VSpGEMM: Exploiting Versal ACAP for High-Performance SpGEMM Acceleration

Kai Shi[1], Zhe Lin[2], Xinya Luan[1], Jianwang Zhai[1] and Kang Zhao[1]

[1] Beijing University of Posts and Telecommunications
[2] Sun Yat-sen University

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Contents

Introduction

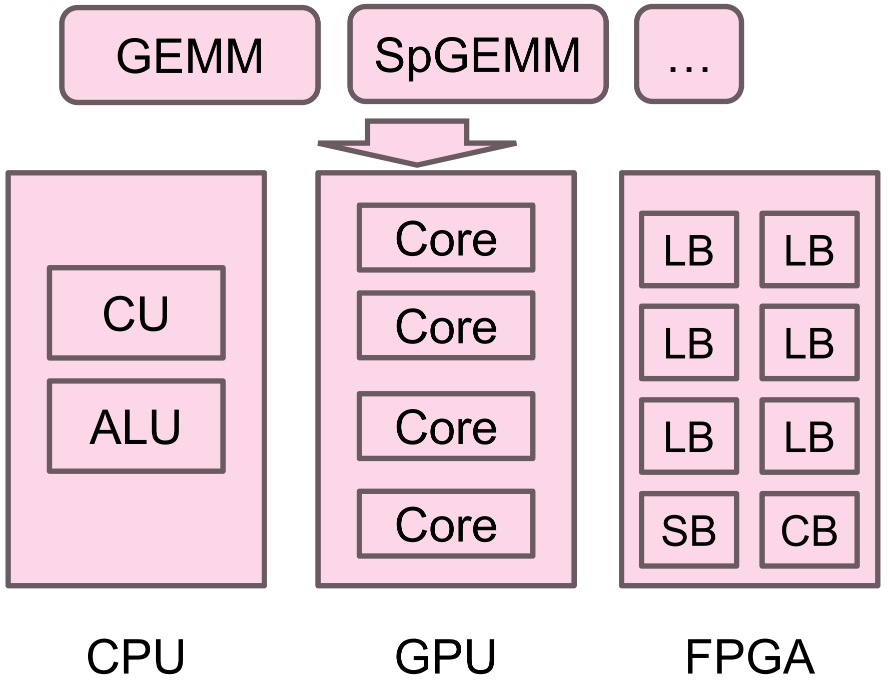
THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Backgroud

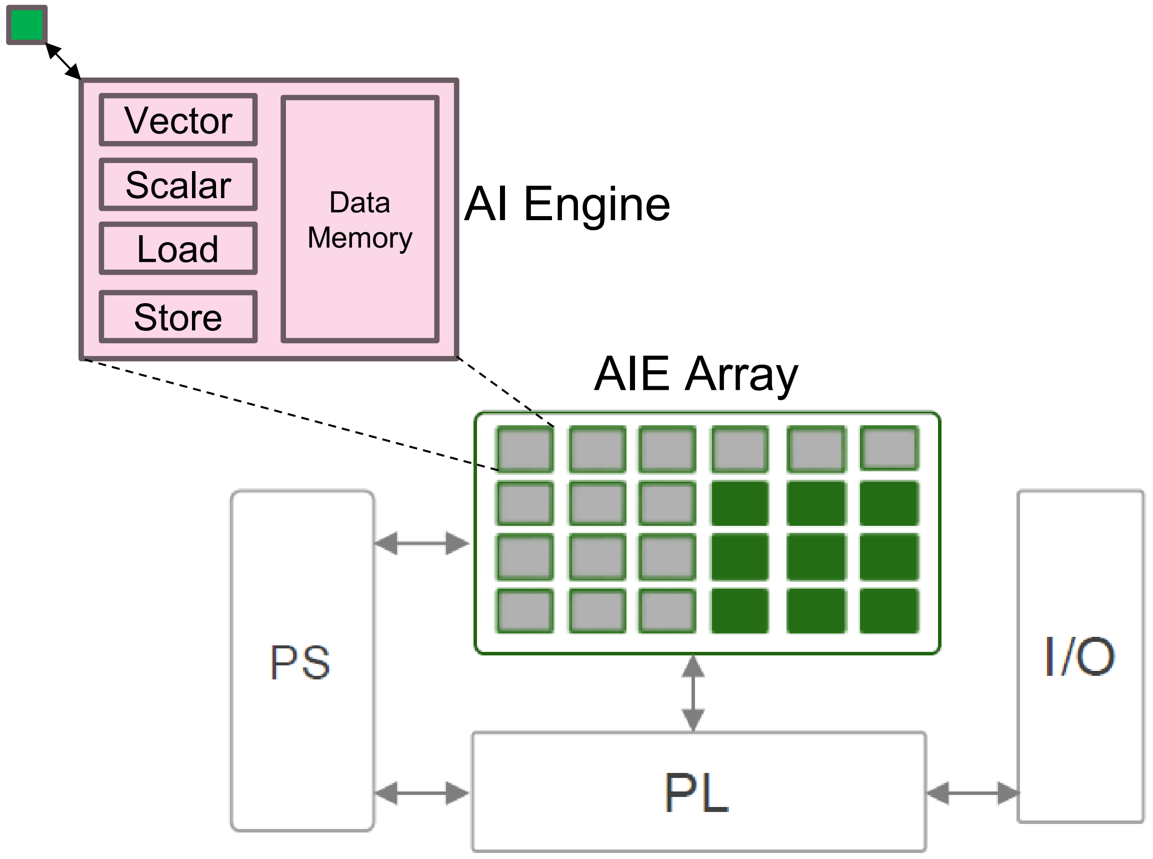
- CPU
  - Precise control units
- GPU
  - Numberous parallel cores
- FPGA
  - Flexible on-chip logic gates

➢ High power consumption
➢ Low energy efficiency
➢ Limited hardware resources



Versal ACAP

- AIE (AI Engines):
  - VLIW & SIMD processors
  - Local Data Memory
  - AXI4-Stream Switch and DMA
- PL (Programmable Logic)
  - Reconfigurable Logics
  - Larger On-Chip Memory
  - Multiple IOs Between AIEs and PL
- PS (Processing System)
  - Arm processor
  - XRT Runtime control

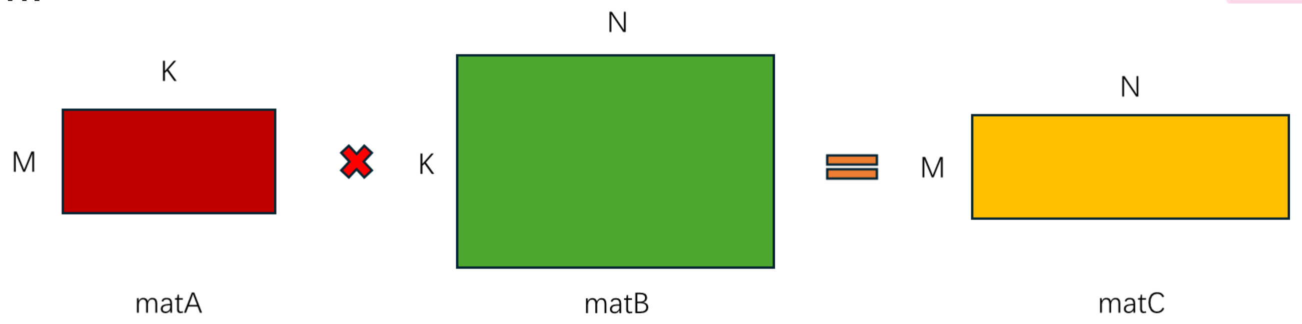


Versal ACAP Architecture

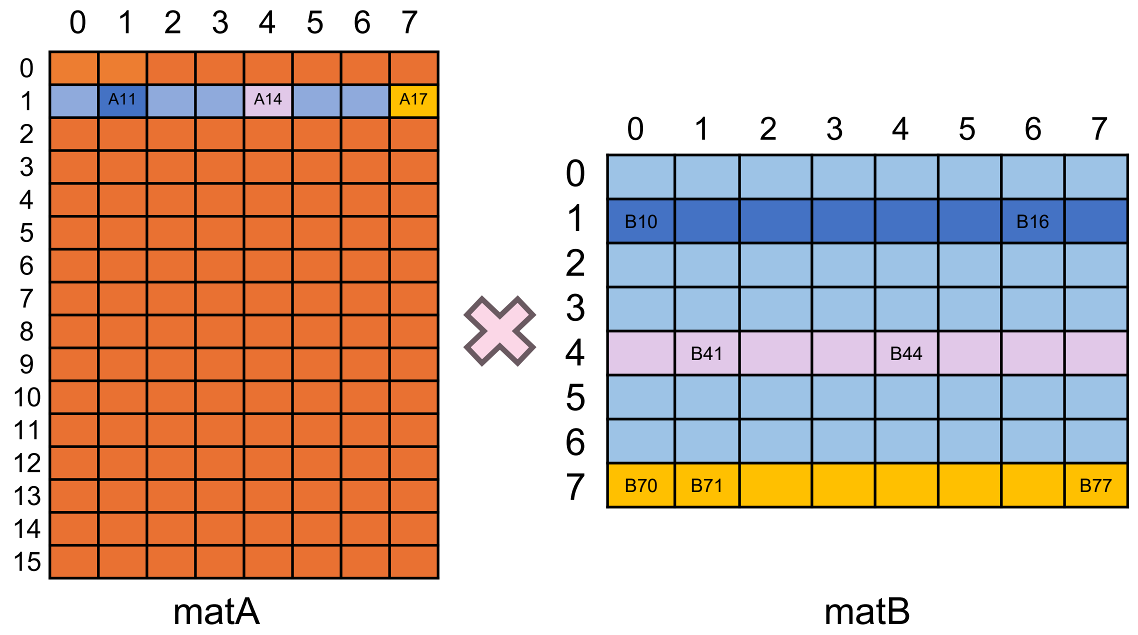
SpGEMM

V.S. GEMM

GEMM:



SpGEMM:



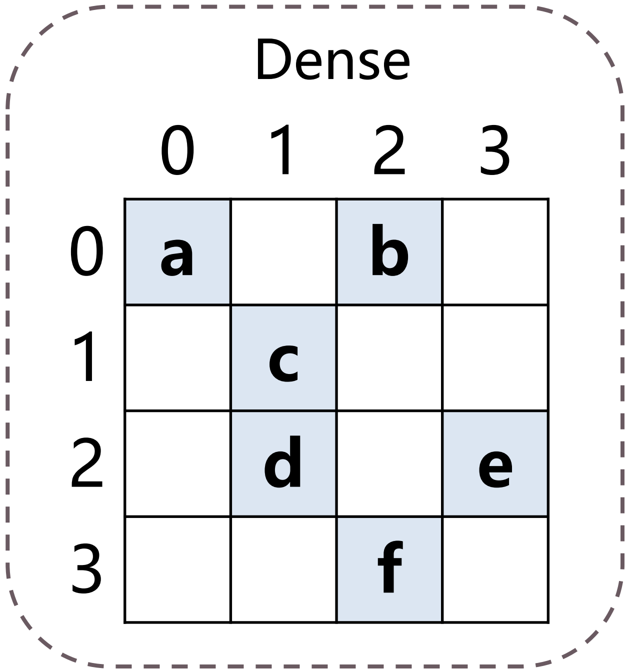
Features:

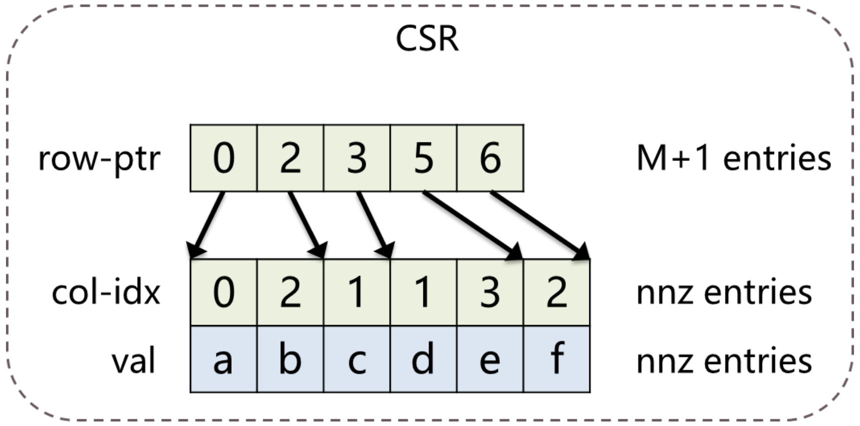
- Sparse: numerous zeros involved
  → Great inefficiency in computation
- Non-2D storage
  → Complicated indexing
- Irregular data access
  → Imbalanced workload on computing

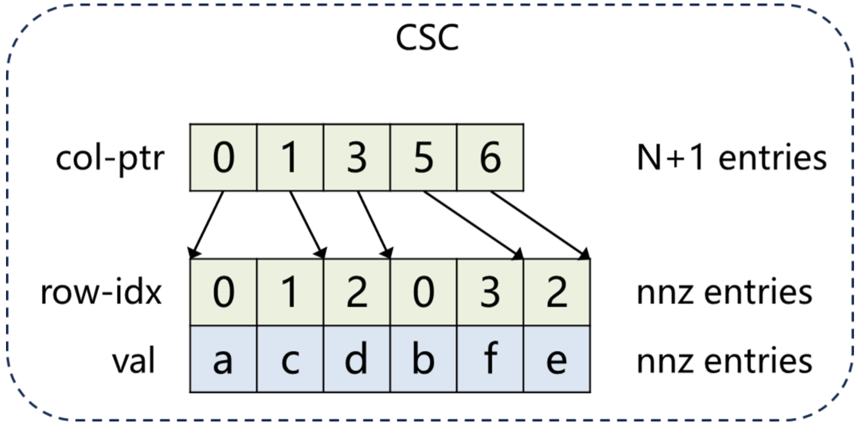
Challenges

Storage Formats

- Monolithic representation
- Multiple formats required
- Hard to utilize AIE vector unit







Challenges

Workload distibutions for AIE

	One-phase method	Two-phase method
Communication Burden (AIE - PL)	High	Low
Computation Overhead (AIE Array)	Low	High

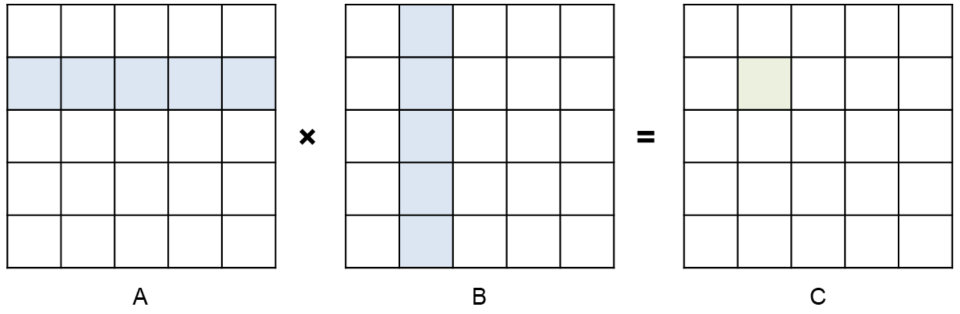
- **One-phase method**
  - index
  - value
  - Temporary intermediate product
  - Result matrix-C

- **Two-phase method**
  - accumulation
    - index
    - value
  - allocation
    - Result matrix-C

Challenges

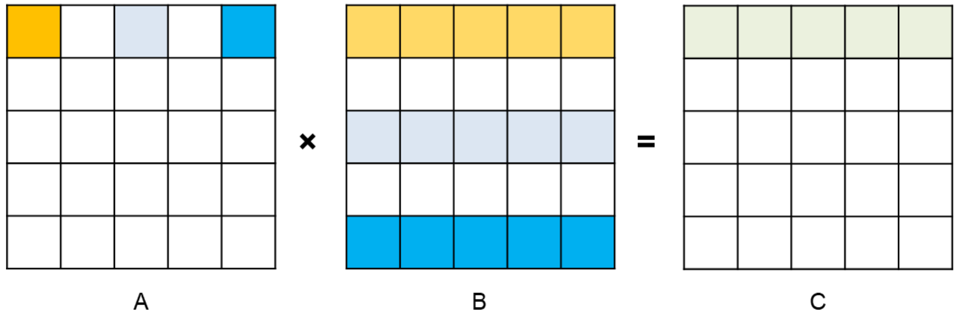
SpGEMM Algorithms

Inner Product



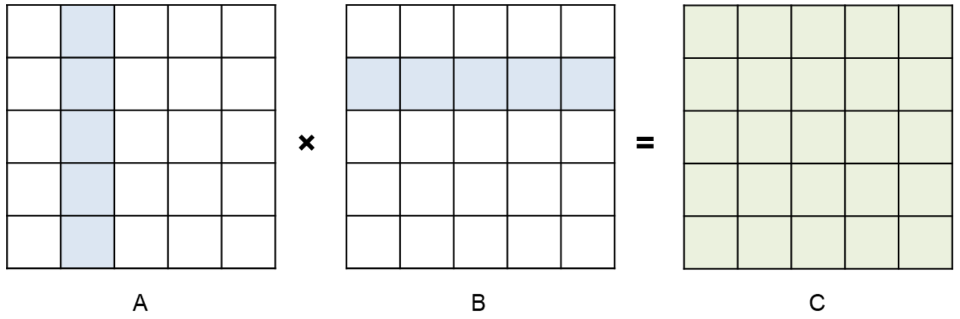
Bad data reuse for AIE tile

Row-wise
(Gustavson's method)



Irregular data access patterns

Outer Product



Large intermediate products

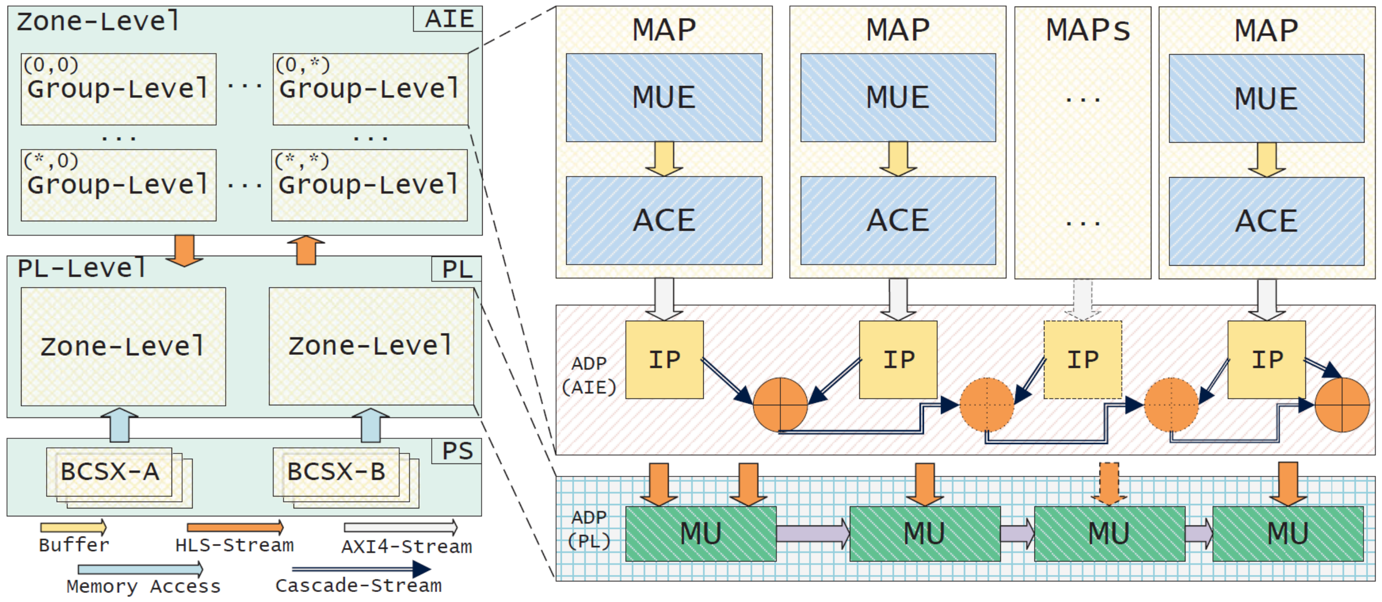
Methodology

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Framework

- VSpGEMM consists of several components:
  - BCSX storage format
  - Multi-level tiling scheme
  - Hyprid workload partitioning method

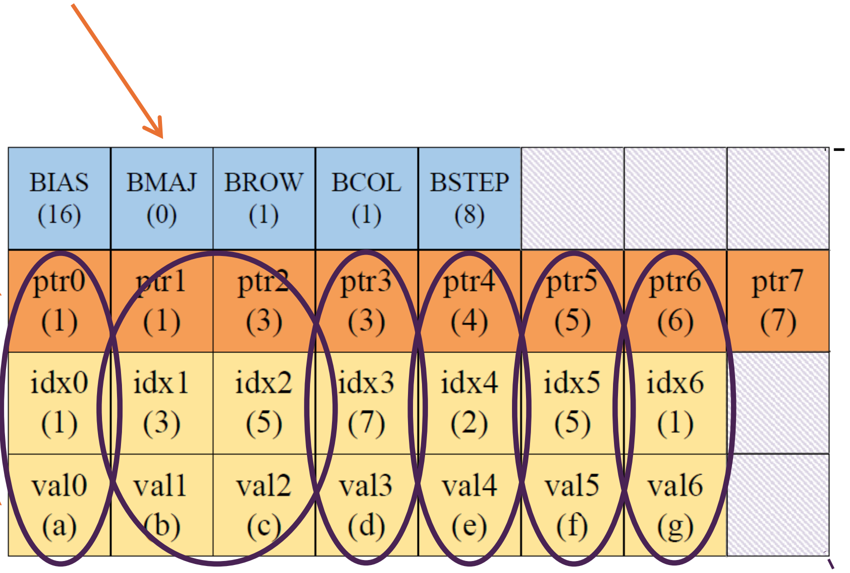


Architecture of VSpGEMM

Storage Format

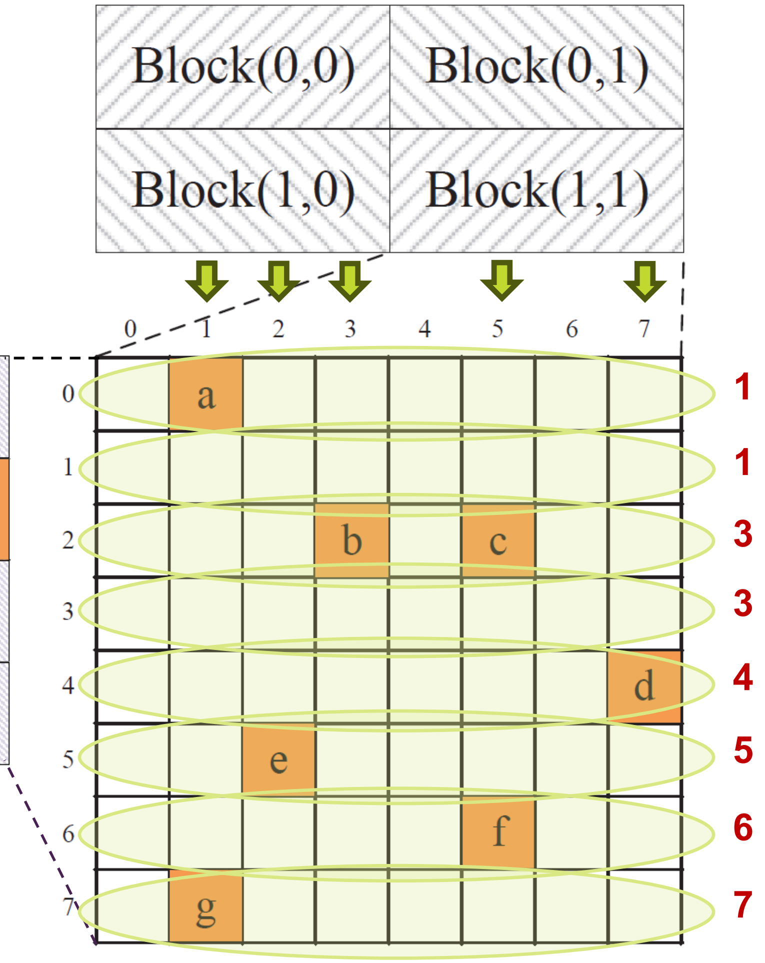
BCSX

- Composition of BCSX
  - Five descriptors:
    - *BIAS, BMAJ, BROW, BCOL, BSTEP*
  - Three arrays:
    - ptr*
    - idx*
    - val*



BIAS (16)	BMAJ (0)	BROW (1)	BCOL (1)	BSTEP (8)			
ptr0 (1)	ptr1 (1)	ptr2 (3)	ptr3 (3)	ptr4 (4)	ptr5 (5)	ptr6 (6)	ptr7 (7)
idx0 (1)	idx1 (3)	idx2 (5)	idx3 (7)	idx4 (2)	idx5 (5)	idx6 (1)	
val0 (a)	val1 (b)	val2 (c)	val3 (d)	val4 (e)	val5 (f)	val6 (g)	

BCSX Storage Format



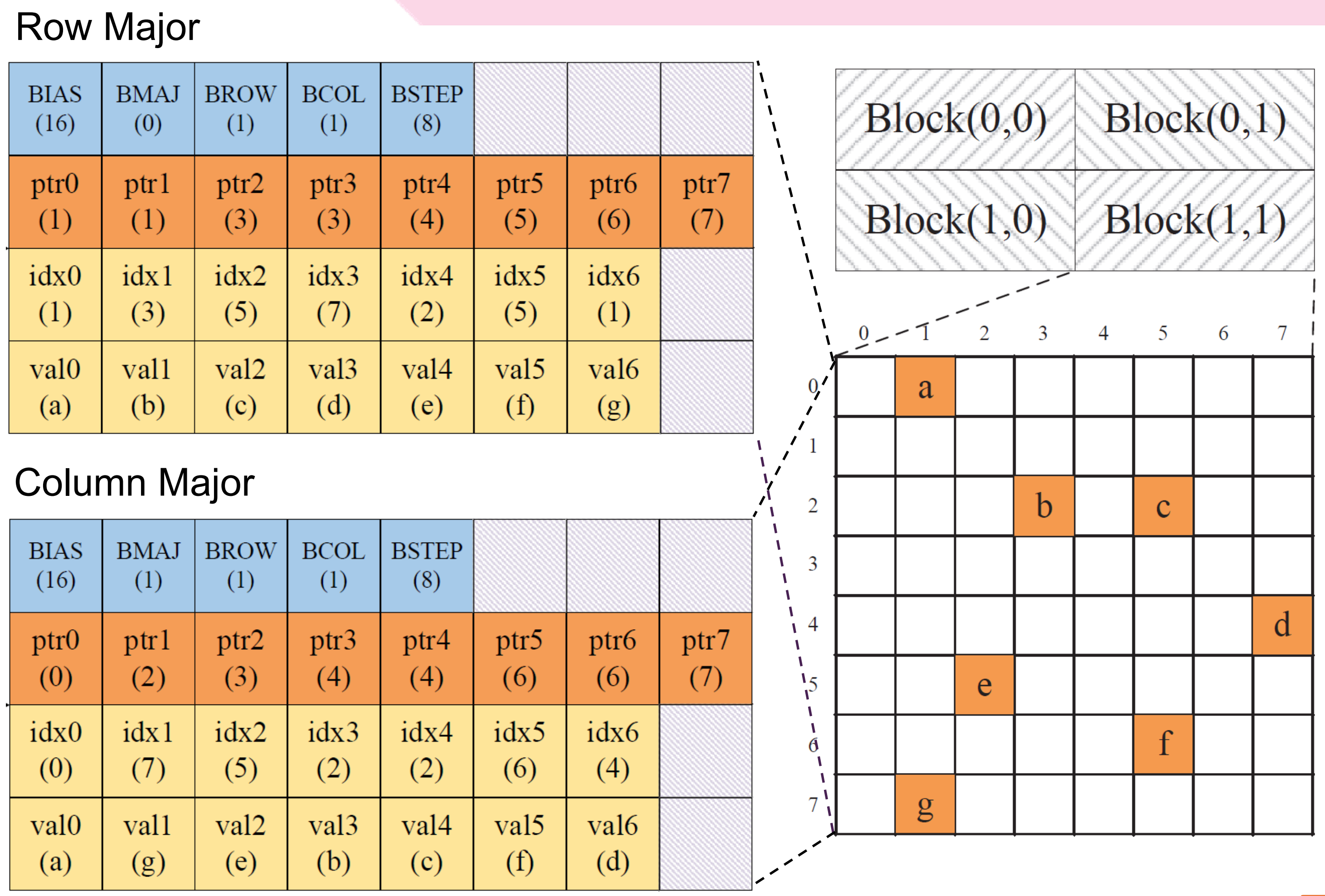
Storage Format

BCSX

- Composition of BCSX
  - Five descriptors:
    - *BIAS, BMAJ, BROW, BCOL, BSTEP*
  - Three arrays:
    - ptr*
    - idx*
    - val*

Storage Format \ Memory Access	Row Major	Column Major	Block-wise Structrue	Vectorized Loading
CSR	✓	×	×	×
CSC	×	✓	×	×
BCSX	✓	✓	✓	✓

Features of BCSX



Row Major

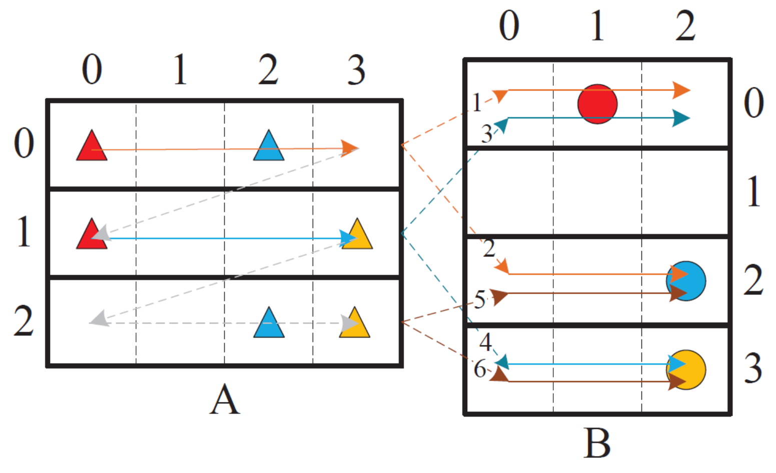
BIAS (16)	BMAJ (0)	BROW (1)	BCOL (1)	BSTEP (8)	▨	▨	▨
ptr0 (1)	ptr1 (1)	ptr2 (3)	ptr3 (3)	ptr4 (4)	ptr5 (5)	ptr6 (6)	ptr7 (7)
idx0 (1)	idx1 (3)	idx2 (5)	idx3 (7)	idx4 (2)	idx5 (5)	idx6 (1)	▨
val0 (a)	val1 (b)	val2 (c)	val3 (d)	val4 (e)	val5 (f)	val6 (g)	▨

Column Major

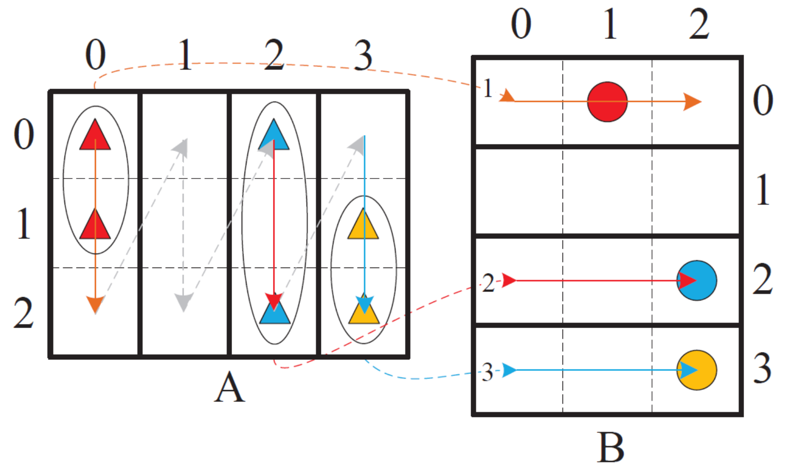
BIAS (16)	BMAJ (1)	BROW (1)	BCOL (1)	BSTEP (8)	▨	▨	▨
ptr0 (0)	ptr1 (2)	ptr2 (3)	ptr3 (4)	ptr4 (4)	ptr5 (6)	ptr6 (6)	ptr7 (7)
idx0 (0)	idx1 (7)	idx2 (5)	idx3 (2)	idx4 (2)	idx5 (6)	idx6 (4)	▨
val0 (a)	val1 (g)	val2 (e)	val3 (b)	val4 (c)	val5 (f)	val6 (d)	▨

Storage Format

SpGEMM Algorithm



Gustavson's Method

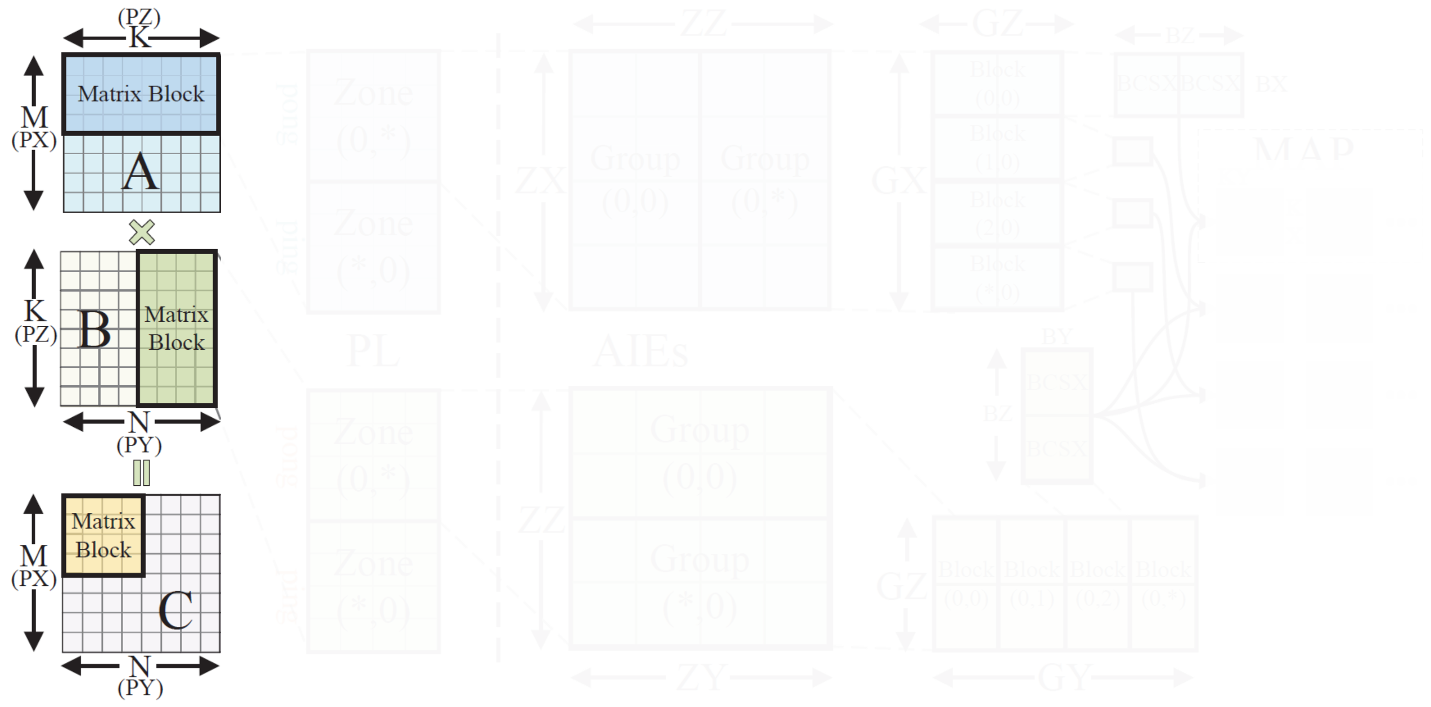


Outer-product Method (Ours)

- Outer product method under BCSX
  - Unify matrix A & B in same format
  - Reuse non-zeros of Matrix B

Multi-Level Tiling

- Input Matrices
  - *A: M × K*
  - *B: K × N*



DDR

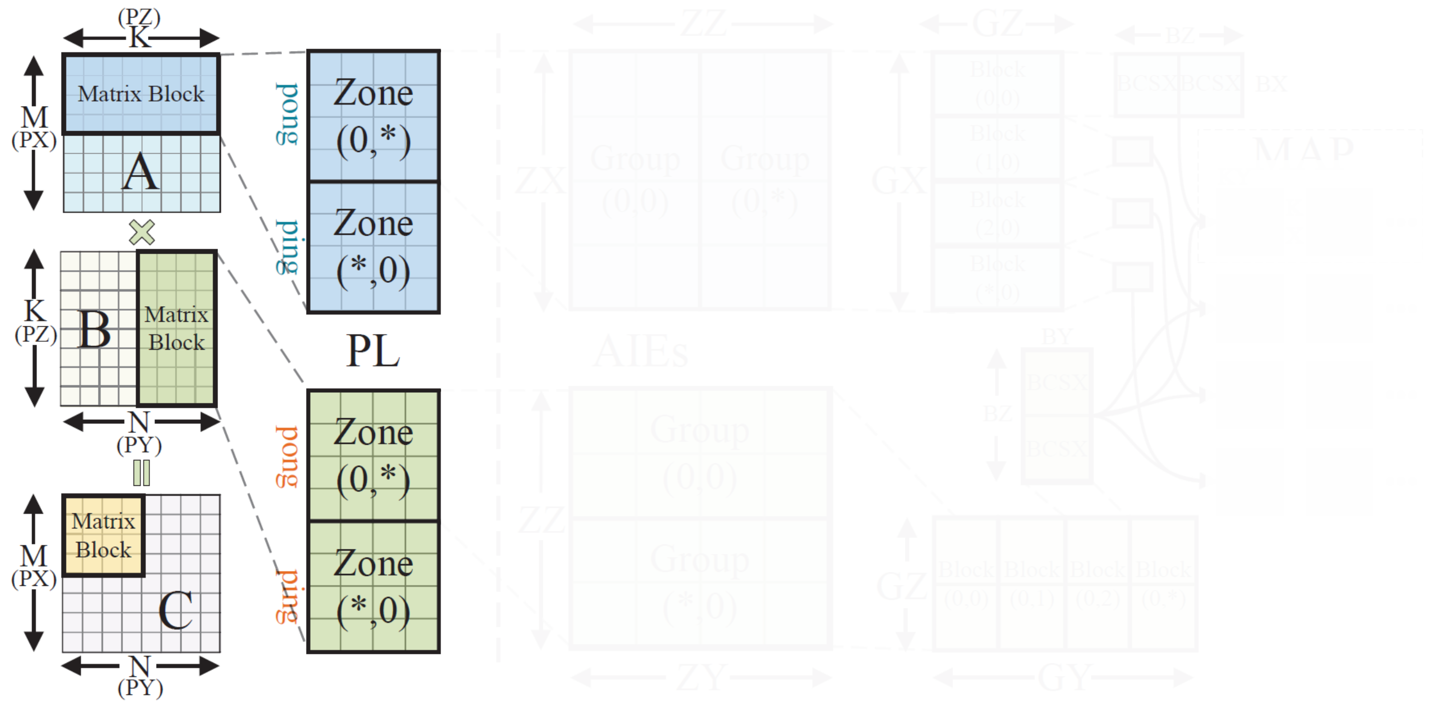
AXI-Stream

Multi-Level Tiling Scheme

Multi-Level Tiling

- PL-Level Tiling

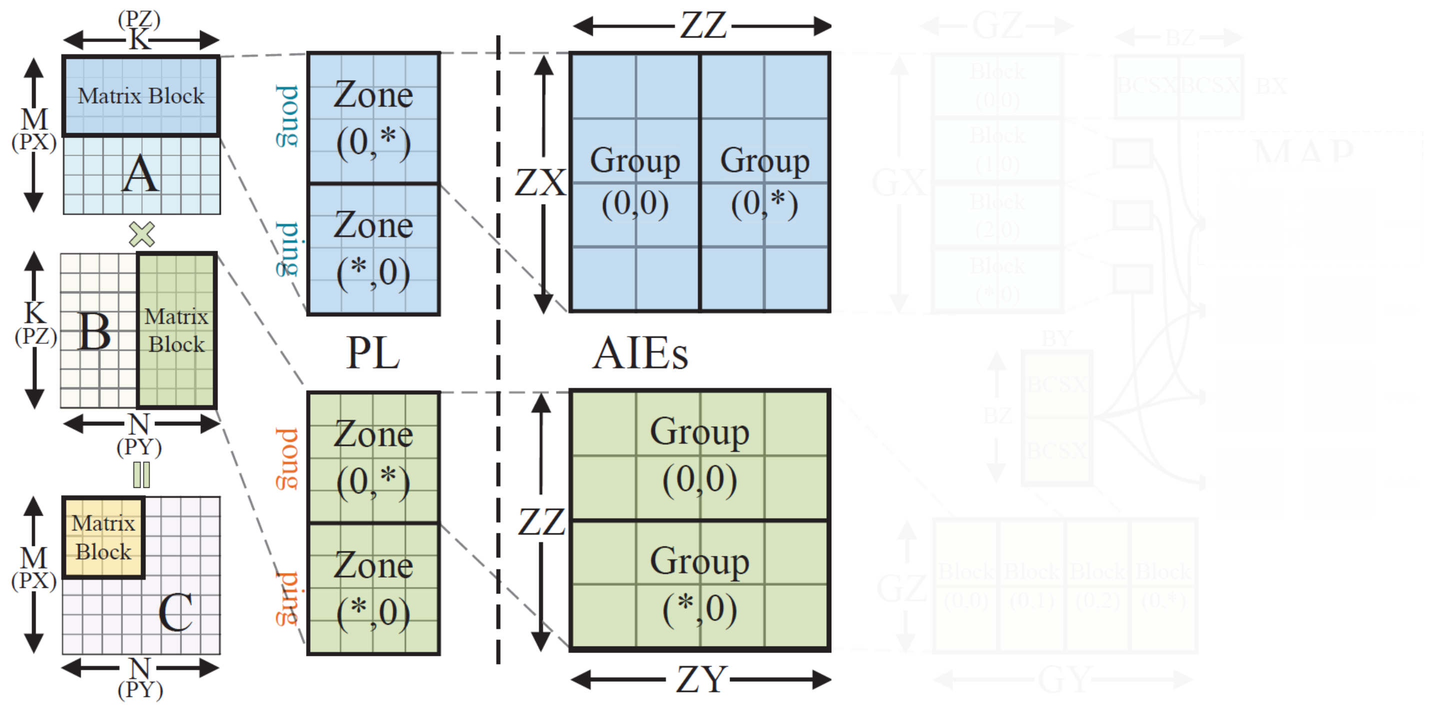
  Sparse Matrices A and B in BCSX are split into *Zone-tiles on PL.*



Multi-Level Tiling Scheme

Multi-Level Tiling

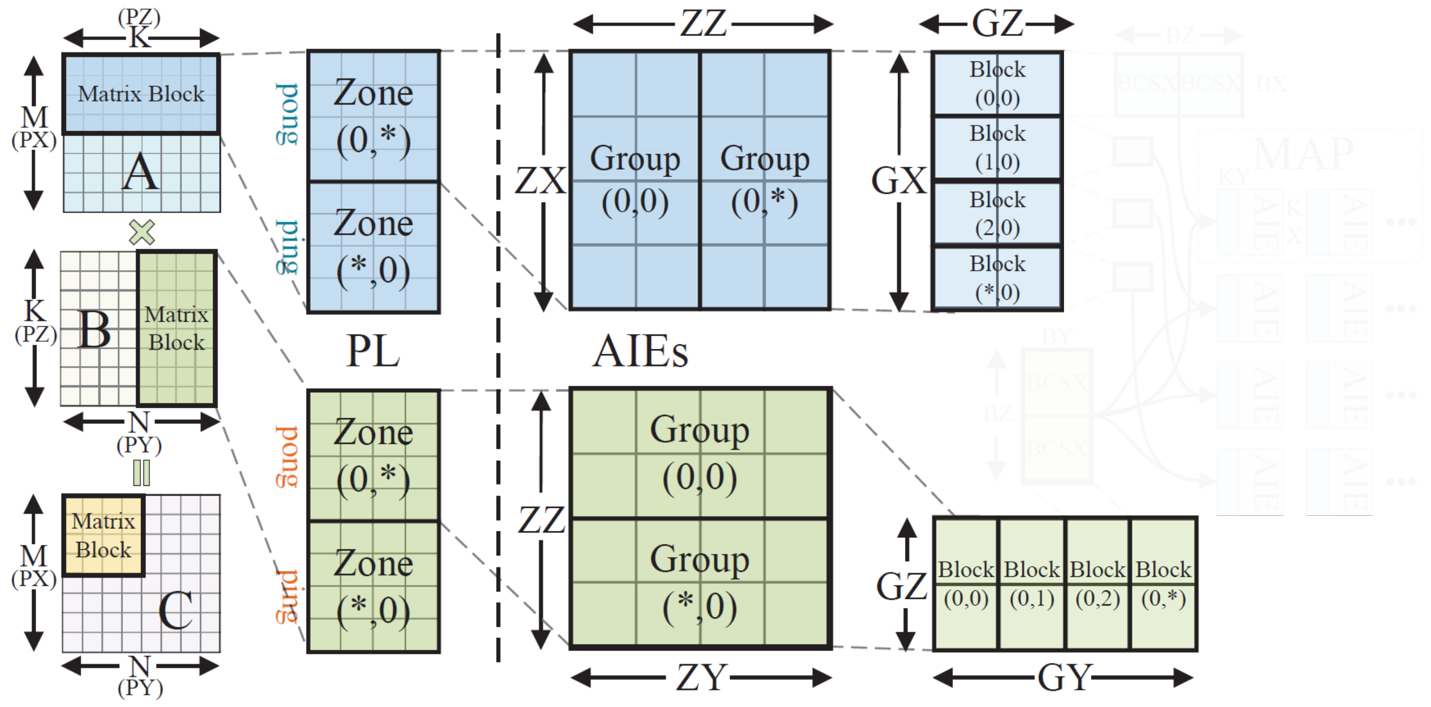
- Zone-Level Tiling
  Each Zone-tile is mapped into whole AIE array for partially SpGEMM.



Multi-Level Tiling Scheme

Multi-Level Tiling

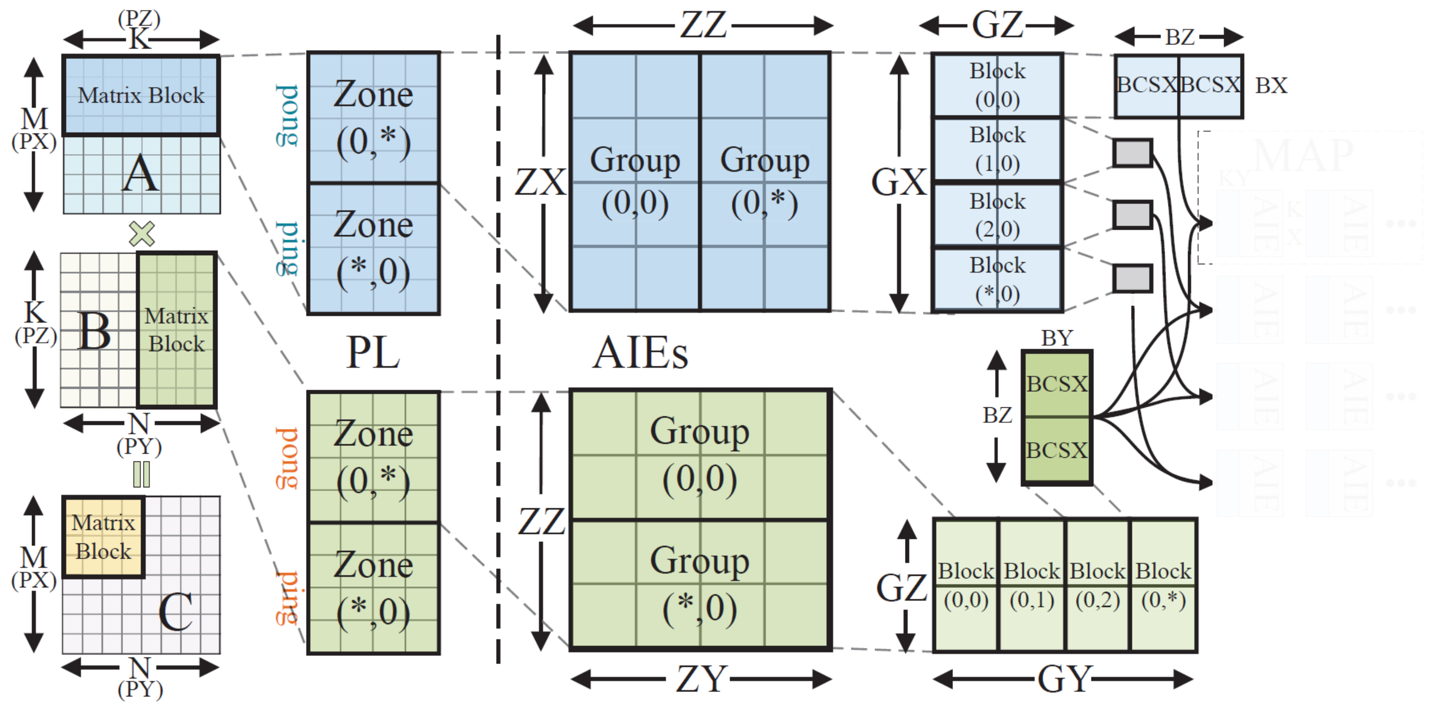
- Group-Level Tiling
  Group-tiles within a Zone-tile are further split to perform local SpGEMM.



Multi-Level Tiling Scheme

Multi-Level Tiling

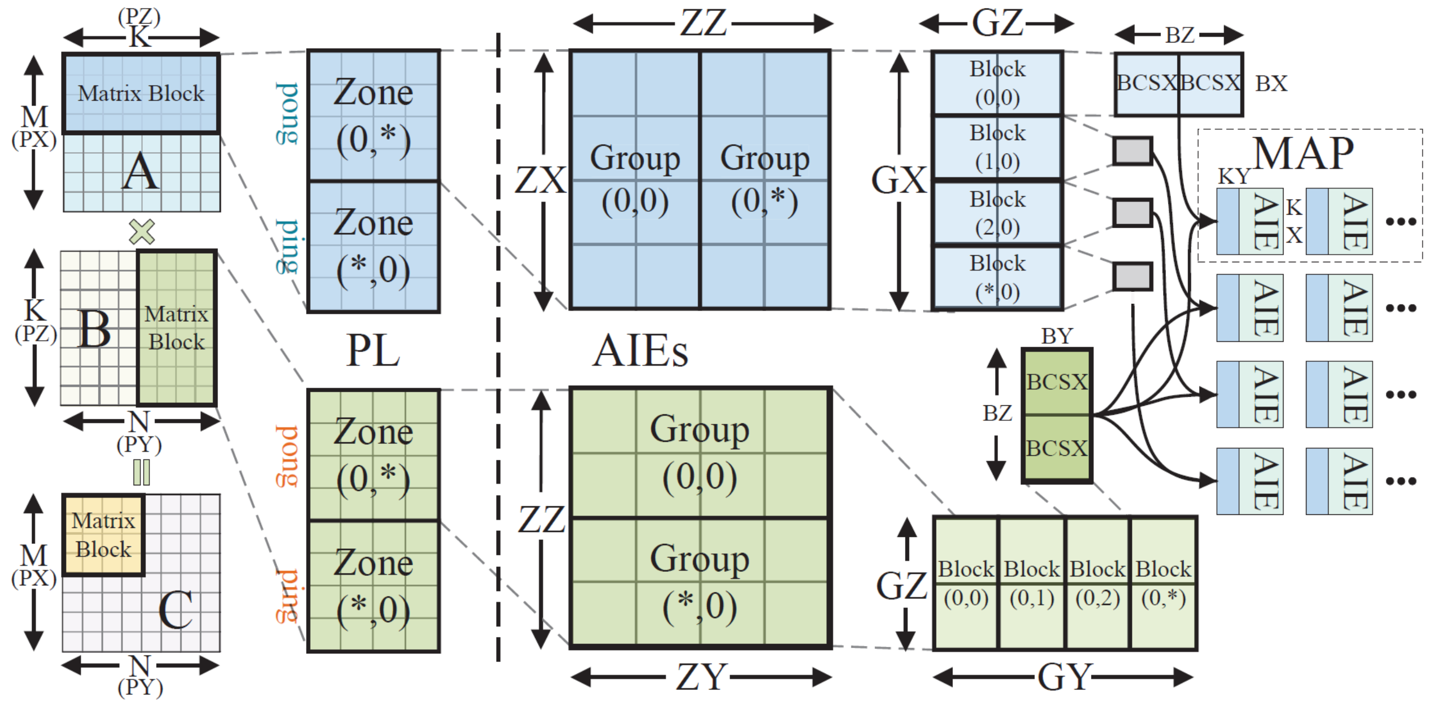
- Block-Level Tiling
  Each Group-tile is then assigned to Block-tiles to execute SpGEMM that reuses input matrix blocks.



Multi-Level Tiling Scheme

Multi-Level Tiling

- Kernel-Level Tiling
  Perform inner-most SpGEMM with matrices in BCSX format, which denotes as a MAP unit.

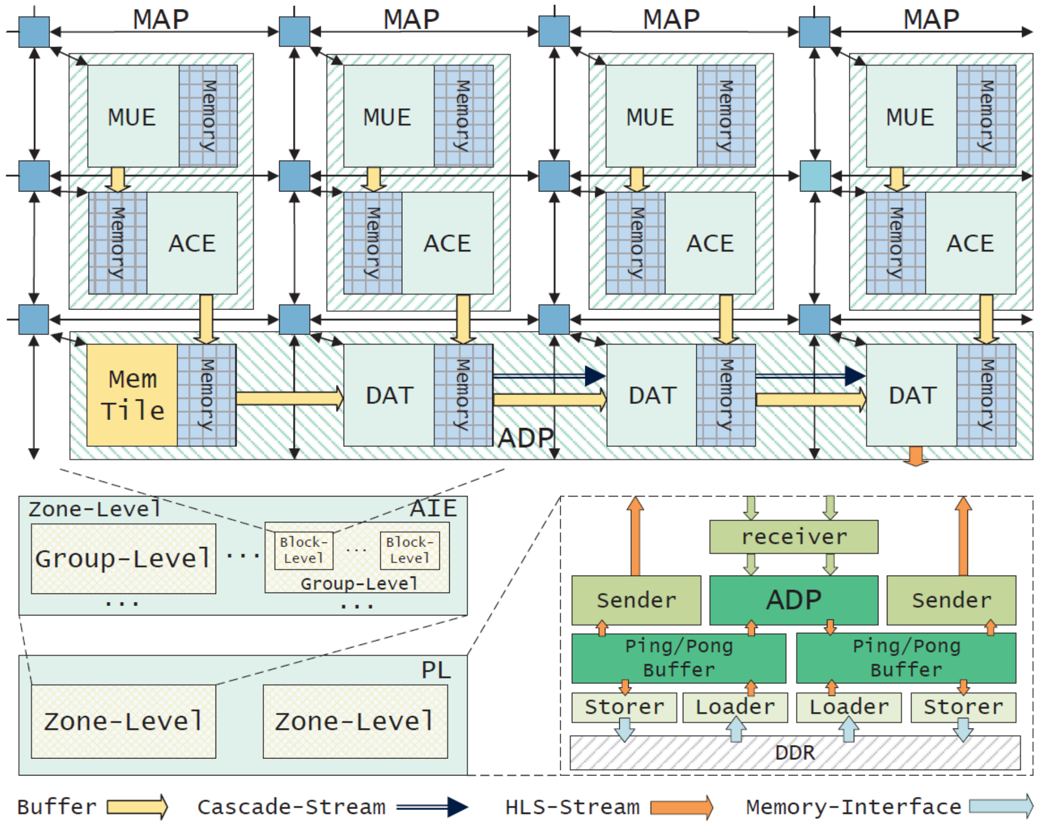


Multi-Level Tiling Scheme

Partition Scheme

MAC-Partition & Hybrid Add-Partition

- MAC-Partition on AIEs
  - MUL-Engine(MUE) + ACC-Engine(ACE), neighboured data access between AIE Tiles.
- Hybrid ADD-Partition (ADP)
  - ADP on AIEs: chained DATs (Dense-acc tiles).
  - ADP on PL: Merge intermediate products in dense matrices with initial interval (II) = 1.
  - Offload the intermediate product merging workload both in AIE and PL, shrinking AIE-PL communication burden.



Multi-Level Tiling Scheme

Evaluation

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Evaluation

Experimental Settings

- SpGEMM Datasets
  - SuiteSparse Matrix Collection
- Versal ACAP Device
  - VCK 190 Evalution Kit
  - PL: 250 MHz, AIE: 1.25 GHz
- Software Tools
  - Vitis 2024.1
  - CUDA 12.6
- GPU Device
  - NVIDIA RTX 4090

Evaluation
Baseline

- GEMM on Versal
    - CHARM [TRETS' 2024]
- SpGEMM on GPU
    - cuSPARSE [CUDA Libraries]

Evaluation

Performance results of VSpGEMM

Both CHARM and VSpGEMM are implemented in INT16 datatype.

Average Speedup:

- V.S. CHARM: 2.65×

Table: Throughput of VSpGEMM

Datasets			CHARM			VSpGEMM (Ours)			Speedup
Benchmark (ID-Name)	M × N	NNZ	AIE Tiles	PLIOs	Throughput (GOPS)	AIE Tiles	PLIOs	Throughput (GOPS)	
2397_football	115 × 115	663	144	64	26.53	58	12	46.64	1.76 ×
1945-TF11	216 × 236	1607	144	64	179.59	176	32	435.65	2.42 ×
1982-GL6_D_10	163 × 341	2053	144	64	147.28	264	44	568.01	3.85 ×
2209-Trefethen_500	500 × 500	4489	198	146	339.17	176	32	1204.4	3.55 ×
2210-Trefethen_700	700 × 700	6677	192	96	1219.9	264	44	2055.5	1.68 ×

Evaluation

Energy Efficiency results of VSpGEMM

Both CHARM and VSpGEMM are implemented in INT16 datatype, cuSPARSE is implemented in FP32 datatype.

Average energy efficiency gain:

- V.S. cuSPARSE: 33.62×
- V.S. CHARM: 2.74×

Table: Latency, power and energy efficiency of VSpGEMM

Benchmark (ID-Name)	cuSPARSE			CHARM			VSpGEMM (Ours)			EE. Gain	
	Latency (ms)	Power (W)	EE. (GOPS/W)	Latency (ms)	Power (W)	EE. (GOPS/W)	Latency (ms)	Power (W)	EE. (GOPS/W)	V.S. cuSPARSE	V.S. CHARM
2397_football	39.21	89	0.08	0.36	22.28	1.19	0.21	15.45	3.02	37.75×	2.54×
1945-TF11	28.60	90	0.21	0.36	23.26	7.72	0.26	24.20	18.00	85.71×	2.33×
1982-GL6_D_10	24.81	89	1.50	0.36	22.41	6.57	0.26	28.19	20.15	13.45×	3.07×
2209-Trefethen_500	25.67	92	2.58	0.55	28.23	12.01	0.46	25.38	47.45	18.38×	3.95×
2210-Trefethen_700	25.91	95	5.61	0.67	31.08	39.25	0.64	28.64	71.77	12.79×	1.83×

Evaluation

Energy Efficiency results of VSpGEMM

Experiments for VSpGEMM V.S. CHARM are implemented in INT16 datatype, and VSpGEMM V.S. cuSPARSE are implemented in FP32 datatype.

Average energy efficiency gain:

- V.S. cuSPARSE: 31.07×

Table: Latency, power and energy efficiency of VSpGEMM

Benchmark (ID-Name)	cuSPARSE			VSpGEMM (Ours, FP32)			CHARM			VSpGEMM (Ours, INT16)			EE. Gain	
	Latency (ms)	Power (W)	EE. (GOPS/W)	Latency (ms)	Power (W)	EE. (GOPS/W)	Latency (ms)	Power (W)	EE. (GOPS/W)	Latency (ms)	Power (W)	EE. (GOPS/W)	V.S. cuSPARSE	V.S. CHARM
2397_football	39.21	89	0.08	0.21	16.95	5.71	0.36	22.28	1.19	0.21	15.45	3.02	71.38×	2.54×
1945-TF11	28.60	90	0.21	0.32	24.39	11.74	0.36	23.26	7.72	0.26	24.20	18.00	55.90×	2.33×
1982-GL6_D_10	24.81	89	1.50	0.32	29.29	12.08	0.36	22.41	6.57	0.26	28.19	20.15	8.05×	3.07×
2209-Trefethen_500	25.67	92	2.58	0.65	25.71	23.23	0.55	28.23	12.01	0.46	25.38	47.45	8.62×	3.95×
2210-Trefethen_700	25.91	95	5.61	1.10	32.10	30.28	0.67	31.08	39.25	0.64	28.64	71.77	5.40×	1.83×

Conclusion

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Conclusion

- A new **compressed storage format, BCSX**, which enables **fixed-pattern data accesses**, preserves **high data locality**, and supports **data preloading**.
- A **multi-level tiling scheme** to hierarchically distribute the computation of SpGEMM to multiple AIEs while enhancing **data reuse** during computation.
- A **hybrid workload partitioning method** that efficiently allocates the intermediate product merging operations to AIEs and PL, ensuring **minimal communication overhead**.
- The first attempt to optimize SpGEMM on Versal ACAP, finally achieving a **2.65×** speedup over CHARM on Versal and a **33.62×** energy efficiency gain against cuSPARSE on the RTX 4090 GPU.

Thank you

Shi Kai

Beijing University of Posts and Telecommunications