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

Lukas Zink  Juan Gómez Luna  Christina Giannoula  Onur Mutlu

**Motivation**

Use Cases: Sparse Matrix Vector Multiplication (SpMV) is a fundamental kernel for sparse neural networks, semiconductor design, chemical/physical optimizations and graph applications.

Problem: Random accesses of the input vector make the kernel fundamentally memory bound on commodity systems. The result is a low hardware utilization and inefficient computation.

**Our Goal**

Perform many SpMVs in parallel on a PIM system to support concurrent tasks, users and SpMM workloads.

Achieving a high throughput, high efficiency and low latency implementation.

Taking advantage of the data locality, DRAM storage capacity and DPU Cores of the PIM System.

Analyzing transfer and kernel performance regarding matrix properties and partitioning formats and selecting the optimal kernel for a given SpMV.

**Cluster Mechanism**

The mechanism separates the entire system into clusters of multiple DPUs where each cluster then computes a single SpMV.

**State-of-the-Art PIM System**

UPMEM PIM architecture with general-purpose processing cores called DPUs:
- 64 DPUs/Rank, 2 Ranks/Module
- 16 threads used
- 32-bit integer arithmetic
- 64-MB DRAM bank (MRAM)
- 4 KB scratchpad (WRAM)

**Matrix Partitioning Techniques**

Horizontal Partitioning

```
A B C D E F G
A B C D E F G
```

Vertical Partitioning

```
A B D C F G
A B D C F G
```

**Testing: Impact of Block Pattern on Performance**

If a Matrix exhibits a block-pattern, a blocked partitioning format consistently performs better.

For matrices without a block-pattern, traditional partitioning formats outperform.

Testing: Prior knowledge about the matrix exhibiting a block pattern leads to a 15 to 32% lower execution time on average.

The matrix analyzer tool of the library can classify the block pattern property with an accuracy of 92% on the matrix test set.

Recommendation: If matrix properties are known in advance, use according format. Analysing properties prior to SpMV only benefits if matrix is reused.

UPMEM system was configured with 20 PIM modules, Texas A&M sparse matrix, with vastly different properties. Testing was performed on a set of 12 matrices from the KB scratchpad.

**Evaluation**

**Testing Methodology**

Testing was performed on a set of 12 matrices from the Texas A&M sparse matrix, with vastly different properties. The UPMEM system was configured with 20 PIM modules, resulting in 2496 DPUs running at 400mhz.

**Throughput Improvement**

Comparison against the latency-oriented SpMV implementation by SparseP using the COO-NZK Kernel - throughput increases from 572 to 2595 depending on matrix size - average throughput increase of 53x - latency increase by 3x.

**System Scalability**

When keeping the workload per rank constant and scaling up the system it can be observed that total execution time increases by 40% when going from 1 to 32 ranks. This can be attributed to the narrow memory bus.

**Conclusion**

In conclusion, the throughput-oriented implementation succeeds at parallel computation of multiple independent SpMVs with a performance that rivals a state-of-the-art CPU-based system. The PIM system executes the selected kernels at a high utilization and efficiency. Limitations of the implementation are the arithmetic throughput limit of the DPUs cores, narrow memory bus and emulated operations for larger data types. These can largely be attributed to the early stage of PIM technology development and are not fundamental limits of the technology itself.

Acknowledgement: We acknowledge support from the UPMEM Corporation for enabling access to their infrastructure.