PIT: Optimization of Dynamic Sparse Deep Learning Models via Permutation Invariant Transformation

Ningxin Zheng, Huiqiang Jiang, Quanlu Zhang, Zhenhua Han, Lingxiao Ma, Yuqing Yang, Fan Yang, Chengruidong Zhang, Lili Qiu, Mao Yang, Lidong Zhou

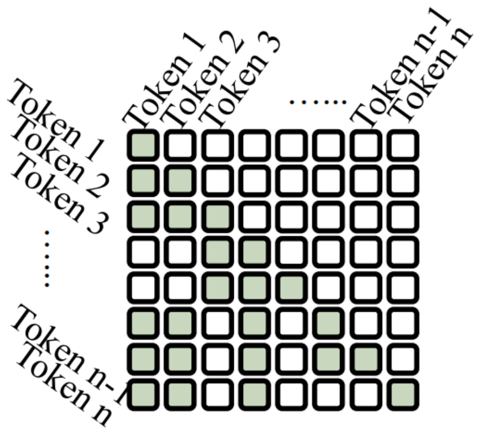
Microsoft Research

Microsoft

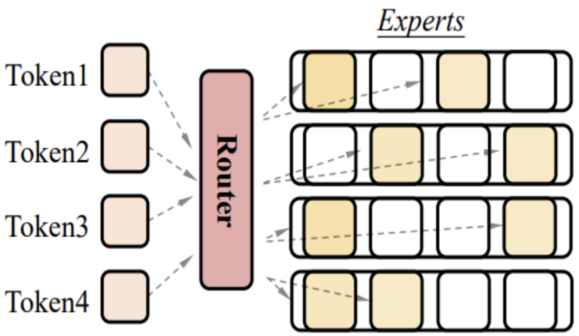
https://github.com/microsoft/SparTA/tree/pit_artifact

Dynamic Sparsity in Deep Learning Models

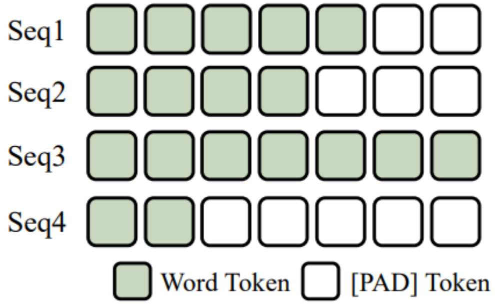
- Dynamic sparsity commonly exists in modern deep learning models (e.g., LLM), which spans in both
  - Weight tensors (pruned models) and activation tensors (sparse attention)
  - Input data (varying seq. length) and model architectures (MoE)
  - Training and inference



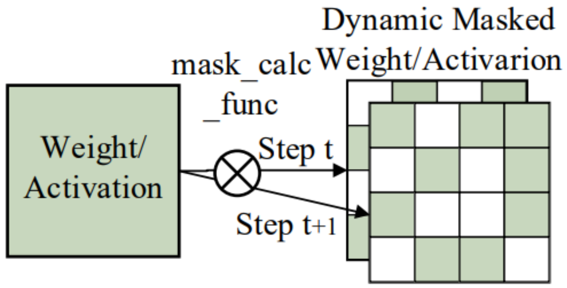
Dynamic Sparse Attention



Mixture-of-Experts (MoE)

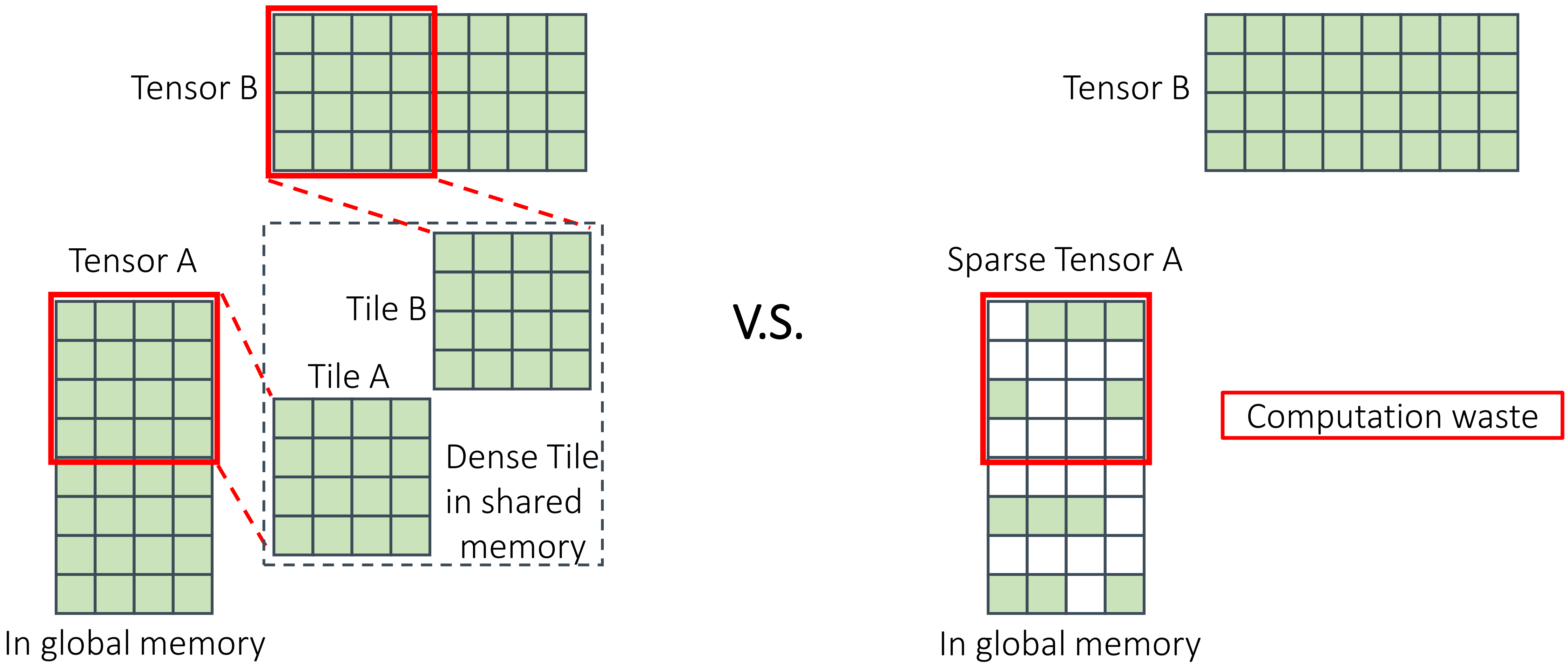


Dynamic sequence length



Sparse Training

Dynamic Sparsity Hardly Aligned to Accelerators

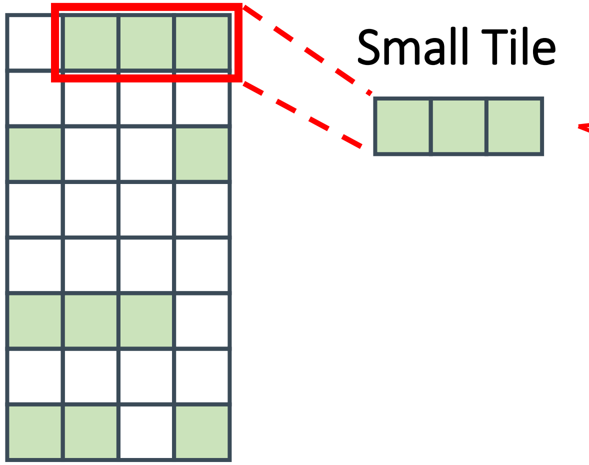


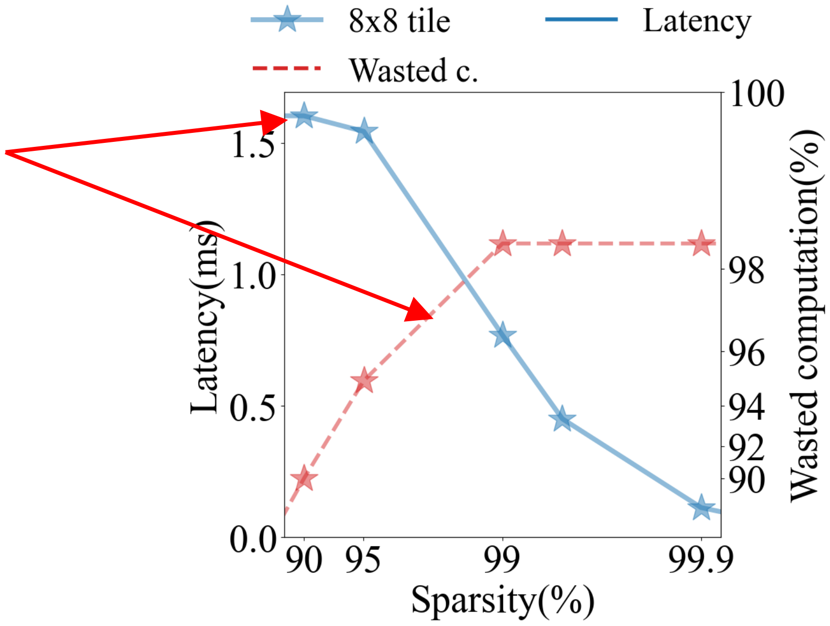
Dense Matrix Multiplication C=A·B

Sparse Matrix Multiplication C=A·B

Dilemma of Tile Covering

Sparse Tensor A

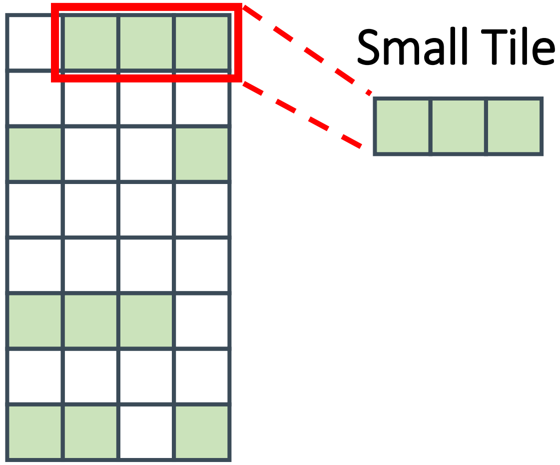


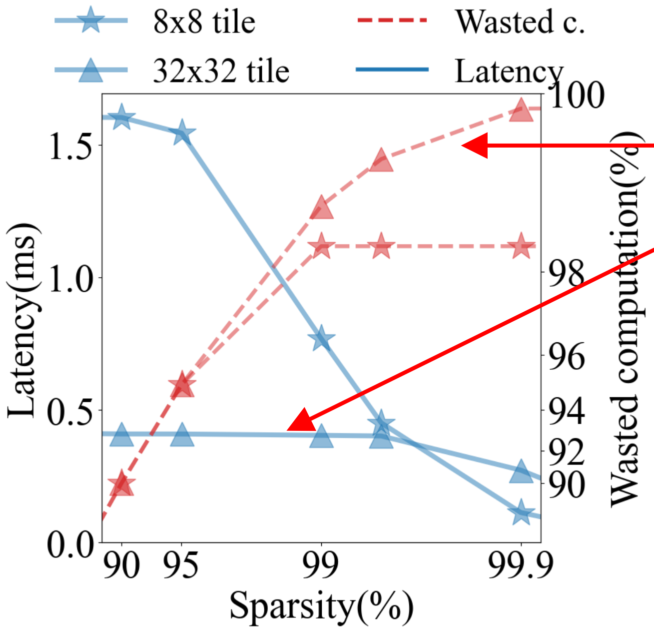


√ Low waste
× Low SM Utilization

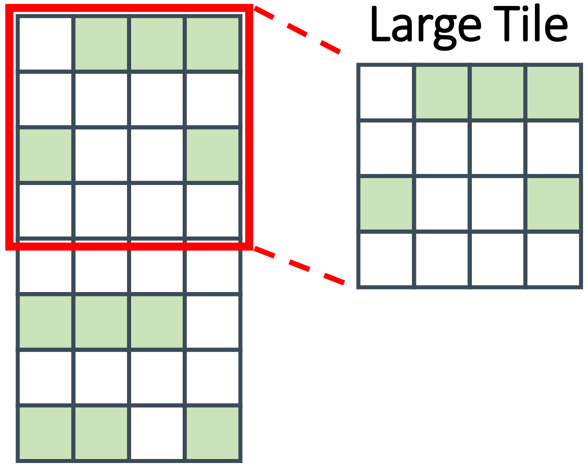
Dilemma of Tile Covering

Sparse Tensor A





Sparse Tensor A

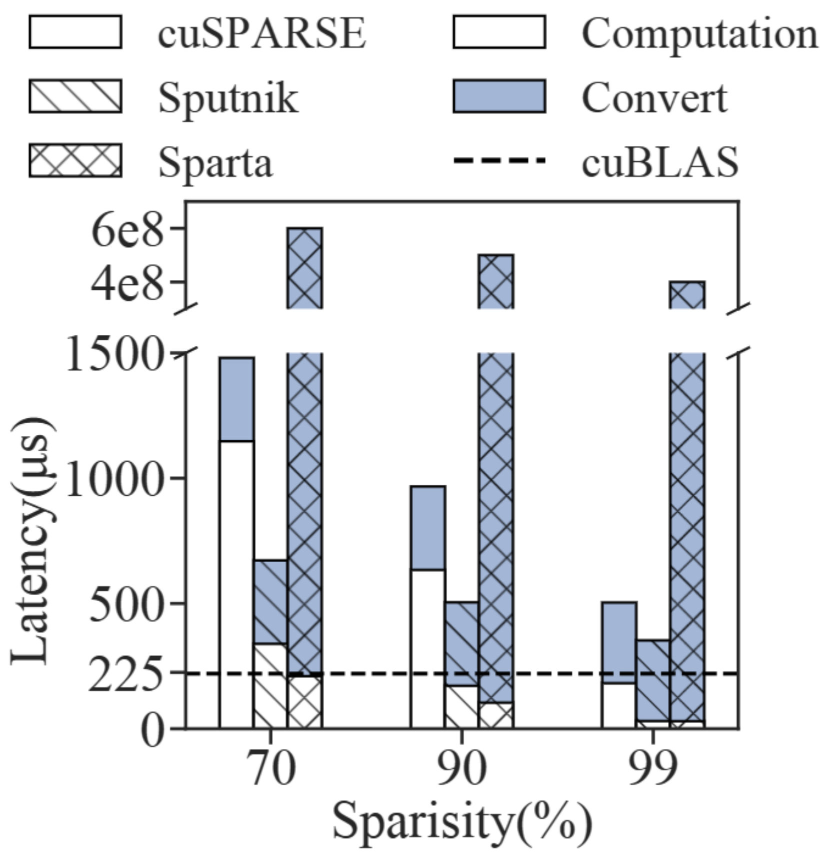


√ Low waste
× Low SM Utilization

√ High SM Utilization
× High waste

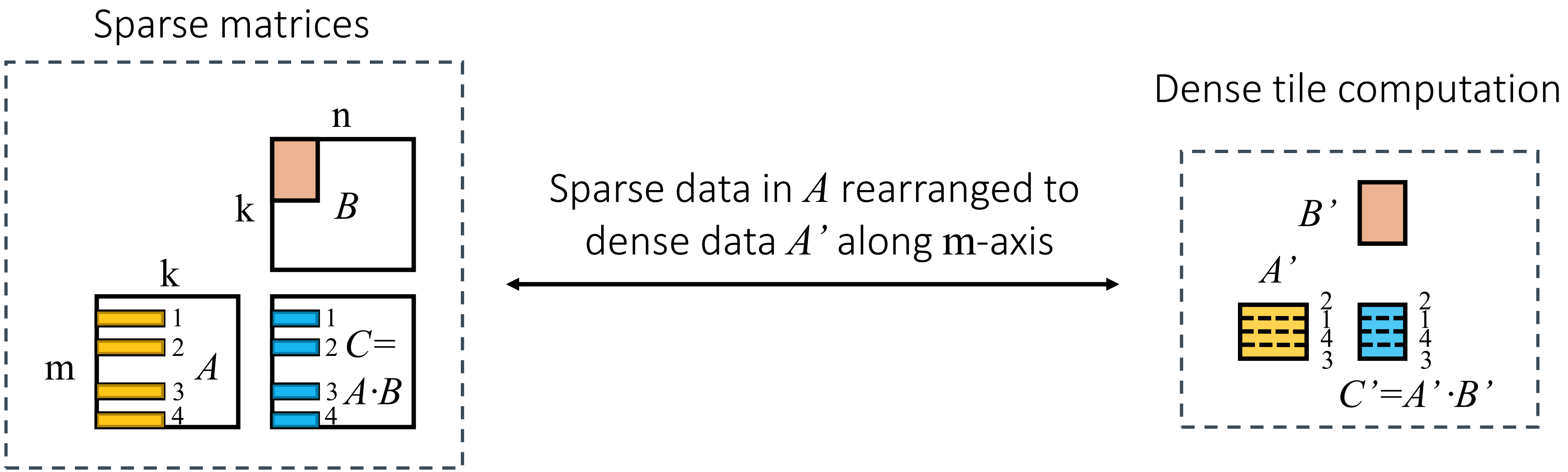
Sparsity-Aware Kernels?

- Build fine-grained index (e.g., CSR) to skip computation
- Significant overhead during index construction and data access
- Worse than the dense counterpart



- We want to achieve:
    - Use computation-efficient large tiles,
    - With low computation waste,
    - With minimal data conversion and access overhead.

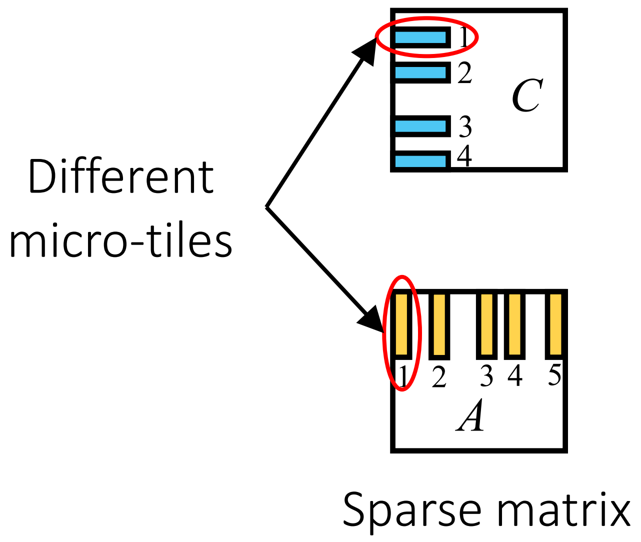
Opportunity: Sparse-to-Dense Transformation

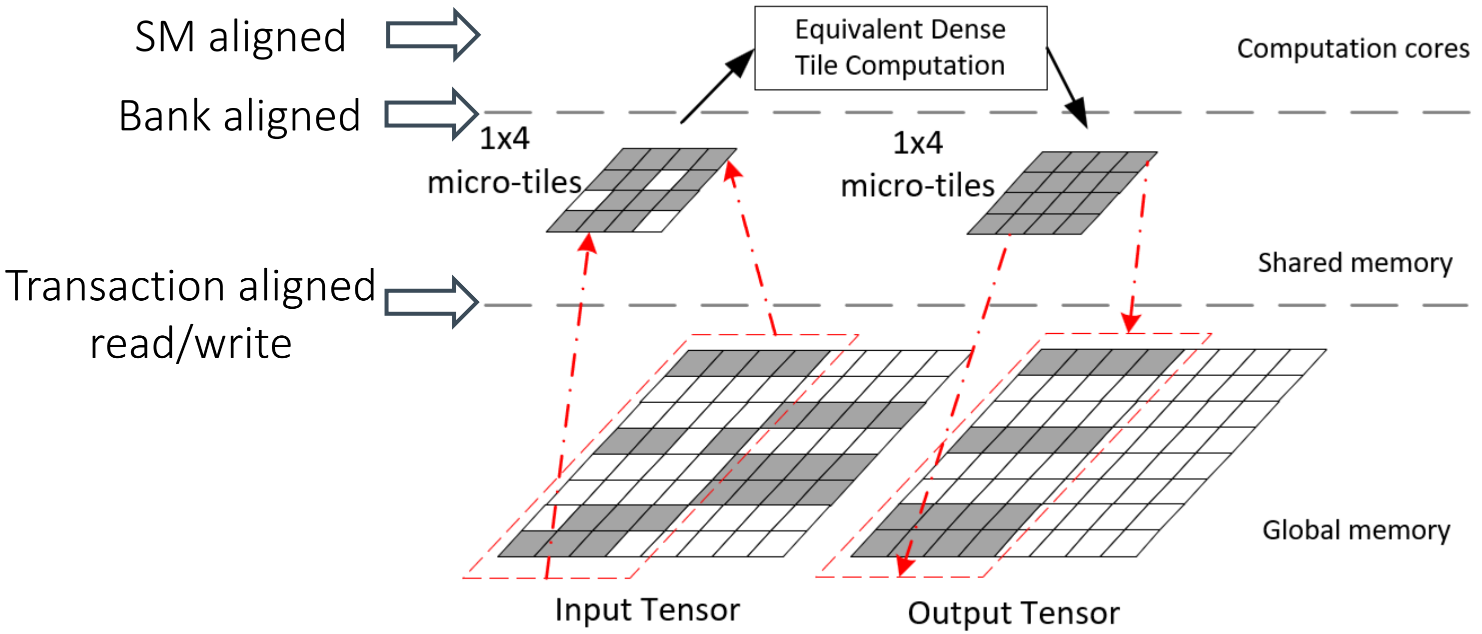


- Data rearrangement does not affect dense computation
- The rearrangement can be out-of-order

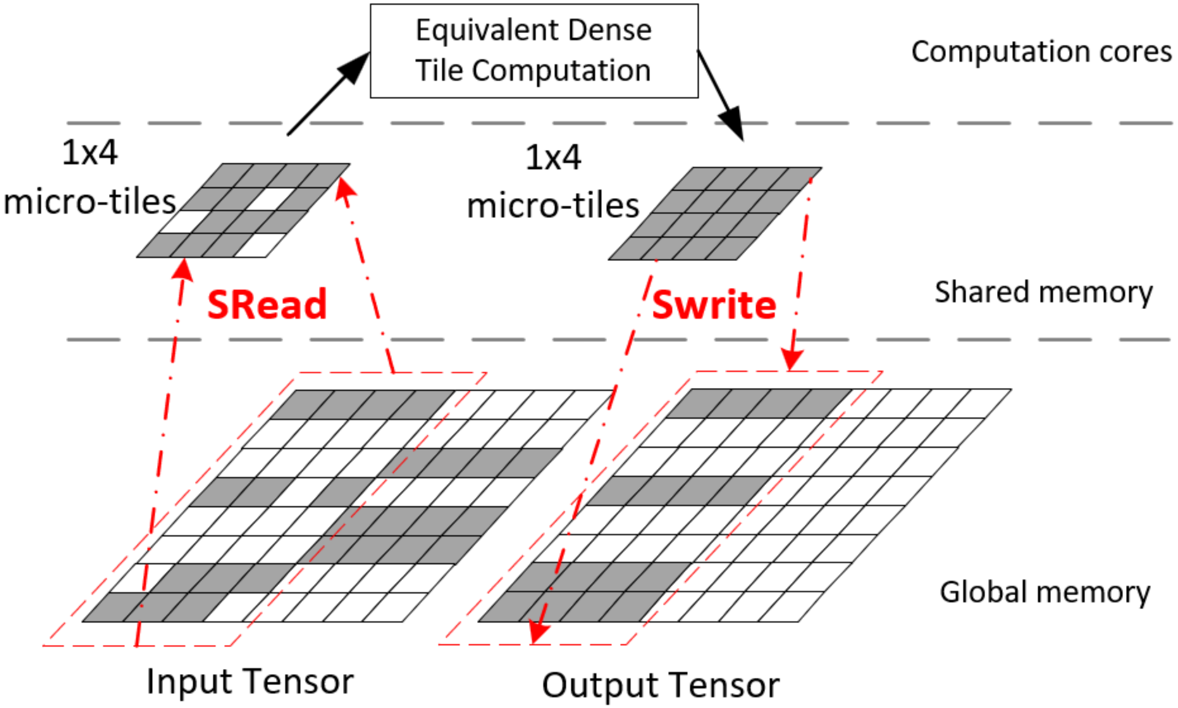
Micro-tile
-- The minimal granularity of rearrangement

- Micro-tile is a small data unit aligned with the hardware read/write granularity of an accelerator (e.g., GPU)
  - Read/write transaction is as small as 32 bytes in CUDA GPUs
  - Enable aligning to every level of an accelerator, e.g., global memory, shared memory, computation instructions





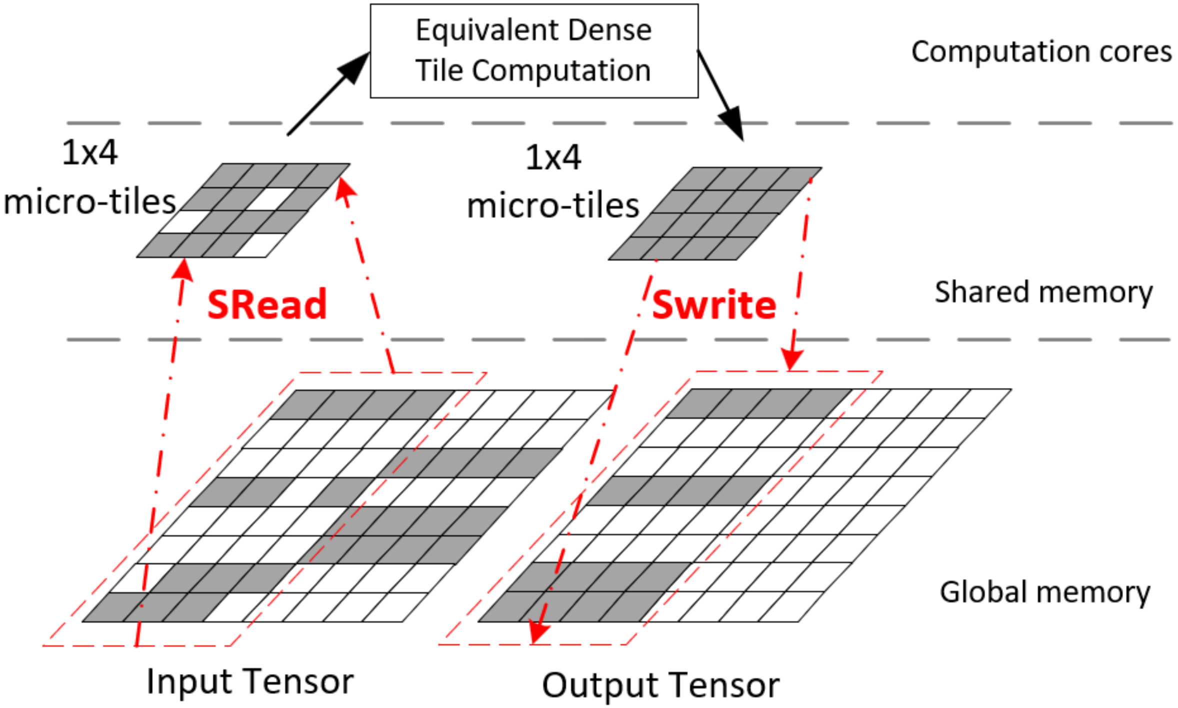
SRead and SWrite Primitives



- Rearrangement piggybacked during data movement across memory hierarchies
- Random access with zero cost due to aligned data granularity (i.e., micro-tile)

SRead and SWrite do **online rearrangement** of micro-tiles

SRead and SWrite Primitives



SRead and SWrite do **online rearrangement** of micro-tiles

```
/*Generated Sparse Kernel*/
__global__ void SparseKernelTemplate(
    struct Tensor Inputs, struct SparseIdx InIdx,
    struct Tensor Output, struct SparseIdx OutIdx,
){
    /* First allocate shared memory */
    InTiBlocks = AllocSharedM(TileInputFormats);
    OutTiBlock = AllocSharedM(TileOutputFormat);
    SRead(Inputs, InTiBlocks, InIdx);
    DenseTileImpl(InTiBlocks,OutTiBlock);
    SWrite(OutTiBlock, Output, OutIdx);
}
```

The sparse kernel template in PIT

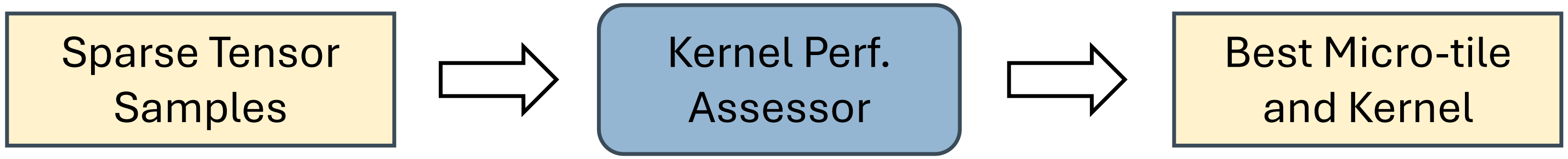
Permutation Invariant Transformation

- An axis of an einsum notation is **PIT-axis** if and only if any shuffling of data on this axis does not affect the correctness of the operator
  - All the computations on a PIT-axis are **commutative** and **associative**

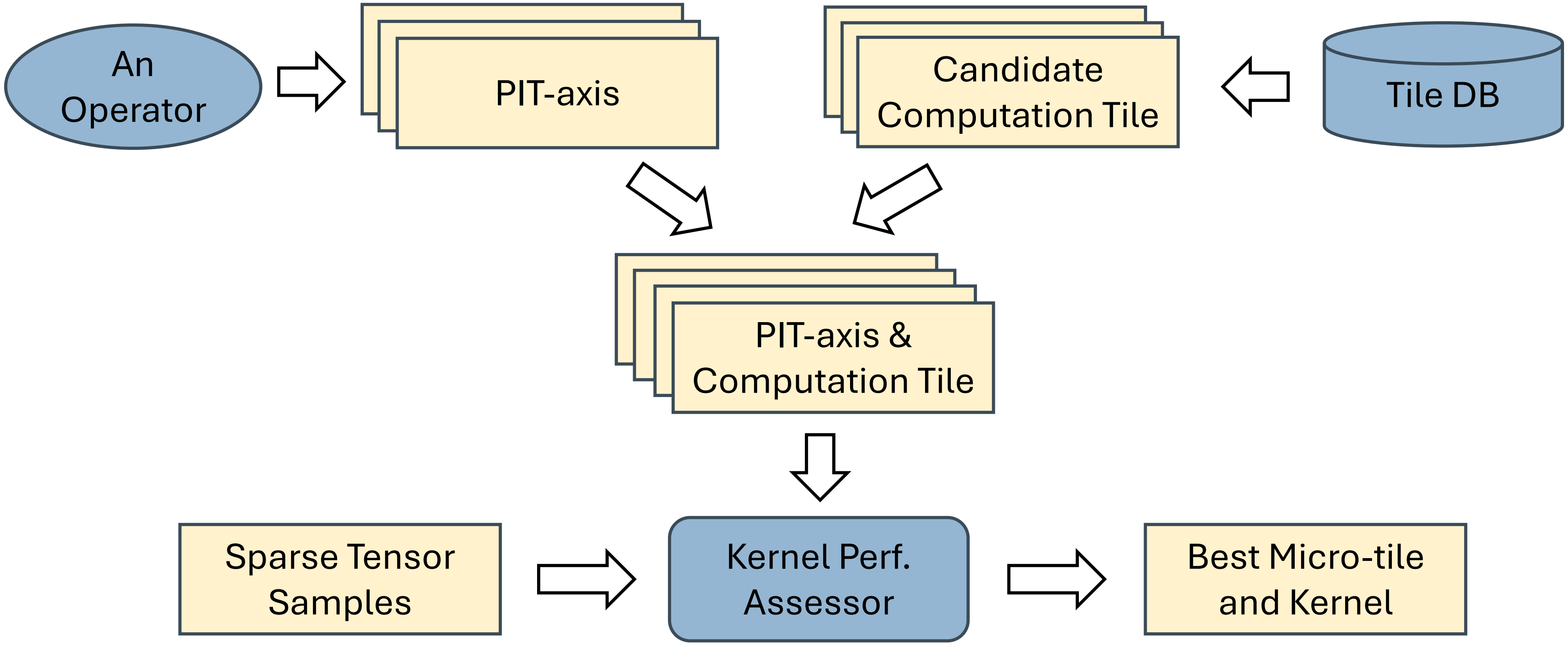
Operator	Tensor Expression	PIT-axis	not PIT-axis
ReduceSum	$C[p]$ += $A[p,l]$	p, l	
Vector Addition	$C[p] = A[p]+B[p]$	p	
MatMul	$C[m,n]$ += $A[m,k]*B[k,n]$	m, n, k	
BatchMatMul	$C[b,m,n]$ += $A[b,m,k]*B[b,k,n]$	b, m, n, k	
Convolution	$C[n,f,x,y]$ += $A[n,m,x+i,y+j]*B[f,m,i,j]$	n, m, f	x, y, i, j

A **PIT rule** contains the combination of a PIT-axis, a micro-tile shape, and a dense computation tile.

Micro-tile Selection for Kernel Construction

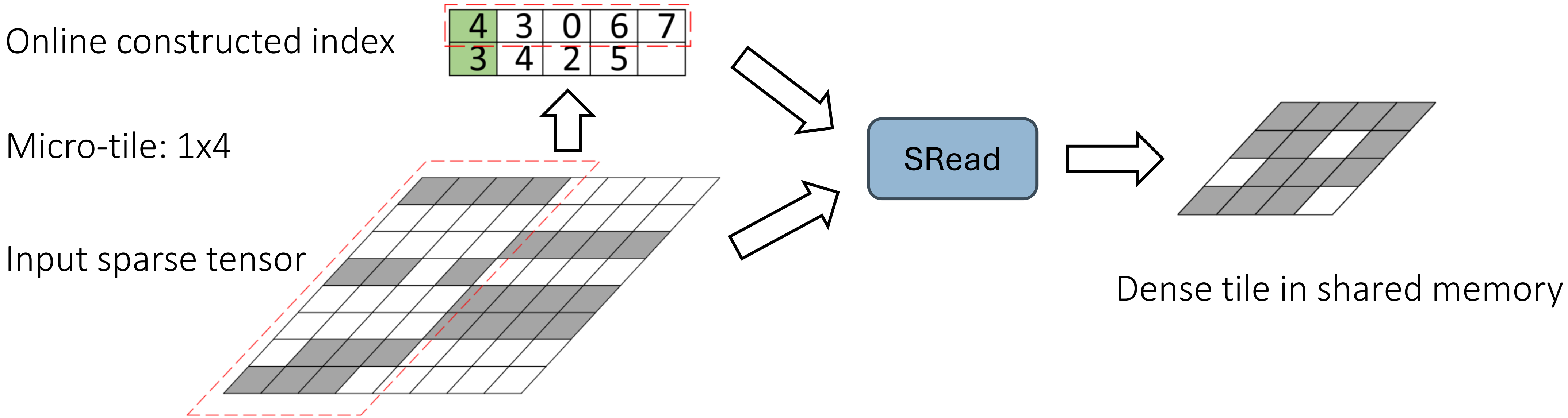


Micro-tile Selection for Kernel Compiling

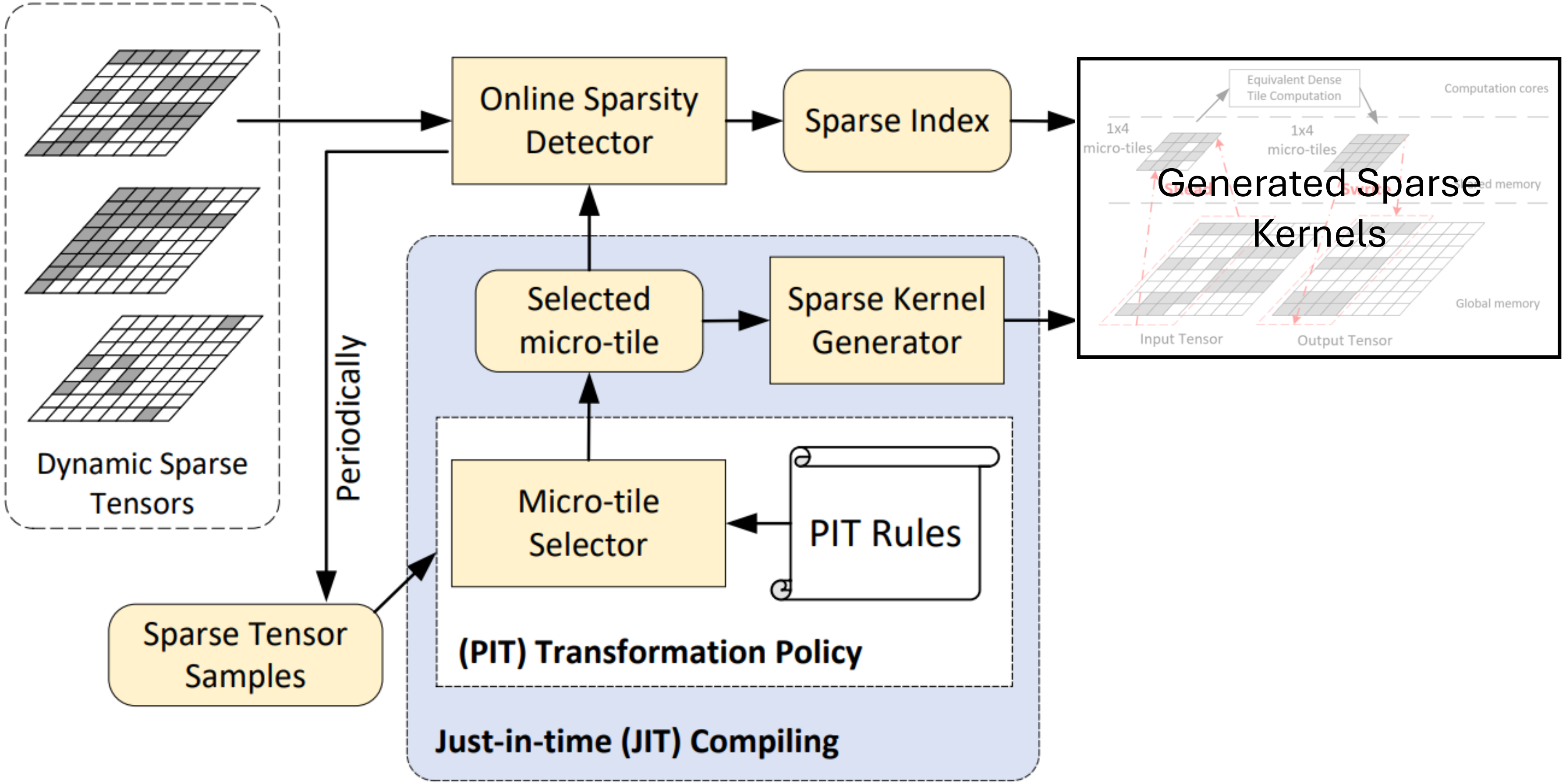


Online Sparsity Detection and Index Construction

- Minimized construction overhead
  - Constructing index without reformatting the sparse tensor (zero copy)
  - Parallelized index construction in an out-of-order manner thanks to PIT
  - Detecting non-zero values at the granularity of micro-tile



PIT Online Execution Workflow



Evaluation

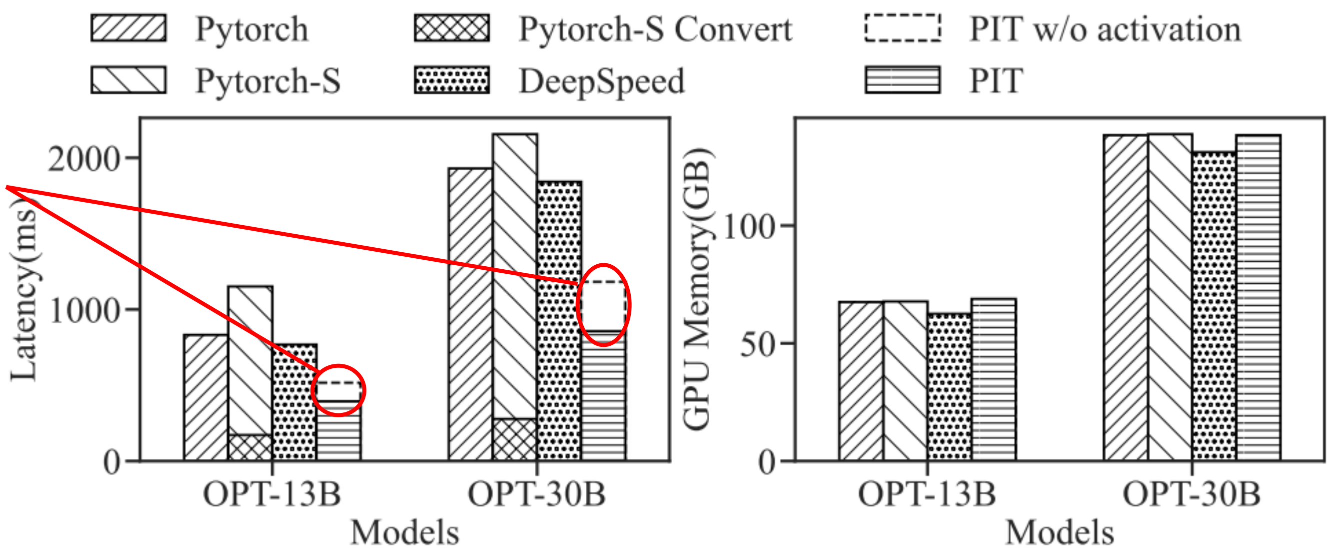
- Comprehensive experiments on popular models, different datasets, precisions, and accelerators
  - Evaluated both inference and training
  - Compared with 6 end-to-end inference libraries
    - PyTorch, PyTorch-S, Tutel, DeepSpeed, MegaBlocks, TurboTransformer
  - Compared with 4 sparse kernel libraries
    - cuSPARSE, OpenAI Triton, Sputnik, SparTA

Models	Datasets	Model Structure	Precision	Devices
Switch Transformers[29]	MNLI [59]	Encoder Decoder MoE	fp16,fp32	A100
Swin-MoE [37]	ImageNet	Encoder MoE	fp16	A100
OPT [66]	Alpaca [58]	Decoder	fp32	V100
BERT [22]	GLUE [59], News [27] etc.	Encoder	fp32	V100
Longformer [14]	Arxiv [21]	Encoder	fp32	V100
MuseFormer [65]	LMD [54]	Decoder	fp32	V100

Evaluation

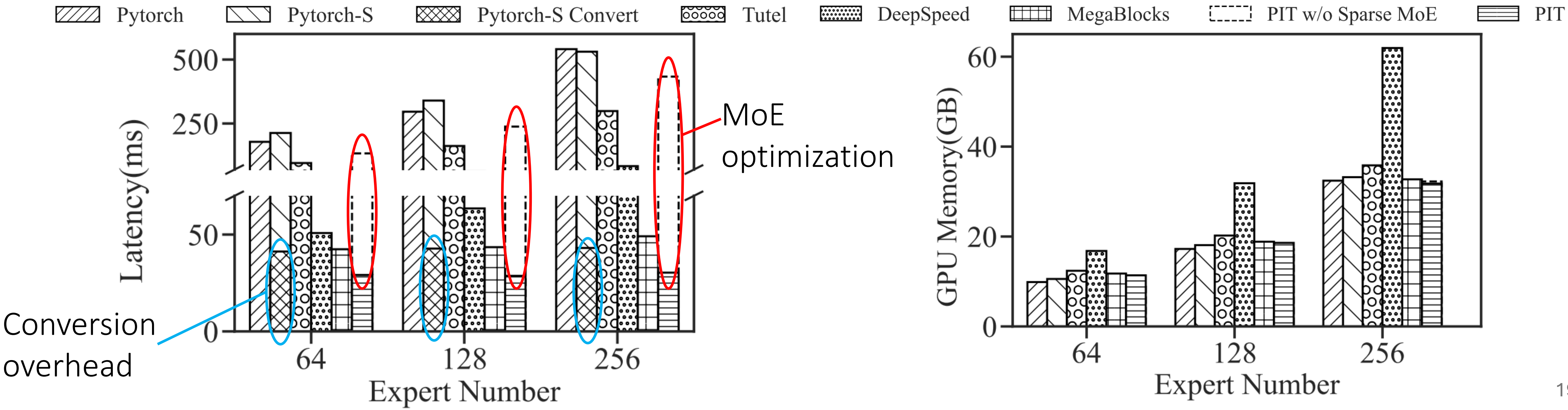
- End-to-End Inference of **OPT**
  - 8xV100-32GB GPUs
  - FP32 inference latency
  - Batch size is 32
  - 2.3x, 2.5x, 2.2x faster over PyTorch, PyTorch-S, DeepSpeed (OPT-30B)
  - Gains from **varying seq. length** and **activation sparse**
  - Memory usage is similar to the baselines

Activation sparse optimization



Evaluation

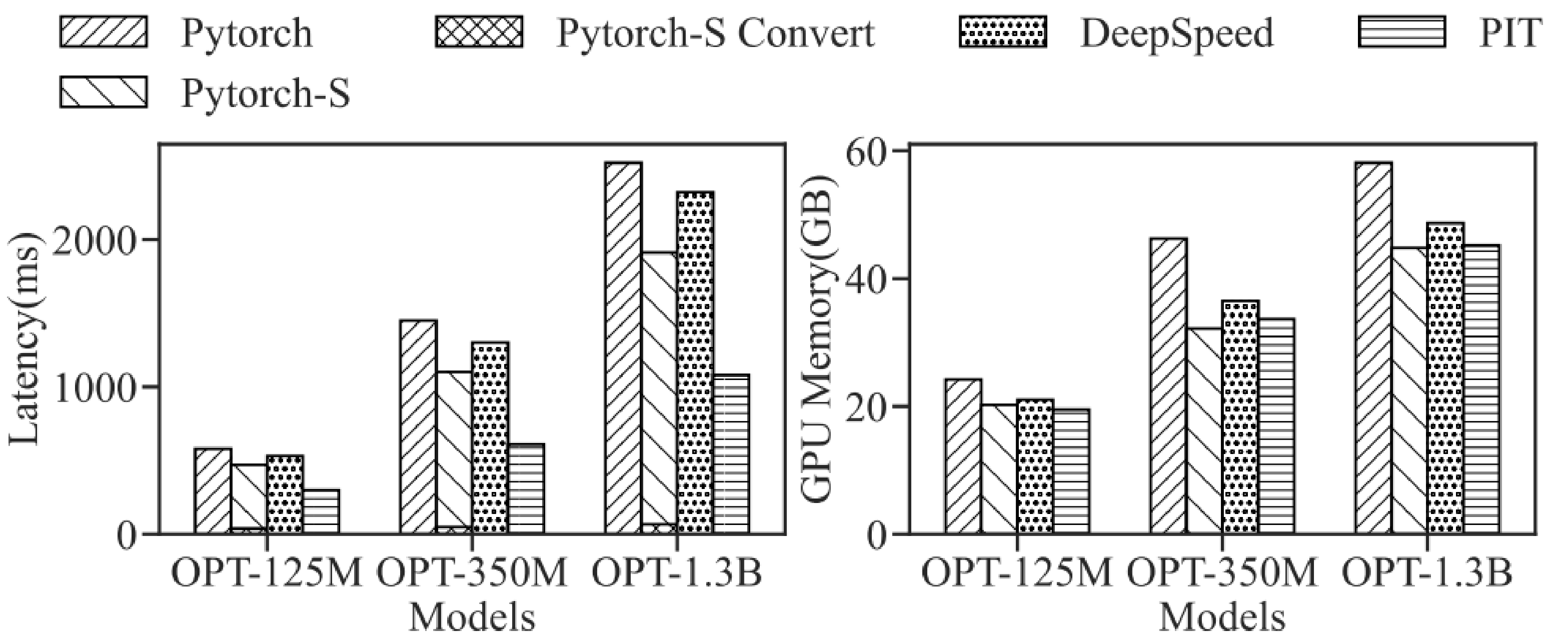
- End-to-End Inference of **Switch Transformer** (MoE)
  - 1xA100-80GB GPU
  - FP16 inference latency
  - Batch size is 8
  - 17.8x, 17.5x, 9.8x, 2.8x, 1.6x faster over PyTorch, PyTorch-S, Tutel, DeepSpeed, MegaBlocks for 256 experts
  - Gain comes from **MoE** and **varying seq. length**
  - Memory usage is low



Evaluation

- End-to-End Training of **OPT**
  - 1xA100-80GB GPU
  - FP32 training speed
  - Batch size is 8 or 4

- 2.3x, 1.8x, 2.1x faster over PyTorch, PyTorch-S, DeepSpeed (OPT-1.3B)
- Gain comes from **varying seq. length** and **sparse attention**
- Memory usage is low

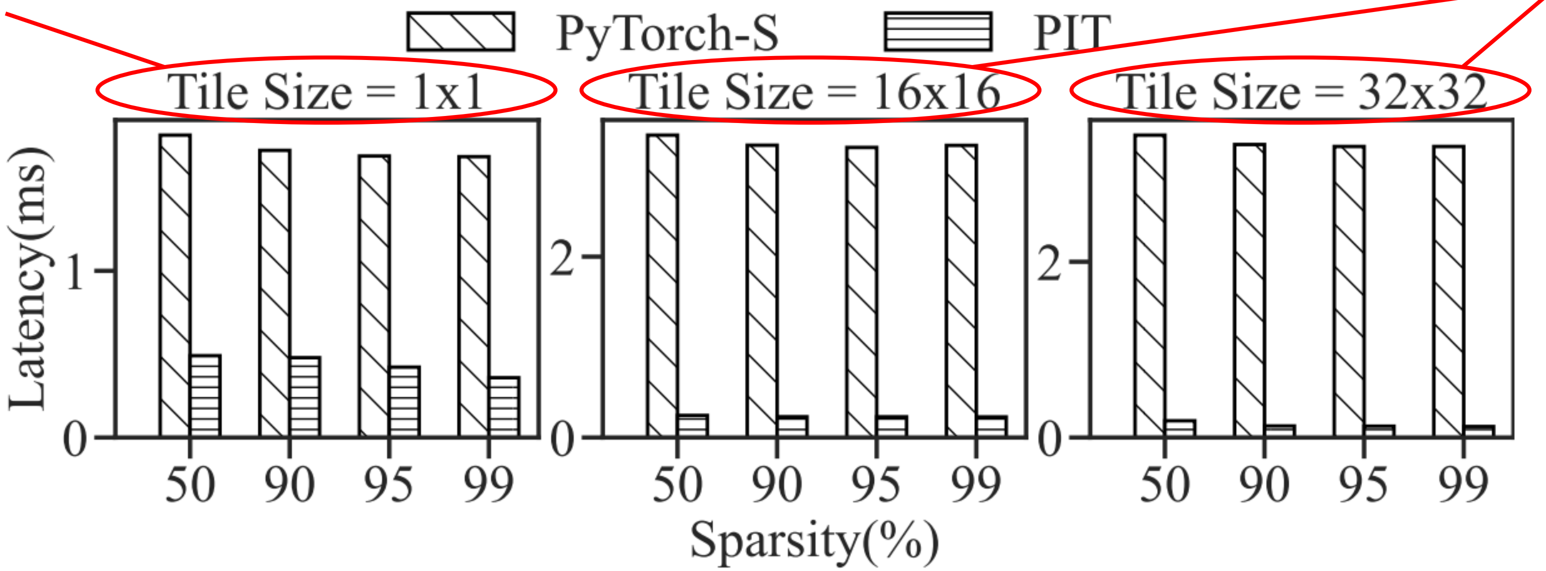


Evaluation

- Sparse Index Conversion Overhead
  - 1xV100-32GB GPU
  - Index construction latency of a sparse tensor with shape 4096x4096
  - Gain comes from **out-of-order index construction** and **zero copy of data**

PyTorch-S chooses cuSPARSE

PyTorch-S chooses OpenAI Triton



Conclusion

- PIT demonstrates a novel and effective way of handling dynamic sparsity, a growing trend in deep learning especially LLMs
- With permutation invariant transformation, PIT achieves high computation efficiency, low computation waste, and minimal data conversion overhead
- The idea of decoupling data format and computation logic in PIT can be generalized to other scenarios, e.g., low-bit computation, mixed precisions

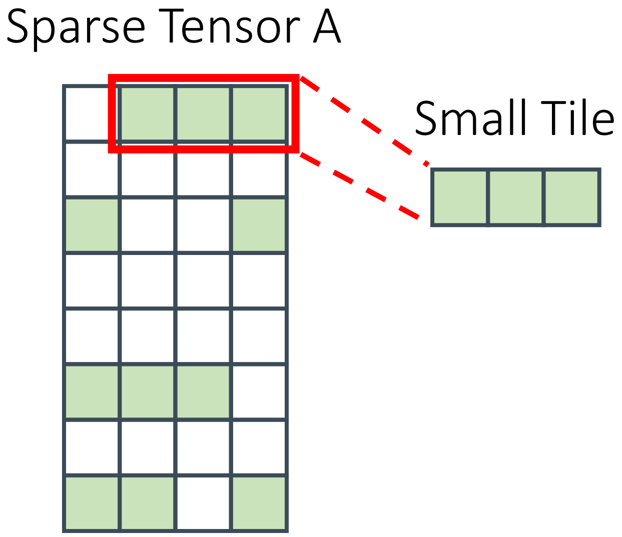
Conclusion

- PIT demonstrates a novel and effective way of handling dynamic sparsity, a growing trend in deep learning especially LLMs
- With permutation invariant transformation, PIT achieves high computation efficiency, low computation waste, and minimal data conversion overhead
- The idea of decoupling data format and computation logic in PIT can be generalized to other scenarios, e.g., low-bit computation, mixed precisions

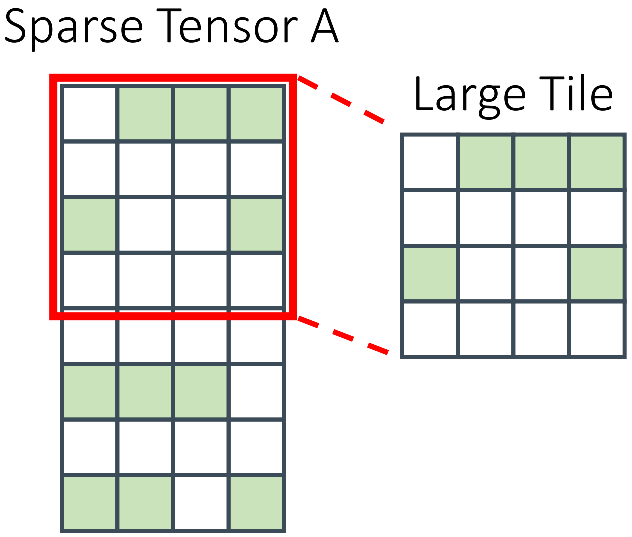
Q&A

Q&A

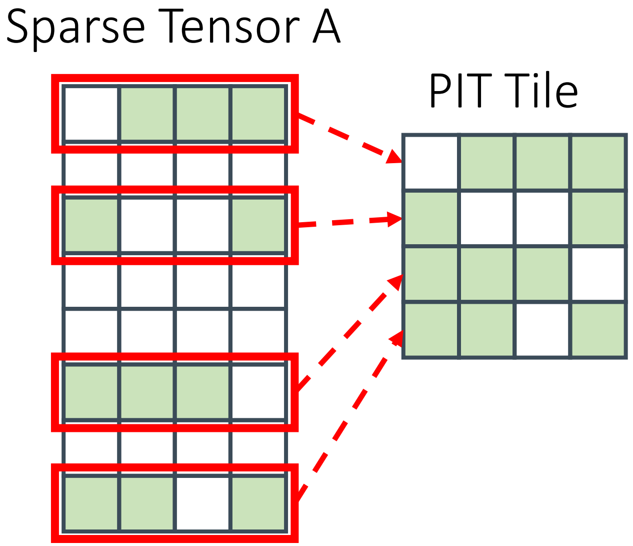
Inefficiency Due to Dynamic Sparsity



√ Low waste
× Low SM Utilization

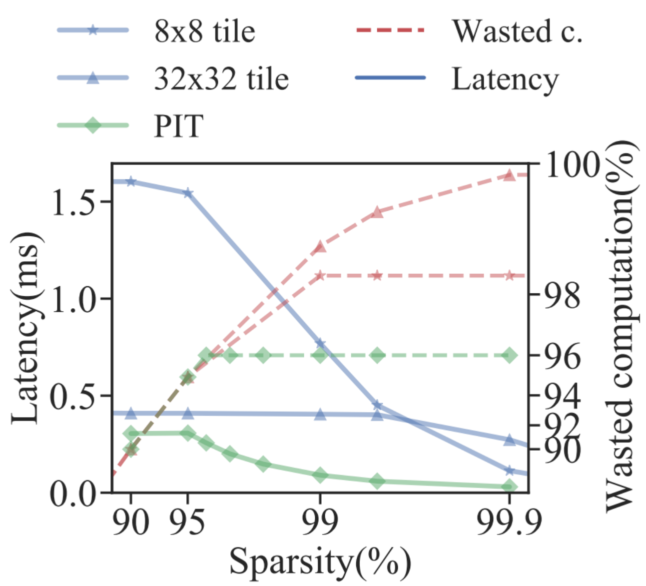


√ High SM Utilization
× High waste

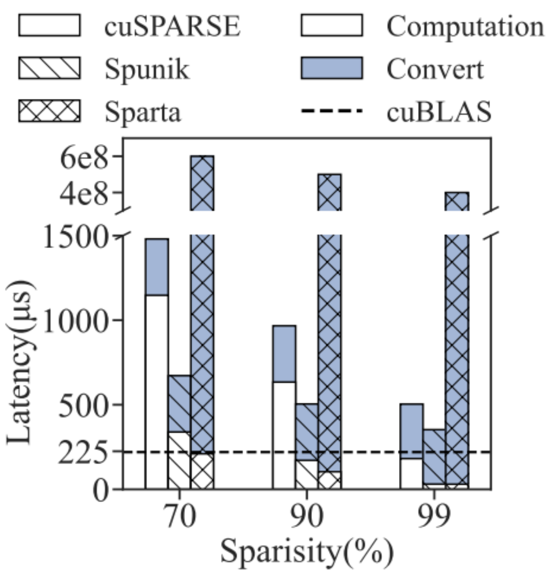


√ High SM Utilization
√ Low waste
√ On-the-fly

Inefficiency Due to Dynamic Sparsity



Smaller tiles (e.g., 8x8) have poor performance due to inefficient tile computation though less wasted computation



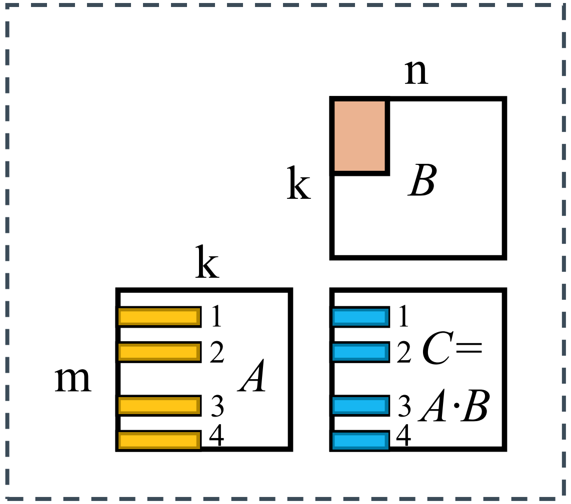
Sparsity specific kernels requires online data format conversion (e.g., CSR), leading to high overhead

Is it possible to leverage computation efficient large tiles while introducing low conversion overhead?

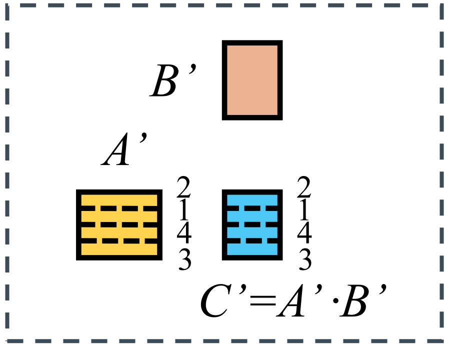
Opportunity: Sparse-to-dense Transformation

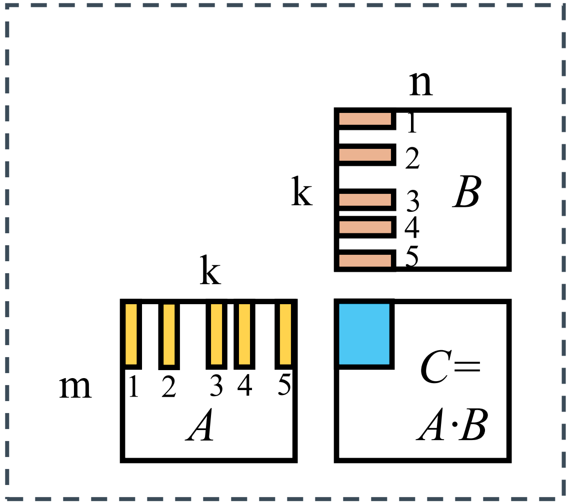
Sparse matrices

Dense tile computation



Sparse data in A rearranged to dense data A' along m-axis





Sparse data in A rearranged to dense data A' along k-axis