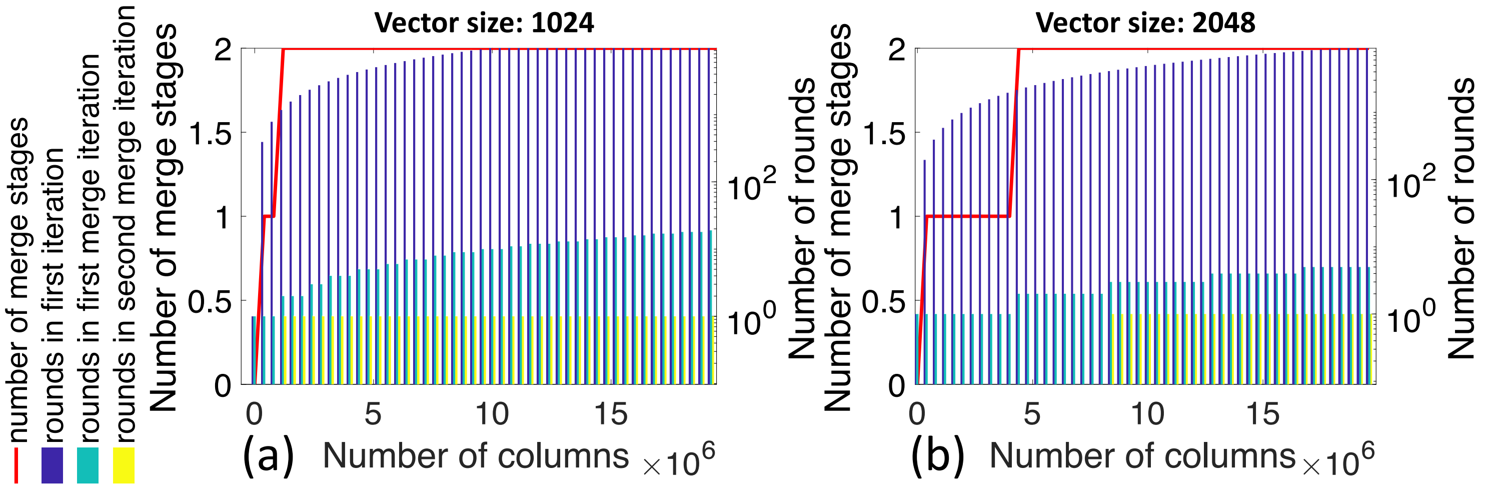


SpMV using Fafnir – compression and iterations

We use list-of-list (LIL) compression format. If only n columns of the matrix fit in the Fafnir, we need to perform SpMV in rounds and iterations:



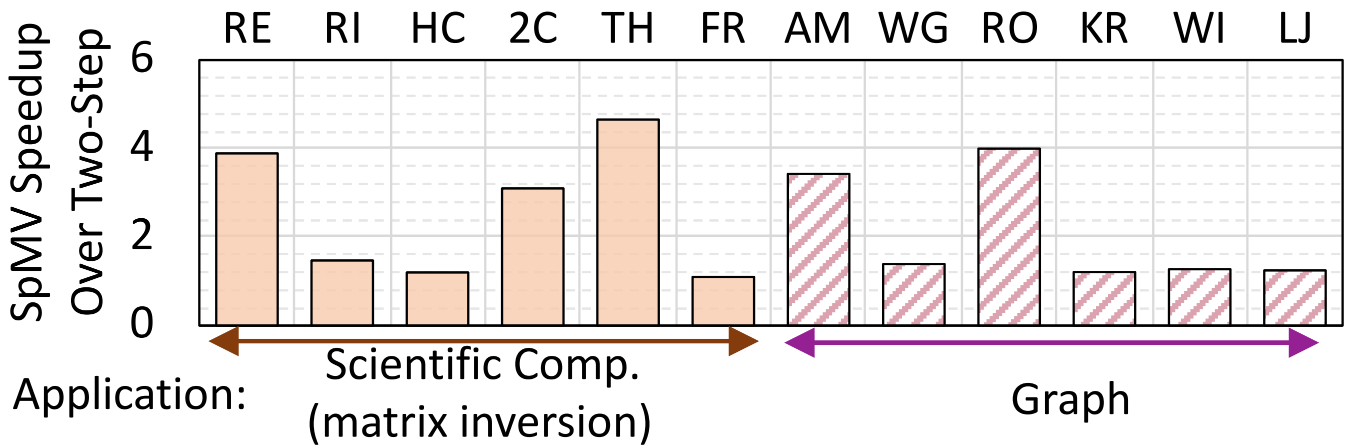
The number of required iterations and rounds per iterations for two vector sizes when the number of columns increases up to 20 million:



Georgia Tech comparch GTCAD Lab

Results of SpMV using Fafnir

- Fafnir performs the **first step** more quickly.
  - Unlike the Two-Step algorithm, Fafnir does not rely on decompression mechanisms and is able to apply SpMV on data as it is streamed from memory.
  - Instead of a chain of adders connected to multipliers, Fafnir uses the tree for the reduction.
- The Two-Step algorithm performs the **merge steps** more quickly.
  - For **smaller** matrices, Fafnir performs more quickly than larger ones.



Two-Step: F. Sadi, et al. "Efficient spmv operation for large and highly sparse matrices using scalable multi-way merge parallelization," in *MICRO*, 2019.