Towards Throughput-oriented Sparse Matrix Vector Multiplication on a Processing-in-Memory System

Lukas Zink

Juan Gómez Luna Christina Giannoula Onur Mutlu

ETH zürich

SAFARI

Executive Summary

Motivation: Sparse Matrix Vector Multiplication is a widely used kernel in sparse neural networks, graphs, physical/chemical optimizations

Problem: on commodity systems, random accesses of the input vector
- Makes SpMV kernel fundamentally memory-bound
- Leads to low hardware utilization and inefficient computation

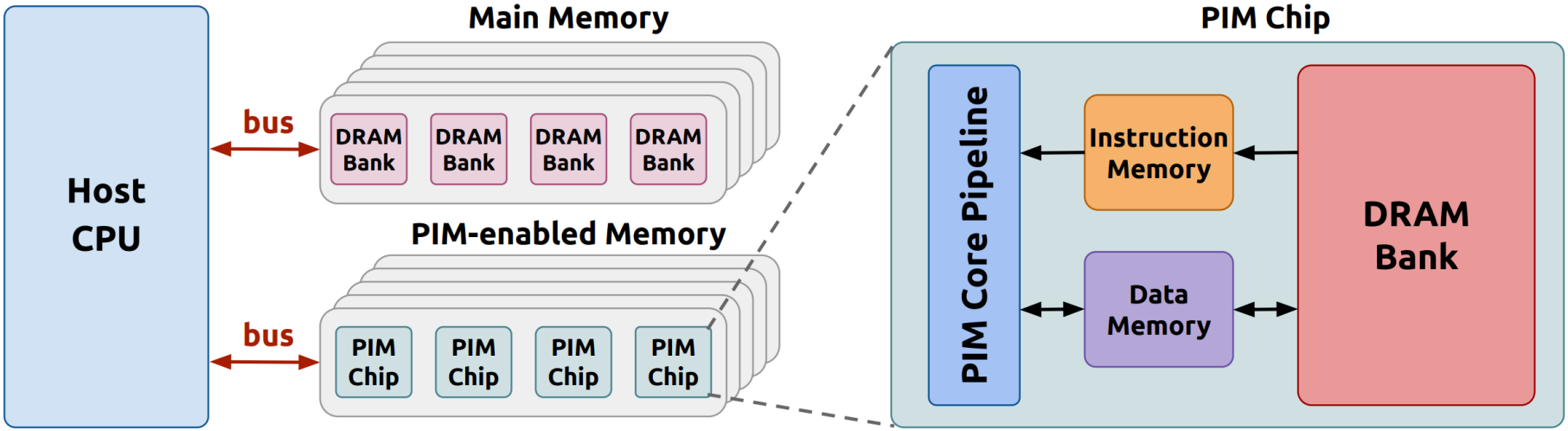
Goal: develop a high throughput, high efficiency and low latency SpMV kernel implementation

Key Observations:
- Processing in memory can alleviate data movement
- Single SpMV workloads do not scale to a large system size

Key idea: Divide PIM system resources to perform many independent SpMVs in parallel utilizing its numerous DPU cores and memory capacity

Results: 93x average throughput improvement, 73% compute utilization

State of-the Art PIM System



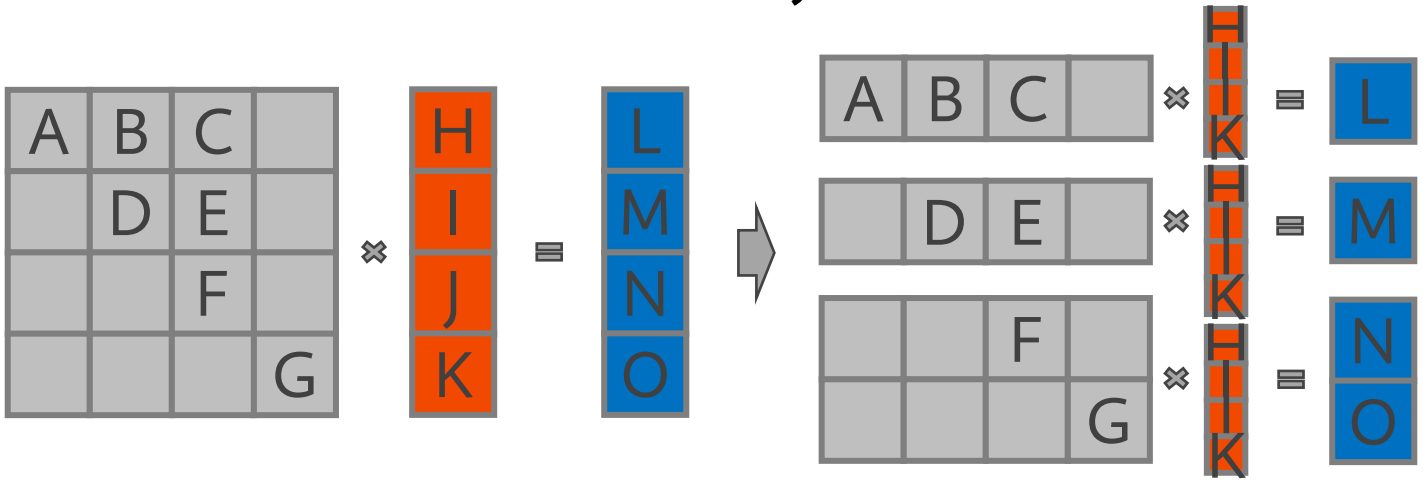
UPMEM PIM architecture with

- general-purpose processing cores called *DPUs*
- *128 DPUs per module*
- 64-MB DRAM bank (*MRAM*)
- 64-KB scratchpad (*WRAM*)

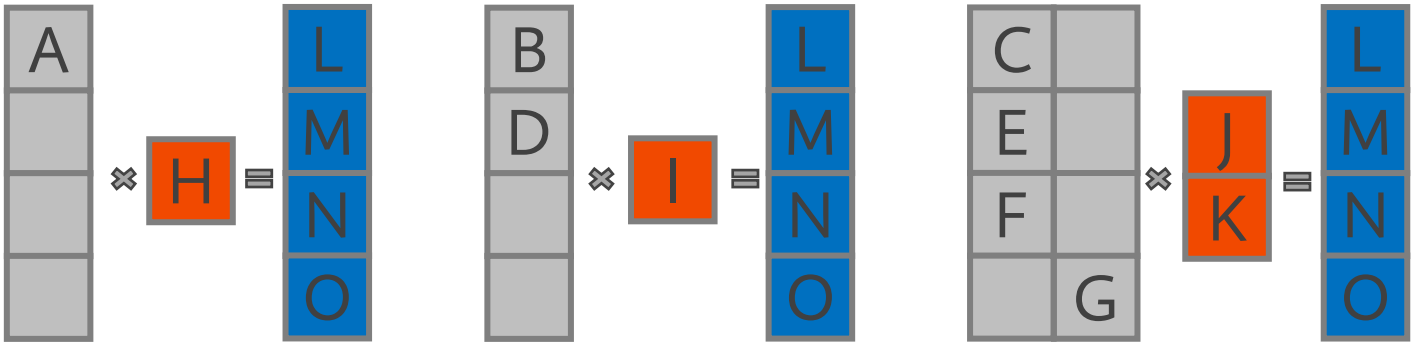
Matrix Partitioning Techniques

- Split workload onto a distributed system

Horizontal Partitioning



Vertical Partitioning

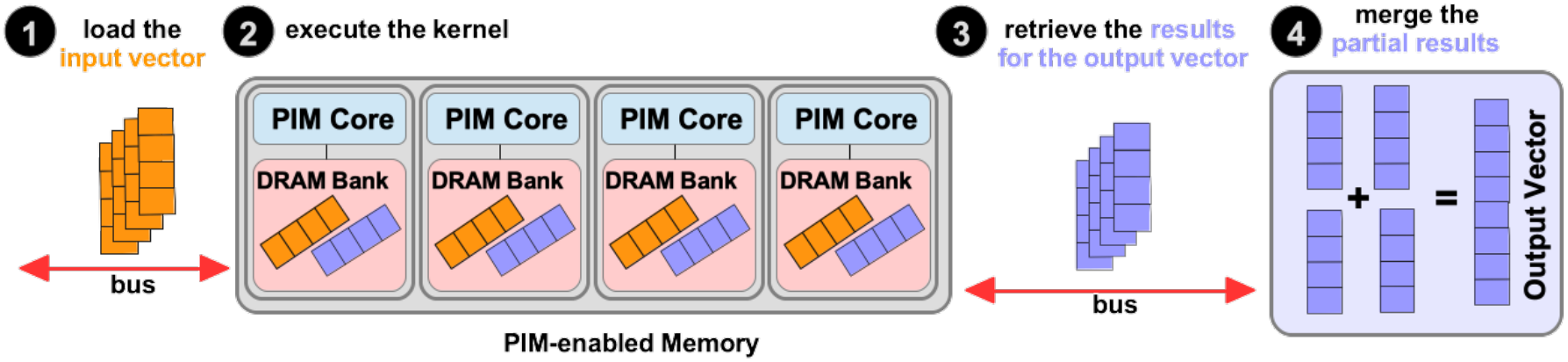


- Combination of both: 2D partitioning

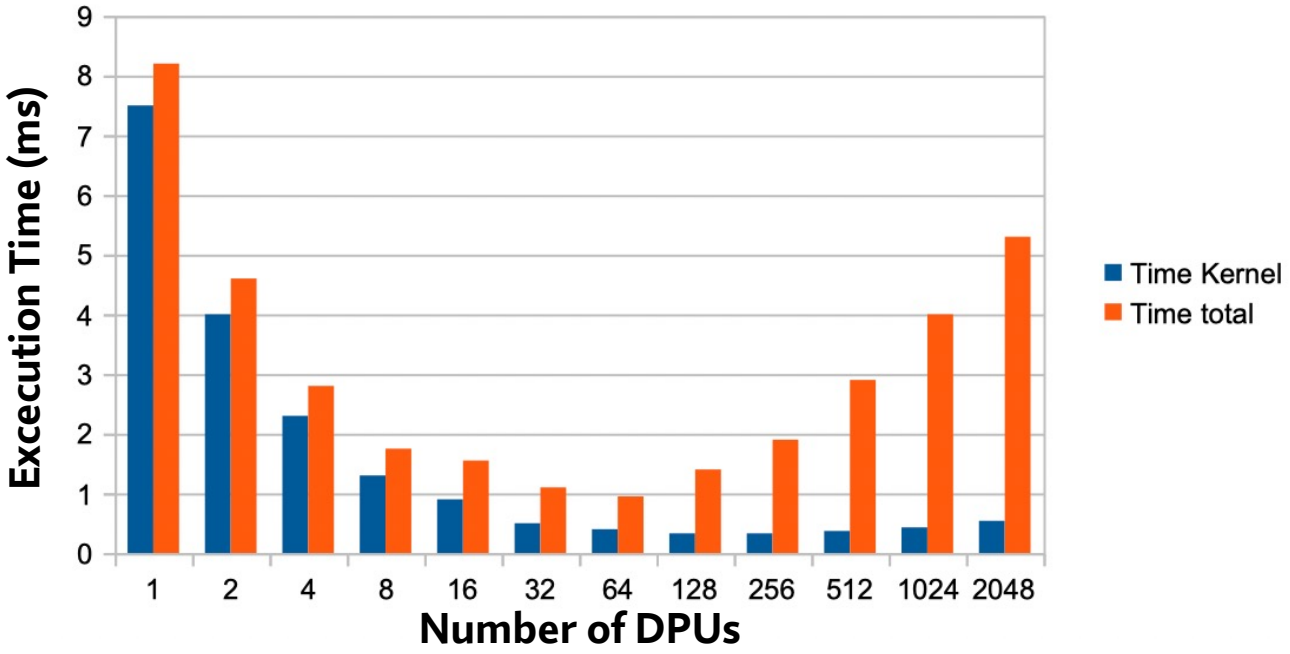
SpMV on a PIM system

Matrix and/or vector are partitioned according to a partitioning technique before the data transfer can commence

Transfers and Kernel are executed in parallel



Scaling of a single SpMV on UPMEM



- Less DPUs: execution time dominated by computation
- More DPUs: execution time dominated by transfer of vectors

Problem: single SpMV Implementation **does not scale** beyond **64 DPUs**

Key Takeaways from Benchmarking

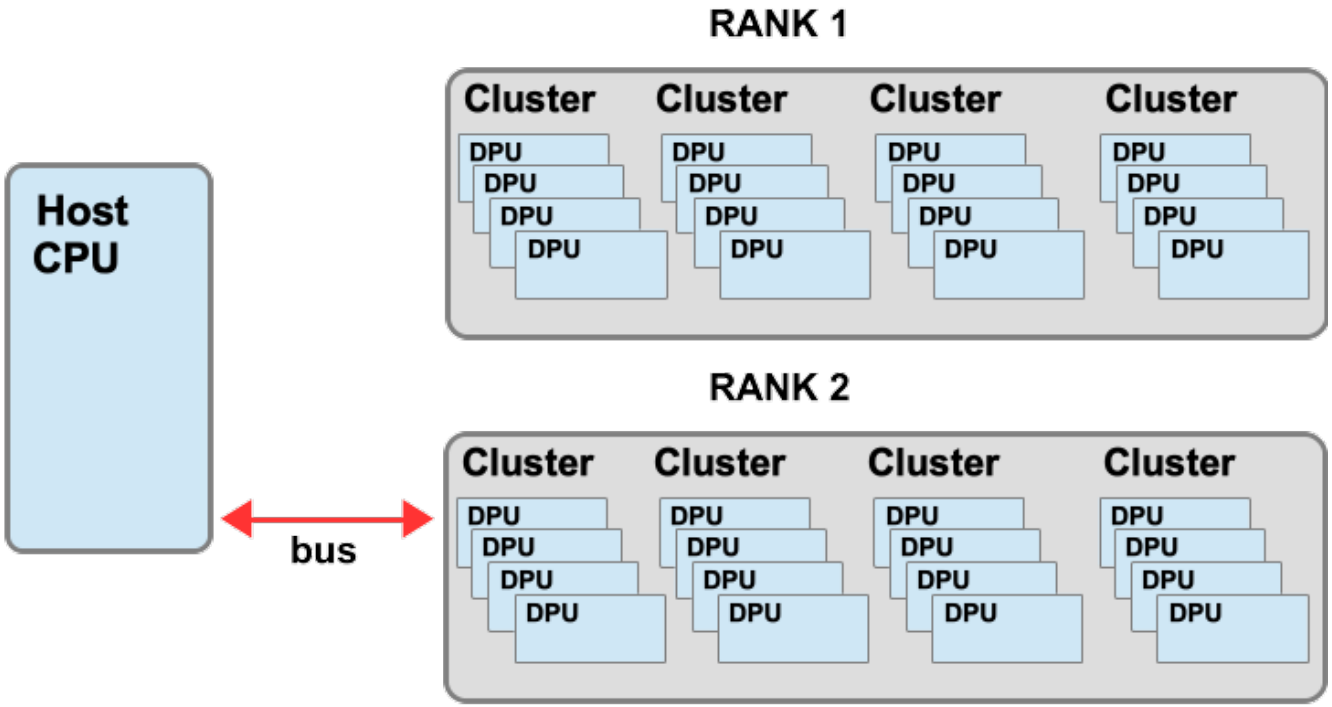
1. **Balancing** of non-zero **matrix elements** across **DPUs** crucial for low kernel execution time

2. **Minimize** total amount of **data transferred** to PIM memory to lower the overall execution time

3. The **CSR-NNZ** and **CSR-BLOCK** kernels achieve the **highest performance** for SpMVs with 8 to 64 DPUs

4. Automatic **selection/adaptation** of **kernels** based on **matrix properties** significantly benefits performance

Mechanism



Perform many SpMVs in parallel on a PIM system to:

- support concurrent tasks, users and SpMM workloads
- provide sufficient load to each DPU, utilize data locality

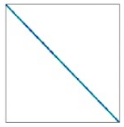
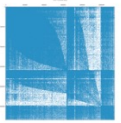
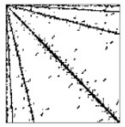
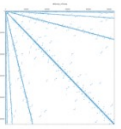
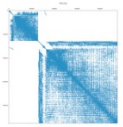
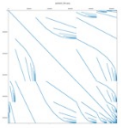
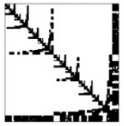
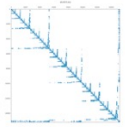
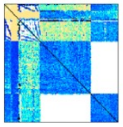
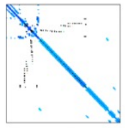
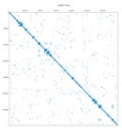
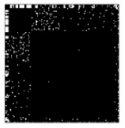
Evaluation Methodology

15 matrices from the Texas A&M Sparse Matrix Collection

Vastly different properties, levels of sparsity, and pattern and applications

UPMEM System Configuration

- 20 PIM modules
- 2496 DPUs @ 400 MHz

af shell1 	Com-Youtube 
delaunay_n13 	delaunay_n19 
ldoor 	parabolic_fem 
pkustk08 	pkustk14 
poisson3Db 	raefsky4 
roadNet-TX 	wing_nodal 

Throughput Improvement

Comparison against the single SpMV version:

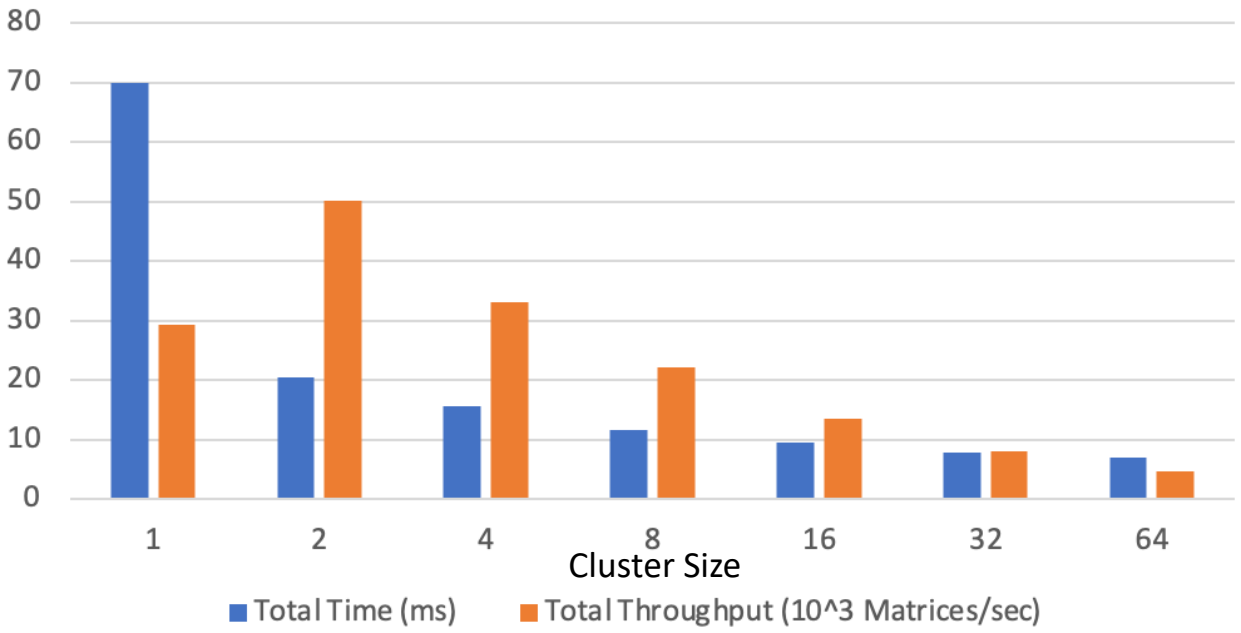
- 57x to 259x throughput improvement
- 93x average improvement
- 3x latency increase

Largely depends on the number of non-zero entries of the matrix

Throughput vs. Latency

Varying cluster sizes changes the performance of the SpMV.

- Decreasing the cluster size allows for **more throughput**
- Increasing the cluster size can **improve latency**



Large cluster size: insufficient load per DPU
Small cluster size: many input vector transfers

Comparison against CPU/GPU

Kernel Throughput:

GPU: NVIDIA A100@1095, cuSparse cuda 11.2:	182 Gflops
CPU: Intel Xeon 8280 Intel MKL:	5.4 Gflops
UPMEM 2496 DPU 400mhz s 32-bit float:	3.3 Gflops
UPMEM 2496 DPU 400mhz 8-bit int:	**10.8 Gops**

Utilization of computing resources:

GPU: NVIDIA A100@1095, cuSparse cuda 11.2:	0.9% (**0.002% tensor**)
CPU: Intel Xeon 8280 Intel MKL:	**16.35%**
UPMEM 2496 DPU 400mhz kernel:	**58-79%**
UPMEM 2496 DPU 400mhz total:	32%

Limitations and Conclusion

Current PIM system limitations:

- Arithmetic throughput of DPUs
- Emulation of larger data type operations
- Narrow memory bus

Conclusion:

- **Highly efficient** due to low data movement
- Throughput rivals CPU-based implementation
- DPU **utilization** close to arithmetic throughput
- SpMV kernel **not memory bound** on this hardware

Towards Throughput-oriented Sparse Matrix Vector Multiplication on a Processing-in-Memory System

Lukas Zink

Juan Gómez Luna Christina Giannoula Onur Mutlu

ETH zürich

SAFARI