Tiled SpMM and its Performance Model on GPUs

Mert Hidayetoglu

University of Illinois at Urbana-Champaign

Organizers: Mert Hidayetoglu, Wen-mei Hwu, Jinjun Xiong, Rakesh Nagi,

Jeff Pool, Sitao Huang, Vikram Mailthody

Tutorial | MLSys 2022 | Aug 30, 2022

https://mlsys.org/virtual/2022/tutorial/2199

Slack for discussions: https://bit.ly/3KLQEUN

Outline

1. Opening remarks and overview of sparsity in ML – 1 pm
2. Tiled SpMM and its performance model on GPUs – 1:50 pm
3. Sparse deep neural network inference on FPGAs – 2:45 pm
4. Break (30 mins)
5. 2:4 Sparsity on GPU Tensor Cores – 4 pm
6. Future work and closing remarks – 4:45 pm

Fundamental Matrix Multiplication Operations

Fundamental Matrix Multiplication Operations

DGEMM

$$A \times B = C$$

Dense Dense Dense

Fundamental Matrix Multiplication Operations

DGEMM

$$A \times B = C$$

Dense Dense Dense

SpMM

$$A \times B = C$$

Sparse Dense Dense

Fundamental Matrix Multiplication Operations

DGEMM

$$A \times B = C$$

Dense Dense Dense

SpMM

$$A \times B = C$$

Sparse Dense Dense

SpGEMM

$$A \times B = C$$

Sparse Sparse Sparse

Fundamental Matrix Multiplication Operations

DGEMM

$$A \times B = C$$

Dense Dense Dense

SpMM

$$A \times B = C$$

Sparse Dense Dense

SpGEMM

$$A \times B = C$$

Sparse Sparse Sparse

Fundamental Matrix Multiplication Operations

DGEMM

$$A \times B = C$$

Dense Dense Dense

SpMM

$$A \times B = C$$

Sparse Dense Dense

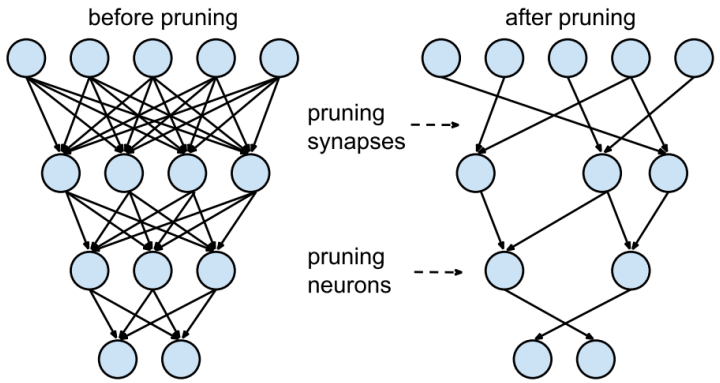
SpGEMM

$$A \times B = C$$

Sparse Sparse Sparse

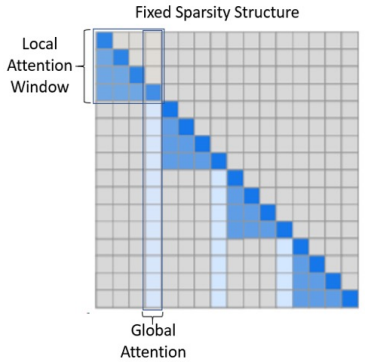
Sparse Connection Layers
Graph Neural Networks
Sparse Attention Layers

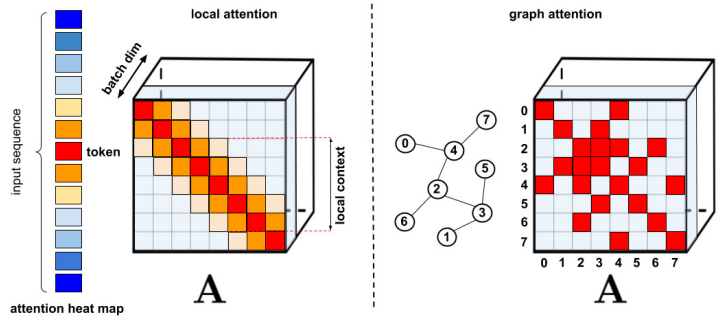
Sparse Neural Networks



[Lecun et al. NIPS'89] [Han et al. NIPS'15]

Sparse Attention Layers



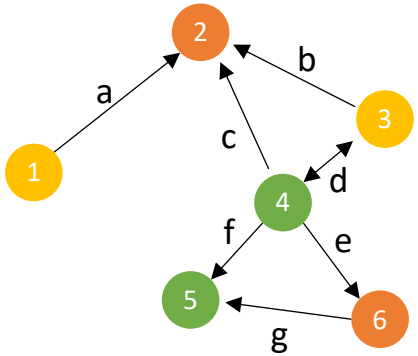


Google ai, "Rethinking Attention with Performers,"

Child, et al., "Generating Long Sequences with Sparse Transformers," Arxiv

Graph Neural Networks

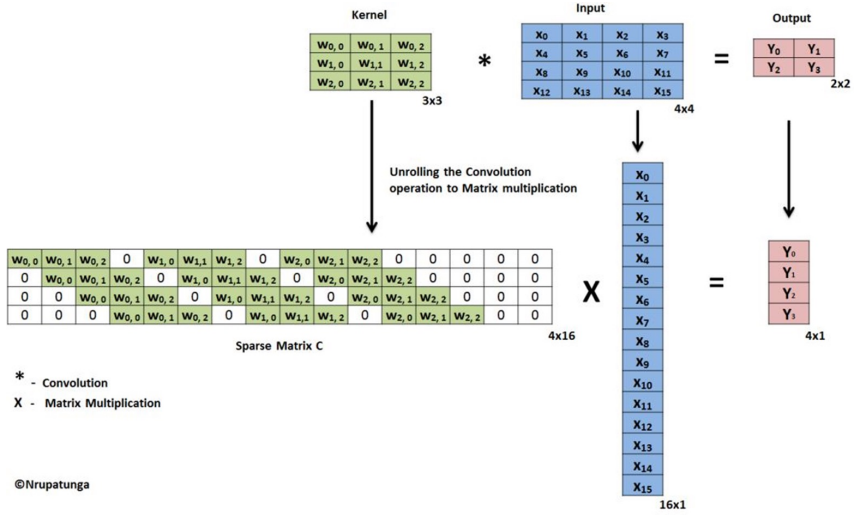
Graph Representation



Matrix Representation

	1	2	3	4	5	6
1		a				
2						
3		b		d		
4		c	d		f	
5						e
6					g	

Convolutional Neural Networks



Sparsity Level ↑

Memory Footprint ↓ 👍

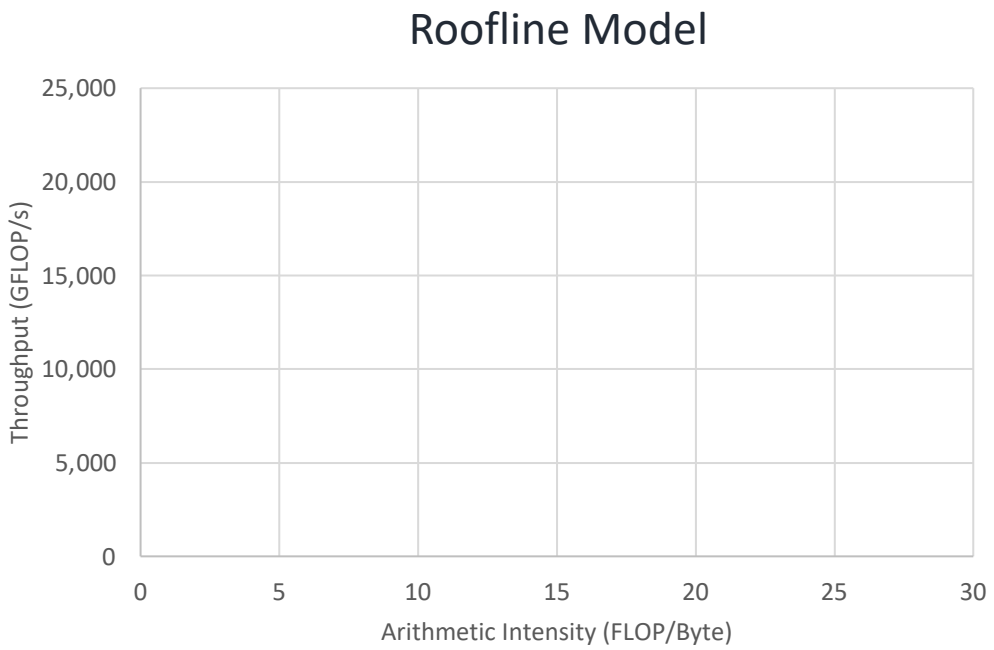
Number of Ops ↓ 👍

Accuracy ↓ 👎

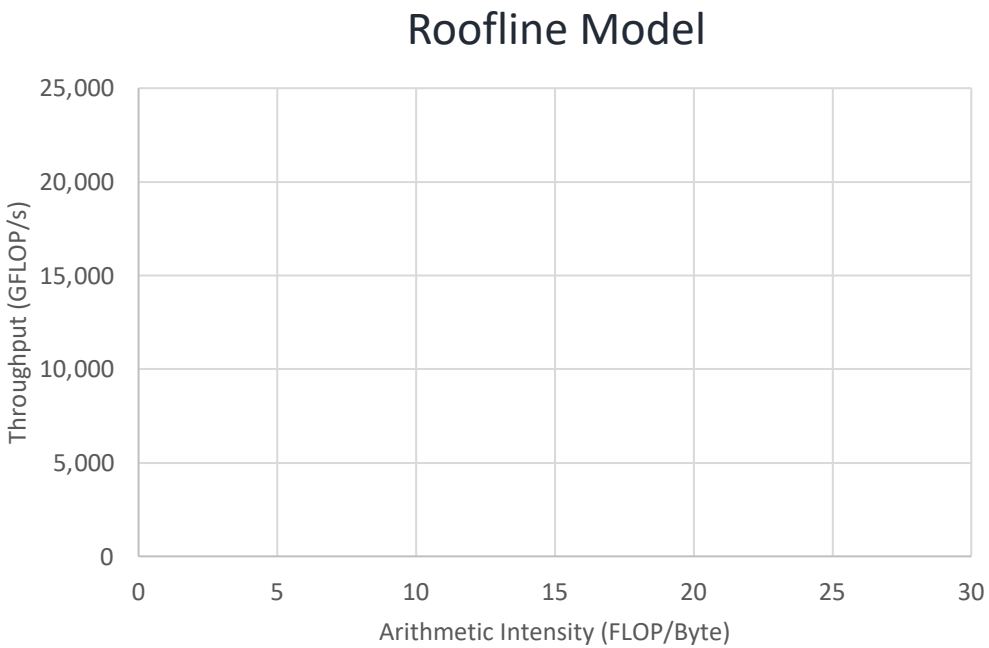
Throughput ?

The Roofline Model

The Roofline Model

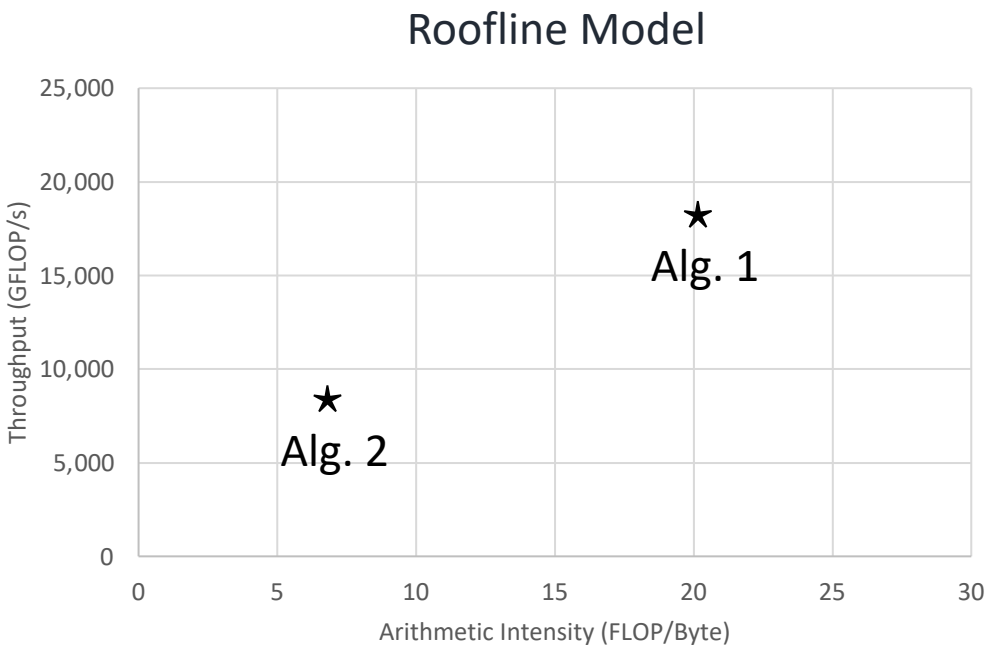


The Roofline Model



Arithmetic Intensity

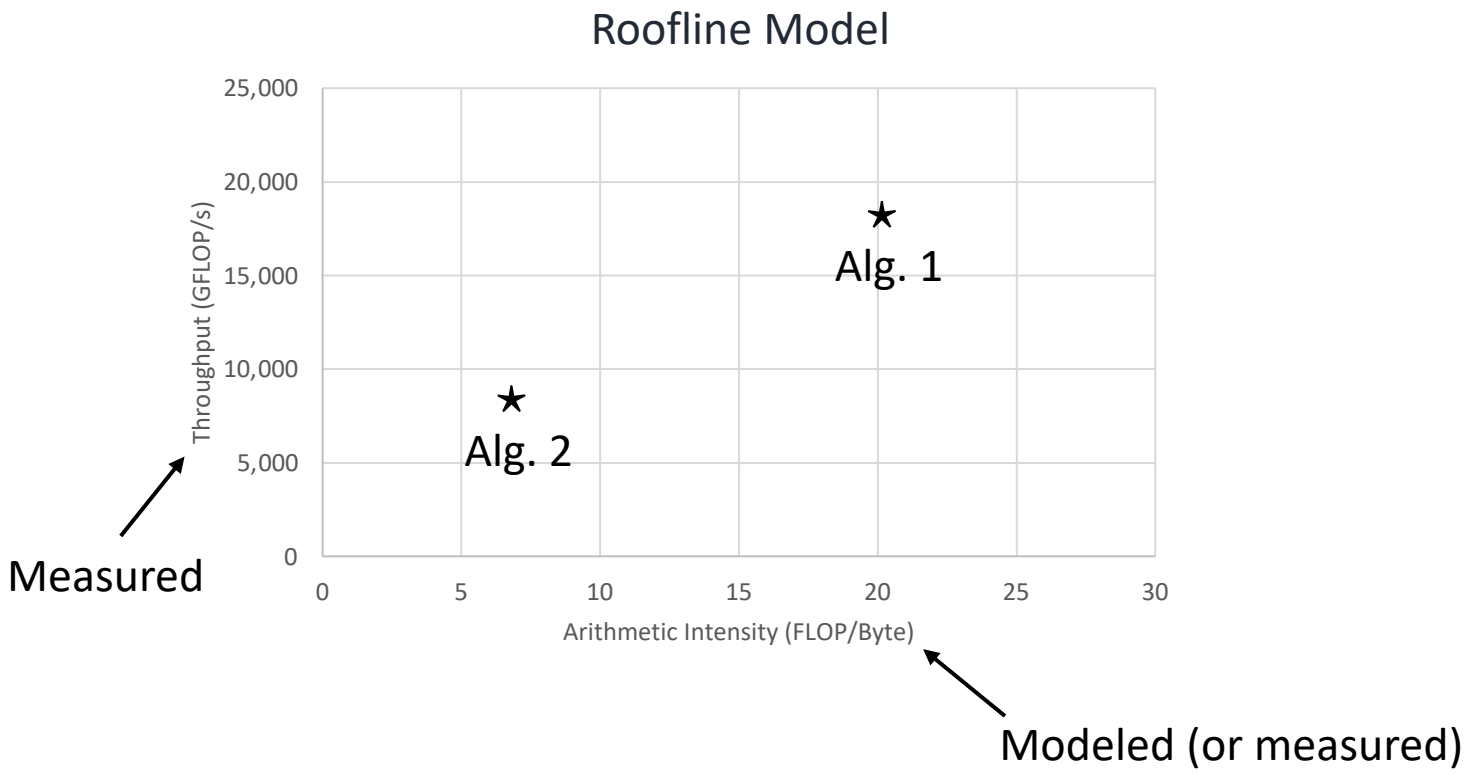
$$I = \frac{\sum \text{Floating Point Operations (GFLOP)}}{\sum \text{Memory Accesses (GB)}}$$



Arithmetic Intensity

$$I = \frac{\sum \text{Floating Point Operations (GFLOP)}}{\sum \text{Memory Accesses (GB)}}$$

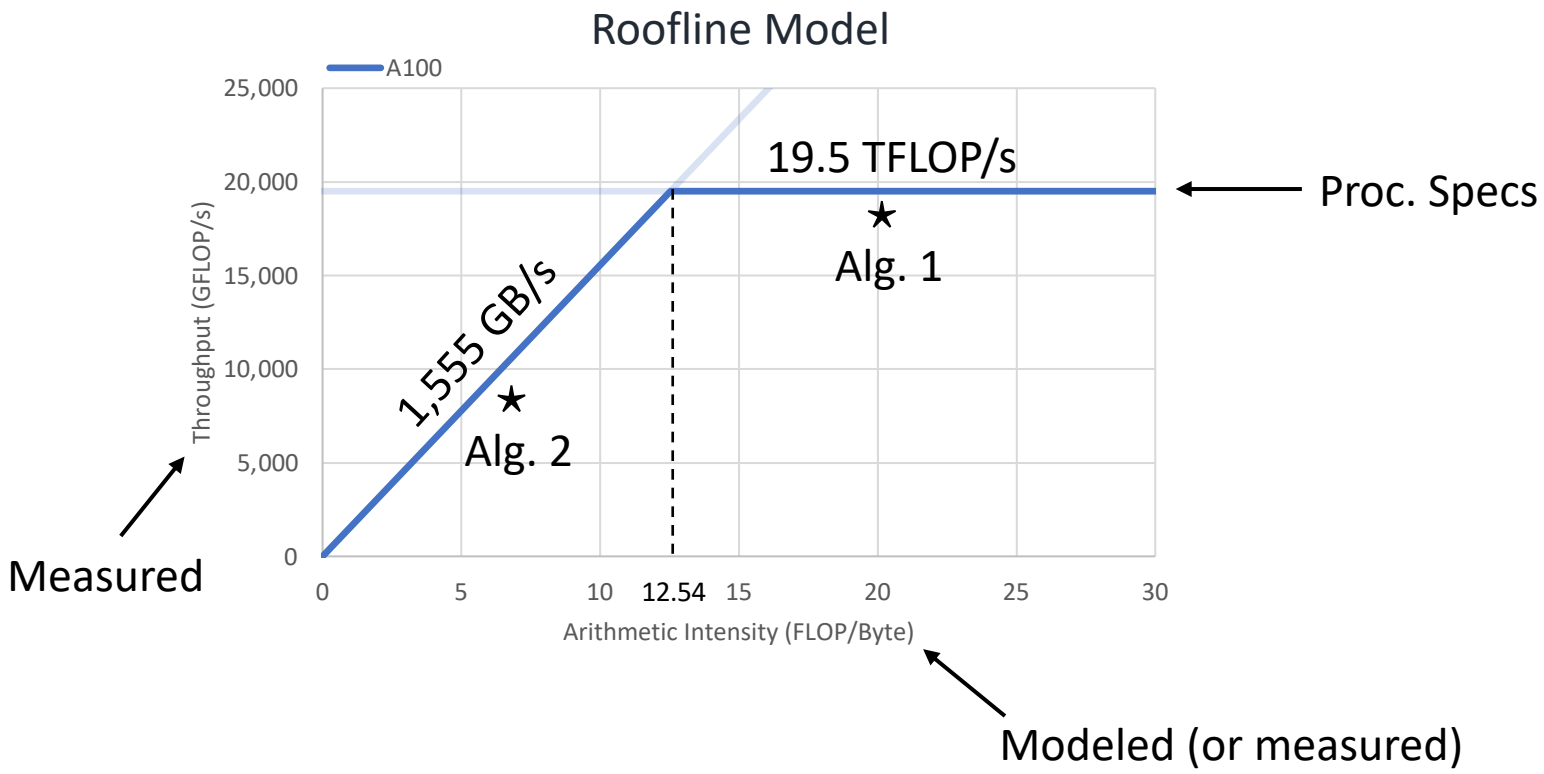
The Roofline Model



Arithmetic Intensity

$$I = \frac{\sum \text{Floating Point Operations (GFLOP)}}{\sum \text{Memory Accesses (GB)}}$$

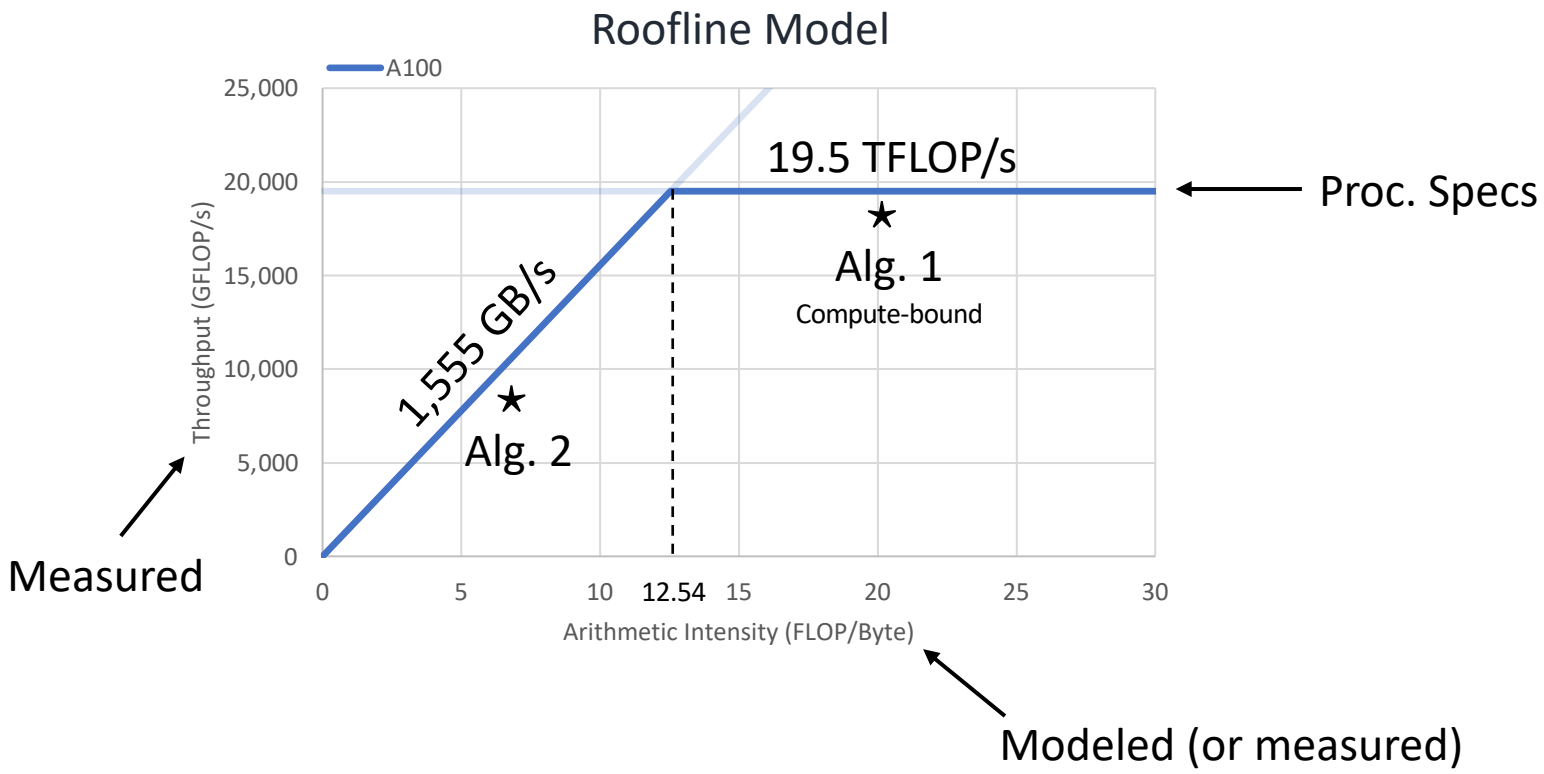
The Roofline Model



Arithmetic Intensity

$$I = \frac{\sum \text{Floating Point Operations (GFLOP)}}{\sum \text{Memory Accesses (GB)}}$$

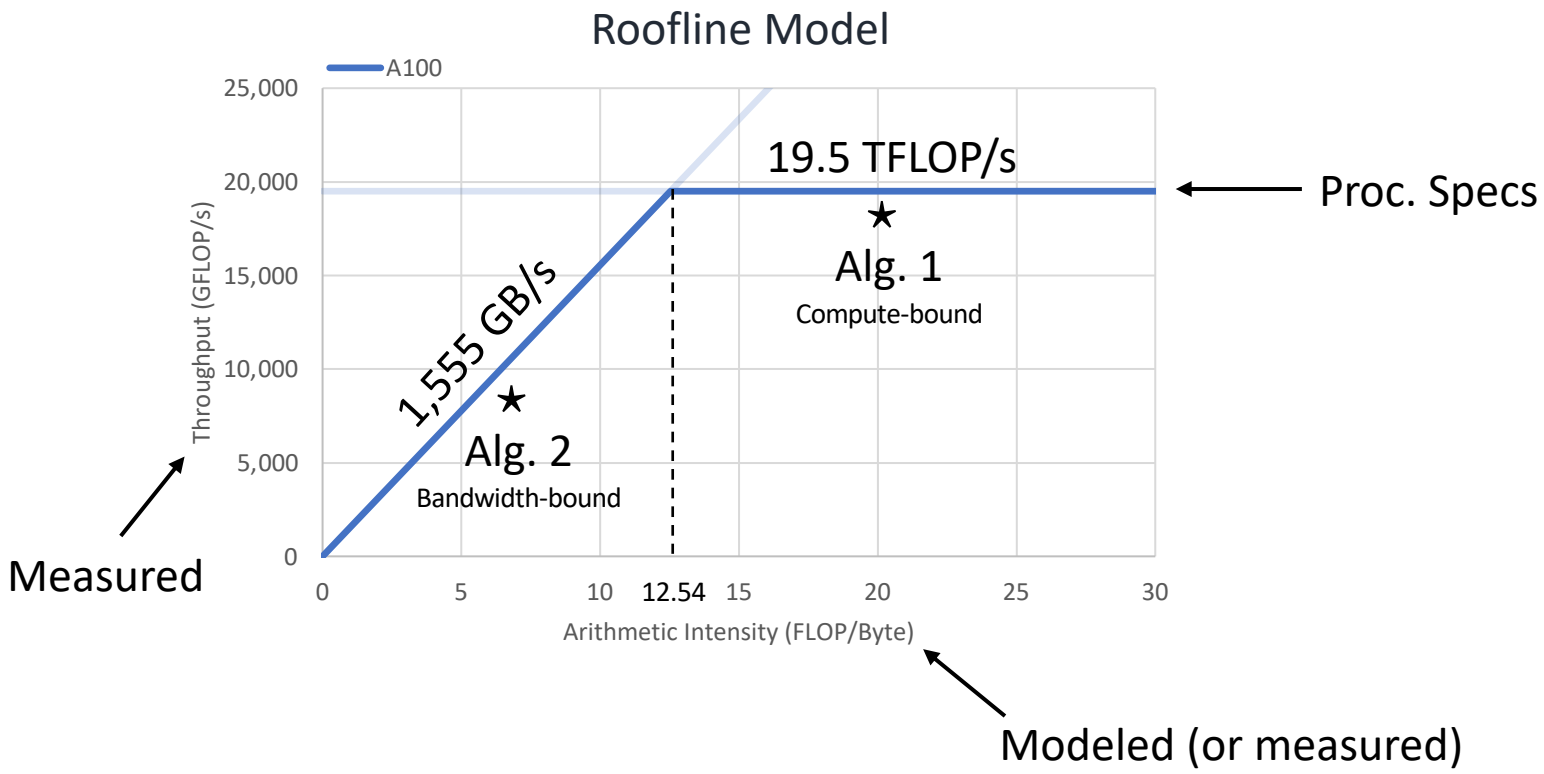
The Roofline Model



Arithmetic Intensity

$$I = \frac{\sum \text{Floating Point Operations (GFLOP)}}{\sum \text{Memory Accesses (GB)}}$$

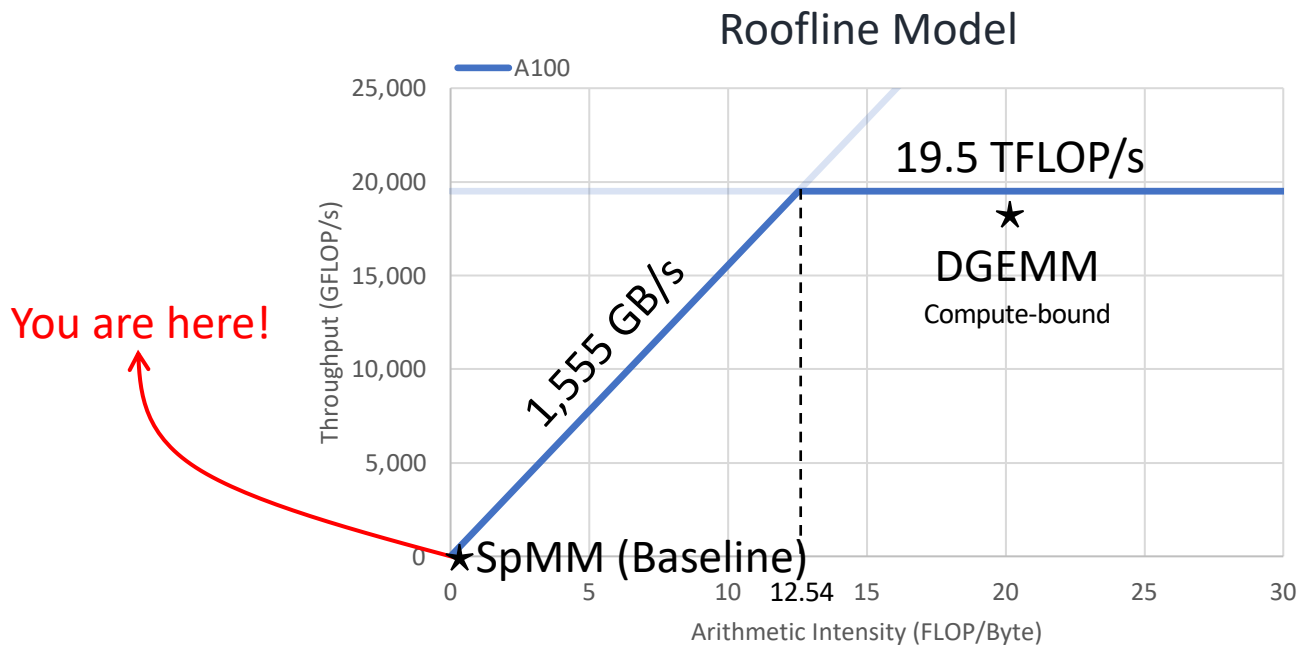
The Roofline Model



Arithmetic Intensity

$$I = \frac{\sum \text{Floating Point Operations (GFLOP)}}{\sum \text{Memory Accesses (GB)}}$$

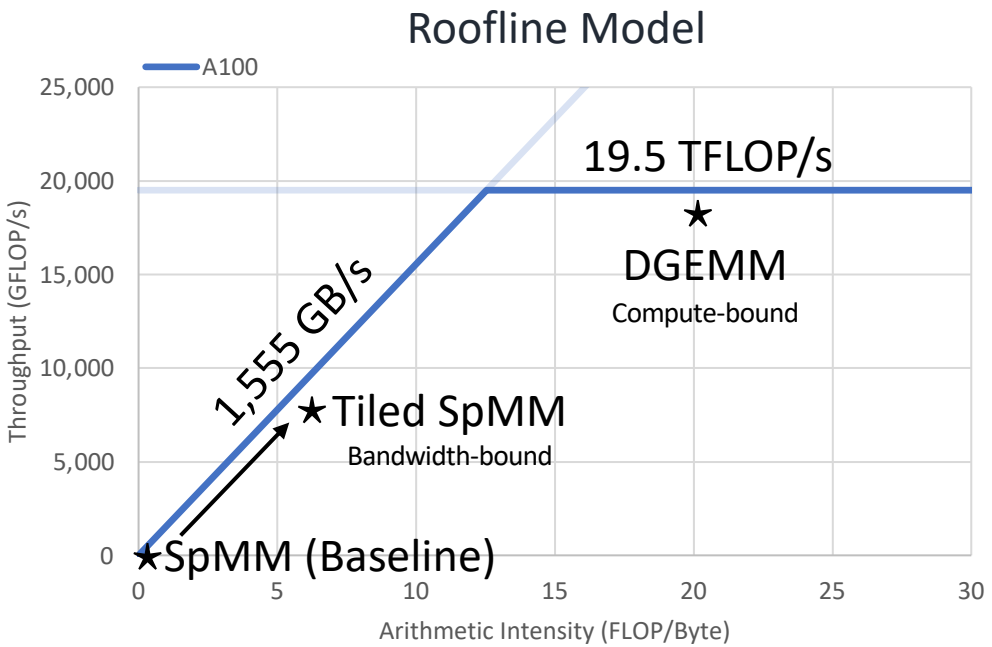
The Roofline Model



Arithmetic Intensity

$$I = 0.17 \text{ FLOP/Byte}$$

The Roofline Model



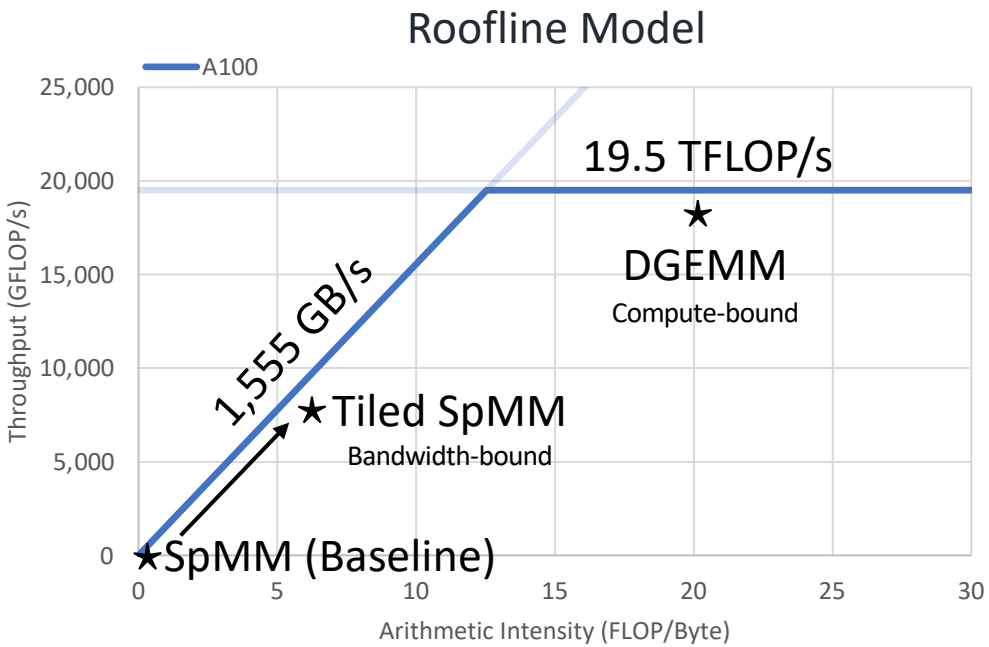
Tiled SpMM

Applies Loop Transformations

Utilizes scratchpad and registers

Improves arithmetic intensity

The Roofline Model



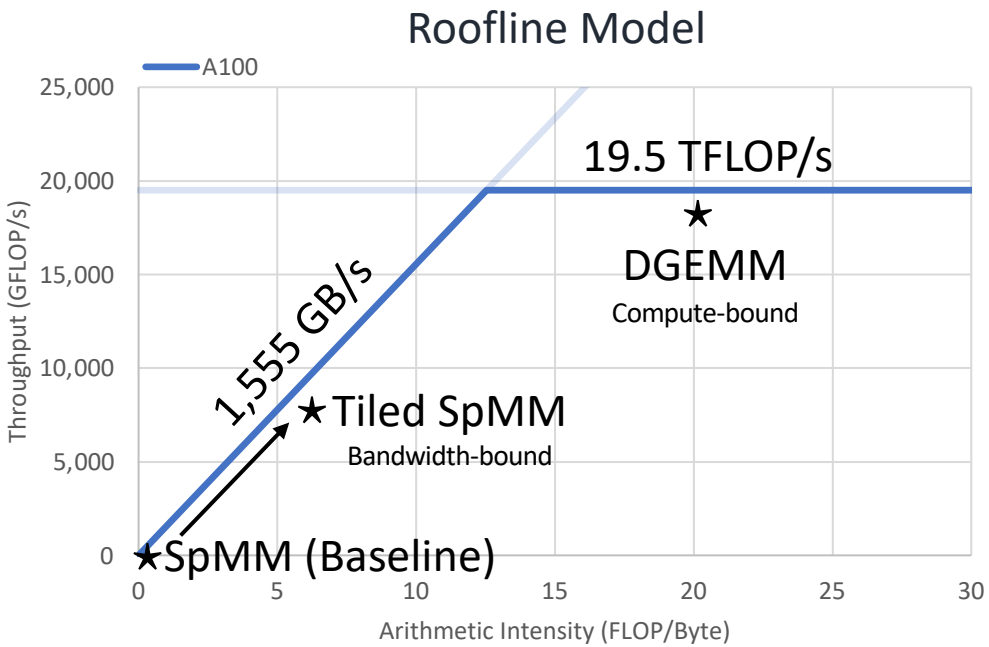
Tiled SpMM

Applies Loop Transformations { Reordering, Tiling, Fusion }

Utilizes scratchpad and registers

Improves arithmetic intensity

The Roofline Model



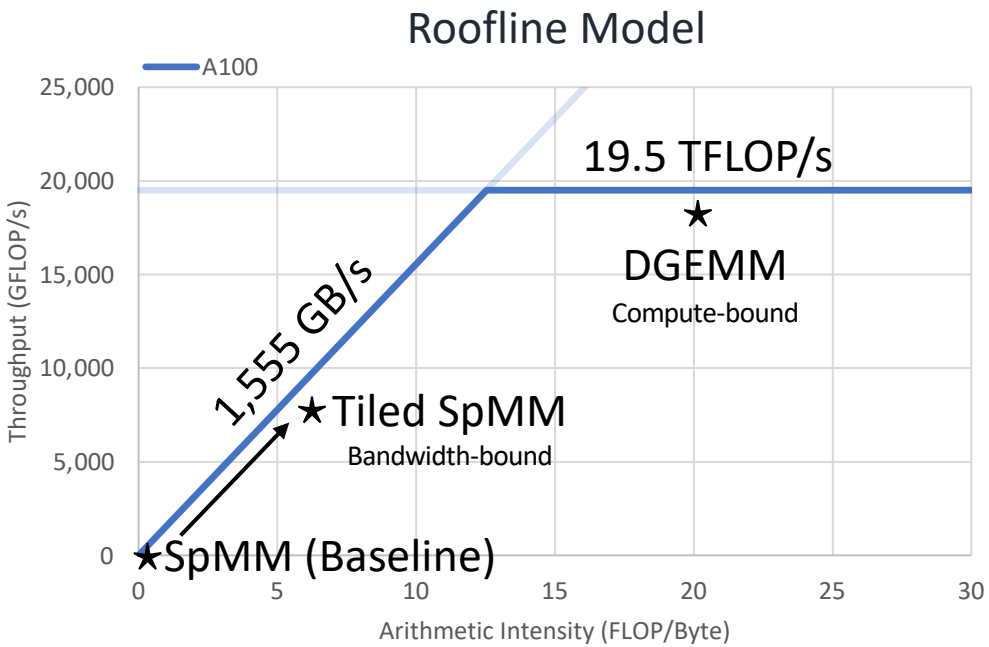
Tiled SpMM

Applies Loop Transformations

Utilizes scratchpad and registers
- Provide on-chip data reuse
- Reduce DRAM memory traffic

Improves arithmetic intensity

The Roofline Model



Tiled SpMM

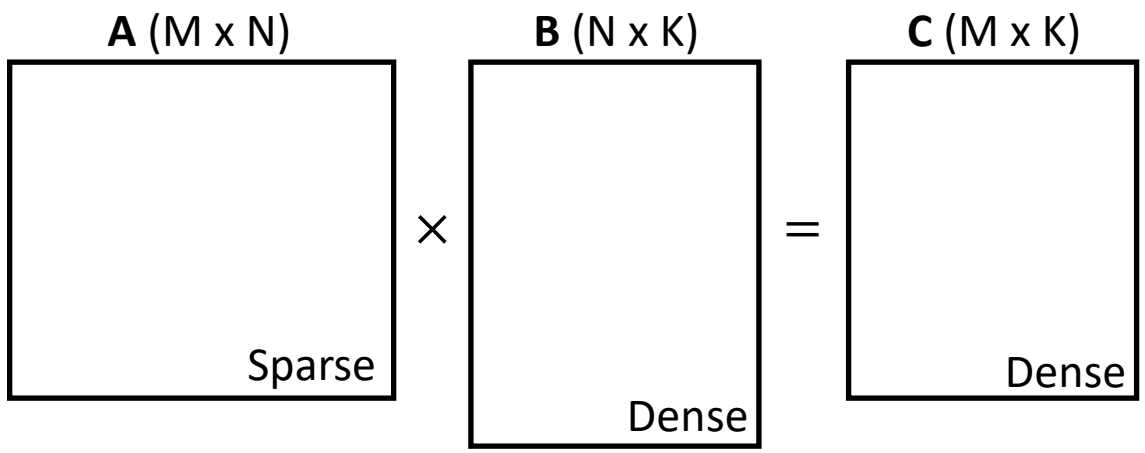
Applies Loop Transformations

Utilizes scratchpad and registers

Improves arithmetic intensity

Modeling Arithmetic Intensity of Baseline SpMM

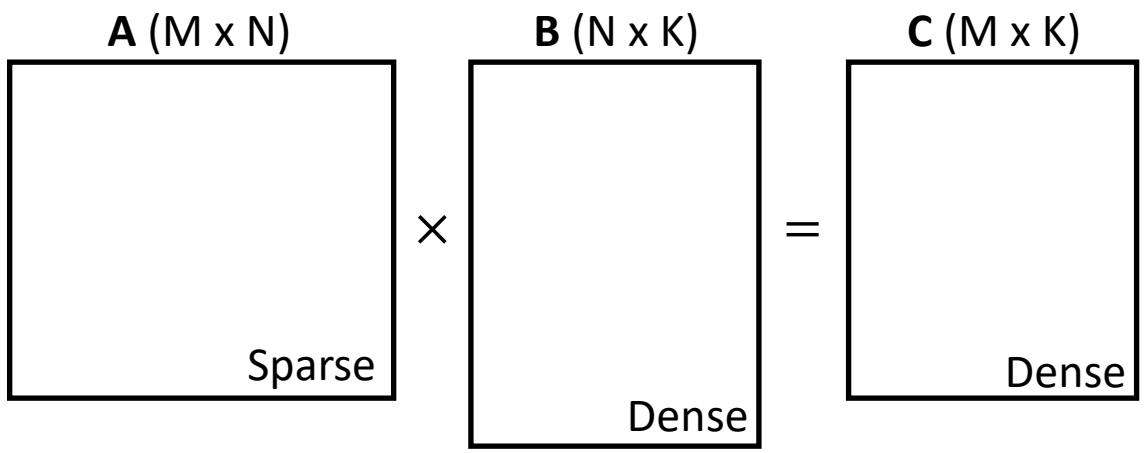
Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)        ← For K columns
  for (int m = 0; m < M; m++)      ← For M Rows
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

Arithmetic Intensity: Algorithmic Model

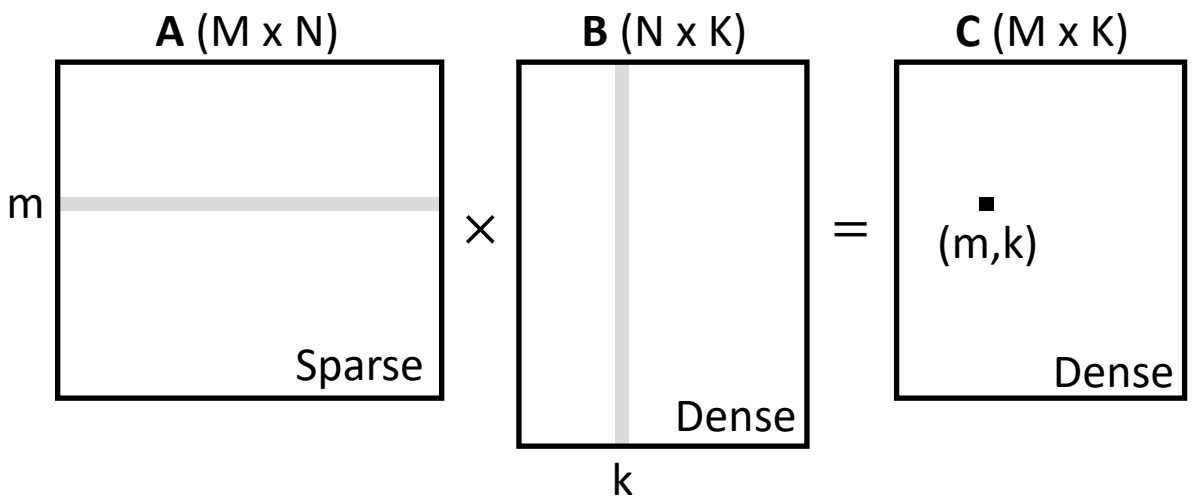


CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

} M x K Threads

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

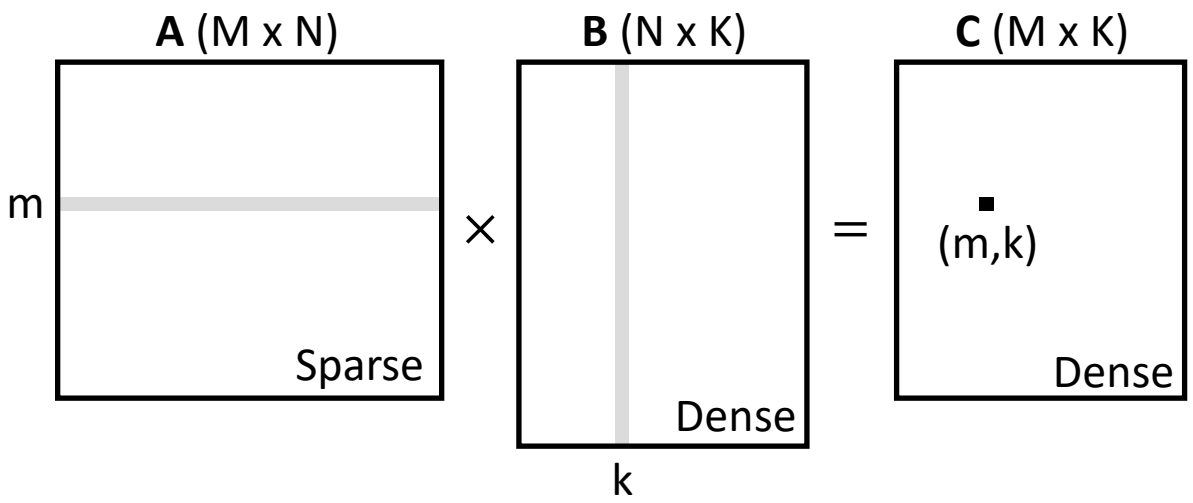
```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
```

} M x K Threads

Each Thread

```
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
```

} M x K Threads

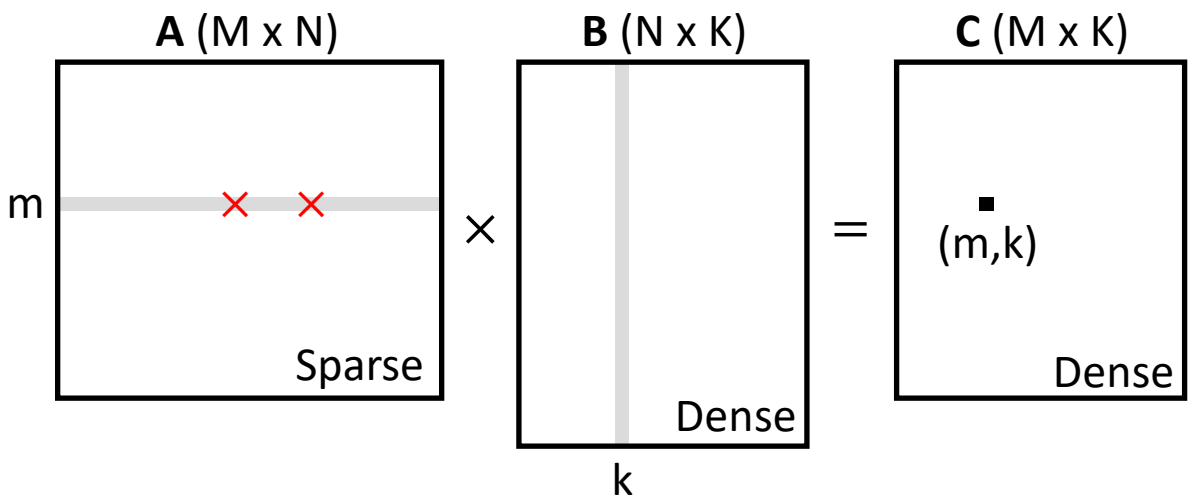
Each Thread

```
{
  float acc = 0;
  for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
  {
    int idx = index[l];
    float val = value[l];
    acc += B[idx][k] * val;
  }
  C[m][k] = acc;
}
```

$$I = \frac{2\mu}{(B_T + B_I)\mu + B_T\mu + B_T + \delta}$$

Algorithmic Intensity

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

M x K Threads

Each Thread

μ times

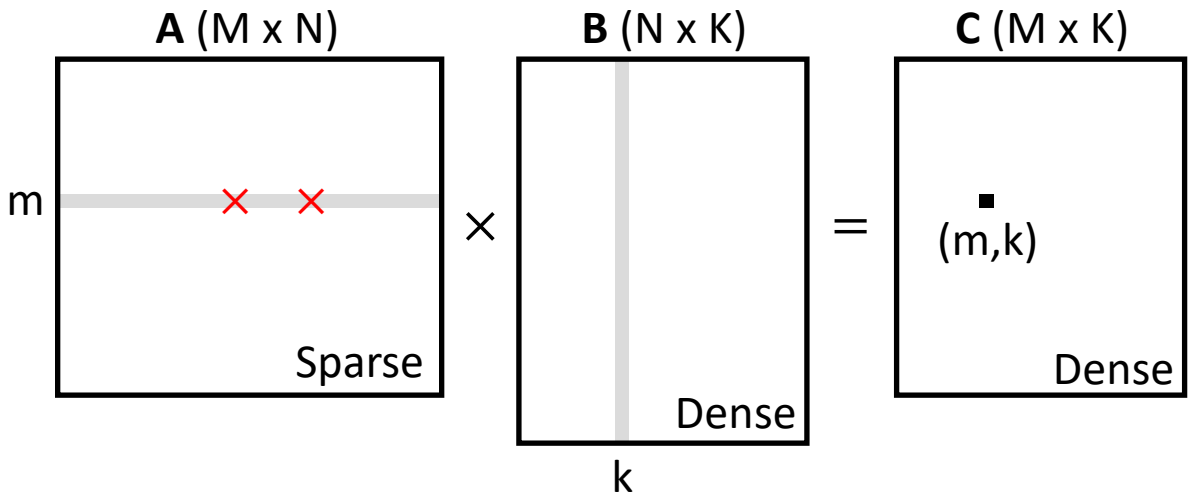
Legend

μ: # of nonzeroes per row

$$I = \frac{2\mu}{(B_T + B_I)\mu + B_T\mu + B_T + \delta}$$

Algorithmic Intensity

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

M x K Threads

Each Thread

μ times

→ Fused Multiply-Add (FMA): 2 FLOPs

FLOPs per nnz (2μ)

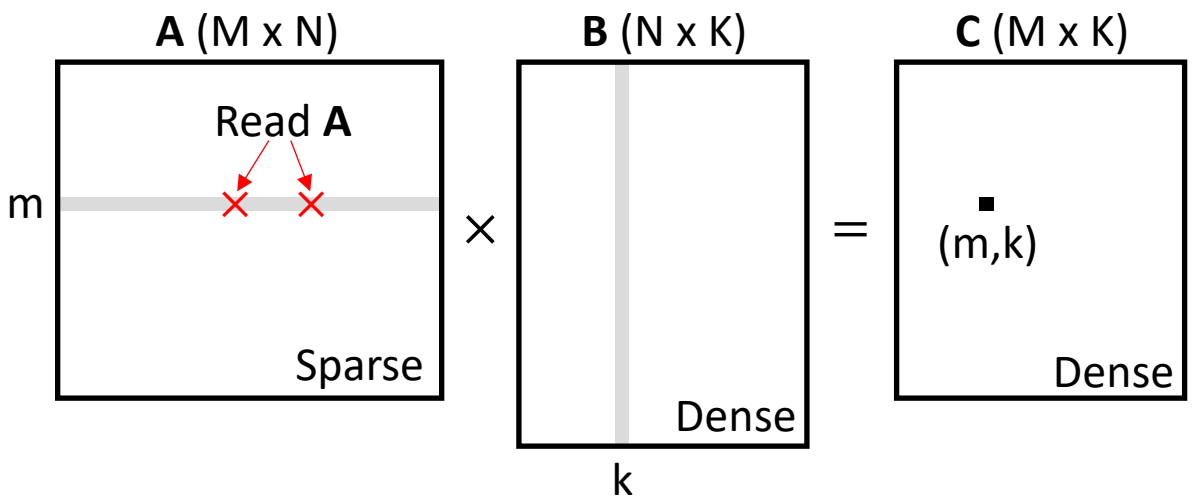
$$I = \frac{2\mu}{(B_T + B_I)\mu + B_T\mu + B_T + \delta}$$

Algorithmic Intensity

Legend

μ: # of nonzeroes per row

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

M x K Threads

Each Thread

μ times

Read **A**

→ Fused Multiply-Add (FMA): 2 FLOPs

$$I = \frac{2\mu}{(B_T + B_I)\mu + B_T\mu + B_T + \delta}$$

FLOPs per nnz

Read **A**

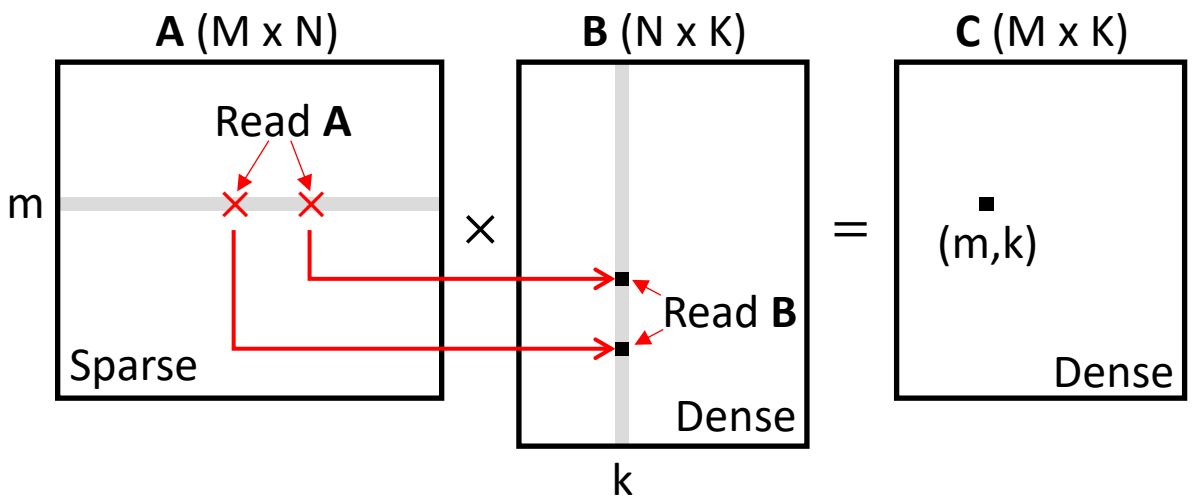
Legend

μ: # of nonzeroes per row

B_T: Bytes per value

B_I: Bytes per index

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

M x K Threads

Each Thread

μ times

Read **A**

Read **B**

→ Fused Multiply-Add (FMA): 2 FLOPs

$$I = \frac{2\mu}{\underbrace{(B_T + B_I)\mu}_{\text{Read A}} + \underbrace{B_T\mu}_{\text{Read B}} + B_T + \delta}$$

FLOPs per nnz

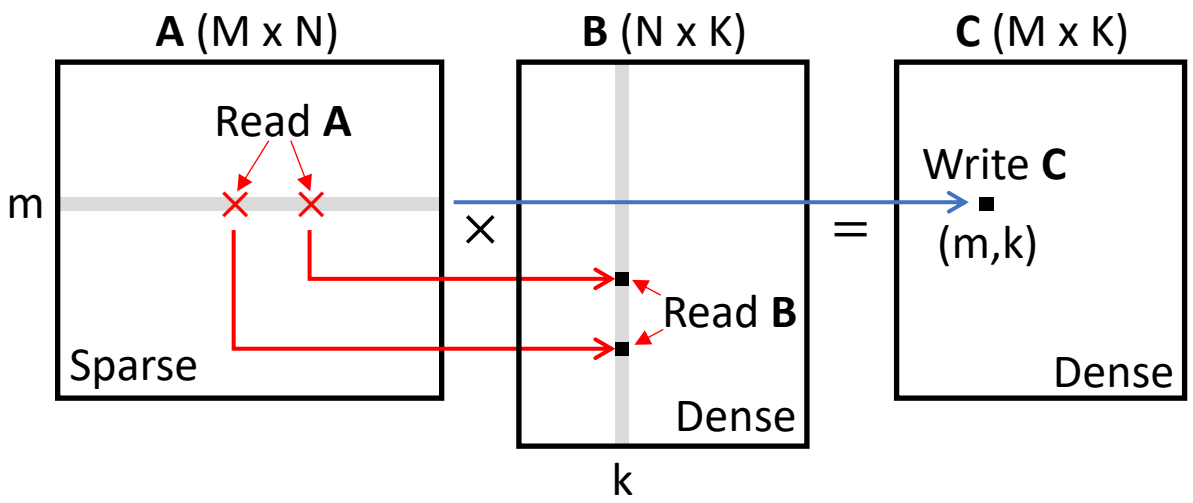
Legend

μ: # of nonzeroes per row

B_T: Bytes per value

B_I: Bytes per index

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)                    } M x K Threads
  for (int m = 0; m < M; m++)
  {                                                Each Thread
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];                        } Read A
      float val = value[l];
      acc += B[idx][k] * val;                    → Fused Multiply-Add (FMA): 2 FLOPs
    }                  Read B
    C[m][k] = acc;
  }          Write C
```

μ times

$$I = \frac{2\mu}{(B_T + B_I)\mu + B_T\mu + B_T + \delta}$$

FLOPs per nnz

$(B_T + B_I)\mu$: Read **A**; $B_T\mu$: Read **B**; B_T: Write **C**

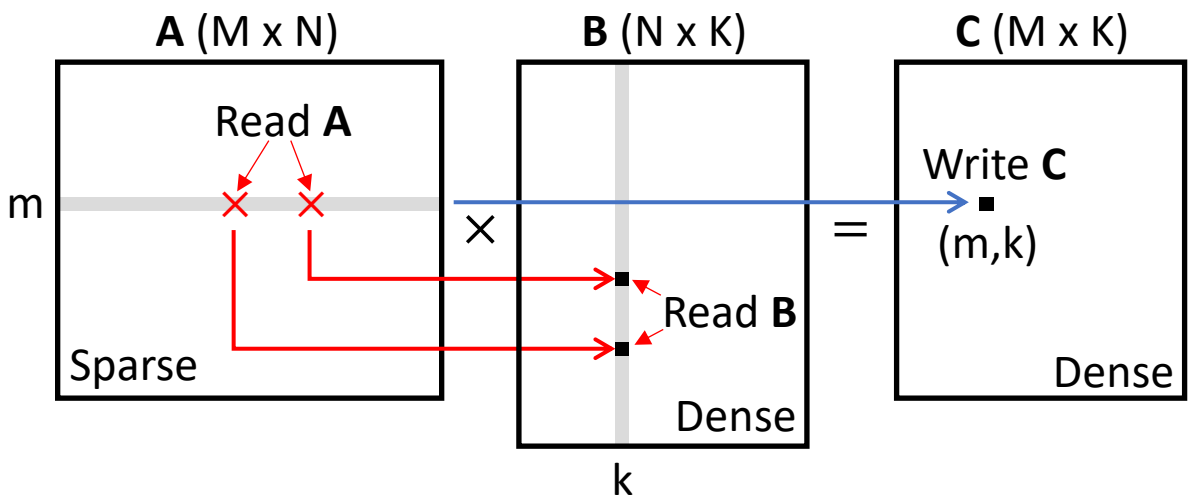
Legend

μ: # of nonzeroes per row

B_T: Bytes per value

B_I: Bytes per index

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

M x K Threads

Each Thread

Metadata

μ times

Read A

Read B

→ Fused Multiply-Add (FMA): 2 FLOPs

Write C

$$I = \frac{2\mu}{\underbrace{(B_T + B_I)\mu}_{\text{Read A}} + \underbrace{B_T\mu}_{\text{Read B}} + \underbrace{B_T}_{\text{Write C}} + \delta}$$

FLOPs per nnz

Read Metadata

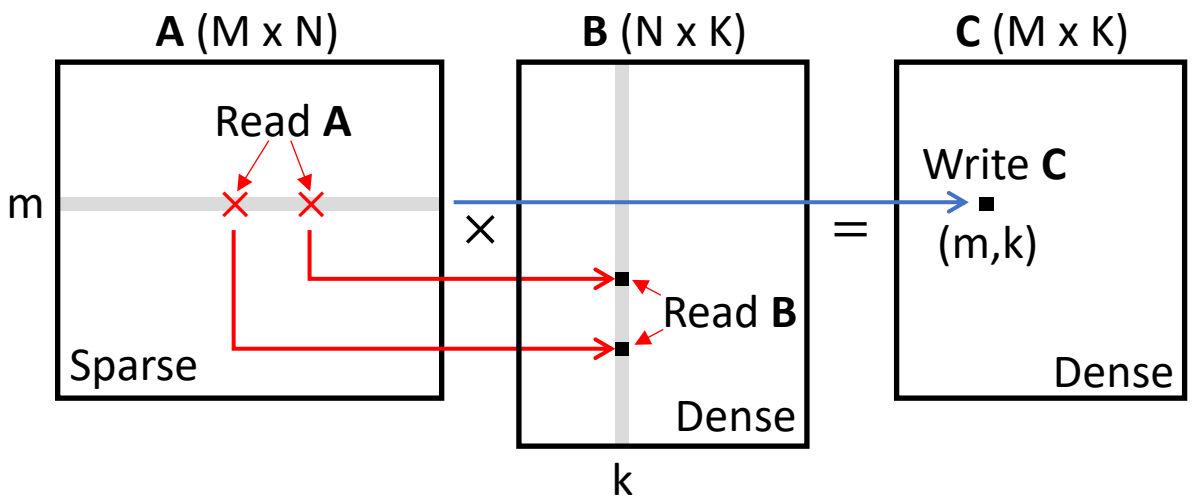
Legend

μ: # of nonzeroes per row

B_T: Bytes per value

B_I: Bytes per index

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

M x K Threads

Each Thread

Metadata

μ times

Read **A**

Read **B**

Fused Multiply-Add (FMA): 2 FLOPs

Write **C**

$$I = \frac{2\mu}{(B_T + B_I)\mu + B_T\mu + B_T + \delta}$$

FLOPs per nnz

FLOPs / byte

Read **A**: $(B_T + B_I)\mu$

Read **B**: $B_T\mu$

Write **C**: B_T

Read Metadata: δ

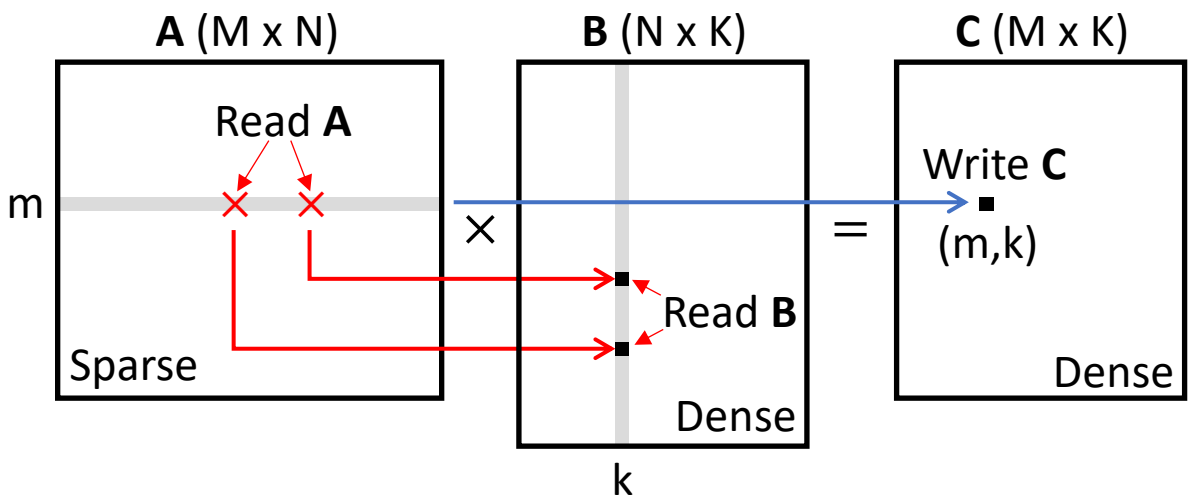
Legend

μ: # of nonzeroes per row

B_T: Bytes per value

B_I: Bytes per index

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
  {
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];
      float val = value[l];
      acc += B[idx][k] * val;
    }
    C[m][k] = acc;
  }
```

M x K Threads

Each Thread

Metadata

μ times

Read **A**

Read **B**

Fused Multiply-Add (FMA): 2 FLOPs

Write **C**

$$I\beta = \frac{2\beta\mu}{\underbrace{(B_T + B_I)\mu}_{\text{Read A}} + \underbrace{B_T\mu}_{\text{Read B}} + \underbrace{B_T}_{\text{Write C}} + \underbrace{\delta}_{\text{Read Metadata}}}$$

FLOPs per nnz

1,555 GB/s

$I\beta$: $\frac{\text{GFLOPs}}{\text{second}}$

Legend

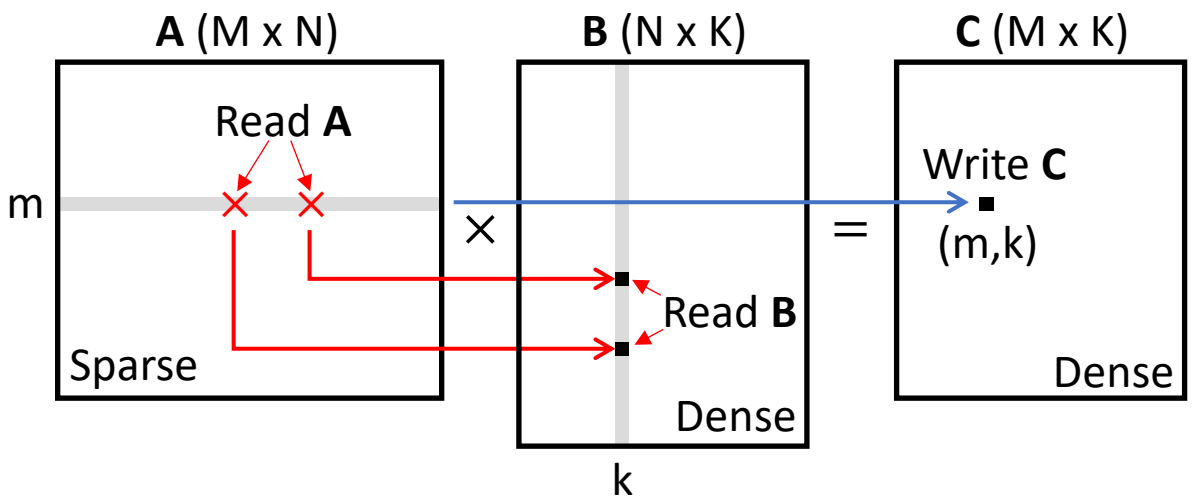
μ: # of nonzeroes per row

B_T: Bytes per value

B_I: Bytes per index

β: Memory B/W

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)                    } M x K Threads
  for (int m = 0; m < M; m++)
  {                                                Each Thread
    float acc = 0;
    for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
    {
      int idx = index[l];                      }
      float val = value[l];                    } Read A
      acc += B[idx][k] * val;
    }                  Read B
    C[m][k] = acc;
  }          Write C
```

μ times

$$\underbrace{\text{130 GFLOPS}}_{\mu = 1} < I\beta = \frac{2\beta\mu}{(B_T + B_I)\mu + B_T\mu + B_T + \delta} < \underbrace{\text{260 GFLOPS}}_{\mu \gg B_T}$$

1,555 GB/s

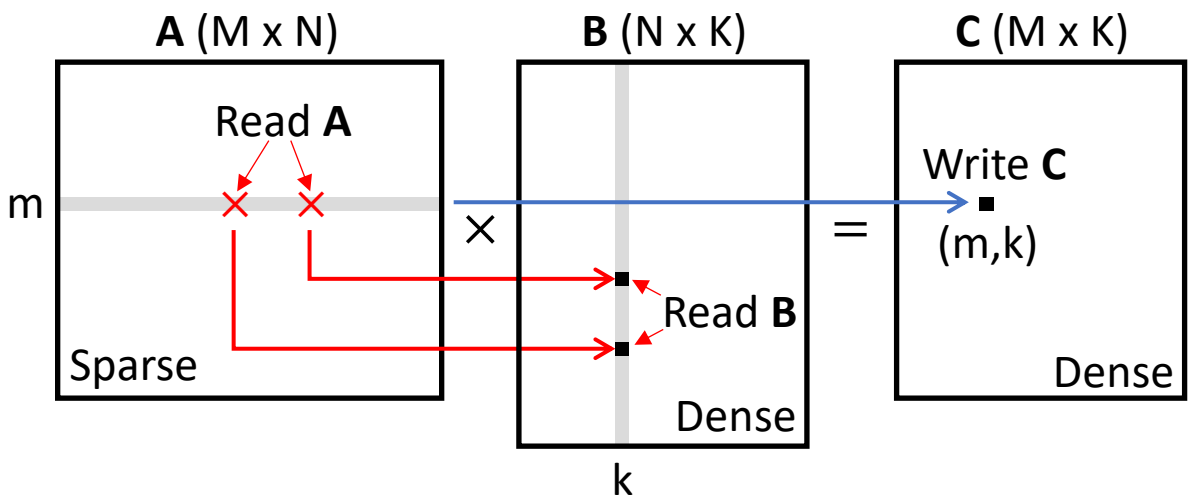
more sparsity
e.g, diagonal matrix

Four bytes

Eight bytes

less sparsity

Arithmetic Intensity: Algorithmic Model



CSR SpMM (Baseline)

```
for (int k = 0; k < K; k++)
  for (int m = 0; m < M; m++)
```
} M x K Threads

Each Thread

```
{
  float acc = 0;
  for (int l = rowPtr[m]; l < rowPtr[m + 1]; l++)
  {
    int idx = index[l];
    float val = value[l];
    acc += B[idx][k] * val;
  }
  C[m][k] = acc;
}
```

μ times — Read **A**

Read **B**

Write **C**

1.3% of the peak compute throughput!

$$\underbrace{130\ \text{GFLOPS}}_{\mu = 1\ \text{more sparsity, e.g, diagonal matrix}} < I\beta = \frac{2\beta\mu}{(B_T + B_I)\mu + B_T\mu + B_T + \delta} < \underbrace{260\ \text{GFLOPS}}_{\mu \gg B_T\ \text{less sparsity}}$$

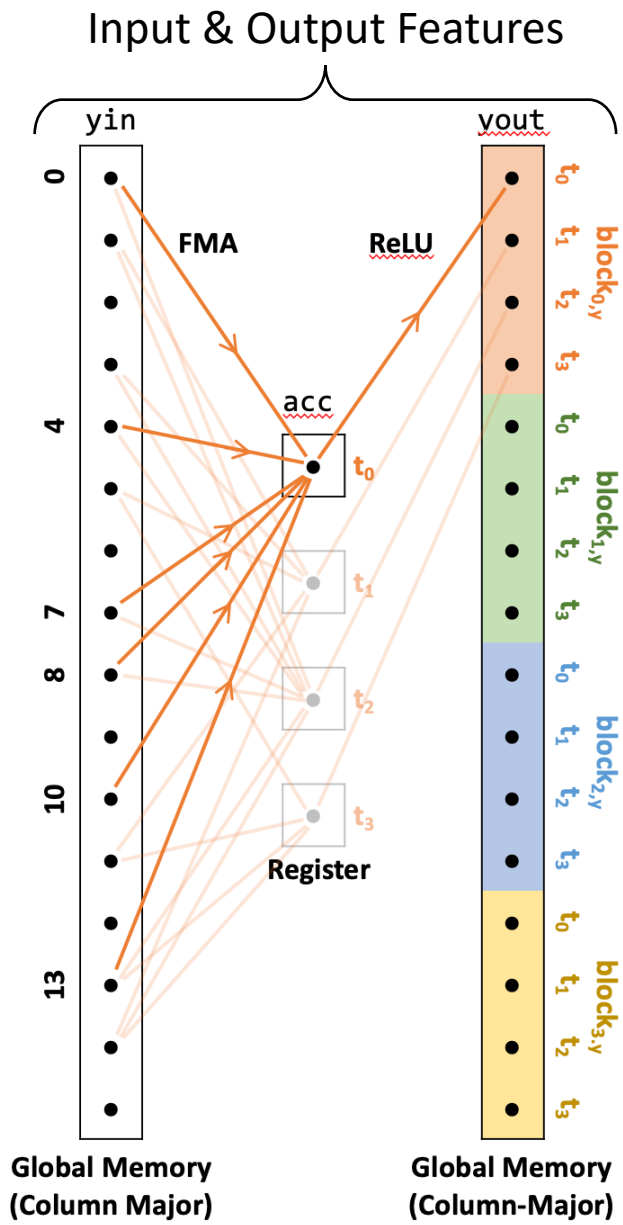
1,555 GB/s

Four bytes

Eight bytes

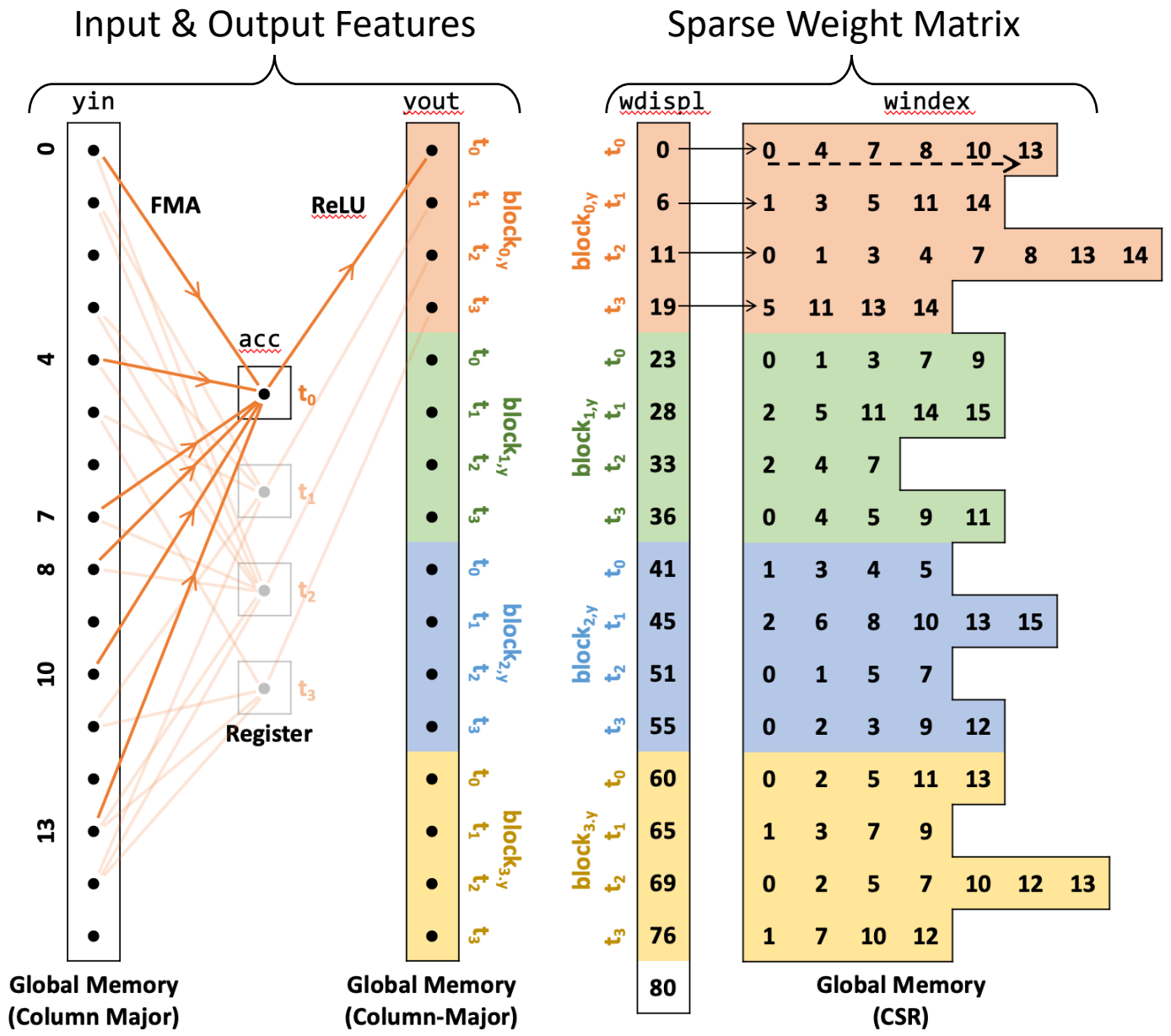
Baseline Sparse Layer Implementation

- Activation features: dense (col.-major)
- Gather approach
- Each thread computes a single output



Baseline Sparse Layer Implementation

- Activation features: dense (col.-major)
- Gather approach
- Each thread computes a single output
- Sparse weights: CSR
- **ReLU is fused with SpMM**

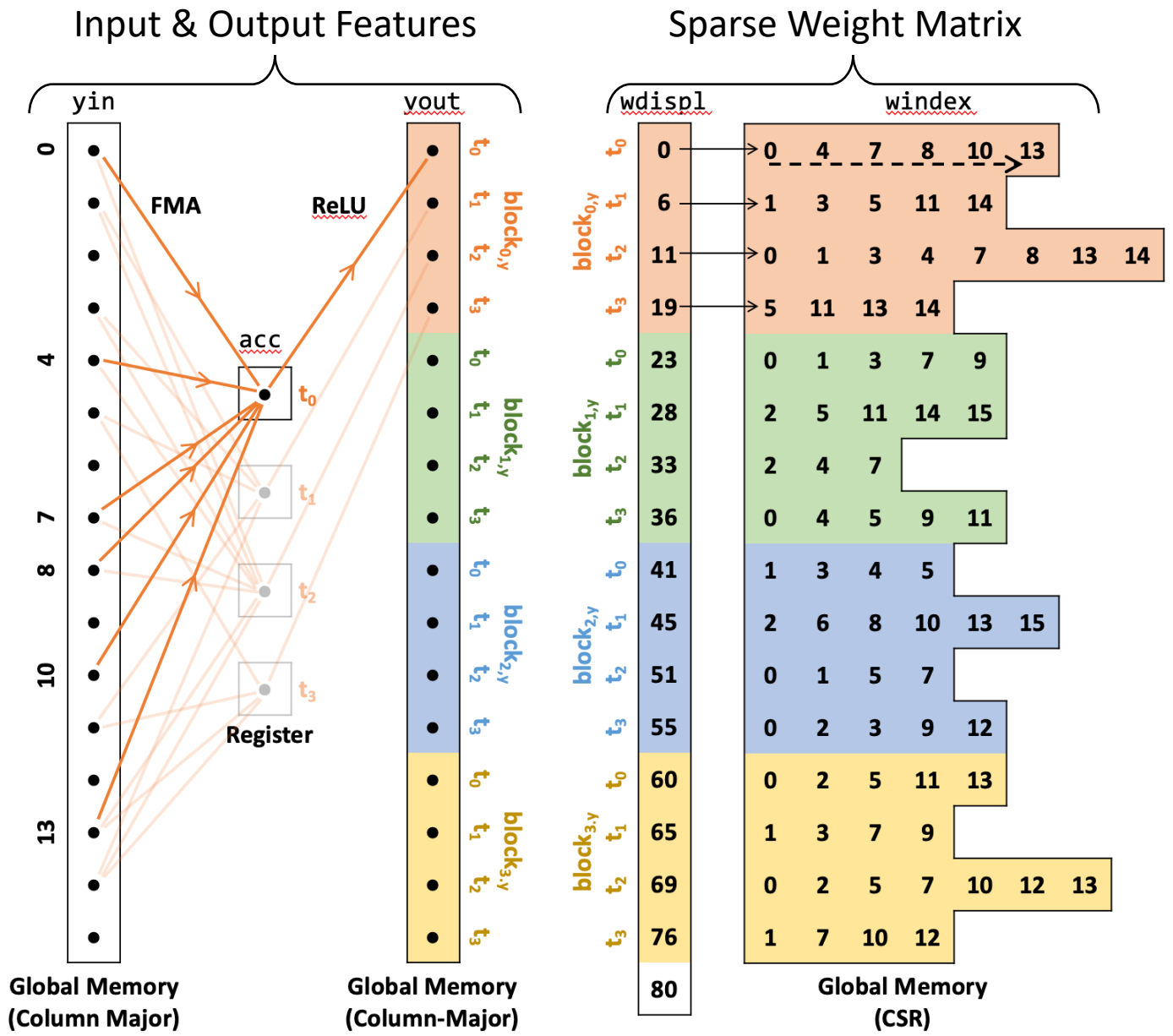


Baseline Sparse Layer Implementation

- Activation features: dense (col.-major)
- Gather approach
- Each thread computes a single output
- Sparse weights: CSR
- **ReLU is fused with SpMM**

Data Access Redundancies

- Weight matrix by all output features
- Input features by different threads



Baseline Sparse Layer Implementation

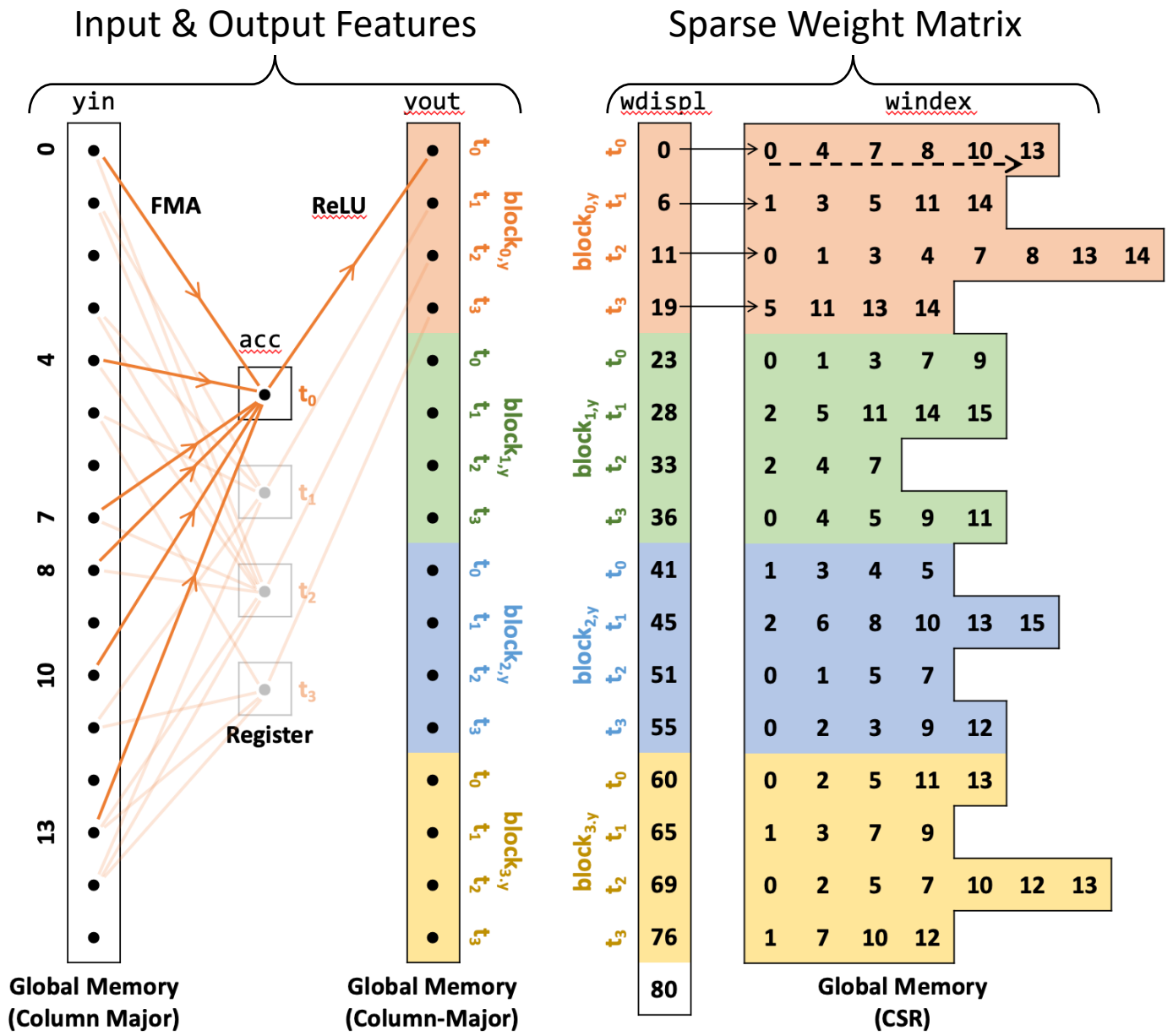
- Activation features: dense (col.-major)
- Gather approach
- Each thread computes a single output
- Sparse weights: CSR
- **ReLU is fused with SpMM**

Data Access Redundancies

- Weight matrix by all output features
- Input features by different threads

Data Access Latencies

- Irregular access to input features
- Uncoalesced access to weight matrix



SpMM Optimization for ML

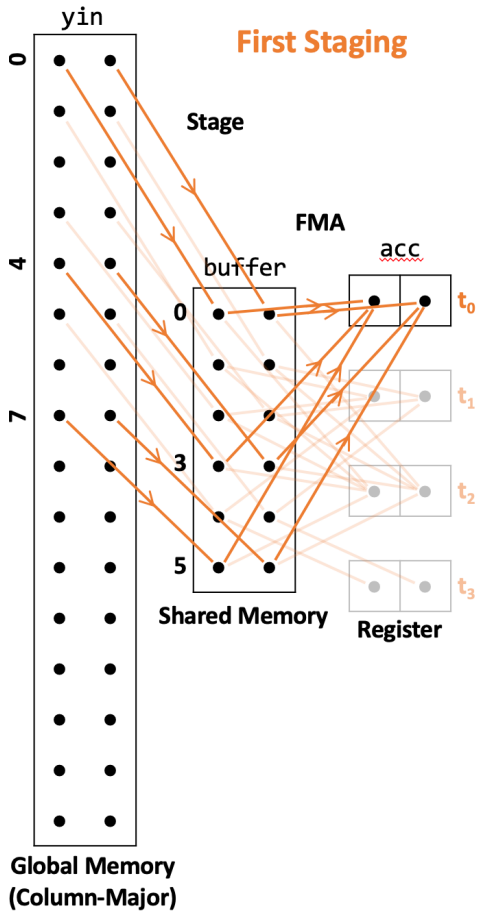
Register Tiling

Scratchpad Tiling

Fusing Activation

Tiled SpMM for Sparse Deep Neural Networks

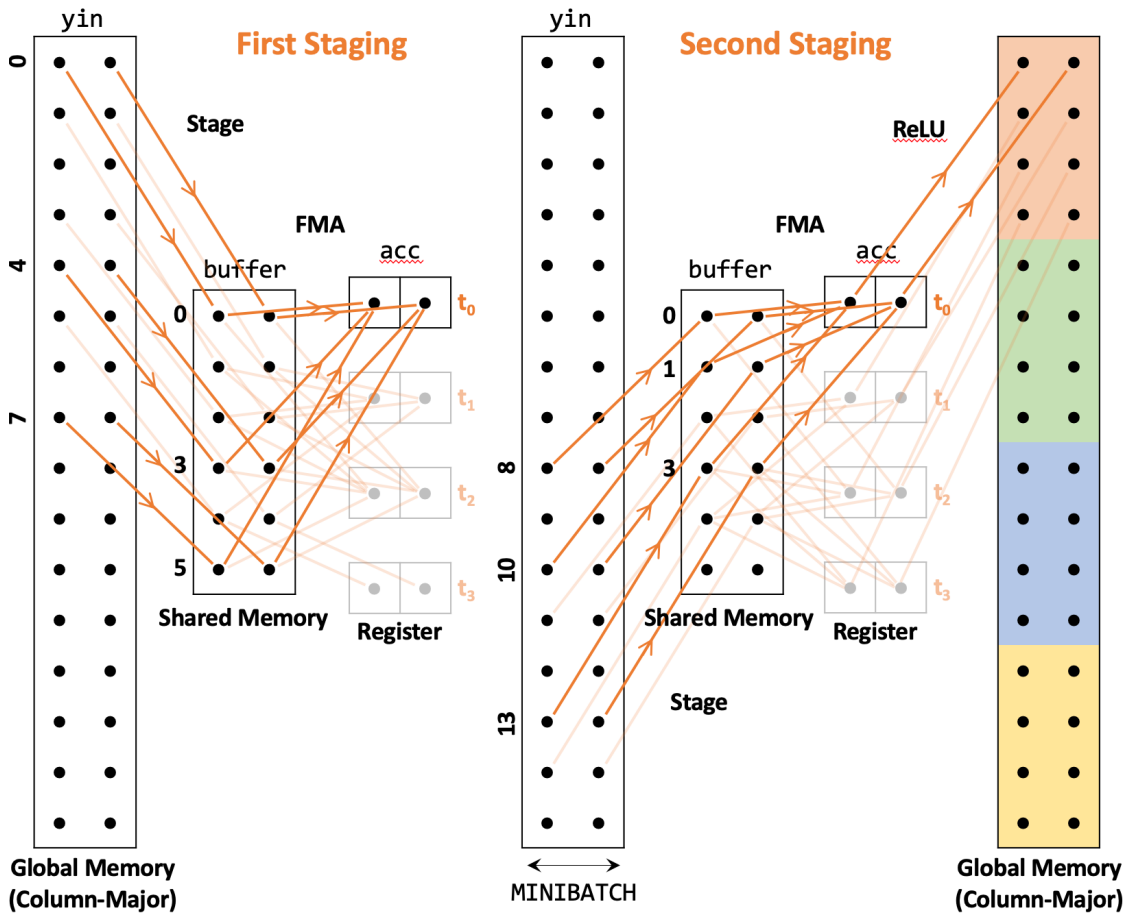
Multi-Level Input Buffering



Hidayetoglu, et al., "At-Scale Deep Neural Network Inference with Efficient GPU Implementation," Arxiv and HPEC'20.

Tiled SpMM for Sparse Deep Neural Networks

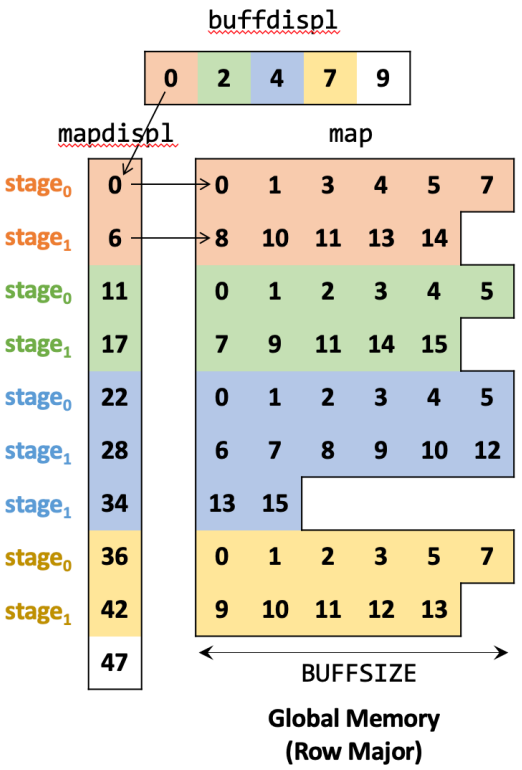
Multi-Level Input Buffering



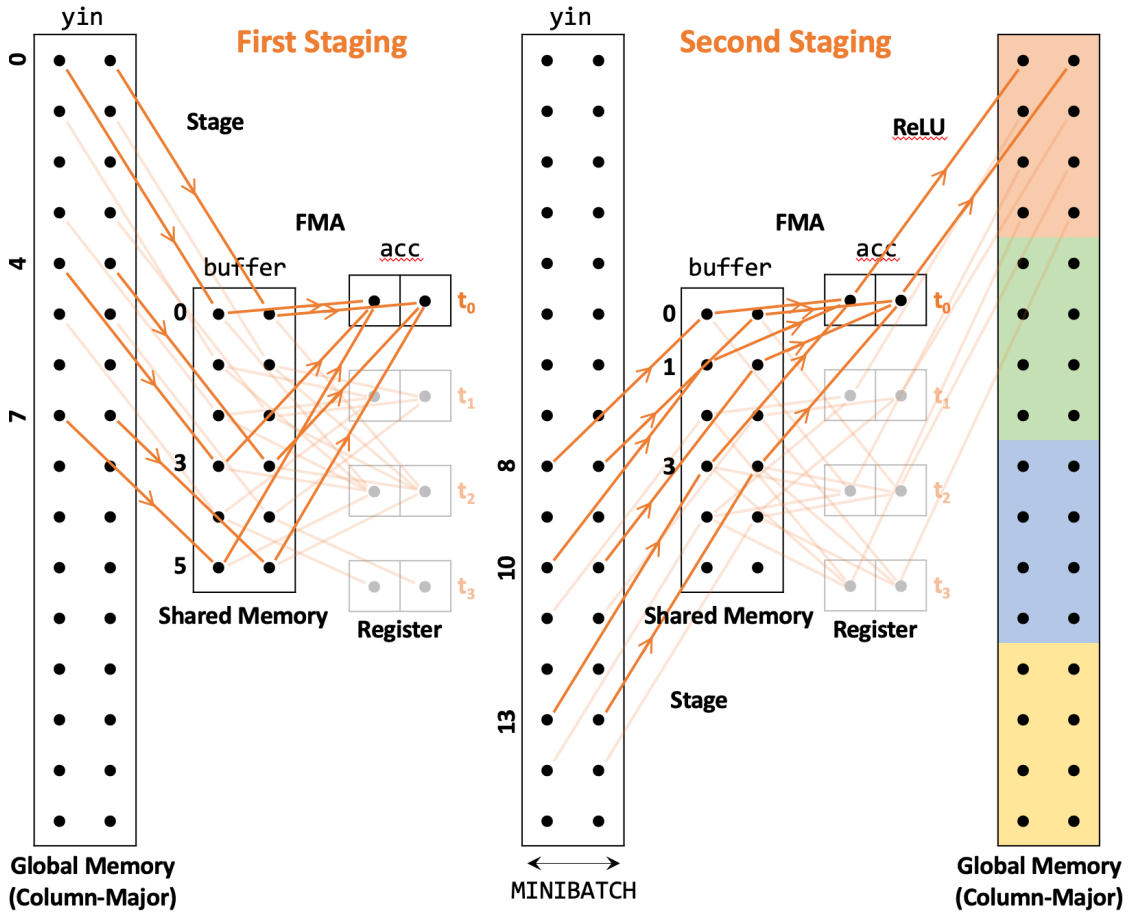
Hidayetoglu, et al., "At-Scale Deep Neural Network Inference with Efficient GPU Implementation," Arxiv and HPEC'20.

Tiled SpMM for Sparse Deep Neural Networks

Intermediate Data Structures



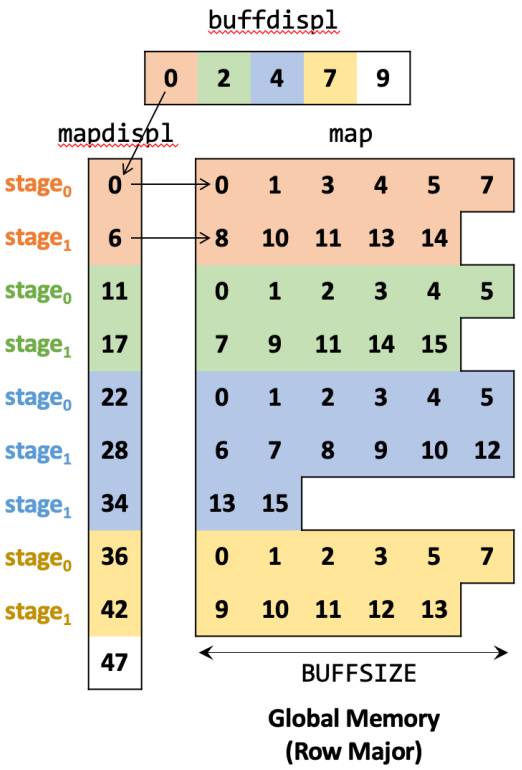
Multi-Level Input Buffering



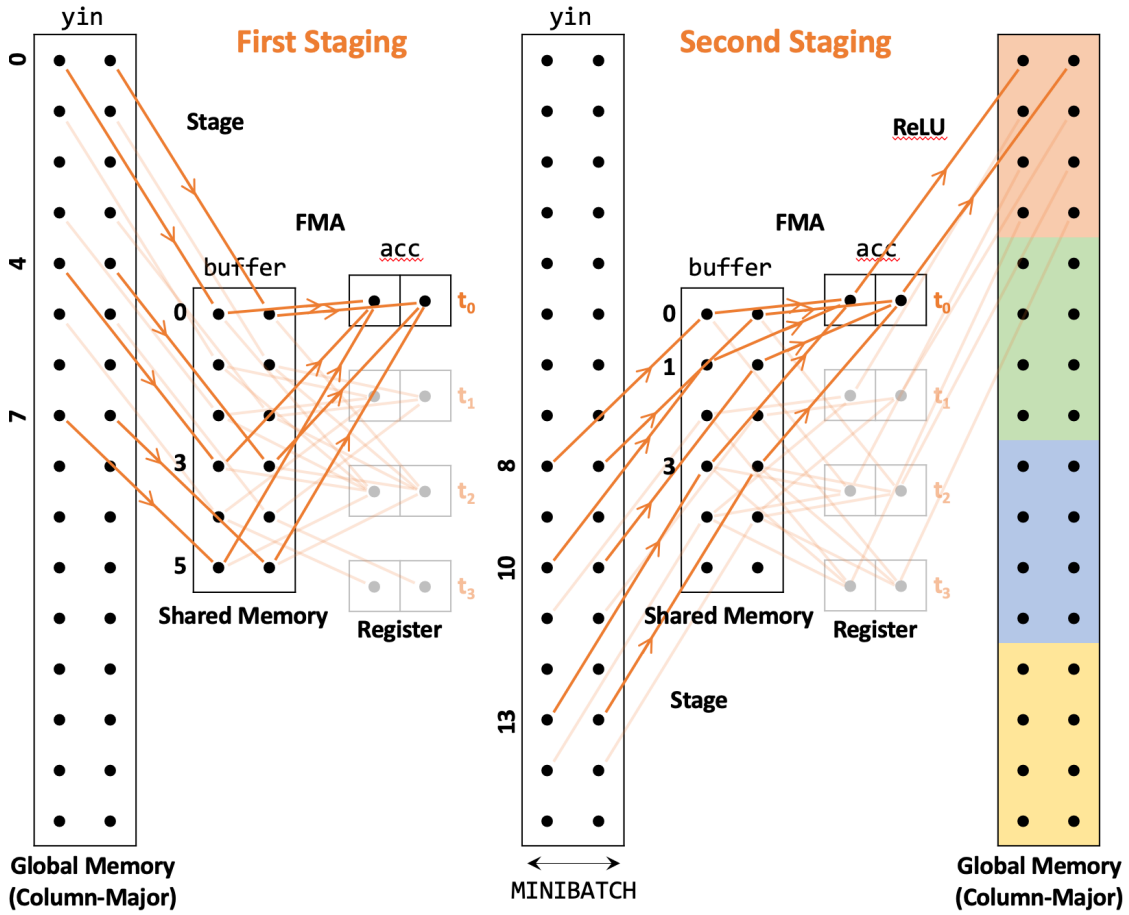
Hidayetoglu, et al., "At-Scale Deep Neural Network Inference with Efficient GPU Implementation," Arxiv and HPEC'20.

Tiled SpMM for Sparse Deep Neural Networks

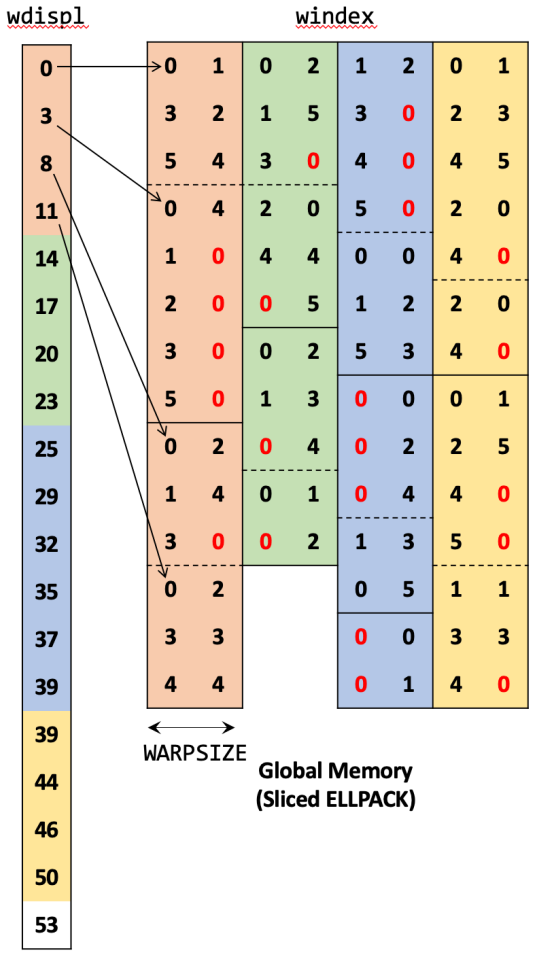
Intermediate Data Structures



Multi-Level Input Buffering



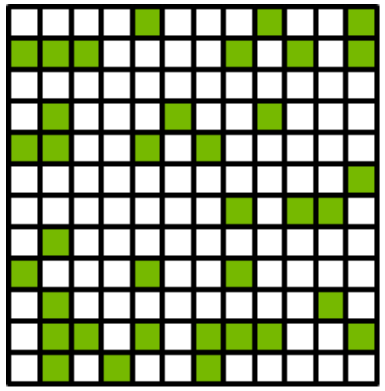
Weights Sliced ELLPACK

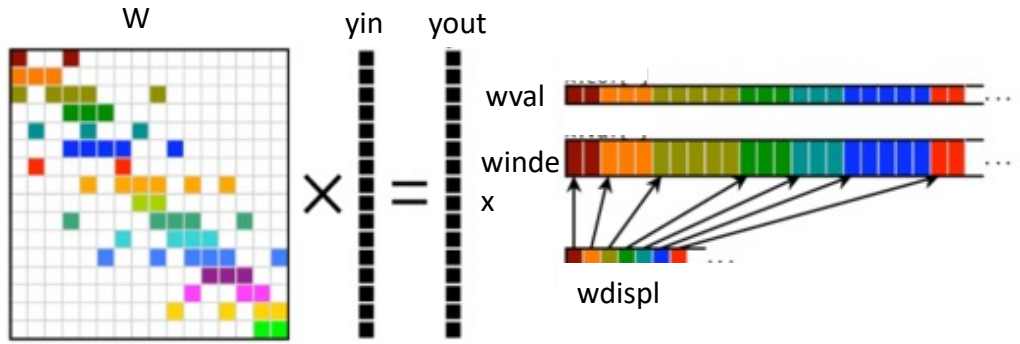


Hidayetoglu, et al., "At-Scale Deep Neural Network Inference with Efficient GPU Implementation," Arxiv and HPEC'20.

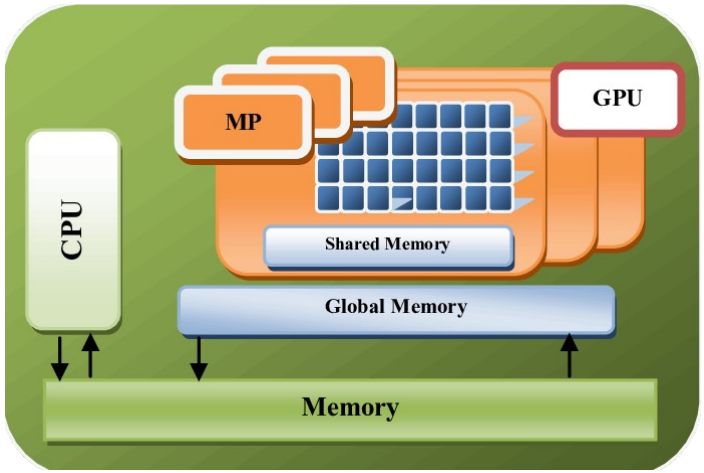
Tiled SpMM acceleration on GPUs – talk by Mert @ 1:50 pm

1. Optimize SpMM data structures and algorithms for generic hardware features

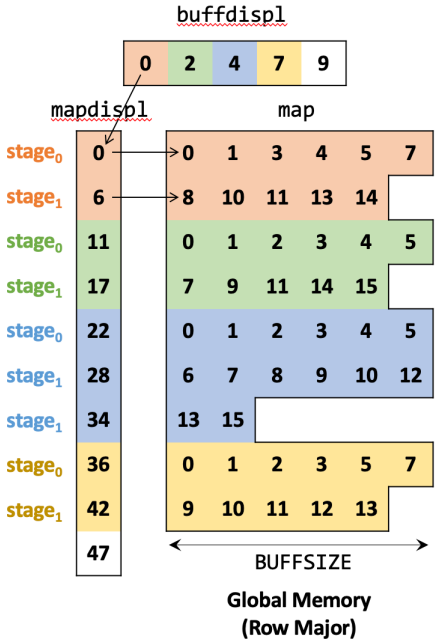




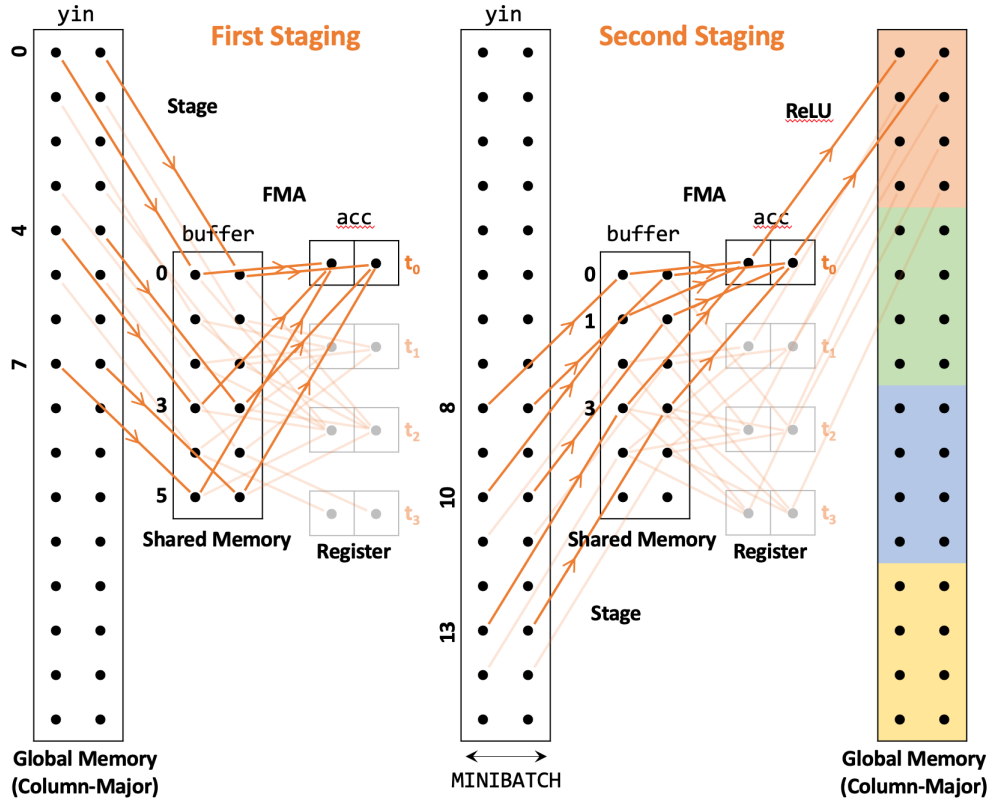
2. Co-optimize SpMM implementation for underlying hardware features



Intermediate Data Structures



Multi-Level Input Buffering

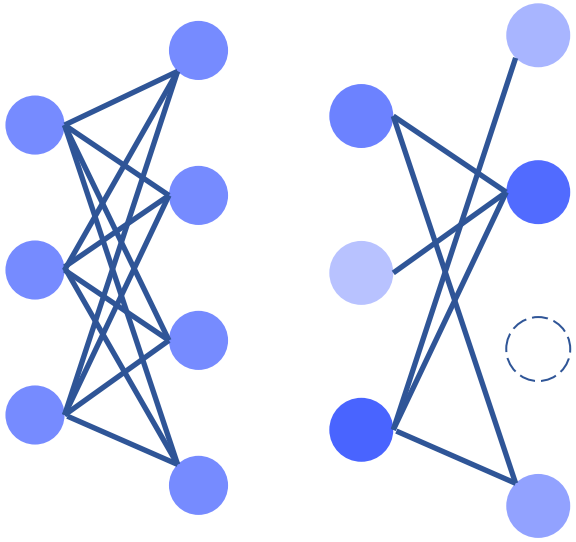


[Hidayetoglu et al: "At-Scale Sparse Deep Neural Network Inference With Efficient GPU Implementation," IEEE GraphChallenge 2020 (Champion)]

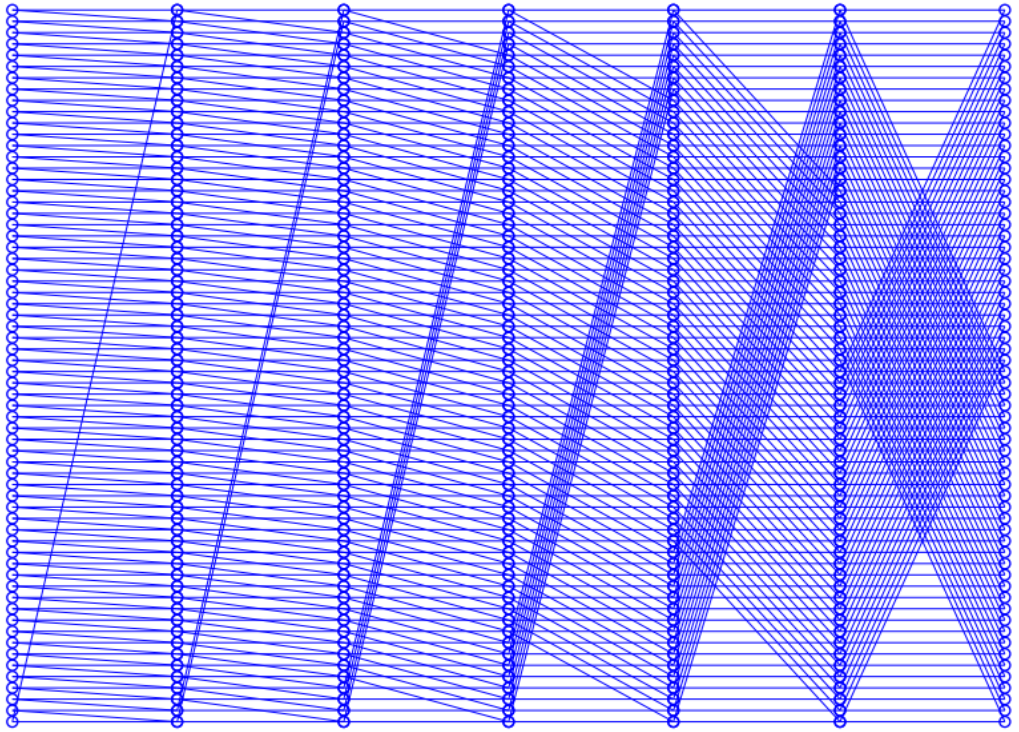
Tiled SpMM for Sparse Deep Neural Networks

MIT/Amazon/IEEE Graph Challenge

grapchallenge.mit.edu



(a) Fully connected layer;
(b) Sparse layer (after pruning)



A sparse DNN with 6 layers, 64 neurons per layer (synthetic, 2 connections per neuron) [1]

[1] J. Kepner et al. Sparse Deep Neural Network Graph Challenge. HPEC 2019.

Tiled SpMM for Sparse Deep Neural Networks

MIT/Amazon/IEEE Graph Challenge

grapchallenge.mit.edu

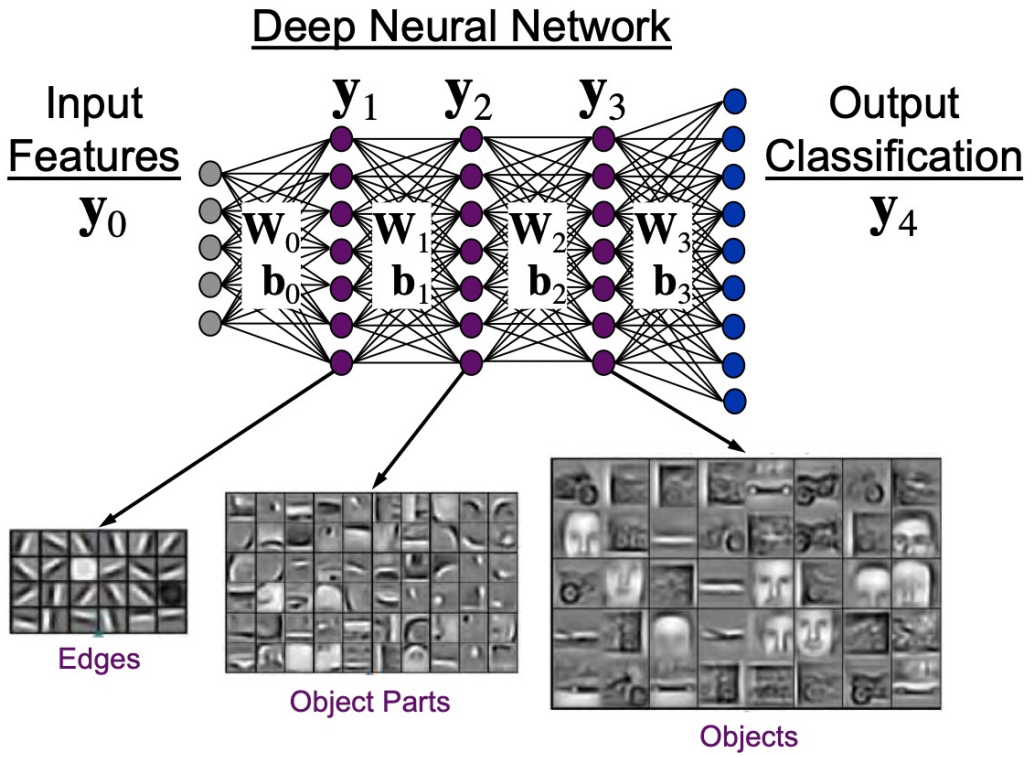
Challenges

- Irregular: Frequent random accesses to memory
- Various Scales: from 120 layers, 1024 neurons/layer to 1920 layers 65536 neurons/layer (16.3 GB)

Computation

- Input and parameters are in graph format (sparse)
- CPU performs data preprocessing
- GPU performs DNN inference: ℓ-th layer:

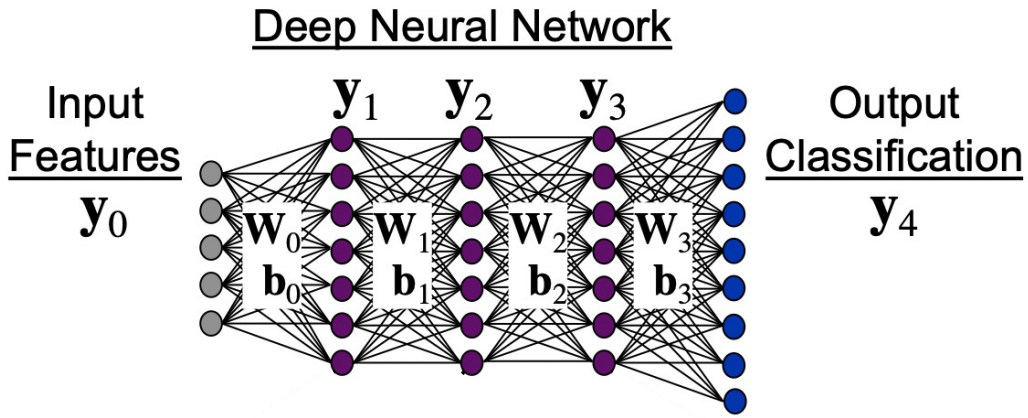
$$\boldsymbol{Y}_{\ell+1} = h(\boldsymbol{W}_{\ell}\boldsymbol{Y}_{\ell} + \boldsymbol{B}_{\ell})$$



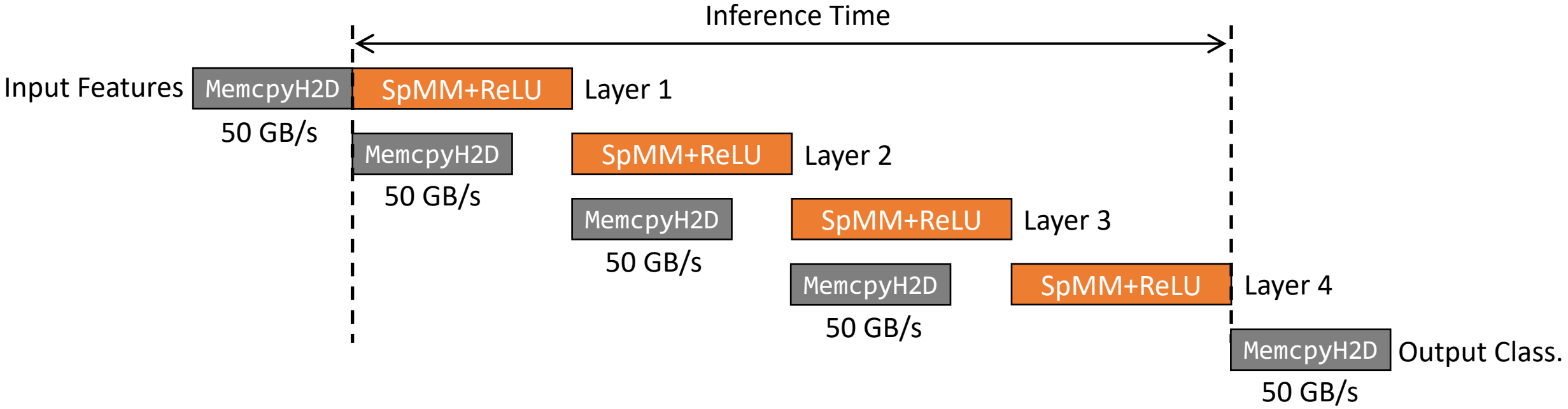
Tiled SpMM for Sparse Deep Neural Networks

Memory Optimizations

- `short` indices
- `half` weights (not used for Graph Challenge)
- Compact `struct` (not used for Graph Challenge)
- Batching for features
- Out-of-core streaming for weights



Out-of-core Streaming



Tiled SpMM for Sparse Deep Neural Networks

MIT/Amazon/IEEE Graph Challenge

grapchallenge.mit.edu

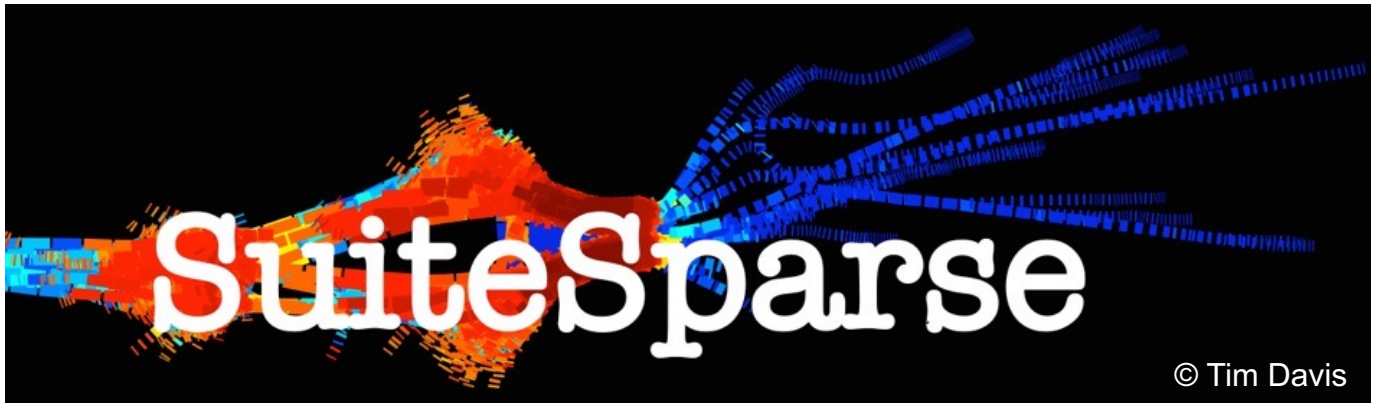
HPEC'20 Champion

Benchmark Throughput (# edges per second)

		This Work	Bisson & Fatica [18] 2019 Champion		Davis et al. [20] 2019 Champion		Ellis & Rajamanickam [21] 2019 Innovation		Wang et al. [22] 2019 Student Innov.		Wang et al. [23] 2019 Finalist	
Neurons	Layers	Throughput	Throughput	Speedup	Throughput	Speedup	Throughput	Speedup	Throughput	Speedup	Throughput	Speedup
1024	120	2.917E+13	4.517E+12	6.46	1.533E+11	190.28	2.760E+11	105.69	1.407E+11	207.32	8.434E+10	345.88
	480	2.930E+13	7.703E+12	3.80	2.935E+11	99.83	2.800E+11	104.64	1.781E+11	164.51	9.643E+10	303.84
	1920	2.883E+13	8.878E+12	3.25	2.754E+11	104.68	2.800E+11	102.96	1.896E+11	152.06	9.600E+10	300.30
4096	120	8.220E+13	6.541E+12	12.57	1.388E+11	592.22	2.120E+11	387.74	1.943E+11	423.06	6.506E+10	1,263.52
	480	8.222E+13	1.231E+13	6.68	1.743E+11	471.72	2.160E+11	380.65	2.141E+11	384.03	6.679E+10	1,230.99
	1920	8.232E+13	1.483E+13	5.55	1.863E+11	441.87	2.160E+11	381.11	2.197E+11	374.69	6.617E+10	1,244.02
16384	120	1.469E+14	1.008E+13	14.57	1.048E+11	1,401.53	1.270E+11	1,156.54	1.966E+11	747.10	3.797E+10	3,867.84
	480	1.394E+14	1.500E+13	9.29	1.156E+11	1,206.23	1.280E+11	1,089.38	2.060E+11	676.89	3.747E+10	3,721.66
	1920	1.464E+14	1.670E+13	8.77	1.203E+11	1,216.96	1.310E+11	1,117.56	1.964E+11	745.52	3.750E+10	3,903.72
65536	120	1.796E+14	9.388E+12	19.13	1.050E+11	1710.29	9.110E+10	1971.24	1.892E11	949.15	–	–
	480	1.703E+14	1.638E+13	10.40	1.091E+11	1,560.59	8.580E+10	1,984.38	1.799E+11	946.41	–	–
	1920	1.714E+14	1.787E+13	9.59	1.127E+11	1,520.59	8.430E+10	2,032.86	–	–	–	–

Hidayetoglu, et al., "At-Scale Deep Neural Network Inference with Efficient GPU Implementation," Arxiv and HPEC'20.

Generalization of Tiled SpMM



2,893 Sparse Matrices

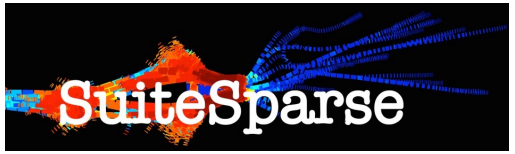
Picked the largest 200 samples in terms of nonzeroes

The matrices in the Collection cover a wide spectrum of domains, including

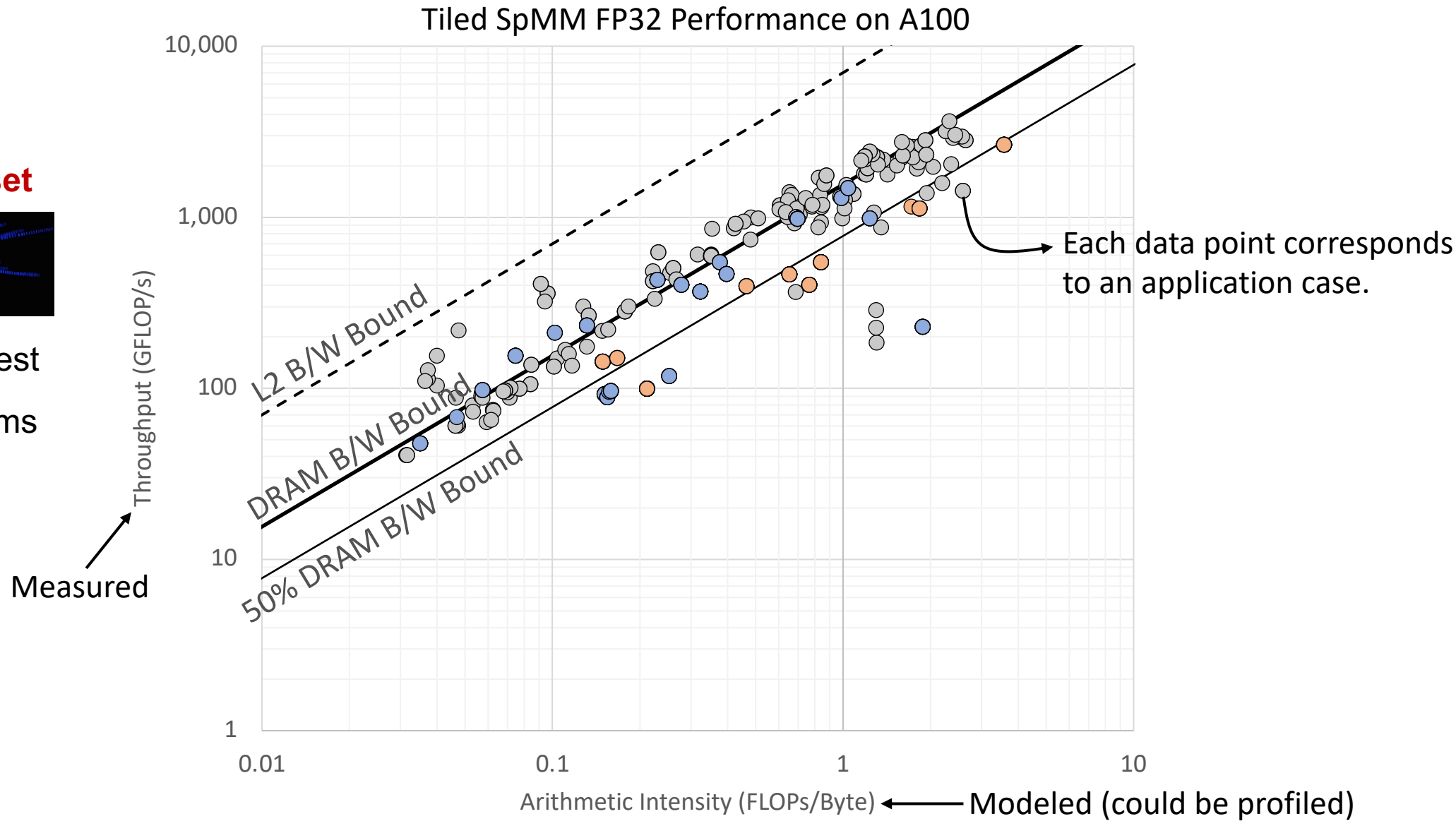
- 2D or 3D Geometric Domains
  - Structural engineering
  - Computational fluid dynamics
  - Model reduction
  - Electromagnetics
  - Semiconductor devices
  - Thermodynamics
  - Material science
  - Acoustics
  - Computer graphics and vision
  - Robotics and kinematics
  - Other spatial discretizations
- Non-Geometric Domains
  - Optimization
  - Economic and financial modeling
  - Circuit simulation
  - Theoretical and quantum chemistry
  - Chemical process simulation
  - Mathematics and statistics
  - Power networks
  - Other networks and graph-structured data

Tiled SpMM on Roofline Plot

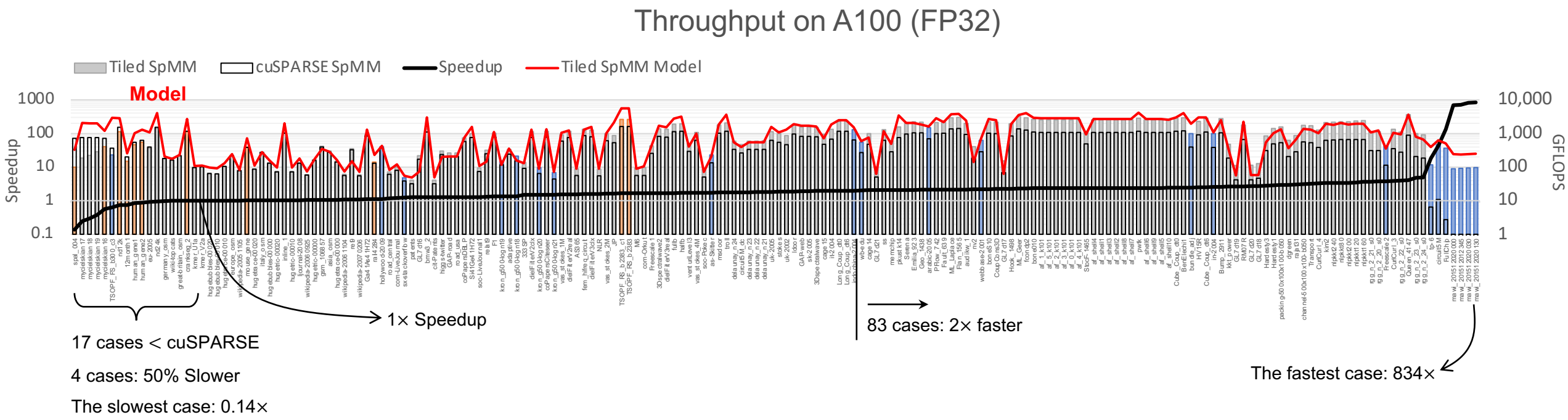
Benchmark Dataset



We picked the largest 200 samples in terms of nonzeroes.



Tiled SpMM vs. cuSPARSE



Speedup over cuSPARSE

(vendor-provided HPC Library)

Average: 19.2×

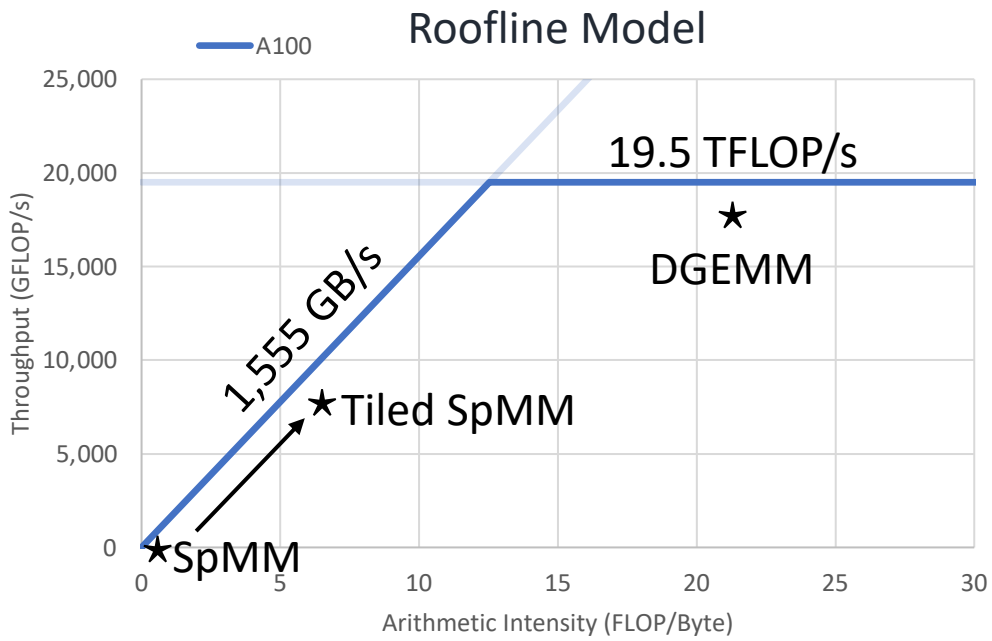
Geo-Mean: 2.03×

Har-Mean: 1.48×

Conclusion

Conclusion

Observation: the bottleneck of sparse computations is data movement over the slowest interconnect, rather than arithmetic operations.

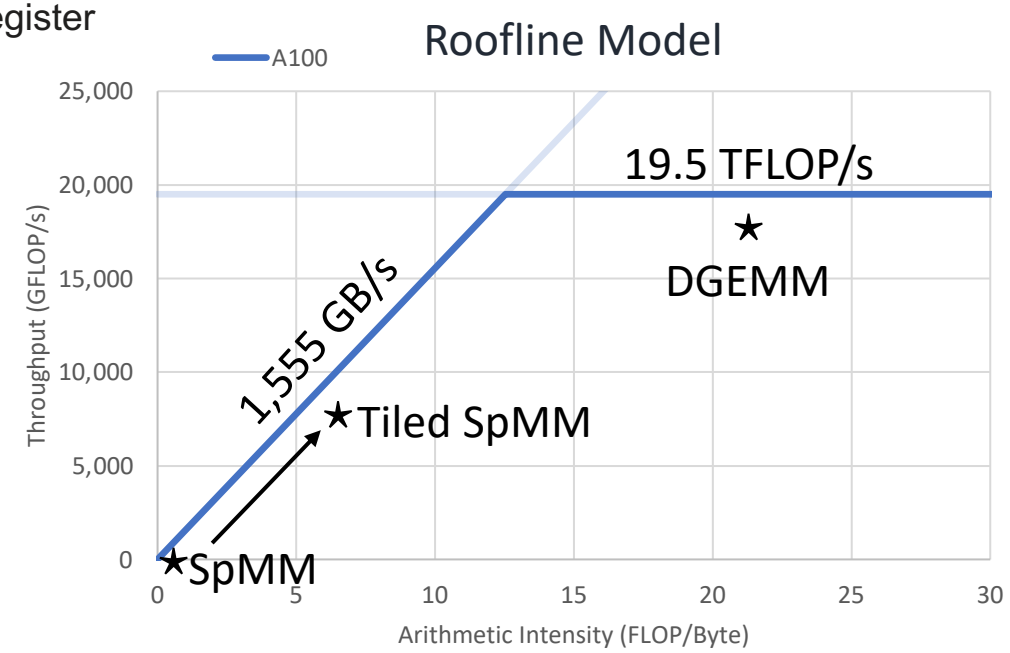


Conclusion

Observation: the bottleneck of sparse computations is data movement over the slowest interconnect, rather than arithmetic operations.

This Work: With the proposed tiling techniques and a careful orchestration of data movement over the memory hierarchy, we can accelerate SpMM on GPUs to a great extent.

DRAM->L2$->L1$/scratchpad->register



Conclusion

Observation: the bottleneck of sparse computations is data movement over the slowest interconnect, rather than arithmetic operations.

This Work: With the proposed tiling techniques and a careful orchestration of data movement over the memory hierarchy, we can accelerate SpMM on GPUs to a great extent.

Work in progress

We developed Tiled SpMM and …

- Benchmarked over 200 sparse matrices from a wide-spectrum of applications
- Verified performance bounds with the **proposed** analytical model
- Obtained 16x speedup over cuSPARSE

Product: Tiled SpMM is open source for other application developers.

https://github.com/merthidayetoglu/SpDNN_Challenge2020