Efficient and High-Performance Sparse Matrix-Vector Multiplication on a Many-Core Array

Peiyao Shi, Aaron Stillmaker, Bevan Baas

15th IEEE MCSoC 2022

University of California, Davis
VLSI Computation Laboratory
Dec 22, 2022

Outline

- **Introduction and Motivation**
- Approach and Implementations
  - Platform: KiloCore
  - SpMV Kernels Mapping
- Simulation Results
- Performance Comparison
- Summary

Sparse matrix-vector multiplication (SpMV)

Matrix-vector multiplication of the form $\boldsymbol{y} = \boldsymbol{Ax}$ is a basic tool of linear algebra.

Matrices that contain mostly zero values are called **sparse**, distinct from matrices where most of the values are nonzero, called **dense**.

The input matrix A and input vector $\boldsymbol{x}$ can be either dense or **sparse**.

The output vector $\boldsymbol{y}$ is in general **dense**.

Sparse Matrix Compressed Formats

Due to their large fraction of zero elements, sparse matrices are often stored in a compact format, i.e., only non-zero elements are preserved.

CSR is the most popular format used in **SpMV** operations due to its space efficiency and fast access latency.

It consists of three arrays: **ptr, indices, and data**, which store

- the cumulative number of non-zero elements up to each row
- the column indices of the non-zero elements
- the values of non-zero elements

Challenges

In addition to the matrix storage format, how to process the multiplication with a dense vector x in parallel is another challenge, which includes two main points:

- **balancing the workload among the distinct cores/threads**
- **accessing the matrix entries and the vector values efficiently**

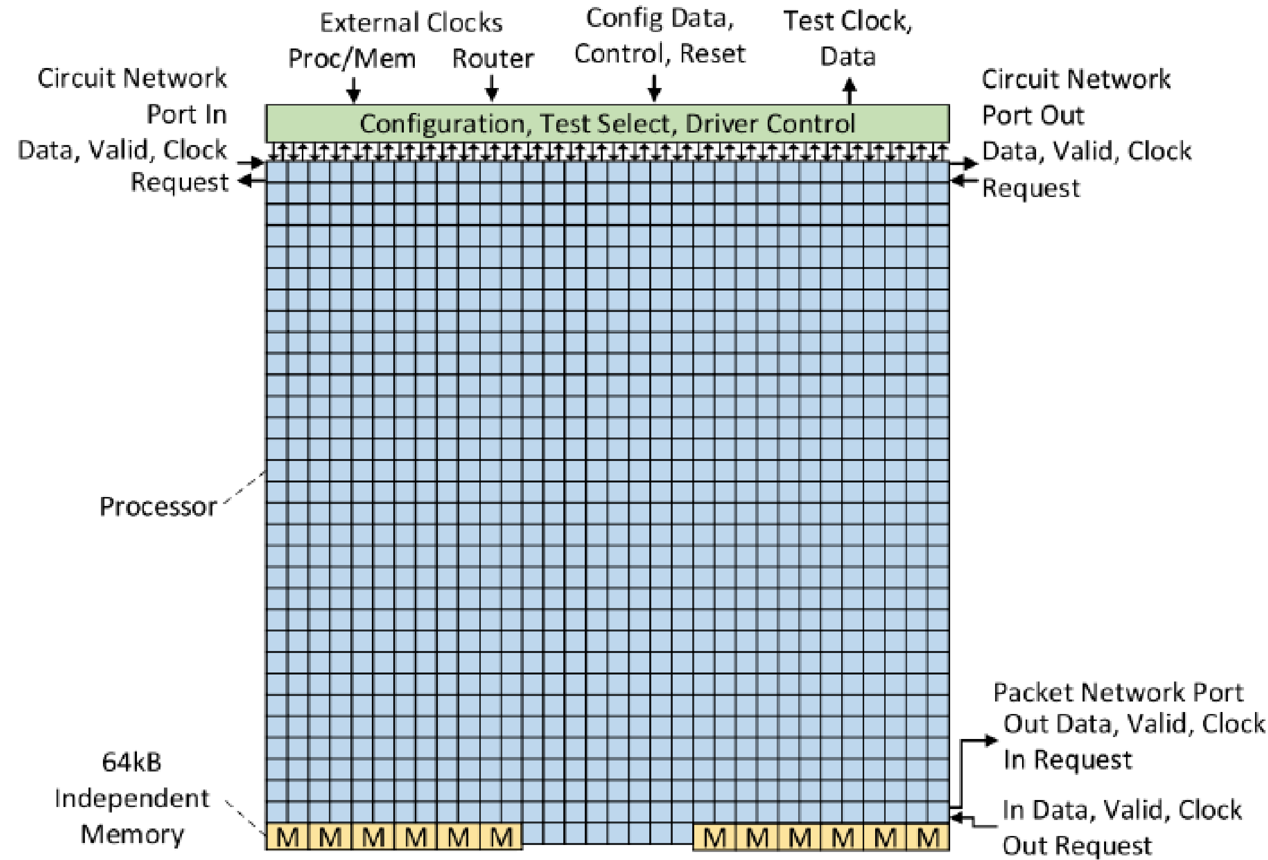
Outline

- Introduction and Motivation
- **Approach and Implementations**
  - Platform: KiloCore
  - SpMV Kernels Mapping
- Simulation Results
- Performance Comparison
- Summary

KiloCore

Fabricated 1000-processor

- 1.78 GHz 32 nm PD-SOI CMOS.
- Only 512 bytes of data storage each per processor
- 12 independent 64 KB shared memories
- Complementary circuit-switched and packet switched networks.
- Occupy only 0.055 μm^2 of chip area each.

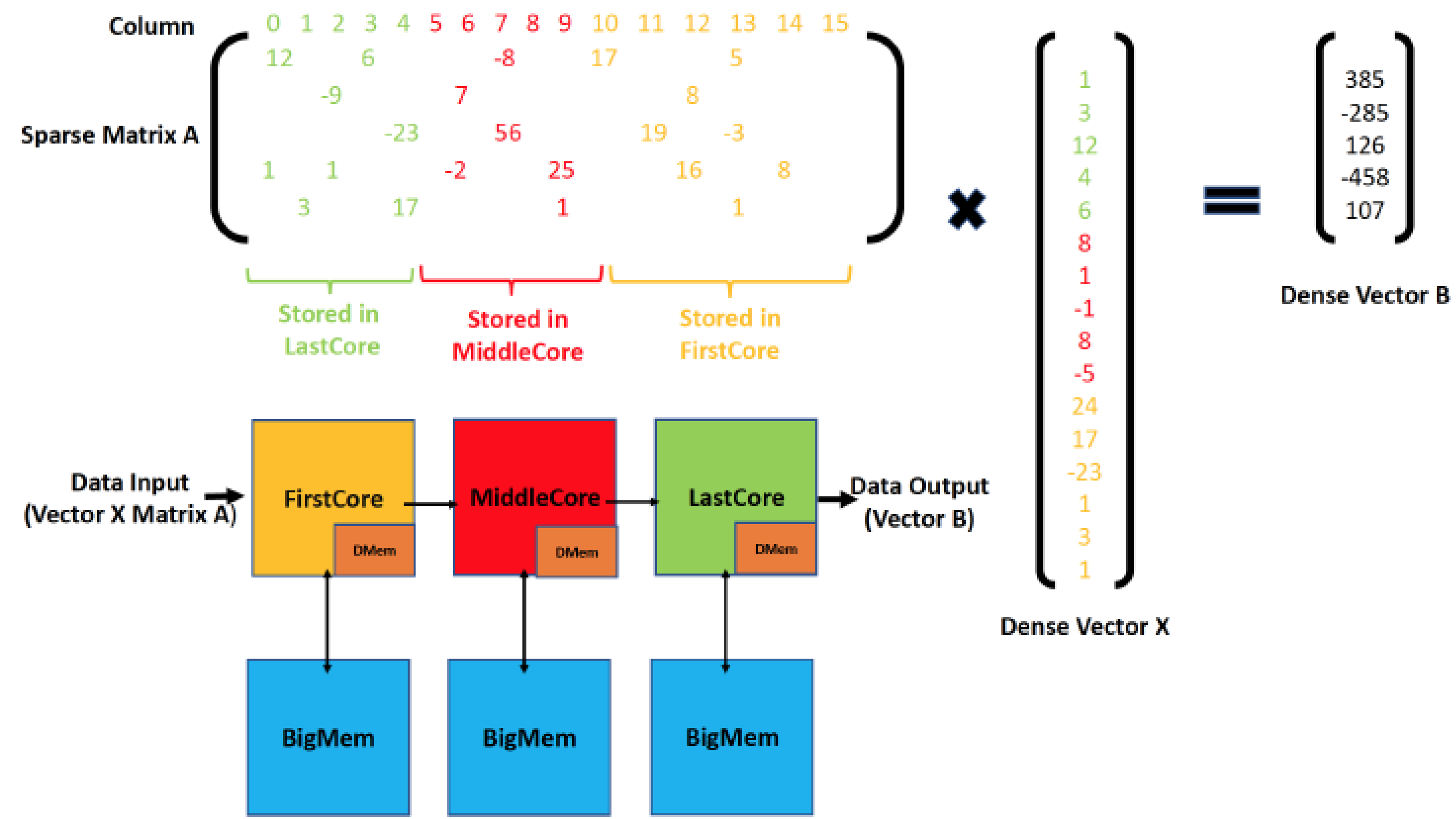


Approach and Implementations

This work presents high performance scalable **CSR SpMV** applications, which were implemented on a fine-grained manycore array of simple high-performance low-power independent programmable processors.

Each **SpMV** implementation consists a set of small modular program kernels operating on each core, making them very easily scalable to different array sizes.

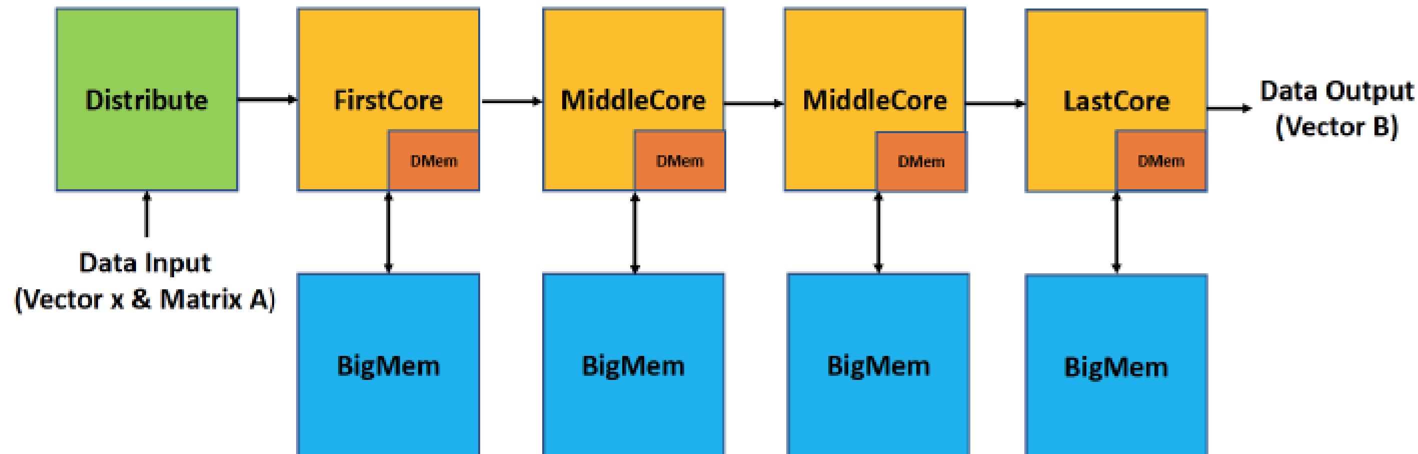
Sparse Matrix-Vector Multiplication Kernels



- Vector X is stored in the core array
- Flows into the Matrix A data
- The data memory on each core is filled first
- All other items are loaded into the large shared memory connected to the core

SpMV Architecture Mapping

BigMemSnake:



Each processor takes in sparse data points, including column indices, matrix values, and tokens, and then determines whether to multiply the matrix term with the corresponding vector item

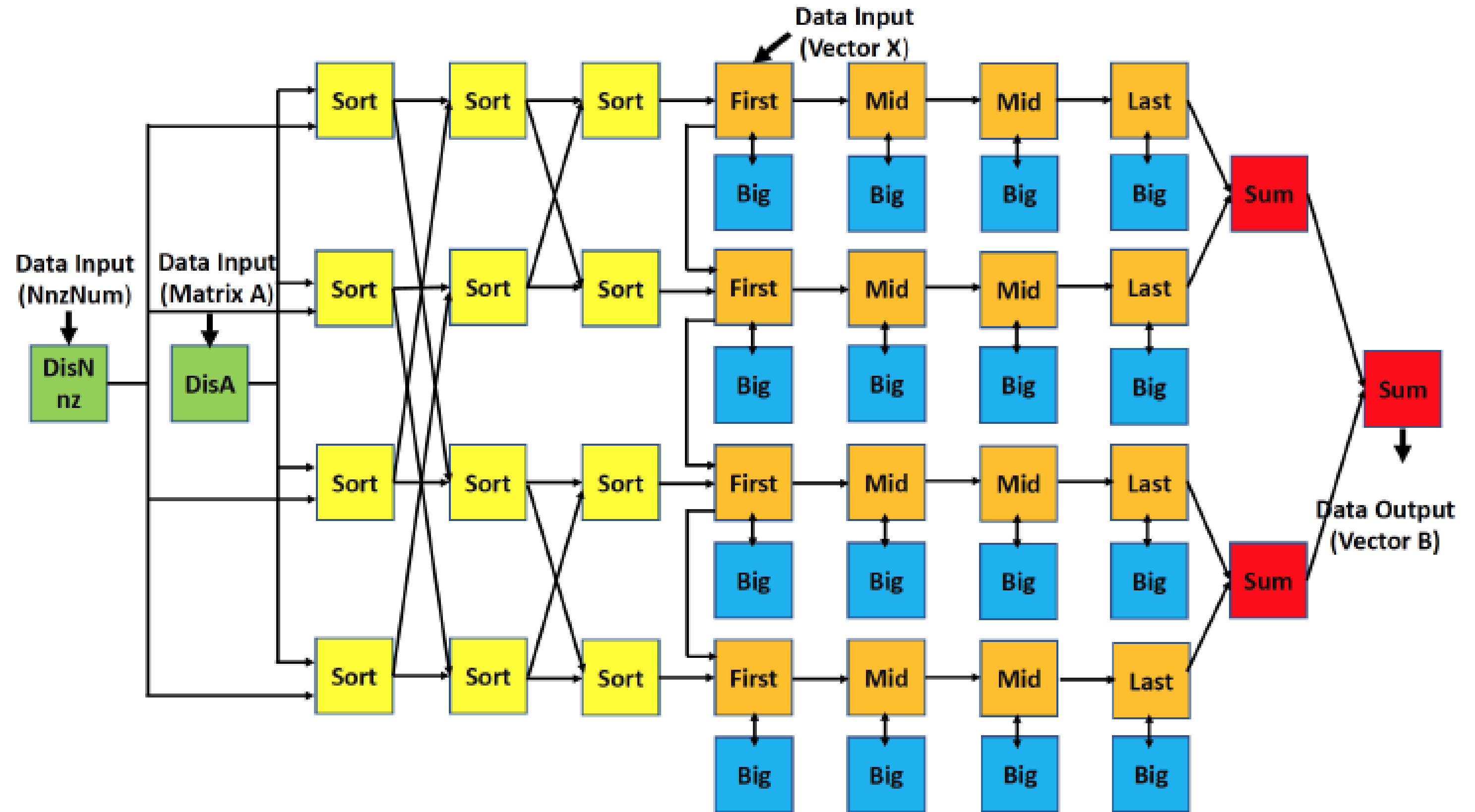
SpMV Architecture Mapping

BigMemSubPara:

Phase one, NNZ per row and matrix data are assigned to the sorting network.

Phase two, the sorting network then sorts the data based on the LSB and second LSB of the column index of each sparse data point.

Phase three, data is transmitted through a set of processing cores connected in 4x4 blocks.

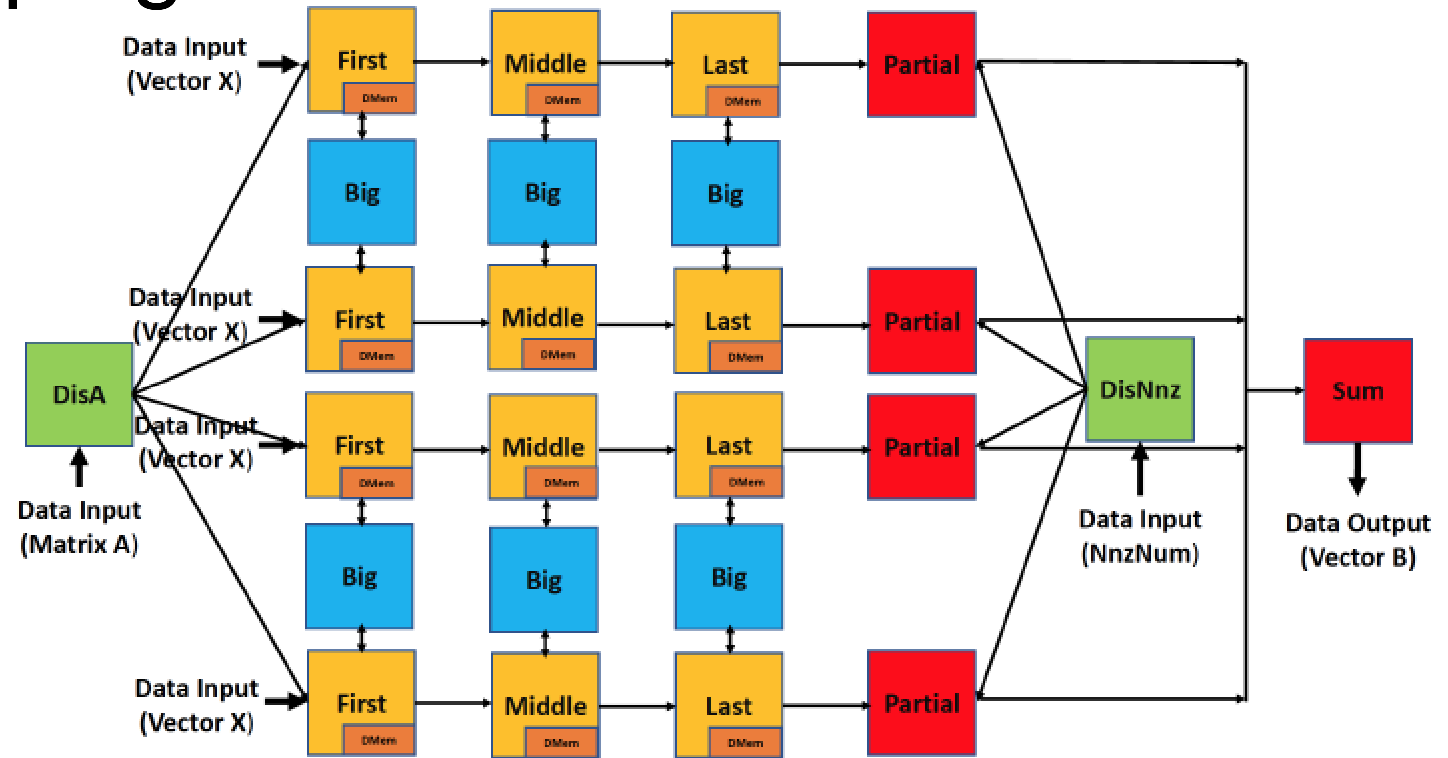


SpMV Architecture Mapping

BigMemPara:
The matrix distribution kernel assigns matrix data.

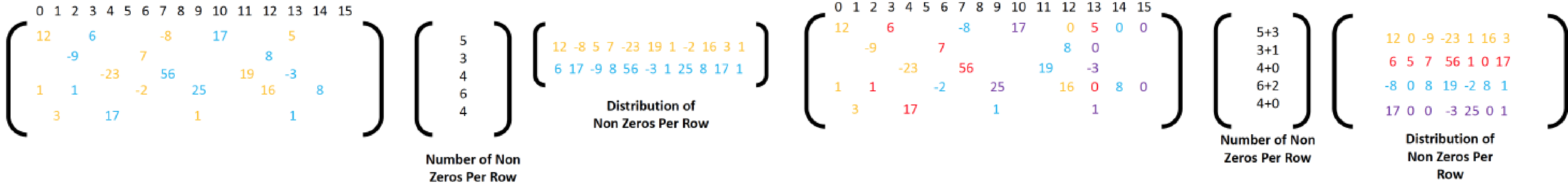
The DisNnz core allocates NNZ per row to the end of processing array.

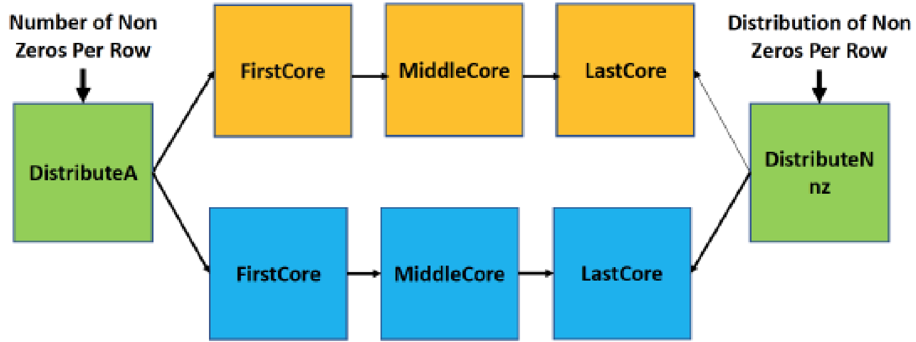
For the processing array, each big shared memory module is arbitrarily accessed by two processing cores from adjacent rows.

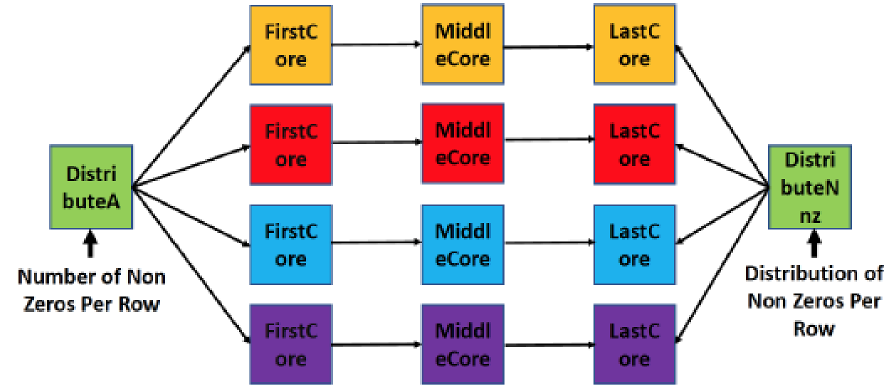


SpMV Architecture Mapping

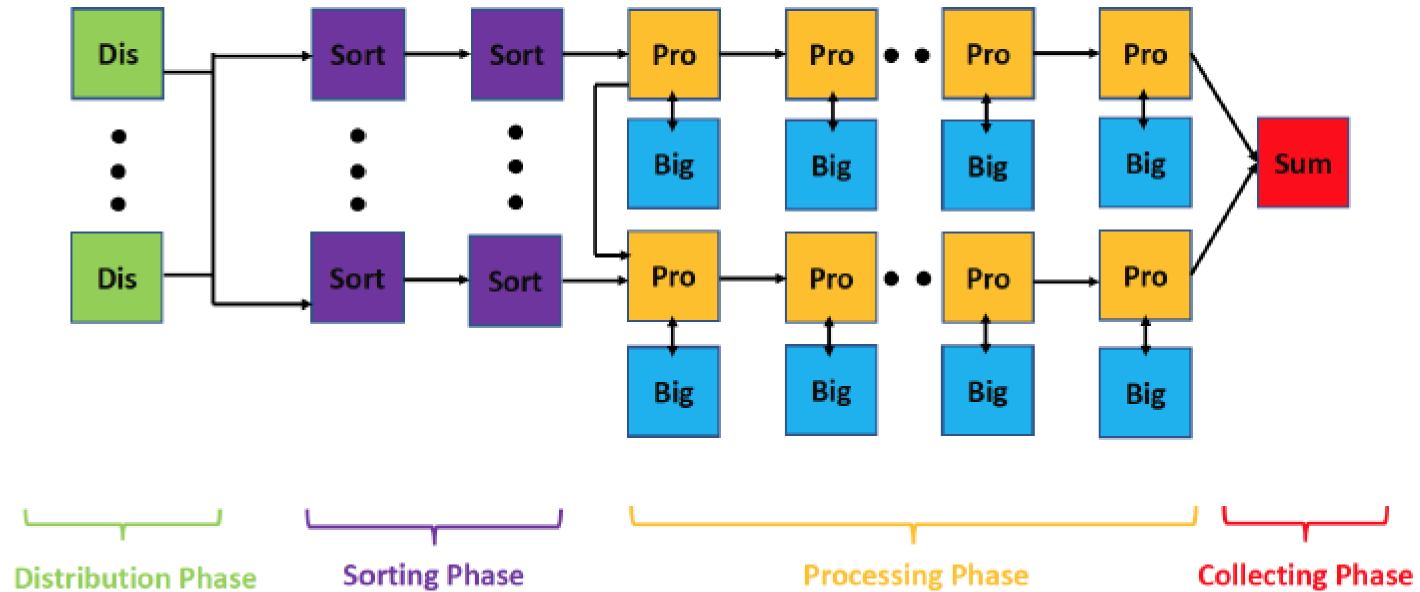
BigMemPara: NNZ per row distribution network







SpMV Architecture Mapping



Each mapping can be divided into up to four different kinds of phases and each phase is built from a similar set of kernels, as shown in the left figure.

Outline

- Introduction and Motivation
- Approach and Implementations
  - Platform: KiloCore
  - SpMV Kernels Mapping
- **Simulation Results**
- Performance Comparison
- Summary

SIMULATION RESULTS

Totally seven sparse integer matrices used to evaluate SpMV performance.
All integer matrix and vector data are in 16-bit fixed-point format. The median column size value for all the integer matrices in the entire database is 7521, while the average density is 0.00945.

THE RELATIVE THROUGHPUT PER AREA OF ALL 8 IMPLEMENTATIONS COMPUTING SEVEN SIMULATED INTEGER MATRICES (16 BIT FIXED POINT). THE VALUES REPORTED (THROUGHPUT PER AREA) ARE NORMALIZED AGAINST THE LEAST EFFICIENT IMPLEMENTATION.

Matrix Name	BigMem Snake	BigMem ParaOne	BigMem ParaTwo	BigMem ParaTwoNNZ	BigMem ParaFour	BigMem ParaFourPad	BigMem SubParaFour	BigMem SubParaFourTable
Andrews	1.64	1.75	2.21	1.95	2.21	**2.46**	1	1
pcb3000	4.29	4.43	**5.49**	4.53	3.67	4.88	1	1.02
p6000	6	6.14	**7.26**	6.09	3.54	6.17	1	1.37
TF17	1.61	1.7	2.16	1.90	2.75	**2.85**	1	1
foldoc	4.53	4.76	**5.94**	4.86	4	4.99	1	1
EAT_RS	4.09	4.13	**5.31**	4.37	4.84	4.71	1	1
ch7-7-b5	1.62	1.88	2.28	2	1.39	**2.44**	1	1

Outline

- Introduction and Motivation
- Approach and Implementations
  - Platform: KiloCore
  - SpMV Kernels Mapping
- Simulation Results
- **Performance Comparison**
- Summary

Performance Comparison

Throughput data for the implementations on the manycore platform, KiloCore, are obtained with a cycle-accurate C++ simulator.

Power measurements from the 32 nm PDSOI CMOS fabricated chip are input to the simulator to obtain power data.

Area usage is physically measured from the fabricated 32 nm PD-SOI CMOS chip, where each processor takes up 0.055 mm2 of area, and each shared memory occupies 0.164 mm2 of area.

Performance Comparison

Right table shows throughput per area for all implementations on different platforms when performing **SpMV** using the sparse matrix ***rail582***, which is relatively large and dense.

The minimum number of cores for the storage of vector x is first used for the **BigMemSnake** implementation, and then increased to create additional implementations.

THROUGHPUT PER AREA FOR VARIOUS SPMV IMPLEMENTATIONS OPERATING ON THE *rail582* SPARSE MATRIX.

Implementation/ Benchmark (Single FP Precision)	Throughput Per Area (MFLOPS/mm^2)
Quadro K620 Merge-based	29.3
Quadro K620 Cusparse CsrMv	21.1
Quadro K620 Cusparse HybMv	20.5
i7-3720QM Merge-based	22.4
BigMemSnake	105
BigMemParaOne	111.6
BigMemParaTwo	143
BigMemParaTwoNnz	125
BigMemParaFour	**208**
BigMemParaFourPad	**208**
BigMemSubParaFour	89
BigMemSubParaFourTable	89

Outline

- Introduction and Motivation
- Approach and Implementations
  - Platform: KiloCore
  - SpMV Kernels Mapping
- Simulation Results
- Performance Comparison
- **Summary**

Summary

The throughput and area consumption of all implementations are measured for performing SpMV on unstructured sparse matrices from distinct scientific workloads of varying sizes.

The proposed many-core implementations provide a **54.3** improvement in area efficiency versus the general-purpose processor SpMV on average, and **40.3** improvement versus the GPU SpMV on average.

Q & A

Thank you!