PPoPP 26

Characterizing Matrix Multiplication Units across General Parallel Patterns in Scientific Computing

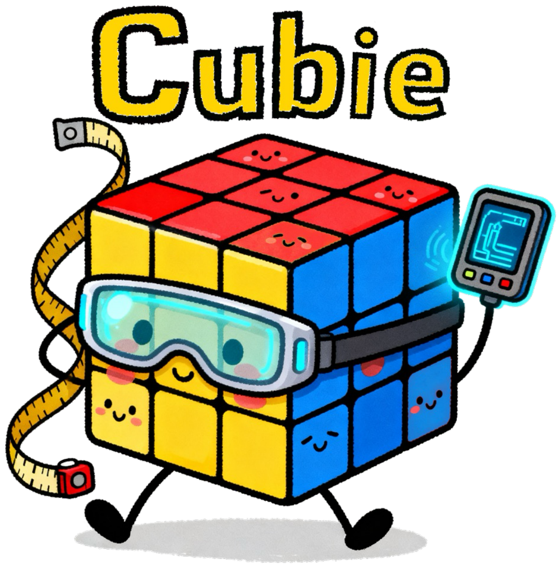
Yuechen Lu[1], Hongwei Zeng[1], Marc Casas[2], Weifeng Liu[1]

[1] China University of Petroleum-Beijing, China

[2] Barcelona Supercomputing Center, Spain

Sydney, Australia · Feb 4, 2026

Code: https://doi.org/10.5281/zenodo.15290623



OUTLINE

1. Background and Motivation
2. The Cubie Benchmark Suite
3. Categorization of MMU Utilization Patterns
4. Experiments
5. Comparison with other Benchmark Suites
6. Conclusion

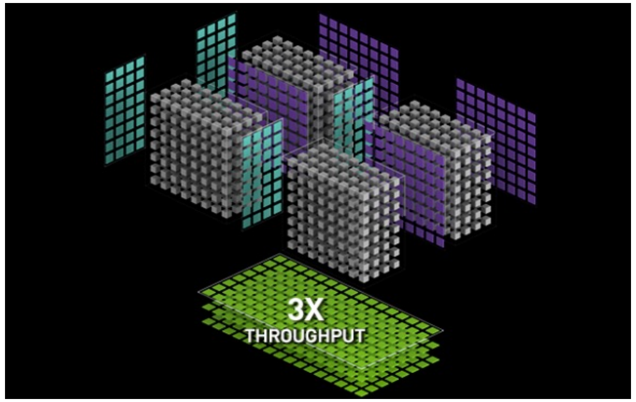
OUTLINE

1. Background and Motivation
2. The Cubie Benchmark Suite
3. Categorization of MMU Utilization Patterns
4. Experiments
5. Comparison with other Benchmark Suites
6. Conclusion

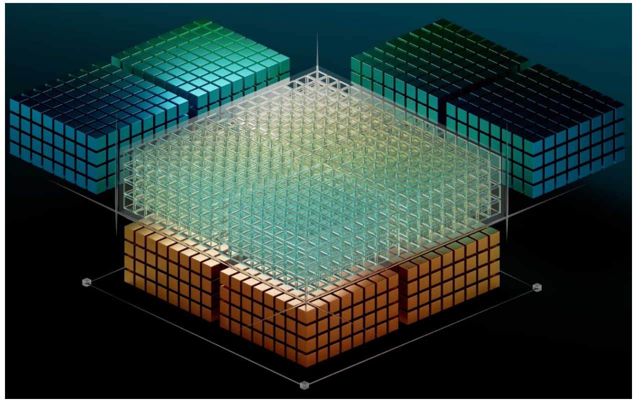
Background and Motivation

- **MMU:** Matrix Multiply-Accumulate Unit
- MMUs have shown strong impact in deep learning, but **their role in scientific computing** is still not well understood.

NVIDIA Tensor Core



AMD Matrix Core

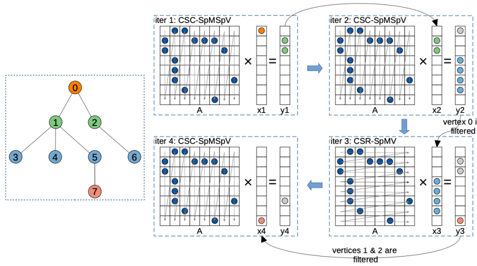


	NVIDIA H100	AMD MI300X	
Peak FP64	25.6 TFLOPS	81.7 TFLOPS	2x
Peak FP64 TC	51.2 TFLOPS	163.4 TFLOPS	
Peak FP32	51.2 TFLOPS	163.4 TFLOPS	7x 4x
Peak FP32 TC	N/A	163.4 TFLOPS	
Peak TF32 TC	378 TFLOPS	653.7 TFLOPS	
Peak FP16	102.4 TFLOPS	N/A	7x
Peak FP16 TC	756 TFLOPS	1307.4 TFLOPS	
Peak BF16	102.4 TFLOPS	N/A	7x
Peak BF16 TC	756 TFLOPS	1307.4 TFLOPS	
Peak FP8 TC	1513 TFLOPS	2614.9 TFLOPS	
Peak INT8 TC	1513 TOPS	2614.9 TOPS	

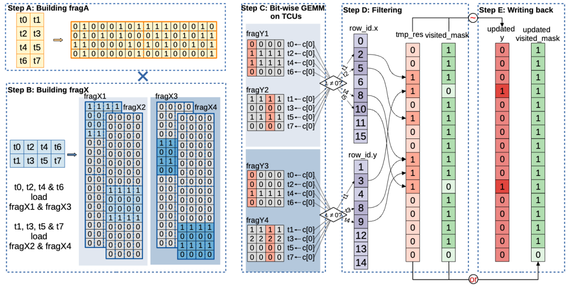
MMUs offer 2～7× higher peak throughput.

Background and Motivation

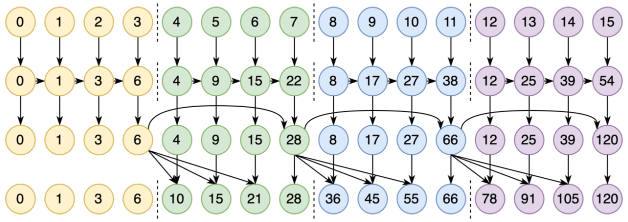
- **Diverse parallel patterns** in scientific workloads make effective MMU utilization nontrivial.
- Recent studies indicate that **MMUs can accelerate key scientific kernels** (Stencil, Scan, BFS…)



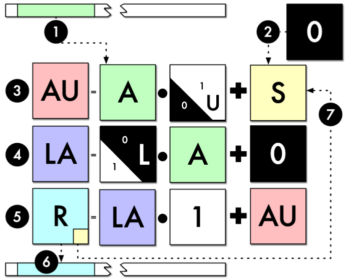
BFS



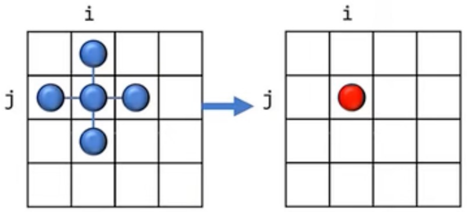
Niu et al. Berrybees BFS



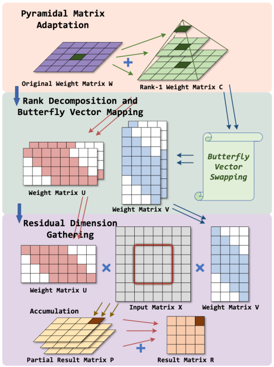
Scan



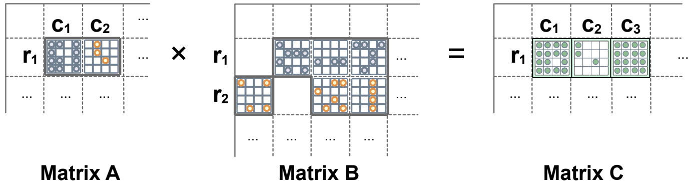
Dakkak et al. TCU Scan



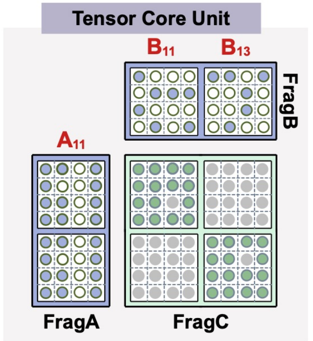
Stencil Computation



Zhang et al. LoRaStencil



SpGEMM

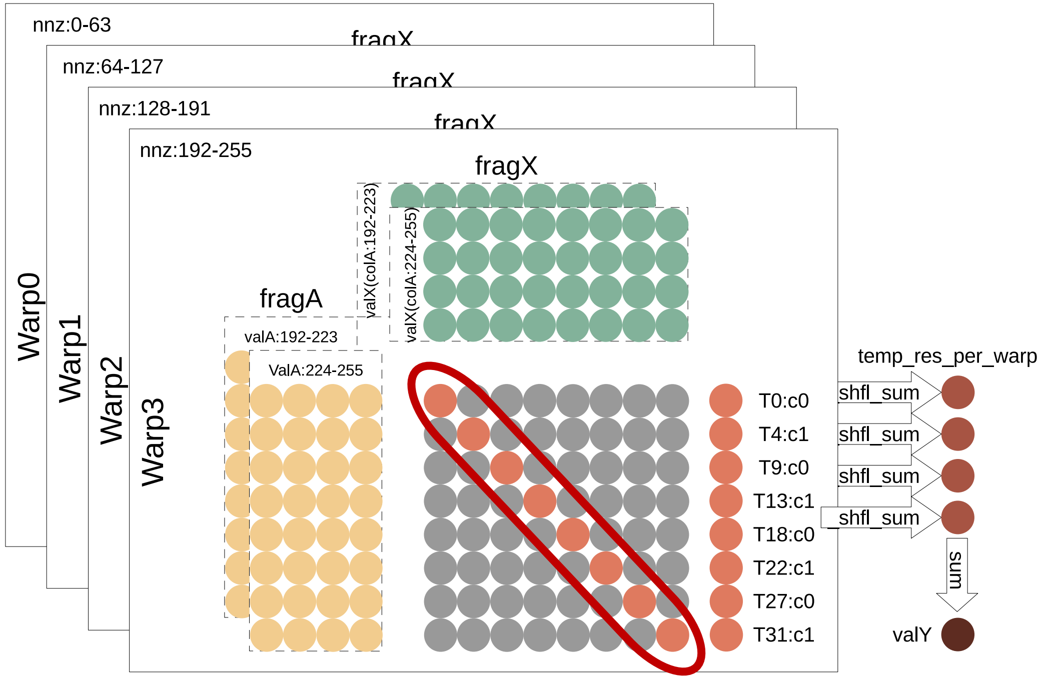


Lu et al. mBSR SpGEMM

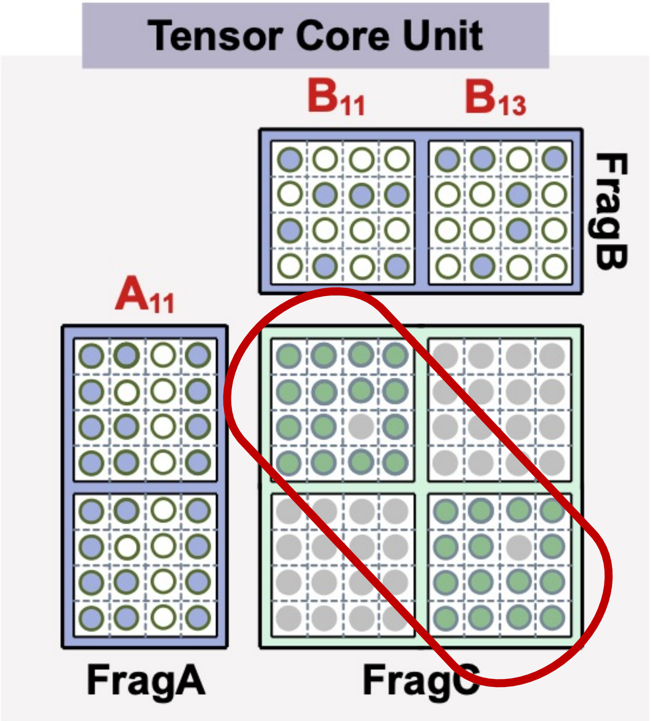
- However, we still lack **a systematic MMU analysis tool** for architecture researchers, parallel algorithm researchers, and HPC application researchers.

Background and Motivation

- **Bandwidth perspective:** SpMV and SpGEMM are **bandwidth bound**. If **bandwidth does not change**, why can MMUs speed them up?
- **Compute perspective:** FP64 Tensor Cores offer only **~2× higher** peak than CUDA Cores, yet many kernels use only a small part of the MMA output (e.g. **1/8 or 1/2**). Why can we still see large speedups (e.g. DASP SpMV can get **5.75×** speedups on cop20k_A over cuSPARSE)?



DASP SpMV

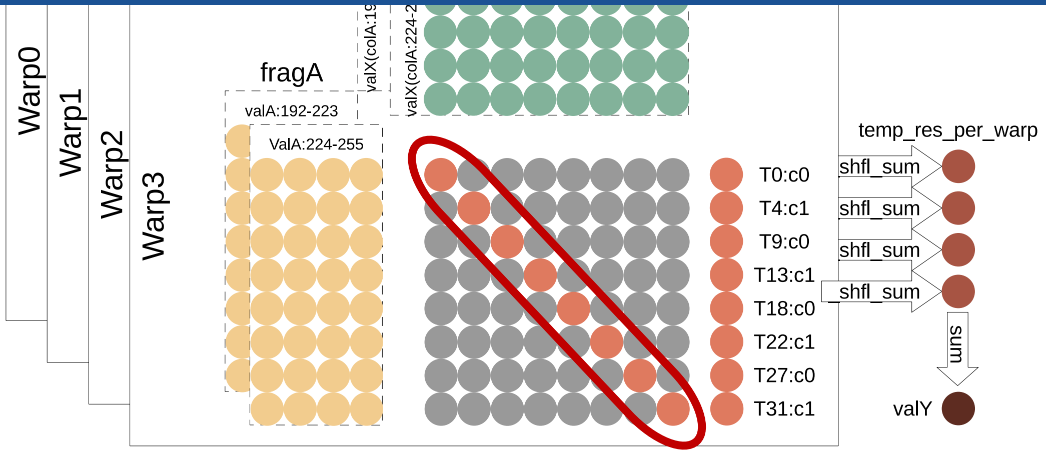


mBSR SpGEMM

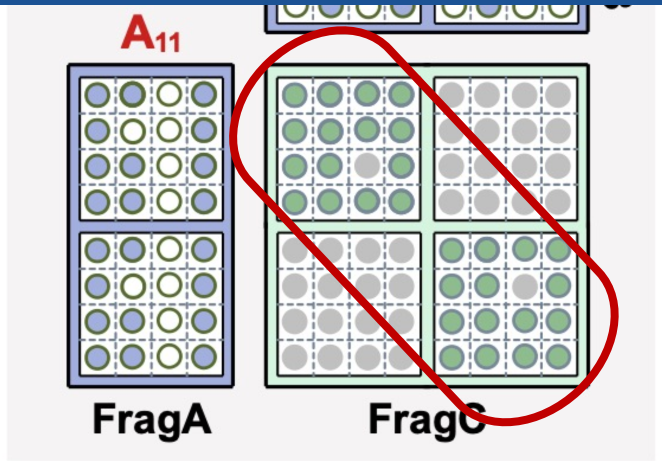
Background and Motivation

- **Bandwidth perspective:** SpMV and SpGEMM are **bandwidth bound**. If **bandwidth does not change**, why can MMUs speed them up?
- **Compute perspective:** FP64 Tensor Cores offer only **~2× higher** peak than CUDA Cores, yet many kernels use only a small part of the MMA output (e.g. **1/8 or 1/2**). Why can we still see large speedups (e.g. DASP SpMV can get **5.75×** speedups on cop20k_A over cuSPARSE)?

A scientific computing benchmark suite for MMUs is needed!



DASP SpMV



mBSR SpGEMM

OUTLINE

1 Background and Motivation

2 The Cubie Benchmark Suite

3 Categorization of MMU Utilization Patterns

4 Experiments

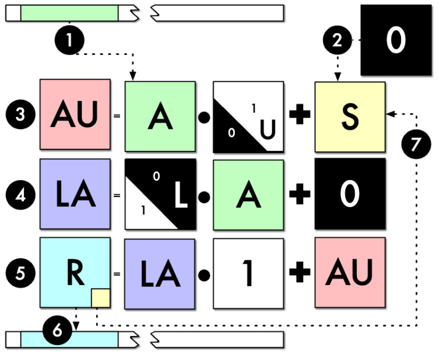
5 Comparison with other Benchmark Suites

6 Conclusion

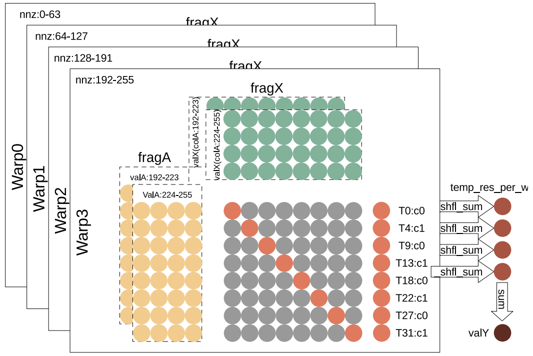
The Cubie Benchmark Suite

- Cubie includes ten open source scientific kernels accelerated with MMUs.

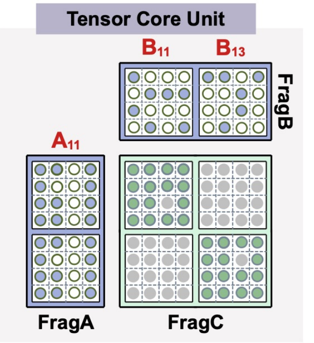
Kernel	Ref	Berkeley Dwarf	Baseline
GEMV	-	Dense LA	cuBLAS
GEMM	cudaSample	Dense LA	cudaSample
SpMV	DASP SpMV	Sparse LA	cuSPARSE
SpGEMM	mBSR SpGEMM	Sparse LA	cuSPARSE
FFT	tcFFT	Spectral methods	cuFFT
Stencil	LoRaStencil	Structured grids	DRStencil
Reduction	TCU-Reduction	MapReduce	CUB
Scan	TCU-Scan	MapReduce	CUB
BFS	BerryBees	Graph traversal	Gunrock
PiC	PiCTC	N-Body methods	-



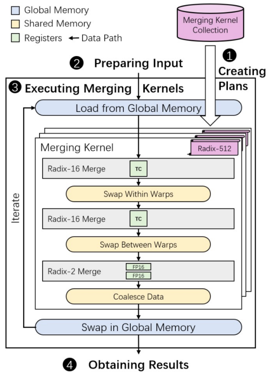
Dakkak et al.
TCU Scan



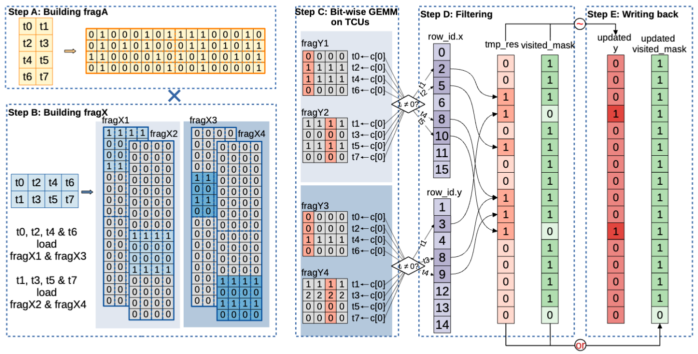
Lu et al.
DASP SpMV



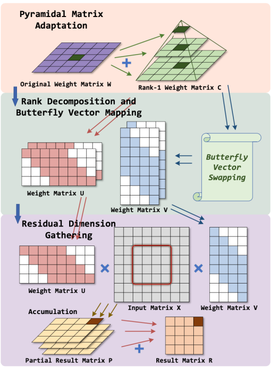
Lu et al.
mBSR SpGEMM



Li et al.
tcFFT



Niu et al.
Berrybees BFS



Zhang et al.
LoRaStencil

Key Observation 1: To exploit MMUs, non-GEMM algorithms in scientific computing often have to modify data structures and reorganize algorithms.

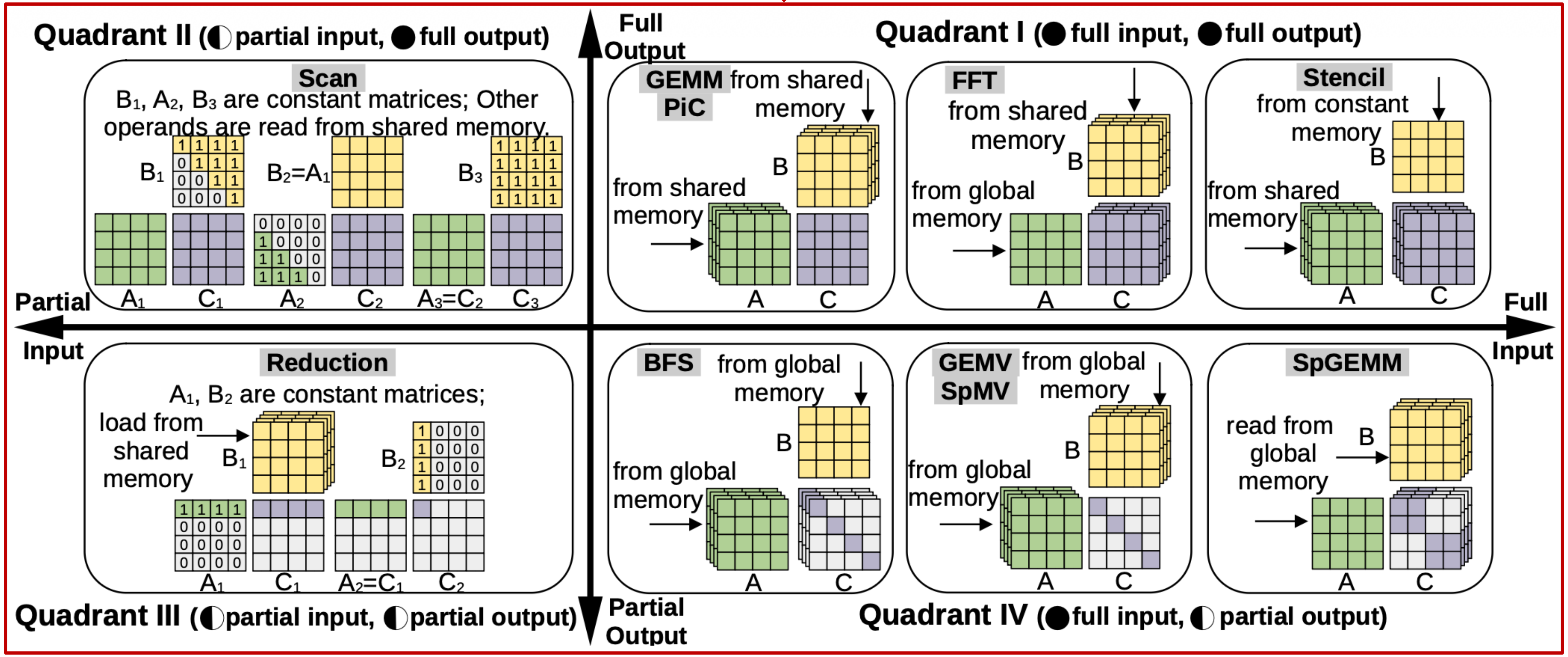
OUTLINE

1. Background and Motivation
2. The Cubie Benchmark Suite
3. Categorization of MMU Utilization Patterns
4. Experiments
5. Comparison with other Benchmark Suites
6. Conclusion

Categorization of MMU Utilization Patterns

- **Two dimensions**: Input utilization and Output utilization
- **Two levels**: Full and Partial

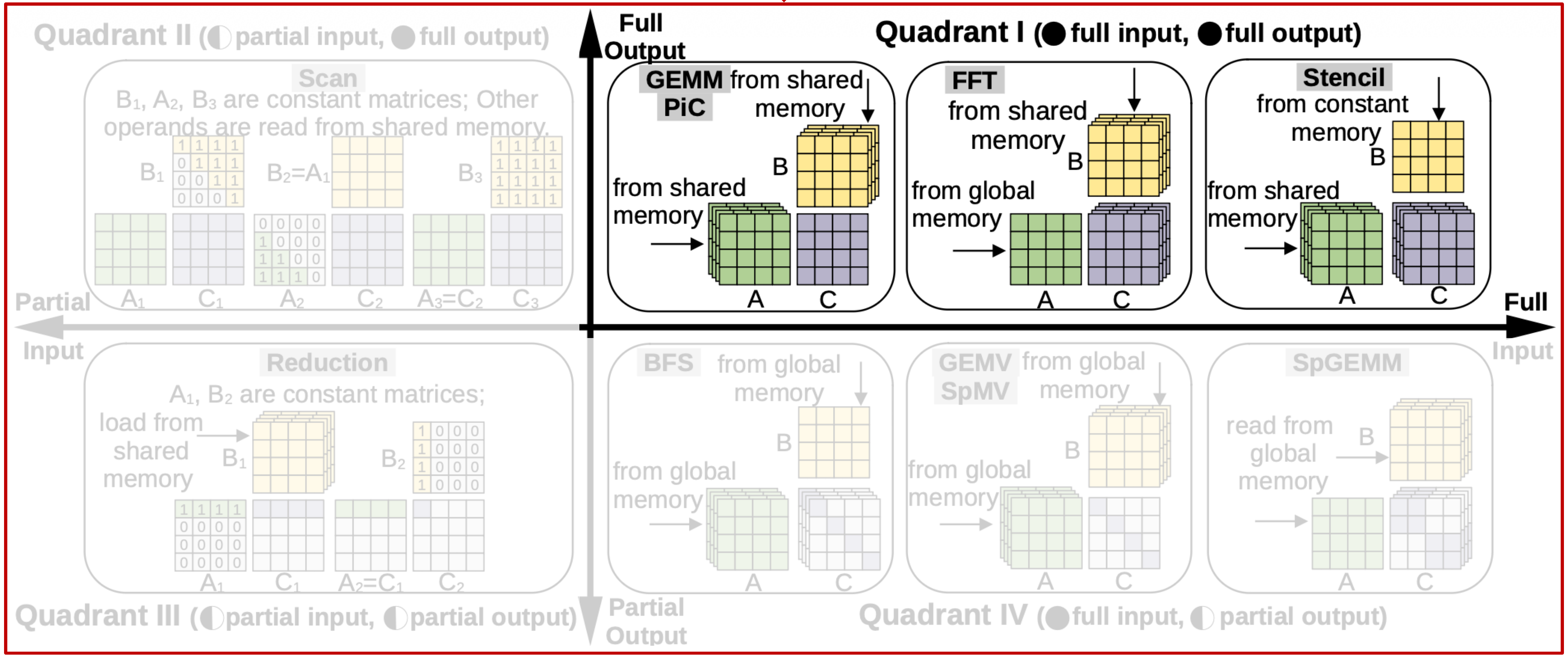
Four Quadrants



Categorization of MMU Utilization Patterns

- **Two dimensions**: Input utilization and Output utilization
- **Two levels**: Full and Partial

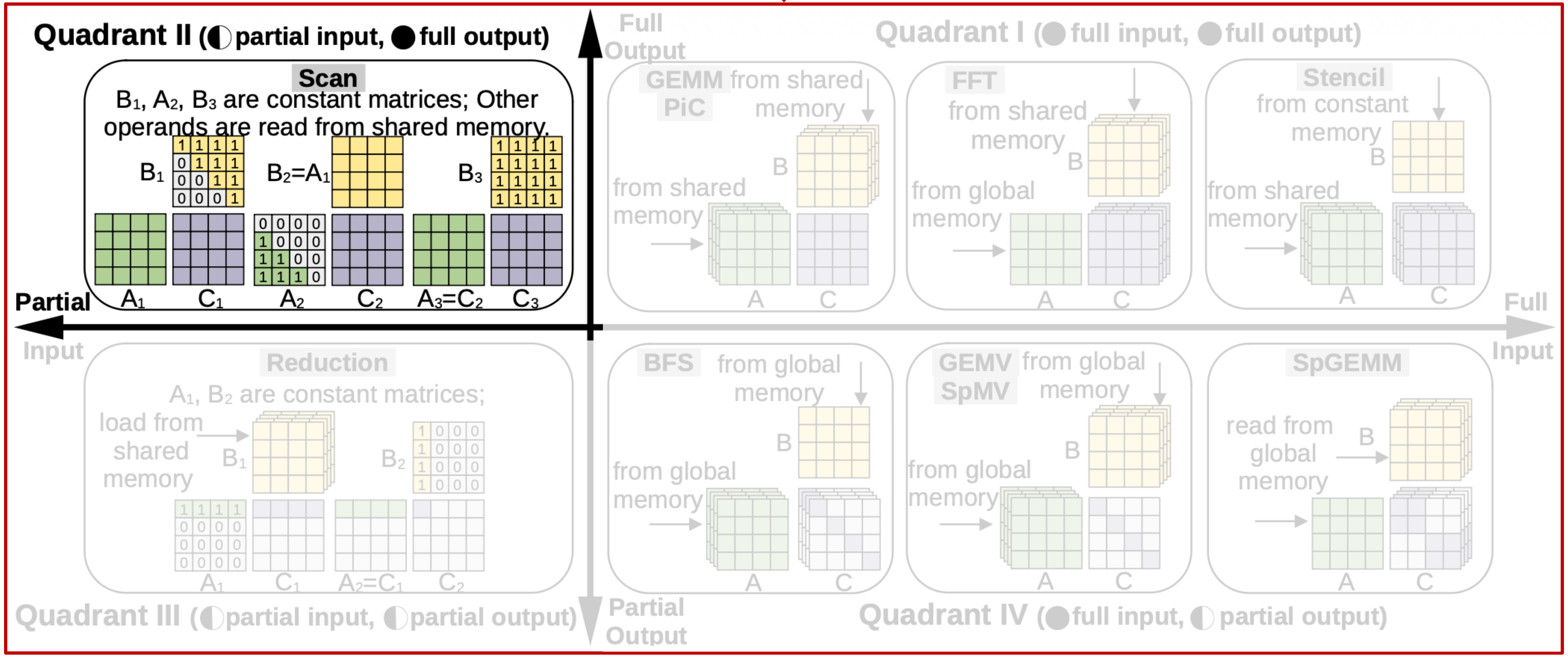
Four Quadrants



Categorization of MMU Utilization Patterns

- **Two dimensions**: Input utilization and Output utilization
- **Two levels**: Full and Partial

Four Quadrants



Quadrant II (◐partial input, ●full output)

Scan: B₁, A₂, B₃ are constant matrices; Other operands are read from shared memory.

1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1

0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

Quadrant I (●full input, ●full output)

GEMM, PiC: B from shared memory; A from shared memory

FFT: B from shared memory; A from global memory

Stencil: B from constant memory; A from shared memory

Quadrant III (◐partial input, ◐partial output)

Reduction: A₁, B₂ are constant matrices; B₁ load from shared memory

1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0

1	1	1	1
0	0	0	0
0	0	0	0
0	0	0	0

Quadrant IV (●full input, ◐partial output)

BFS: B from global memory; A from global memory

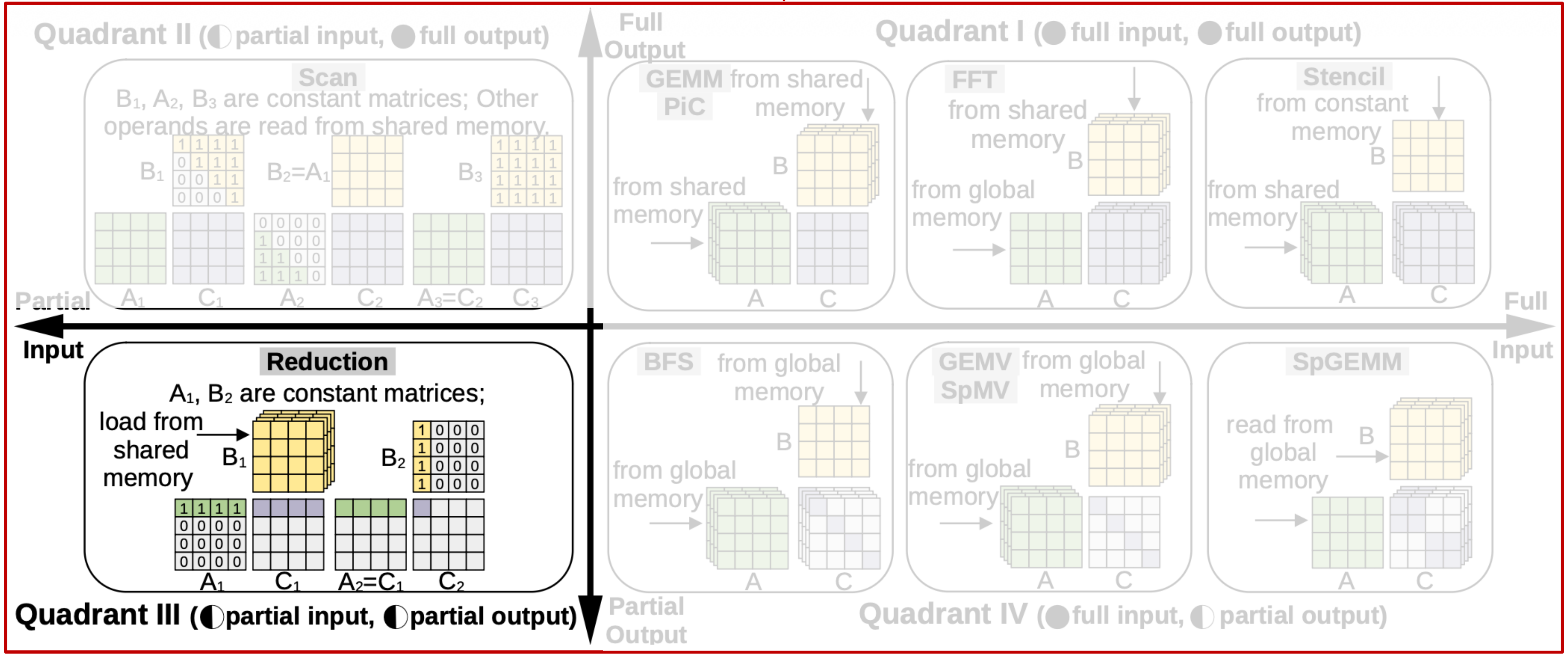
GEMV, SpMV: B from global memory; A from global memory

SpGEMM: read from global memory

Categorization of MMU Utilization Patterns

- **Two dimensions**: Input utilization and Output utilization
- **Two levels**: Full and Partial

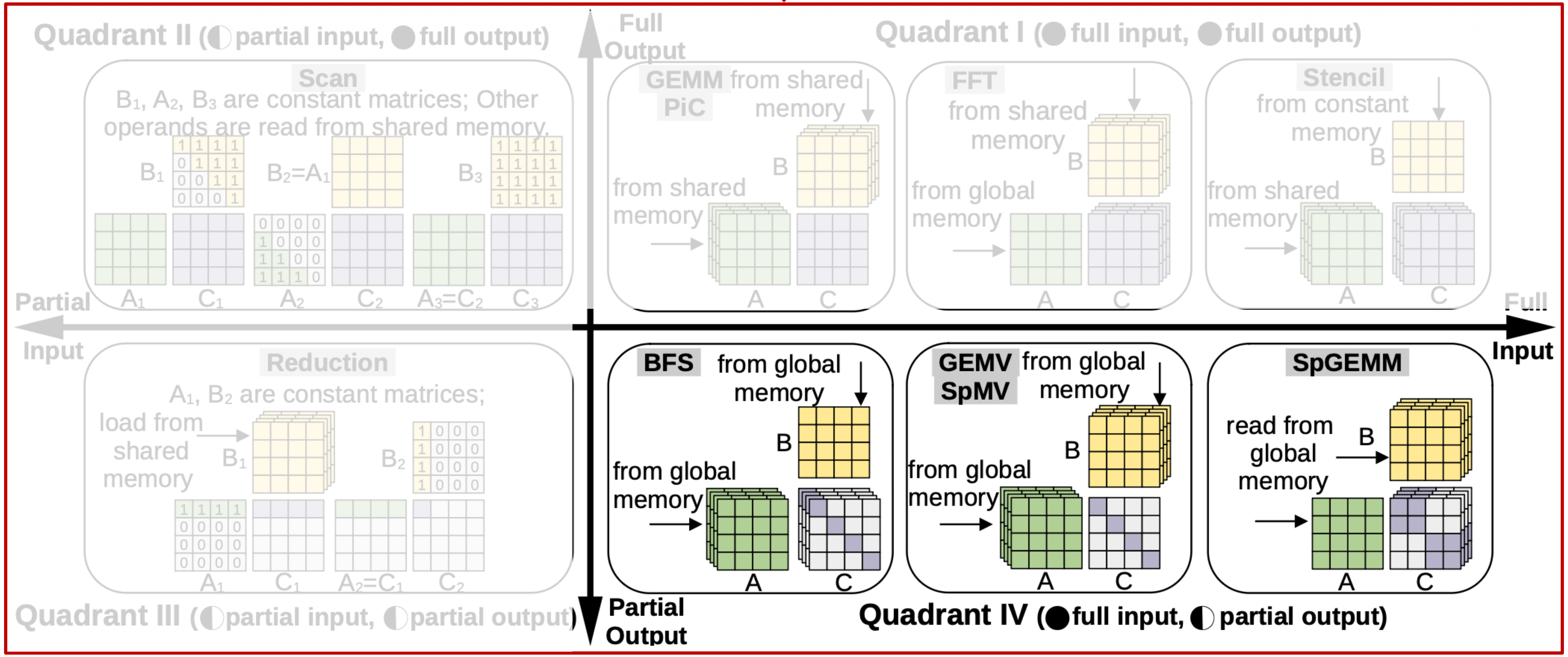
Four Quadrants



Categorization of MMU Utilization Patterns

- **Two dimensions**: Input utilization and Output utilization
- **Two levels**: Full and Partial

Four Quadrants



Key Observation 2: Scientific kernels may not fully utilize the dense input and output matrices of MMUs, exhibiting distinct utilization patterns in four quadrants characterized by varying levels of density.

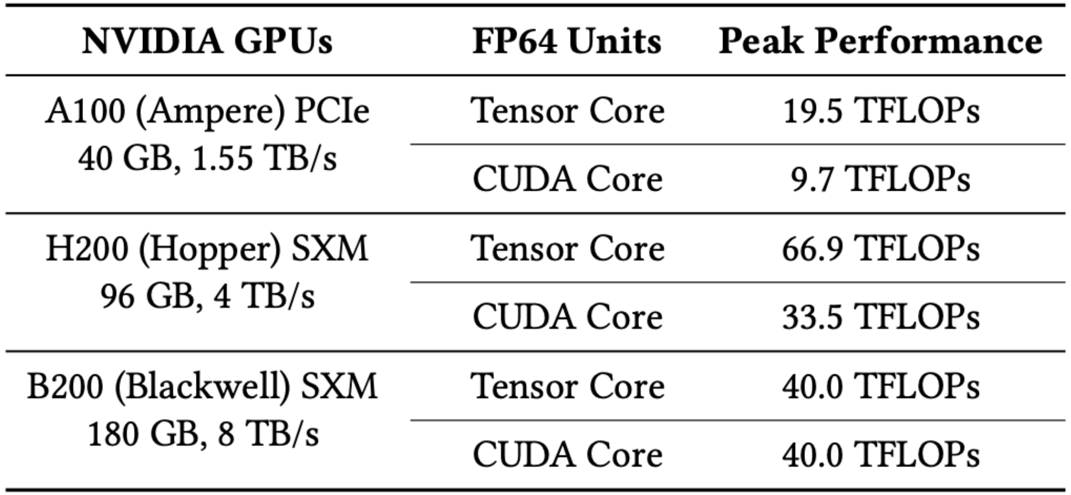
OUTLINE

1. Background and Motivation
2. The Cubie Benchmark Suite
3. Categorization of MMU Utilization Patterns
4. **Experiments**
5. Comparison with other Benchmark Suites
6. Conclusion

Experiments - Setup

- We evaluate Cubie on **NVIDIA A100 (Ampere), H200 (Hopper), and B200 (Blackwell) GPUs,** using five test cases per workload.

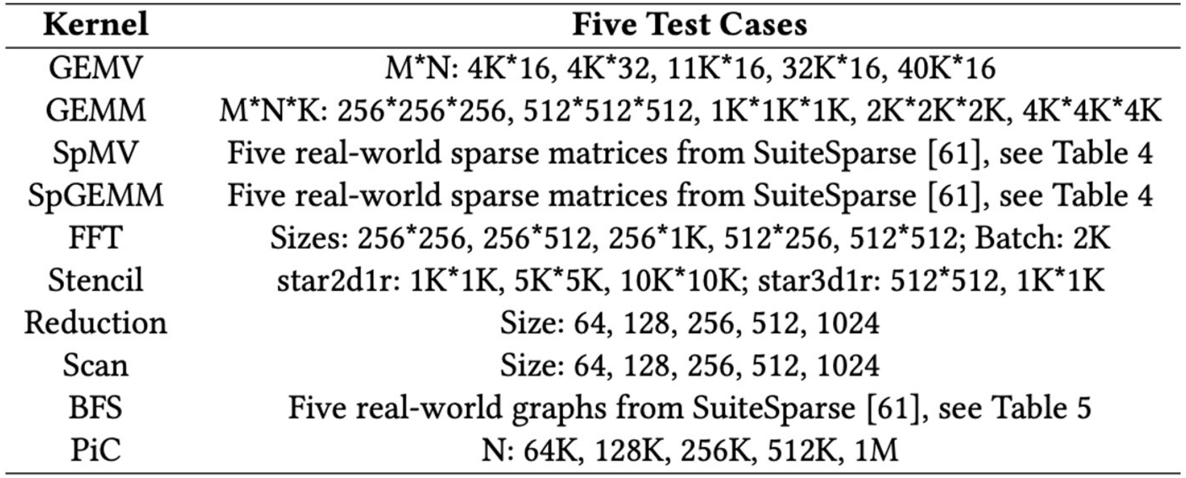
- **Experiments Setup**



NVIDIA GPUs	FP64 Units	Peak Performance
A100 (Ampere) PCIe 40 GB, 1.55 TB/s	Tensor Core	19.5 TFLOPs
	CUDA Core	9.7 TFLOPs
H200 (Hopper) SXM 96 GB, 4 TB/s	Tensor Core	66.9 TFLOPs
	CUDA Core	33.5 TFLOPs
B200 (Blackwell) SXM 180 GB, 8 TB/s	Tensor Core	40.0 TFLOPs
	CUDA Core	40.0 TFLOPs

Specifications of A100, H200, and B200

- **Test Cases**



Kernel	Five Test Cases
GEMV	M*N: 4K*16, 4K*32, 11K*16, 32K*16, 40K*16
GEMM	M*N*K: 256*256*256, 512*512*512, 1K*1K*1K, 2K*2K*2K, 4K*4K*4K
SpMV	Five real-world sparse matrices from SuiteSparse [61], see Table 4
SpGEMM	Five real-world sparse matrices from SuiteSparse [61], see Table 4
FFT	Sizes: 256*256, 256*512, 256*1K, 512*256, 512*512; Batch: 2K
Stencil	star2d1r: 1K*1K, 5K*5K, 10K*10K; star3d1r: 512*512, 1K*1K
Reduction	Size: 64, 128, 256, 512, 1024
Scan	Size: 64, 128, 256, 512, 1024
BFS	Five real-world graphs from SuiteSparse [61], see Table 5
PiC	N: 64K, 128K, 256K, 512K, 1M

Five test cases for each kernel

Experiments - Algorithmic Implementation Variants

- To study performance changes and determine whether they come from **MMU usage** or **algorithm design**, we consider three implementation variants.

Example: GEMV (*Ax*= *y*)

- Tensor Core version (TC)

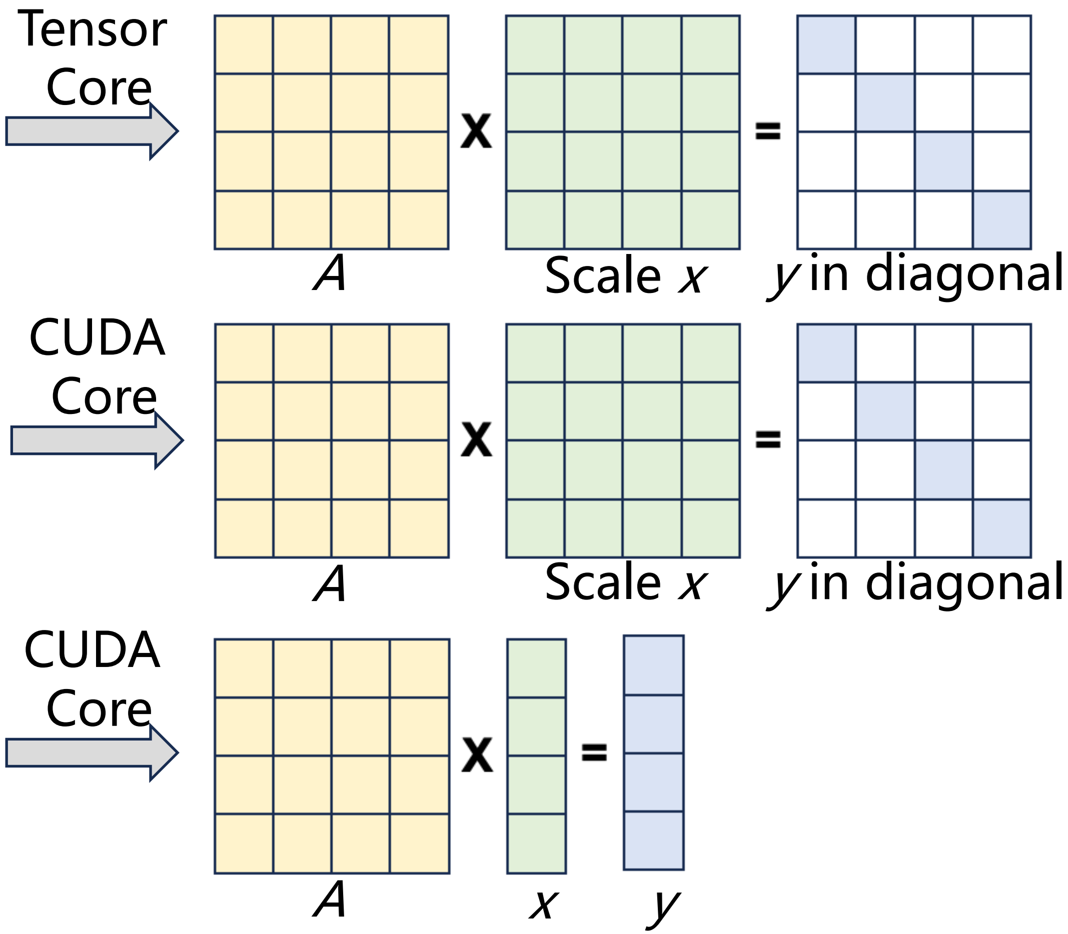
  Uses **Tensor Cores** for computation, calling **FP64 MMA** instructions.

- CUDA Core MMA Replacement (CC)

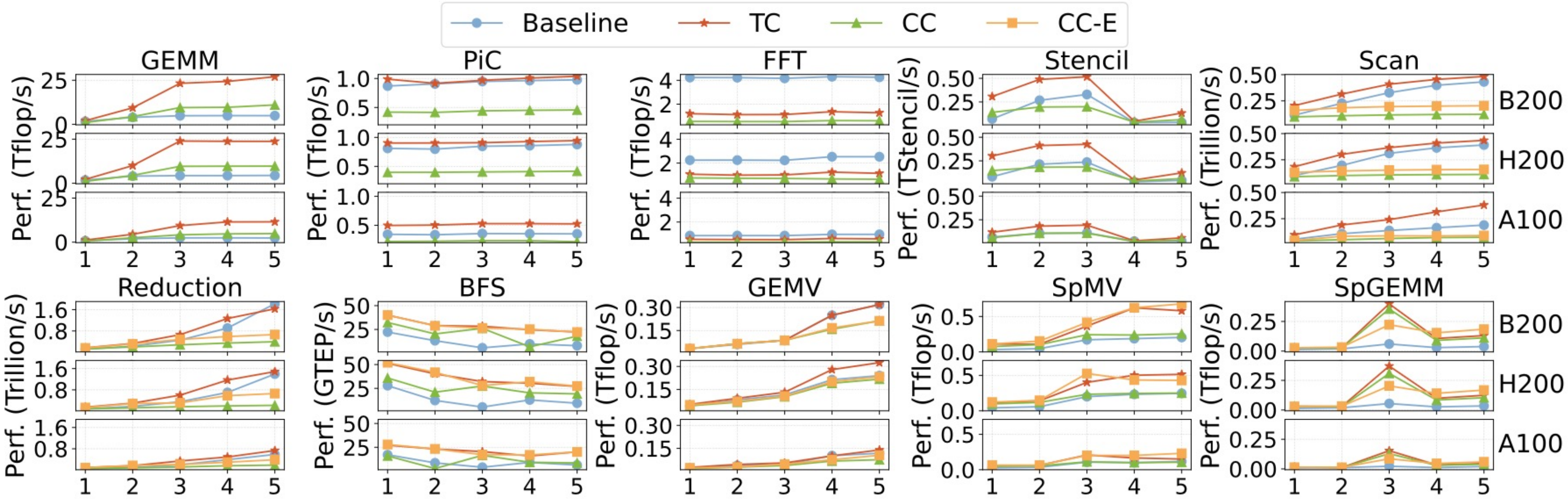
  Keeps the **same data structures and algorithm** as TC, but replaces **MMA** with **CUDA Core** computation.

- CUDA Core Essential Replacement (CC-E)

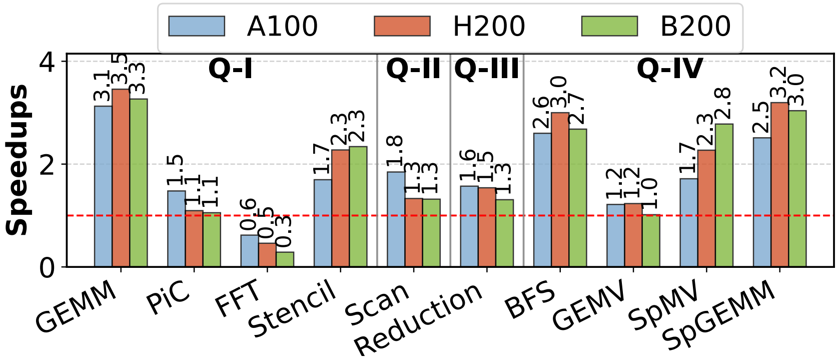
  Computes only the **essential parts** on **CUDA Cores**, removing extra work introduced by MMU mapping.



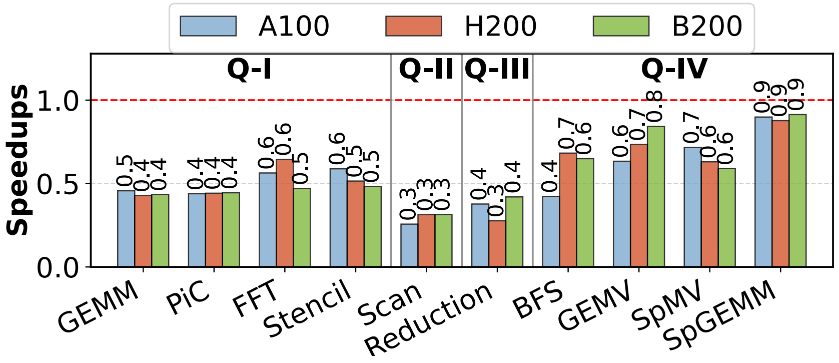
Experiments - Performance



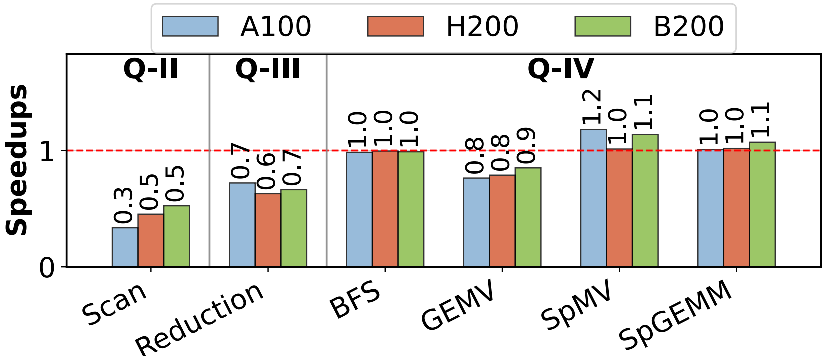
Performance comparison of baselines, TC, CC, and CC-E implementations for all workloads on the three GPUs.



Speedups of TC versions compared to their baselines across all workloads.



Speedups of CC replacements over TC versions across all workloads.



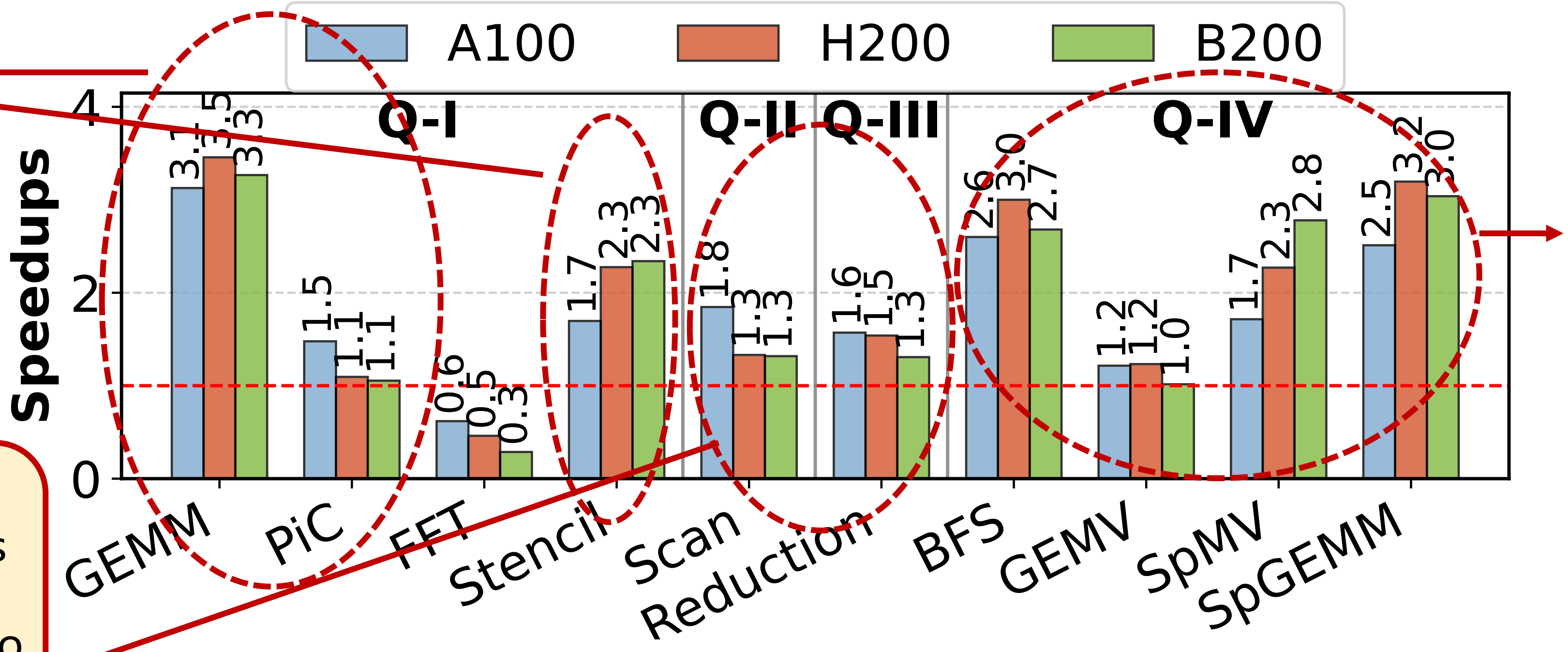
Speedups of CC-E replacements over TC versions across all workloads.

Experiments - Performance

- Do MMU accelerated kernels outperform vector based implementations? → **TC vs. Baseline**

MMA input and output tiles are well utilized. TC show portable speedups across architectures.

Using constant matrices as operands reduces data movement, leading to better performance.



Speedups of TC versions compared to their baselines across all workloads.

With higher memory bandwidth on H200 and B200, TC shows a clear advantage over the baseline.

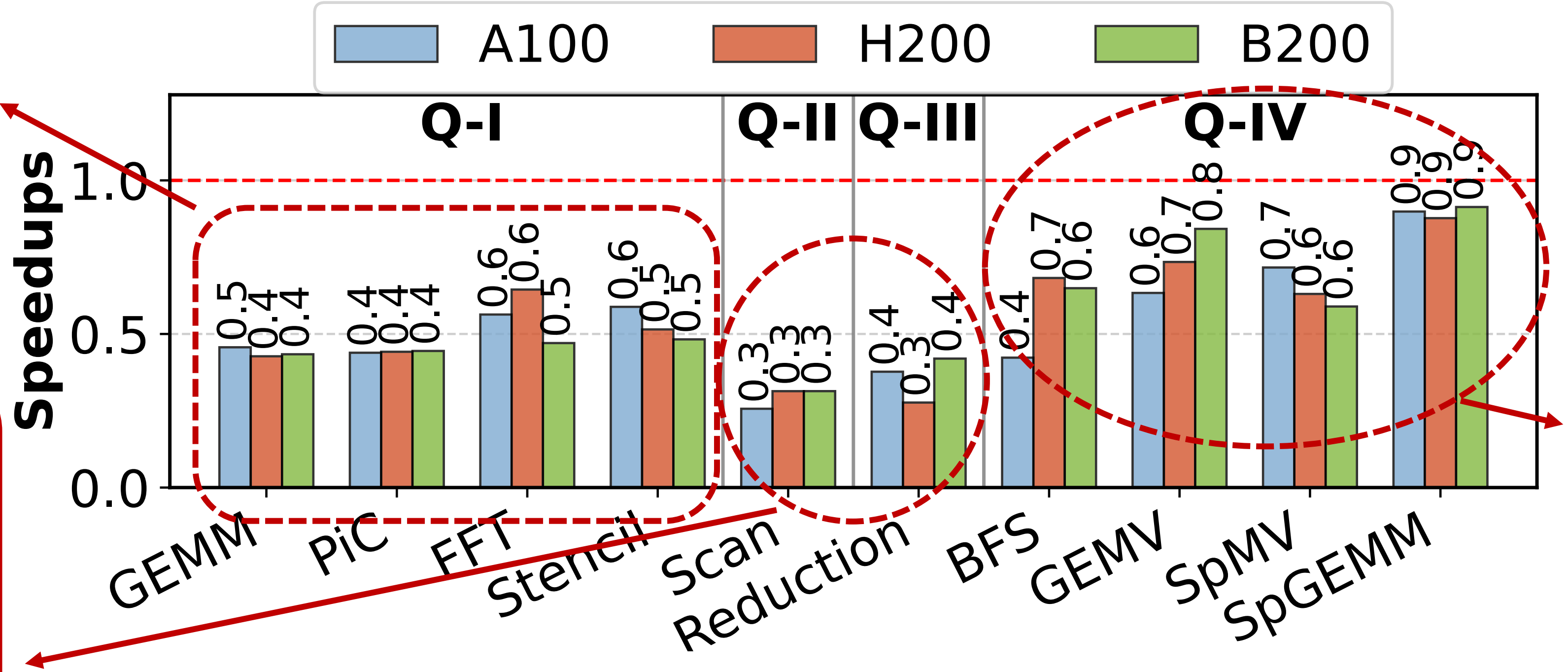
Key Observation 3: MMU-accelerated workloads consistently outperform vector baselines in most cases, and exhibit performance portability across the Ampere, Hopper, and Blackwell architectures.

Experiments - Performance

- With the same data structures and algorithms, how much speedup comes purely from MMU hardware? → **CC vs. TC**

Consistent with peak throughput gaps, CC is ~2× slower than TC.

Beyond peak throughput, CUDA cores cannot exploit constant matrices as effectively as Tensor Cores.



Speedups of CC replacements over TC versions across all workloads.

The gap is smaller, but TC still wins: TC can benefit bandwidth bound kernels.

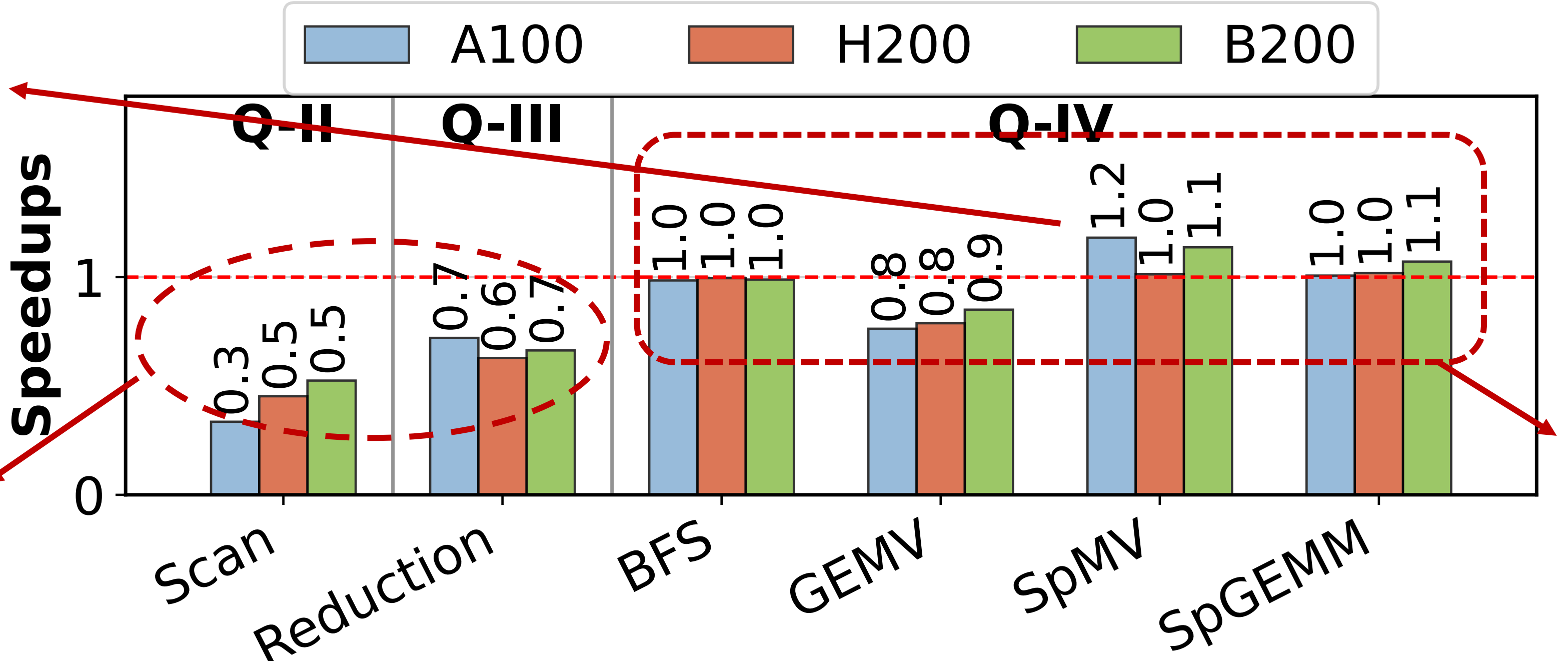
Key Observation 4: Removing the impact of data structures and algorithms (replacing MMU instructions with equivalent vector unit operations), MMUs account for 10% to 200% of the performance gains.

Experiments - Performance

- Is the redundant work introduced for MMU mapping worth it? Would vector units be faster after removing it? → **CC-E vs. TC**

For SpMV, after removing redundancy, CC-E can be further improved.

TC is still faster overall, so the redundancy is generally worthwhile.



Speedups of CC-E replacements over TC versions across all workloads.

For most kernels, CC-E is close to TC. Since TC also beats the baseline and CC, the introduced redundancy is usually justified.

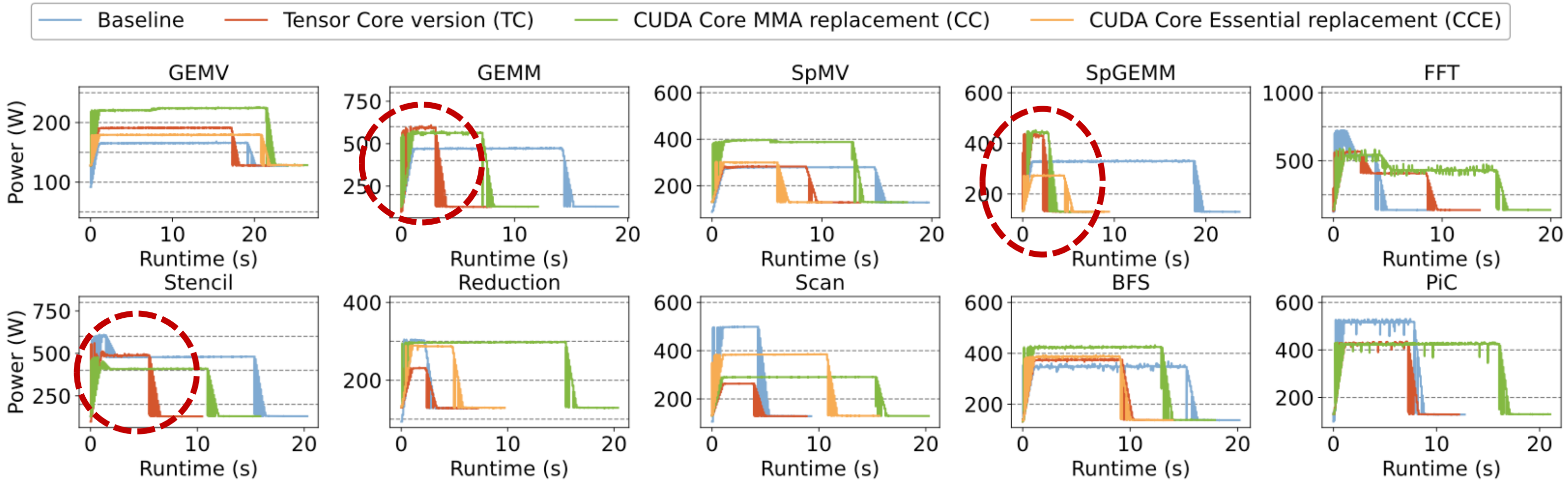
Key Observation 5: Generally, the redundant computations introduced to enable MMU-friendly matrix computing patterns should not be removed. The only exception is SpMV, where avoiding the redundancy yields up to 20% higher performance.

Experiments - Power and Energy

$EDP = Average\ Power \times Execution\ Time^2$

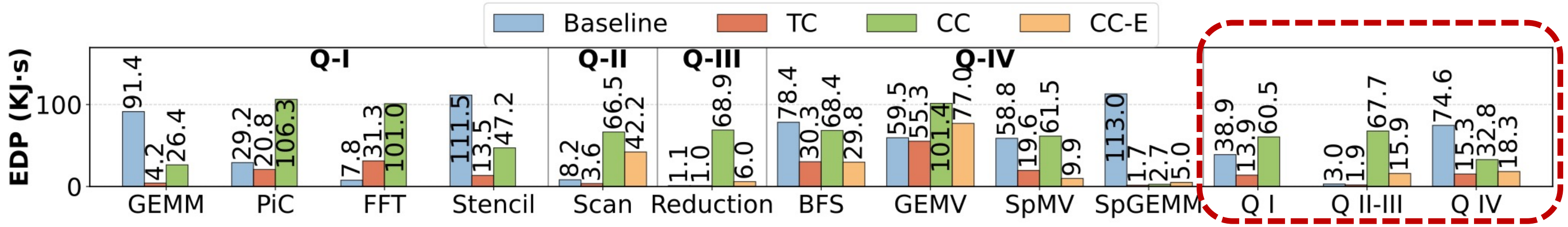
- We measure **power**, **energy**, and **EDP** (Energy–Delay Product, **lower is better**) for each workload on H200.

Instantaneous power of TC can be similar to CC.



Power consumption over time of baselines and three implementations for all workloads on H200.

But TC can finish faster, so energy and EDP are lower overall.



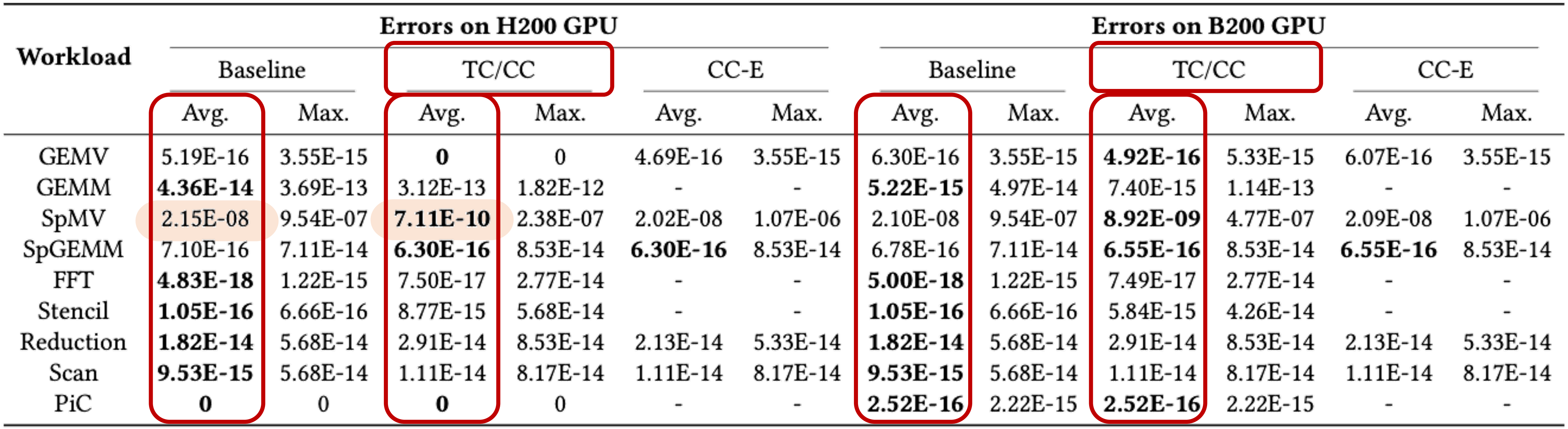
The EDP comparison of baselines, TC, CC, and CC-E implementations for all workloads on H200.

Key Observation 6: MMUs exhibit similar power consumption to vector units but complete computations significantly faster, resulting in 30% to 80% lower geomean EDP across all workloads.

Experiments - Numerical Accuracy

- We measure FP64 numerical errors on H200 and B200, using the serial CPU results as the reference.

TC and CC show identical average and maximum errors.



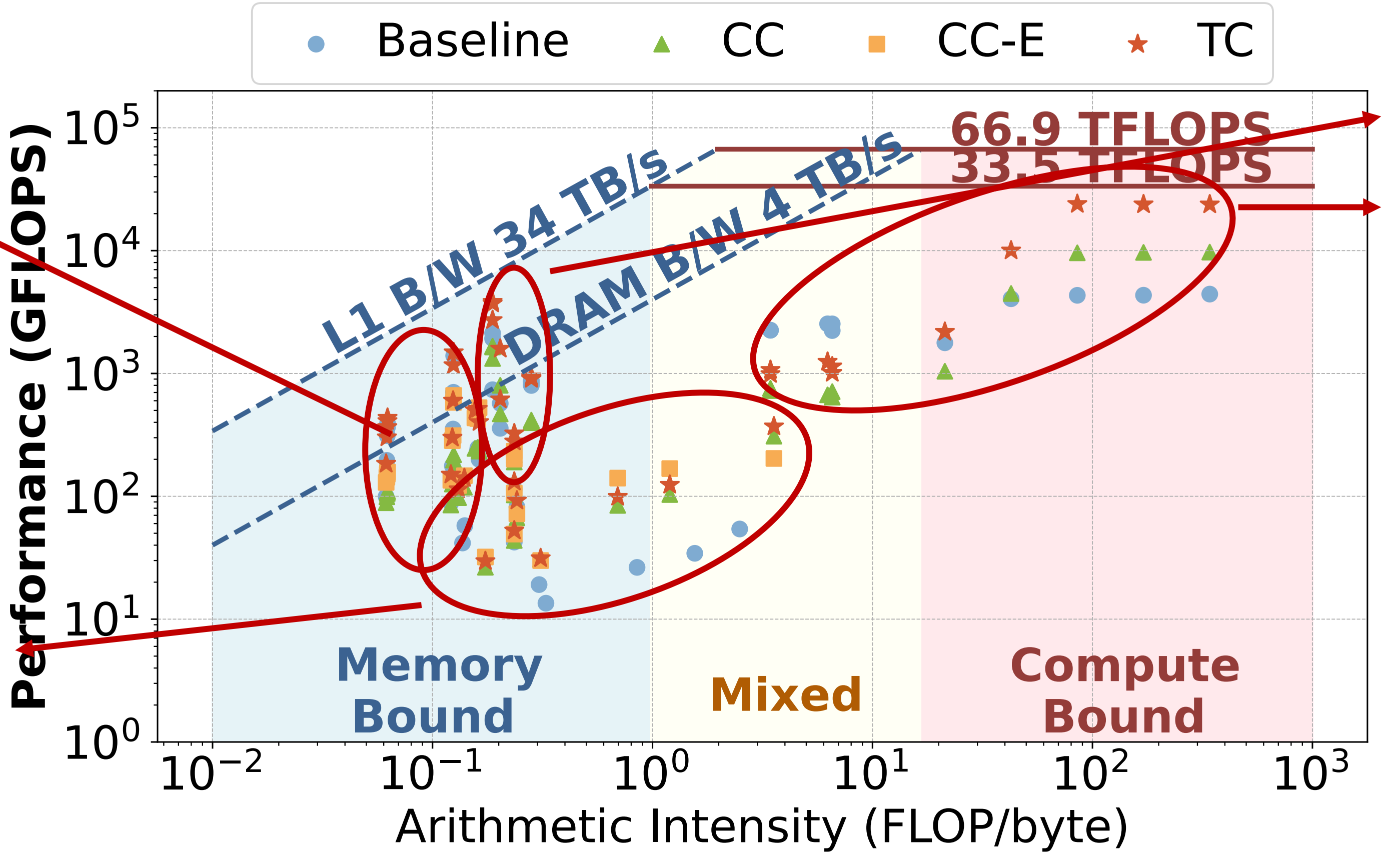
Workload	Errors on H200 GPU						Errors on B200 GPU					
	Baseline		TC/CC		CC-E		Baseline		TC/CC		CC-E	
	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
GEMV	5.19E-16	3.55E-15	**0**	0	4.69E-16	3.55E-15	6.30E-16	3.55E-15	**4.92E-16**	5.33E-15	6.07E-16	3.55E-15
GEMM	**4.36E-14**	3.69E-13	3.12E-13	1.82E-12	-	-	**5.22E-15**	4.97E-14	7.40E-15	1.14E-13	-	-
SpMV	2.15E-08	9.54E-07	**7.11E-10**	2.38E-07	2.02E-08	1.07E-06	2.10E-08	9.54E-07	**8.92E-09**	4.77E-07	2.09E-08	1.07E-06
SpGEMM	7.10E-16	7.11E-14	**6.30E-16**	8.53E-14	**6.30E-16**	8.53E-14	6.78E-16	7.11E-14	**6.55E-16**	8.53E-14	**6.55E-16**	8.53E-14
FFT	**4.83E-18**	1.22E-15	7.50E-17	2.77E-14	-	-	**5.00E-18**	1.22E-15	7.49E-17	2.77E-14	-	-
Stencil	**1.05E-16**	6.66E-16	8.77E-15	5.68E-14	-	-	**1.05E-16**	6.66E-16	5.84E-15	4.26E-14	-	-
Reduction	**1.82E-14**	5.68E-14	2.91E-14	8.53E-14	2.13E-14	5.33E-14	**1.82E-14**	5.68E-14	2.91E-14	8.53E-14	2.13E-14	5.33E-14
Scan	**9.53E-15**	5.68E-14	1.11E-14	8.17E-14	1.11E-14	8.17E-14	**9.53E-15**	5.68E-14	1.11E-14	8.17E-14	1.11E-14	8.17E-14
PiC	**0**	0	**0**	0	-	-	**2.52E-16**	2.22E-15	**2.52E-16**	2.22E-15	-	-

Errors can vary from Baseline to TC/CC, sometimes by more than one order of magnitude.

Key Observation 7: MMUs and vector units provide comparable numerical accuracy, but algorithmic transformations for MMU utilization can induce significant numerical deviations that undermine the reproducibility of scientific results.

Experiments - Performance Model

- Cache-aware roofline model



In Q II–III, Reduction and Scan use segment processing and are cache friendly, so TC can even exceed the DRAM bandwidth roofline.

In Q-IV, TC, CC, and CC-E change memory access patterns and get closer to the bandwidth roofline than the baseline (blue dot).

In Q-I, FFT and GEMM have higher arithmetic intensity, while Stencil and PiC are lower. Without advanced GEMM optimizations, TC GEMM does not reach peak throughput.

Key Observation 8: Adapting data layouts and algorithms for MMUs fundamentally alters memory access patterns, often yielding more regular access and significant performance gains.

OUTLINE

1 Background and Motivation

2 The Cubie Benchmark Suite

3 Categorization of MMU Utilization Patterns

4 Experiments

5 Comparison with other Benchmark Suites

6 Conclusion

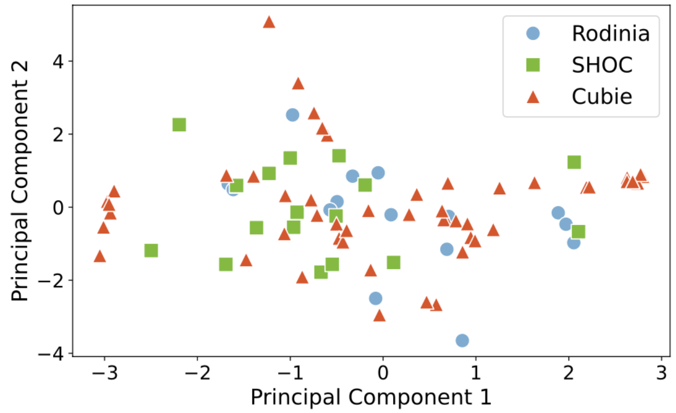
Comparison with other Benchmark Suites

- Compared with Rodinia and SHOC: Cubie covers **more Berkeley Dwarfs** and offers **broader characterization.**

Dwarf / Feature	Rodinia		SHOC		Cubie	
	[44]		[59]		(this work)	
Dense linear algebra	3		2		2	
Sparse linear algebra	-		-		2	
Spectral methods	-	5	1	5	1	7
N-Body	-		1		1	
Structured grids	4		1		1	
Unstructured grids	2		-		-	
MapReduce	-		3		2	
Graph traversal	2		-		1	
Dynamic programming	1		-		-	
Parallelization pattern	✓				✓	
Performance	✓		✓		✓	
Power and energy	✓	4	✓	4	✓	5
Precision					✓	
Memory bandwidth			✓		✓	
CPU-GPU data transfer	✓		✓			

- We collect the following NCU metrics and run PCA. Cubie workloads show a **wider spread** in the principal component space.

Metric Name in NCU	Description
gpu__dram_throughput	global mem. throughput
l1tex__t_sector_hit_rate	L1 cache hit rate
lts__t_sector_hit_rate	L2 cache hit rate
l1tex__data_bank_conflicts_pipe_lsu_mem_shared	shared mem. bank conflicts
sm__inst_executed.avg.per_cycle_active	inst. per cycle
sm__inst_executed_pipe_lsu	inst. by lsu pipes
sm__inst_executed_pipe_fma	inst. by fma pipes
sm__inst_executed_pipe_tensor	inst. by tensor pipes
sm__pipe_tensor_cycles_active	tensor active cycles



Key Observation 9: Originally developed with the primary goal of evaluating MMUs, the Cubie benchmark suite encompasses a wide range of behaviors in scientific programs, positioning it as an effective tool for assessing modern processors.

OUTLINE

1 Background and Motivation

2 The Cubie Benchmark Suite

3 Categorization of MMU Utilization Patterns

4 Experiments

5 Comparison with other Benchmark Suites

6 Conclusion

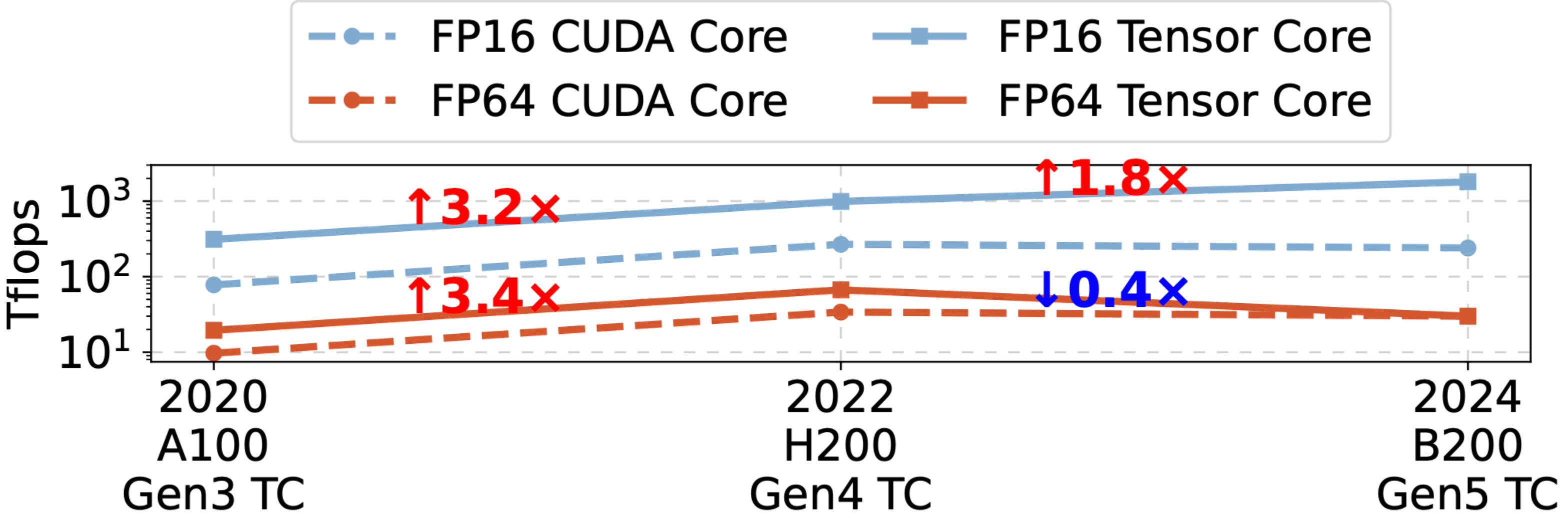
Conclusion

- We present **Cubie**, a benchmark suite of **MMU optimized scientific kernels**. Cubie covers diverse parallel patterns and kernel behaviors, and evaluates **performance**, **power**, and **numerical accuracy**, providing practical insights for architecture, algorithm, and application researchers.

Concerns	Arch.	Alg.	App.	Observations
Compute Patterns	✓	✓		O1、O2
Performance Portability		✓	✓	O3
Necessity of MMUs	✓	✓		O4、O5
Power and Energy	✓		✓	O6
Numerical Precision	✓	✓	✓	O7
Memory	✓	✓		O8
Workload Diversity	✓		✓	O9

Concerns and corresponding observations for architecture, algorithm, and application researchers.

A Call for Preserving FP64 MMU Capability



FP16 TC: continues to grow across A100 → H200 → B200.

FP64 TC: drops on B200.

- **Our results show FP64 MMU acceleration benefits most scientific workloads.**
- **Future GPUs should KEEP FP64 MMUs as a core capability!**

Thanks for Listening!
Any Questions?

Characterizing Matrix Multiplication Units across General Parallel Patterns in Scientific Computing

Artifacts Available acm V1.1

Artifacts Evaluated acm Reusable V1.1

Results Reproduced acm V1.1

Yuechen Lu[1]， Hongwei Zeng[1]， Marc Casas[2]， Weifeng Liu[1]

[1] China University of Petroleum- Beijing, China

[2] Barcelona Supercomputing Center, Spain

Sydney, Australia · Feb 4, 2026

Code: https://doi.org/10.5281/zenodo.15290623