SC20

Everywhere we are | more than hpc.

Accelerating Sparse DNN Models without Hardware-Support via Tile-Wise Sparsity

2020/11 • Shanghai

Accelerating Sparse DNN Models without Hardware-Support via Tile-Wise Sparsity

Cong Guo[1], Bo Yang Hsueh[2], Jingwen Leng[1], Yuxian Qiu[1], Yue Guan[1], Zehuan Wang[2], Xiaoying Jia[2], Xipeng Li[2], Yuhao Zhu[3] and Minyi Guo[1]

[1]Shanghai Jiao Tong University, Emerging Parallel Computing Center, REArch (Resilient and Efficient Architecture) group
[2]NVIDIA, [3]University of Rochester

Outline

- Background & Motivation

Tile-Wise Sparsity

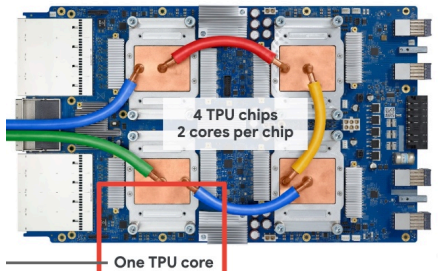
Efficient GPU Implementation

Evaluation

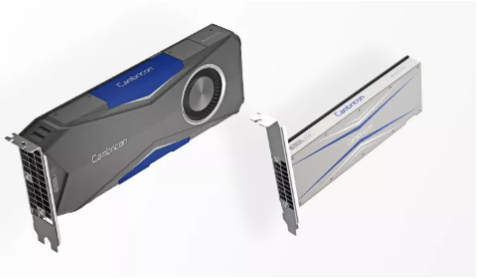
Dense GEMM Accelerator

GEMM-based accelerators are dominant owing to their wide applicability.

Nvidia GPU Tensor Core

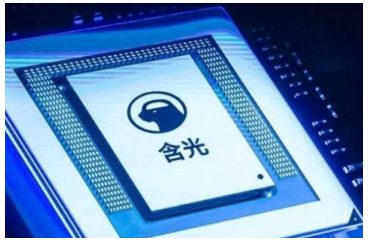


Google TPU

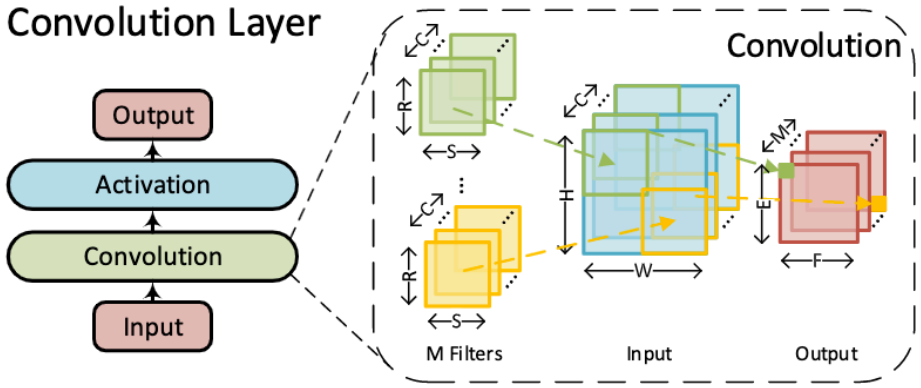


Cambrian MLU

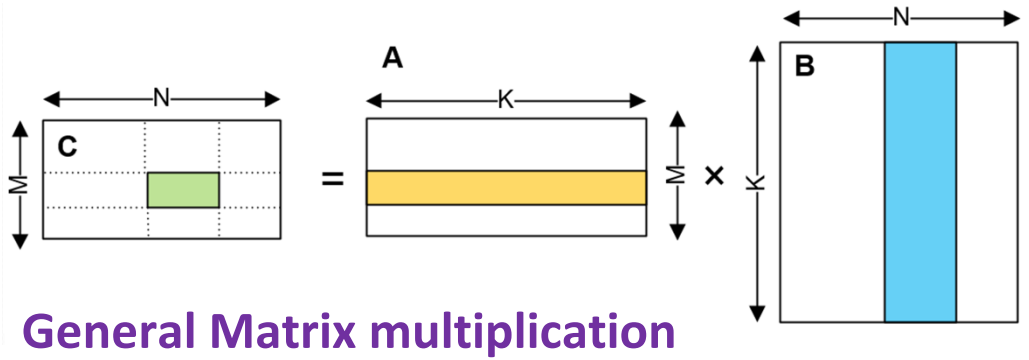
Huawei Ascend



Alibaba Hanguang



Img2col



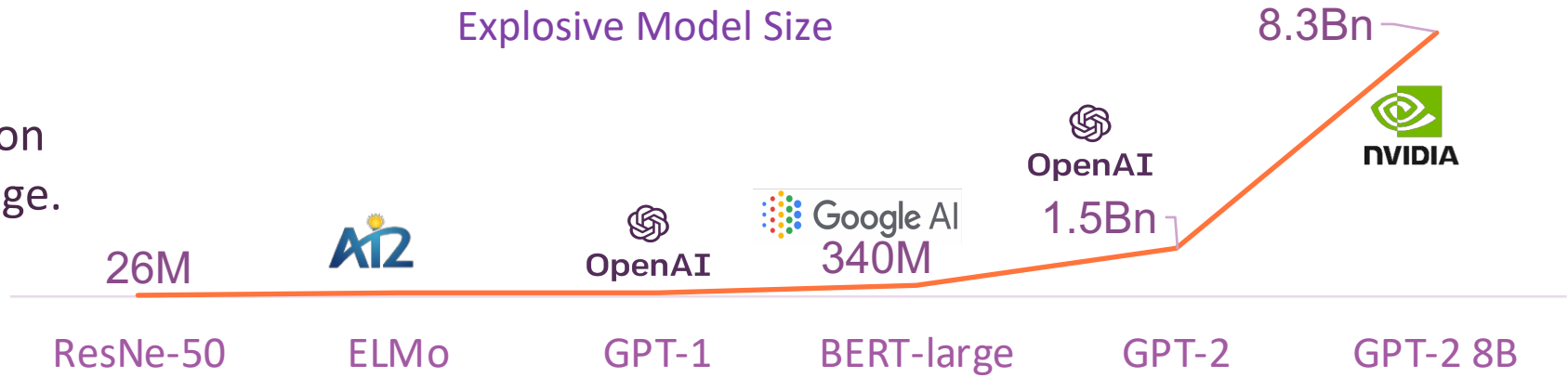
General Matrix multiplication

Convolution operations that dominate computer vision models are converted to the GEMM.

NLP models are naturally equivalent to the GEMM operation.

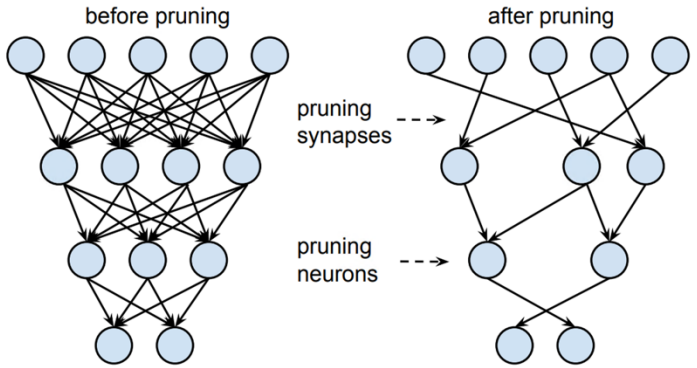
DNN Models and Pruning

Enormous computation cost and memory usage.



Pruning

Fewer parameters and less computation cost



VGG-16
92.5%
Sparsity

[Song Han, etc. NIPS'15]

The DNN models are sparse! Pruning is an effective and promising approach to reduce the DNN latency.

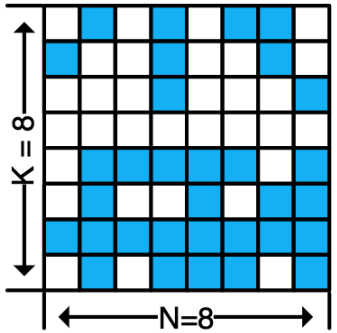
Sparsity Pattern

Irregular, Random
High Accuracy
Low efficiency

Balance

Regular, Structured
Low Accuracy
High efficiency

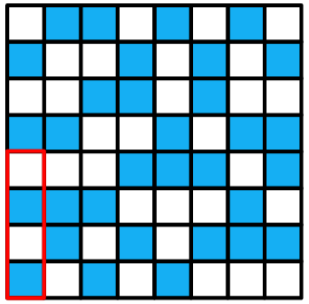
Element-Wise (EW)



No constraint (1*1 block)

Software:
MKL, *cuSparse*
Hardware:
OuterSPACE, [HPCA'18]
SpArch, [HPCA'20]

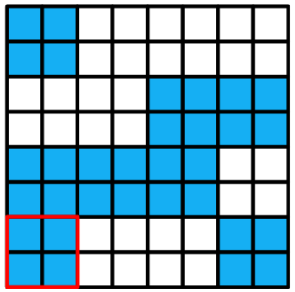
Vector-Wise (VW)



Fixed sparsity of each vector

Software:
Balanced Sparsity [AAAI'19]
Hardware:
Bank-Balanced Sparsity [FPGA'19]
Sparse Tensor Core [MICRO'19]
Tesla A100 [GTC'20]

Block-Wise (BW)

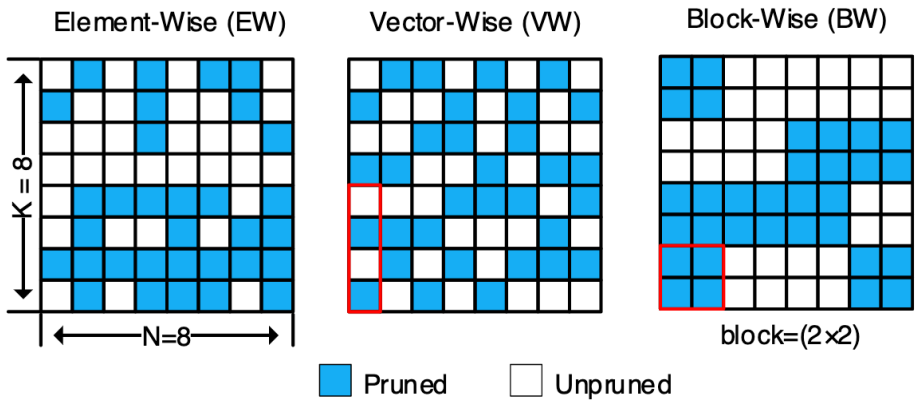


n*n block

Software:
Block-sparse [*arXiv'17*]
8x8,16x16(CUDA) 32x32, 64x64(Tensor)

BW is friendly to dense GEMM accelerator.

Sparsity Pattern Efficiency



Pattern	Core	Library	Speedup	Density
Element Wise	CUDA	cuSparse	0.06x	63%
Vector Wise	CUDA	cuSparse	0.07x	56%
Block Wise	Tensor	Block-sparse	0.33x	50%

[Song Han, etc. NIPS'15] *[Block-Sparse, arXiv'17]*

[Sparse Tensor Core, Micro'19] *[Balanced Sparsity, AAAI'19]*

GPU: Tesla V100 32GB
Workload: BERT(MNLI)
Software: TensorFlow 1.15(Fine-tune)
cuBlas, cuSparse and Block-sparse (Inference)
Accuracy loss < 1%

1. BW achieves the best performance.
2. BW is still 3× slower than the dense model on the tensor core.
3. They are all Inefficient on the existing dense GEMM hardware.

A new sparsity pattern that can match the existing hardware features while maintaining the fine granularity, which is critical for achieving the high model accuracy.

Outline

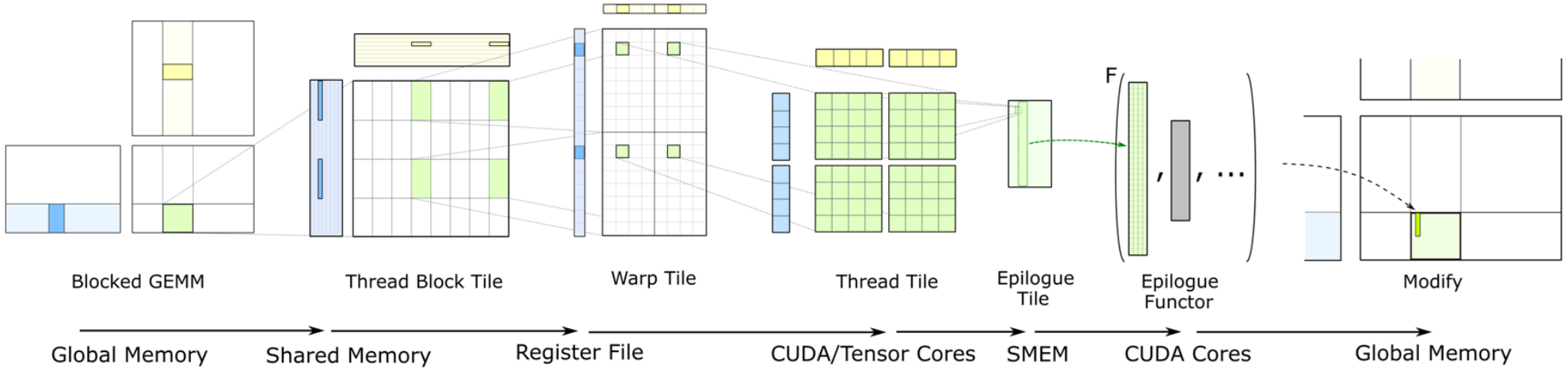
Background & Motivation

- Tile-Wise Sparsity

Efficient GPU Implementation

Evaluation

Algorithm-software Co-designed Tile-Wise Sparsity

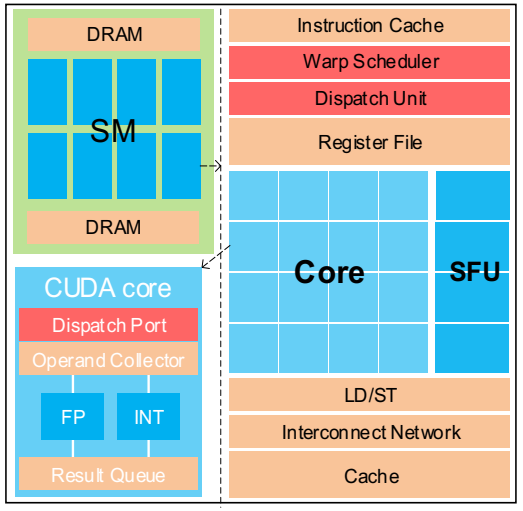


$C = A \times B$

Software: CUTLASS – Tiling GEMM

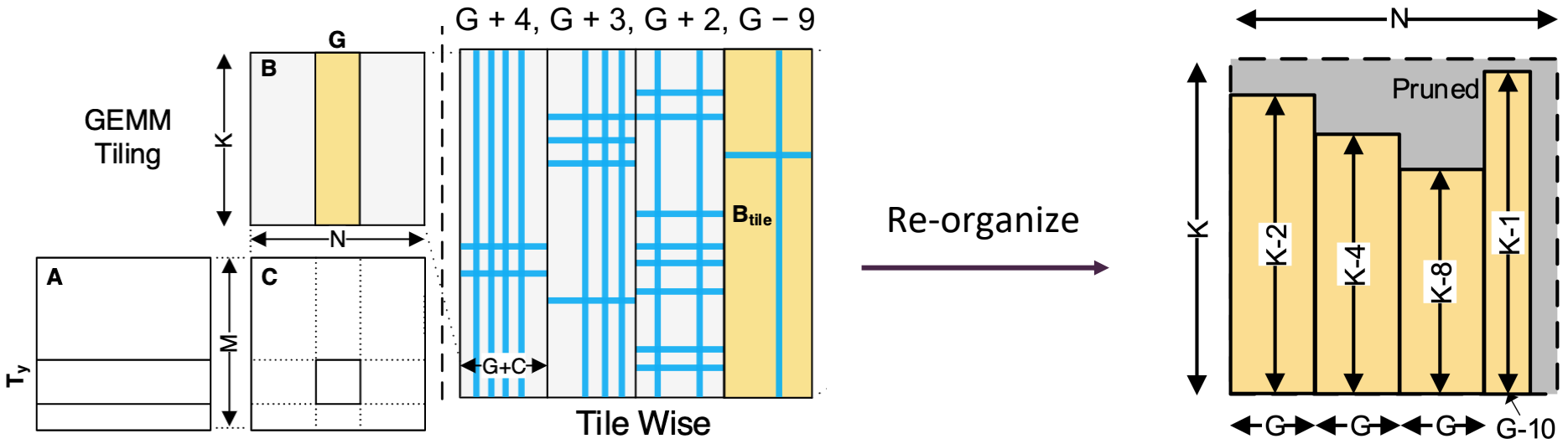
Tile-Wise Sparsity. An algorithm-software co-designed pruning method that reduces the DNN latency on existing dense architectures while maintaining high accuracy without special hardware support.

key insight: a tiling-friendly sparsity pattern



Hardware: Tesla V100

Tile-Wise Sparsity



GEMM: M, N, K
T_x = G = Granularity
T_y = Tile Length (y)
C = pruned column

The key idea of our tile-wise pattern is to prune each B_{tile} with the regular row and column pruning.

The tiling based GEMM is widely used in the dense GEMM accelerators, such as TPU, not only GPU.

Pruning algorithm

- Importance Score
- Gradually Pruning
- Global Weight Pruning
- Apriori Tuning

More details on the paper…

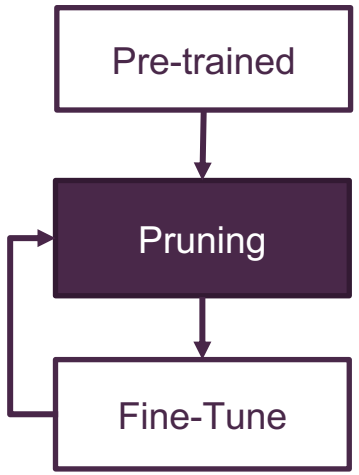
$$\Delta L(w) = \sqrt{(L(w = w_i) - L(w = 0))^2}$$

$$L(w = 0) = L(w_i) + \frac{\partial L(w_i)}{\partial w} * w_i + R_1(w = 0)$$

$$\Delta L(w) \approx \sqrt{(\frac{\partial L(w_i)}{\partial w} * w_i)^2}$$

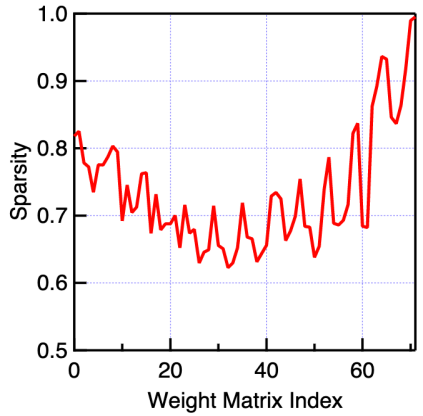
Importance Score

[P. Molchanov , etc. CVPR 2019]



Gradually Pruning

[Song Han, etc. NIPS'15]



Uneven distribution of EW

Global Weight Pruning
Apriori Tuning

Outline

Background & Motivation

Tile-Wise Sparsity

- Efficient GPU Implementation

Evaluation

Efficient GPU Implementation

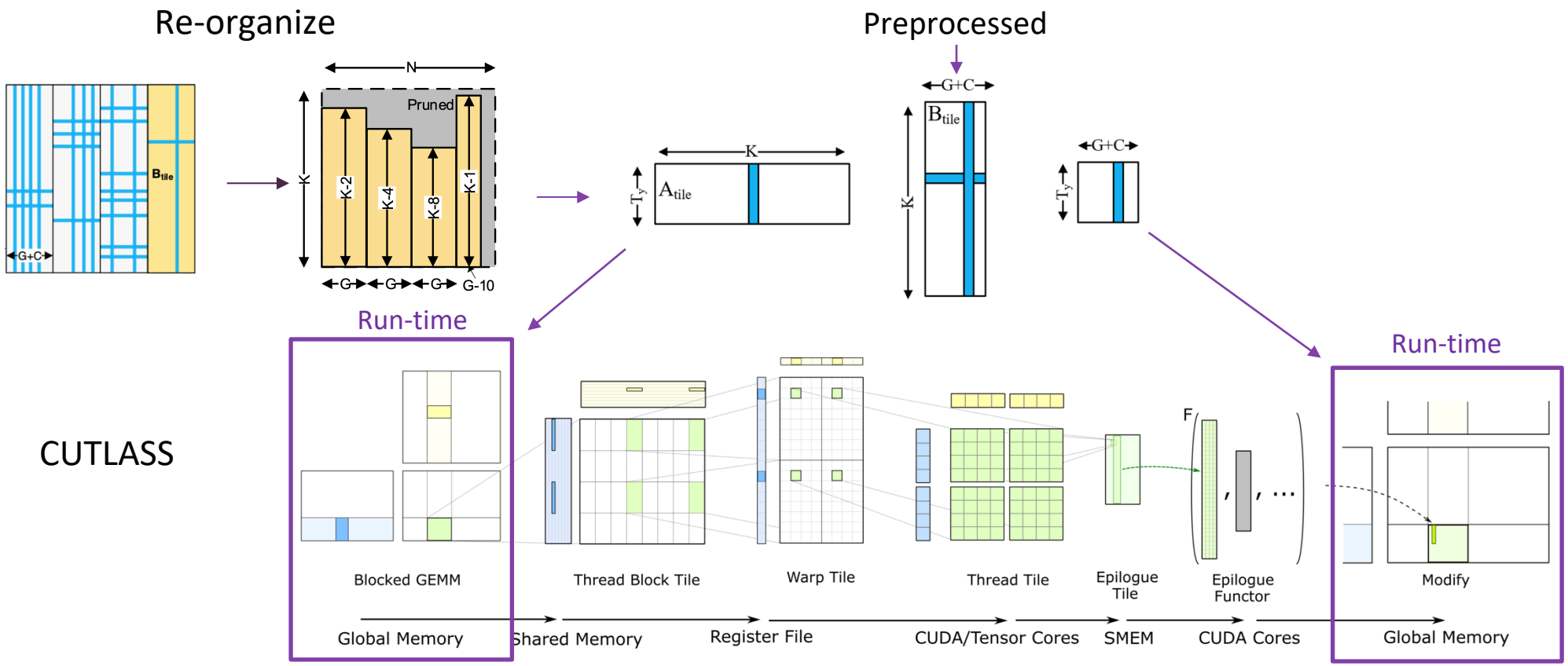
Goal:

Execute TW sparsity on GPU (including CUDA core and Tensor core) efficiently.

Three optimizations leveraging GPU's programming features:

1. Memory accesses coalesce (via memory layout transpose)
2. Kernel reduction (via fusion)
3. Load imbalance mitigation (via concurrent kernel)

Baseline GEMM Tiling



A[offset$_A$+ offset$_k$[k]]
B[offset$_B$]
C[offset$_C$+ offset$_n$[n]]

Sparsity in the Global
Density in the Core
Execute Efficiently!

Memory Accesses Coalesce

Optimization 1

Column skipping

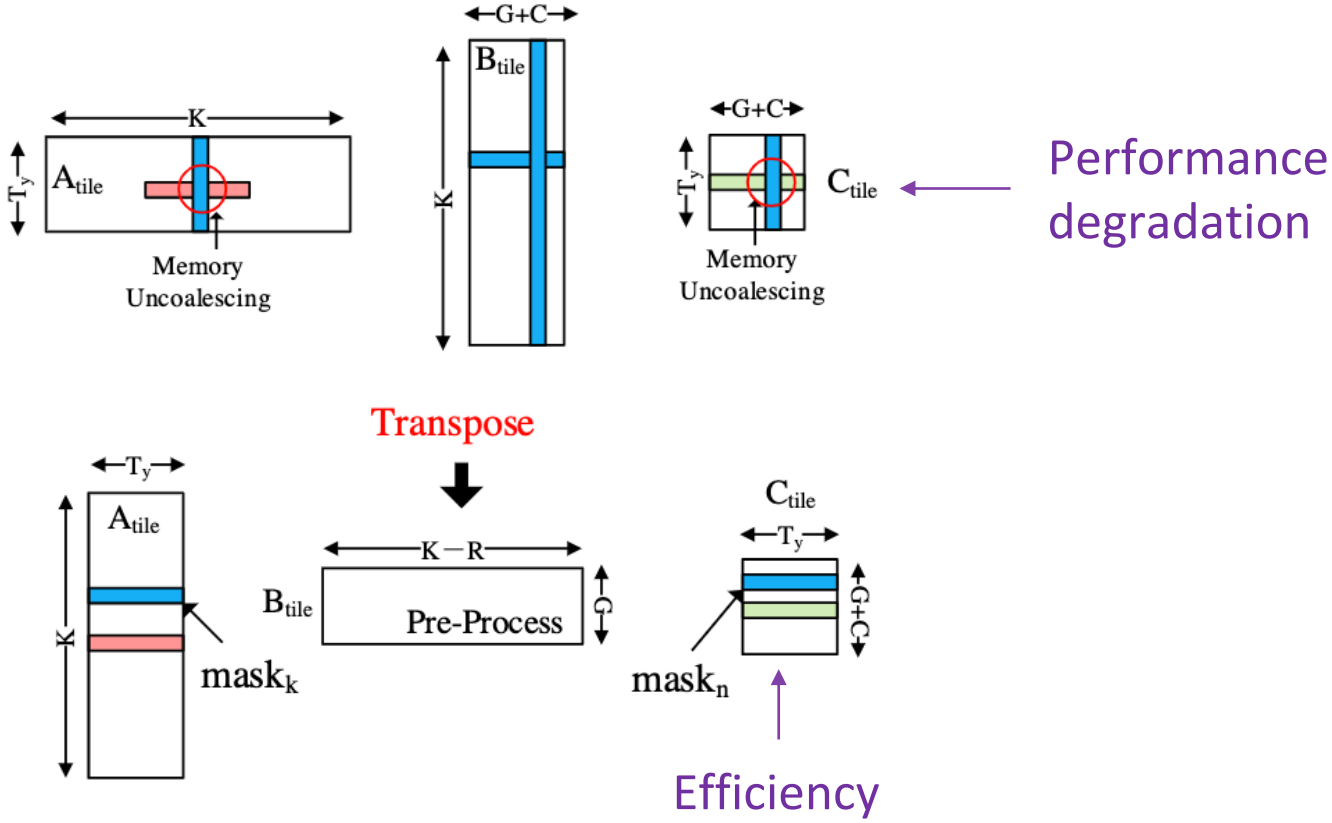
↓

Transpose to eliminate Memory uncoalescing

↓

Row skipping

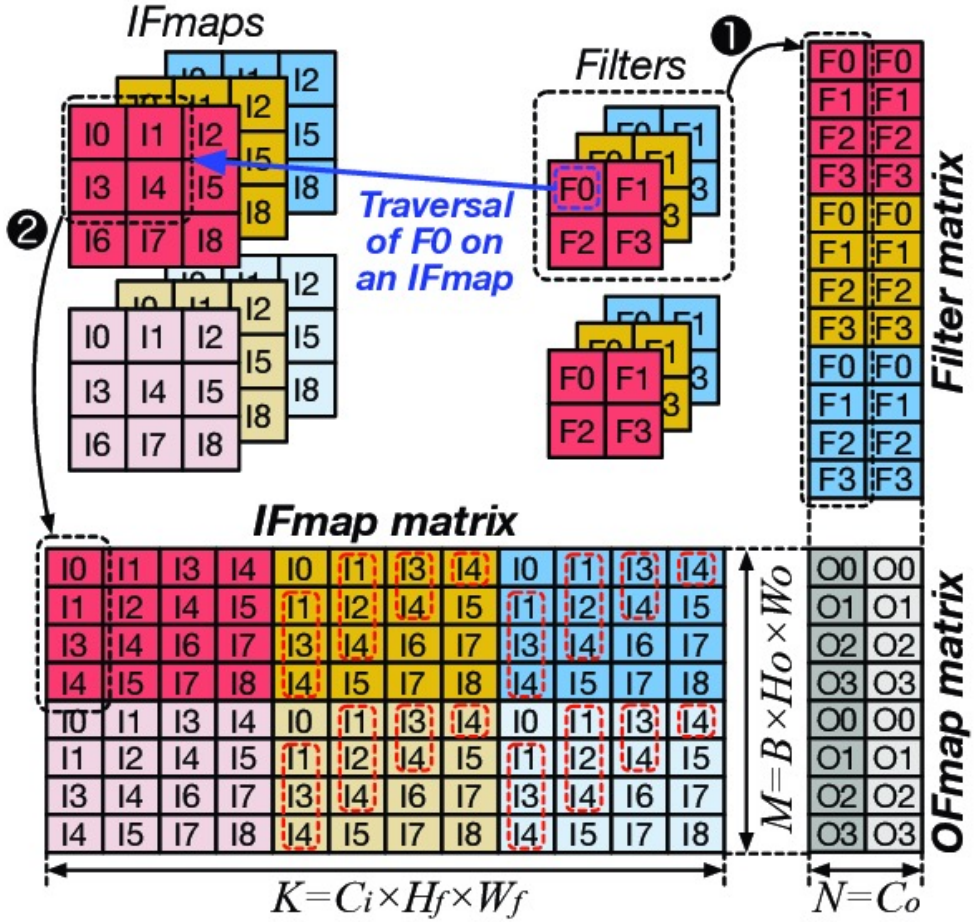
Efficiency



Kernel Fusion

Optimization 2

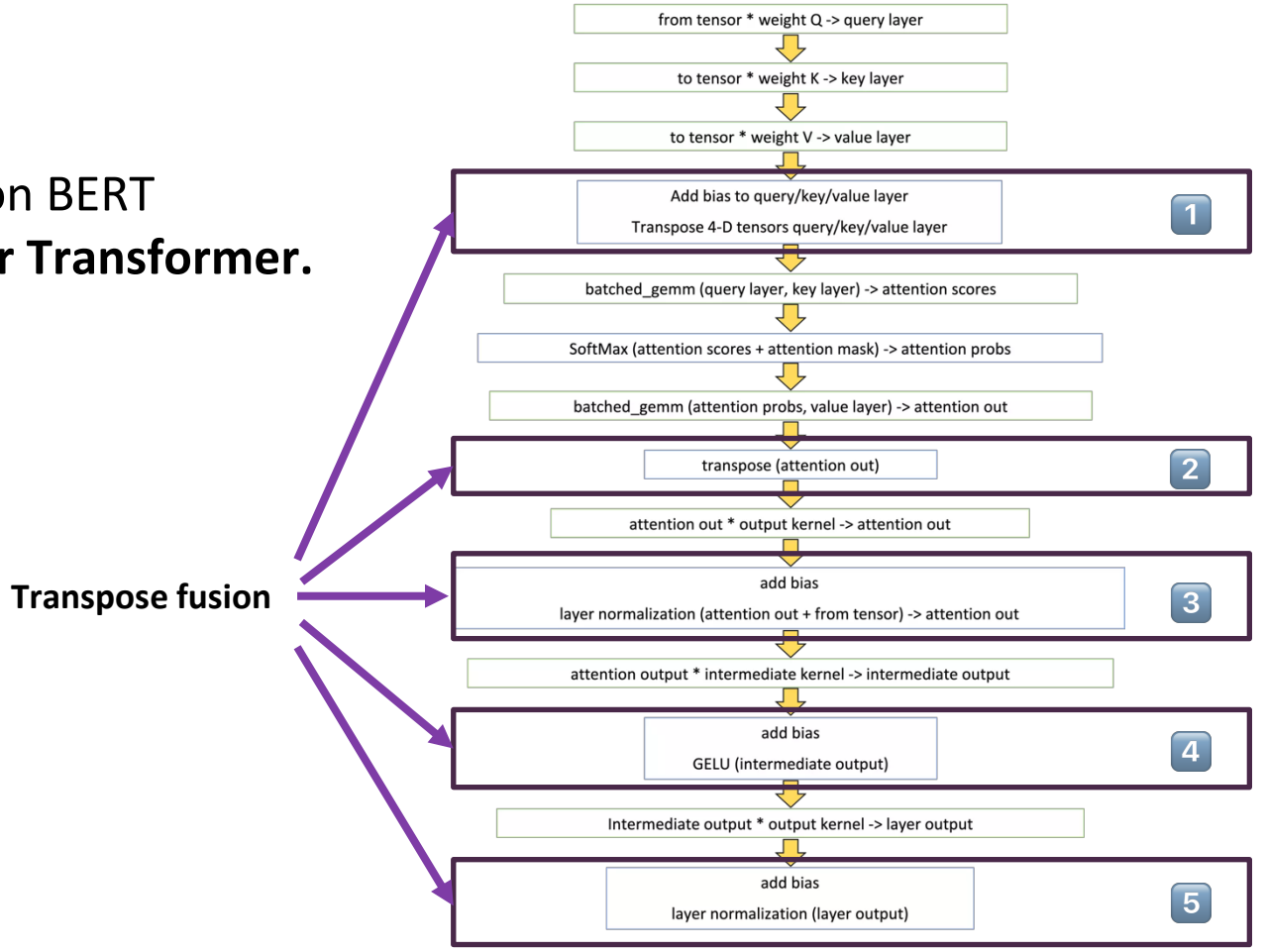
- Fused with img2col on CNN
- Transpose is free to GPU and TPU



Kernel Fusion

Optimization 2

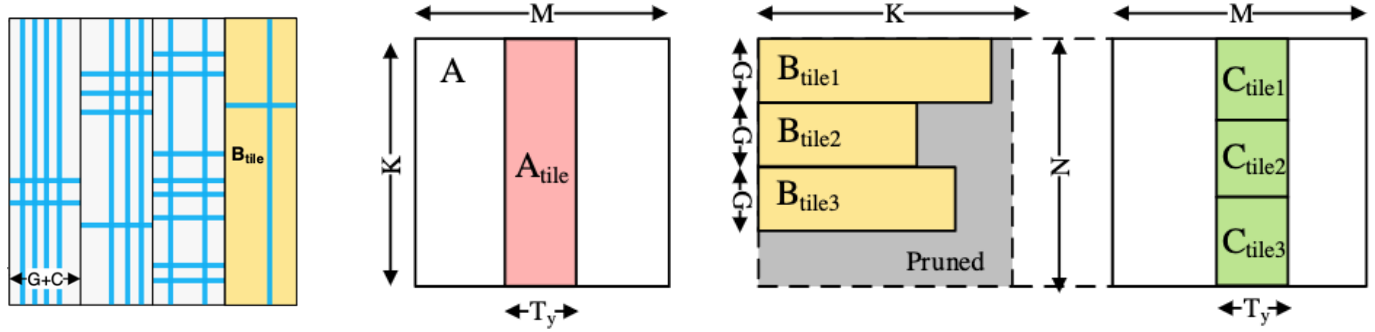
Fused with Transpose on BERT
Based on **NVIDIA Faster Transformer.**



Load Imbalance Mitigation

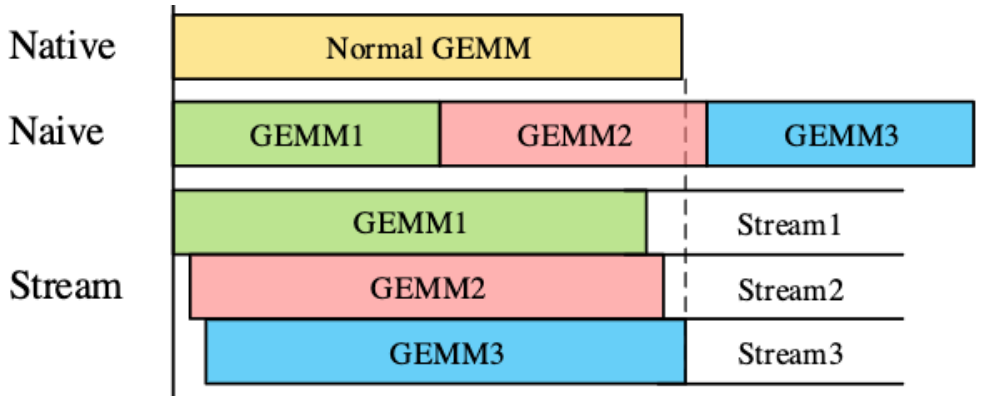
Optimization 3

Condensed Tile



$\lceil \frac{N}{G} \rceil$ tiles

Multi-Stream



Concurrent kernel execution

Overlap the computation of different tiles by assigning to different streams, and rely on the underlying scheduler to maximize the resource utilization.

Outline

Background & Motivation

Tile-Wise Sparsity

Efficient GPU Implementation

- Evaluation

Methodology

Hardware: NVIDIA Tesla V100 32GB GPU

Sparsity pattern:

Pattern	Core	Library
Tile Wise (TW)	Tensor-fp16 CUDA-fp32	Tile Sparsity*
Block Wise (BW)	Tensor-fp16	Block-sparse
Element Wise (EW)	CUDA-fp32	cuSparse
Vector Wise (VW)	CUDA-fp32	cuSparse**

*Based on CUTLASS 1.3
**V100 can not support sparse tensor core.

DNN models and datasets:

Models	Datasets
BERT-Base	GLUE dataset: MNLI, MRPC, SST, CoLA, RTE, QNLI SQuAD
VGG-16 (CNN)	ImageNet
NMT (LSTM)	IWSLT English-Vietnamese dataset

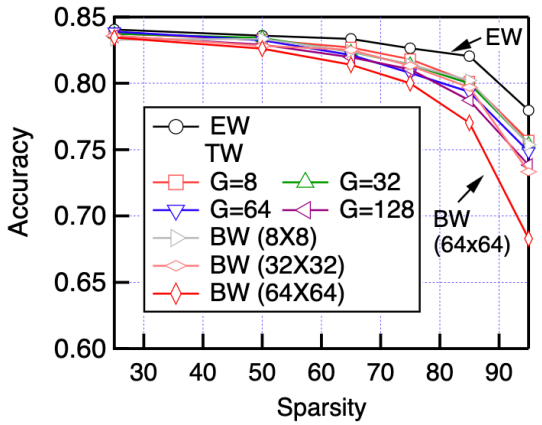
In the rest of this section, we focus on the GEMM execution time unless explicitly mentioned.

Impact of TW Granularity

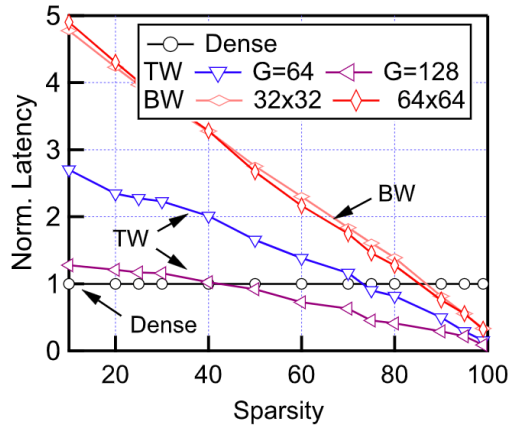
Workload:

Pattern	Granularity	Critical Sparsity*
Dense	128	0%
EW	-	-
BW	32	~85%
BW	64	~85%
TW	64	75%
TW	128	40%

* With the sparsity, the pruning method starts to outperform the dense model latency.
The lower the better.



(a) Accuracy.



(b) Normalized latency.

At the sparsity of 75%, TW-128 has accuracy loss of about 0.9% and 2.4% compared to EW and the baseline dense model at 75% sparsity, respectively. With only 40% sparsity, TW-128 starts to outperform the dense model latency.

BW-64 experiences the most drastic accuracy drop of 4% at 75% sparsity. BW-64 is faster than the dense model only when the sparsity is greater than 85%, which leads to an accuracy loss as high as 10%.

TW exceeds BW in both of speedup and model accuracy. G=128 is sufficient to maintain the model accuracy while providing significant latency reduction for TW.

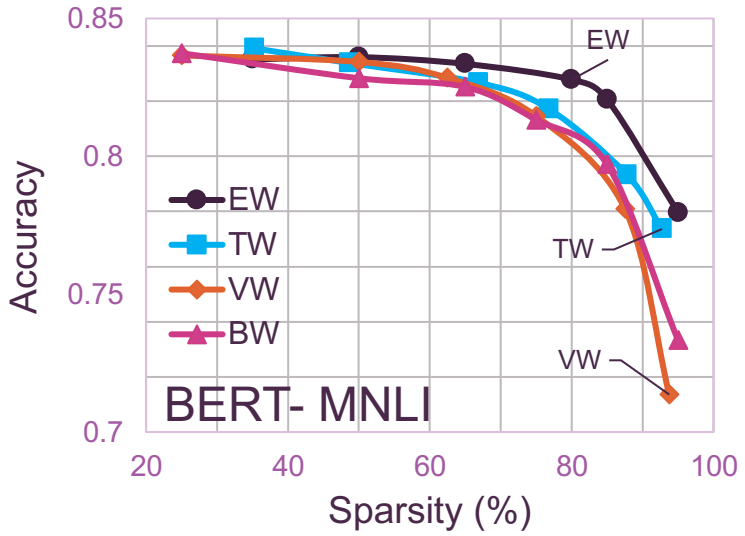
Accuracy

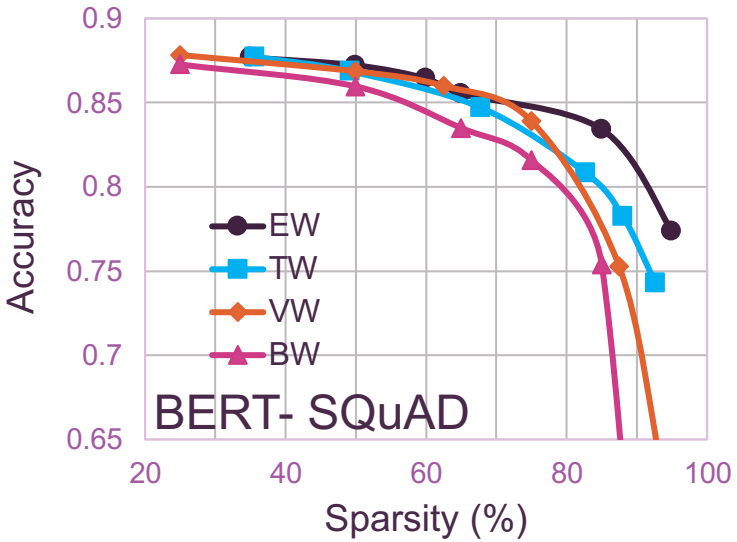
Workload:

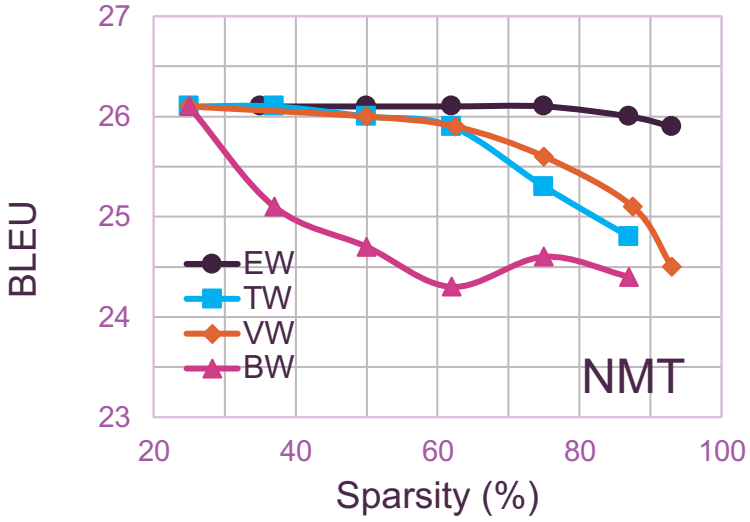
Pattern	Granularity
EW	-
BW	32 * 32
TW	128
VW	16

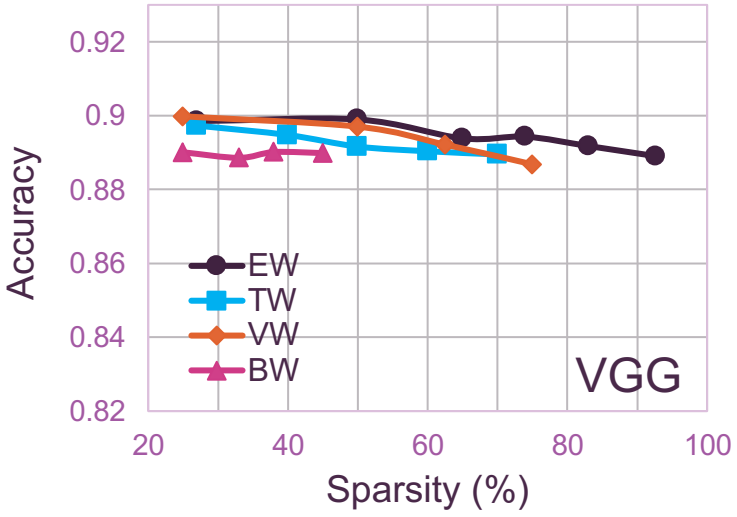
EW the best.
BW the worst.

The accuracy of TW and VW are similar when the sparsity is below 70%. With high sparsity (> 70%), TW generally outperforms the VW with the exception of NMT.



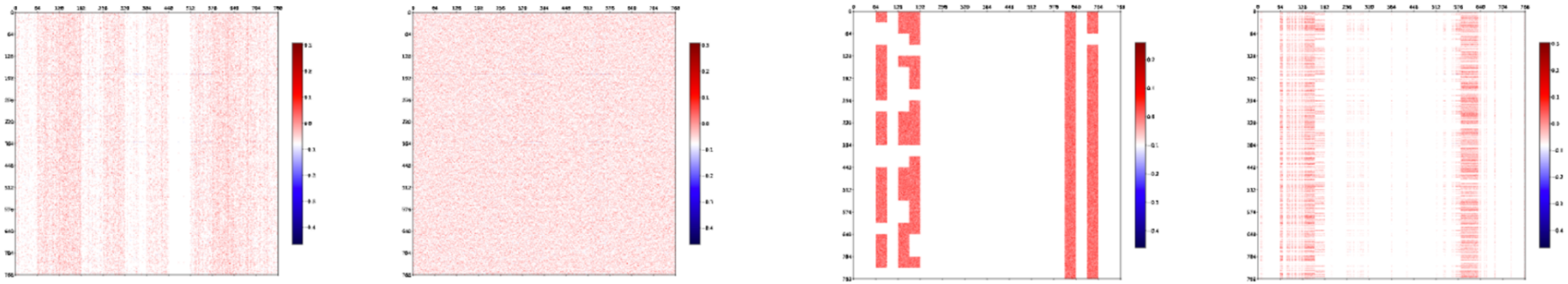






Sparsity Pattern

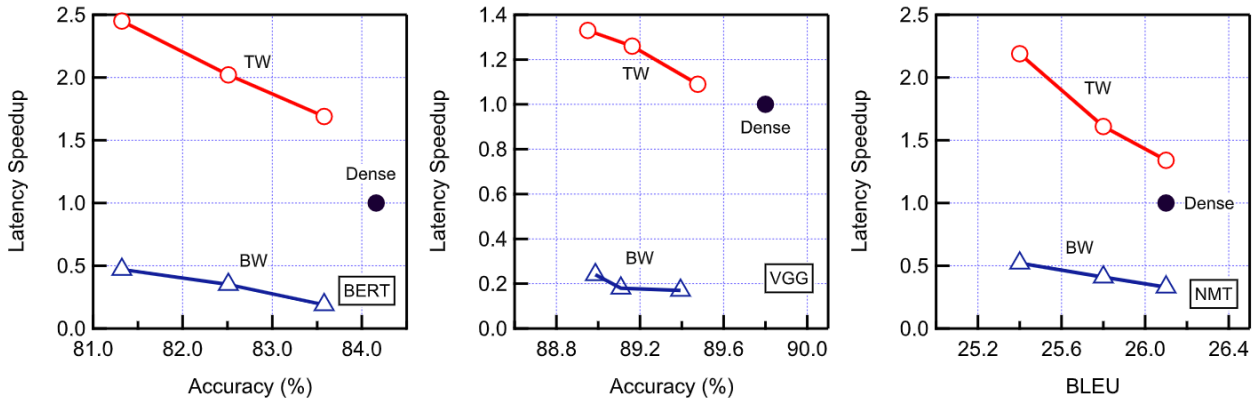
BERT-base Layer-0 W_Q



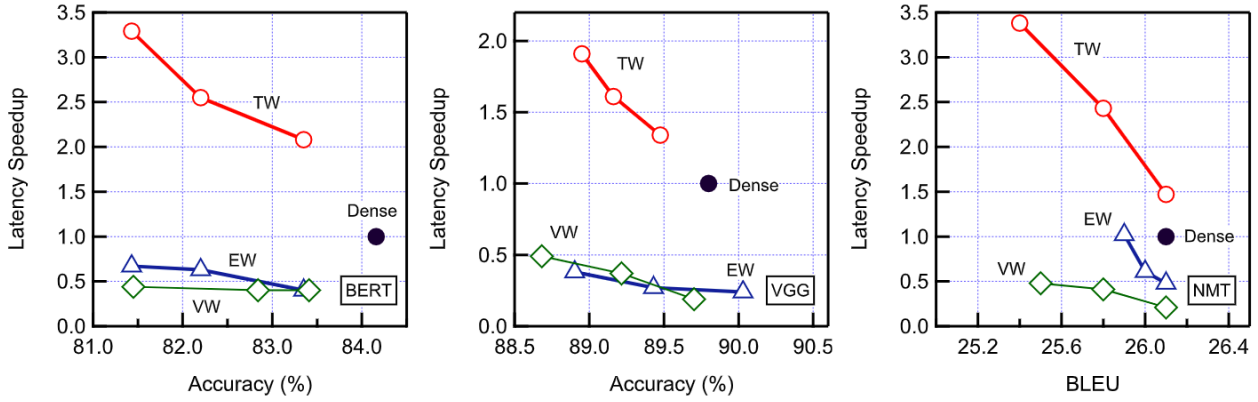
(a) EW (b) VW (c) BW (d) TW

Irregularity: EW > TW > VW, BW

Speedup on GEMM



(a) TW vs BW on tensor core.



(b) TW vs VW and EW on CUDA core.

BERT accuracy loss < 3%
VGG accuracy loss < 1%
NMT BLEU loss < 1

Tensor cores: TW 1.95× speedup
CUDA cores: TW 2.86× speedup

TW achieves the meaningful latency reduction on both tensor cores and CUDA cores owing to its compatibility with dense GEMM, while all other sparsity patterns cause the actual slowdown.

End-to-end Latency and Impact of Optimizations

Time (ms)	Dense	W/o Transpose	Explicit Transpose	Fused Transpose
GEMM	32.38 (71%)	37.6	14.29	14.29 (51%)
non-GEMM	12.99	12.99	12.99	13.93
Transpose	0	0	5.18	0
Total	45.37	50.59	32.46	28.22
Speedup	1	0.9	1.4	1.61

GEMM: 2.26x than Dense and 2.63x than w/o Transpose

Fusion Transpose: 2% overhead

Without Fusion Transpose: ~10% overhead

End-to-end: 1.61x speedup

BERT-Base model with 75% sparsity on tensor core

Without transpose: Performance degradation.
With explicit transpose: 10% overhead. -- Optimization 1
With fusion transpose: 2% overhead. -- Optimization 2
End-to-end speedup: 1.61x.

Conclusion

- TW achieves the meaningful speedup on both tensor cores (1.95×) and CUDA cores(2.86×) with a high model accuracy, while all other sparsity patterns cause the actual slowdown.
- The tiling GEMM algorithm is widely used in the dense GEMM-based accelerators. In other words, supporting TW on other platforms like TPU is feasible.
- Proposed a new DNN model sparsity design insight based on the Tile-Wise algorithm-software optimization.
- Tile Sparsity is open source!

https://github.com/clevercool/TileSparsity

Thanks !

Questions?