Efficient Arbitrary Precision Acceleration for Large Language Models on GPU Tensor Cores

Shaobo Ma, Chao Fang, Haikuo Shao, Zhongfeng Wang

ICAIS Lab, Nanjing University, China

Jan 23, 2025

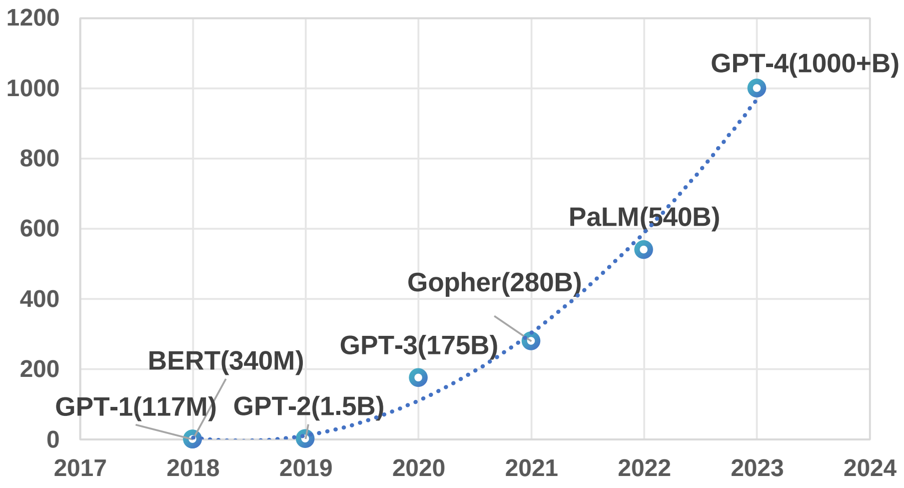
Outlines

01

Background & Motivation

1.1.1 Background: Quantization of LLMs

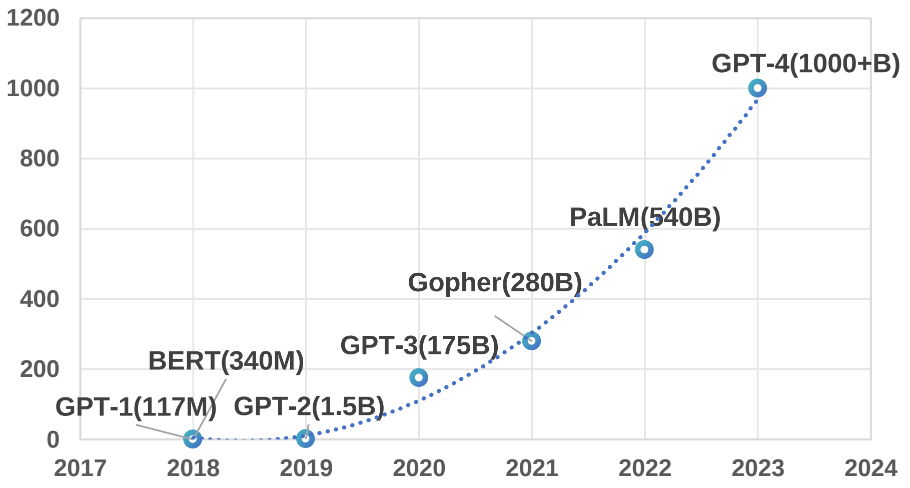
- **Challenges** Brought by the Growth in Size of LLMs
  - More memory (storage)
  - More computational power and time (inference)



Growth in Size of Transformer Models

1.1.1 Background: Quantization of LLMs

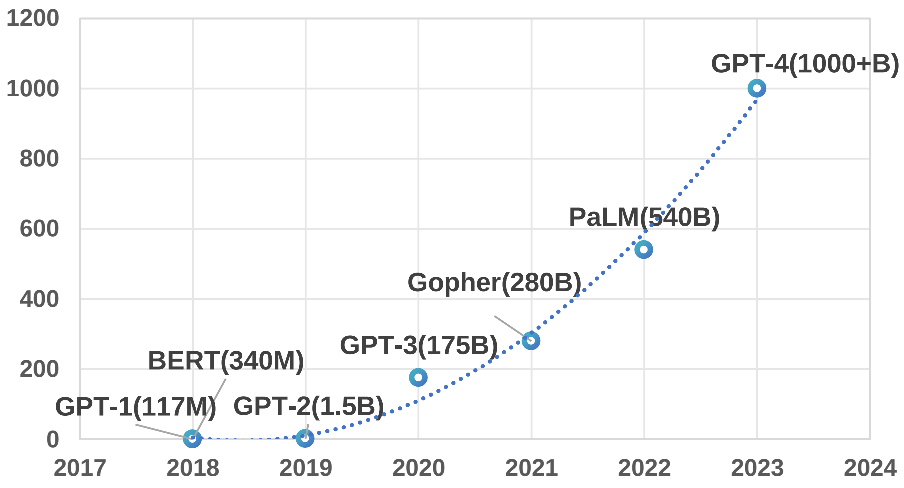
- **Challenges** Brought by the Growth in Size of LLMs
  - ◆ More memory (storage)
  - ◆ More computational power and time (inference)
- One Effective Method——**Model quantization**
  - ◆ Storage requirement
  - ◆ Computational overhead



Growth in Size of Transformer Models

1.1.1 Background: Quantization of LLMs

- **Challenges** Brought by the Growth in Size of LLMs
  - ◆ More memory (storage)
  - ◆ More computational power and time (inference)
- One Effective Method——**Model quantization**
  - ◆ Storage requirement
  - ◆ Computational overhead
- **Quantization Works**
  - ◆ GPTQ (3-4bit) [1]
  - ◆ TSLD (2bit) [2]
  - ◆ OneBit (1bit) [3]



Growth in Size of Transformer Models

Models	FP16 (GB)	GPTQ 3bit (GB)	TSLD (GB)	OneBit (GB)
LLaMA-7B	13.5	2.5	1.7	1.3
LLaMA-13B	26.0	4.9	3.3	2.2
LLaMA-30B	65.1	12.2	8.1	4.9
LLaMA-65B	130.6	24.5	16.3	9.2

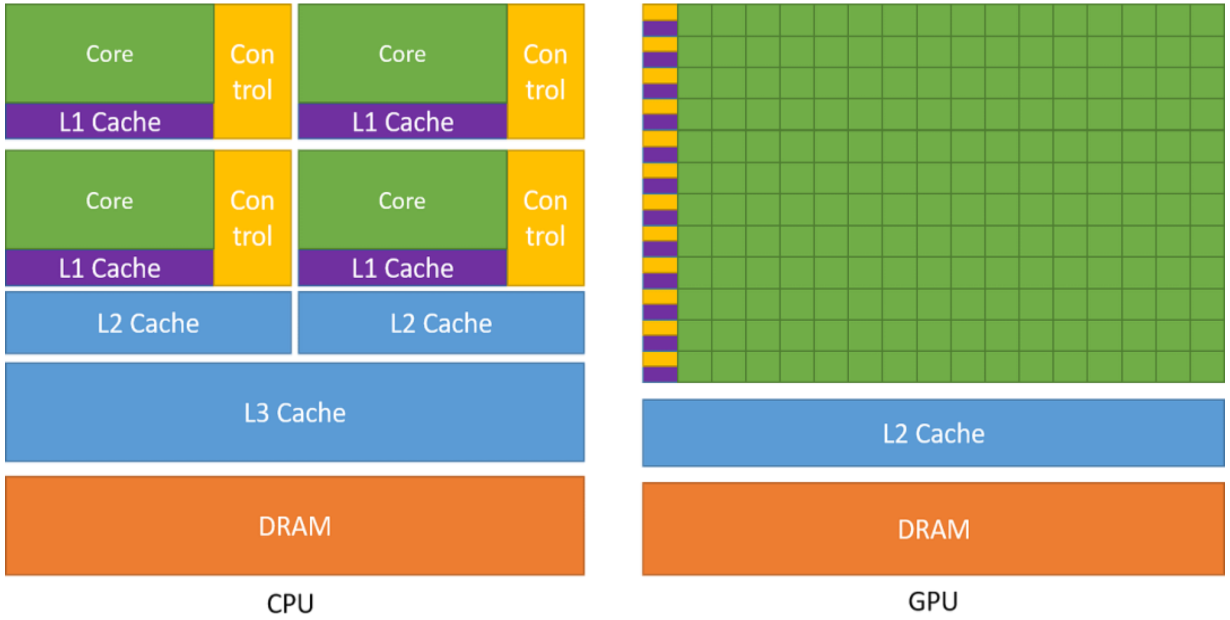
Storage Reduction Brought by Model Quantization

[1] Frantar, Elias, et al. "Gptq: Accurate post-training quantization for generative pre-trained transformers." arXiv preprint arXiv:2210.17323 (2022).
[2] Kim, Minsoo, et al. "Token-scaled logit distillation for ternary weight generative language models." Advances in Neural Information Processing Systems 36 (2024).
[3] Xu, Yuzhuang, et al. "OneBit: Towards Extremely Low-bit Large Language Models." arXiv preprint arXiv:2402.11295 (2024).

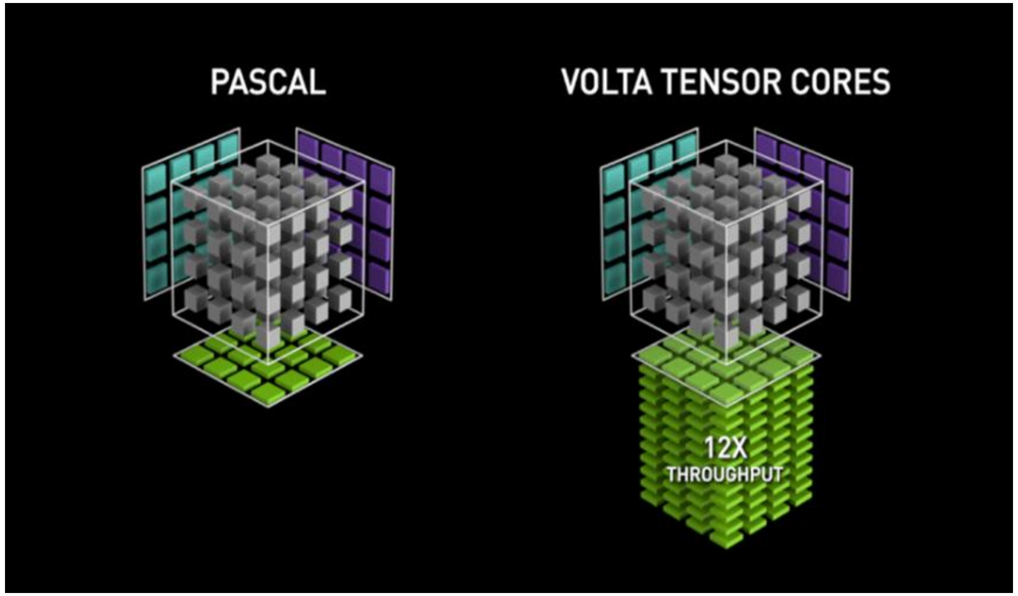
1.1.2 Background: GPU and Tensor Core

- **GPU:** Graphics Processing Unit
  - ◆ Highly parrallel computing architecture
  - ◆ Multi-level memory hierarchy

- **Tensor Core (TC):** Specialized Processing Unit
  - ◆ Optimized for matrix operations
  - ◆ Low-precision computing



Comparation Between CPU and GPU Architecture



Tensor Core Acceleration of Matrix Multiplication

1.2.1 Motivation: Limited Data Format

- **Problem:** Limited Data Format Support in GPU and TC
  - ◆ Mismatch with the quantized data format (INT2 [1, 2] / INT3 [3, 4])

	Supported CUDA Core Precisions									Supported Tensor Core Precisions								
	FP8	**FP16**	**FP32**	**FP64**	**INT1**	**INT4**	**INT8**	**TF32**	**BF16**	**FP8**	**FP16**	**FP32**	**FP64**	**INT1**	**INT4**	**INT8**	**TF32**	**BF16**
NVIDIA Tesla P4	No	No	Yes	Yes	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No
NVIDIA P100	No	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No
NVIDIA Volta	No	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No	No
NVIDIA Turing	No	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes	Yes	No	No
NVIDIA A100	No	Yes	Yes	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
NVIDIA H100	No	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes

Modern NVIDIA GPU Precision Support

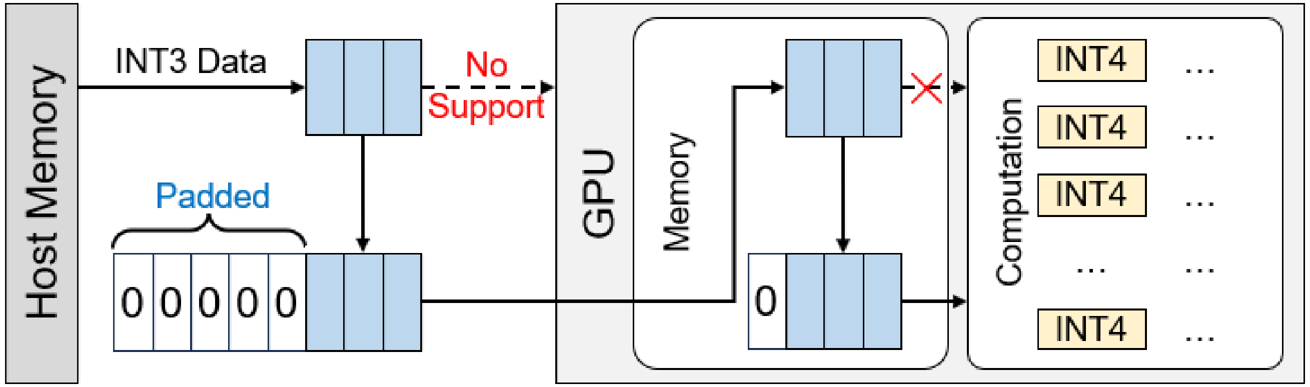
[1] Kim, Minsoo, et al. "Token-scaled logit distillation for ternary weight generative language models." Advances in Neural Information Processing Systems 36 (2024).
[2]Chen, Mengzhao, et al. "Efficientqat: Efficient quantization-aware training for large language models." arXiv preprint arXiv:2407.11062 (2024).
[3] Frantar, Elias, et al. "Gptq: Accurate post-training quantization for generative pre-trained transformers." arXiv preprint arXiv:2210.17323 (2022).
[4] Lin, Ji, et al. "AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration." Proceedings of Machine Learning and Systems 6 (2024): 87-100.

1.2.1 Motivation: Limited Data Format

- **Problem:** Limited Data Format Support in GPU and TC
  - ◆ Mismatch with the quantized data format (INT2 [1, 2] / INT3 [3, 4])

- Current approach: computation by padding to higher-bit data format

Extra computation and memory overhead ☹

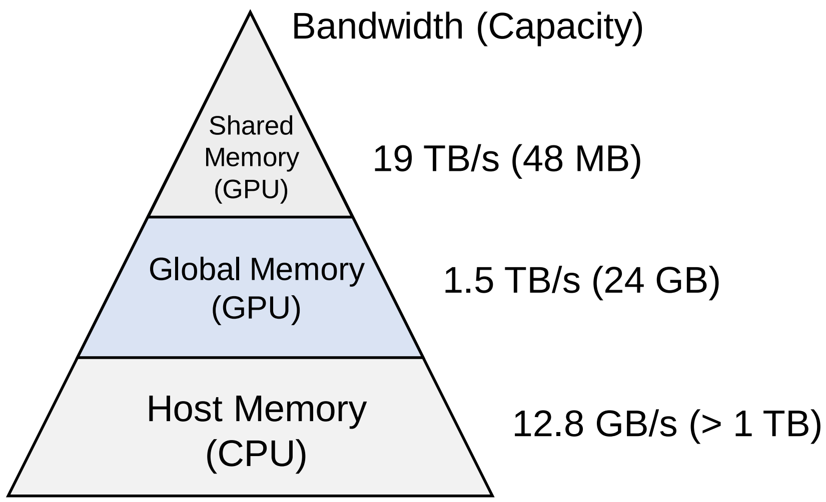


GPU Computation with Limited Data Format Support

1.2.2 Motivation: Inefficient Memory Management

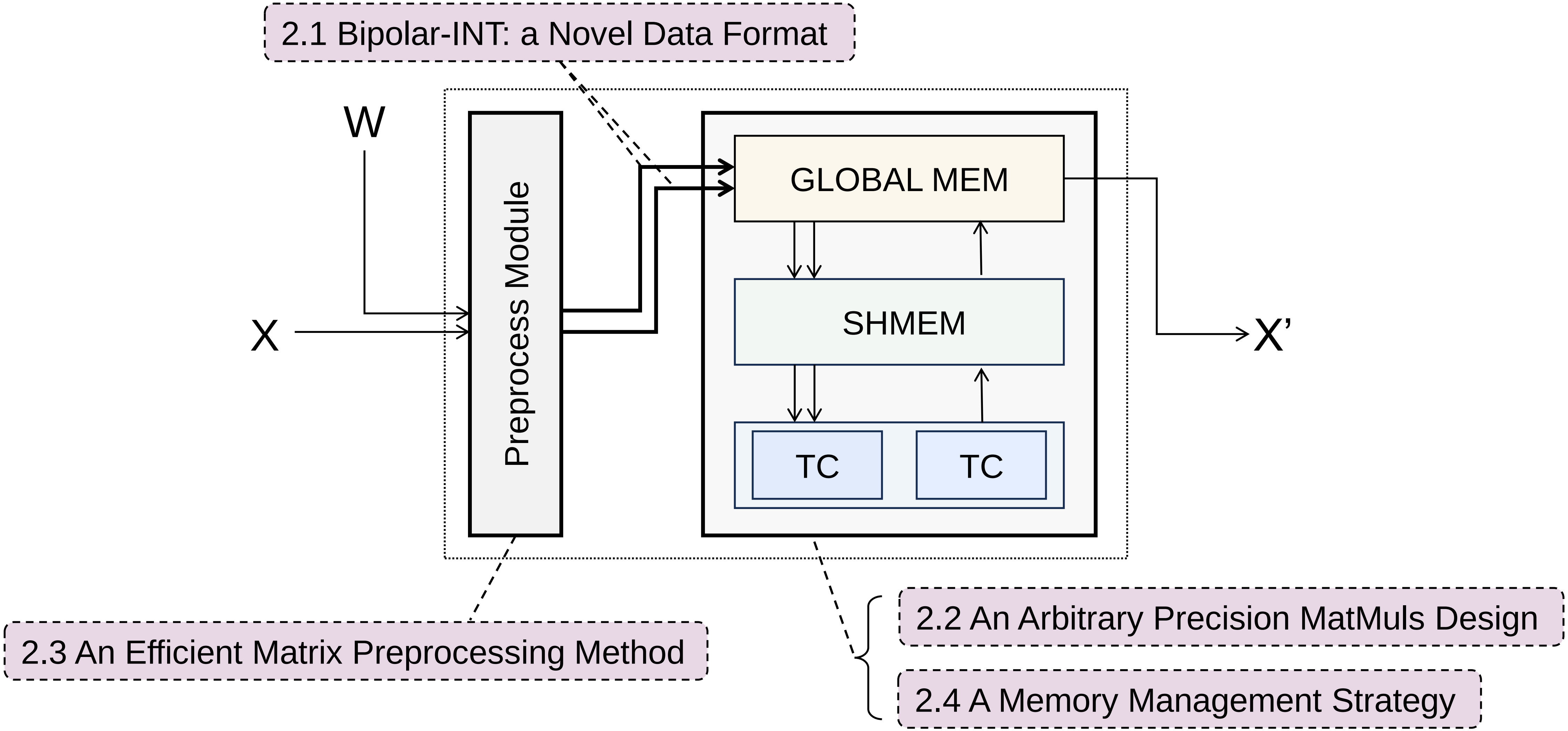
- **Characteristics** of Different Levels of Storage
  - ◆ More memory, slower speed
  - ◆ Smaller range, faster speed

- **Disadvantages** of Direct Memory Mangement
  - ◆ Inefficient memory transfer
  - ◆ Slow global memory access
  - ◆ Threads contend for shared memory



Comparison of Memory Bandwidth and Capacity

1.3 Our Contributions



02

Our Works

2.1 Bipolar-INT Data Format

- Interpret **"0"** as **"-1"** in calculation
  - Example

Unsigned INT: $5 = 0 + 4 + 0 + 1 \Longrightarrow$ 0 1 0 1

Bipolar-INT: $5 = 8 - 4 + 2 - 1 \Longrightarrow$ 1 0 1 0

(-1) (-1)

  - Range

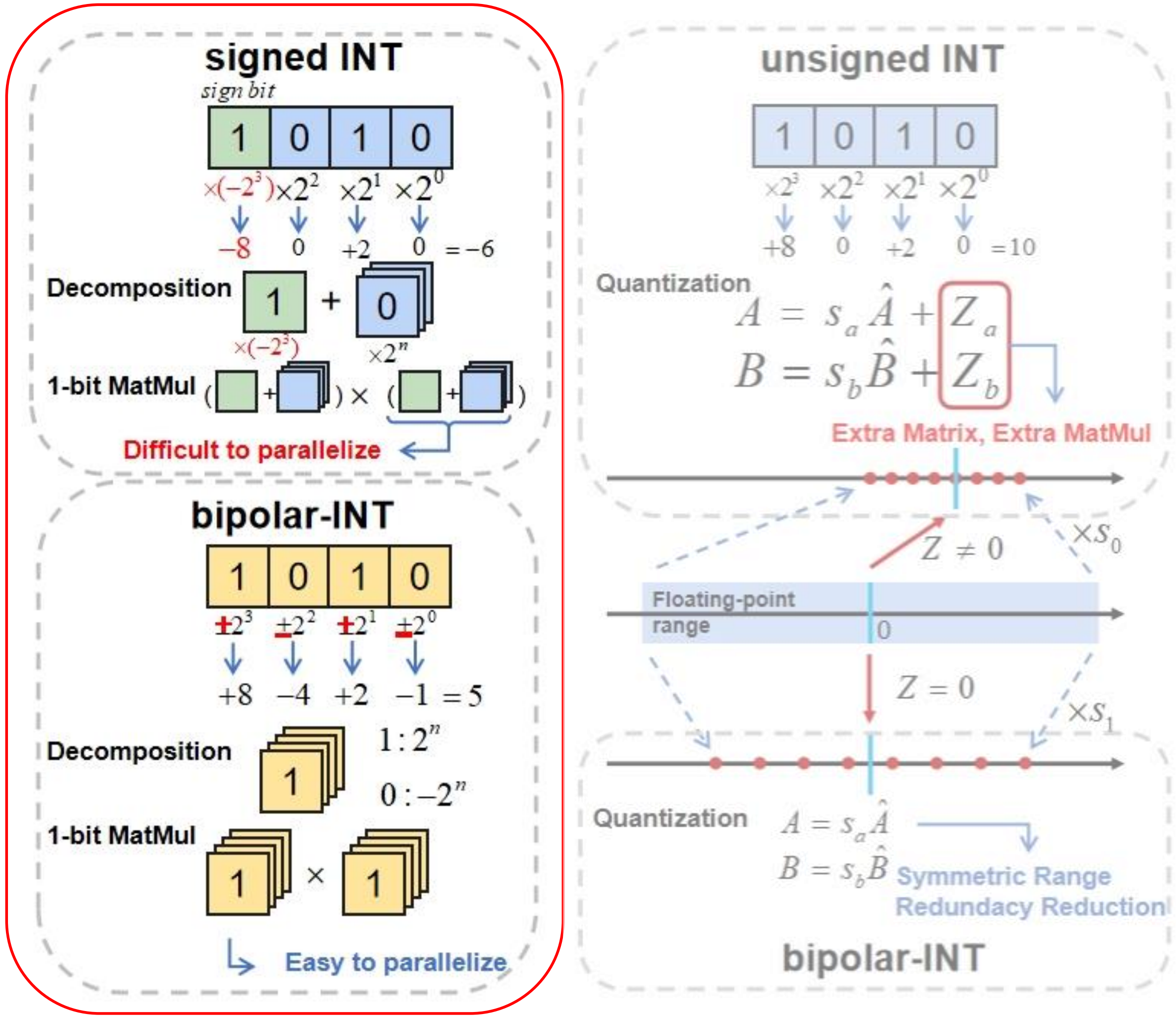
$-2^n \sim +2^n$ **All Odd Numbers**

0 0 0 0 — $-8-4-2-1=-15$

1 1 1 1 — $8+4+2+1=15$

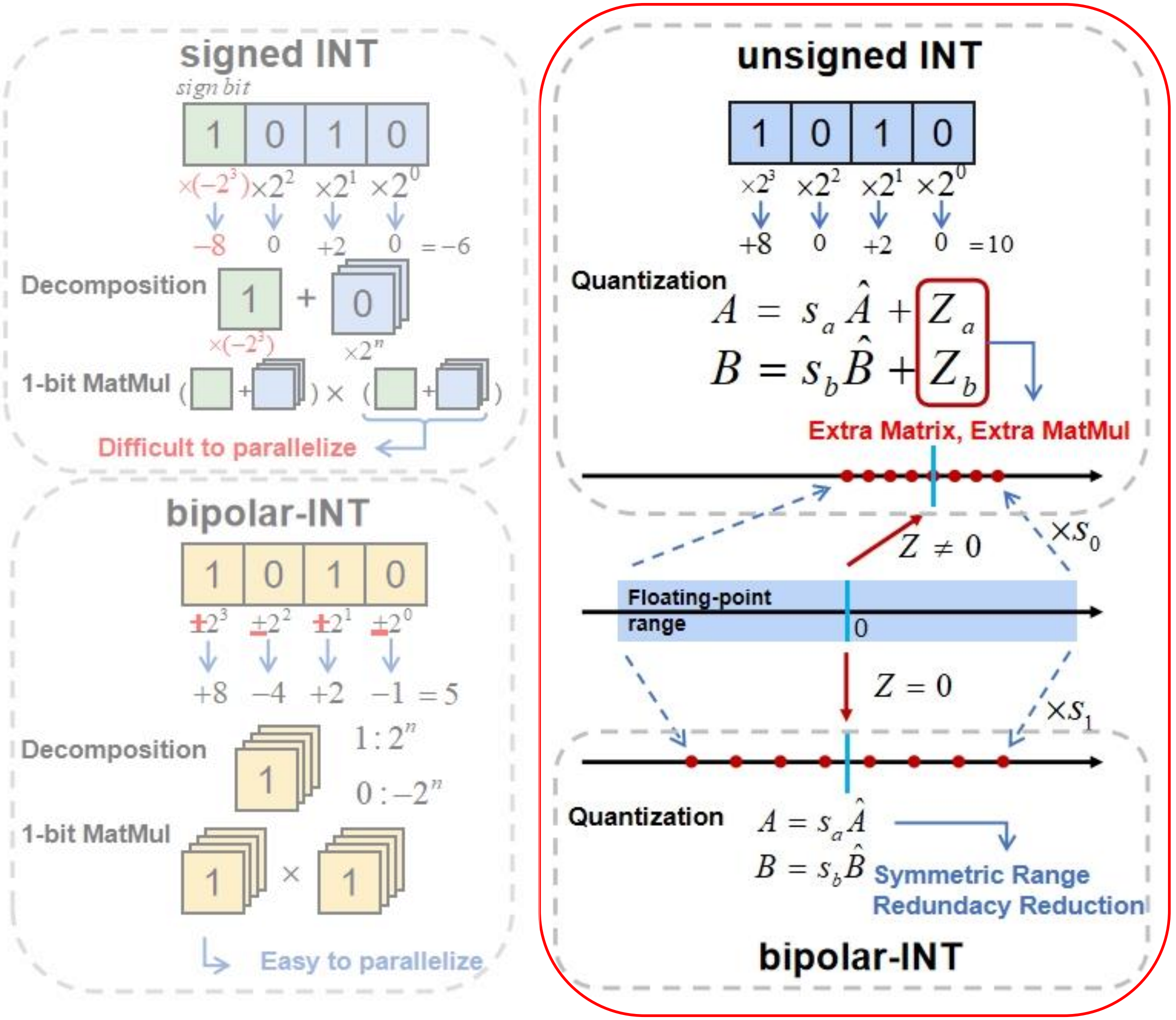
2.1.1 Comparison with Signed INT

- Compared with **Signed INT**
  - ◆ Without sign bit
  - ◆ Easy to parallelize
- Compared with Unsigned INT
  - ◆ Symmetric range
  - ◆ Redundancy Reduction



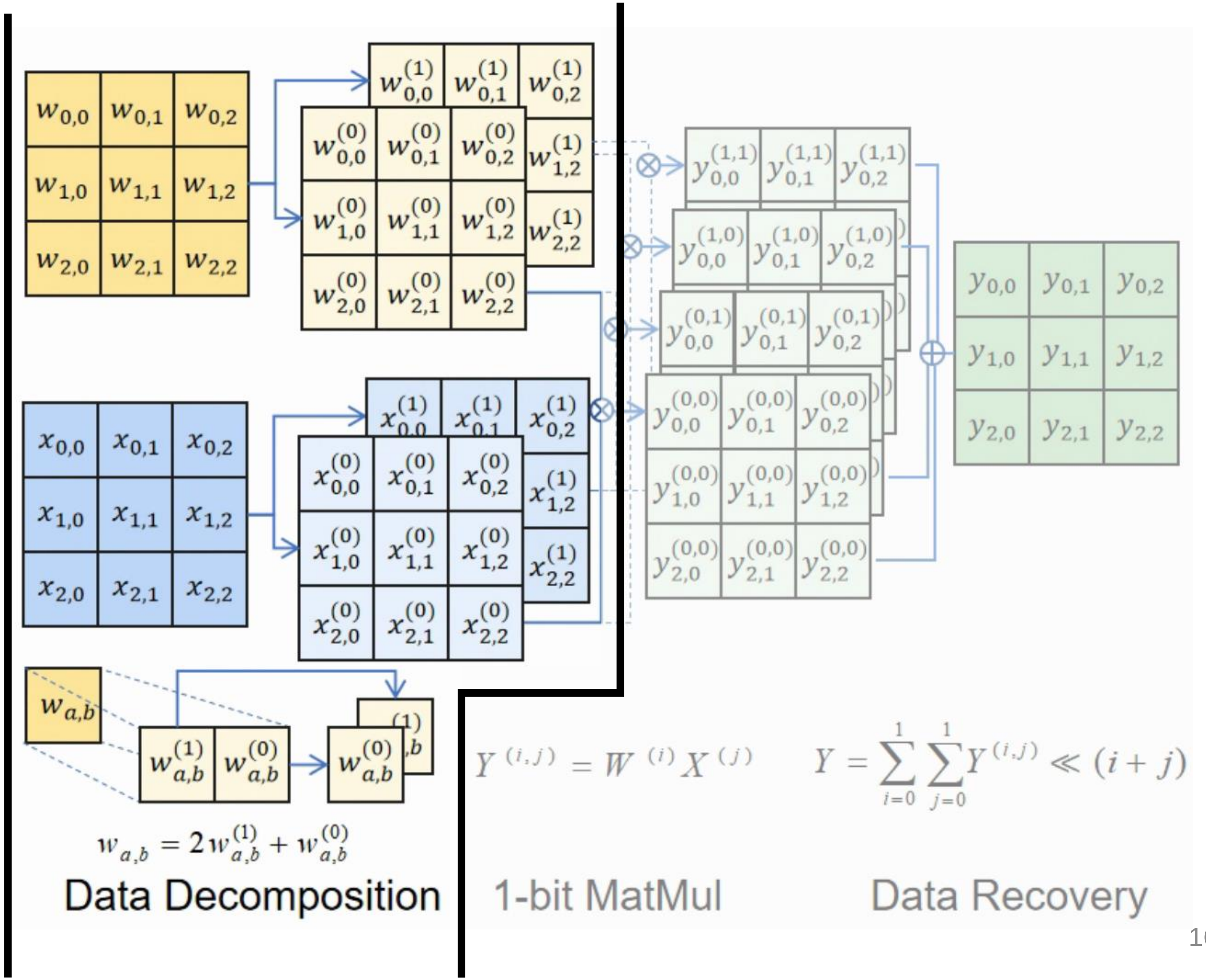
2.1.2 Comparison with Unsigned INT

- Compared with Signed INT
  - ◆ Without sign bit
  - ◆ Easy to parallelize
- Compared with **Unsigned INT**
  - ◆ Symmetric range
  - ◆ Redundancy Reduction



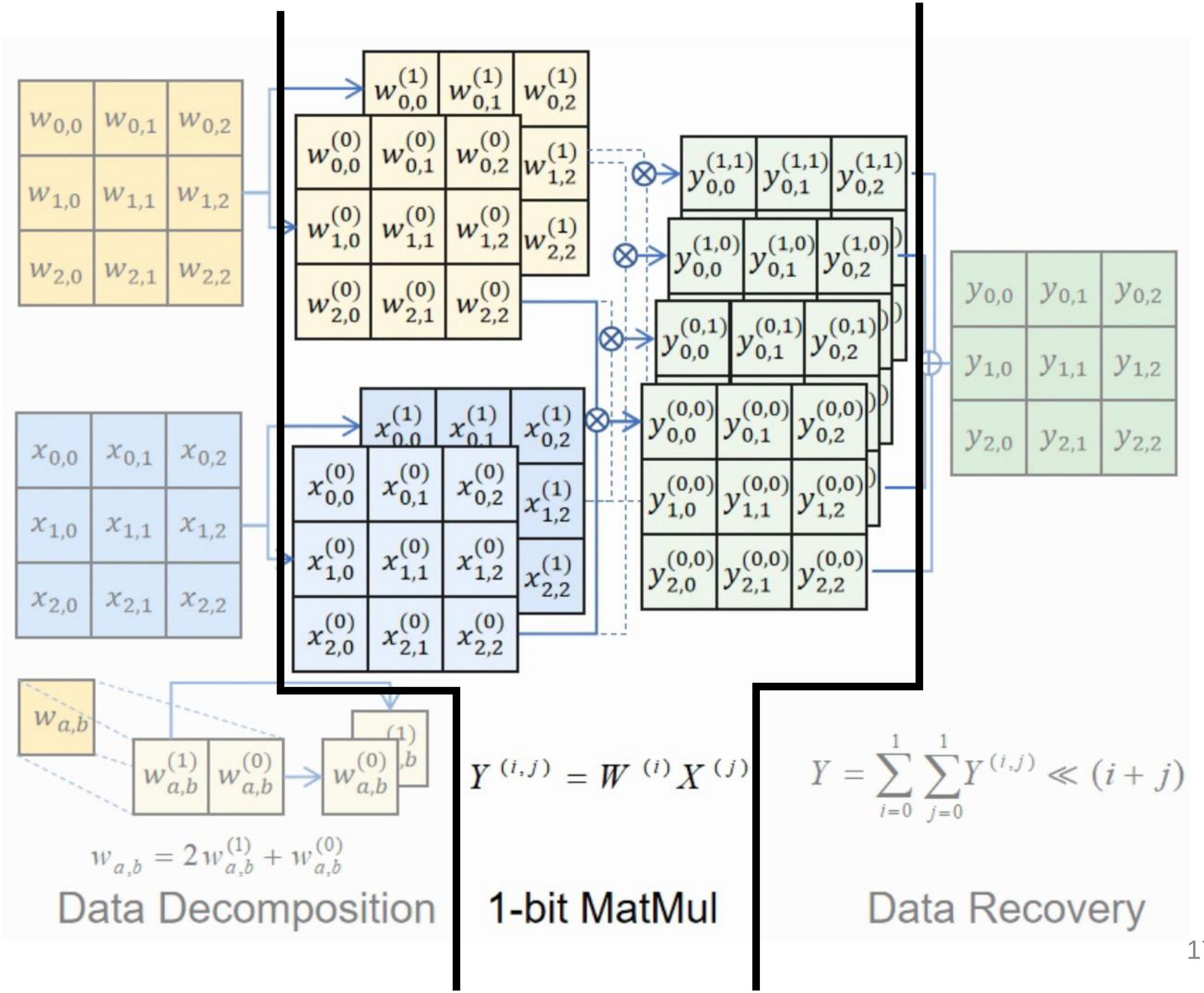
2.2 Bit-Wise MatMul Reconstitution (1)

- Data Decomposition
  - ◆ Split input data bit by bit
  - ◆ Divide into 1-bit matrices
- 1-bit MatMul
- Data Recovery



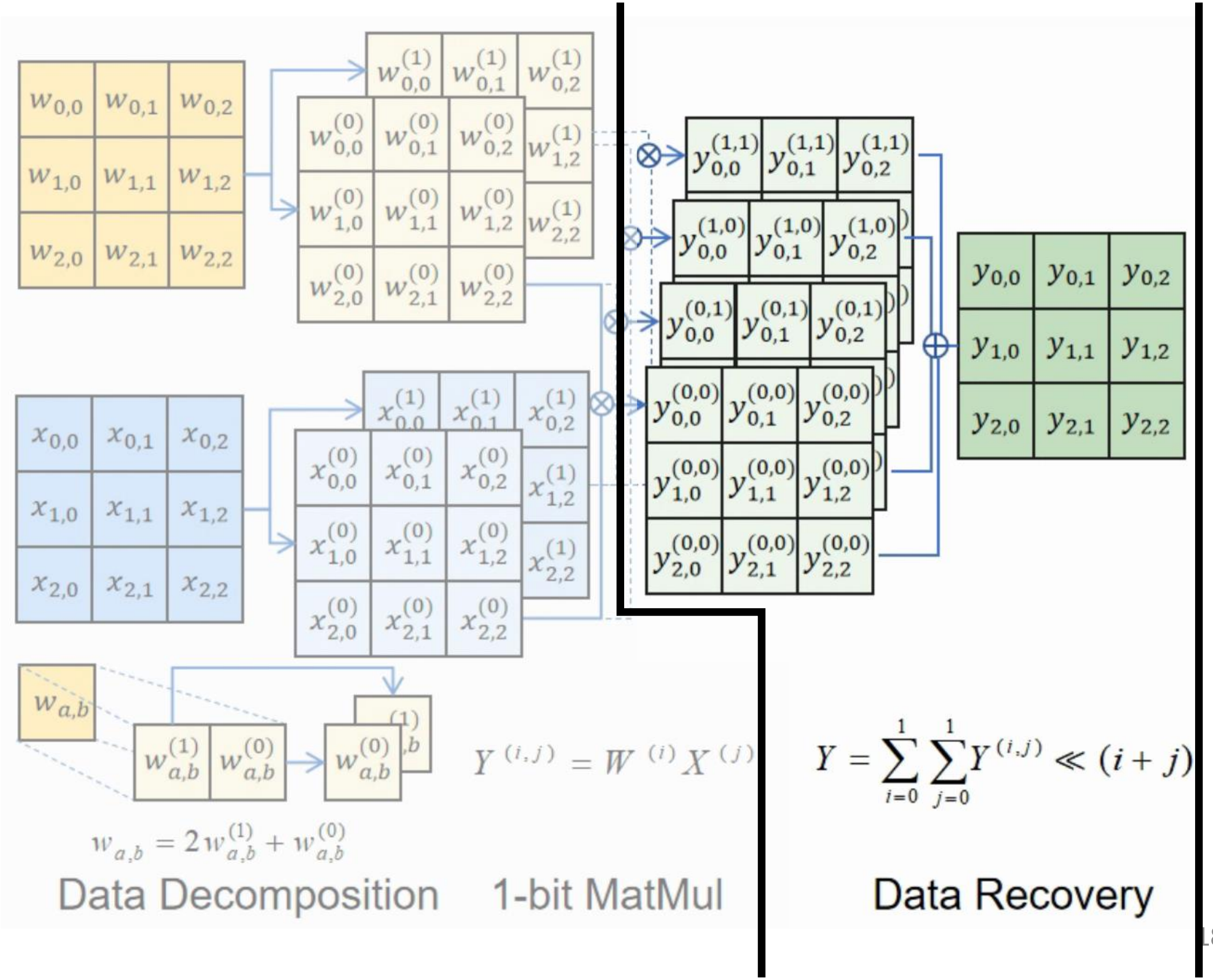
2.2 Bit-Wise MatMul Reconstitution (2)

- Data Decomposition
  - ◆ Split input data bit by bit
  - ◆ Divide into 1-bit matrices
- 1-bit MatMul
  - ◆ Pairwise combine input
  - ◆ Output intermediate matrices
- Data Recovery



2.2 Bit-Wise MatMul Reconstitution (3)

- Data Decomposition
  - ◆ Split input data bit by bit
  - ◆ Divide into 1-bit matrices
- 1-bit MatMul
  - ◆ Pairwise combine input
  - ◆ Output intermediate matrices
- Data Recovery
  - ◆ Shift and add matrices
  - ◆ Output final result



2.2 Bit-Wise MatMul Reconstitution (4)

- Applicable to Arbitrary Precision MatMul
  - Any INT-like data format can be decomposed into 1-bit matrices
  - GPU TC supports 1-bit MatMul computation
- Applicable to **Both** INT and Bipolar-INT
  - INT: implement 1-bit MatMul using **“AND”** operation
  - Bipolar-INT: implement 1-bit Matmul using **“XOR”** operation

w	x	y
0	0	0
0	1	0
1	0	0
1	1	1

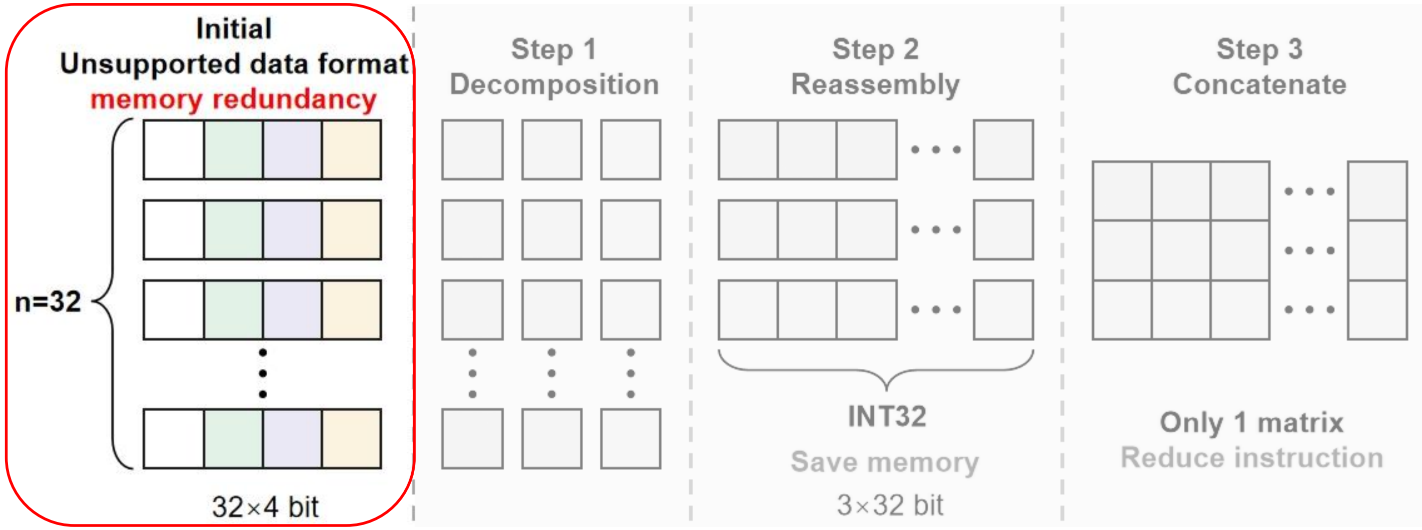
1-bit INT Multiplication is Implemented as AND Logic

w	x	y
0(-1)	0(-1)	1
0(-1)	1	0(-1)
1	0(-1)	0(-1)
1	1	1

1-bit Bipolar-INT Multiplication is Implemented as XOR Logic

2.3 Matrix Decomposition and Reassembly

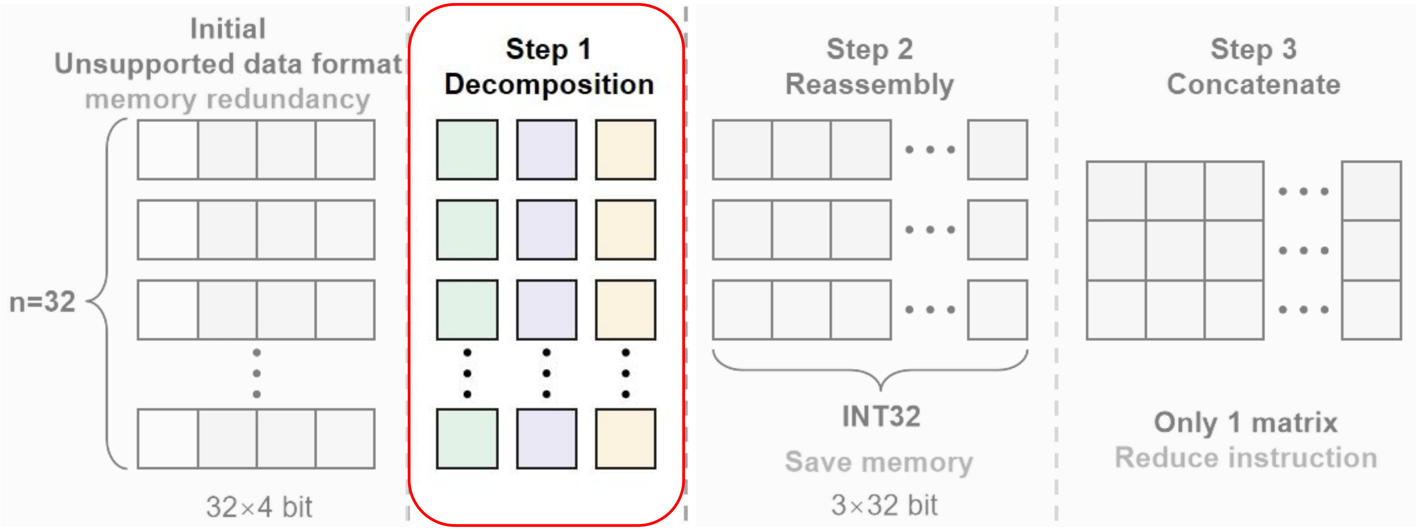
- The Necessity of Input Data Preprocessing
  - ◆ Memory redundancy due to unsupported data format
  - ◆ Subsequent computations require bitwise decomposition



Matrix Decomposition and Reassembly, Taking 3-bit Data as an Example

2.3 Matrix Decomposition and Reassembly (1)

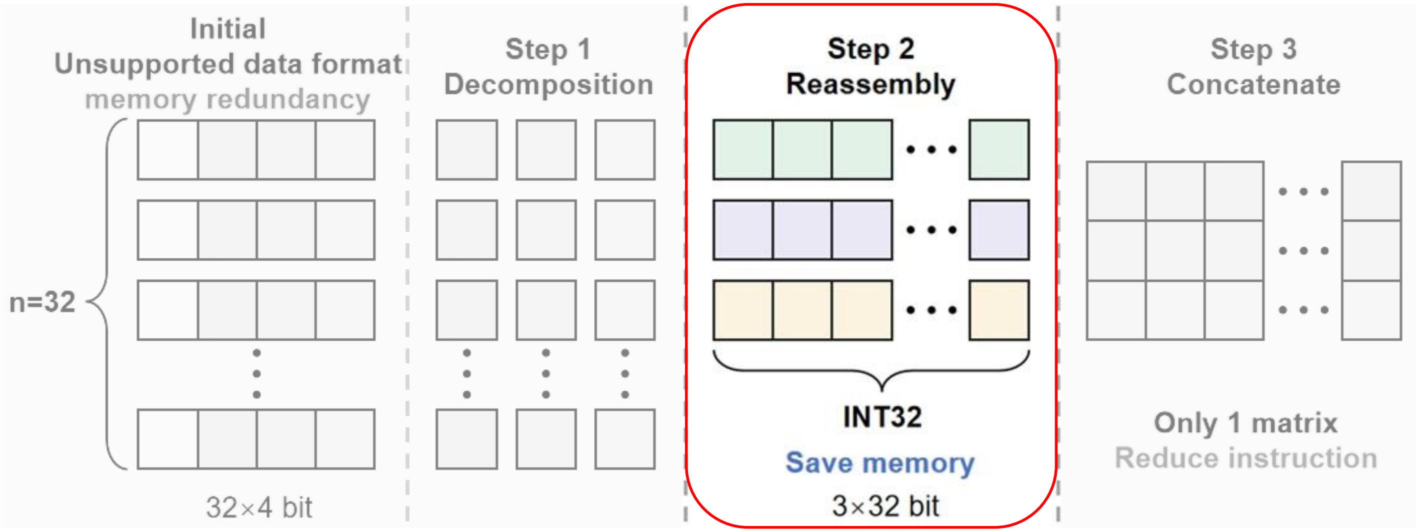
- Matrix Decomposition
  - Break down each bit and regroup them
  - Eliminate the redundancy due to unsupported data formats



Matrix Decomposition and Reassembly, Taking 3-bit Data as an Example

2.3 Matrix Decomposition and Reassembly (2)

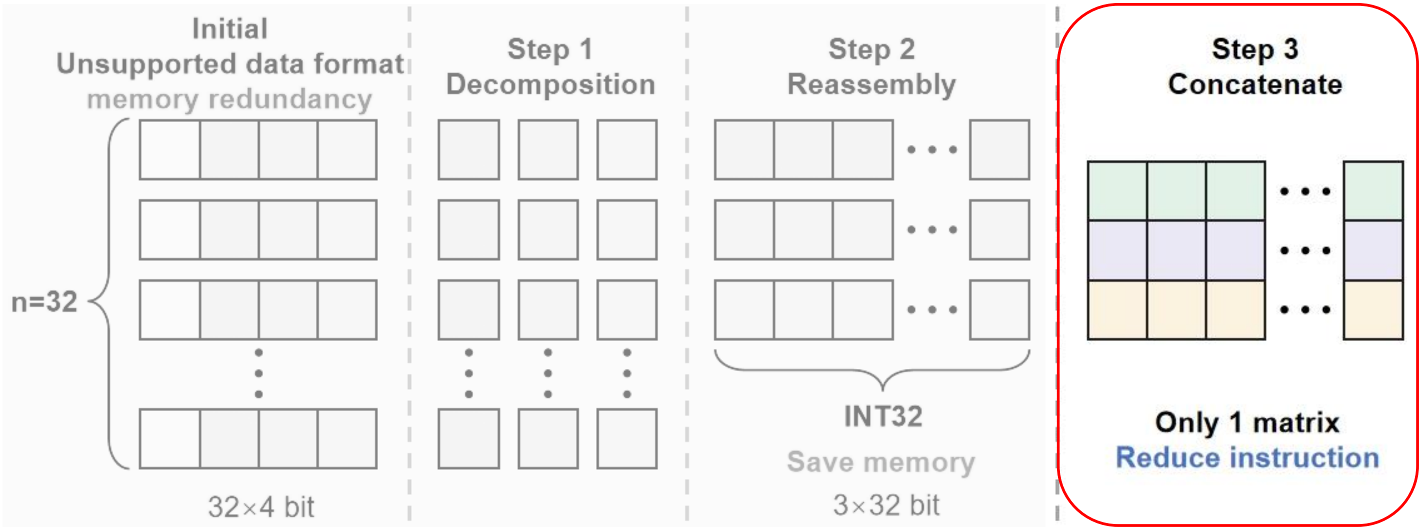
- Data Reassembly
  - ◆ Reassemble data using 32-bit unsigned INTs
  - ◆ Align with the native support, thereby enhance transfer speed



Matrix Decomposition and Reassembly, Taking 3-bit Data as an Example

2.3 Matrix Decomposition and Reassembly (3)

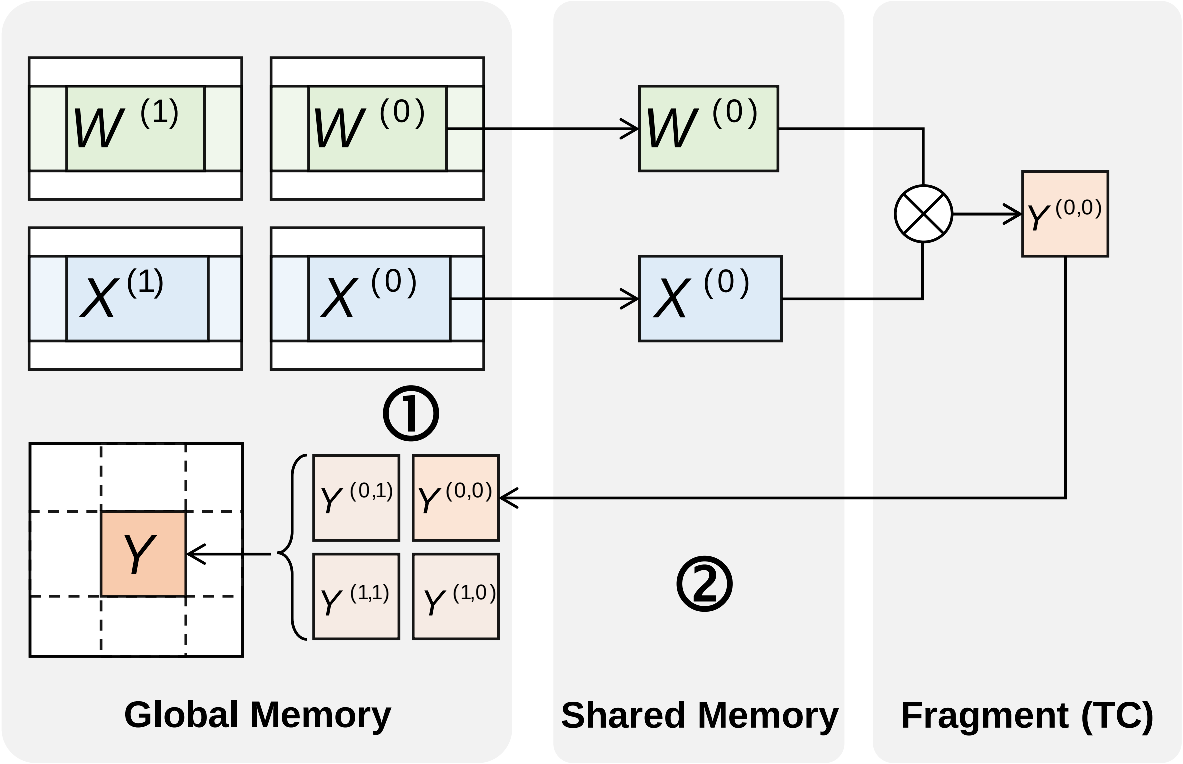
- Matrix Concatenate
  - ◆ Concatenate processed matrices into a single matrix
  - ◆ Reduce transmission instructions to further improve transfer speed
  - ◆ Facilitate subsequent computations



Matrix Decomposition and Reassembly, Taking 3-bit Data as an Example

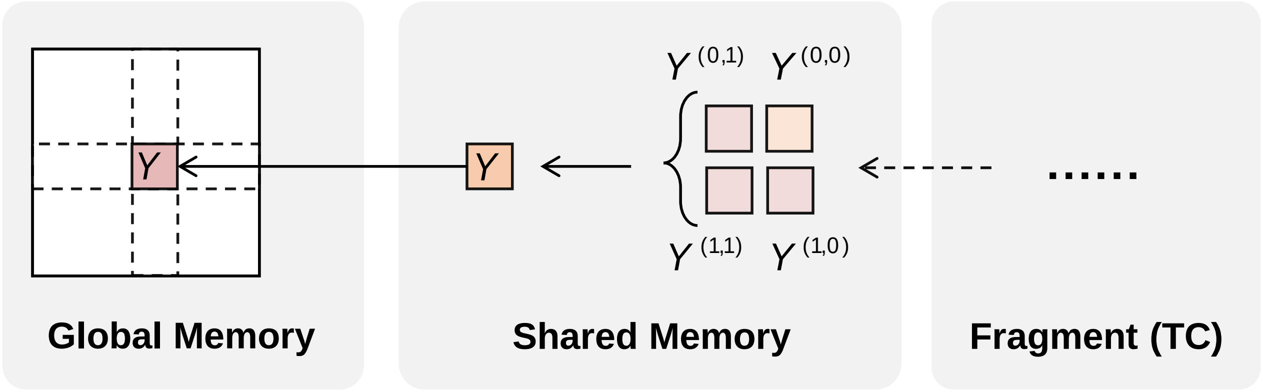
2.4 Recovery-Oriented Memory Scheduling

- GPU Implementation of Bit-Wise MatMul (2.2): General Approach
  - Each SM handles one pair of 1-bit WX matrices
  - Shift and add in global memory
- Low Efficiency Reasons
  - Matrix recovery in global memory ①
  - Low utilization of shared memory ②
- Optimization Goals
  - Reduce computation in global memory
  - Move matrix recovery to shared memory



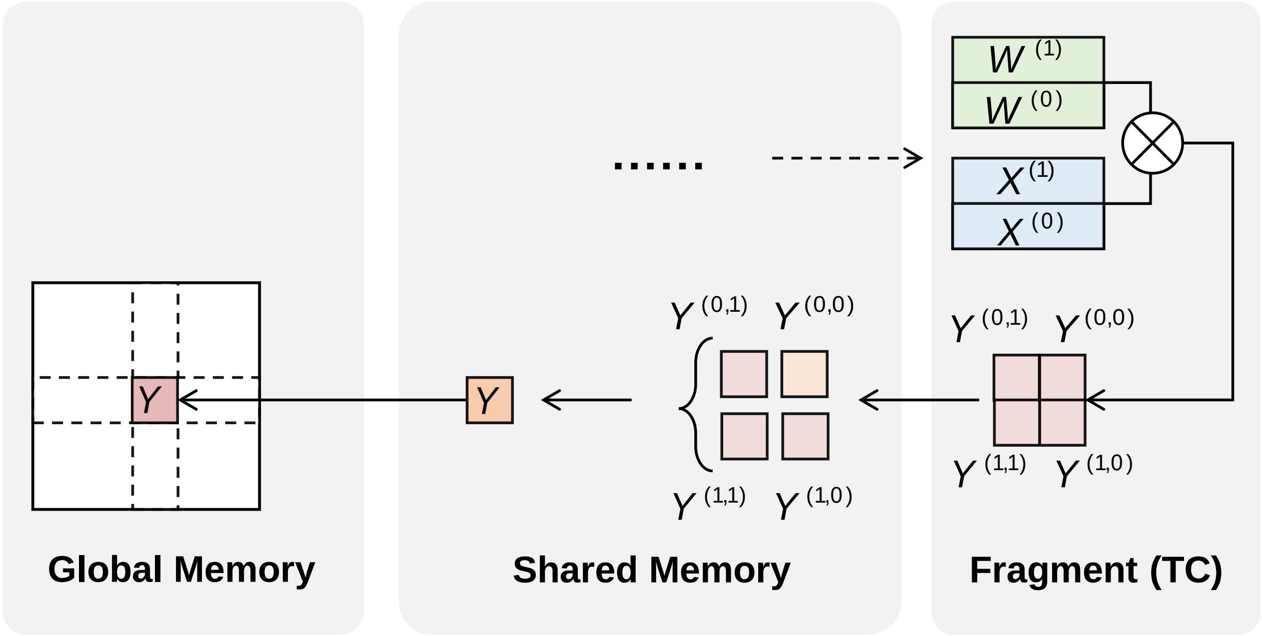
2.4.1 Matrix Recovery in Shared Memory

- Complete Matrix Recovery in the Shared Memory of a Single SM (Streaming Multiprocessor)
  - ◆ Obtain all intermediate matrices for the output
  - ◆ Shift and add in shared memory



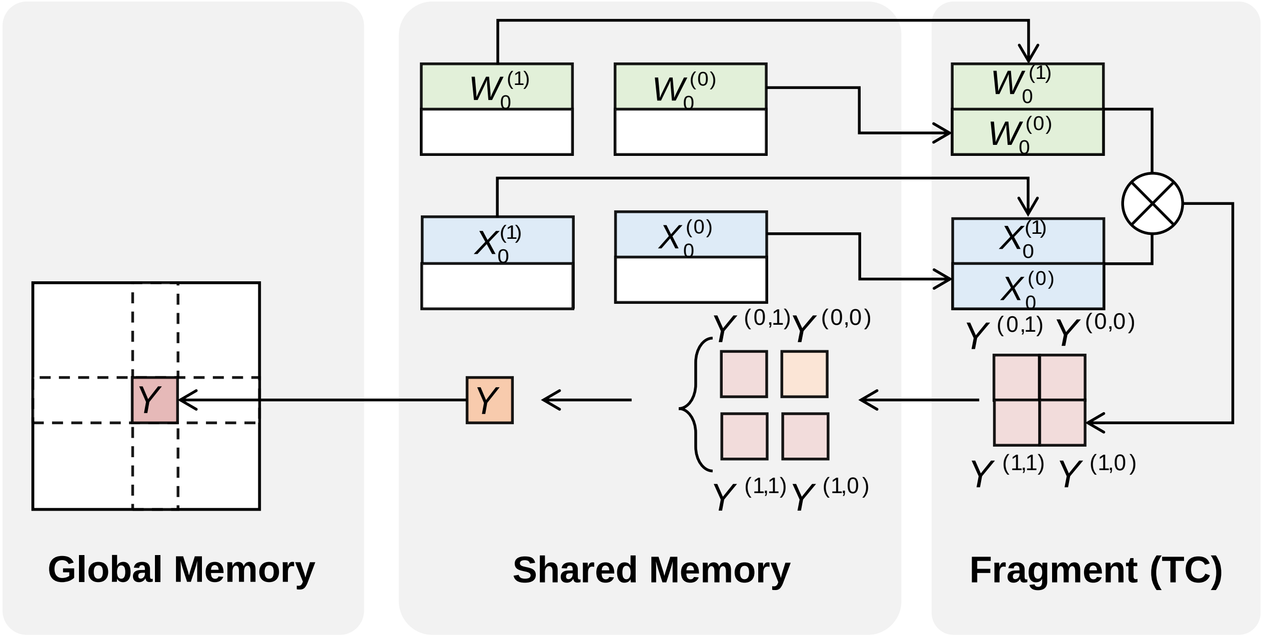
2.4.2 Compute All Intermediate Matrices

- Pairwise Combine of ***W*** and ***X*** with Different Bitwidth
  - It can be achieved within one MatMul computation
  - Implement 1-bit MatMul in Tensor Cores



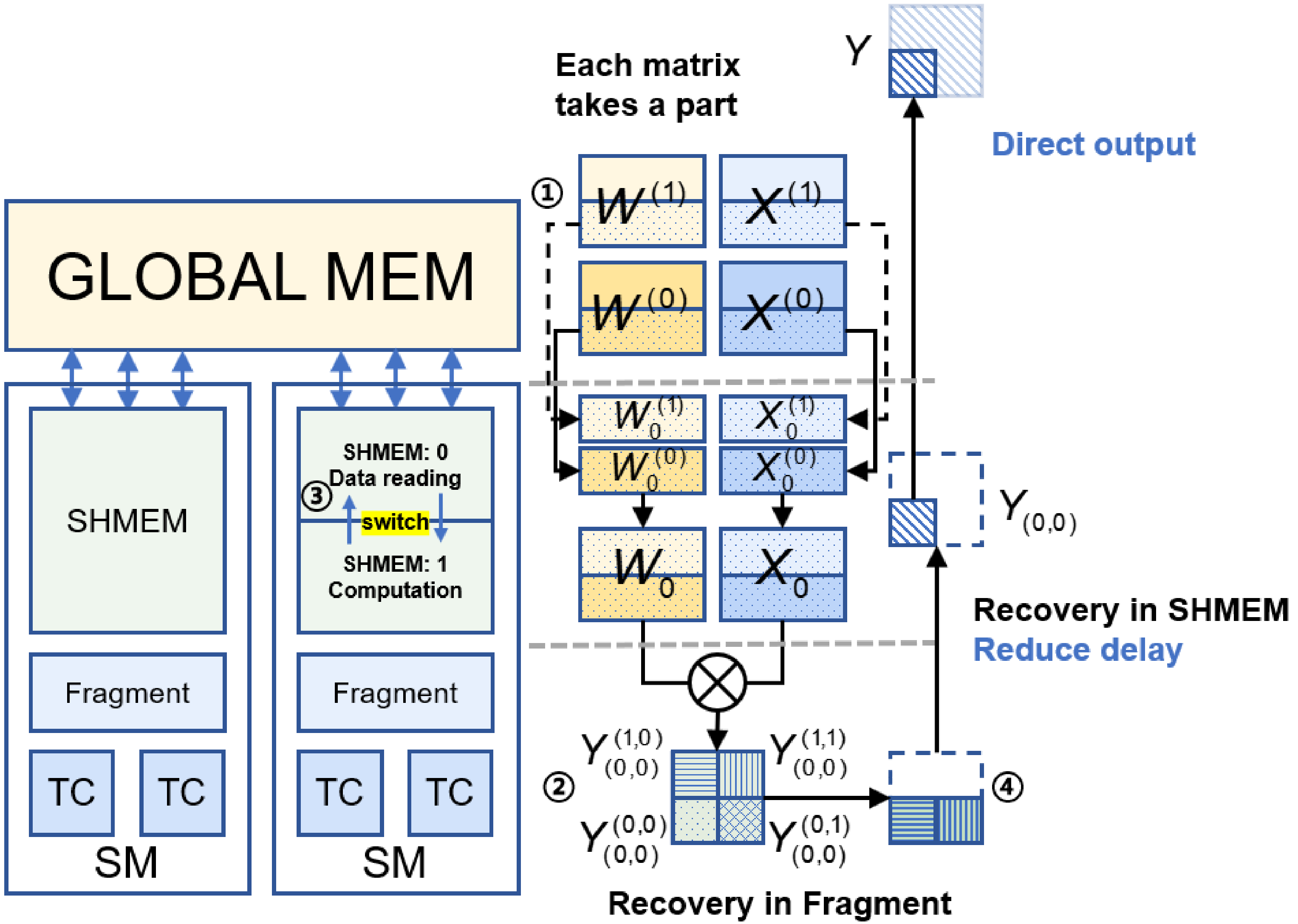
2.4.3 Matrix Concatenation in Shared Memory

- Read Data from All 1-bit Matrices and Concatenate
  - ◆ Includes all data required for output



2.4.4 Overall Scheduling

- Recovery-Oriented Memory Scheduling
  - ◆ ①Matrix Concatenation in Shared Memory
  - ◆ ②Compute All Intermediate Matrices
  - ◆ ④Matrix Recovery in Shared Memory



03 Experiments

3.1 Experimental Setup

- **Computing Platforms:** NVIDIA RTX 3090 GPU (Ampere Architecture)
- **Compilation Environment:** CUDA-11.8 and CUTLASS-2.11
- **Baselines:** Pytorch FP32, Pytorch FP16, CUTLASS INT4, CUTLASS INT1, APNN-TC [1], BSTC [2], BTC [3]
- **LLM Models:** LLaMA2-7B, OPT-6.7B, BLOOM-7B
- **Workloads:**
  - Square matrices MatMuls
  - LLM-specific matrices MatMuls
  - LLM models inference speed evaluation

[1] Feng, Boyuan, et al. "Apnn-tc: Accelerating arbitrary precision neural networks on ampere gpu tensor cores." *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 2021.
[2] Li, Ang, et al. "BSTC: A novel binarized-soft-tensor-core design for accelerating bit-based approximated neural nets." *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 2019.
[3] Li, Ang, and Simon Su. "Accelerating binarized neural networks via bit-tensor-cores in turing gpus." *IEEE Transactions on Parallel and Distributed Systems* 32.7 (2020): 1878-1891.

3.2.1 Arbitrary Precision Square MatMuls

- Comparison with FP32 and FP16 towards large square MatMuls
  - **193×** speedup over FP32 (4k/4k/4k, W1A2)
  - **66.7×** speedup over FP16 (4k/4k/4k, W1A2)

M/N/K	**1k/1k/1k**		**2k/2k/2k**		**4k/4k/4k**	
Schemes	Latency	Speedup	Latency	Speedup	Latency	Speedup
FP32	121us	1.00×	779us	1.00×	5690us	1.00×
FP16	44.2us	2.73×	263us	2.96×	1960us	2.90×
CUTLASS INT4	15.8us	7.61×	66.5us	11.7×	386us	14.7×
CUTLASS INT1	9.3us	13.0×	36.9us	21.1×	161us	35.3×
W3A4 (ours)	12.4us	9.74×	50.4us	15.4×	184us	31.0×
W2A2 (ours)	8.7us	13.9×	18.1us	43.0×	46.5us	122×
W1A2 (ours)	9.0us	13.4×	11.7us	66.4×	29.5us	193×

66.7×

3.2.1 Arbitrary Precision Square MatMuls

- Comparison with CUTLASS INT1 and INT4 towards Large Square MatMuls
  - ◆ More than **13×** speedup over CUTLASS INT4 (4k/4k/4k, W1A2)
  - ◆ **5.5×** faster than CUTLASS INT1 (4k/4k/4k, W1A2)
  - ◆ **3.5×** faster than CUTLASS INT1 (4k/4k/4k, W2A2)

M/N/K	**1k/1k/1k**		**2k/2k/2k**		**4k/4k/4k**	
Schemes	Latency	Speedup	Latency	Speedup	Latency	Speedup
FP32	121us	1.00×	779us	1.00×	5690us	1.00×
FP16	44.2us	2.73×	263us	2.96×	1960us	2.90×
CUTLASS INT4	15.8us	7.61×	66.5us	11.7×	386us	14.7×
CUTLASS INT1	9.3us	13.0×	36.9us	21.1×	161us	35.3×
W3A4 (ours)	12.4us	9.74×	50.4us	15.4×	184us	31.0×
W2A2 (ours)	8.7us	13.9×	18.1us	43.0×	46.5us	122×
W1A2 (ours)	9.0us	13.4×	11.7us	66.4×	29.5us	193×

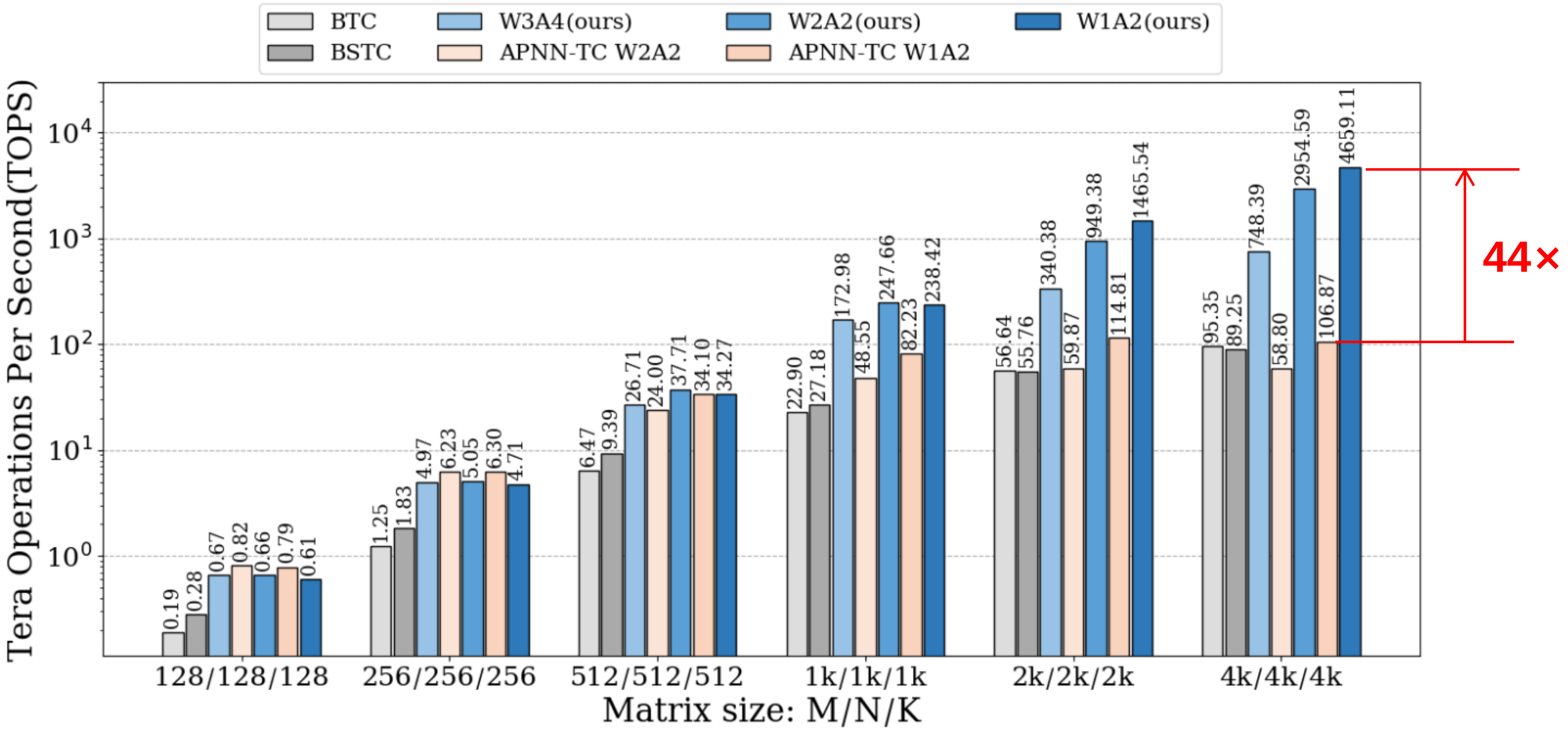
13.1×

3.5×

5.5×

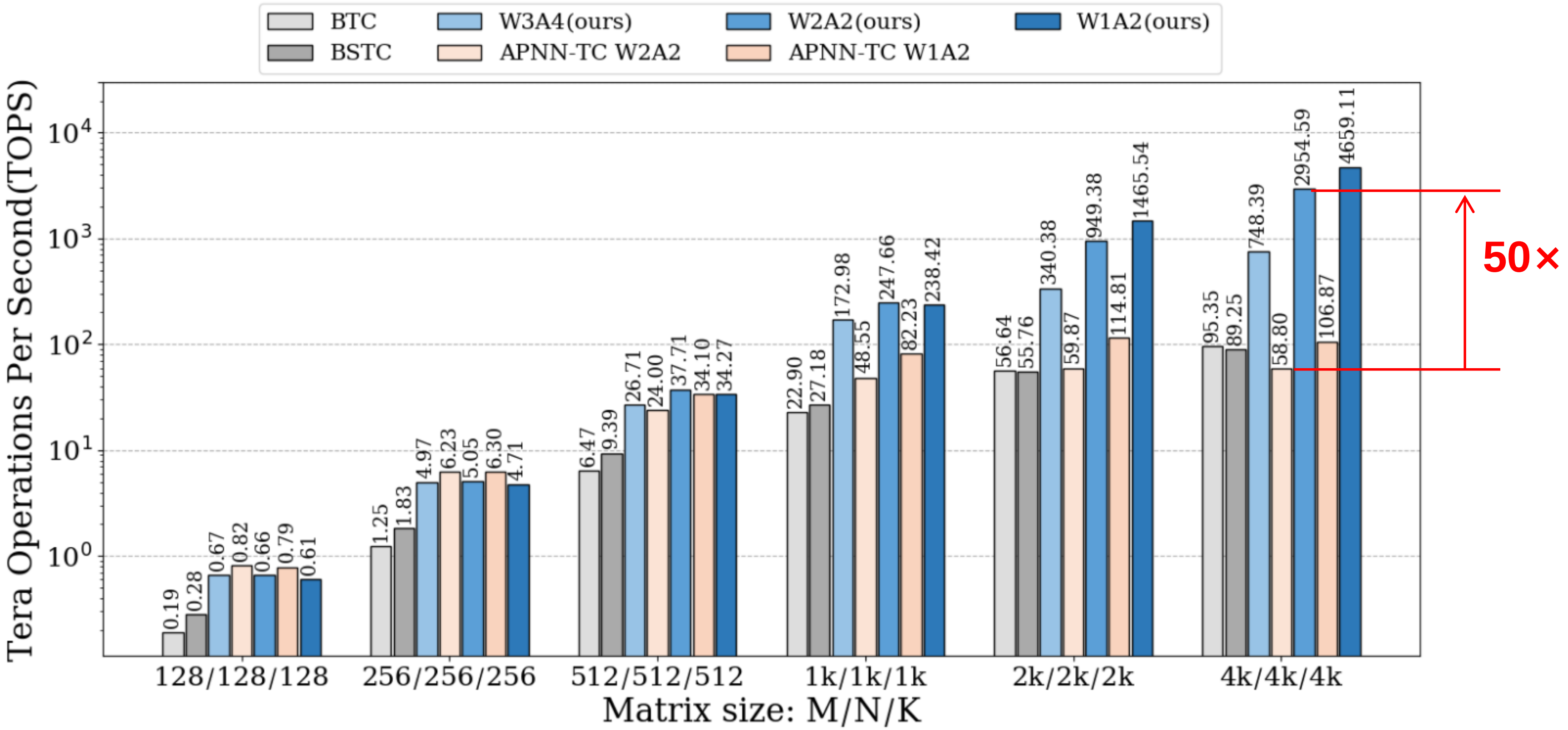
3.2.1 Arbitrary precision Square MatMuls

- Comparison of Throughput with Other Methods
  - ◆ **44×** TOPS over APNN-TC (W1A2)
  - ◆ 50× TOPS over APNN-TC (W2A2)



3.2.1 Arbitrary precision Square MatMuls

- Comparison of Throughput with Other Methods
  - ◆ 44× TOPS over APNN-TC (W1A2)
  - ◆ **50×** TOPS over APNN-TC (W2A2)



3.2.2 Arbitrary Precision LLM-specific MatMuls

- Extract the MatMul parameters from Llama2-7B
  - More than 90× speedup over FP32
  - Significant speedup over FP16

M/N/K	1k/4k/4k		1k/10.5k/4k		1k/4k/10.5k	
Schemes	Latency	Speedup	Latency	Speedup	Latency	Speedup
FP32	3.12ms	1.00×	8.21ms	1.00×	8.36ms	1.00×
FP16	1.07ms	2.91×	1.47ms	5.58×	1.58ms	5.30×
CUTLASS INT4	0.238ms	13.1×	0.574ms	14.3×	0.548ms	15.3×
CUTLASS INT1	0.097ms	32.1×	0.255ms	32.2×	0.188ms	44.6×
W3A4 (ours)	0.194ms	16.1×	0.523ms	15.7×	0.540ms	15.5×
W2A2 (ours)	0.059ms	53.2×	0.143ms	57.6×	0.165ms	50.7×
W1A2 (ours)	0.034ms	**91.2×**	0.084ms	**98.1×**	0.082ms	**102×**

31.3× **17.6×** **19.2×**

3.2.2 Arbitrary Precision LLM-specific MatMuls

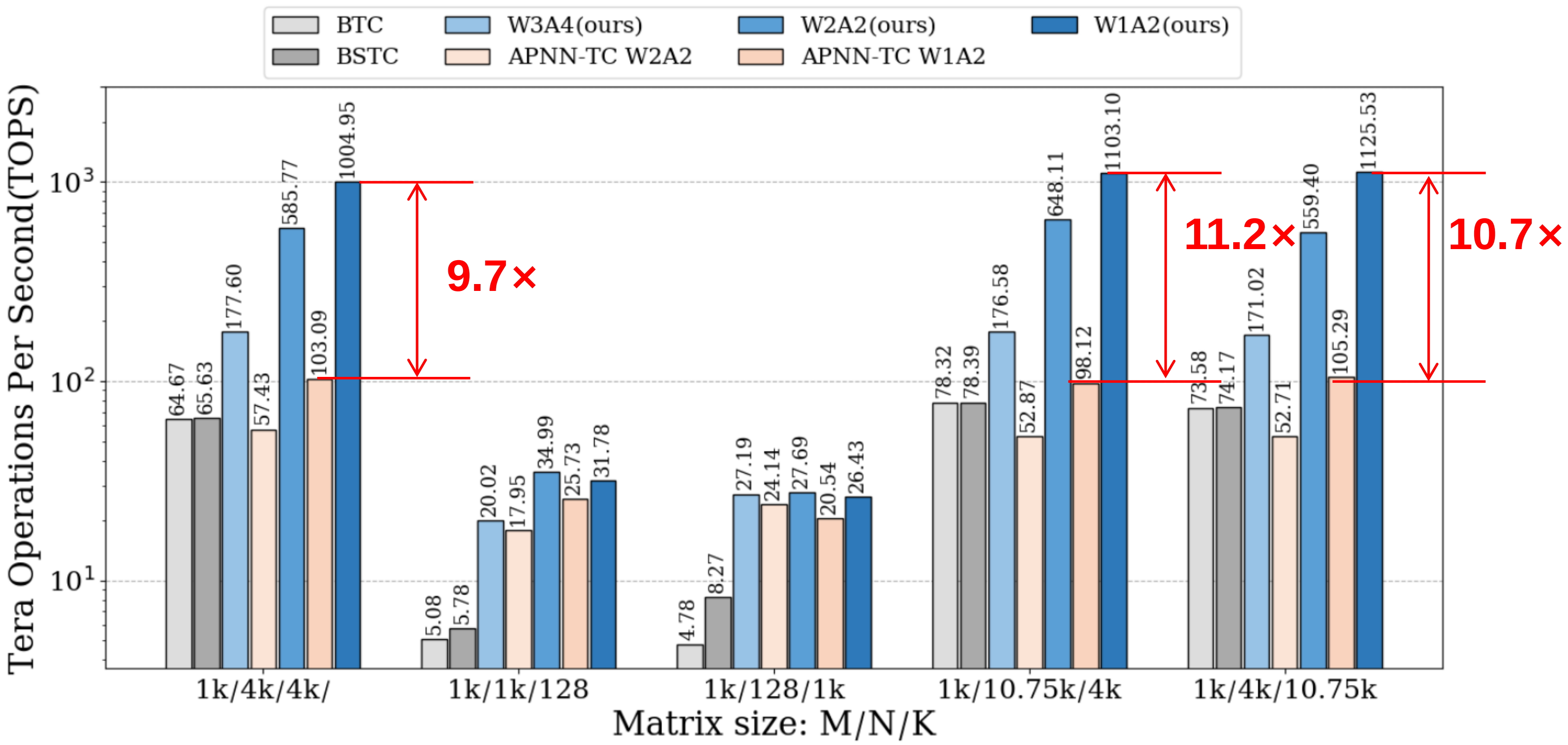
- Extract the MatMul parameters from Llama2-7B
  - About **7×** speedup over CUTLASS INT4
  - Over **2.2×** speedup over CUTLASS INT1

M/N/K	1k/4k/4k		1k/10.5k/4k		1k/4k/10.5k	
Schemes	Latency	Speedup	Latency	Speedup	Latency	Speedup
FP32	3.12ms	1.00×	8.21ms	1.00×	8.36ms	1.00×
FP16	1.07ms	2.91×	1.47ms	5.58×	1.58ms	5.30×
CUTLASS INT4	0.238ms	13.1×	0.574ms	14.3×	0.548ms	15.3×
CUTLASS INT1	0.097ms	32.1×	0.255ms	32.2×	0.188ms	44.6×
W3A4 (ours)	0.194ms	16.1×	0.523ms	15.7×	0.540ms	15.5×
W2A2 (ours)	0.059ms	53.2×	0.143ms	57.6×	0.165ms	50.7×
W1A2 (ours)	0.034ms	**91.2×**	0.084ms	**98.1×**	0.082ms	**102×**

2.84× 7.0× 3.05× 6.9× 2.29× 6.7×

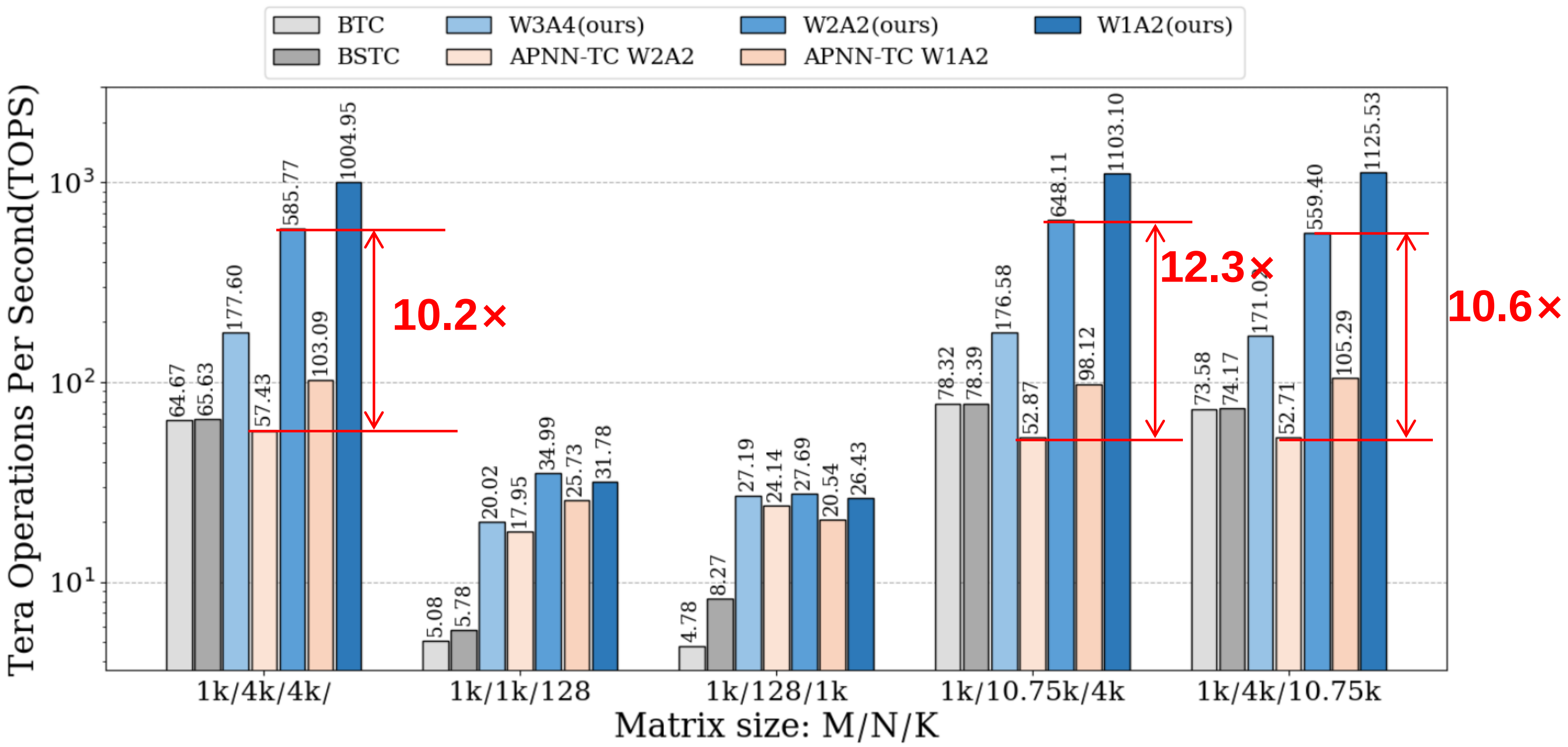
3.2.2 Arbitrary Precision LLM-specific MatMuls

- Comparison of Throughput with Other Methods
  - ◆ About **10×** TOPS over APNN-TC (W1A2)
  - ◆ More than 10× TOPS over APNN-TC (W2A2)



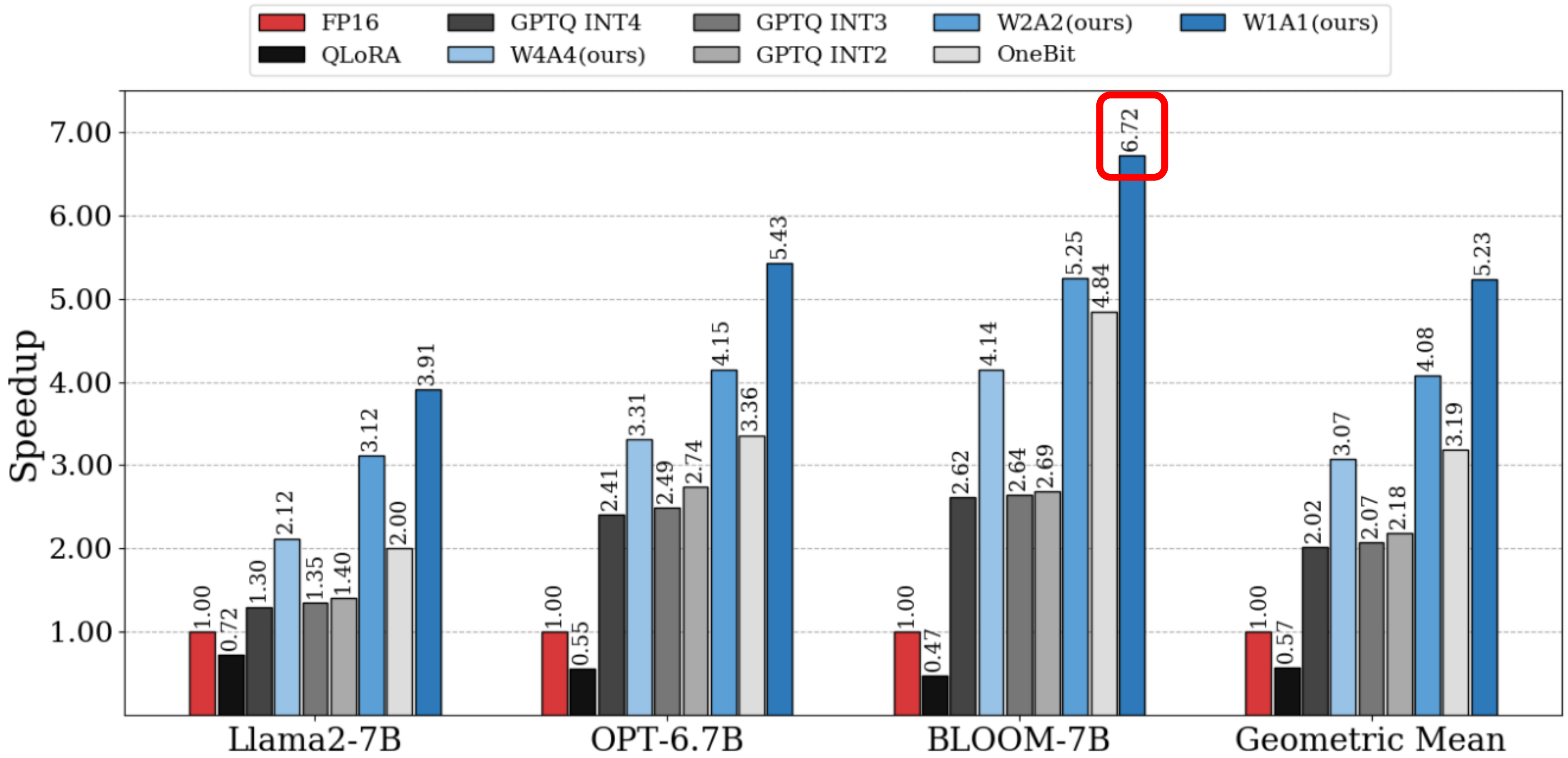
3.2.2 Arbitrary Precision LLM-specific MatMuls

- Comparison of Throughput with Other Methods
  - About 10× TOPS over APNN-TC (W1A2)
  - More than **10×** TOPS over APNN-TC (W2A2)



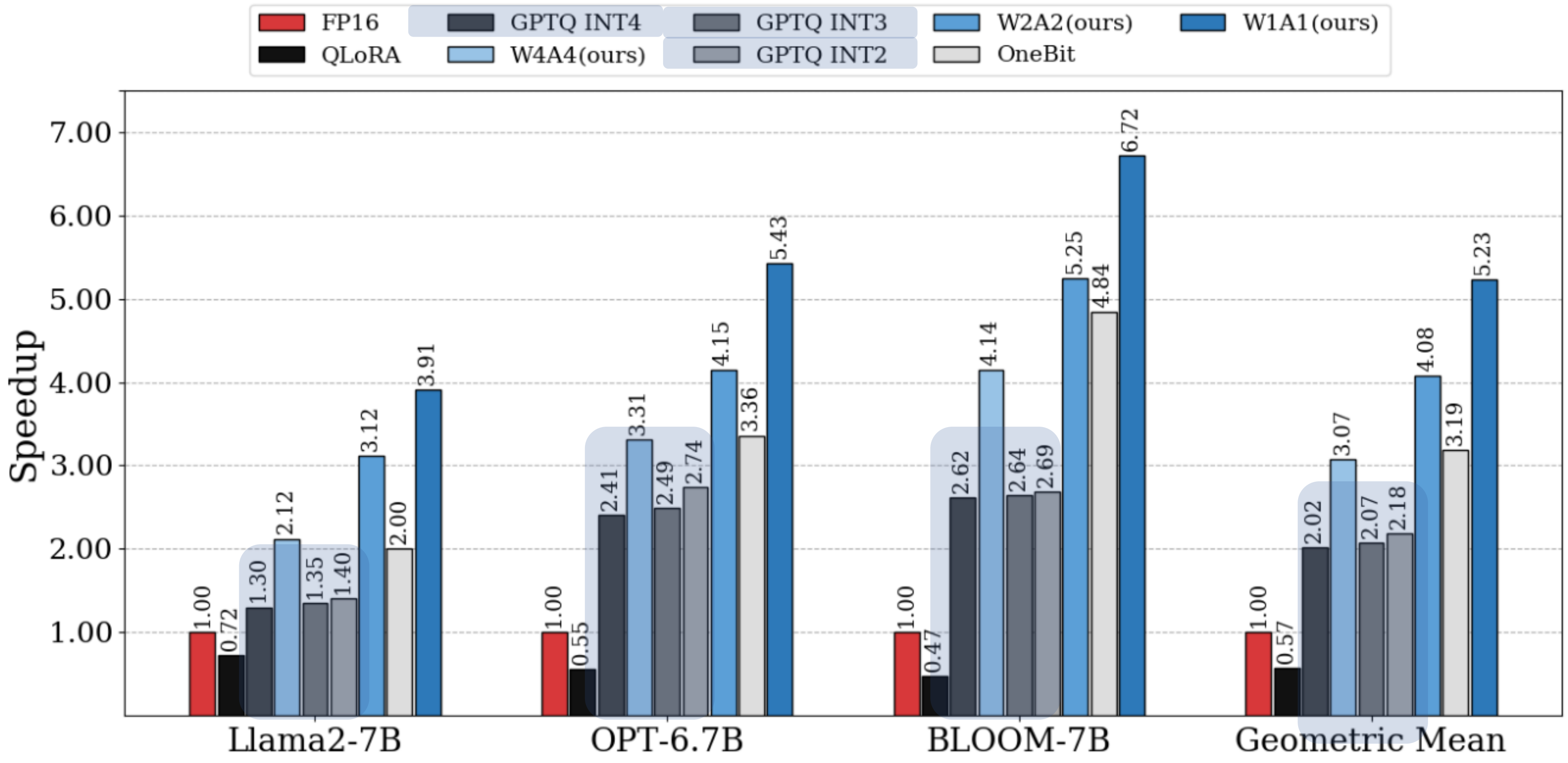
3.3 Arbitrary Precision LLM Evaluation

- Single Inference of Different LLMs
  - ◆ Up to **6x** speedup compared to FP16 (W1A2)



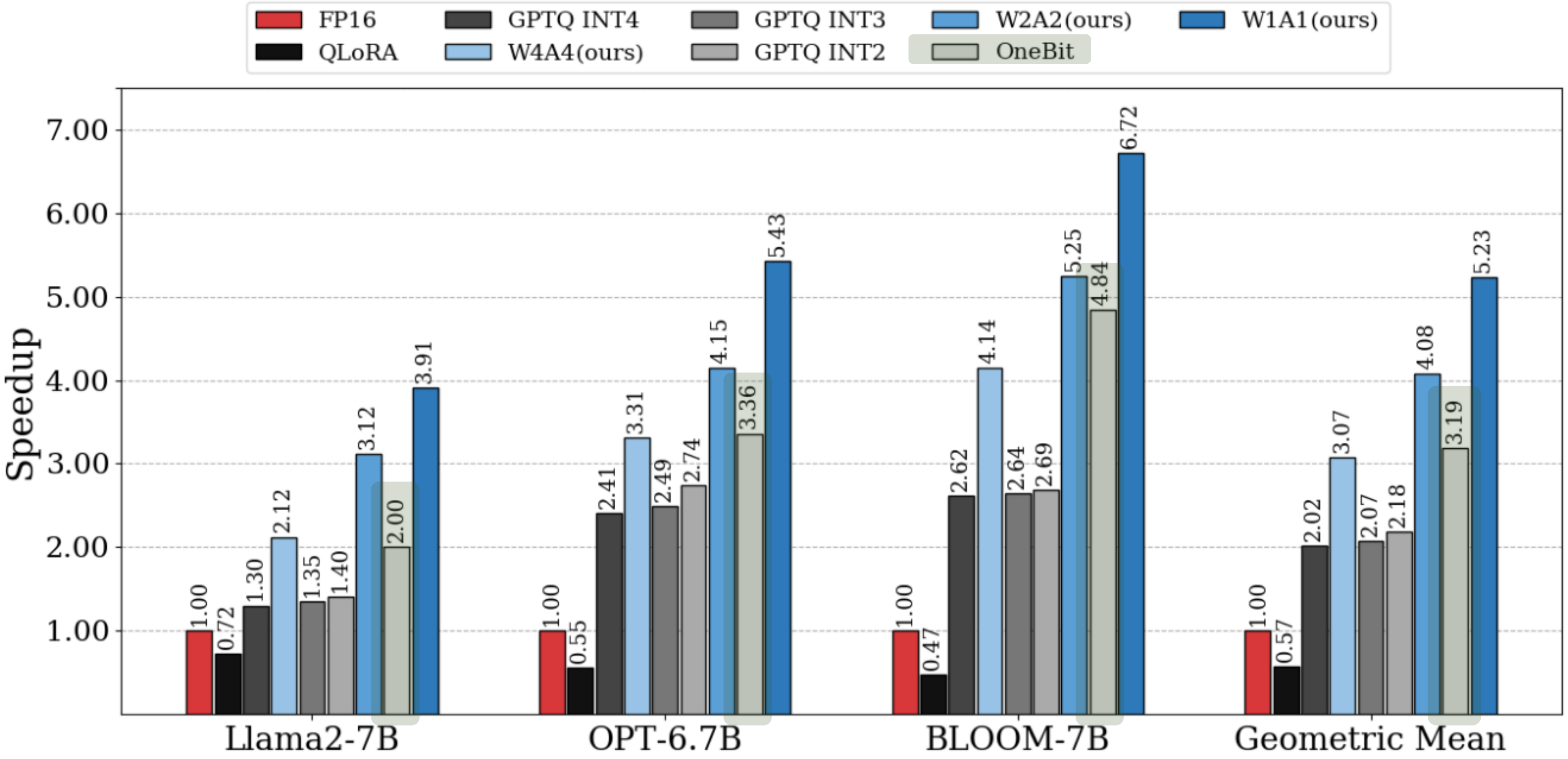
3.3 Arbitrary Precision LLM Evaluation

- Compared with GPTQ INT2/3/4 (Use CUTLASS INT4 Kernel)
  - ◆ The inference speed of GPTQ is almost identical



3.3 Arbitrary Precision LLM Evaluation

- Compared with OneBit (Use CUTLASS INT1 Kernel)
  - ◆ W1A2 and W2A2 configurations still achieve speedup



04 Conclusion

Conclusion

- Efficient Arbitrary Precision Acceleration for Large Language Models on GPU Tensor Cores
  - ◆ Bipolar-INT Data Format
  - ◆ Bit-Wise MatMul Reconstitution
  - ◆ Matrix Decomposition and Reassembly
  - ◆ Recovery-Oriented Memory Scheduling
- Achieves a **5.5×** Speedup Compared to NVIDIA CUTLASS
- Achieves a **44×** Speedup Compared to Existing Solutions
- The Model Inference Speed is **3.9-6.7×** Faster Compared to FP16
- The Model Inference Speed is **1.2-2×** Faster than Quantized Model with CUTLASS Kernel

南京大學
NANJING UNIVERSITY

Thank you for listening!

Welcome to contact us by email:
shaoboma@smail.nju.edu.cn

ICAIS Lab, Nanjing University, China