Efficient Quantized Sparse Matrix Operations on Tensor Cores

Shigang Li[*], Kazuki Osawa[+], Torsten Hoefler[+]
[*]School of Computer Science, Beijing University of Posts and Telecommunications
[+]Department of Computer Science, ETH Zurich

SC22 Dallas, TX | hpc accelerates.

Efficient Quantized Sparse Matrix Operations on Tensor Cores

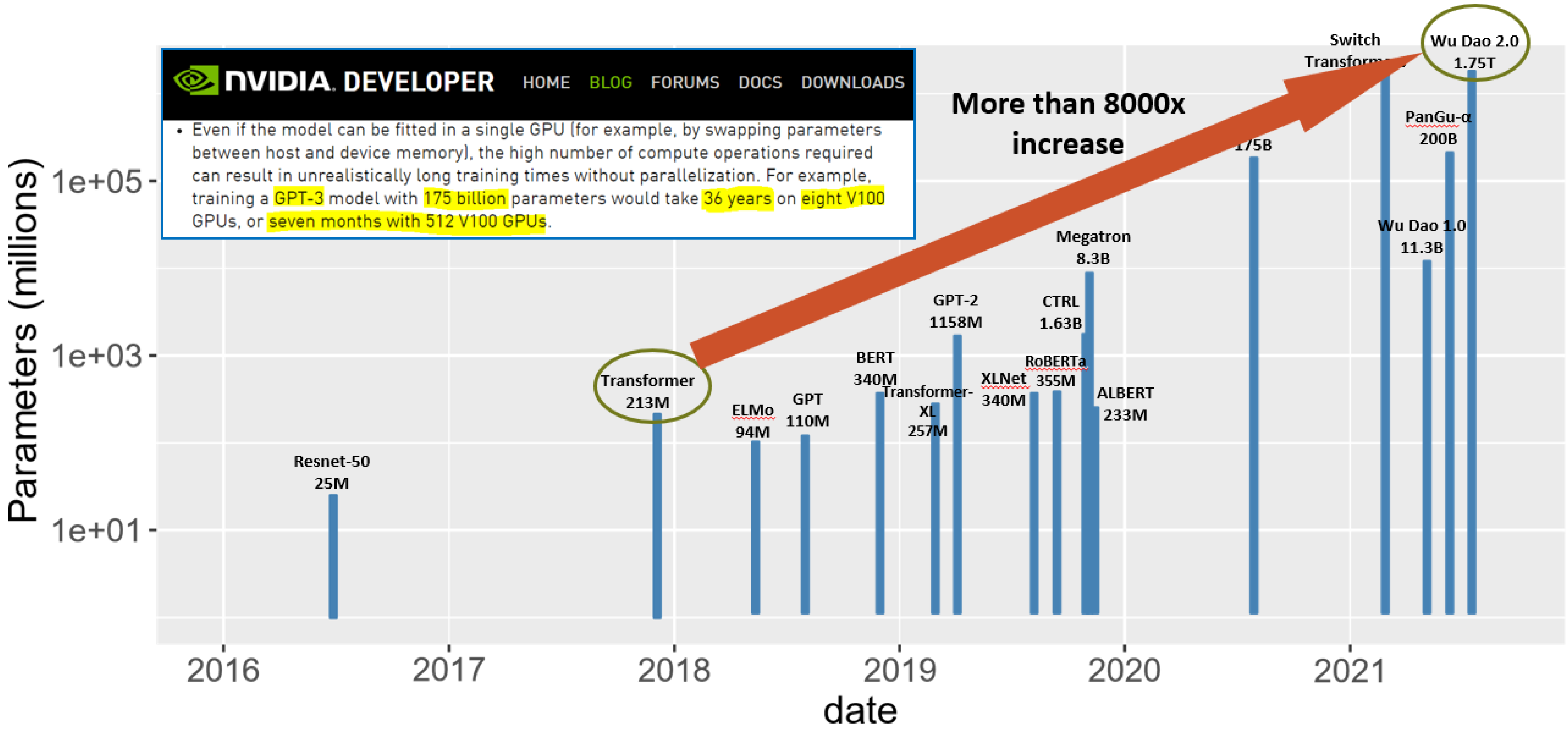
Shigang Li, Kazuki Osawa, Torsten Hoefler



SC22, Dallas, TX, USA

Nov. 2022

Model size is growing exponentially



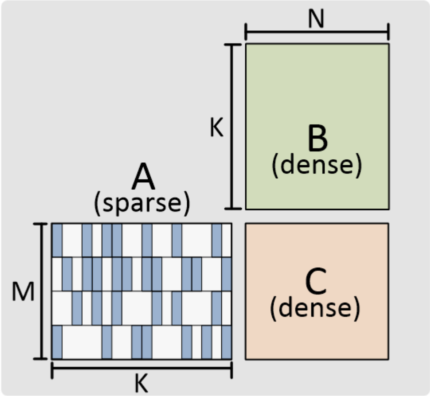
Models are also compressible

- **Sparsification**

SpMM

1. Self-attention in **sparse Transformers**
2. Forward pass of **pruned models**

...



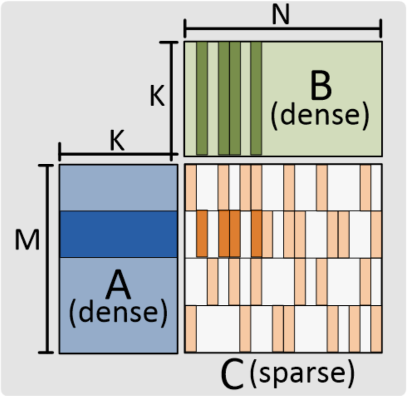
Sparsity in scientific: > 99%

Sparsity in DL: 50% ~ 90%

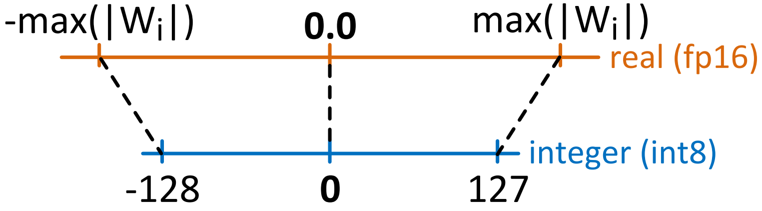
SDDMM

1. Attention score in **sparse Transformers**
2. Backward pass of **pruned models**

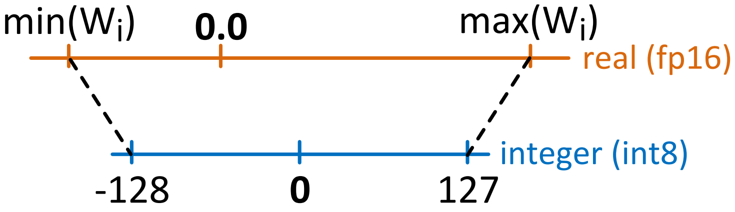
...



- **Quantization**



Uniform symmetric quantization

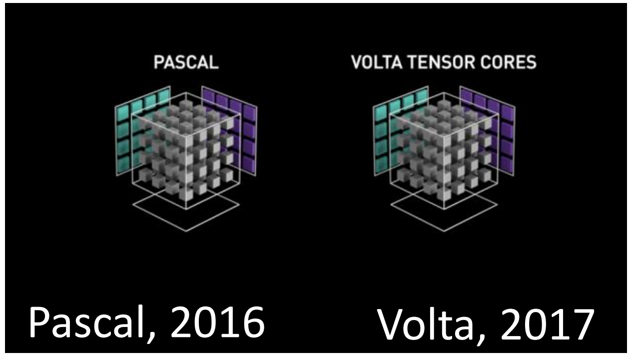


Uniform asymmetric quantization

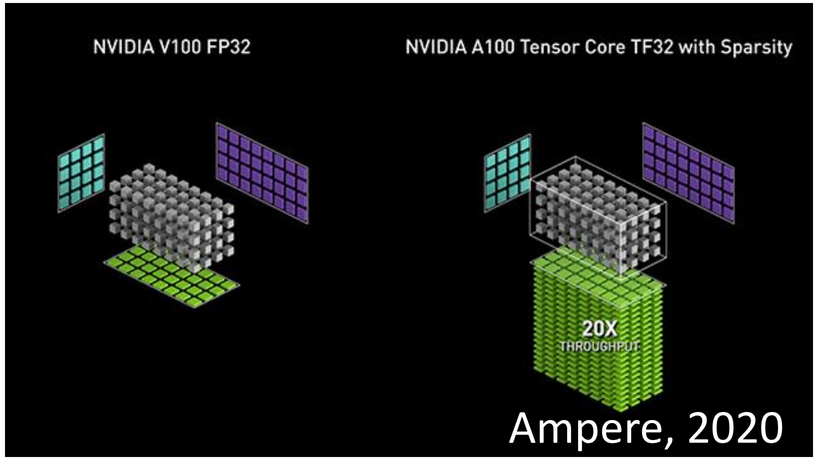
- **Combining sparsification with quantization**

Mart van Baalen et al., Bayesian bits: Unifying quantization and pruning**, NeurIPS 2020**
H. Yang et al., Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization based approach**, CVPR 2020**
S. Han et al., Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding**, ICLR 2016**
...

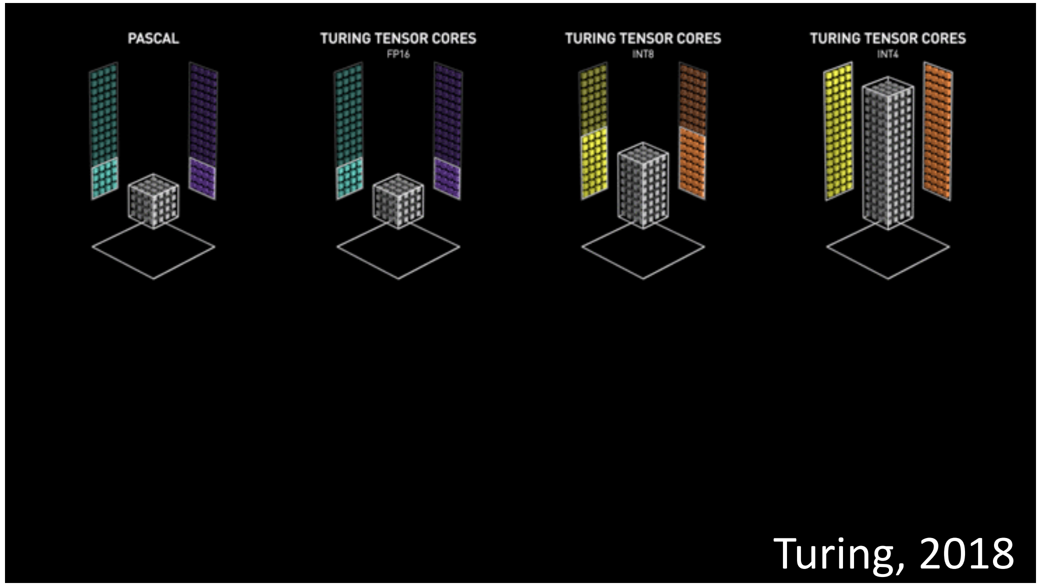
Tensor cores for deep learning acceleration



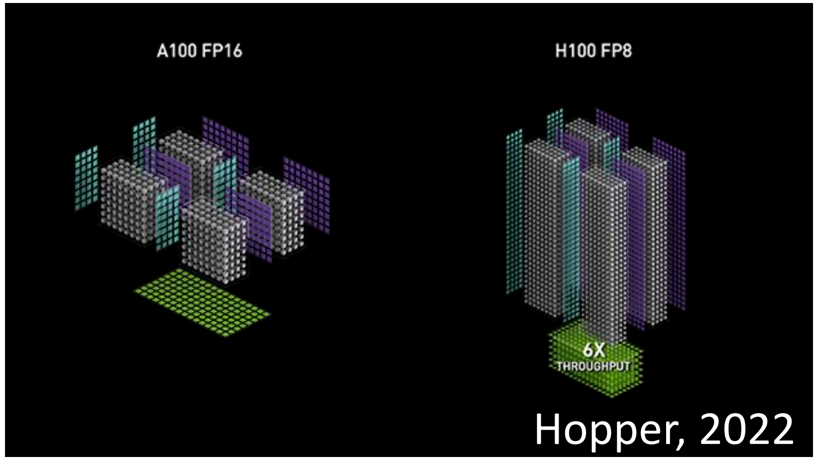
(18.7 TFLOPS for fp16) (125 TFLOPS for fp16)



(624 TOPS for int8)



(261 TOPS for int8)

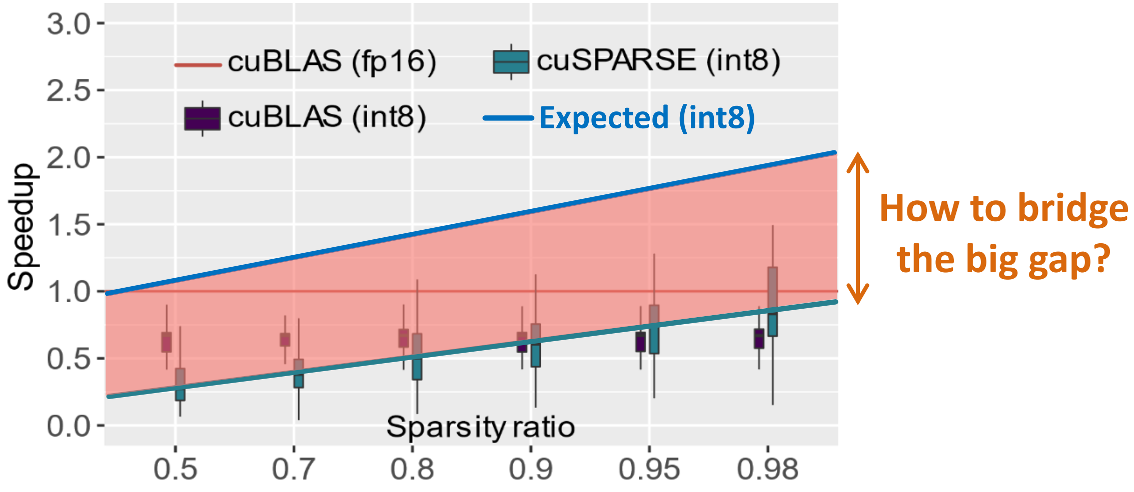


(2,000 TOPS for int8)

Images and GIFs in this slide are from ***https://www.nvidia.com/en-us/data-center/tensor-cores/***

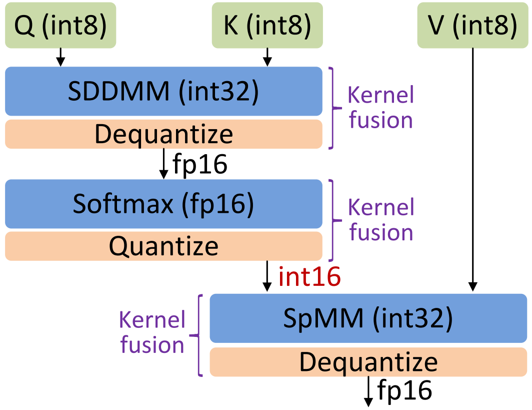
Challenges

(1) How to achieve practical speedup in a large range of sparsity ratio, e.g., **50% ~ 98%**?



SpMM with 2x1 block sparsity on A100 tensor cores, using 1,536 sparse matrices from DL domain

(2) How to efficiently support sparse workloads with **mixed precision (two input matrices with different precision)**, e.g., **8-bit weights and 4-bit activation**?



Sparse self-attention with mixed precision

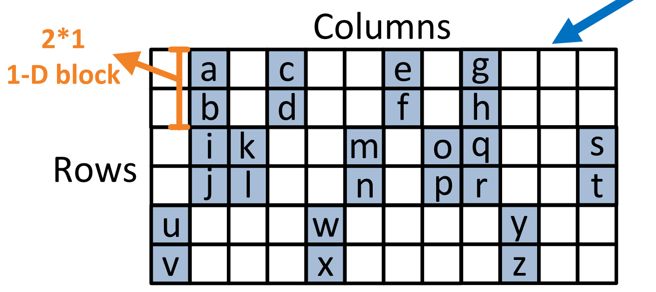
	Hopper	Ampere	Turing	Volta
Supported Tensor Core precisions	FP64, TF32, bfloat16, FP16, FP8, INT8	FP64, TF32, bfloat16, FP16, INT8, INT4, INT1	FP16, INT8, INT4, INT1	FP16

Two input matrices must be the same precision

Libraries of sparse matrix computation

[1] Mixed precision means two input matrices with different precision

Library	Precision: fp16	Precision: int8	Precision: int4	Precision: mixed[1]	Sparsity: granularity	Sparsity: DL-friendly?	Tensor Core
cuSPARSE [10]	✓	✓	✗	✗	fine-grained	👎	👎
	✓	✓	✗	✗	block	👍	👍
cuSPARSELt [11]	✓	✓	✓	✗	2:4 structured	👍	👍
Sputnik [13]	✓	✗	✗	✗	fine-grained	👍	👎
vectorSparse [14]	✓	✗	✗	✗	1-D block	👍	👍
Magicube (ours)	✗	✓	✓	✓	1-D block	👍	👍



Sparse matrix with 1-D non-zero blocks

Data layout of *m8n8k16* for *int8 mma* on Tensor Cores

Thread0 provides **b_{00}**, **b_{01}**, **b_{02}**, and **b_{03}**, and each **b_{xx}** is an **8-bit integer**

B
(column-major)

k = 16

Row \ Col	0	1	…	7
0 1 2 3	T0: {b_{00}, b_{10}, b_{20}, b_{30}}	T4: {b_{01}, b_{11}, b_{21}, b_{31}}		T28: {b_{07}, b_{17}, b_{27}, b_{37}}
4 5 6 7	T1: {b_{40}, b_{50}, b_{60}, b_{70}}	T5: {b_{41}, b_{51}, b_{61}, b_{71}}		T29: {b_{47}, b_{57}, b_{67}, b_{77}}
8 9 10 11	T2: {b_{80}, b_{90}, b_{a0}, b_{b0}}	T6: {b_{81}, b_{91}, b_{a1}, b_{b1}}		T30: {b_{87}, b_{97}, b_{a7}, b_{b7}}
12 13 14 15	T3: {b_{c0}, b_{d0}, b_{e0}, b_{f0}}	T7: {b_{c1}, b_{d1}, b_{e1}, b_{f1}}		T31: {b_{c7}, b_{d7}, b_{e7}, b_{f7}}

n = 8

Thread0 provides **a_{00}**, **a_{01}**, **a_{02}**, and **a_{03}**, and each **a_{xx}** is an **8-bit integer**

A
(row-major)

m = 8

Row \ Col	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
0	T0: {a_{00}, a_{01}, a_{02}, a_{03}}	T1: {a_{04}, a_{05}, a_{06}, a_{07}}	T2: {a_{08}, a_{09}, a_{0a}, a_{0b}}	T3: {a_{0c}, a_{0d}, a_{0e}, a_{0f}}
1	T4: {a_{10}, a_{11}, a_{12}, a_{13}}	T5: {a_{14}, a_{15}, a_{16}, a_{17}}	T6: {a_{18}, a_{10}, a_{1a}, a_{1b}}	T7: {a_{1c}, a_{1d}, a_{1e}, a_{1f}}
…				
7	T28:{a_{70}, a_{71}, a_{72}, a_{73}}	T29:{a_{74}, a_{75}, a_{76}, a_{77}}	T30:{a_{78}, a_{79}, a_{7a}, a_{7b}}	T31:{a_{7c}, a_{7d}, a_{7e}, a_{7f}}

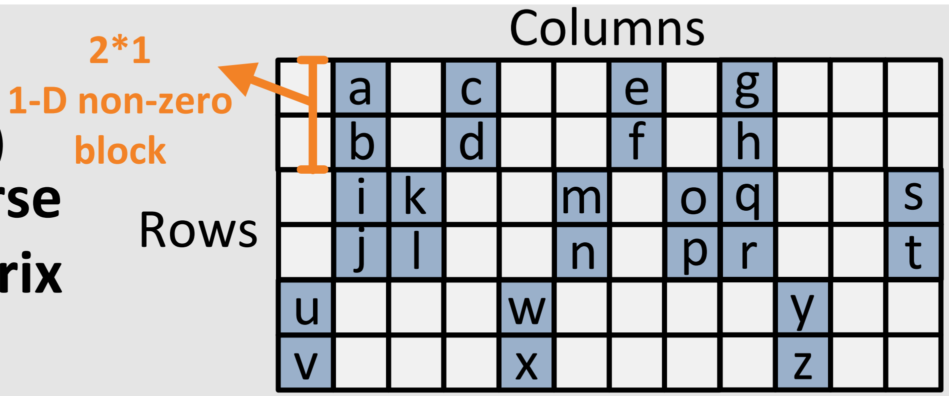
c_{xx} is int32

C (row-major)

T0: {c_{00}, c_{01}}	T1: {c_{02}, c_{03}}	T2: {c_{04}, c_{05}}	T3: {c_{06}, c_{07}}
T4: {c_{10}, c_{11}}	T5: {c_{12}, c_{13}}	T6: {c_{14}, c_{15}}	T7: {c_{16}, c_{17}}
T28: {c_{70}, c_{71}}	T29: {c_{72}, c_{73}}	T30: {c_{74}, c_{75}}	T31: {c_{76}, c_{77}}

SR-BCRS sparse matrix format

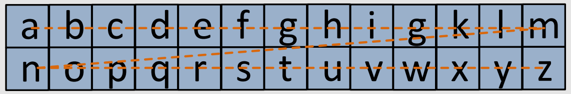
(a) **Sparse Matrix**



Sparse matrix with **1-D block non-zeros,**
the **length of the 1-D block = 2, 4, or 8**

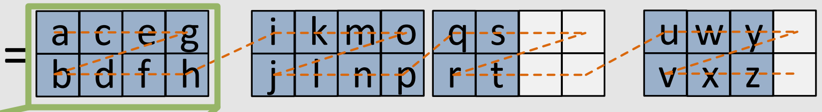
(b) **BCRS** format

Row pointers = [0, 4, 10, 13]
Column indices =
[1, 3, 6, 8, 1, 2, 5, 7, 8, 11, 0, 4, 9]
Values =



(c) **SR-BCRS** format (stride=4)

Row pointers = [0, 4, 4, 10, 12, 15]
Column indices =
[1, 3, 6, 8, 1, 2, 5, 7, 8, 11, *, *, 0, 4, 9, *]
Values =



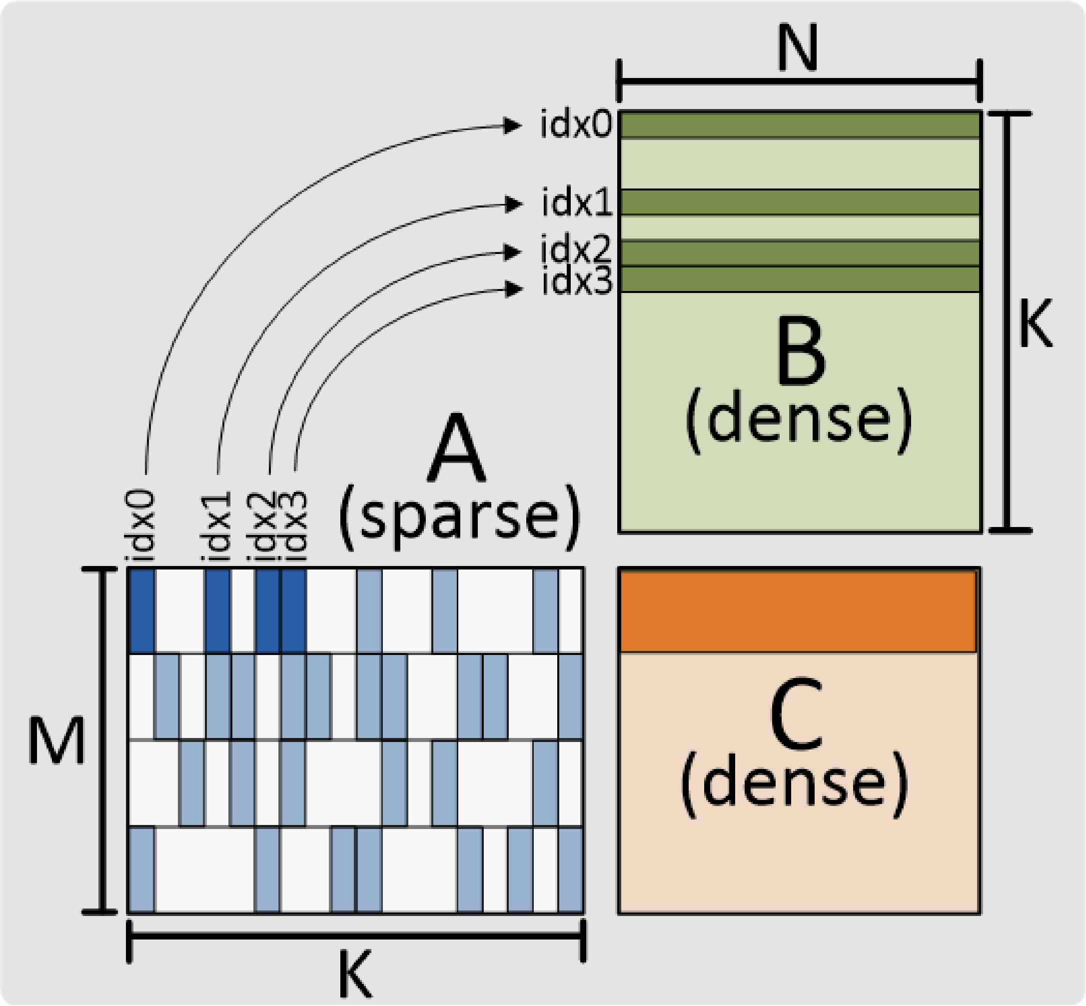
SR-BCRS (ours) is more friendly to Tensor Cores

Matrix A for *mma*

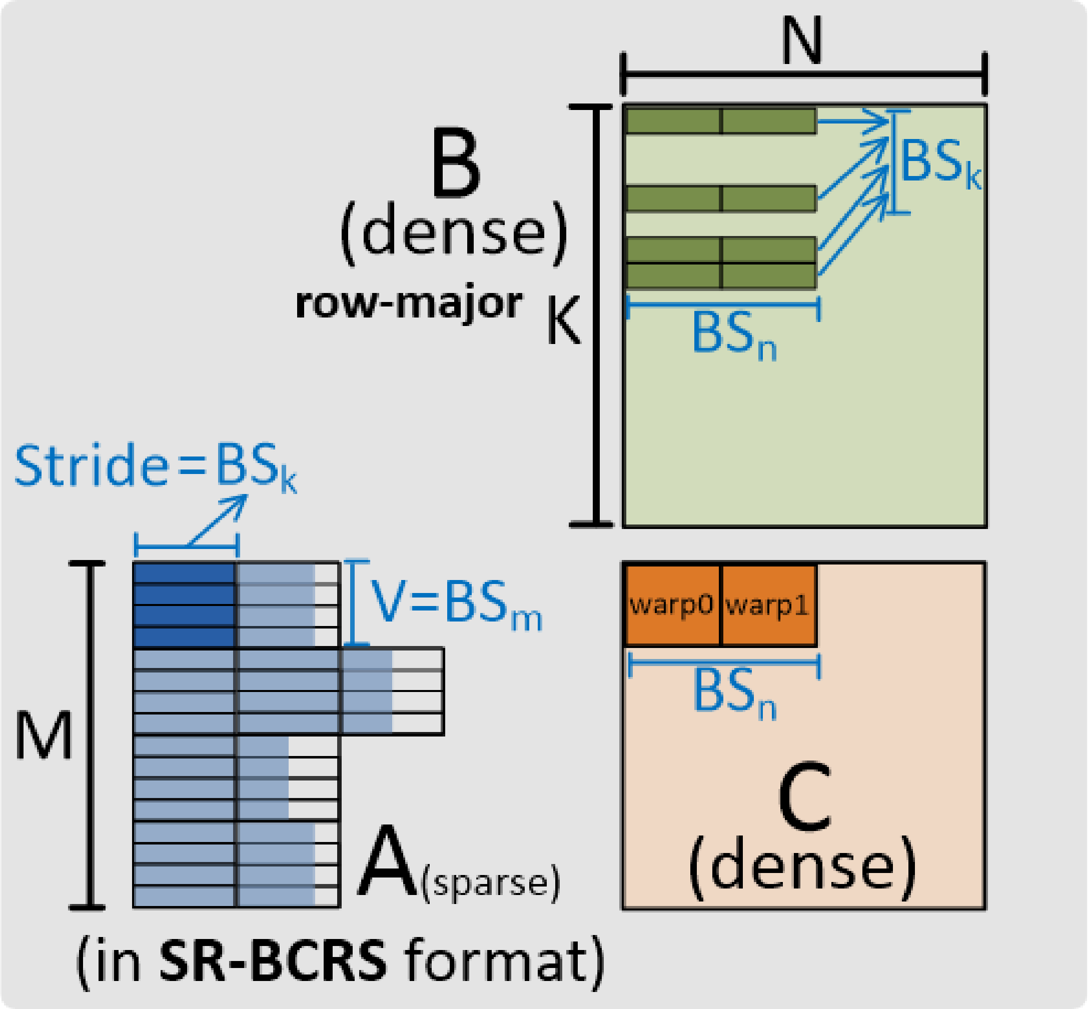
A (row-major)

Row \ Col	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
0	T0: {a_{00}, a_{01}, a_{02}, a_{03}}	T1: {a_{04}, a_{05}, a_{06}, a_{07}}	T2: {a_{08}, a_{09}, a_{0a}, a_{0b}}	T3: {a_{0c}, a_{0d}, a_{0e}, a_{0f}}
1	T4: {a_{10}, a_{11}, a_{12}, a_{13}}	T5: {a_{14}, a_{15}, a_{16}, a_{17}}	T6: {a_{18}, a_{10}, a_{1a}, a_{1b}}	T7: {a_{1c}, a_{1d}, a_{1e}, a_{1f}}
…				
7	T28:{a_{70}, a_{71}, a_{72}, a_{73}}	T29:{a_{74}, a_{75}, a_{76}, a_{77}}	T30:{a_{78}, a_{79}, a_{7a}, a_{7b}}	T31:{a_{7c}, a_{7d}, a_{7e}, a_{7f}}

SpMM in Magicube

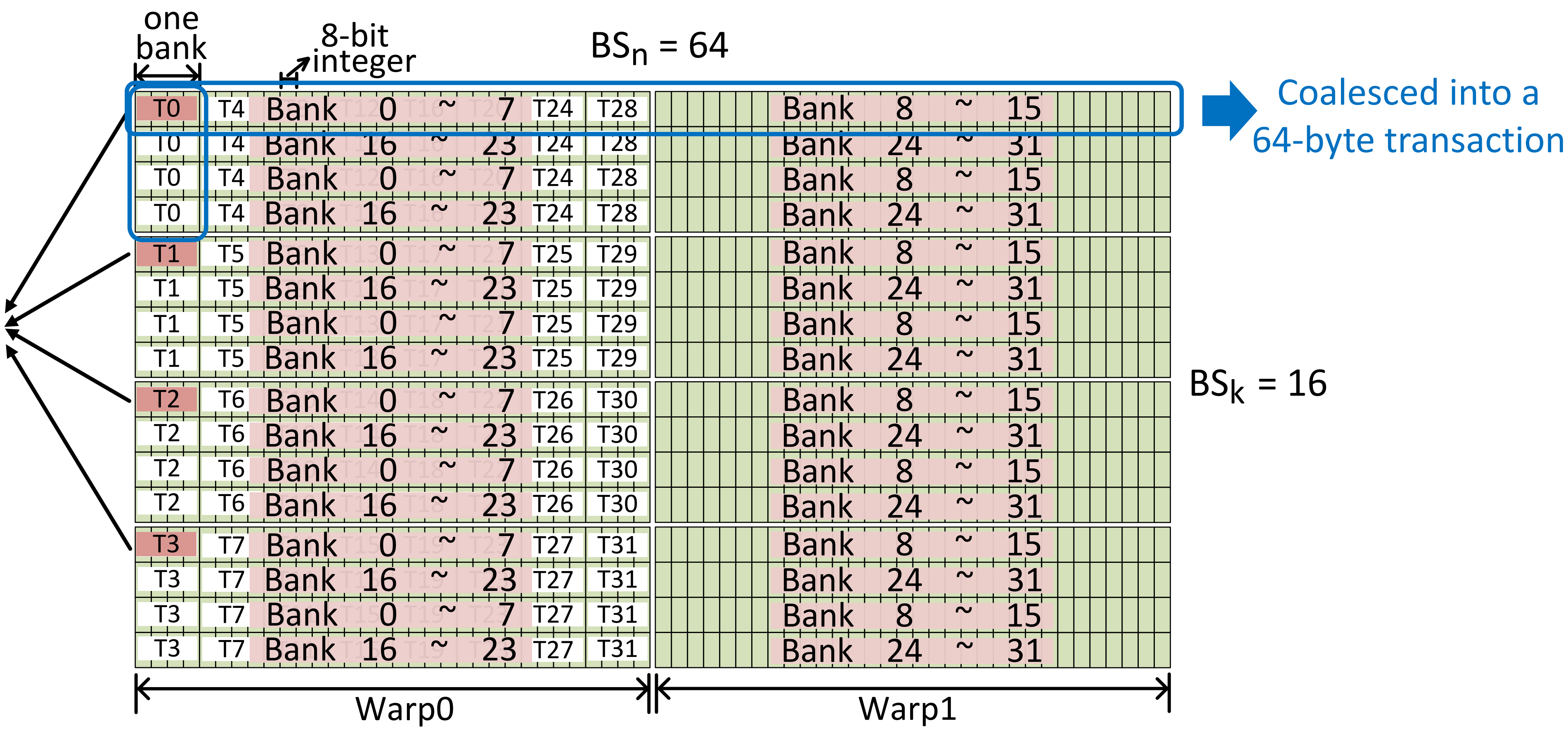


(a) SpMM

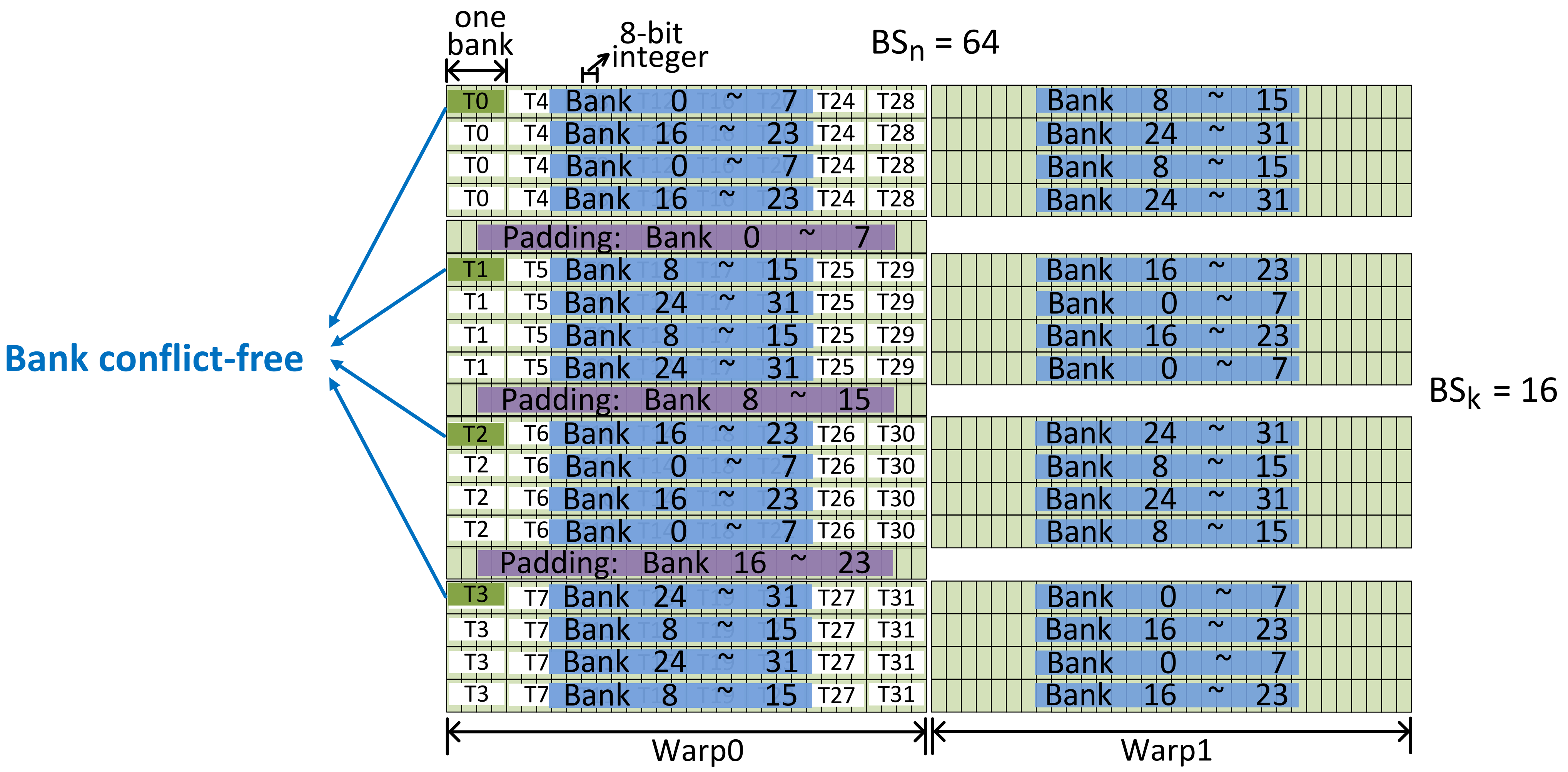


(b) SpMM in Magicube at thread-block level

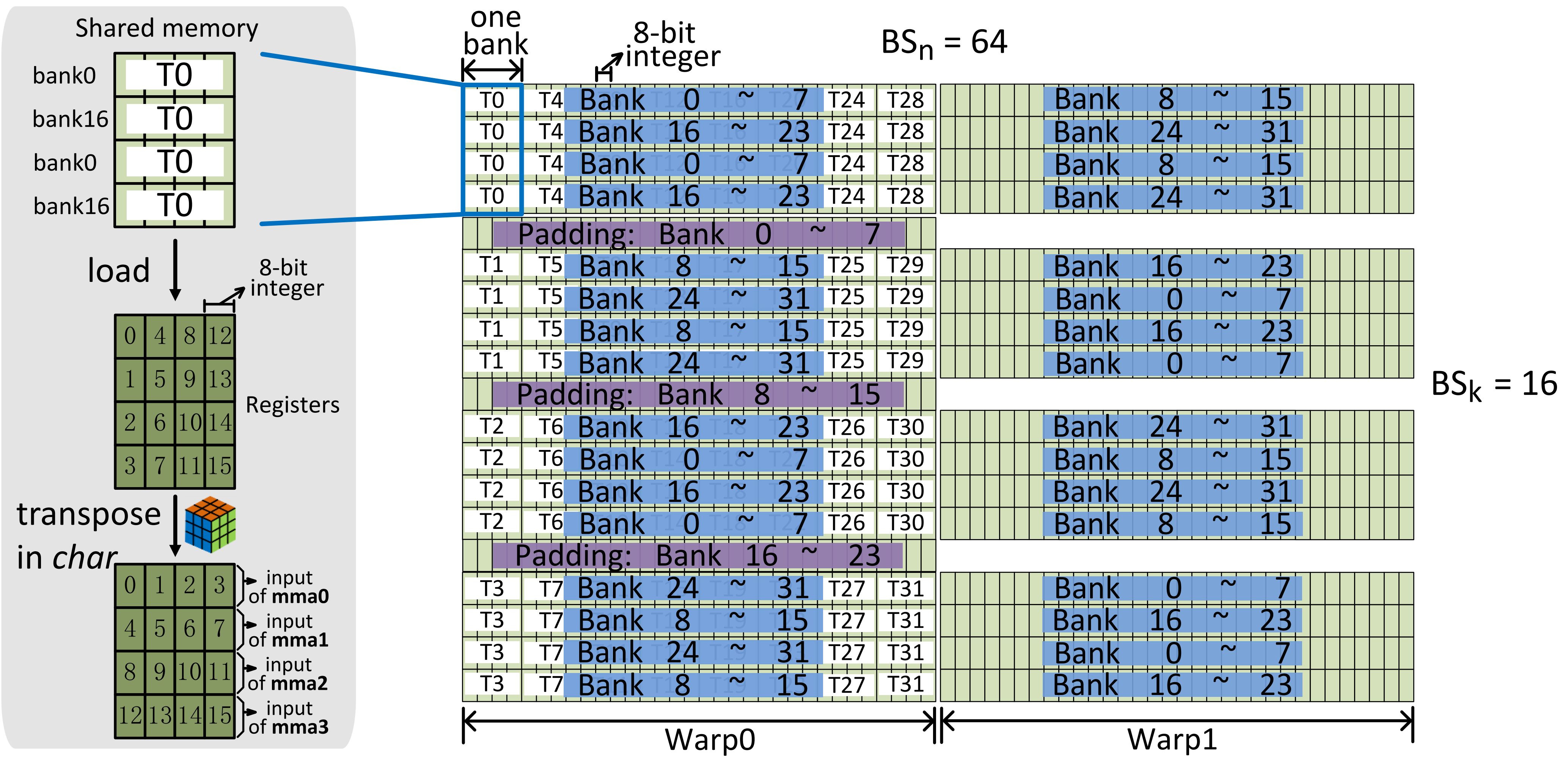
Load rows of matrix B to shared memory for *int8*



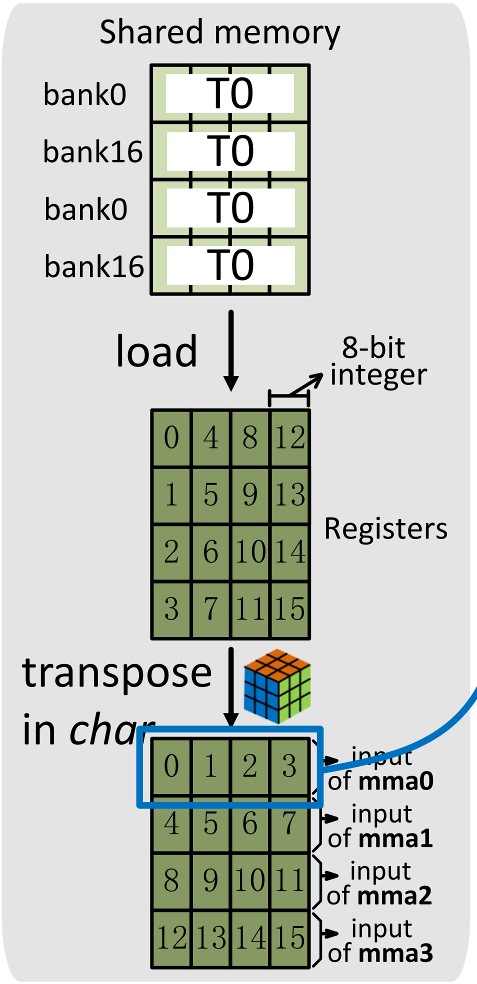
Load blocks of matrix B to shared memory for *int8*

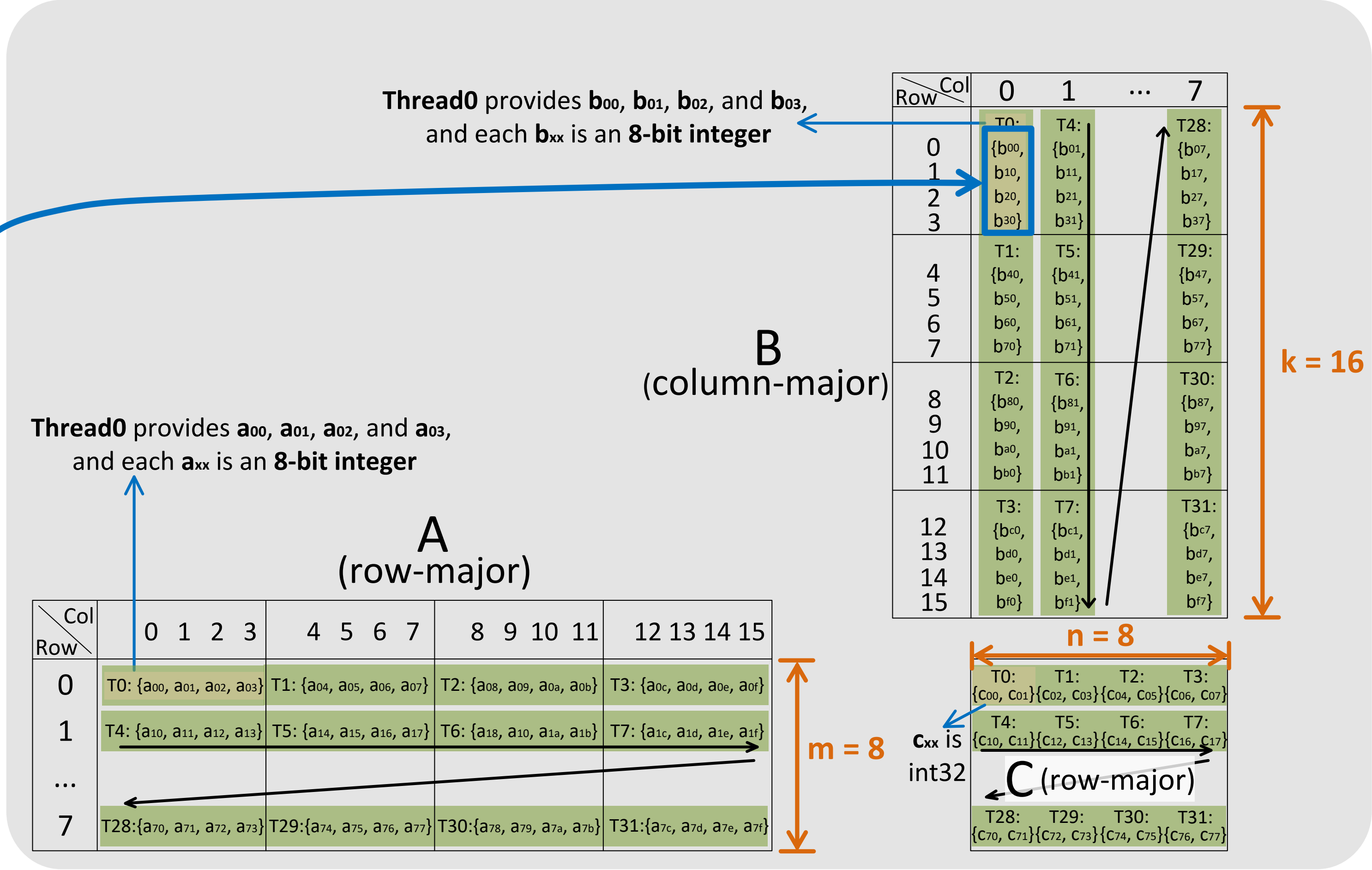


Local transpose on registers for *int8*



Local transpose on registers for *int8*





Data layout of *m8n8k16* for *int8 mma*

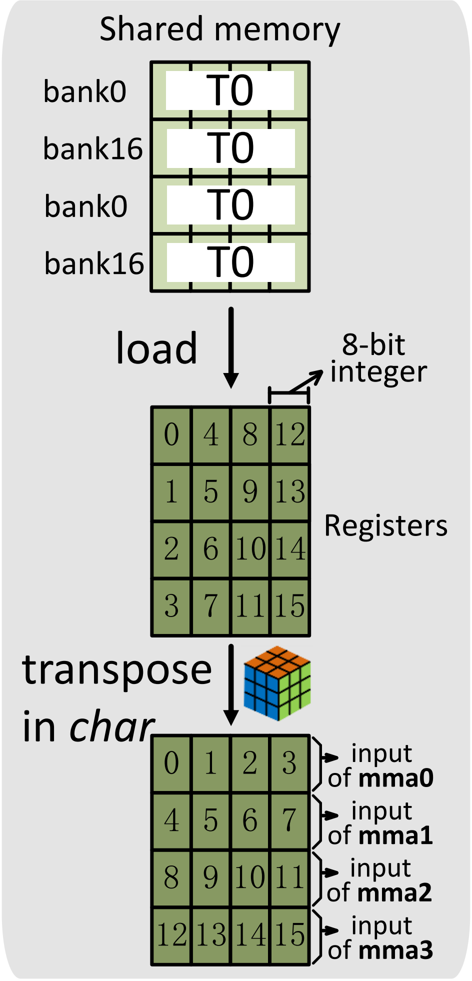
Efficiency is guaranteed by:

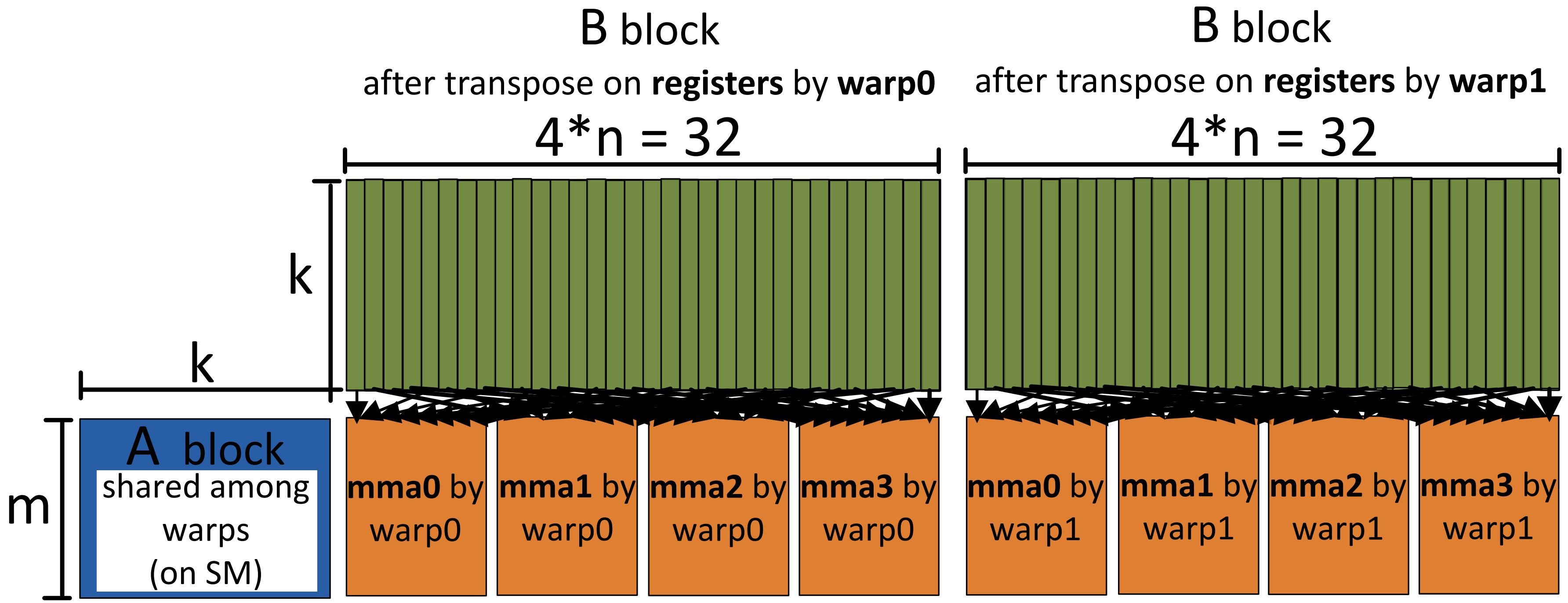
(1) coalesced global memory access

(2) conflict-free shared memory access

(3) fast transpose on registers

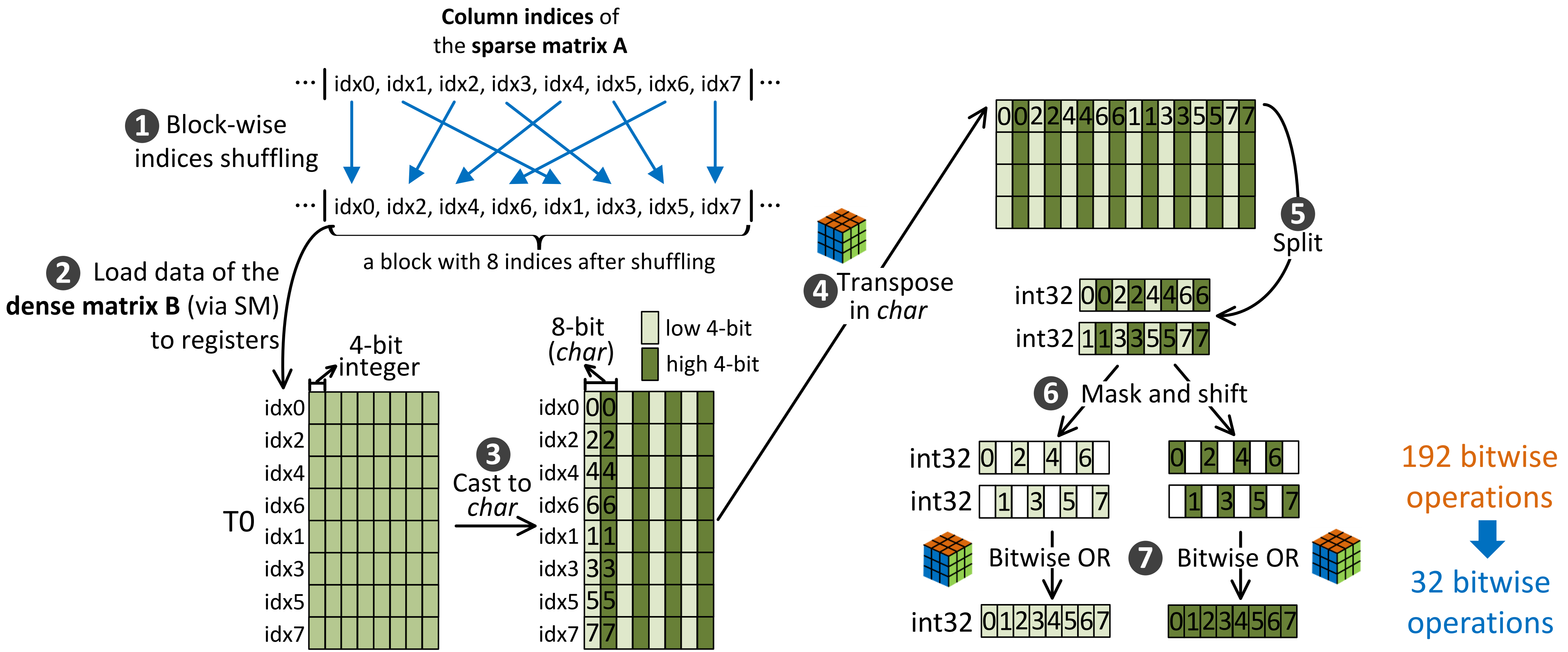
MMAs in SpMM with *int8*



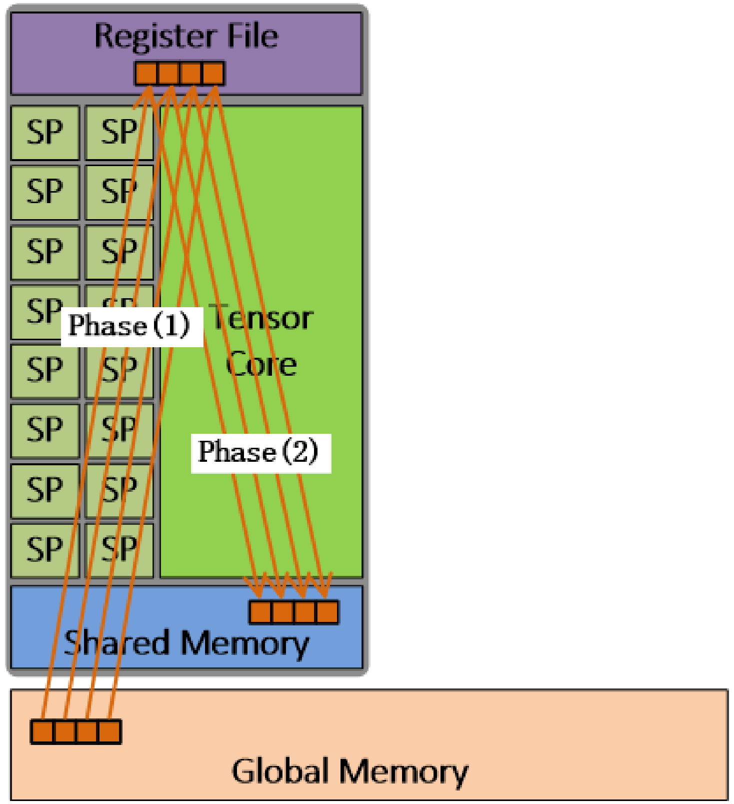


The warp-level view of *MMAs* in SpMM with *int8*

Efficient local transpose for *int4* with indices shuffling



Prefetch data blocks of matrix B of SpMM



Load data from GM to SM

Algorithm 1 Prefetch the data block of dense matrix B

$steps = nnz\ /\ BS_k$;

Load_A_values_and_indices_to_shared(0);
__syncthreads();
Prefetch_B_values_to_registers(0);

→ Cold start

for i=1; i < steps; i++ **do**

Store_B_values_on_regs_to_shared(i-1);
Load_A_values_and_indices_to_shared(i);
__syncthreads();

→ Load data and indices to SM

Prefetch_B_values_to_registers(i);
MMA_compute_tiles(i-1);
__syncthreads();

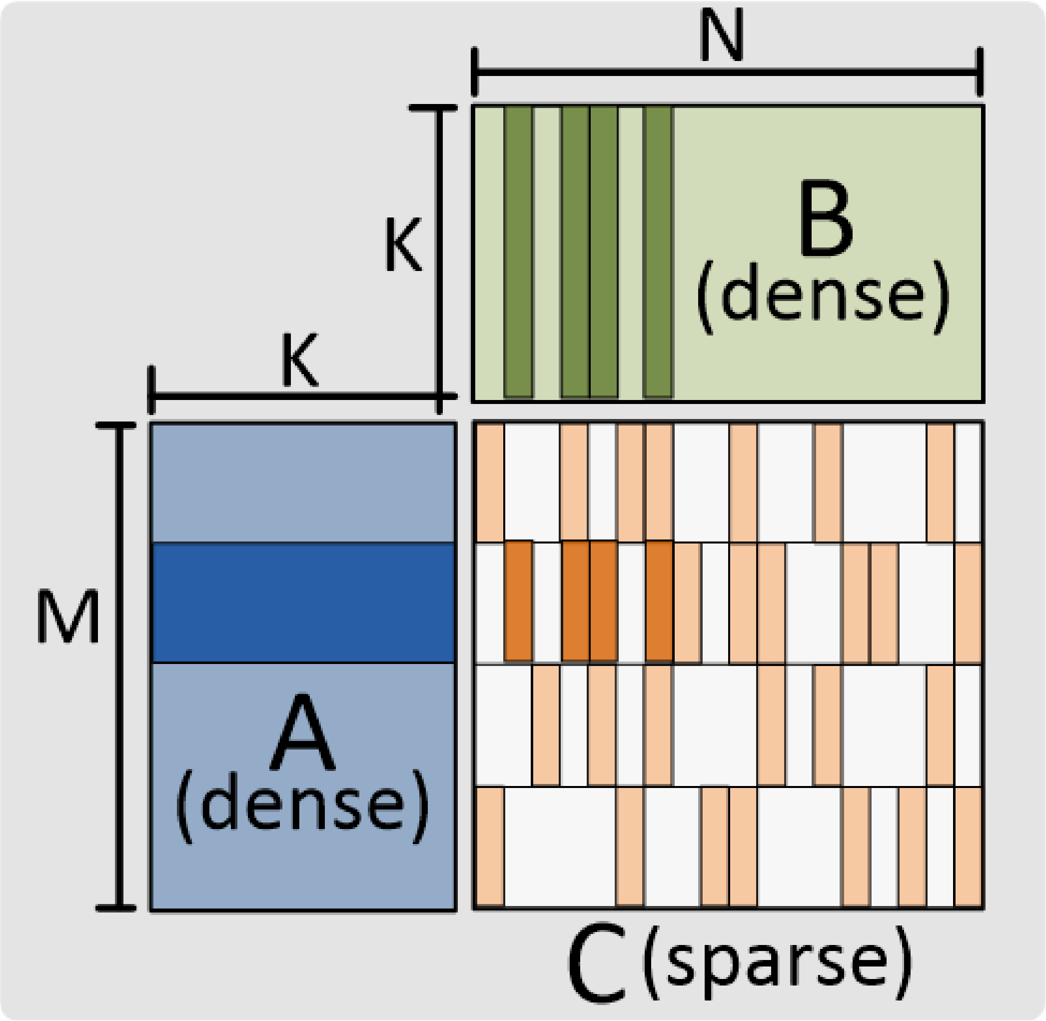
→ Overlap prefetch with *MMA*

end for

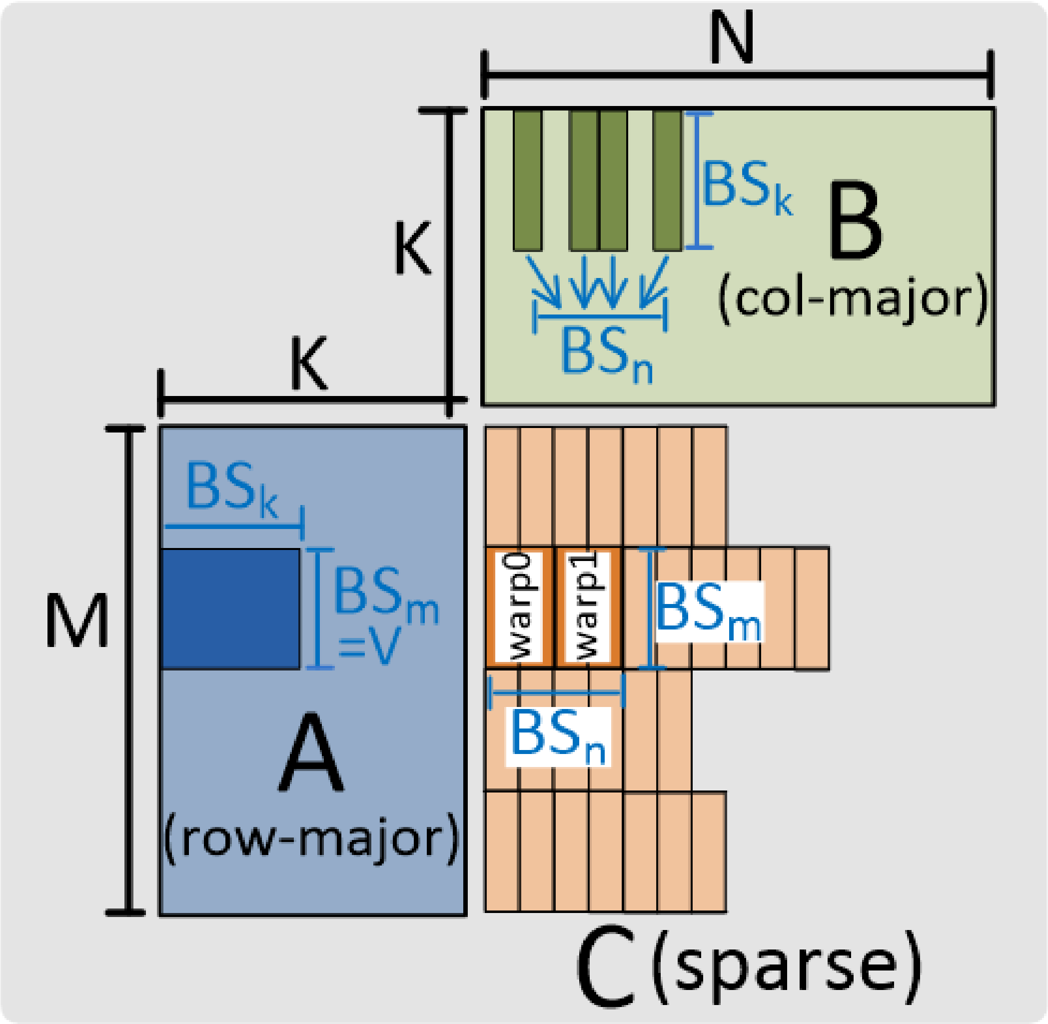
Store_B_values_on_regs_to_shared(i-1);
__syncthreads();
MMA_compute_tiles(i-1);

→ The tail of pipeline

SDDMM in Magicube

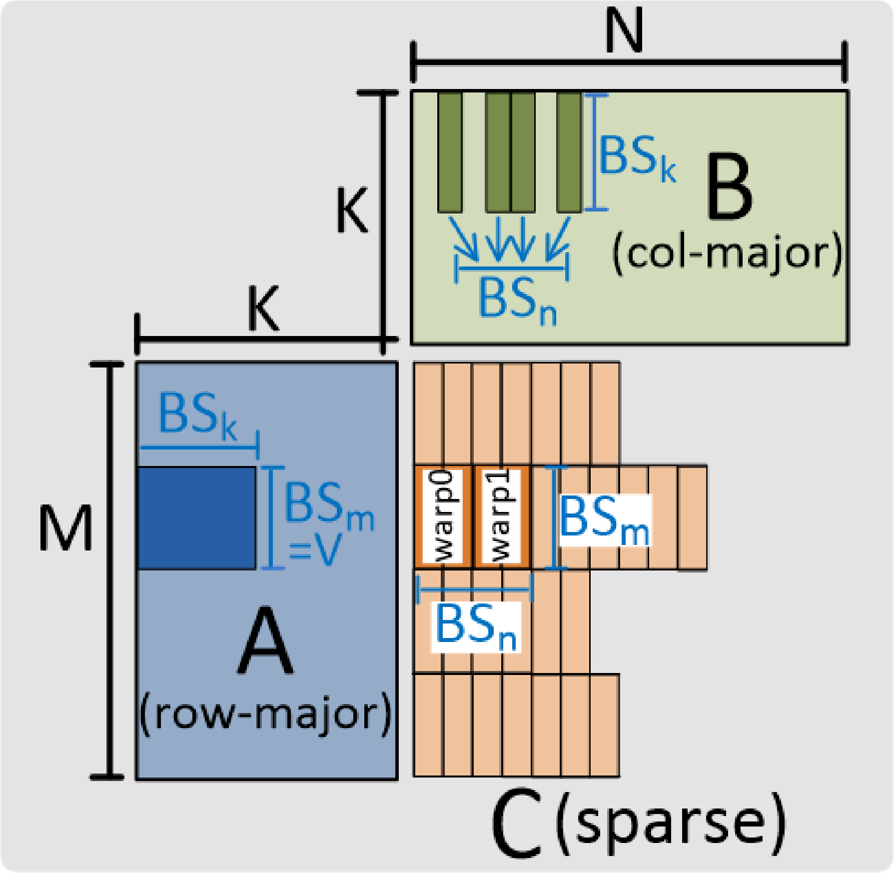


(a) SDDMM

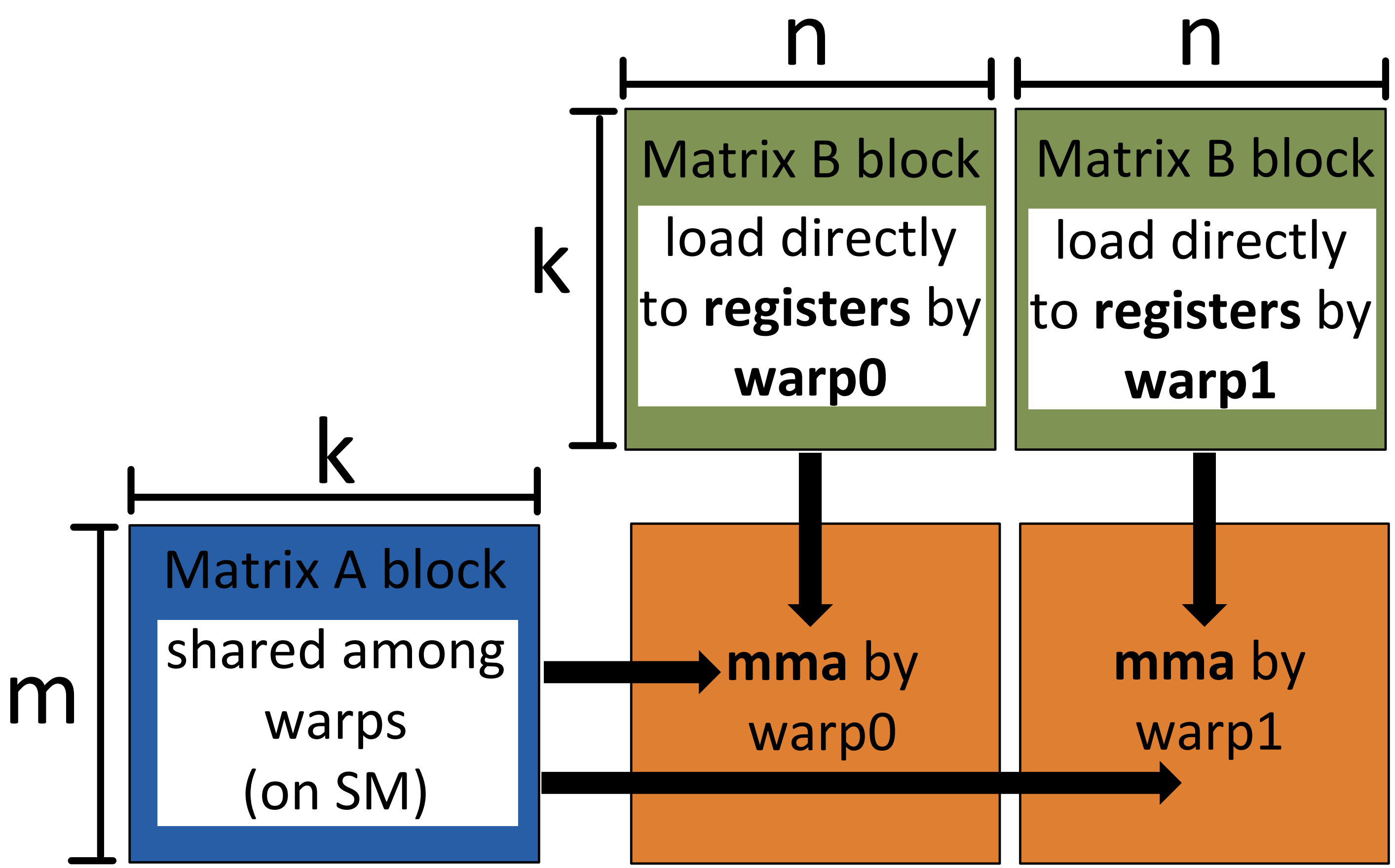


(b) SDDMM in Magicube at thread-block level

MMAs in SDDMM



The **thread-block level view** of SDDMM



The **warp-level view** of *MMAs* in SDDMM

Mixed precision

- a is an 8-bit **unsigned** integer, b is unsigned 4-bit

  a = 11101101 (237 in decimal)

  ↓Split

  $a_{7\sim4}$ 1110 unsigned — $a_{3\sim0}$ 1101 unsigned

  ↓Recover

  $a = 2^4 * a_{7\sim4} + a_{3\sim0}$

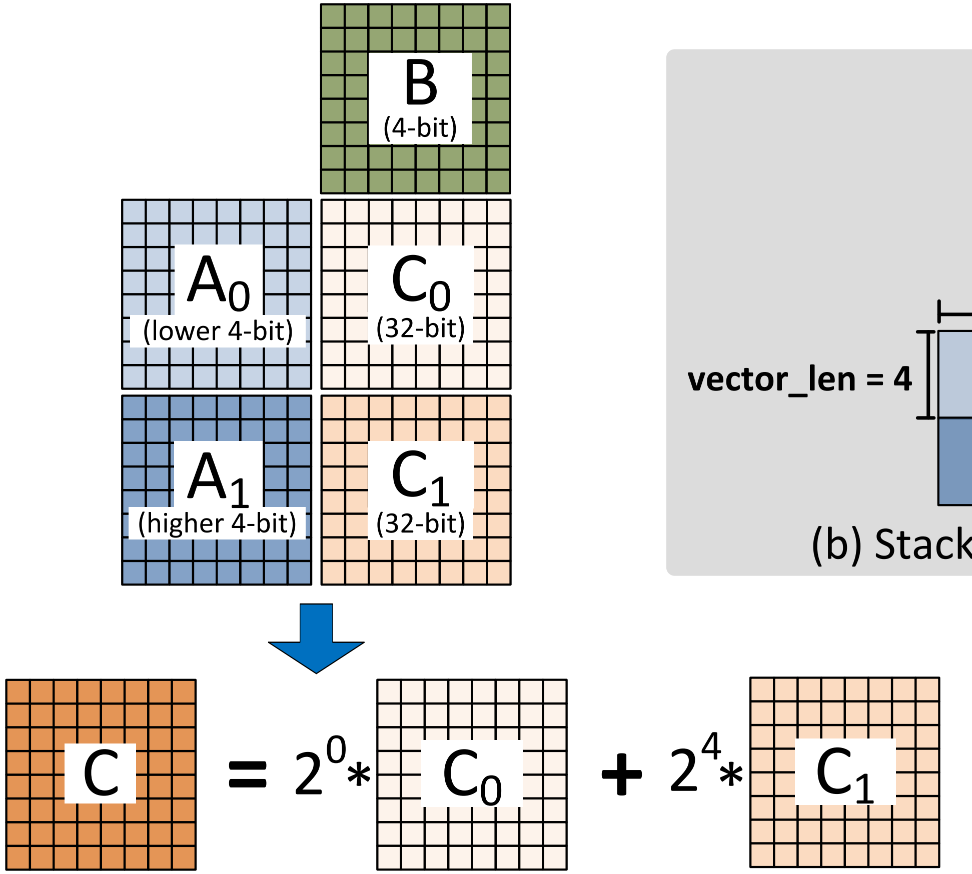
  $a*b = 2^4 * a_{7\sim4} * b + a_{3\sim0} * b$

- a is an 8-bit **signed** integer, b is signed 4-bit

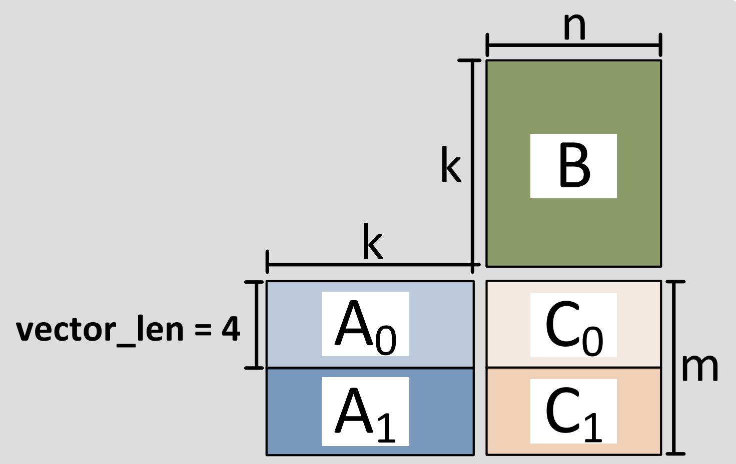
  a = 11101101 (-19 in decimal)

  ↓Split

  $a_{7\sim4}$ 1110 signed — $a_{3\sim0}$ 1101 unsigned



(a) Emulation of **A (8-bit)** * **B (4-bit)** using **4-bit** mma



(b) Stacked into a single ***mma***

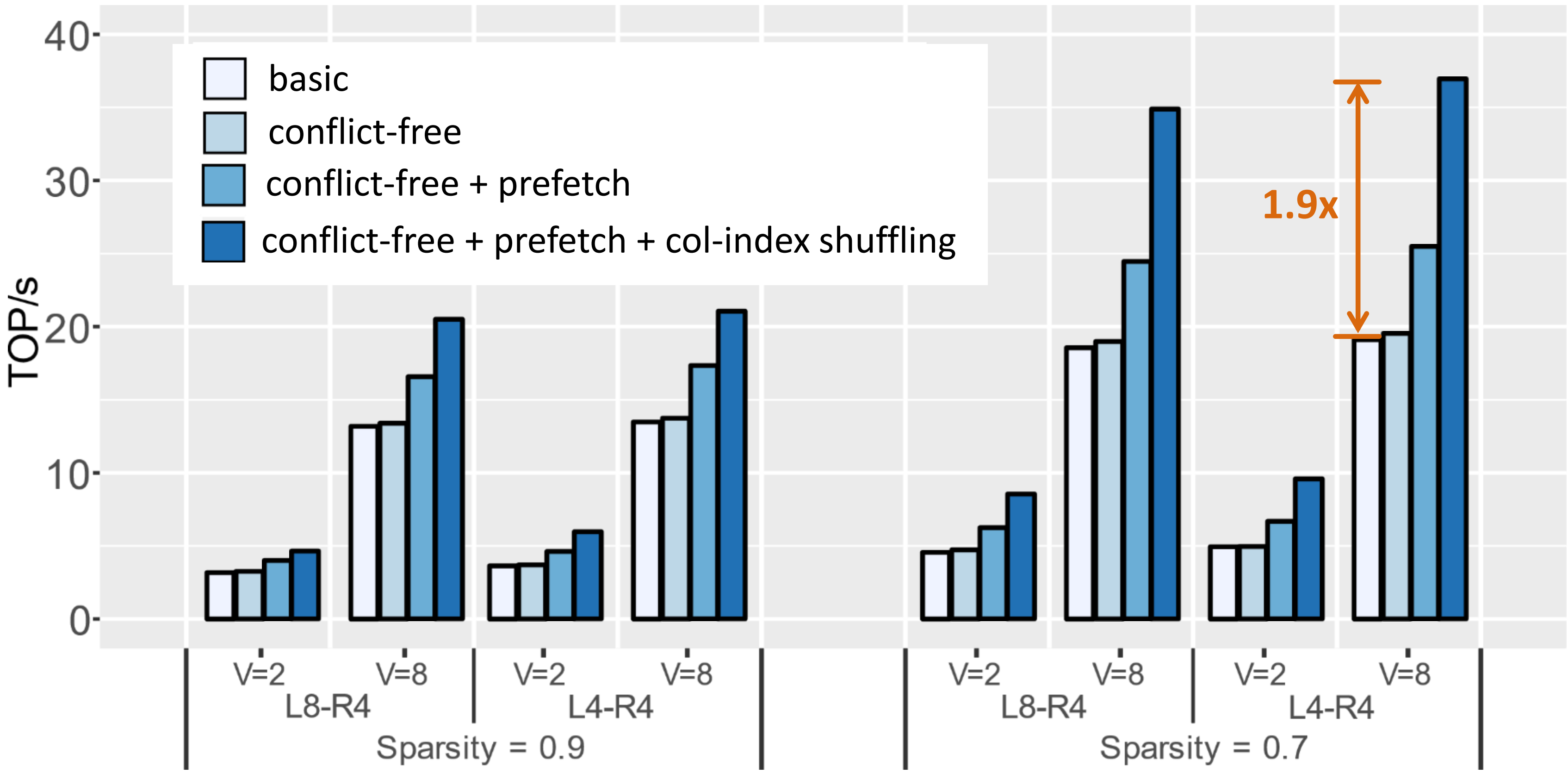
Evaluation

- NVIDIA A100-SXM4-40GB GPU
  - total 108 SMs
  - each SM has 192KB configurable L1 cache and shared memory, and 256KB registers
  - supported datatypes on Tensor Core: int8, int4, int1, fp16, bf16, tf32, fp64
- Compare the performance of **Magicube** with sparse libraries (**vectorSparse, cuSPARSE**) and dense libraries (**cuBLAS**, **cuDNN**)
- **Micro-benchmarks**: 1,536 sparse matrices from Deep Learning Matrix Collection (DLMC) with sparsity 50%~98%, dilating each scalar with 1-D blocks (length V = 2, 4, 8)
- **Case study**: end-to-end sparse Transformer inference

One streaming multiprocessor (SM) of GA100

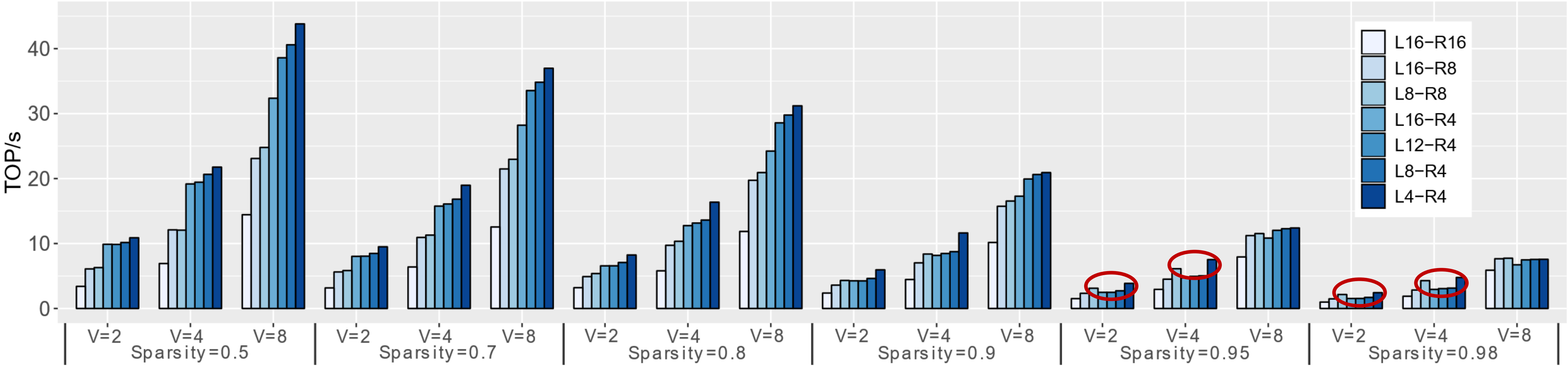
This image is from: R. Krashinsky, et al. *https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/* May, 2020

Ablation study for SpMM in Magicube



Ablation study for optimizations of SpMM

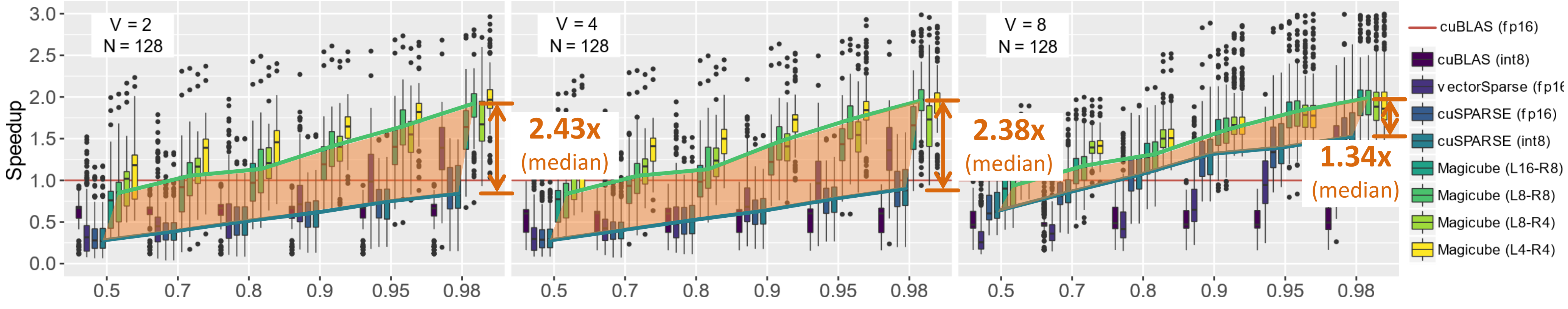
SpMM with mixed precision in Magicube



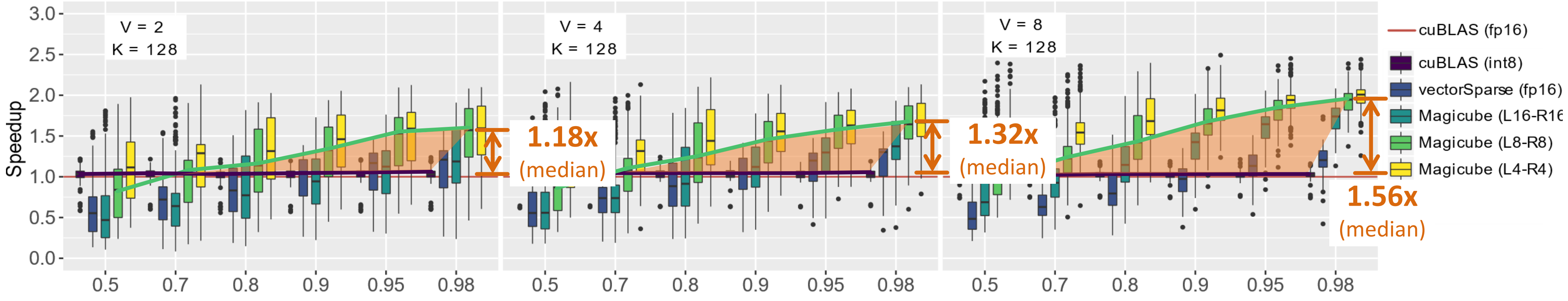
SpMM with mixed precision

Lx-Ry means **x-bit A matrix** multiplied by **y-bit B matrix**

Benchmarking SpMM and SDDMM



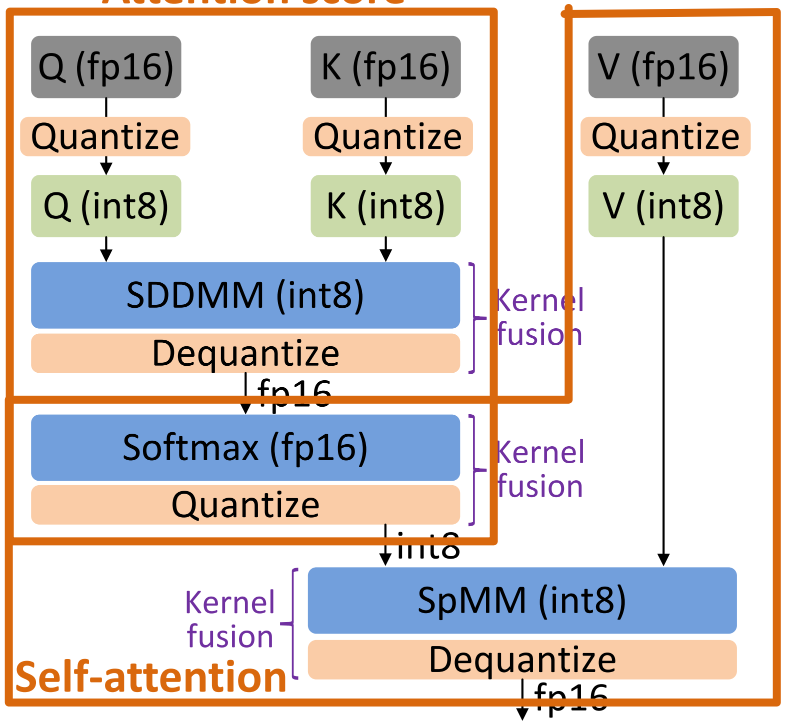
SpMM on A100 GPU using 1,536 sparse matrices



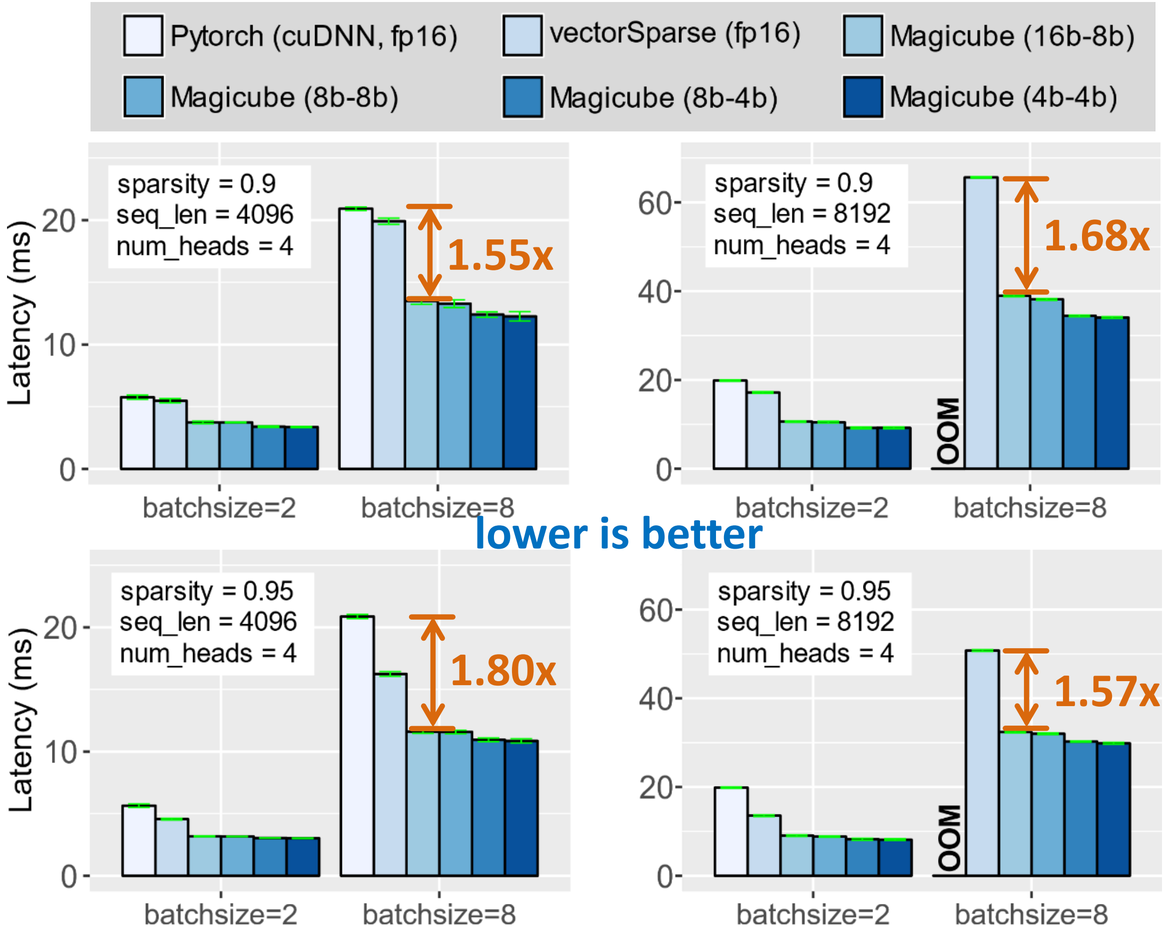
SDDMM on A100 GPU using 1,536 sparse matrices

End-to-end sparse Transformer inference

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T \odot M}{\sqrt{d_k}}\right) V$$



Quantized self-attention with sparse attention mask



Latency of end-to-end inference of sparse Transformer

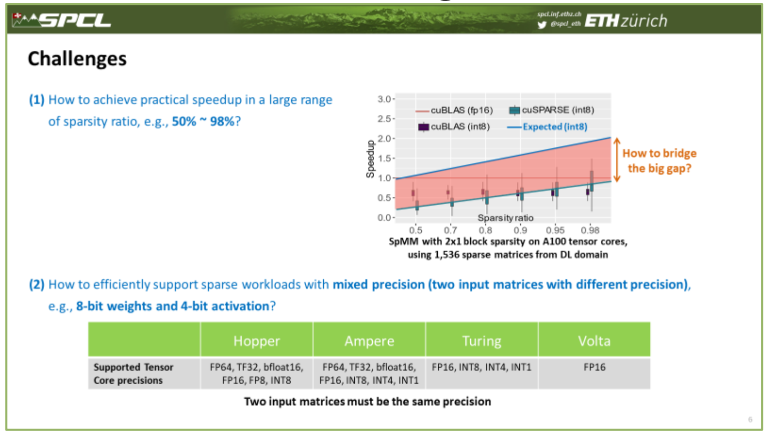
End-to-end sparse Transformer inference

dense		sparsity=0.9			
PyTorch (cuDNN, fp32)	PyTorch (cuDNN, fp16)	vectorSparse (fp16)	Magicube (16b-8b)	Magicube (8b-8b)	Magicube (8b-4b)
57.36%	**57.50%**	57.14%	**57.32%**	57.11%	56.79%

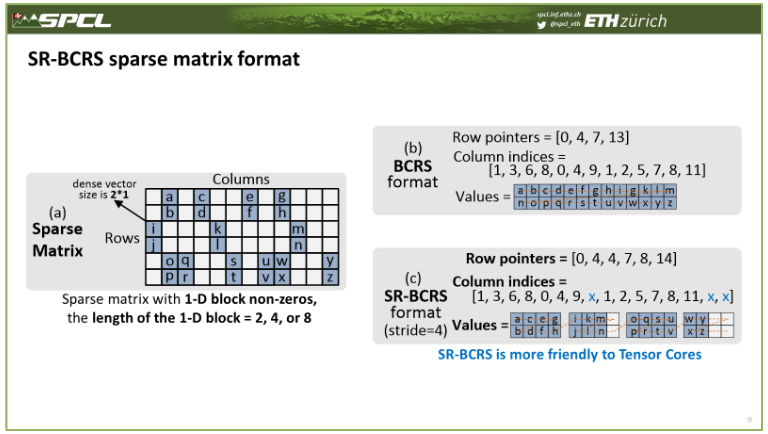
Test accuracy of text classification using sparse Transformer
with num_heads=4 and seq_len=4,096

Conclusion

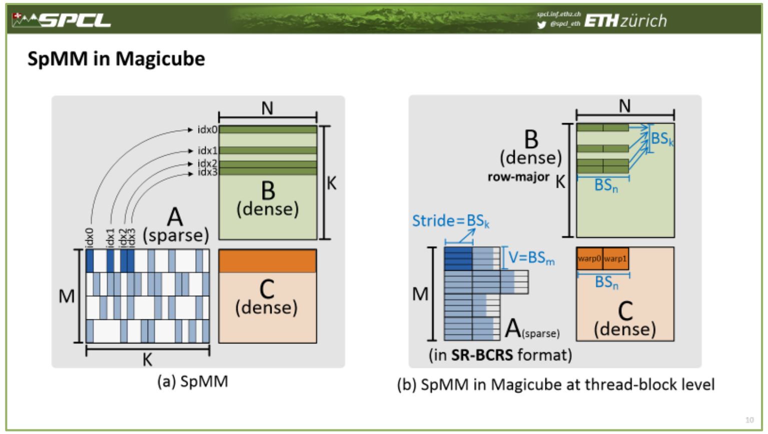
1. Challenges



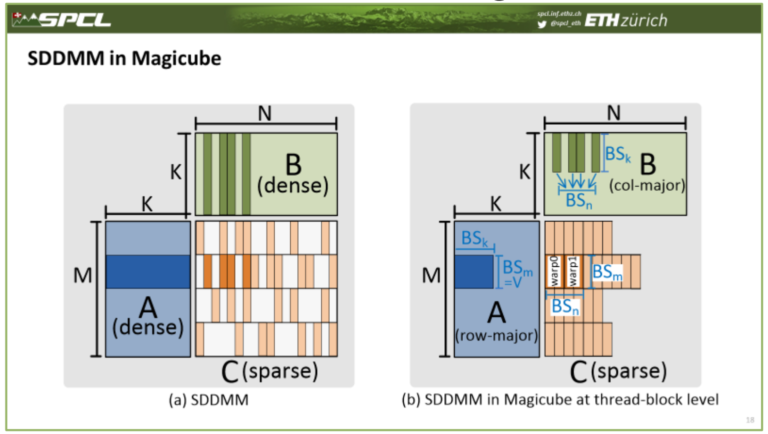
2. SR-BCRS format



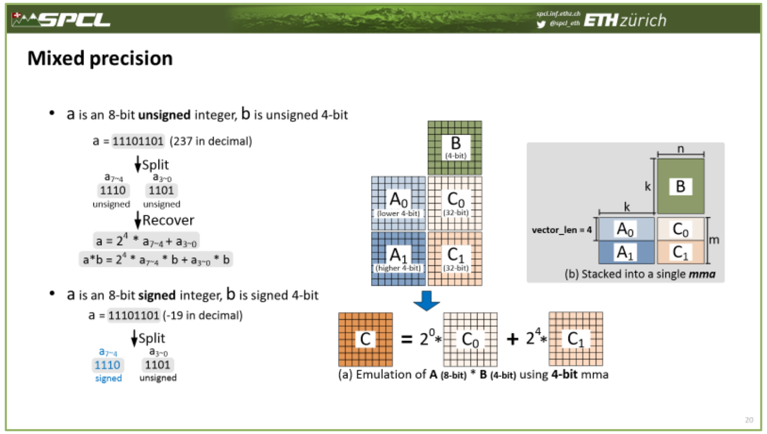
3. SpMM in Magicube



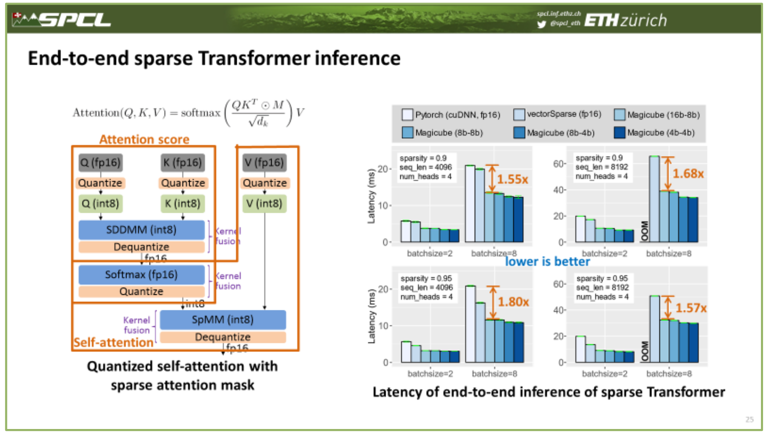
4. SDDMM in Magicube



5. Mixed precision



6. Evaluation



https://zenodo.org/record/6924338
https://github.com/Shigangli/Magicube