SpInfer: Leveraging Low-Level Sparsity for Efficient Large Language Model Inference on GPUs

Ruibo Fan, et al.
EuroSys 2025

Jinho Choi

2025.09.12

HYPER ACCEL

Contents

Introduction

- ❑ LLM poses a signifcant challenge to memory and computation cost.
  - **Unstructured Pruning**: Zero-out less important elements in weight matrices.
  - LLM uses unstructured pruning to make weight matrices sparse, and accelerate via SpMM(Sparse Matrix Multiplication) kernel. Usually, LLM's weight matrices are pruned to 50% sparsity(low-sparsity).
  - However, (1) Sparse Matrix at low sparsity incurs extra storage overhead (2) SpMM kernel for low sparsity doesn't achieve practical acceleration at GPU.

HYPER ACCEL

Introduction

❑ Sparse Matrix at low sparsity incurs extra storage overhead.

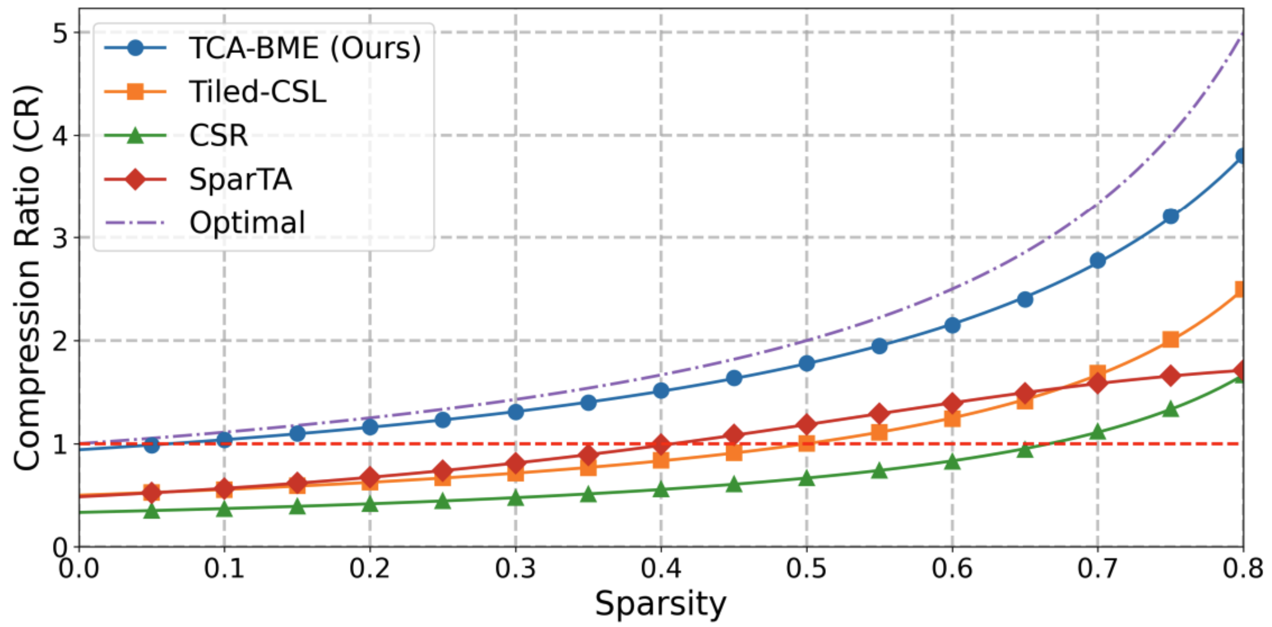


Figure 3. Compression Ratio (CR) across varying sparsity levels for different sparse matrix formats.

Introduction

- SpMM kernel for low sparsity doesn't achieve practical acceleration in GPU.

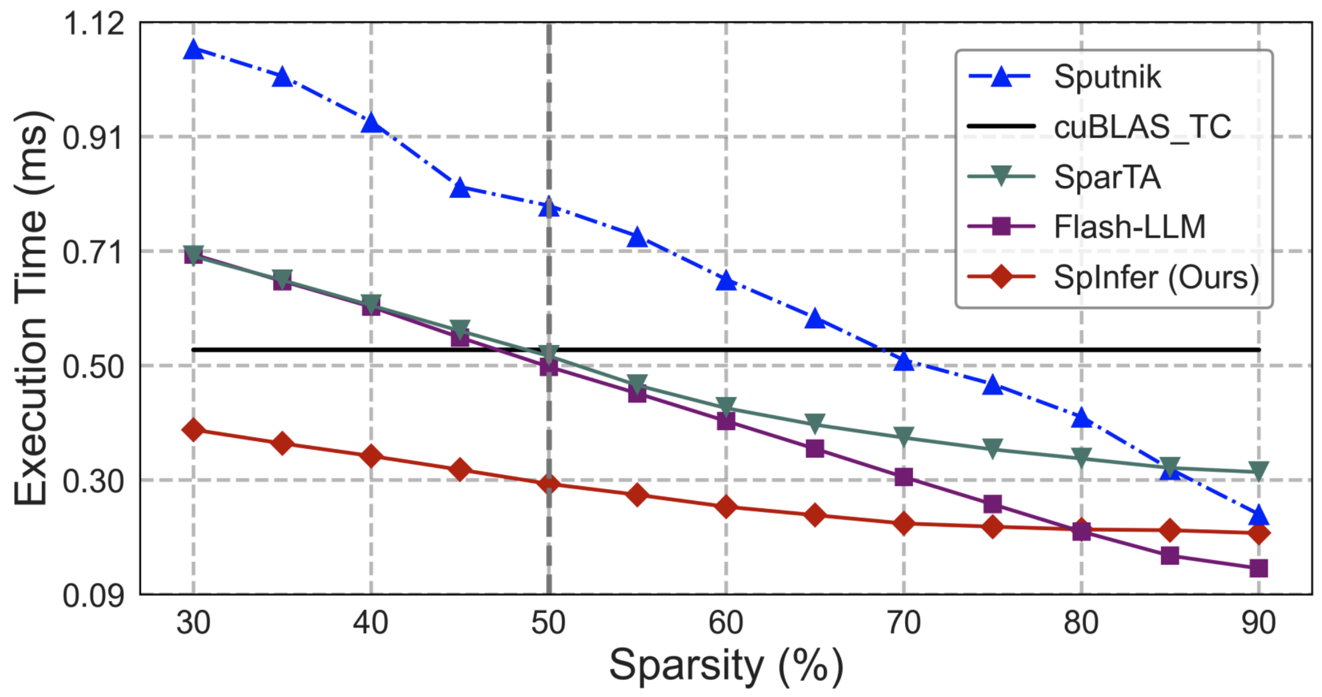


Figure 1. Execution time comparison of unstructured SpMM implementations against cuBLAS on Nvidia RTX4090 (M/K/N=28K/8K/16, typical in LLM inference).

Background

❑ Unstructured Pruning

- Making the weight matrix "sparse" to reduce memory footprints.

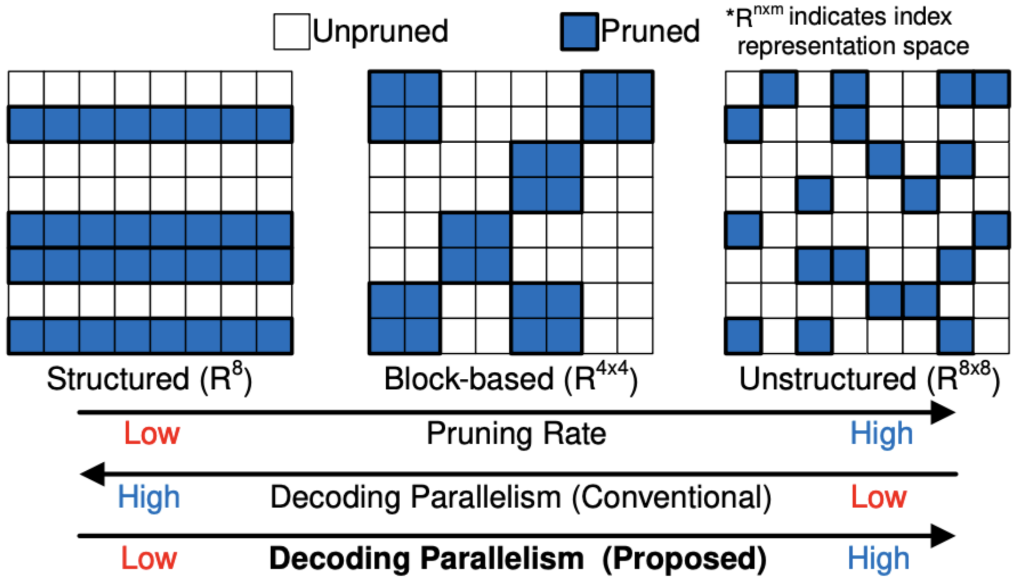
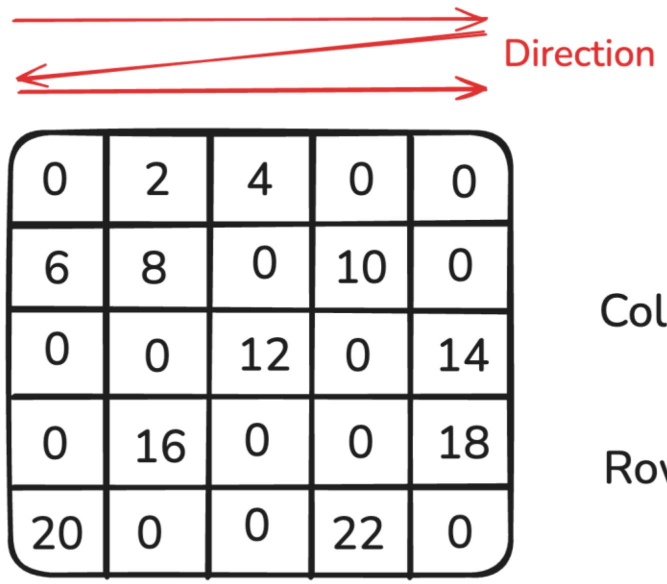

Unpruned
Pruned
*R^nxm indicates index representation space
Structured (R^8)
Block-based (R^4x4)
Unstructured (R^8x8)
Low
Pruning Rate
High
High
Decoding Parallelism (Conventional)
Low
Low
Decoding Parallelism (Proposed)
High

Figure 2: Several types of pruning granularity. In the conventional sparse formats, as a sparse matrix becomes more structured to gain parallelism in decoding, pruning rate becomes lower in general.

Background

❑ Sparse Matrix Indexing Overhead for CSR Encoding

- Required Memory(in Bytes): (2B+4B) × NNZ + 4B × (M+1) where,
  NNZ: Number of Non-Zero element
  M: Number of Rows
- As sparsity gets lower, NNZ increases. This leads to huge memory footprint in index.



$$\begin{array}{|c|c|c|c|c|}\hline 0 & 2 & 4 & 0 & 0 \\ \hline 6 & 8 & 0 & 10 & 0 \\ \hline 0 & 0 & 12 & 0 & 14 \\ \hline 0 & 16 & 0 & 0 & 18 \\ \hline 20 & 0 & 0 & 22 & 0 \\ \hline\end{array}$$

row-change @ 2   @5   @7   @9   @11

Values(FP16) = [2, 4, | 6, 8, 10, | 12, 14, | 16, 18, | 20, 22 |]

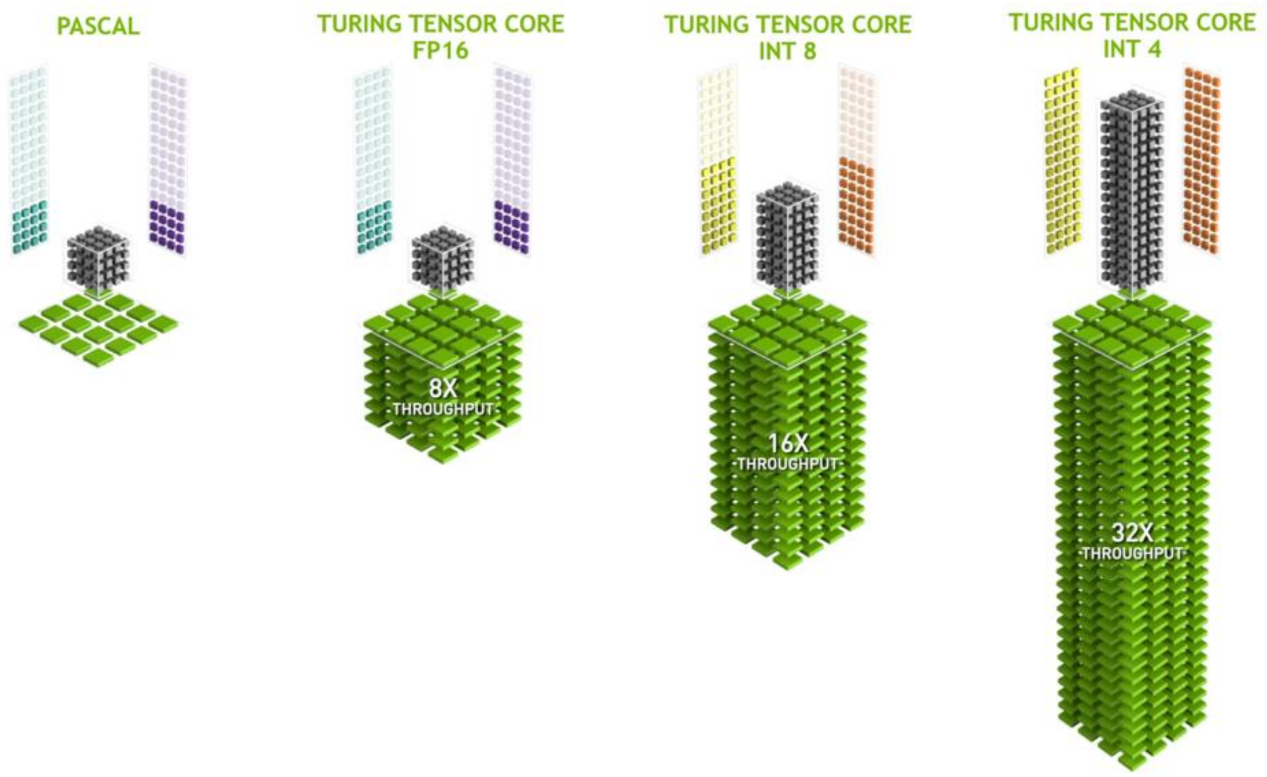
Column Indices(INT32) = [1, 2, 0, 1, 3, 2, 4, 1, 4, 0, 3]

Row-change Indices(INT32) = [0, 2, 5, 7, 9, 11]

Background

❑ Tensor Core

- Tensor Core is a specialized HW component for MMA operation.
- CUDA PTX supports (16 x 16) @ (16 x 8) + (16 x 8) operation for FP16.



HYPER ACCEL

Contributions

- SpInfer framework's contributions:
  - Introduce Tensor-Core-Aware Bitmap Encoding(TCA-BME) format for efficient Sparse Matrix Compression in low sparsity level.
  - Introduce SpMM kernel that utilize Nvidia Tensor Cores.

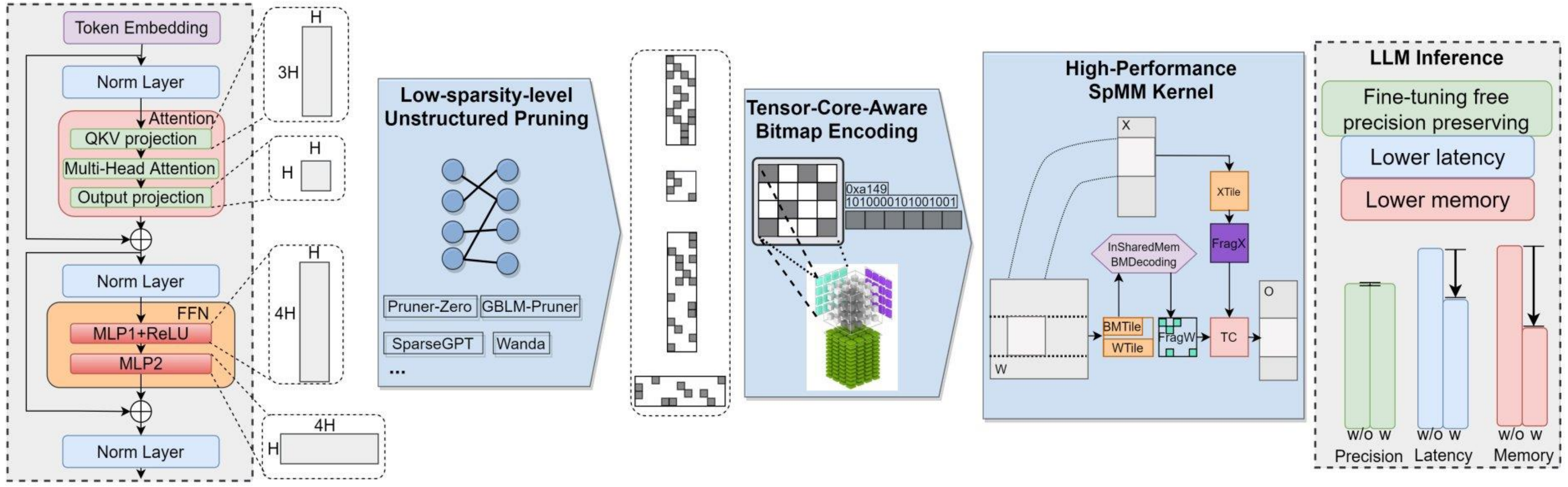
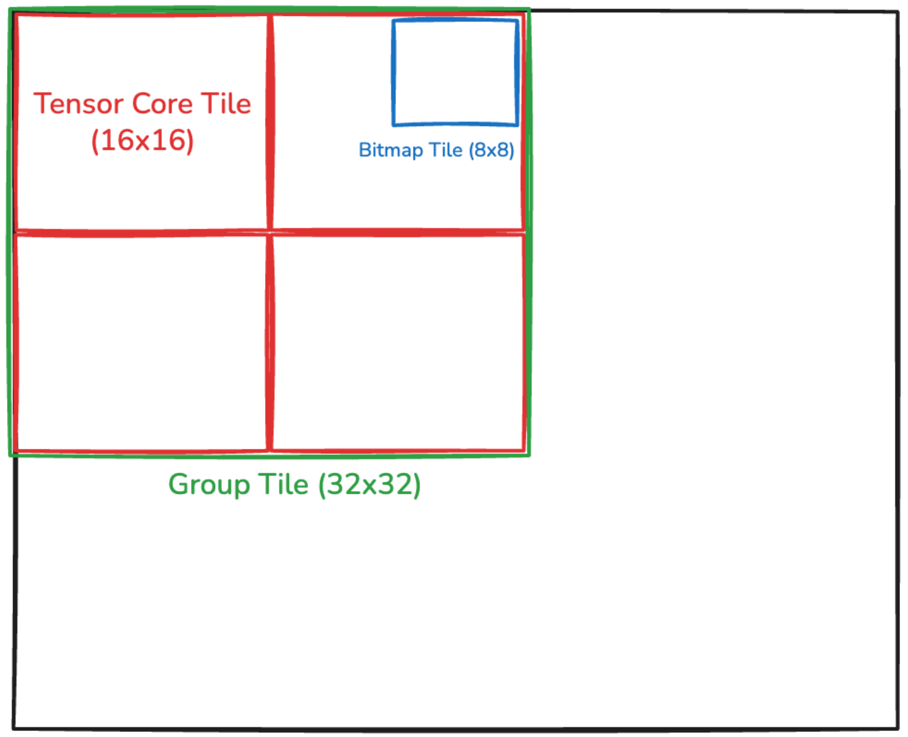


Figure 5. System overview of SpInfer.

Design Space Exploration

❑ Tensor-Core-Aware BitMap Encoding(TCA-BME)

- Sparse Matrix Encoding format(similar to CSR, COO) that targets tensor cores.
- Tiling Matrix into three levels: Bitmap Tile, Tensor Core Tile and Group tile.

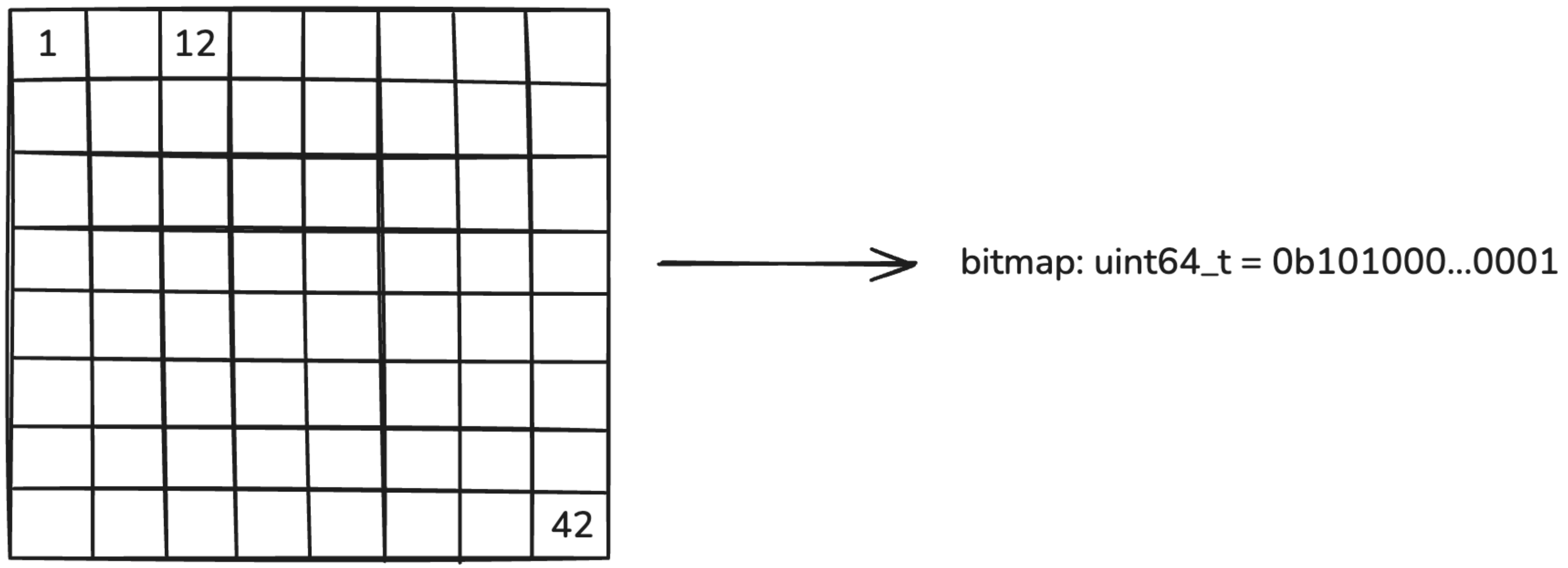


Sparse Weight Matrix (256x256)

Design Space Exploration

❑ Bitmap Tile (8x8)

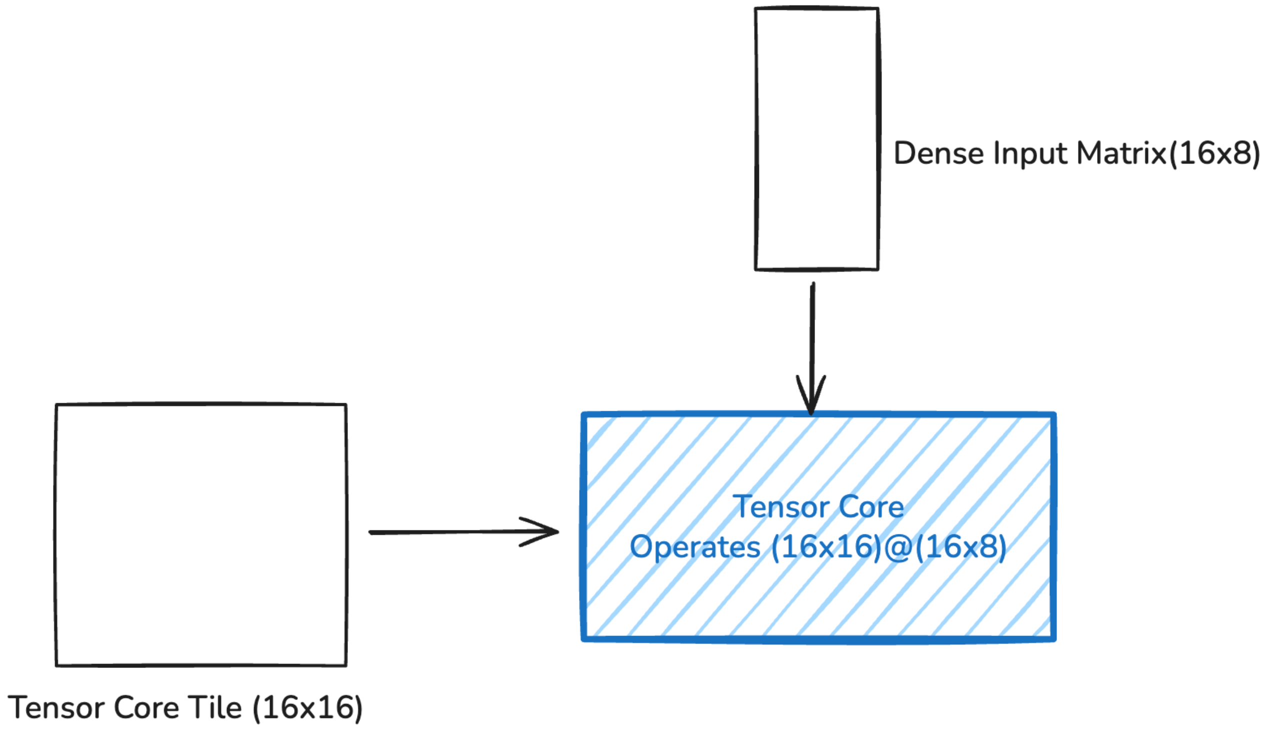
- Represent presence of non-zero values by bitmap.
- uint64_t is used to represent bitmap for 8x8 Tile.



Bitmap Tile(8x8)

Design Space Exploration

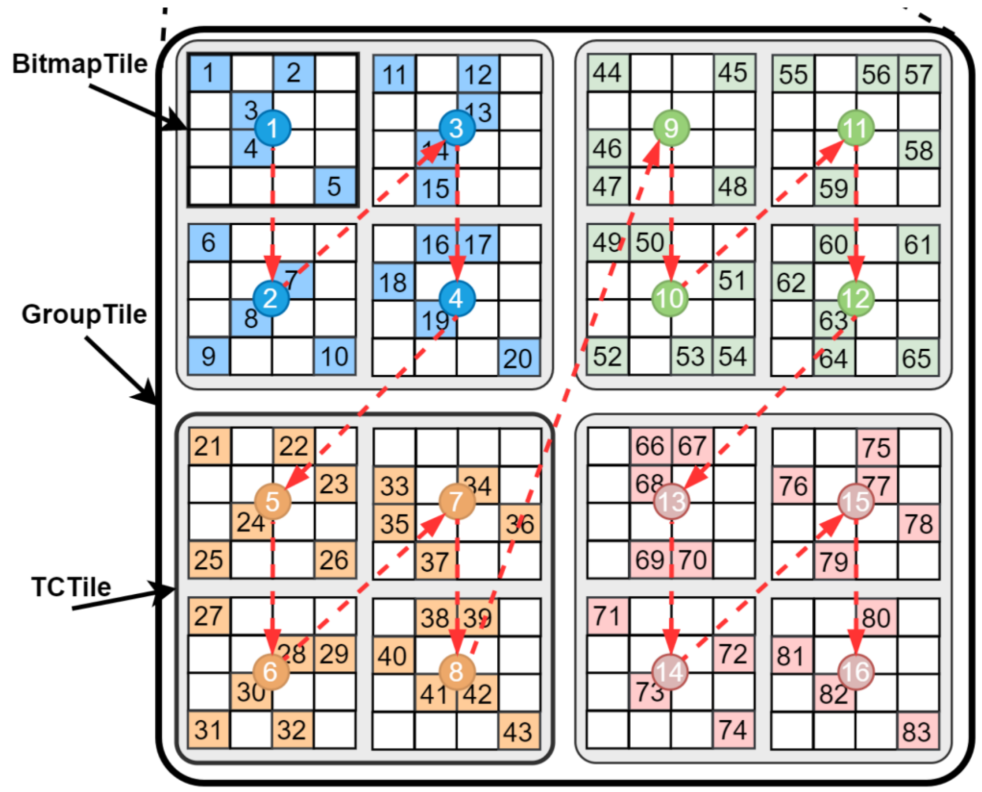
- ❑ Tensor Core Tile (16x16)
  - 2x2 Bitmap Tile consists a single Tensor Core Tile.
  - Tensor Core Tile fits into Tensor Core operation.



HYPER ACCEL

Design Space Exploration

- ❑ Group Tile (32x32)
  - 2x2 Tensor Core Tile consists a single Group Tile.
  - Figure below shows BitmapTile as 4x4, but it's actually 8x8.



HYPER ACCEL

Design Space Exploration

- TCA-BME(Tensor-Core-Aware Bit Map Encoding)
  - TCA-BME consists of three data structure: GroupTileOffsets(INT32), Values(FP16) and bitmaps(uint64_t)

Design Space Exploration

- TCA-BME(Tensor-Core-Aware Bit Map Encoding)

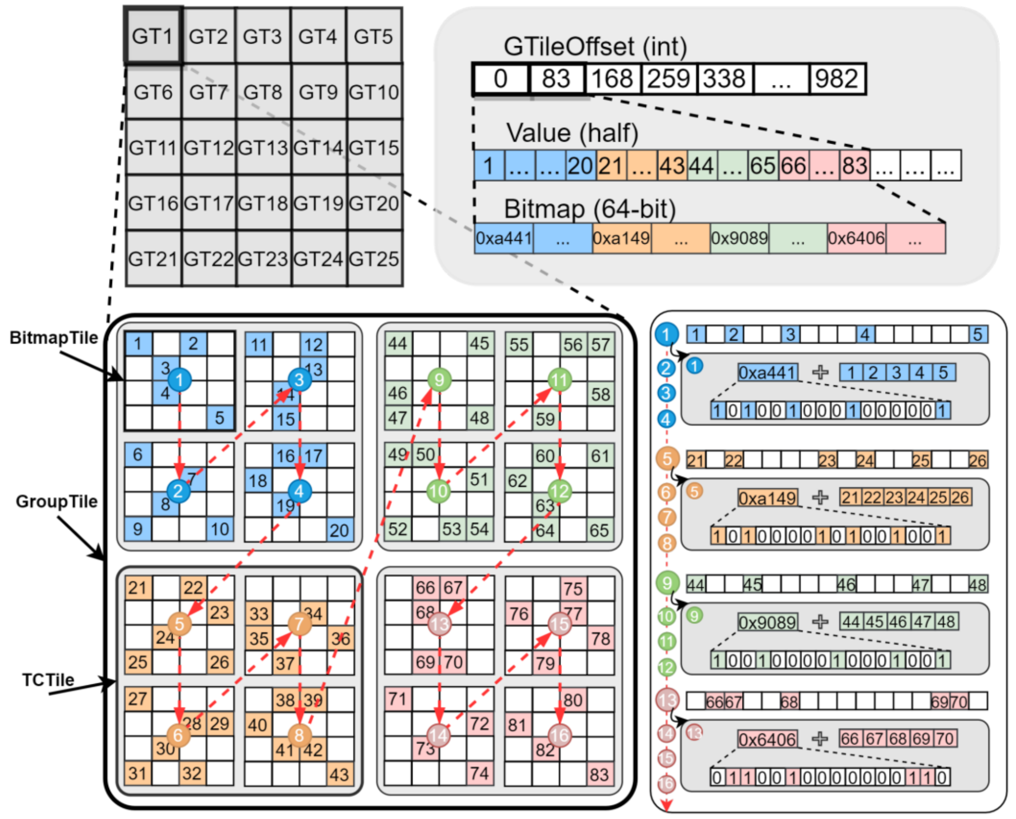


Figure 6. Tensor-Core-Aware Bitmap Encoding. BitmapTile is actually 8×8, shown as 4×4 for illustration.

Design Space Exploration

- Flow for SpInfer
  1. Load the GroupTile from Global Memory to Shared Memory.
  2. Decode GroupTile and load 16x16 matrix into register.
  3. Load dense Input Matrix(XTile) from Global Memory to Shared Memory.
  4. Load XTile from Shared Memory to register.
  5. Calculation in Tensor Core.

Design Space Exploration

❑ Flow for SpInfer

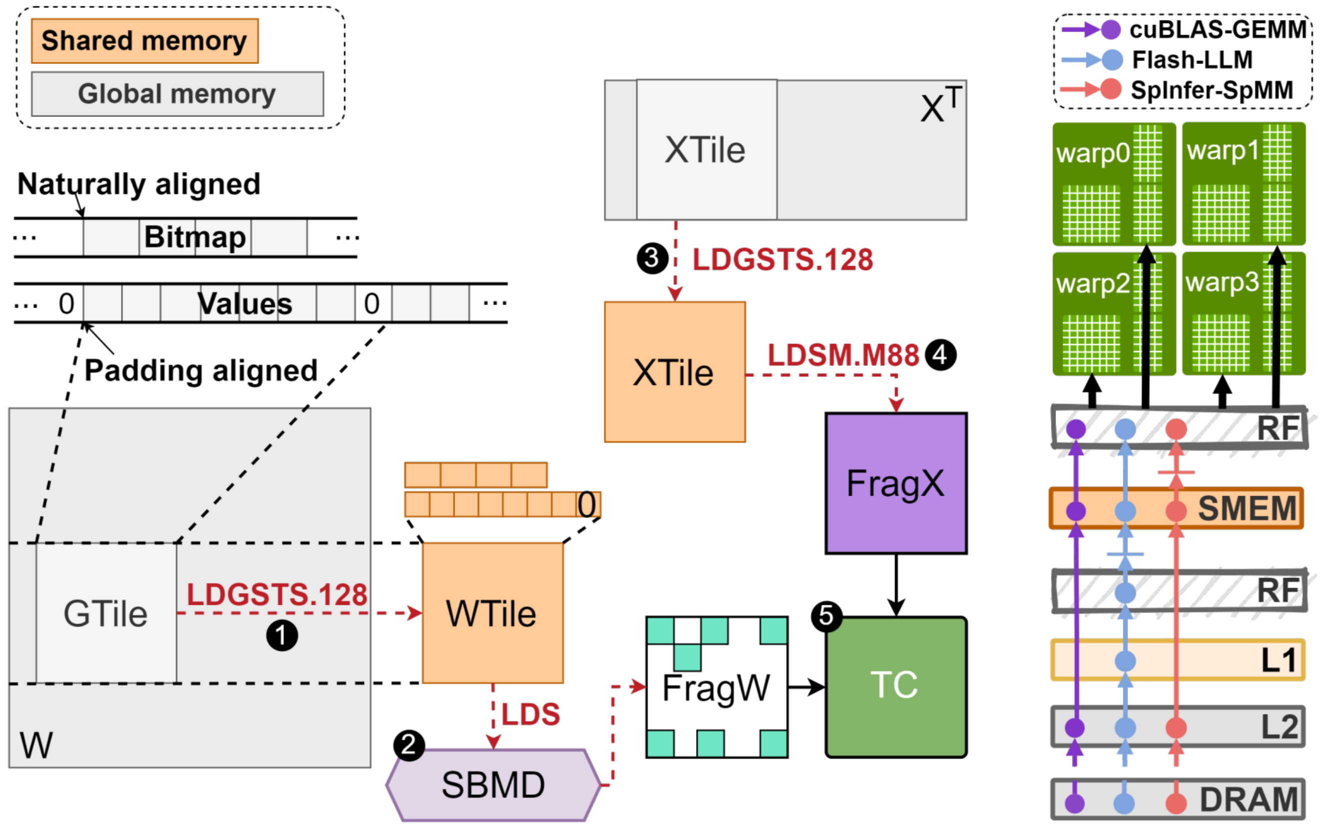


Figure 7. Data movement and instruction pipeline.

Design Space Exploration

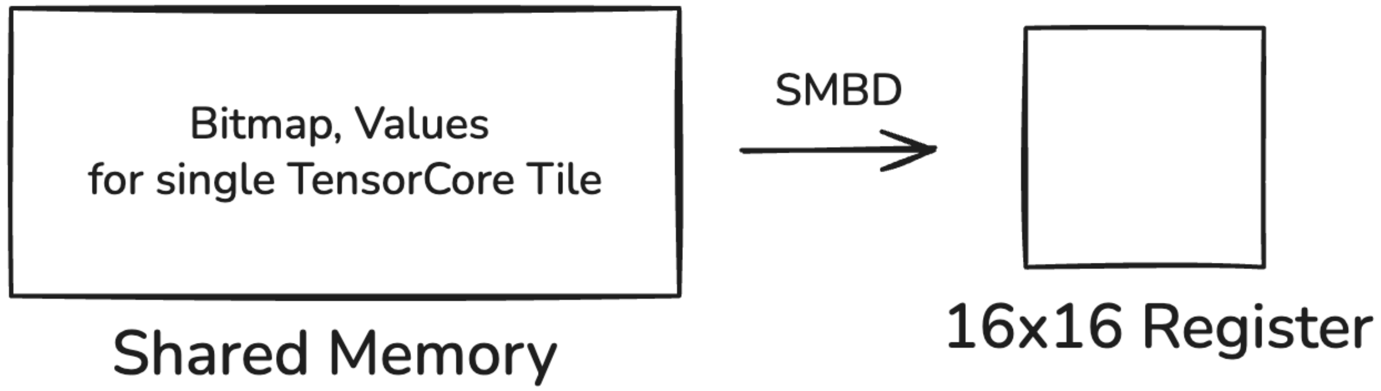
- Flow for SpInfer
    1. **Load the GroupTile from Global Memory to Shared Memory.**
    2. Decode GroupTile and load 16x16 matrix into register.
    3. Load dense Input Matrix(XTile) from Global Memory to Shared Memory.
    4. Load XTile from Shared Memory to register.
    5. Calculation in Tensor Core.

Design Space Exploration

- Flow for SpInfer
  1. Load the GroupTile from Global Memory to Shared Memory.
  2. **Decode GroupTile and load 16x16 matrix into register. ← SMBD**
  3. Load dense Input Matrix(XTile) from Global Memory to Shared Memory.
  4. Load XTile from Shared Memory to register.
  5. Calculation in Tensor Core.

Design Space Exploration

- Shared Memory Bitmap Decoding(SMBD)
  - Decode TCA-BME format
  - Load Sparse Matrix from Shared Memory to Registers.



Design Space Exploration

- SMBD for single TensorCoreTile(16x16)
  1. Store 2x2 uint64_t bitmap into registers. (Ra0, Ra1, Ra2, Ra3)
  2. Assign each thread to calculate the value offset:
     - Use "popcount()" to calculate the value offset of each BitmapTiles.
     - Use "maskedpopcount()" to calculate the value offset within each BitmapTiles.
  3. Load the values to fill 16x16 Registers.

HYPER ACCEL

Design Space Exploration

- SMBD for single TensorCoreTile(16x16)
  1. **Store 2x2 uint64_t bitmap into registers. (Ra0, Ra1, Ra2, Ra3)**
  2. Assign each thread to calculate the value offset:
     - Use "popcount()" to calculate the value offset of each BitmapTiles.
     - Use "maskedpopcount()" to calculate the value offset within each BitmapTiles.
  3. Load the values to fill 16x16 Registers.

Design Space Exploration

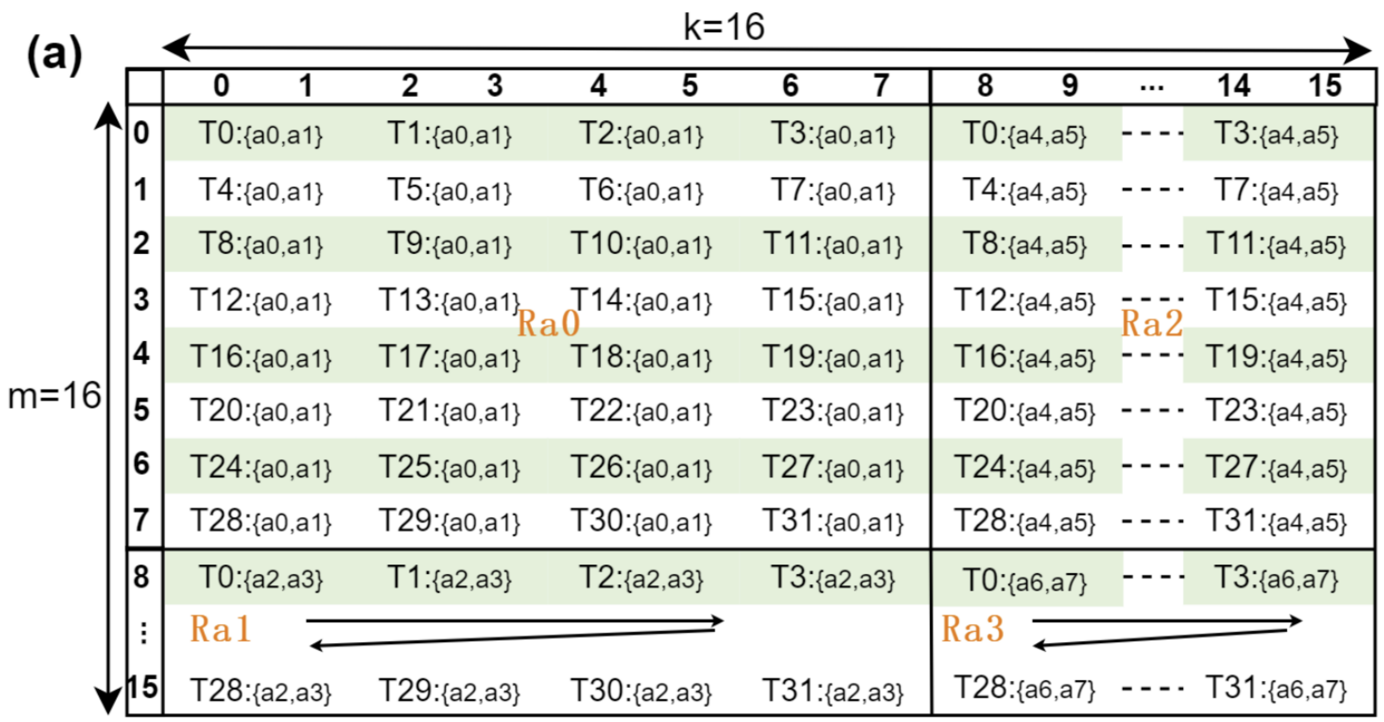
- SMBD for single TensorCoreTile(16x16)
  1. Store 2x2 uint64_t bitmap into registers. (Ra0, Ra1, Ra2, Ra3)
  2. **Assign each thread to calculate the value offset**:
     - Use "popcount()" to calculate the value offset of each BitmapTiles.
     - Use "maskedpopcount()" to calculate the value offset within each BitmapTiles.
  3. Load the values to fill 16x16 Registers.

HYPER ACCEL

Design Space Exploration

- SMBD for single TensorCoreTile(16x16)
  - Distribute the calculation to 32 threads(1 warp)
  - A thread handles eight elements.(ex. T0 loading each to {a0, a1, …, a7})

(a)



	0 1	2 3	4 5	6 7	8 9	…	14 15
0	T0:{a0,a1}	T1:{a0,a1}	T2:{a0,a1}	T3:{a0,a1}	T0:{a4,a5}	----	T3:{a4,a5}
1	T4:{a0,a1}	T5:{a0,a1}	T6:{a0,a1}	T7:{a0,a1}	T4:{a4,a5}	----	T7:{a4,a5}
2	T8:{a0,a1}	T9:{a0,a1}	T10:{a0,a1}	T11:{a0,a1}	T8:{a4,a5}	----	T11:{a4,a5}
3	T12:{a0,a1}	T13:{a0,a1}	T14:{a0,a1}	T15:{a0,a1}	T12:{a4,a5}	----	T15:{a4,a5}
4	T16:{a0,a1}	T17:{a0,a1}	T18:{a0,a1}	T19:{a0,a1}	T16:{a4,a5}	----	T19:{a4,a5}
5	T20:{a0,a1}	T21:{a0,a1}	T22:{a0,a1}	T23:{a0,a1}	T20:{a4,a5}	----	T23:{a4,a5}
6	T24:{a0,a1}	T25:{a0,a1}	T26:{a0,a1}	T27:{a0,a1}	T24:{a4,a5}	----	T27:{a4,a5}
7	T28:{a0,a1}	T29:{a0,a1}	T30:{a0,a1}	T31:{a0,a1}	T28:{a4,a5}	----	T31:{a4,a5}
8	T0:{a2,a3}	T1:{a2,a3}	T2:{a2,a3}	T3:{a2,a3}	T0:{a6,a7}	----	T3:{a6,a7}
⋮							
15	T28:{a2,a3}	T29:{a2,a3}	T30:{a2,a3}	T31:{a2,a3}	T28:{a6,a7}	----	T31:{a6,a7}

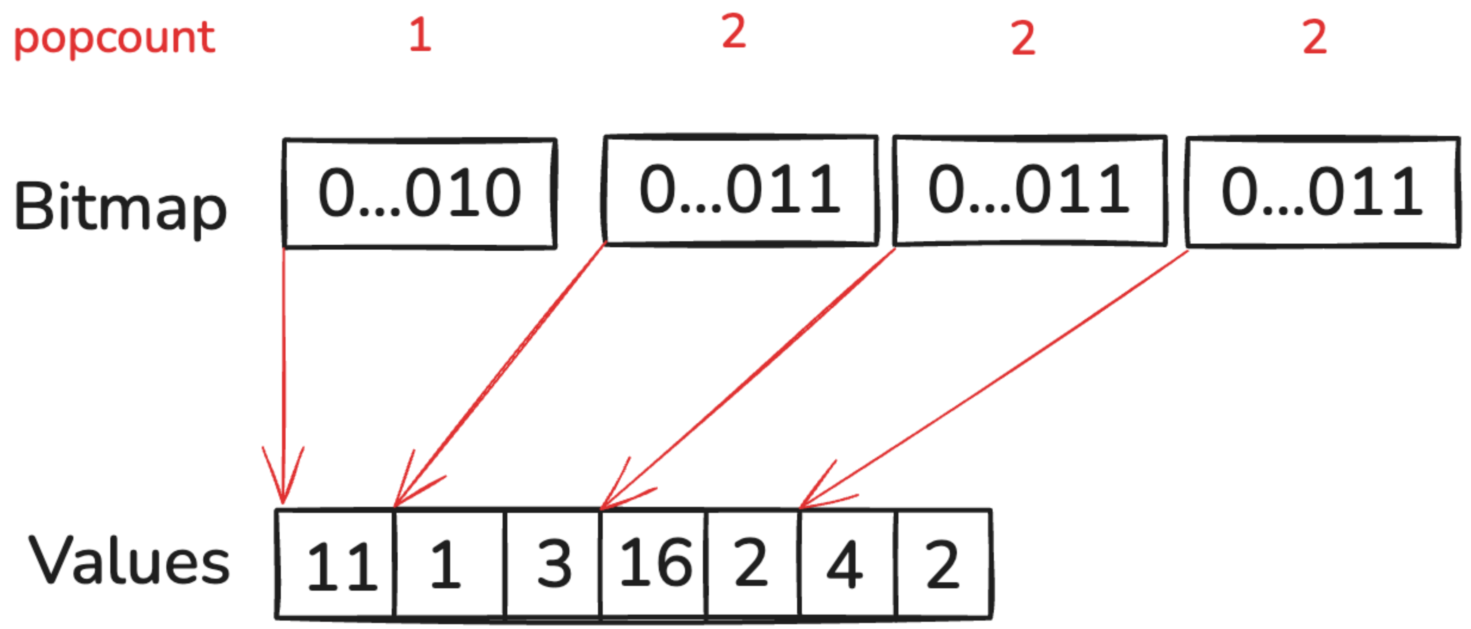
Design Space Exploration

- SMBD for single TensorCoreTile(16x16)
  1. Store 2x2 uint64_t bitmap into registers. (Ra0, Ra1, Ra2, Ra3)
  2. Assign each thread to calculate the value offset:
     - **Use "popcount()" to calculate the value offset of each BitmapTiles.**
     - Use "maskedpopcount()" to calculate the value offset within each BitmapTiles.
  3. Load the values to fill 16x16 Registers.

HYPER ACCEL

Design Space Exploration

- SMBD for single TensorCoreTile(16x16)
  - "popcount()" is an CUDA ISA that counts number of "1" bits in 64-bit bitmap.
  - Using "popcount()", we can calculate the values offset for each BitmapTile.

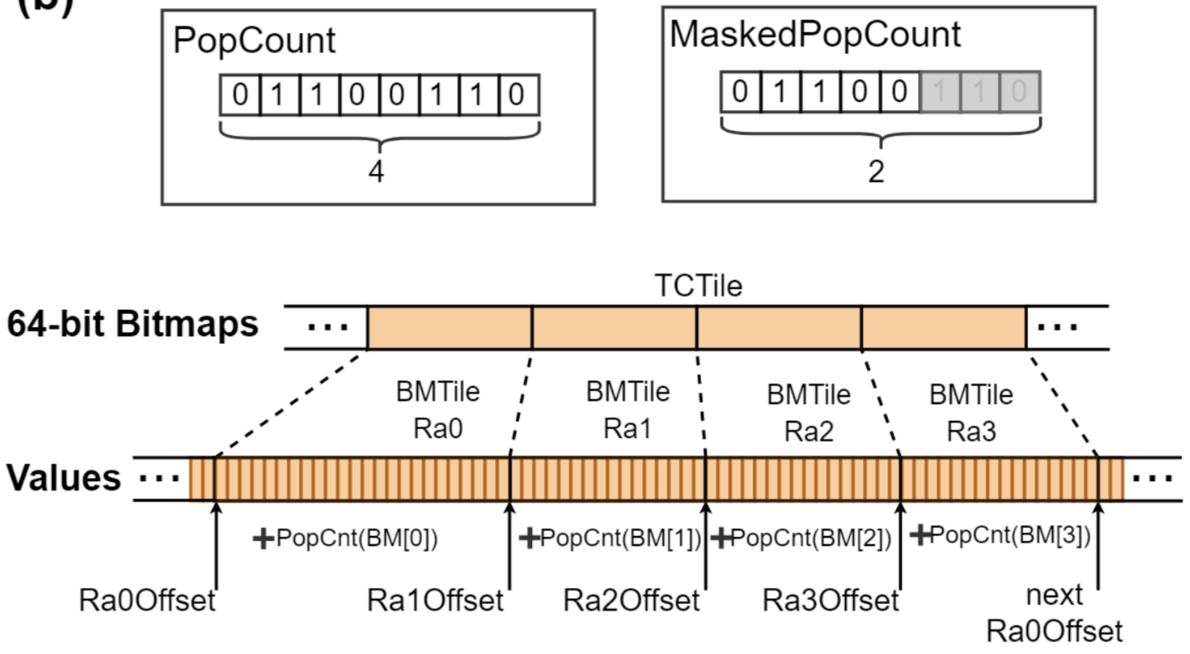


HYPER ACCEL

Design Space Exploration

- SMBD for single TensorCoreTile(16x16)
  - "popcount()" is an CUDA ISA that counts number of "1" bits in 64-bit bitmap.
  - Using "popcount()", we can calculate the values offset for each BitmapTile.

(b)



HYPER ACCEL

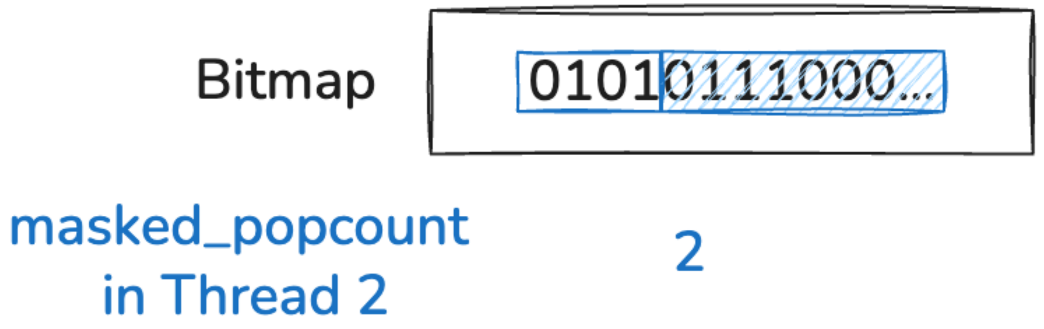
Design Space Exploration

- SMBD for single TensorCoreTile(16x16)
  1. Store 2x2 uint64_t bitmap into registers. (Ra0, Ra1, Ra2, Ra3)
  2. Assign each thread to calculate the value offset:
     - Use "popcount()" to calculate the value offset of each BitmapTiles.
     - **Use "maskedpopcount()" to calculate the value offset within each BitmapTiles.**
  3. Load the values to fill 16x16 Registers.

HYPER ACCEL

Design Space Exploration

- SMBD for single BitmapTile(8x8)
  - "masked_popcount()" is a custom kernel that is similar to "popcount()".

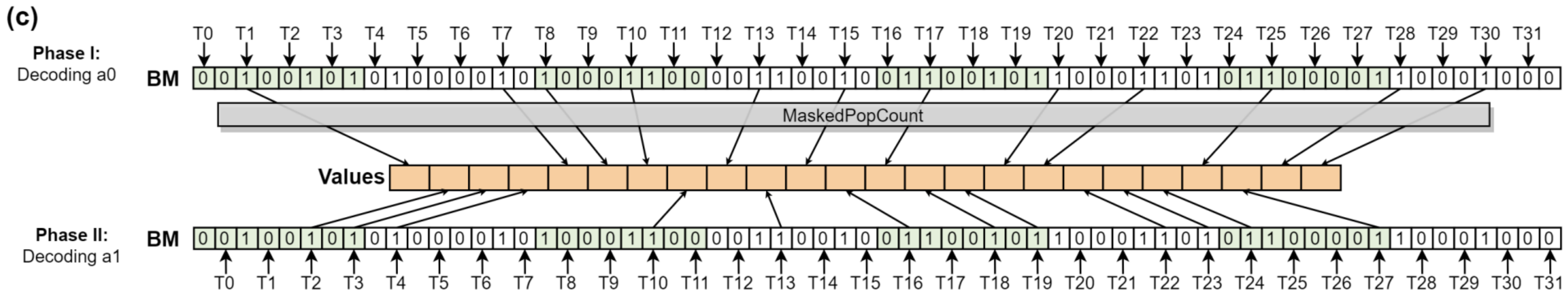


Algorithm 2 MaskedPopCount pseudo code

```
Input: Bitmap $b$, Thread Lane ID $l$
Output: Count of preceding ones $count$
1: $offset = l \times 2$                      ▹ Calculate base offset
2: $mask = (1 \ll offset) - 1$                ▹ Generate preceding mask
3: $count = PopCount(b \& mask)$              ▹ Count ones before offset
4: return $count$
```

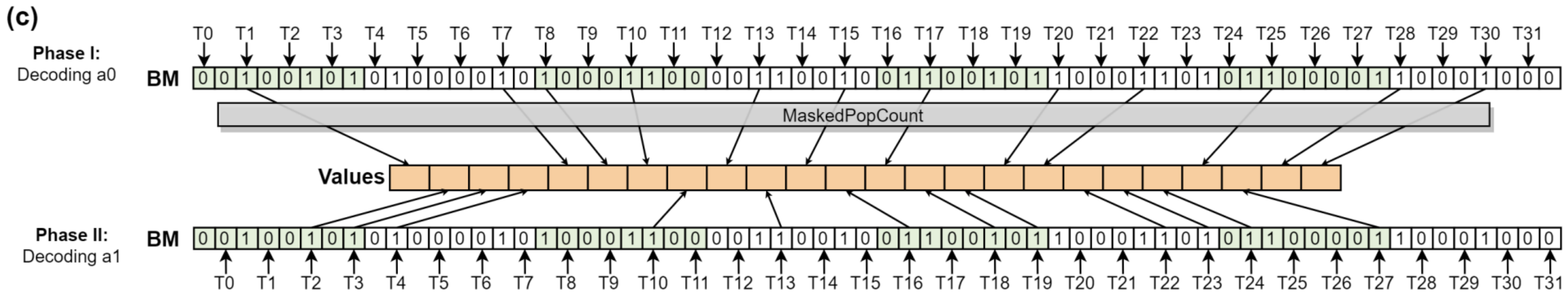
Design Space Exploration

- SMBD for single BitmapTile(8x8)
  - By using "masked_popcount()", each thread can calculate the Value offset within each BitmapTiles.



Design Space Exploration

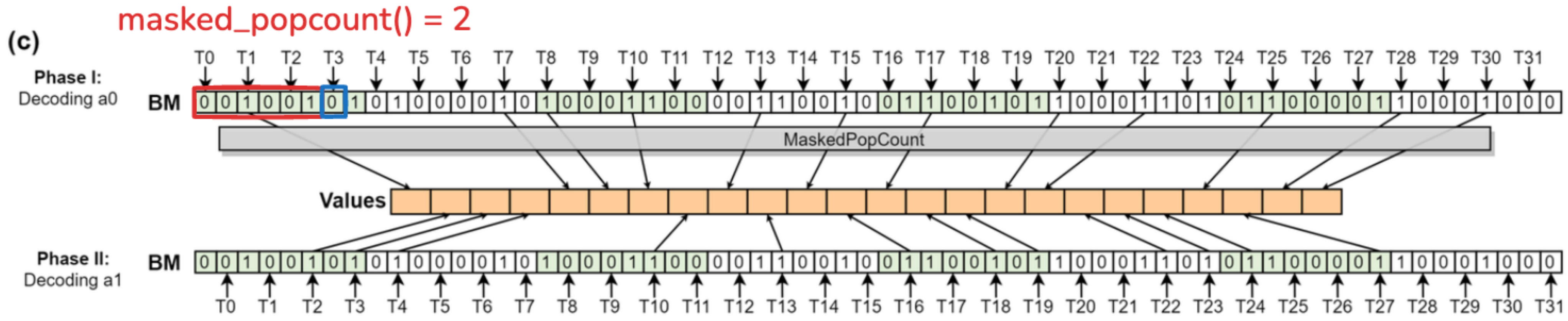
- SMBD for single BitmapTile(8x8)
  - Phase 1. Compute masked_popcount() to get a0.
  - Phase 2. Reuse the computed value from Phase 1, and get a1.



Design Space Exploration

- SMBD for single BitmapTile(8x8)
  - **Phase 1. Compute masked_popcount() to get a0.**
  - Phase 2. Reuse the computed value from Phase 1, and get a1.
  - Example. Phase 1 for "Thread 3"
    → masked_popcount() is 2. But since Bitmap's bit is "0", a0 is zero-value.

Phase 1 for Thread 3

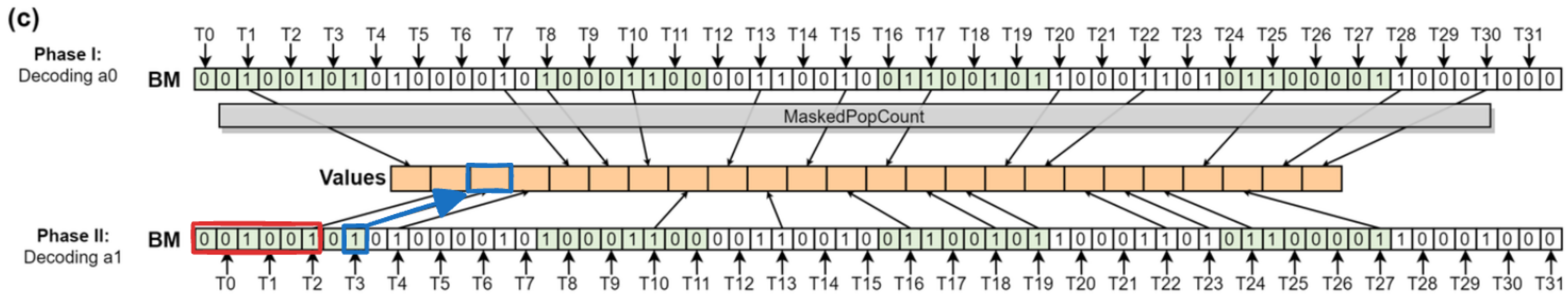


masked_popcount() = 2
(c)
Phase I:
Decoding a0
BM
T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15 T16 T17 T18 T19 T20 T21 T22 T23 T24 T25 T26 T27 T28 T29 T30 T31
MaskedPopCount
Values
Phase II:
Decoding a1
BM

Design Space Exploration

- SMBD for single BitmapTile(8x8)
  - Phase 1. Compute masked_popcount() to get a0.
  - **Phase 2. Reuse the computed results from Phase 1, and get a1.**
  - Example. Phase 2 for "Thread 3"
    → At Phase 1, a0 was zero-value and masked_popcount() was 2. So the Values offset for a1 is 2.

Phase 2 for Thread 3



Design Space Exploration

- Pipelining the SpInfer flow
  - Maintain two buffers of SHEM&Registers for pipelining.

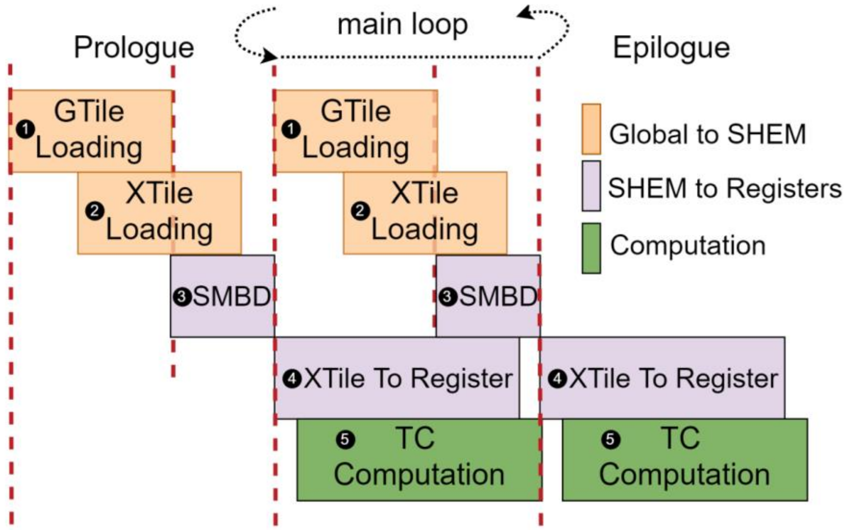


Figure 9. Schematic representation of the asynchronous pipeline design. The pipeline depth is 2.

Evaluation

- E2E Environment 1
  - Intel Xeon Platinum 8352V CPU(2.10 GHz)
  - 4 Nvidia RTX4090 GPUs connected via PCIe
- E2E Environment 2
  - Xeon Gold 6133 CPU(2.5 GHz)
  - 4 Nvidia A6000 GPU connected via NVLink

HYPER ACCEL

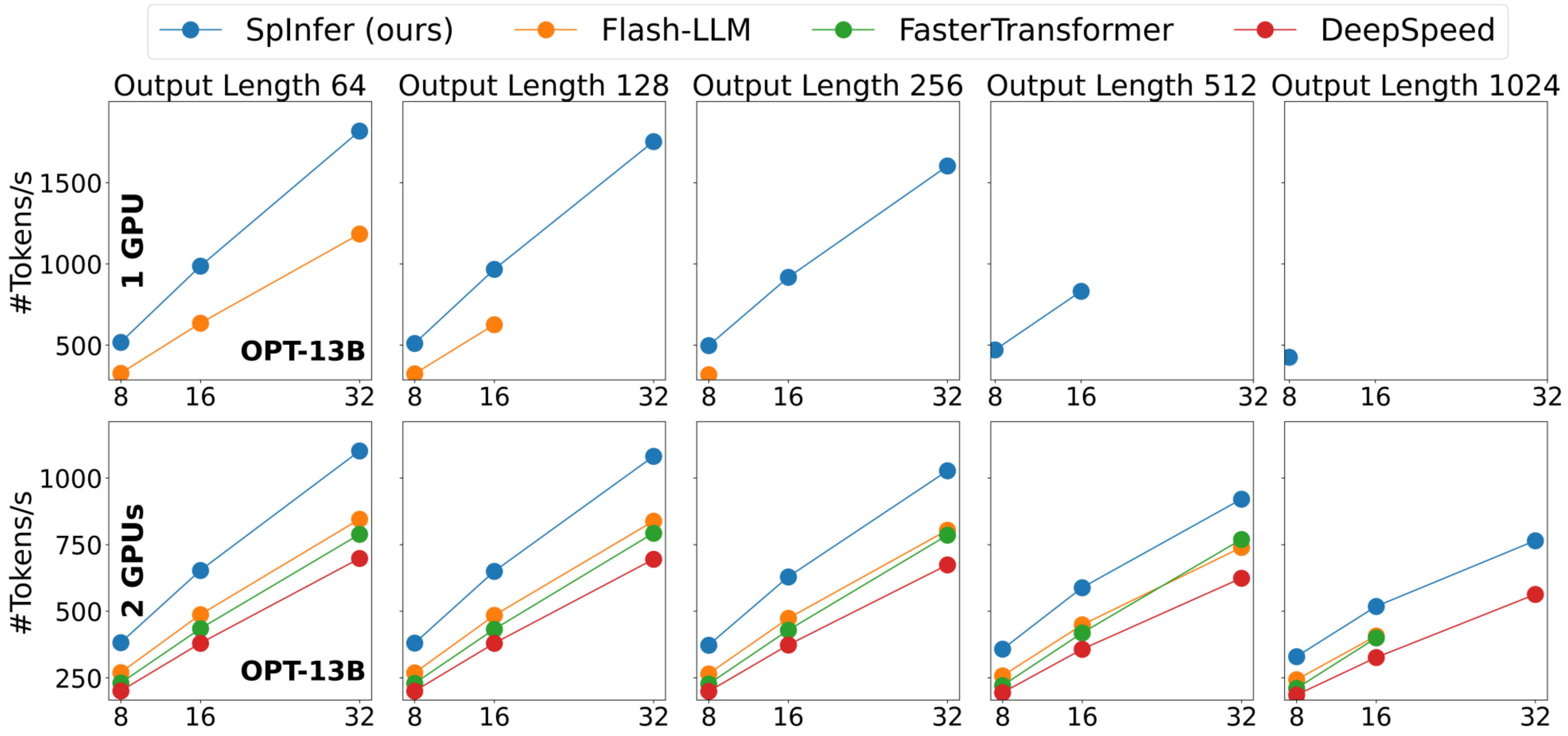
Evaluation

❑ Environments Details

- Compared with Flash-LLM, DeepSpeed, FasterTransformer.
- Using OPT-13B, OPT-30B, OPT-66B model.
- Use Wanda algorithm for unstructured pruning, setting sparsity to 60%.
- Execution time is measured by average Wall-clock-time of 100 iterations.

Evaluation

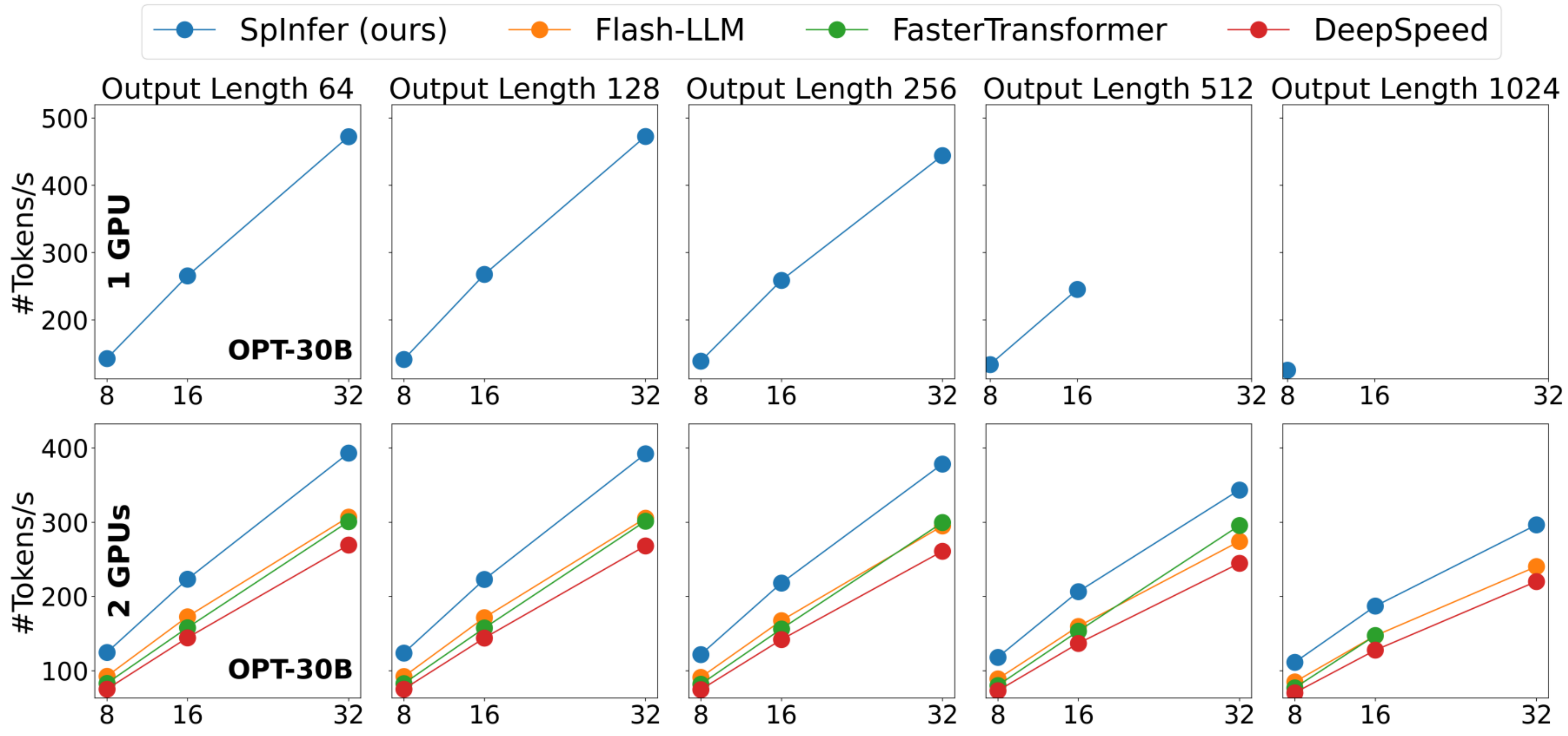
- ❑ Evaluation on RTX4090 GPUs



HYPER ACCEL

Evaluation

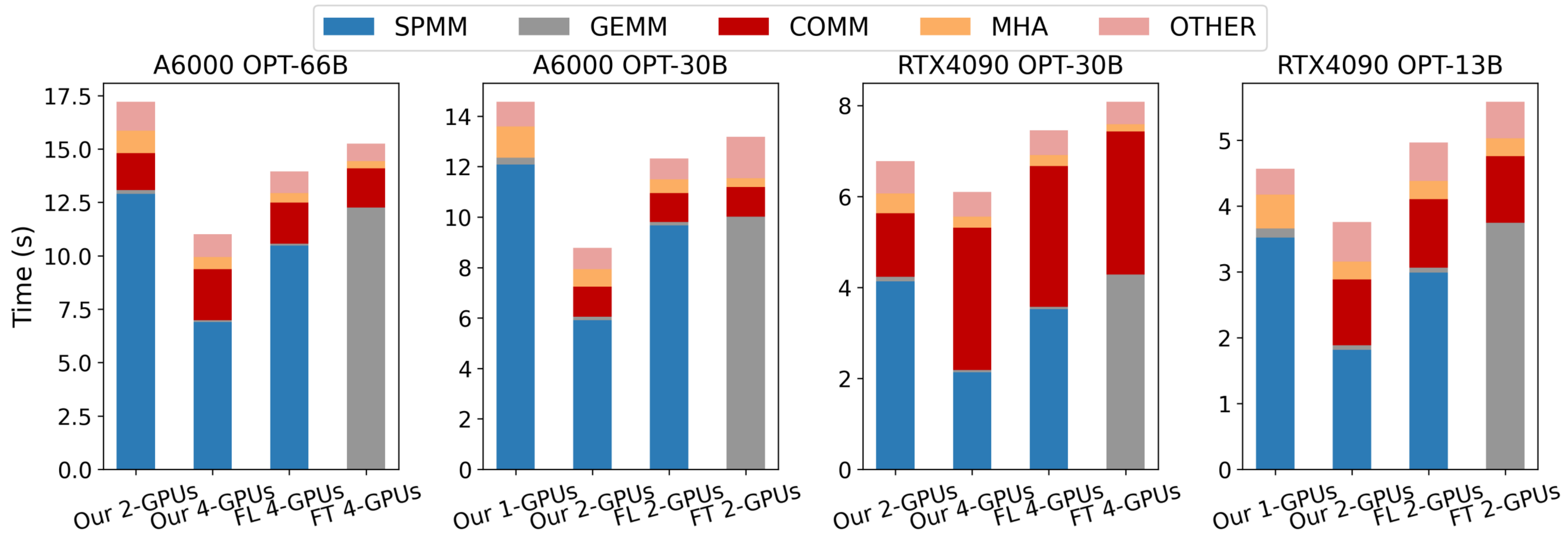
- Evaluation on A6000 GPUs



HYPER ACCEL

Evaluation

❑ E2E Execution Time Breakdown



HYPER ACCEL

Evaluation

- Limitations
  - Ineffective in long-prefill phase due to SMBD compute overhead.
  - CuBLAS_TC(Dense) doesn't need any complex decoding during transmitting data from shared_memory to registers.
  - FYI. "NxMxK" mma means (NxK) @ (KxM) + (NxM) operation.

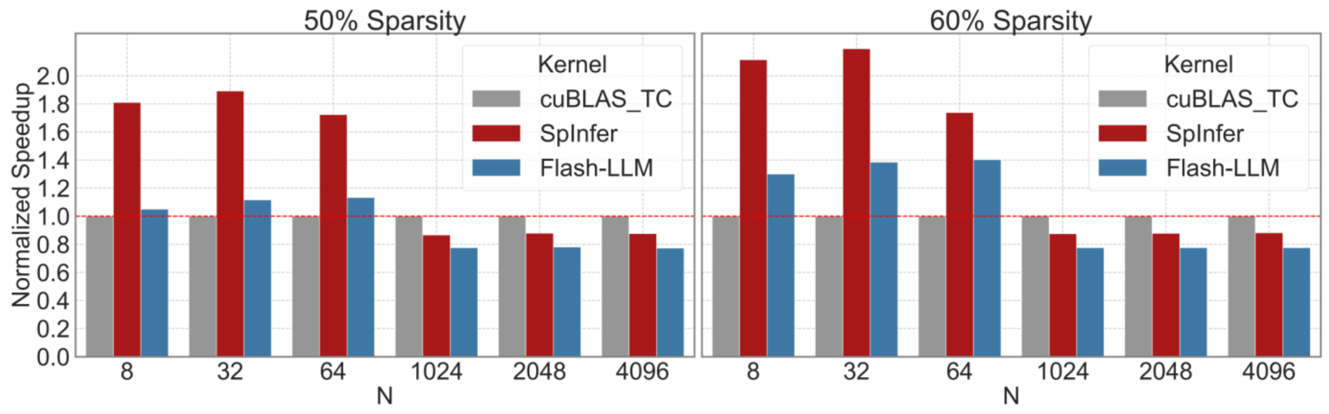


Figure 16. Performance comparison under small and large N settings. $M = 28672$ and $K = 8192$.

HYPER ACCEL

Conclusion

- ❑ SpInfer successfully reduces memory usage. (weight memory)
- ❑ TCA-BME concept can be used for various Accelerators.
- ❑ Used for disaggregated serving system
  - Prefill phase: cuBLAS_TC
  - Decode phase: SpInfer

HYPER ACCEL