

# Spconv 2.x Algorithm

Author: FindDefinition (<https://github.com/FindDefinition>)

# Dense Convolution

## Definitions

3
3
2
2
1

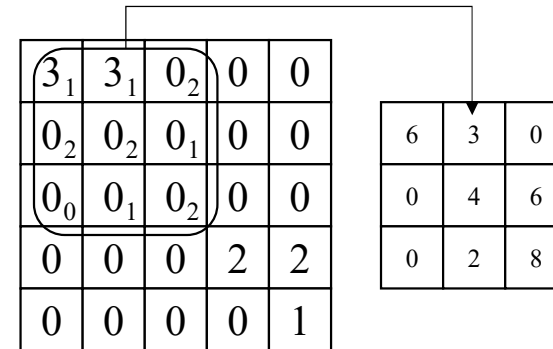
(0, 0)
(0, 1)
(3, 3)
(3, 4)
(4, 4)

3	3	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	2	2
0	0	0	0	1

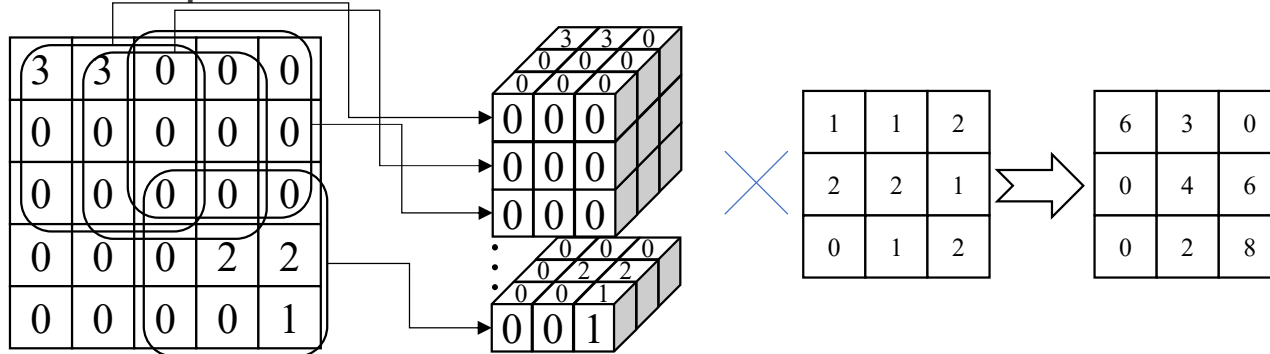
1	1	2
2	2	1
0	1	2

Sparse Data    Sparse Coords    Dense Data    Filters

## Dense Convolution



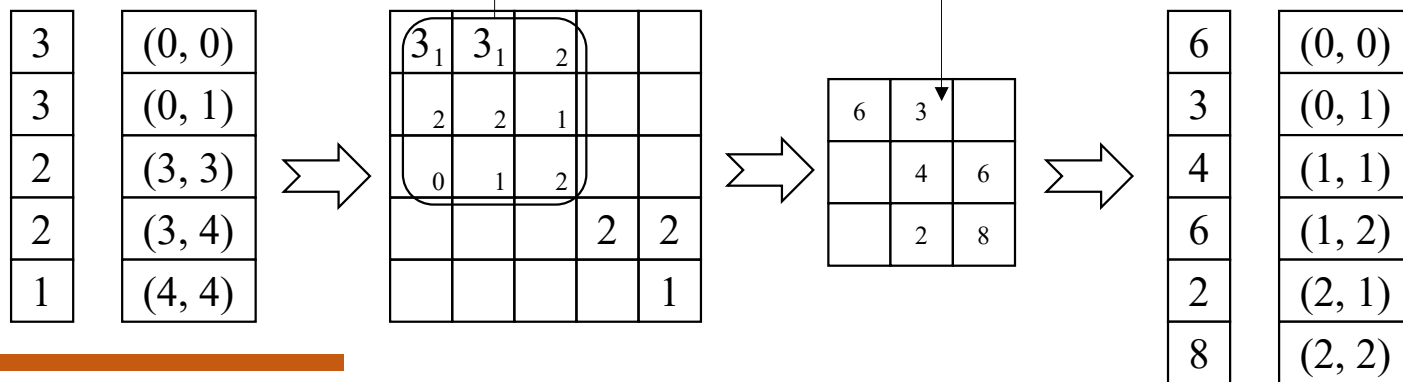
## Explicit Gemm Convolution



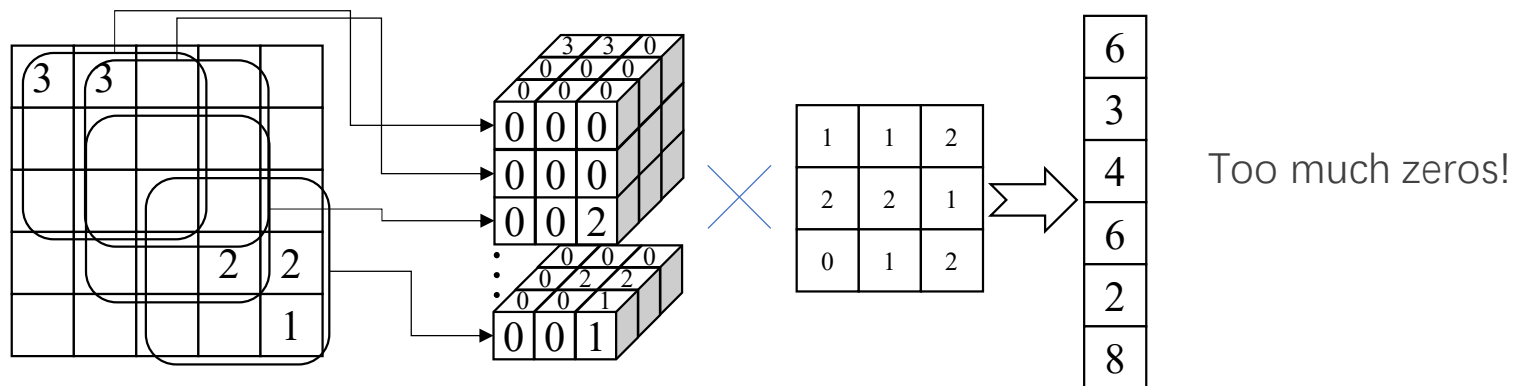
Explicit Gemm Conv Algorithm construct a matrix from input data, then do standard matrix multiply to get output.

# Sparse Convolution

## Basic Sparse Convolution

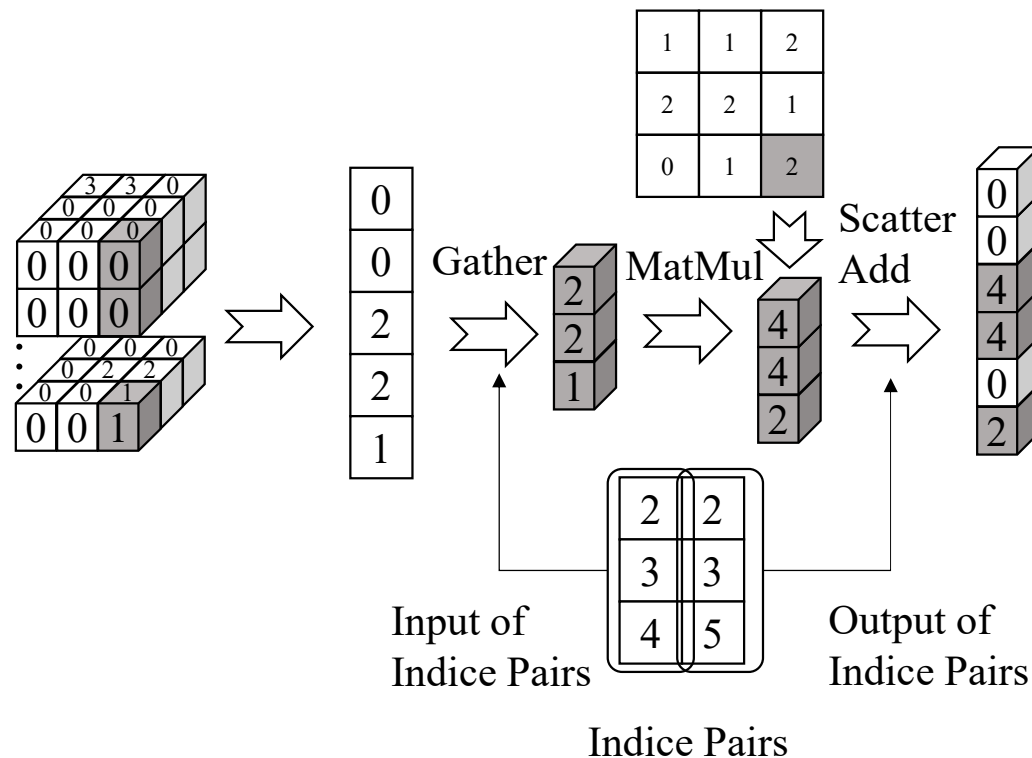


## Explicit Gemm Sparse Convolution



# Sparse Conv: Explicit Native

## Native Sparse Convolution Algorithm: Gather-Gemm-ScatterAdd



## Pros

- Minimal MMAs
- Easy to implement

## Cons

- Serialized IO
- Too much write operations

# Sparse Conv: Explicit Native

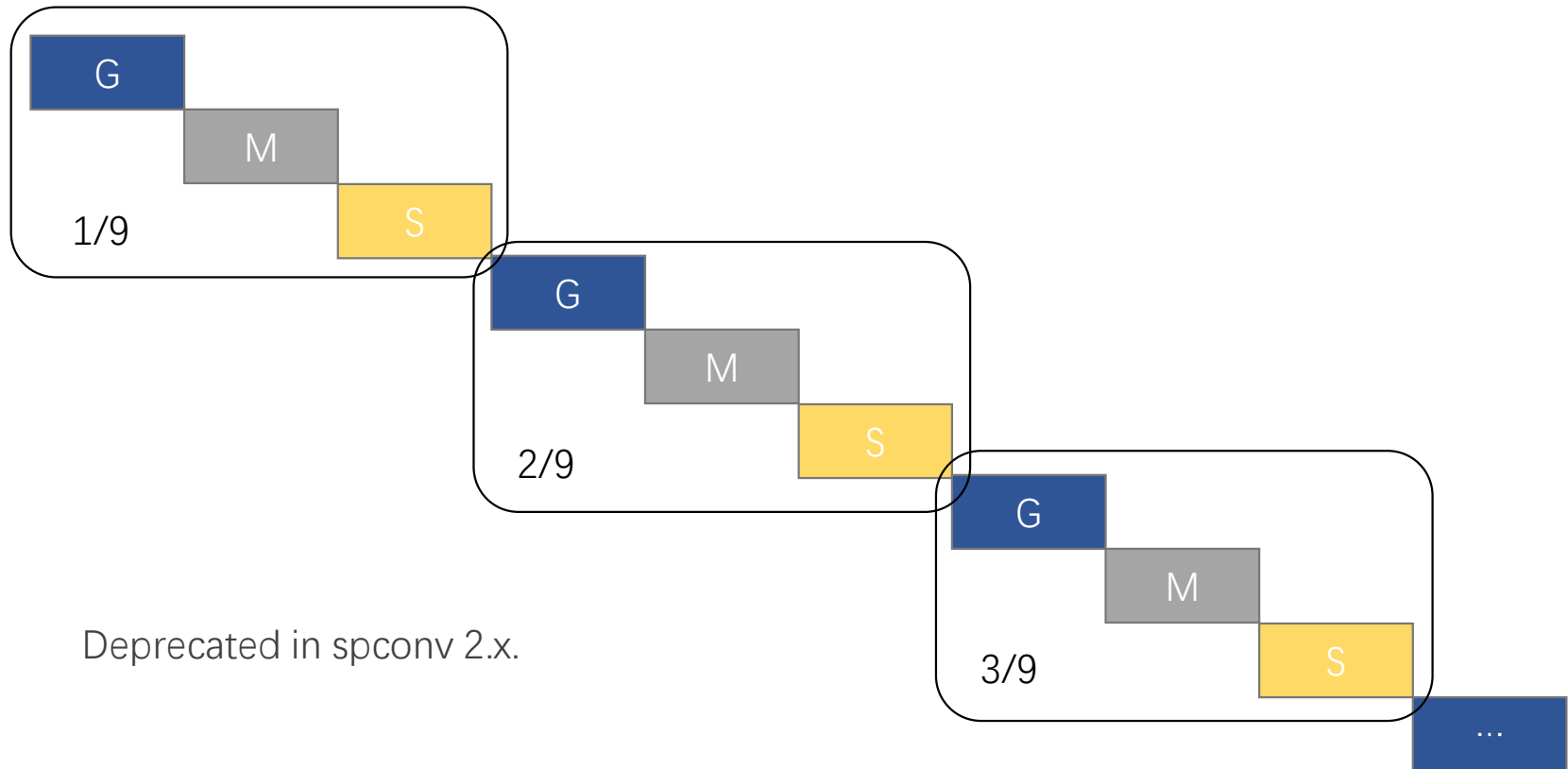
## Gather-Gemm-ScatterAdd Pipeline

3x3 conv

G: Gather

M: Matmul

S: Scatter



Deprecated in spconv 2.x.

# Sparse Conv: Fused Native

Gemm Kernel: Overlapped compute and Read

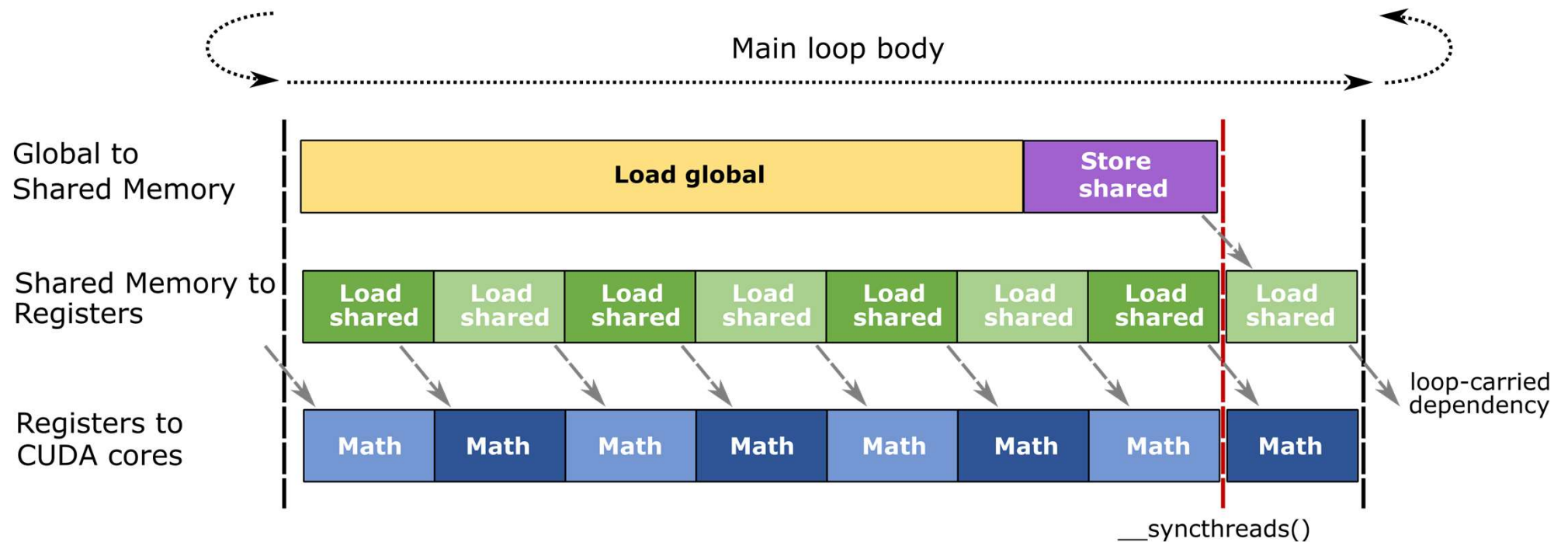


Image takes from <https://github.com/NVIDIA/cutlass>

# Sparse Conv: Fused Native

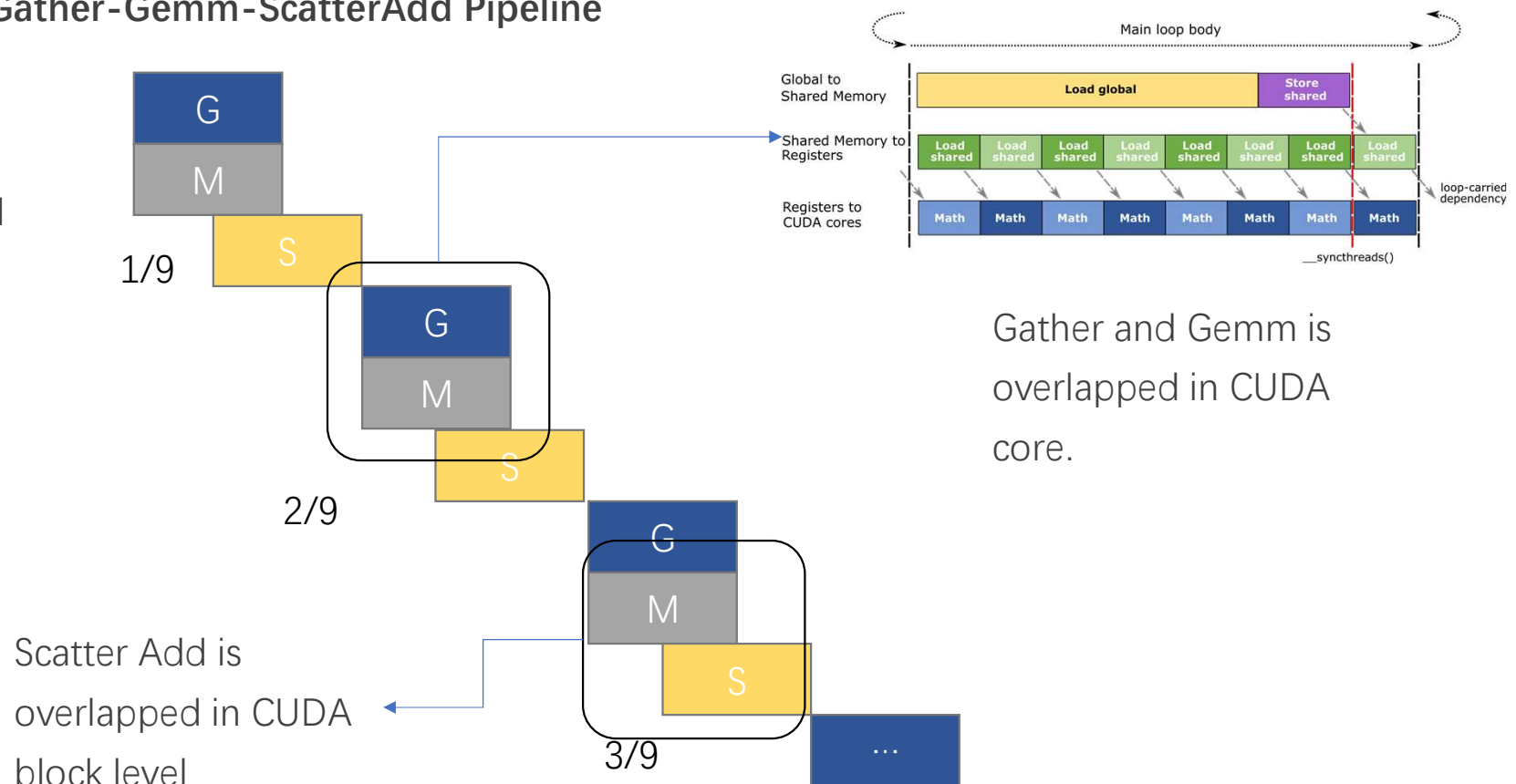
## Fused Gather-Gemm-ScatterAdd Pipeline

3x3 conv

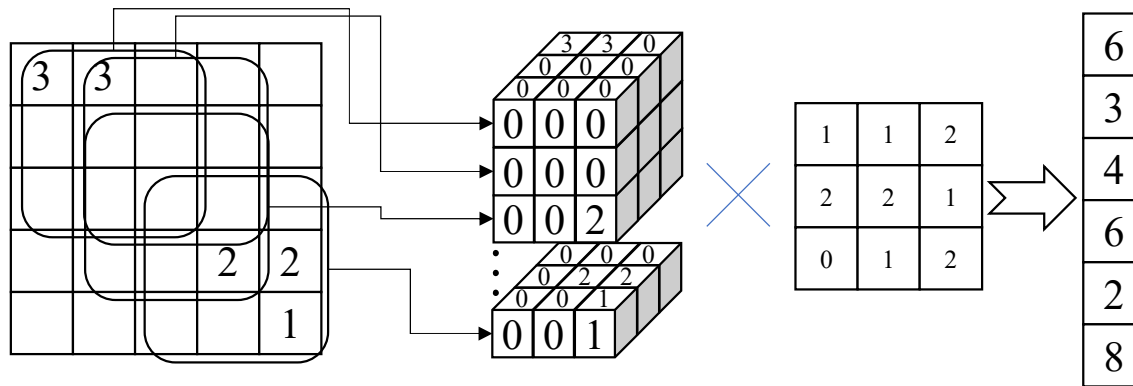
G: Gather

M: Matmul

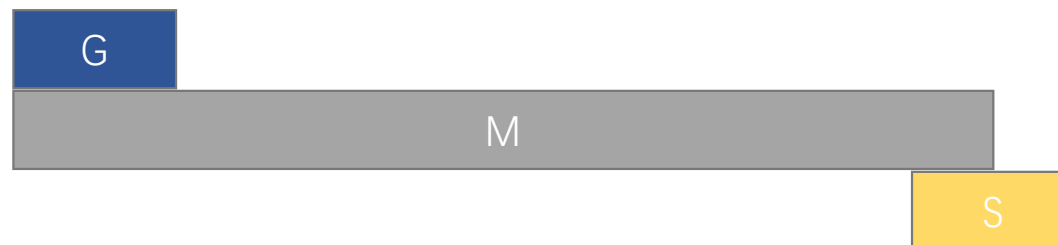
S: Scatter



# Sparse Conv: Implicit Gemm (1st try)



All calculation fused  
to one kernel, minimal  
IO, but too much  
zeros cause too much  
compute!





# Sparse Conv: Implicit Gemm (1st try)

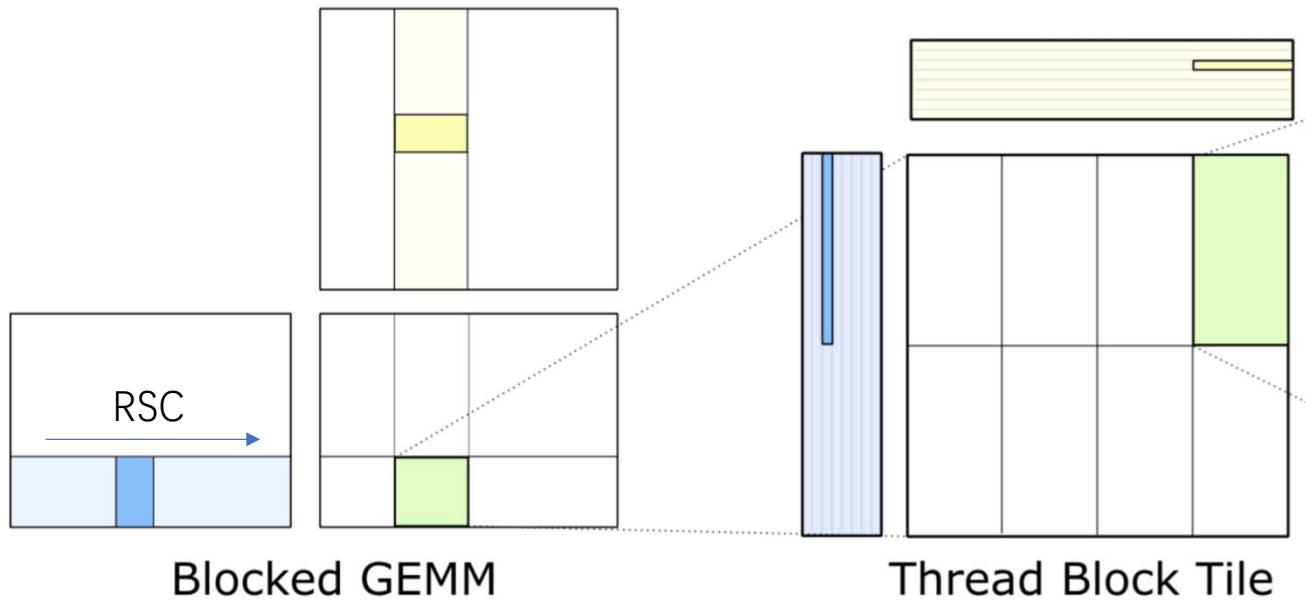
Input:  $[N, C]$

Filters:  $[K, R, S, C]$

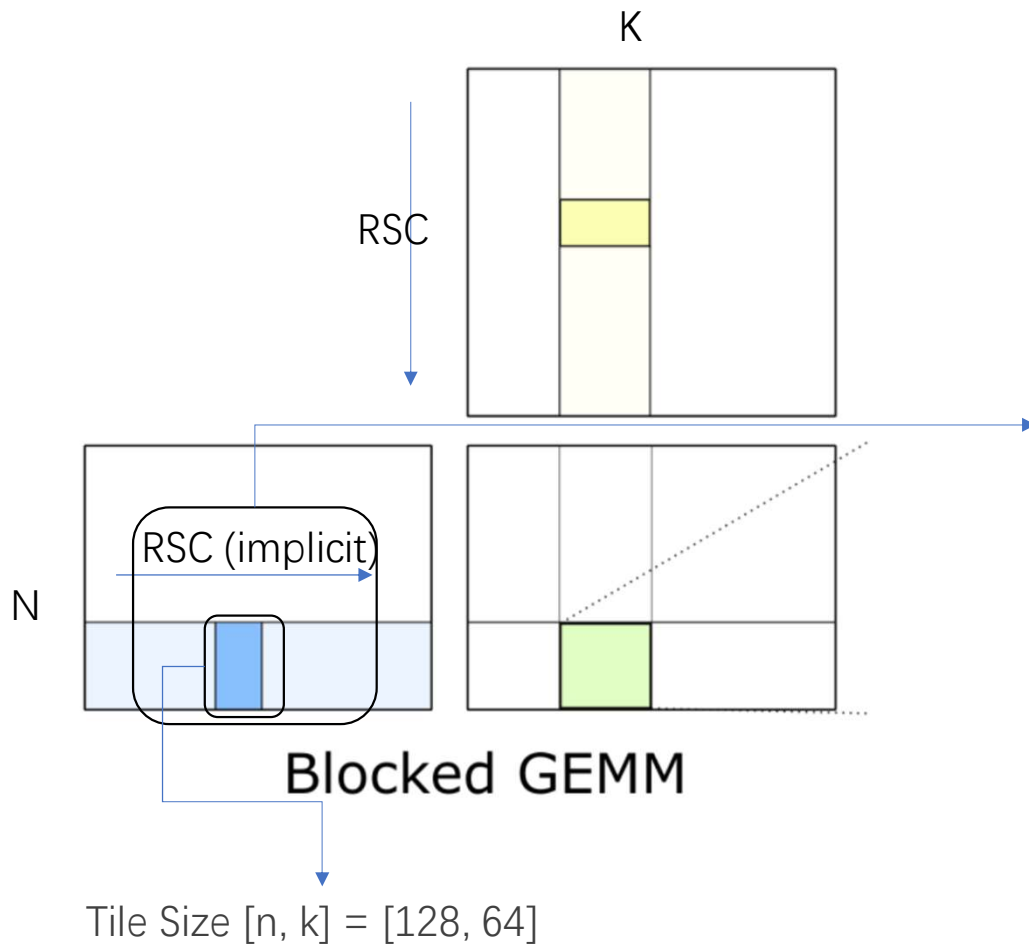
Output:  $[N, K]$

Input

$[N, C] \Rightarrow [N, R, S, C] \Rightarrow [N, RSC] @ [K, RSC].T \Rightarrow [N, K]$



# Sparse Conv: Implicit Gemm (1st try)



Assume  $R = 3, S = 3, C = 128$ ,  
Tile Size  $[n, k] = [128, 64]$

$RS = (0, 0)$ , $N$ extent: $[0-128]$ , $C$ extent: $[0-64]$
$RS = (0, 0)$ , $N$ extent: $[0-128]$ , $C$ extent: $[64-128]$
$RS = (0, 1)$ , $N$ extent: $[0-128]$ , $C$ extent: $[0-64]$
$RS = (0, 1)$ , $N$ extent: $[0-128]$ , $C$ extent: $[64-128]$

⋮

$RS = (2, 2)$ , $N$ extent: $[0-128]$ , $C$ extent: $[0-64]$
$RS = (2, 2)$ , $N$ extent: $[0-128]$ , $C$ extent: $[64-128]$

# Sparse Conv: Implicit Gemm (1st try)

RS = (0, 0), N extent: [0-128], C extent: [0-64]
RS = (0, 0), N extent: [0-128], C extent: [64-128]
RS = (0, 1), N extent: [0-128], C extent: [0-64]
RS = (0, 1), N extent: [0-128], C extent: [64-128]
⋮
RS = (2, 2), N extent: [0-128], C extent: [0-64]
RS = (2, 2), N extent: [0-128], C extent: [64-128]

We need to read whole 128 lines of input data and compute them, because CUDA is parallel in block level, we can only skip zeros in block level, not thread level!

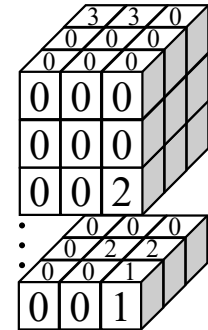
If the whole block is zero, then we can skip them to save compute time.

# Sparse Conv: Masked Implicit Gemm

We want to skip white boxes  
in figure below.

RS = (0, 0), N extent: [0-128], C extent: [0-64]
RS = (0, 0), N extent: [0-128], C extent: [64-128]
RS = (0, 1) , N extent: [0-128], C extent: [0-64]
RS = (0, 1) , N extent: [0-128], C extent: [64-128]
⋮
RS = (2, 2) , N extent: [0-128], C extent: [0-64]
RS = (2, 2) , N extent: [0-128], C extent: [64-128]

Recall Explicit Gemm  
Algorithm, for every line  
(RSC) of input, we already  
know some RS is zero.  
But we can't ensure RS in  
whole block is zero.



# Sparse Conv: Masked Implicit Gemm

**Sort!!!**

Assume Conv1d, Kernel size  
is 5

N=0	0	1	0	0	1
N=1	1	1	0	1	0
N=2	0	1	1	0	1
N=3	0	0	0	0	1
N=4	0	1	0	0	1

Sort  
⇒

N=3	0	0	0	0	1
N=0	0	1	0	0	1
N=4	0	1	0	0	1
N=2	0	1	1	0	1
N=1	1	1	0	1	0

Reduce



0	1	0	0	1
---	---	---	---	---

If  $n$  (tile size  $[n, k]$ ) is 3, we  
can skip in block level  
according to reduced mask!

# Sparse Conv: Masked Implicit Gemm

RS = (0, 0), N extent: [0-128], C extent: [0-64]
RS = (0, 0), N extent: [0-128], C extent: [64-128]
RS = (0, 1), N extent: [0-128], C extent: [0-64]
RS = (0, 1), N extent: [0-128], C extent: [64-128]
⋮
RS = (2, 2), N extent: [0-128], C extent: [0-64]
RS = (2, 2), N extent: [0-128], C extent: [64-128]

The whole white block is skipped based on reduced mask in 128 lines.

In practical SubMConv3d data, valid location is 780k origin implicit gemm size is 6000k, if n (tile size [n, k]) = 32, after reduce, we can get 1500k locations,  $\frac{3}{4}$  zeros are skipped.

# Sparse Conv: Masked Implicit Gemm

## Pros

- Fuse All operations to one Kernel

## Cons

- Still need to calculate much zeros

### F32/F16

### Native

### Implicit Gemm

Forward

21.7ms/13.7ms 23.5ms/11.2ms

Backward

41.9ms/25.2ms 51.0ms/13.8ms

Float 32 is slow because The bottleneck of kernel is float 32 computation! For float32, more zeros, slower running

# Sparse Conv: Split-Masked Implicit Gemm

780k -> 1500k, we still calculate almost 100% more zeros in previous algorithm. Why sort can't resolve mask perfectly?

Answer: Symmetric Distribution of SubMConv mask

Assume Conv1d, Kernel size  
is 5, if we have a mask:

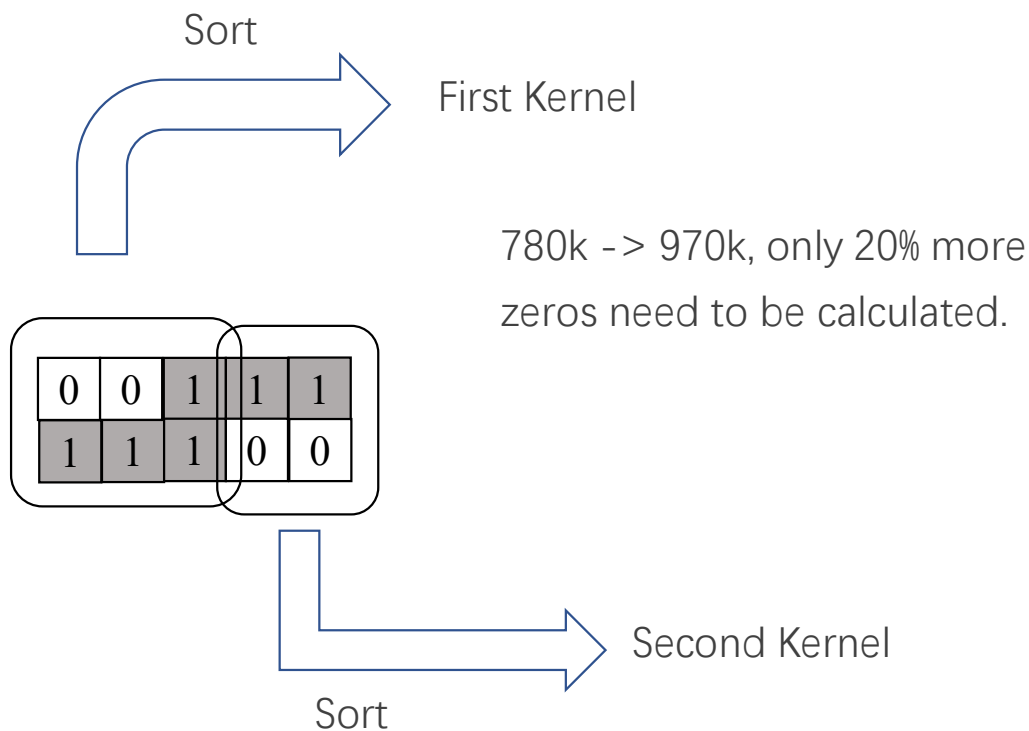
0	0	1	1	1
---	---	---	---	---

A symmetric mask always exists  
for SubMConv:

1	1	1	0	0
---	---	---	---	---



# Sparse Conv: Split-Masked Implicit Gemm



## Pros

- Minimal Zeros

## Cons

- Sort twice
- Generate More argsort data

## F32/F16

Forward  
Backward

## Native

21.7ms/13.7ms  
41.9ms/25.2ms

## Implicit Gemm

23.5ms/11.2ms  
51.0ms/13.8ms

## Implicit Gemm Split Mask

22ms/12.2ms  
41.1ms/12.2ms



Thanks!

---

