Spconv 2.x Algorithm

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

Dense Convolution

Definitions

3
3
2
2
1

Sparse Data

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

Sparse Coords

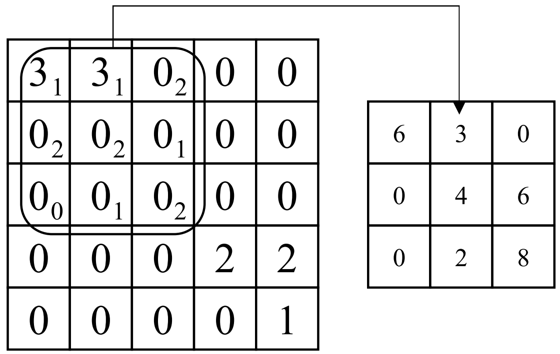
3	3	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	2	2
0	0	0	0	1

Dense Data

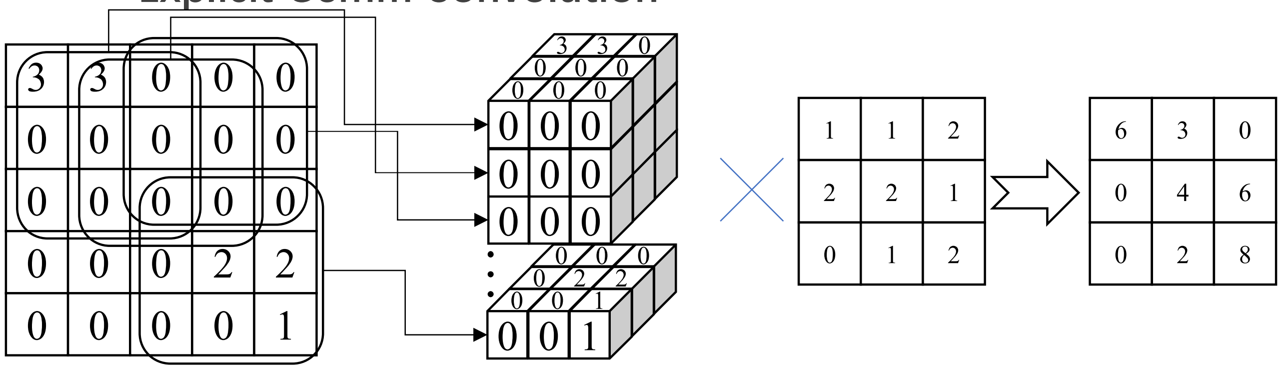
1	1	2
2	2	1
0	1	2

Filters

Dense Convolution



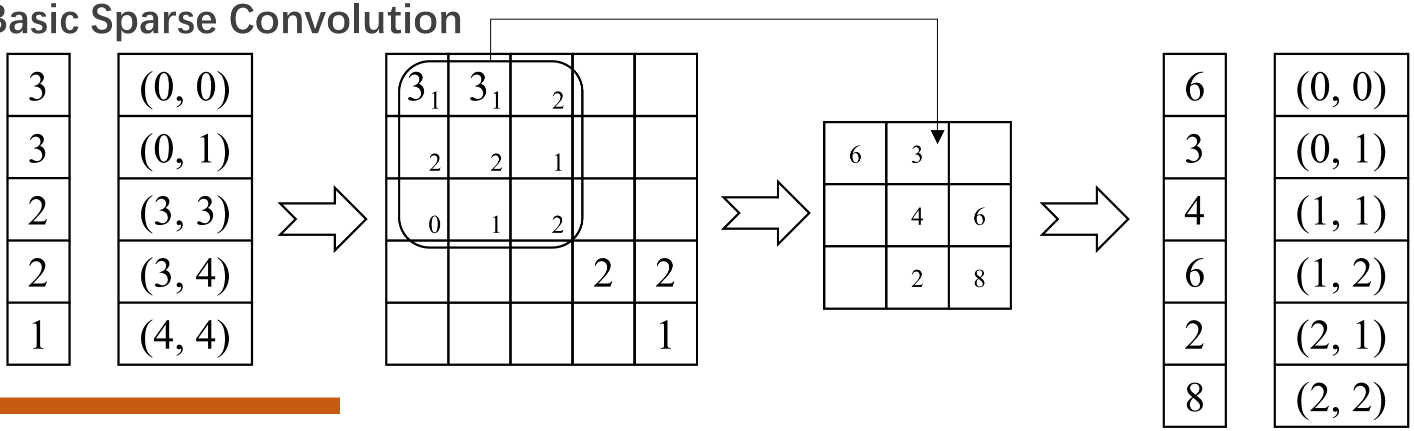
Explicit Gemm Convolution



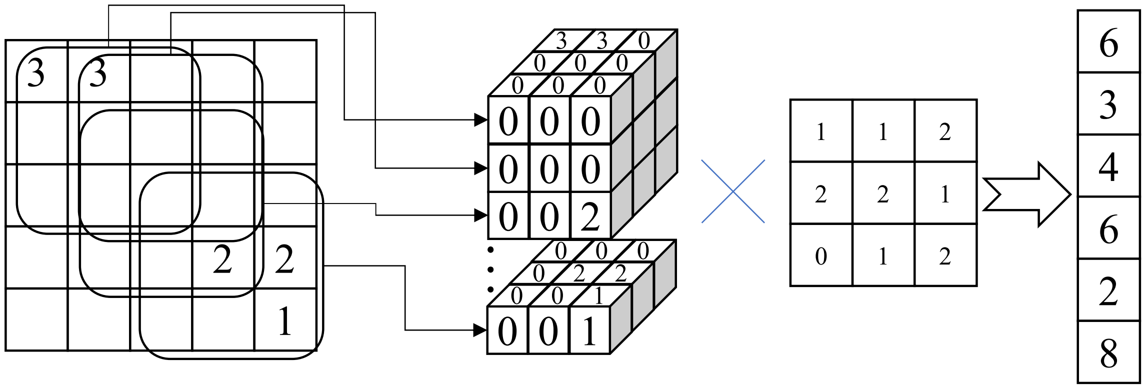
Explicit Gemm Conv Algorithm construct a matrix from input data, then do standard matrix multiply to get output.

Sparse Convolution

Basic Sparse Convolution



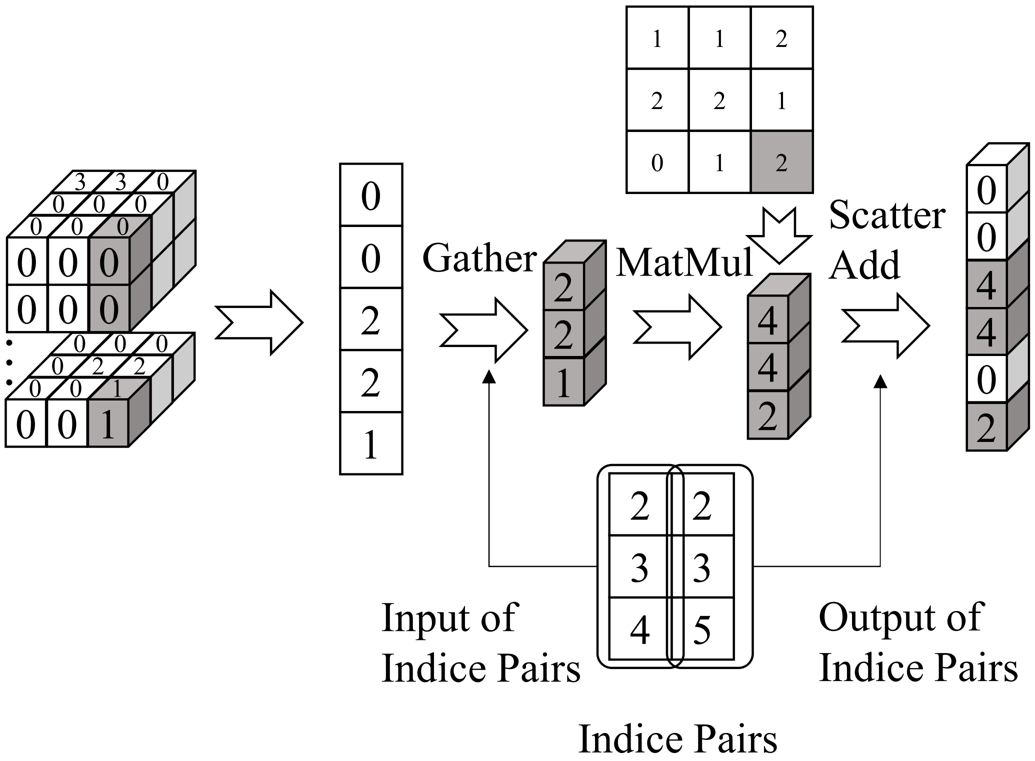
Explicit Gemm Sparse Convolution



Too much zeros!

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

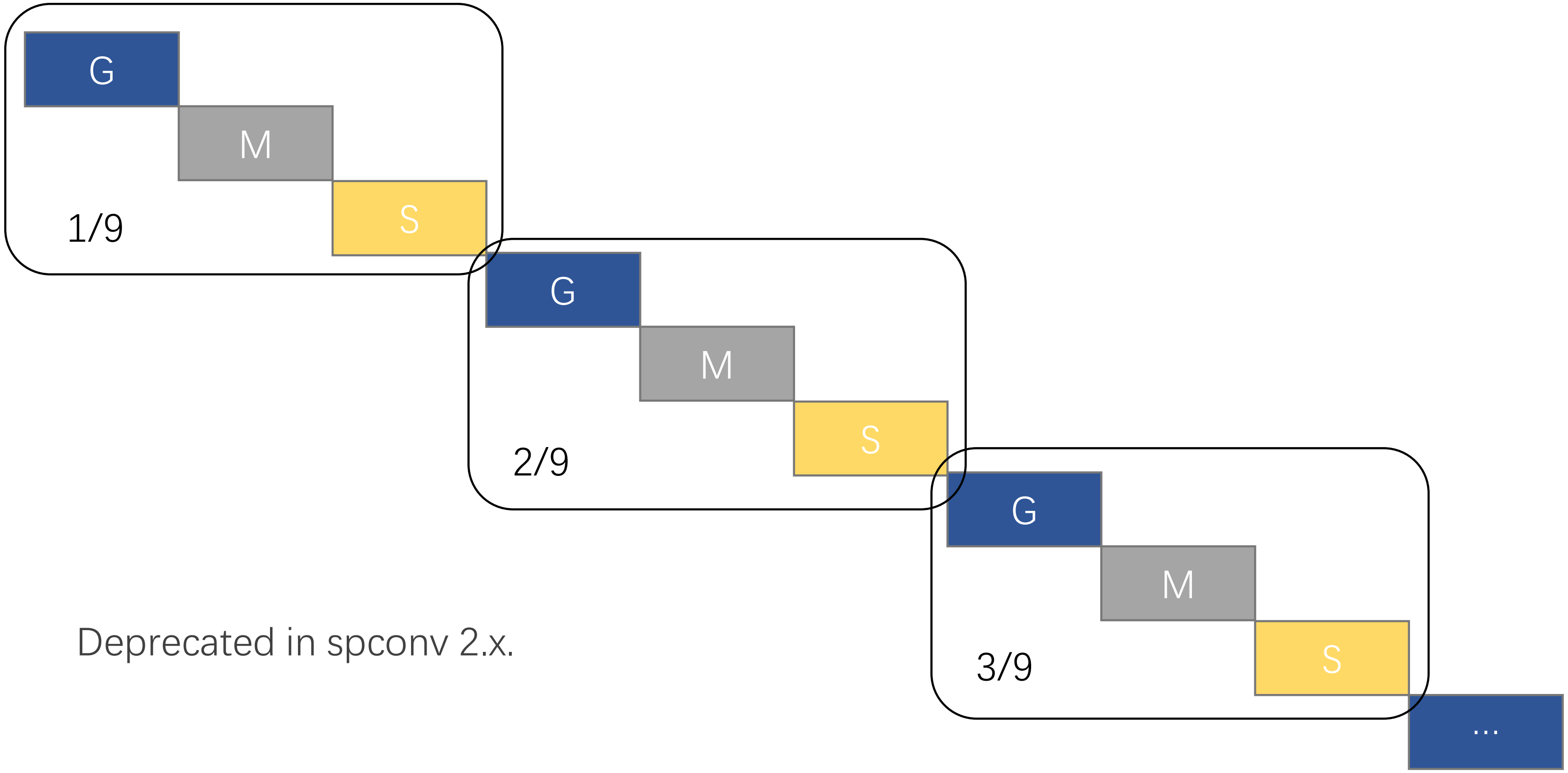
Gather-Gemm-ScatterAdd Pipeline

3x3 conv

G: Gather

M: Matmul

S: Scatter



Deprecated in spconv 2.x.

Sparse Conv: Fused Native

Gemm Kernel: Overlapped compute and Read

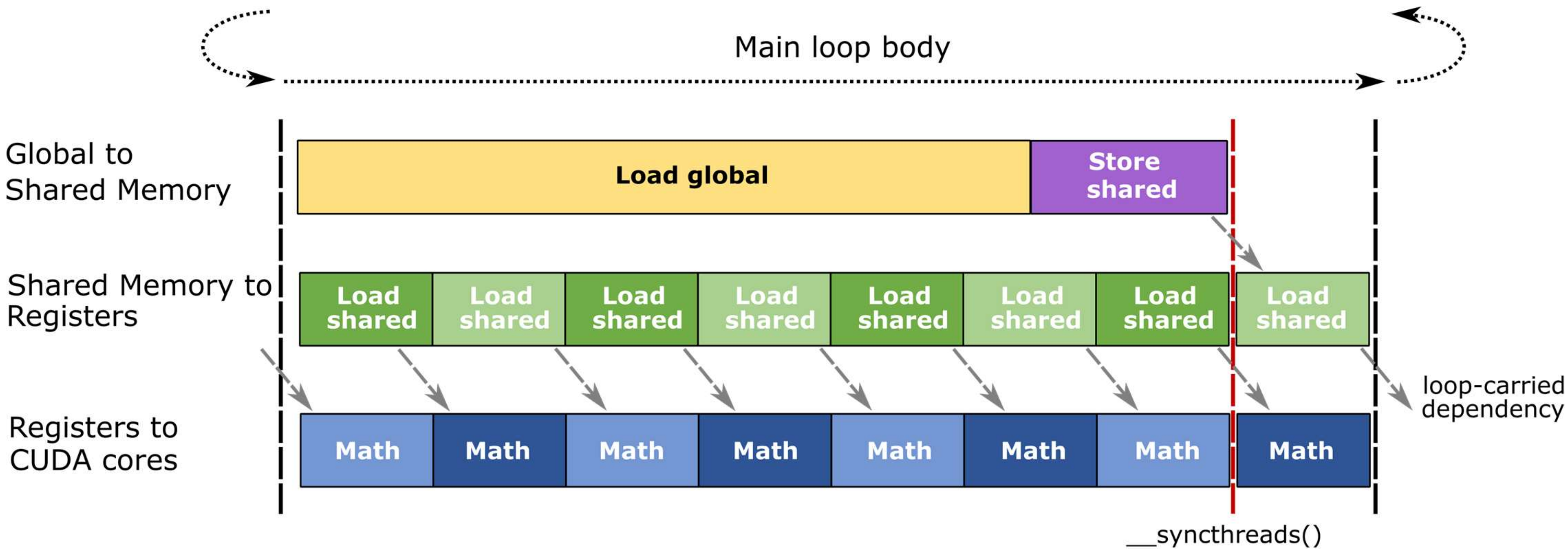


Image takes from https://github.com/NVIDIA/cutlass

Sparse Conv: Fused Native

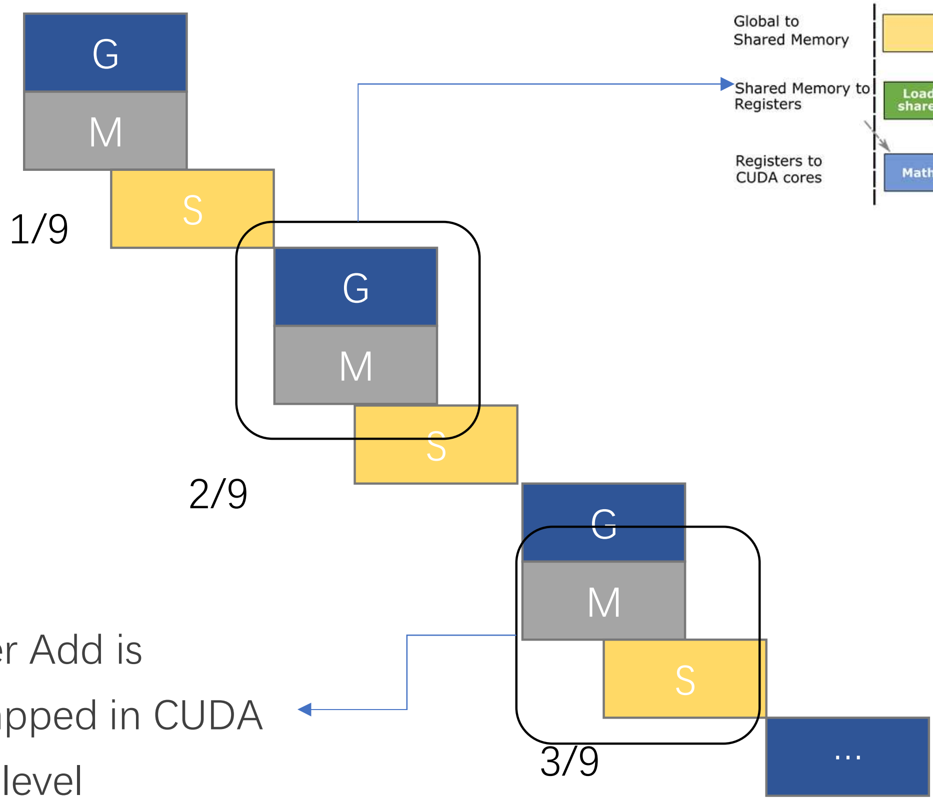
Fused Gather-Gemm-ScatterAdd Pipeline

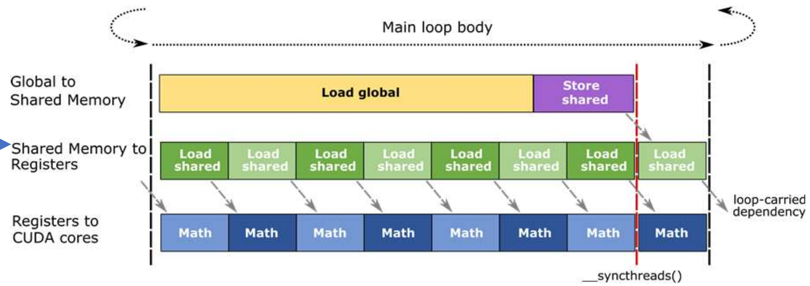
3x3 conv

G: Gather

M: Matmul

S: Scatter

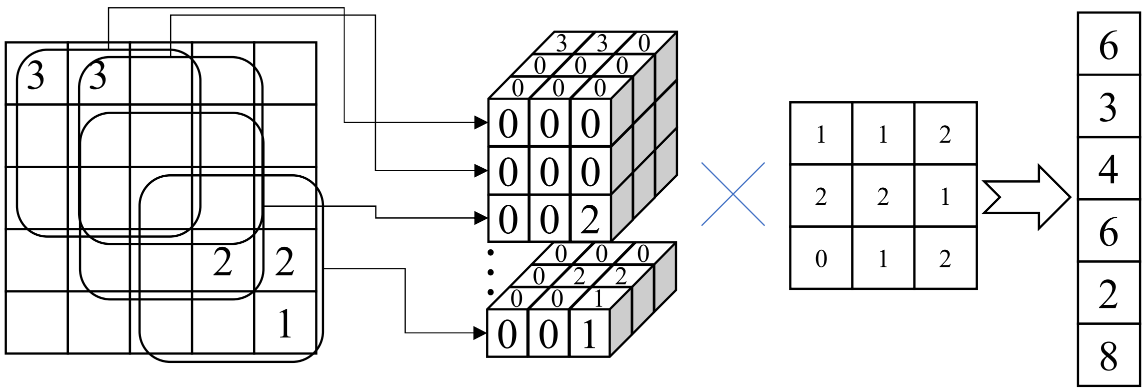




Gather and Gemm is overlapped in CUDA core.

Scatter Add is overlapped in CUDA block level

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)

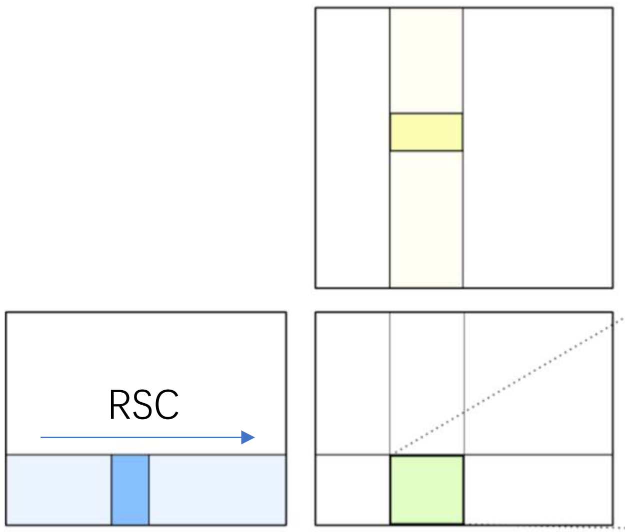
Input: [N, C]

Filters: [K, R, S, C]

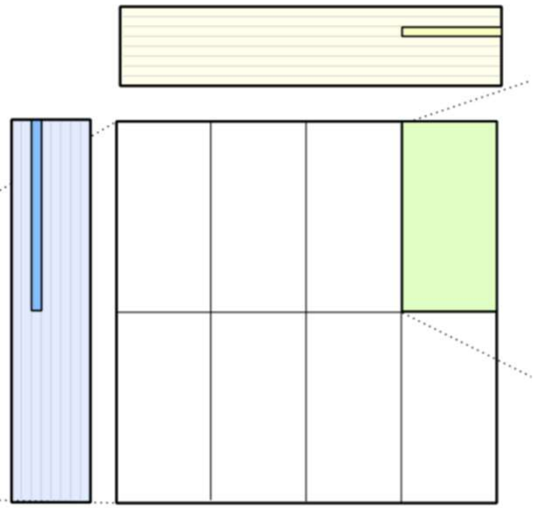
Output: [N, K]

Input

[N, C] => [N, R, S, C] => [N, RSC] @ [K, RSC].T => [N, K]

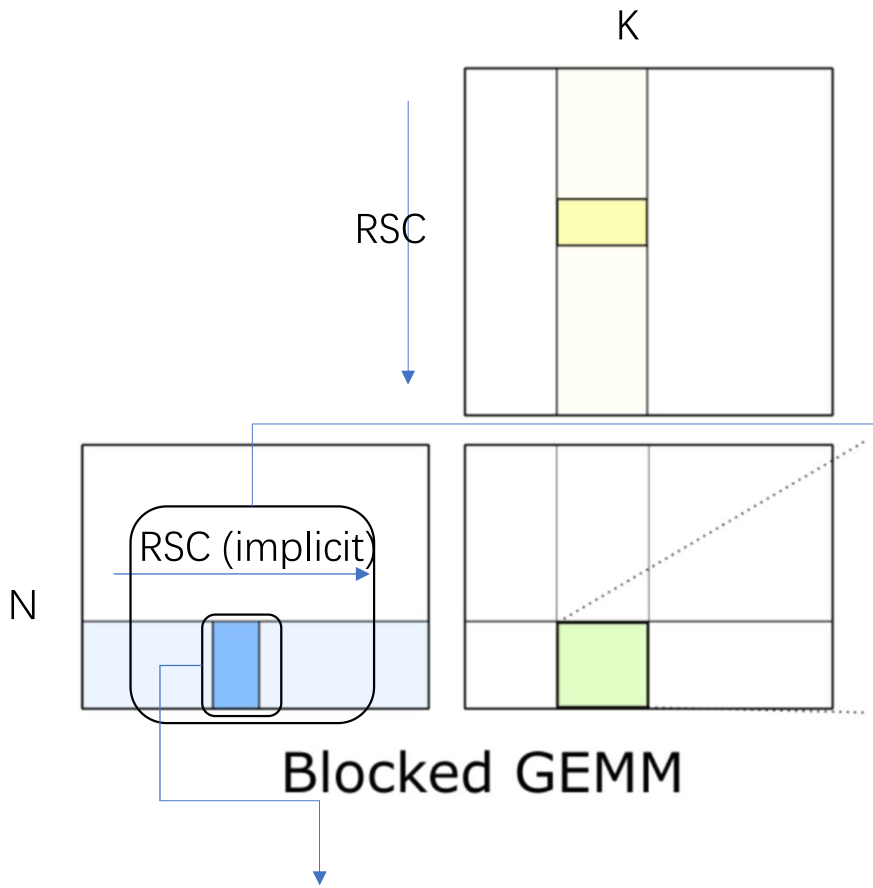


Blocked GEMM



Thread Block Tile

Sparse Conv: Implicit Gemm (1st try)



Tile Size [n, k] = [128, 64]

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]

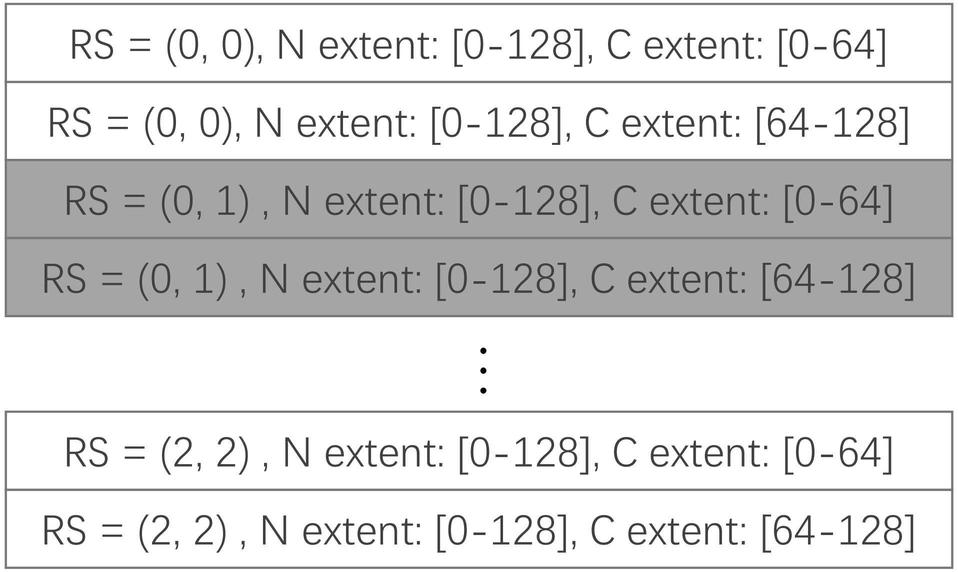
RS = (2, 2) , N extent: [0-128], C extent: [64-128]

We need to read whole I28 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!

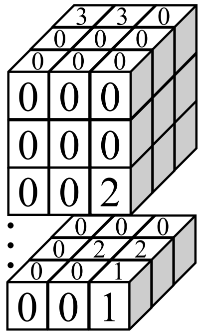
If the whole block is zero, then we can skip them to save compute time.

Sparse Conv: Masked Implicit Gemm

We want to skip white boxes
in figure below.



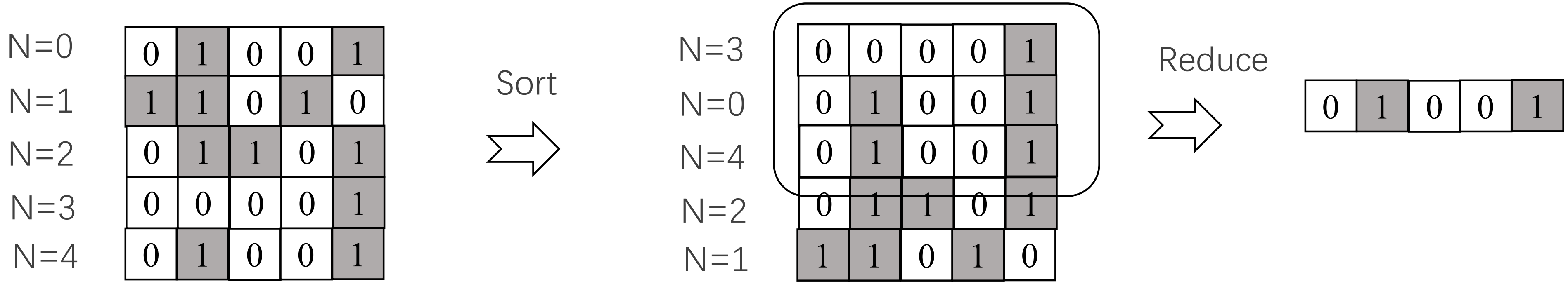
Recall Explicit Gemm Algorithm, for every line (RSC) of input, we already know some RS is zero. But we can't ensure RS in whole block is zero.



Sparse Conv: Masked Implicit Gemm

Sort!!!

Assume Conv1d, Kernel size is 5



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, ¾ 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

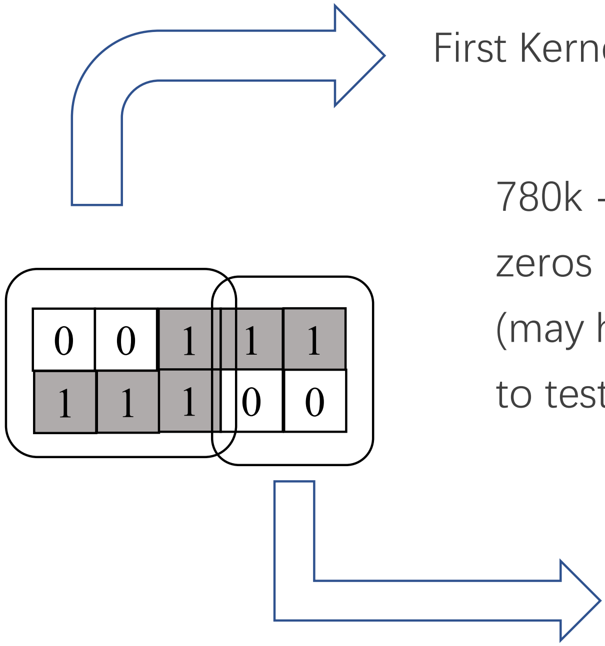
Float 32 is slow because The bottleneck of kernel is float 32 computation! For float32, more zeros, slower running

Sparse Conv: Split-Masked Implicit Gemm

780k -> 1500k, we still calculate almost 100% more zeros in previous algorithm. How to resolve mask perfectly?

Answer: Split Mask to two parts

Sparse Conv: Split-Masked Implicit Gemm



780k -> 970k, only 20% more zeros need to be calculated. (may have data bias here, need to test more data)

Pros

- Minimal Zeros

Cons

- Sort twice
- Generate More Temp data

F32/F16	Native	Implicit Gemm	Implicit Gemm Split Mask
Forward	21.7ms/13.7ms	23.5ms/11.2ms	22ms/12.2ms
Backward	41.9ms/25.2ms	51.0ms/13.8ms	41.1ms/12.2ms

Thanks!