



CUTLASS: Python API, Enhancements, and CUTLASS 3.0 Preview

Andrew Kerr, Cris Cecka | GTC Fall 2022

Acknowledgements

CUTLASS GitHub Community

2.1K stars, 250K clones/month, 50 contributors, and many active users.

Many contributions and PRs from outside of NVIDIA

Integrated into TVM, PyG, and others

CUTLASS Engineers

Andrew Kerr, Haicheng Wu, Cris Cecka, Pradeep Ramani, Aniket Shivam, Vijay Thakkar, Jin Wang, Honghao Lu, Ethan Yan, Shang Zhang, Jack Chen, Petrick Liu, Zhaodong Chen, Yujia Zhai, Jack Kosaian, Dustyn Blasig, Duane Merrill

CUTLASS Product Management

Matthew Nicely, Timothy Costa

Contributors

Vedaanta Agarwalla, Roman Anders, Maximilien Breughe, Naila Farooqui, Manish Gupta, Markus Hohnerbach, Gautam Jain, Alan Kaatz, Wei Liu, Piotr Majcher, Dhiraj Reddy Nallapa, Kyrylo Perelygin, Paul Springer, Pawel Tabaszewski, Chinmay Talegaonkar, John Tran, Yang Xu, Scott Yokim

Acknowledgements

Bing Xu, Leyuan Wang, Masahiro Masuda, Hao Lu, Olivier Giroux, Mostafa Hagog, Bryce Lelbach, Julien Demouth, Joel McCormack, Aartem Belevich, Peter Han, Timmy Liu, Yang Wang, Nich Zhao, Jack Yang, Vicki Wang, Junkai Wu, Ivan Yin, Aditya Alturi, Takuma Yamaguchi, Stephen Jones, Luke Durant, Harun Bayraktar

CUTLASS

CUDA C++ Template Library for Deep Learning and High Performance Computing

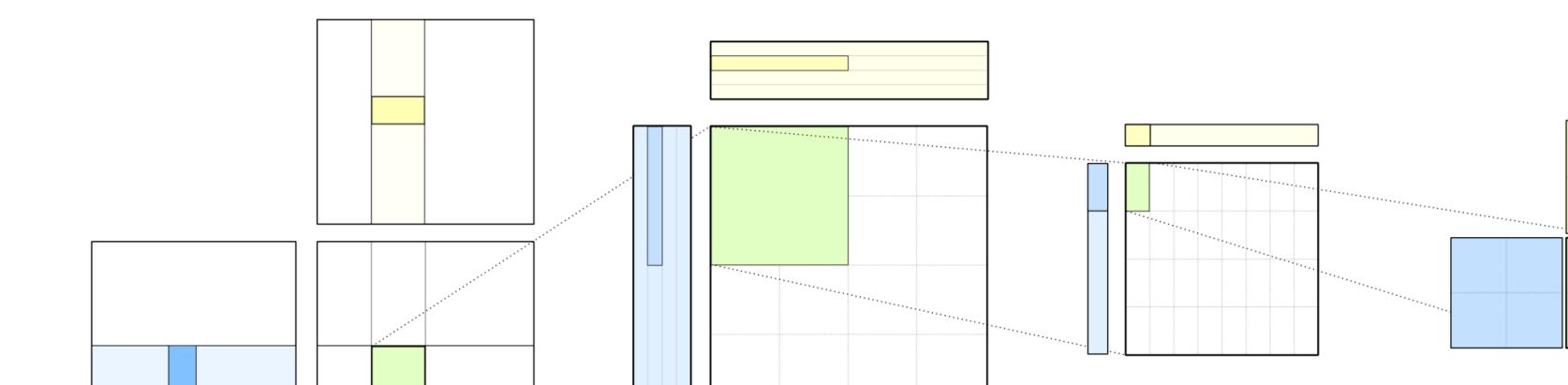
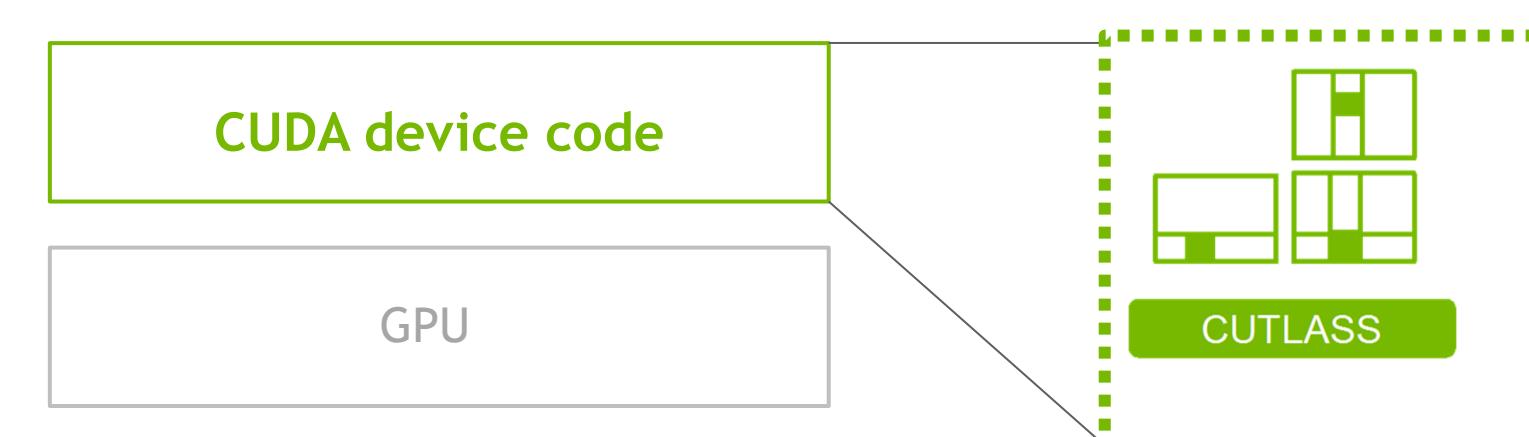


CUTLASS: optimal CUDA C++ templates for matrix computations at all scopes and scales

Device	{ GEMM, Convolution, Reductions } x { all data types } x { SIMT, Tensor Cores } x { all architectures }
Kernel	GEMM, Batched GEMM, Convolution, Reduction, Fused output operations, Fused input operations
Collective	Pipelined Matrix Multiply, Epilogue, Collective access to tensors, Convolution matrix access
Atom	Tensor Core Multiply-Add operations, Efficient access to permuted tensor layouts
Thread	Numeric conversion, <functional> operators on arrays, complex<T>, fast math algorithms
Architecture intrinsic	Templates wrapping architecture-specific PTX instructions (e.g. mma, cp.async, ldmatrix, cvt)

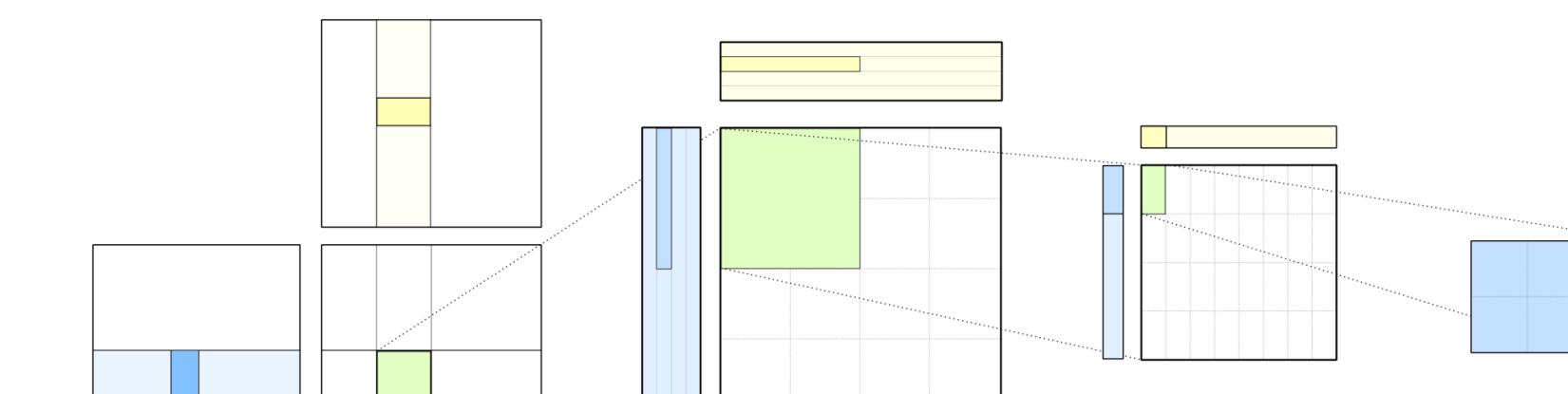
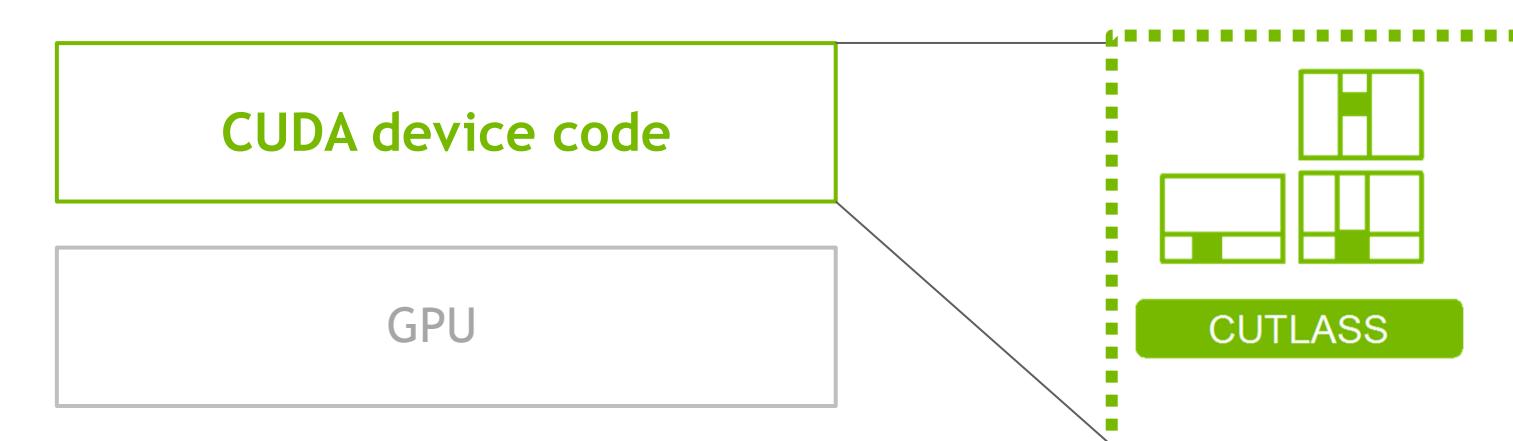
Open source: <https://github.com/NVIDIA/cutlass> (new BSD license)

- Latest revision: CUTLASS 2.11
- Documentation: <https://github.com/NVIDIA/cutlass#documentation>
- Functionality: <https://github.com/NVIDIA/cutlass/blob/master/media/docs/functionality.md>
- Presented: [GTC'18](#), [GTC'19](#), [GTC'20](#), [GTC'21](#), [GTC'22](#)



CUTLASS and CUDA Compiler

- CUTLASS team works closely with NVCC team and NVResearch team to optimize the performance of different DL/HPC kernels.
- Many thanks to Duane Merrill, Malay Sanghi, Howard Chen, Rishkul Kulkarni, Hyun Dok Cho, Fei Peng, Balaji Atukuri, Eddie Gornish, Justin Holewinski, Kartik Haria, Christos Angelopoulos, Xingxing Pan, Jayashree Venkatesh, Xiaohua Zhang, Jerry Zheng, Shekhar Divekar, Cody Addison, Hari Sandanagobalane, Gautam Chakrabarti, Chu-Cheow Lim, Brian Deitrich, Dibyapran Sanyal, Vatsa Santhanam, and many other compiler team members.
- Different versions of NVCCs enable optimizations for different types of kernels
 - 11.3 - Tensor Core GEMMs
 - 11.4 - Tensor Core Implicit GEMMs (Fprop, Dgrad, Wgrad)
 - 11.5 - Sparse Tensor Core GEMMs
 - 11.6 - FP32 Tensor Core Emulation Kernels
 - 11.7 - SIMT kernels; performance variation in Tensor Core kernels
 - **11.8 - Best performance for CUTLASS kernels, particularly Grouped GEMM**





Agenda

- Roadmap
- CUTLASS Python
- CUTLASS 2.x Enhancements
- CUTLASS 3.0 Preview

CUTLASS Roadmap

CUTLASS 2.x and 3.x

CUDA 11.7	CUDA 11.8	CUDA 12.0
CUTLASS 2.10 CUTLASS Python Enhancements: <ul style="list-style-type: none">- Grouped GEMM optimizations- Fused Softmax, Layernorm, and Multihead Attention- Grouped and Depthwise separable convolution	CUTLASS 2.11 NVIDIA Hopper <ul style="list-style-type: none">- 2x FP64 Tensor Cores- FP8 numeric types Stream-K Dynamic Scheduling Algorithm	CUTLASS 3.0 Preview Release 4 th Generation Tensor Cores in CUDA CuTe Programming Model

August

2022

October

November



CUTLASS Python

- Python-based API
- Kernel Fusion
- JIT Compilation and Caching
- Examples and Performance

Why CUTLASS PYTHON?

Bridge the CUTLASS C++ APIs to Python Environment

CUTLASS CUDA C++ Templates

```
using Gemm = cutlass::gemm::device::Gemm<ElementInputA,  
    LayoutInputA,  
    ElementInputB,  
    LayoutInputB,  
    ElementOutput,  
    LayoutOutput,  
    ElementAccumulator,  
    MMAOp,  
    SmArch,  
    ShapeMMAThreadBlock,  
    ShapeMMAWarp,  
    ShapeMMAOp,  
    EpilogueOp,  
    SwizzleThreadBlock,  
    NumStages>;
```



How to construct the template instances ?

How to implement the dispatch layer that picks
the best kernel based on data type, tensor
layout, alignment, and problem sizes?

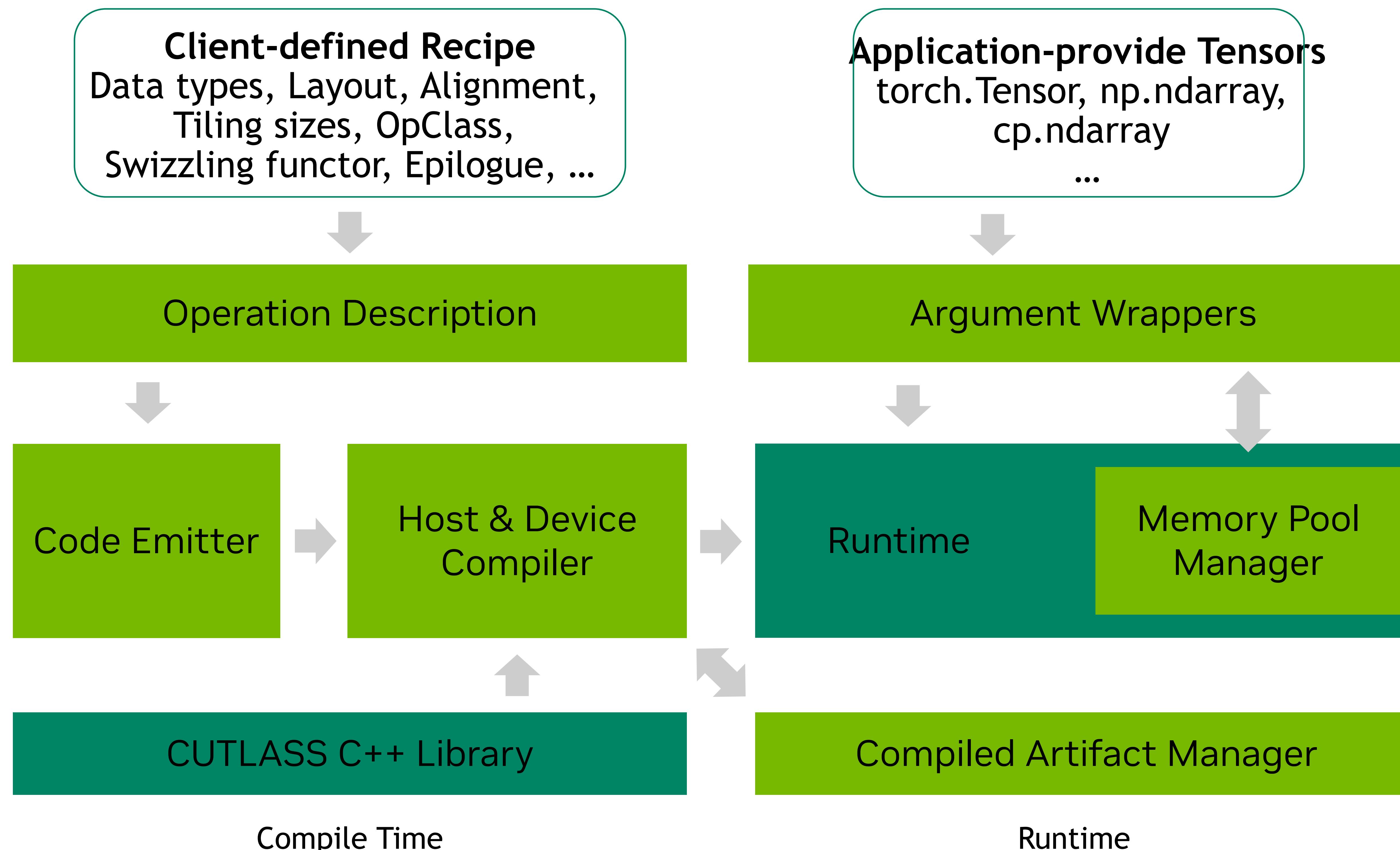
How to feed user environment arguments to
CUTLASS Kernels ?

How to ...

User Environment in Python



CUTLASS Python Architecture



CUTLASS Python Example

Math Instruction

```
1 import pycutlass
2 from pycutlass import *
3 import numpy as np
4
5 # Allocate memory pool
6 pycutlass.get_memory_pool(2**8, 2**32)
7
8 # User-defined Recipe
9 # Data types, layouts, alignments, tiling sizes, opclass,
10 # swizzling functor, epilogu
11 math_inst = MathInstruction(
12     instruction_shape=[16, 8, 16],
13     element_a=cutlass.float16,
14     element_b=cutlass.float16,
15     element_accumulator=cutlass.float32,
16     opcode_class=cutlass.OpClass.TensorOp,
17     math_operation=MathOperation.multiply_add
18 )
19
```

Tile Description

```
20 tile_description = TileDescription(
21     threadblock_shape=[128, 128, 32],
22     stages=3, warp_count=[2, 2, 1],
23     math_instruction=math_inst, min_compute=80, max_compute=80
24 )
```

Tensor Operands

```
25 A = TensorDescription(
26     element=cutlass.float16, layout=cutlass.RowMajor,
27     alignment=8
28 )
```

```
30 B = TensorDescription(
31     element=cutlass.float16, layout=cutlass.RowMajor,
32     alignment=8
33 )
```

```
36 C = TensorDescription(
37     element=cutlass.float32, layout=cutlass.RowMajor,
38     alignment=4
39 )
```

Epilogue Functor

```
41 epilogue_functor = LinearCombination(
42     cutlass.float32, 1, cutlass.float32, cutlass.float32
43 )
```

```
45 # Operation Description
46 operation = GemmOperationUniversal(
47     arch=80, tile_description=tile_description,
48     A=A, B=B, C=C, element_epilogue=cutlass.float32,
49     epilogue_functor=epilogue_functor,
50     swizzling_functor=cutlass.IdentitySwizzle1
51 )
52
53 # Compile the operation
54 pycutlass.compiler.add_module([operation])
55
```

Operation Description

JIT Compilation

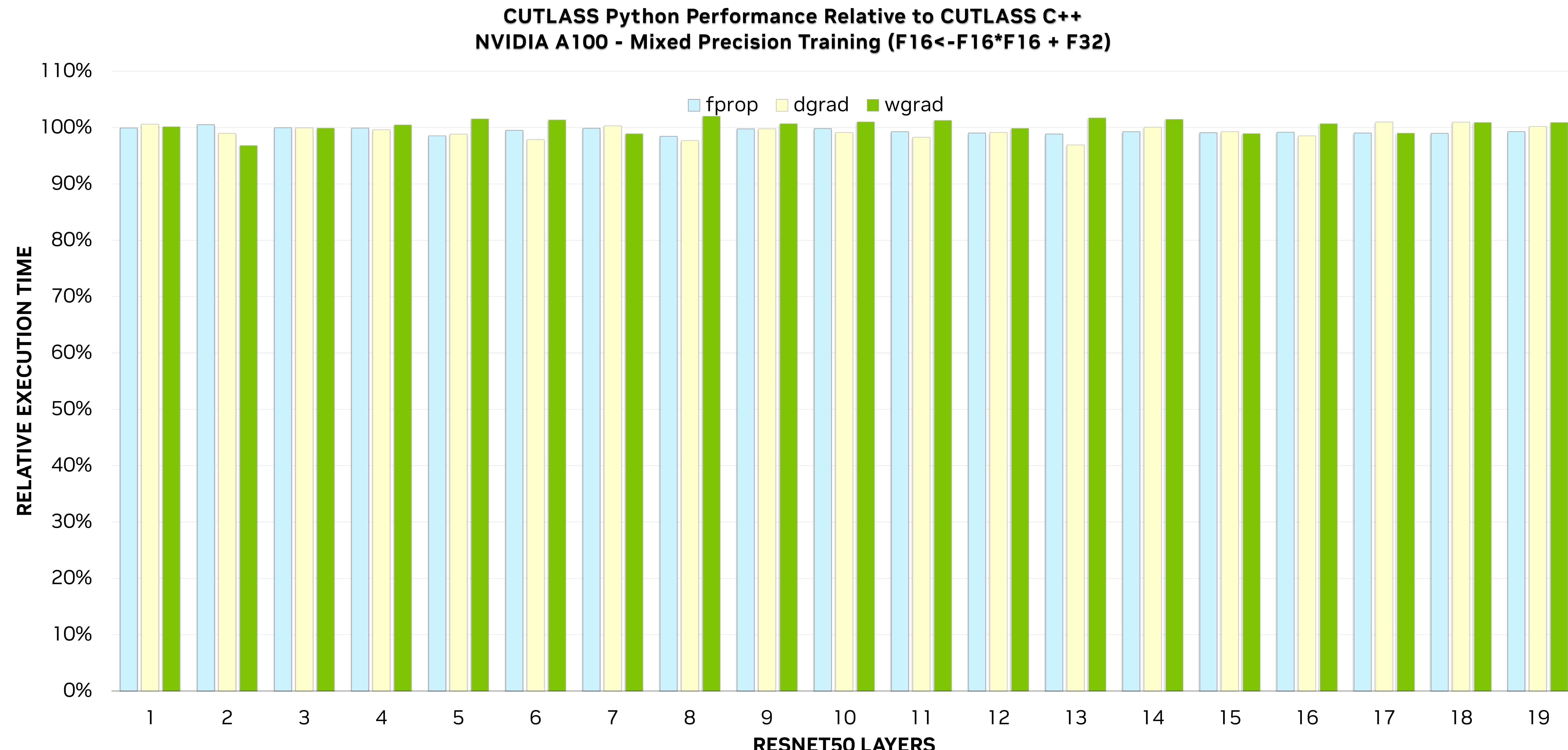
```
57 # User-provide tensors and problem size
58 problem_size = cutlass.gemm.GemmCoord(512, 256, 128)
59
60 tensor_A = np.random.randn(512, 128).astype(np.float16)
61 tensor_B = np.random.randn(128, 256).astype(np.float16)
62 tensor_C = np.random.randn(512, 256).astype(np.float32)
63 tensor_D = np.empty_like(tensor_C)
64
65 # Argument Wrapper
66 arguments = GemmArguments(
67     operation=operation, problem_size=problem_size,
68     A=tensor_A, B=tensor_B, C=tensor_C, D=tensor_D,
69     output_op = operation.epilogue_type(1.0, 0.0),
70     gemm_mode=cutlass.gemm.Mode.Gemm
71 )
72
73 # Runtime
74 operation.run(arguments)
75 arguments.sync()
```

User-provided Tensors
torch.Tensor, np.ndarray,
cp.ndarray ...

Launch

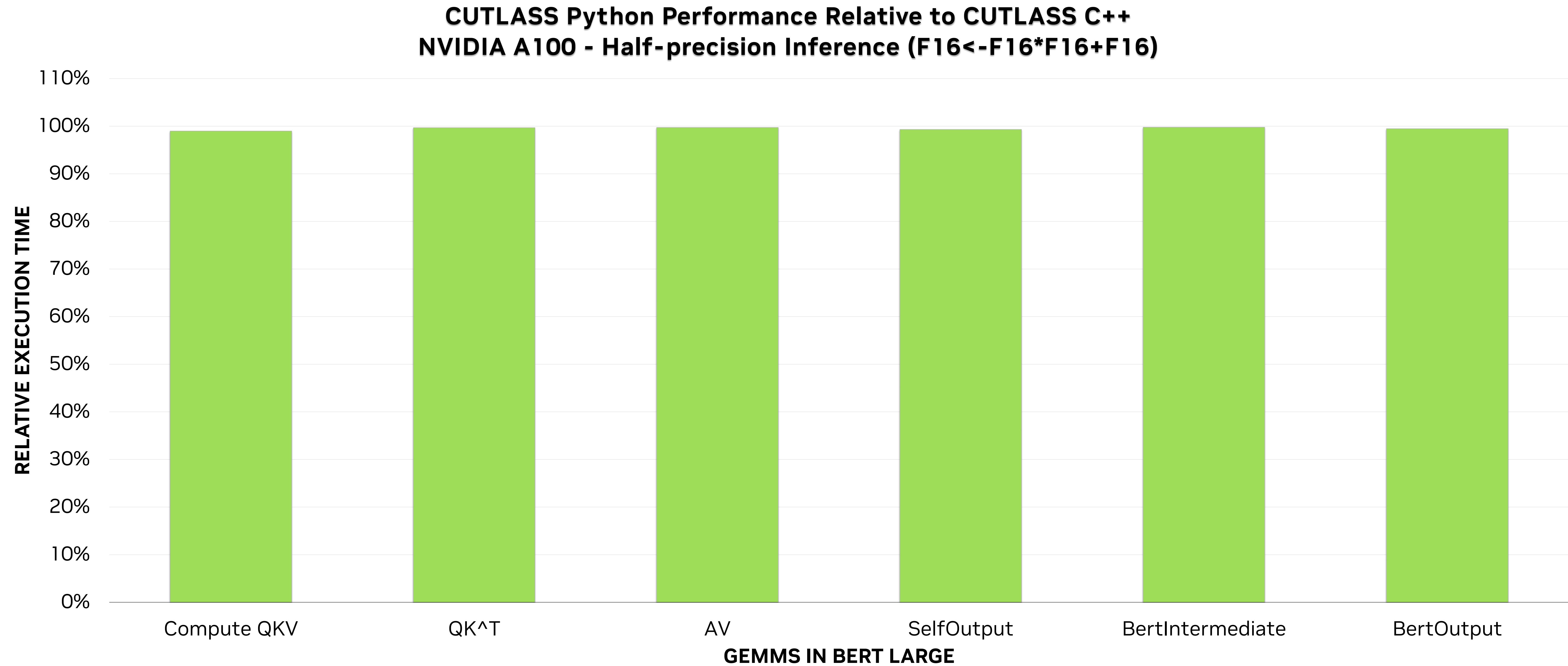
CUTLASS Python Performance compared with CUTLASS C++

Conv2D



CUTLASS Python Performance compared with CUTLASS C++

GEMM



CUTLASS PYTHON

Expose the rich fusion recipes in CUTLASS to Python

Phase	Pattern	Use-case
Mainloop	Elementwise compute	Conversion, scaling, masking
	Reduction of A or B	GEMM + bias grad
	Broadcast	Load vector and broadcast over channels
Epilogue	Multiplicity of tensor operands	GEMM + GELU with Aux tensor; GEMM + RELU with output bitmask GEMM + dRELU loading bitmask
	Multiplicity of vector operands	Alpha and beta scaling as vector
	Broadcast over column	Bias add
	Reduce over column	GEMM + bias Grad Layer norm
Composition	Reduce over row	GEMM + Softmax
	Elementwise compute	Arithmetic, conversion, activation functions
Composition	Back-to-back GEMM	
	Back-to-back CONV	



CUTLASS 2.x Enhancements

- GEMM Fusion: Permutation, Fused Softmax, Layer Norm, and Multihead Attention
- Convolution: Grouped and Depthwise Separable
- Stream-K
- NVIDIA Hopper Enablement

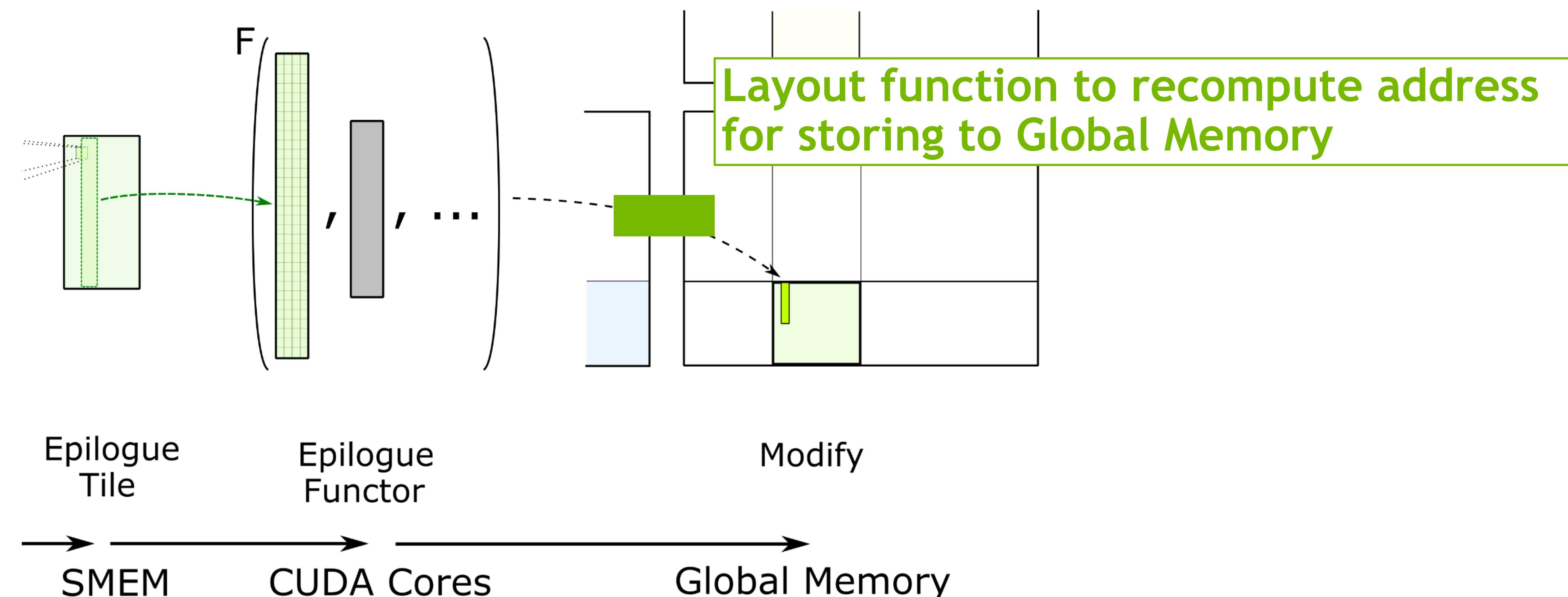
GEMM Permute

CUTLASS Implementation and Layout Function

- GEMM followed by layout transformations are common in DL applications

```
# BERT self attention
context_layer = torch.matmul(attention_probs, value_layer)
context_layer = context_layer.permute(0, 2, 1, 3).contiguous()
```

- In CUTLASS, GEMM epilogue exchanges data through Shared Memory then cooperatively access Global Memory using efficient striped access patterns.
- In GEMM epilogue, we provided **Layout plugin** (include/cutlass/layout/permute.h) as permutations on CUTLASS layouts
- May be composed with existing Global Memory address computation to reorder data in memory



GEMM Permute

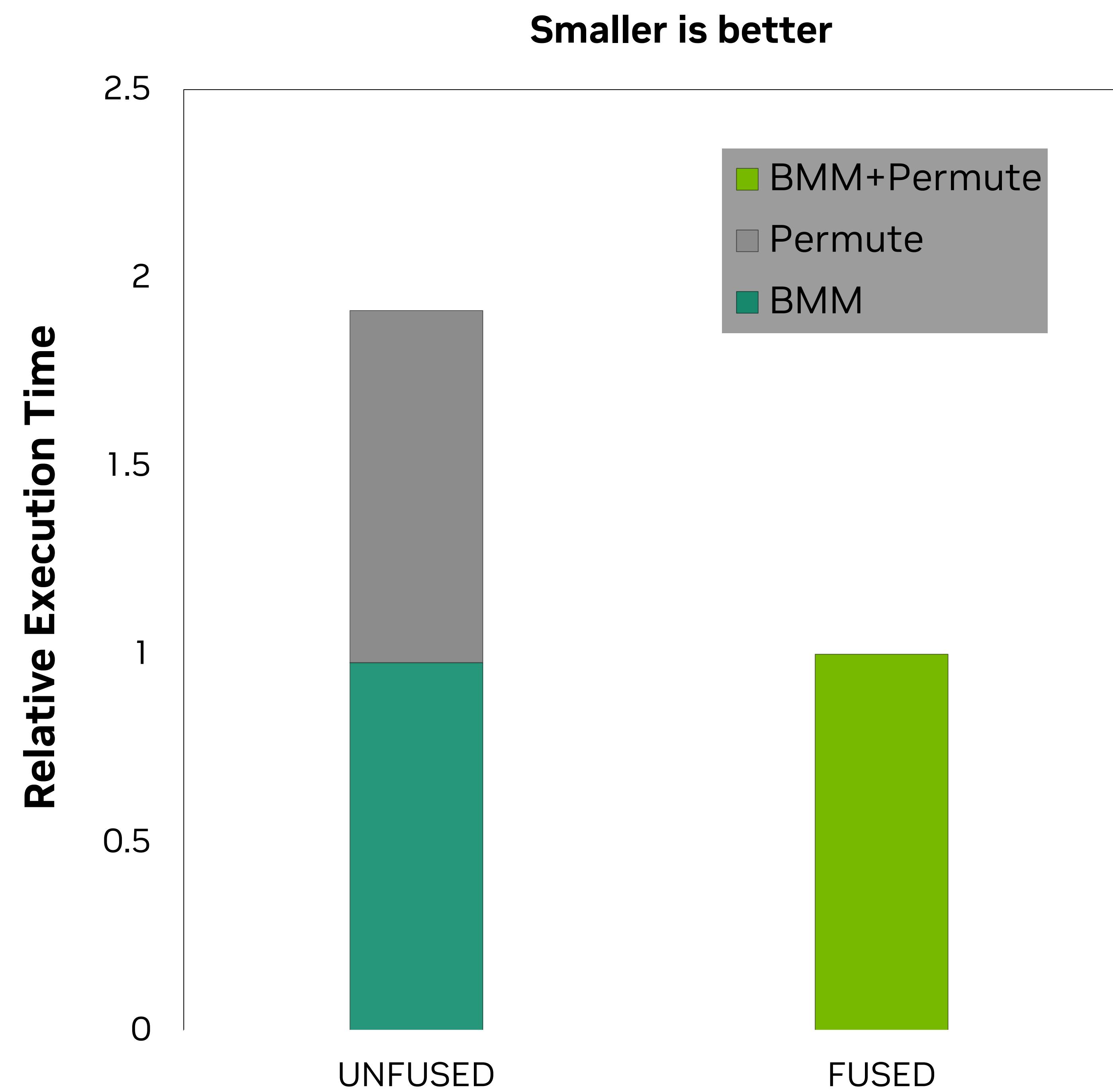
Performance

Experiment: NVIDIA A100

- BatchedGEMM size: --batch-count=1536 --m=128 --n=64 --k=128
- output tensor shape: [128*12, 128, 64]
- reshaped as: [128, 12, 128, 64]
- finally permuted as: [128, **128**, **12**, 64]

fused	unfused
BMM + Permute: 0.09789 ms	BMM: 0.09569 ms
	Permute: 0.09159 ms
Total: 0.09789 ms	0.1873 ms

Speedup: 1.91x



Fused Layer Normalization

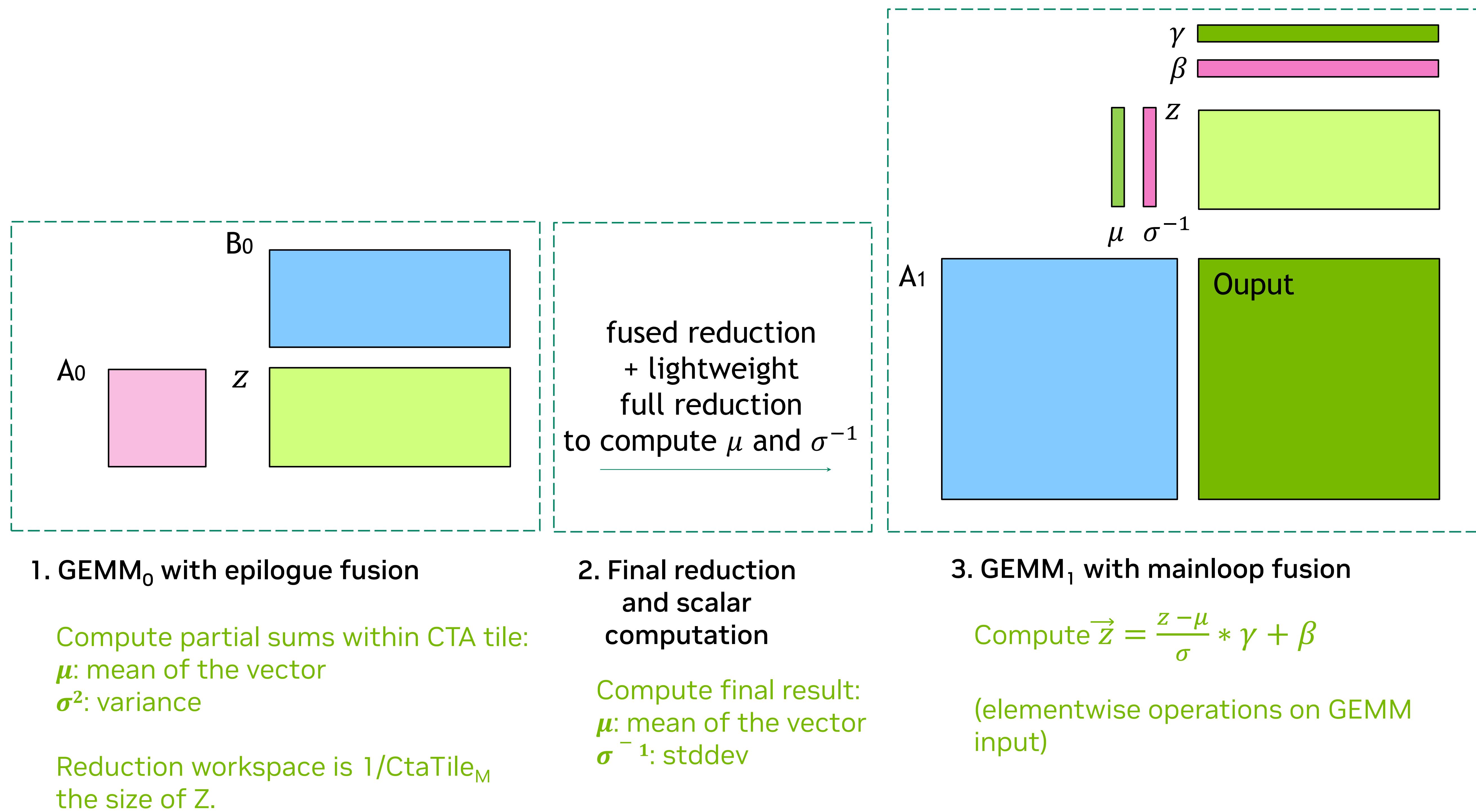
GEMM₀ → Normalization → GEMM₁

- Layer Normalization:

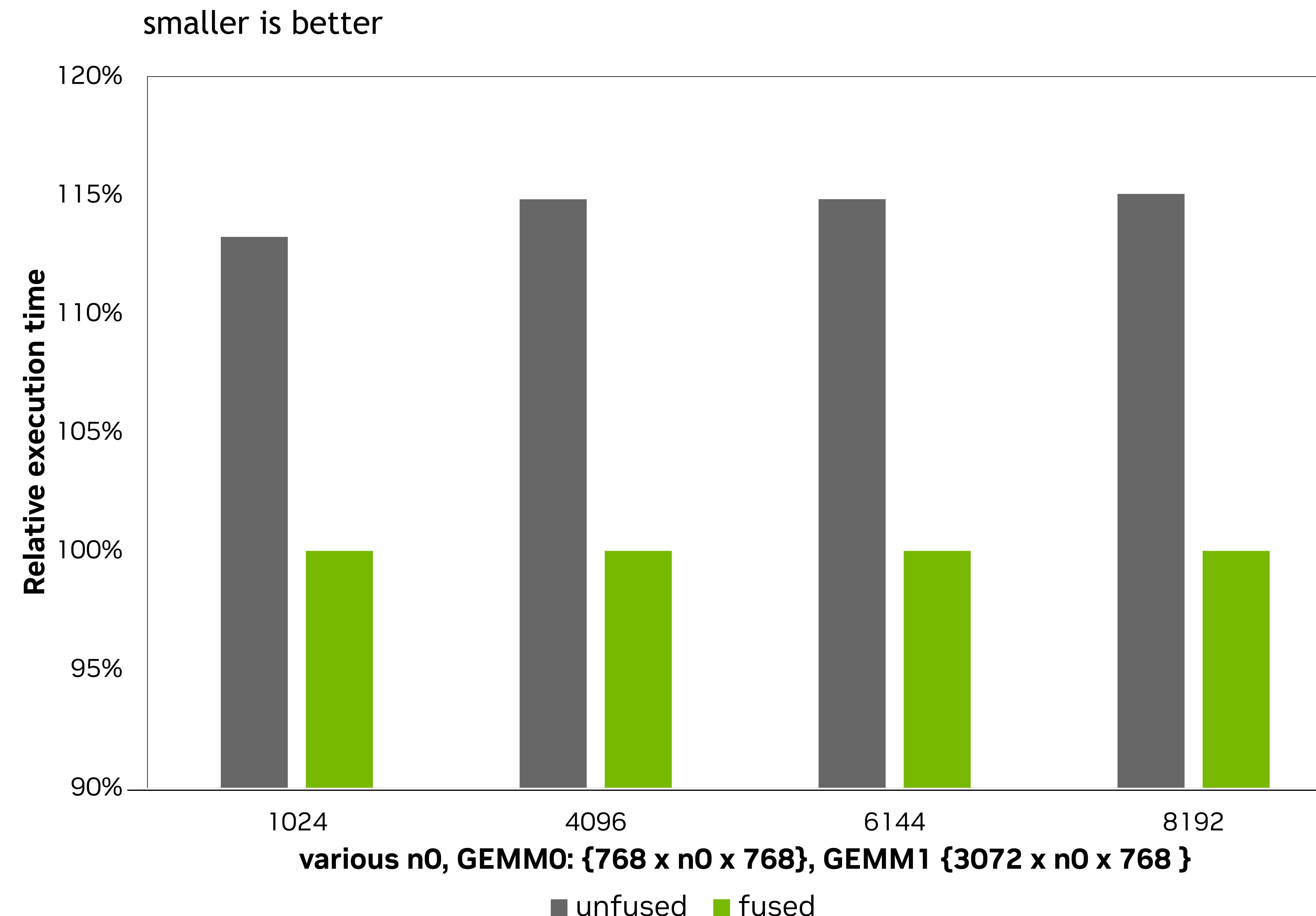
$$\vec{z} = \frac{z - \mu}{\sigma} * \gamma + \beta$$

- \vec{z} : an input/output vector
- μ : mean of the vector
- σ : standard deviation
- γ : a scale vector
- β : a bias vector

Normalization can be decomposed and fused with GEMM layers



CUTLASS Fully-Fused Layer Normalization Example Benchmark



Fused Softmax

- Softmax with max trick:

$$\sigma(\vec{z})_i = \frac{e^{z_i - \max}}{\sum_j^K e^{z_j - \max}}$$

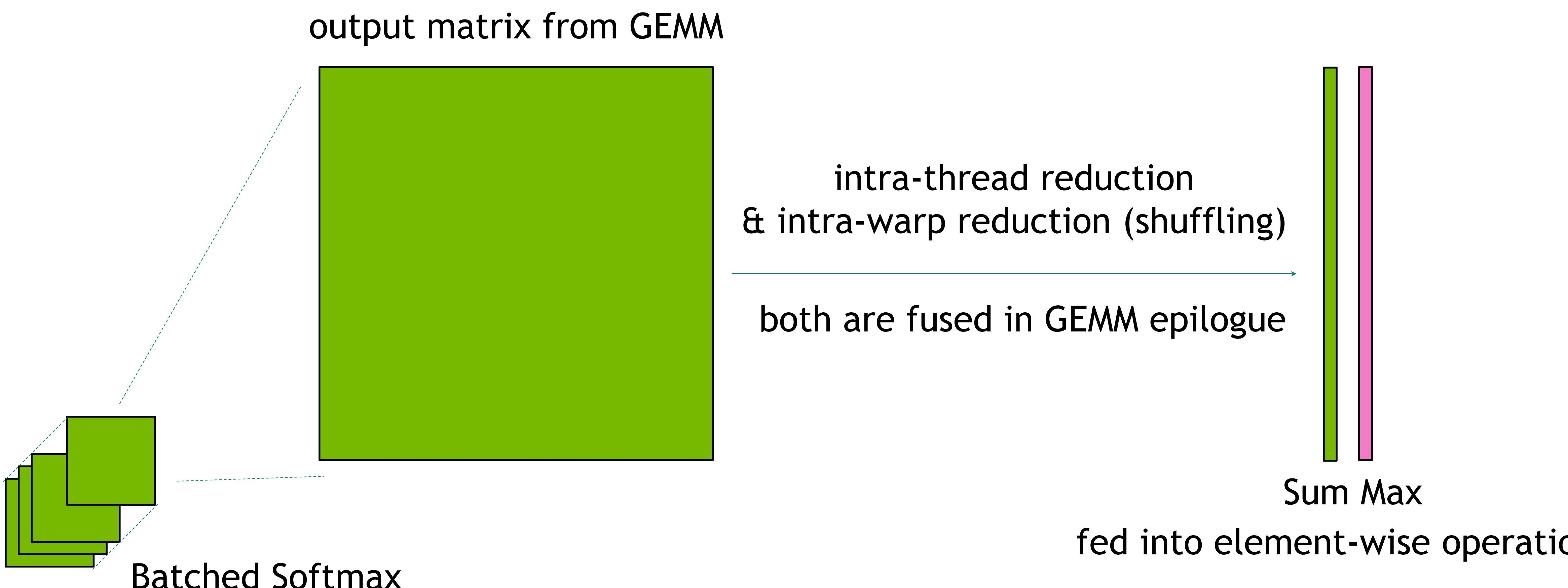
Reductions:

1. Max
2. Sum

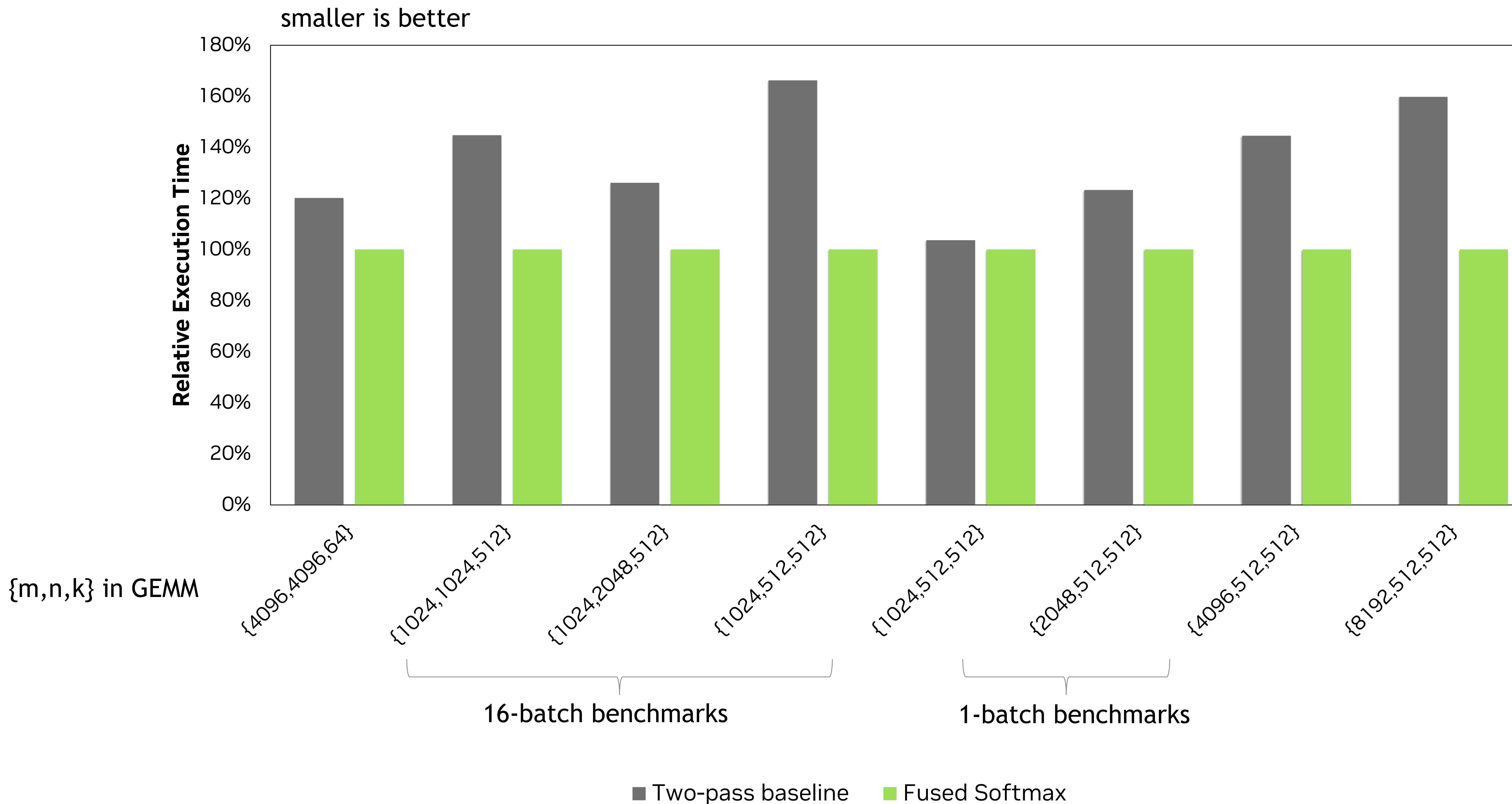
Elementwise:

1. Exponential
2. Scale

- $\sigma(\vec{z})_i$: the i^{th} output softmax vector
- \vec{z} : an input vector
- e^{z_i} : standard exponential function for input vector
- \max : the maximal of input vector



CUTLASS Softmax Example with Batch Support Benchmark



Fused Multihead Attention (MHA)

- Multihead Attention:

$$P = \text{scalar} * Q * K$$

$$P = \text{softmax}(P)$$

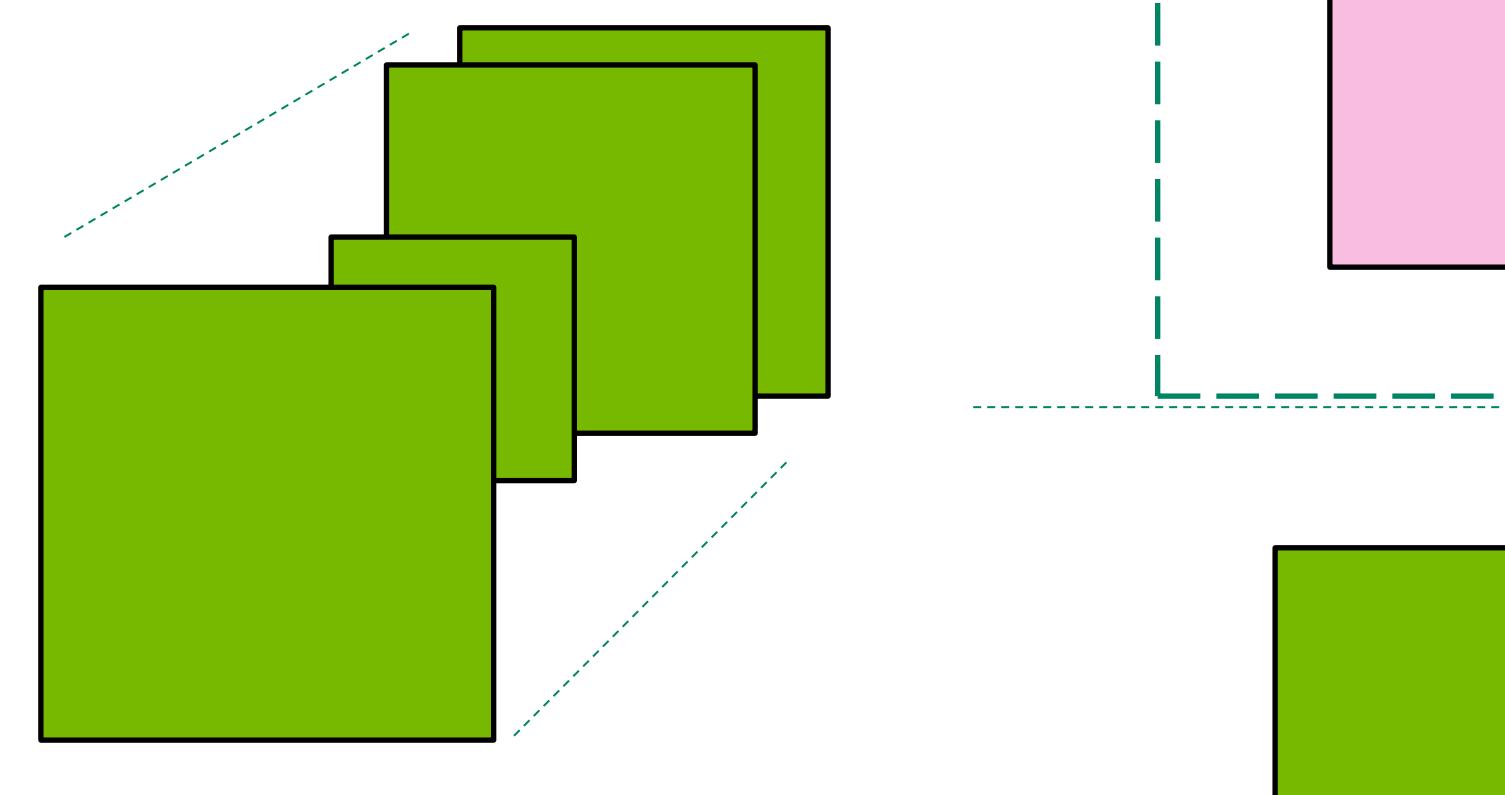
$$O = P * V$$

BS: batch size

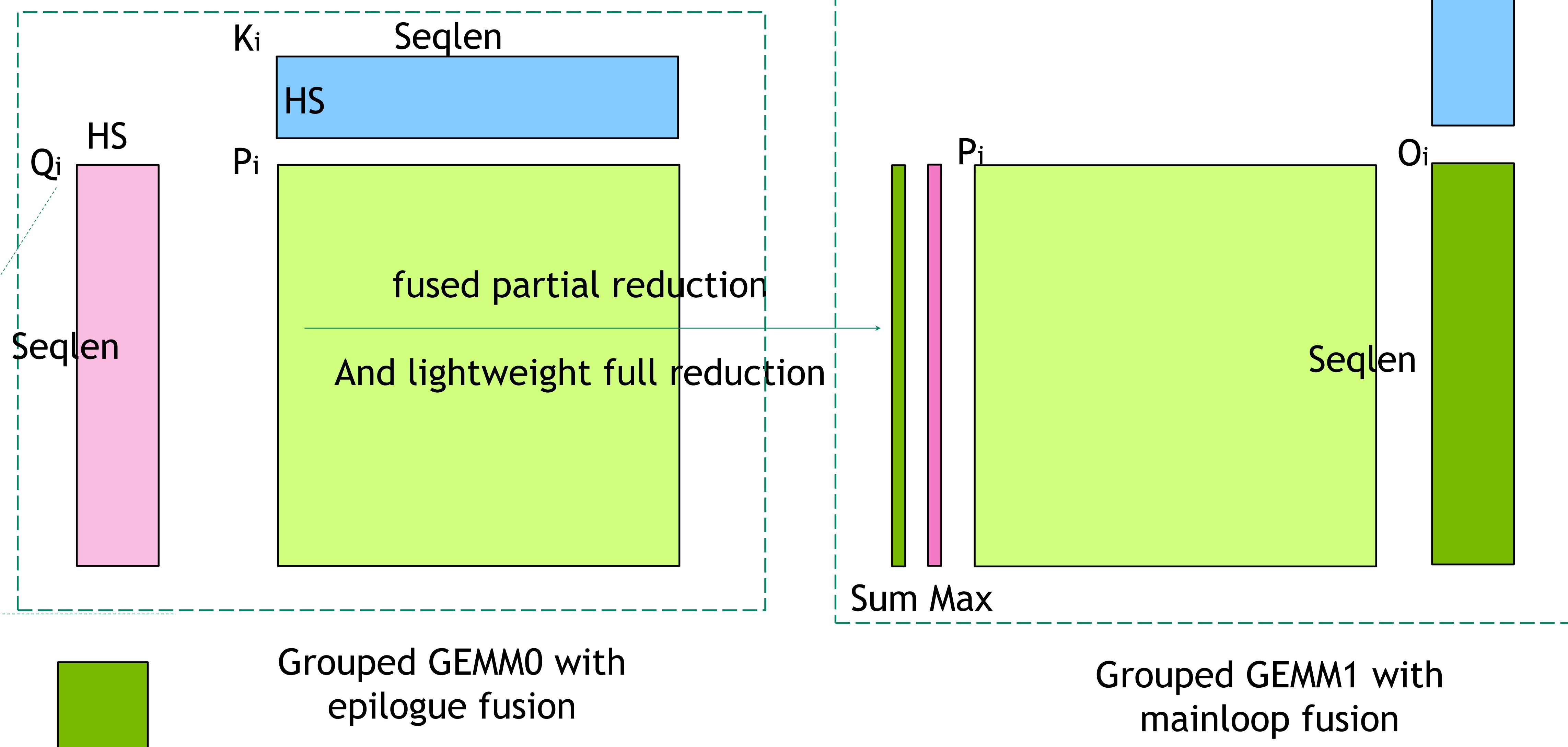
HS: head size

HN: head number

Seqlen: sequence length

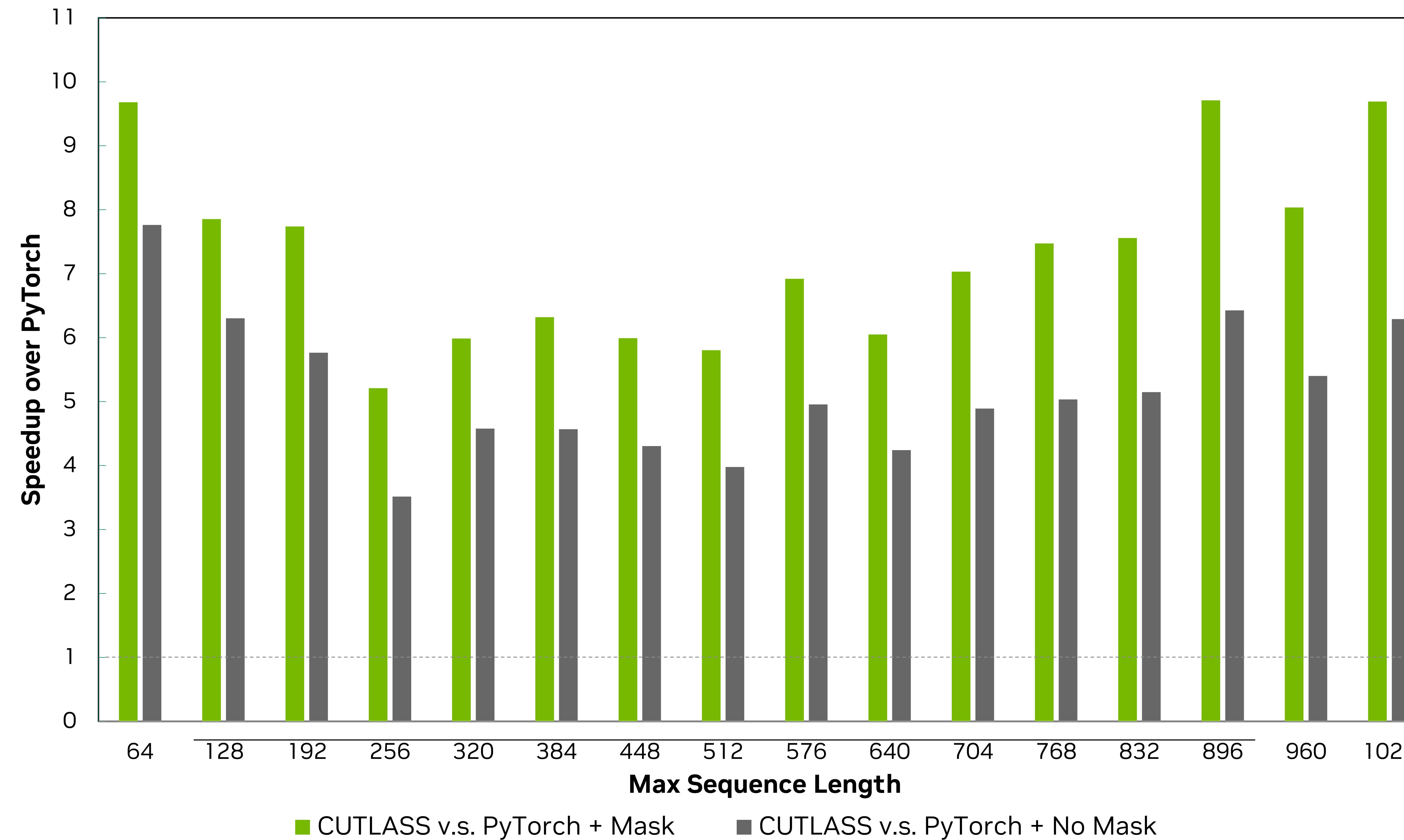


Fused MHA CUTLASS example With Grouped GEMM



CUTLASS Fused Multihead Attention Example Benchmark

~6.25X faster than PyTorch in average!





CUTLASS 2.x Enhancements

- Fused Softmax, Layer Norm, Permutation, and Multihead Attention
- Convolution: Grouped and Depthwise Separable
- Stream-K
- NVIDIA Hopper Enablement

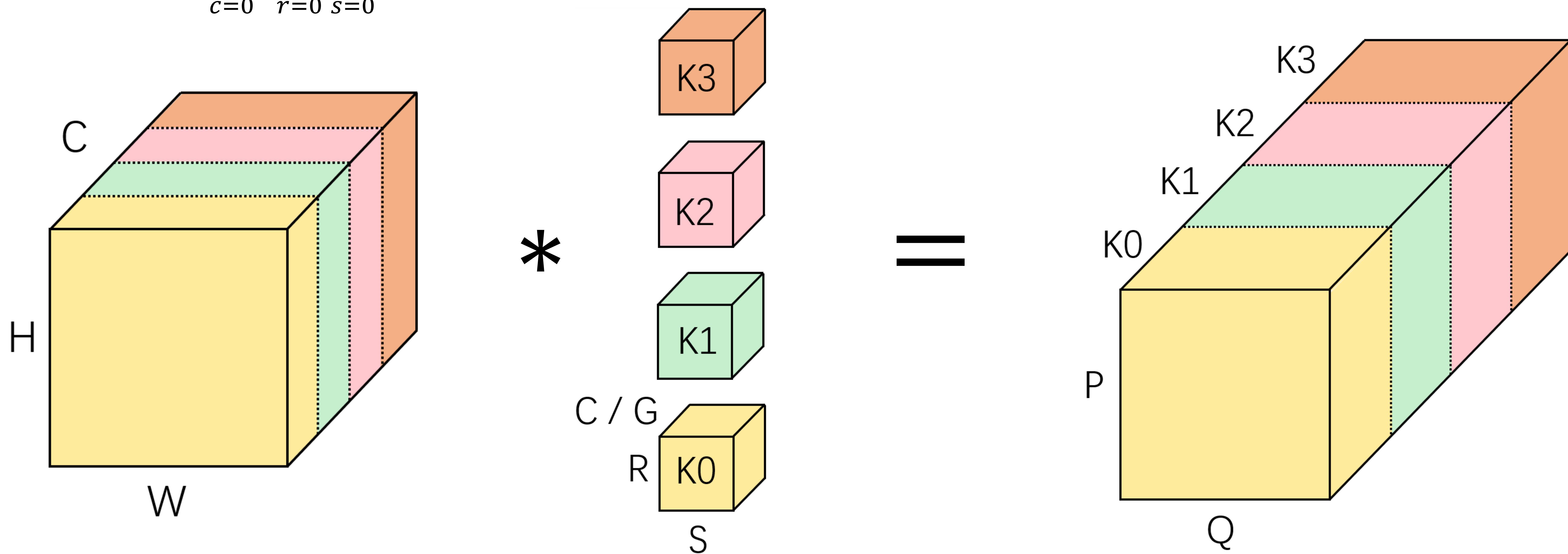
Grouped Convolution

Grouping filters to decrease complexity of convolution layer.

Each filter group will only convolve with a group of image channels.

`filter.channels = image.channels / groups`

$$y[n, p, q, k] = \sum_{c=0}^{\frac{C}{g}-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} (x[n, \bar{h}(p, r), \bar{w}(q, s), c * g + k] w[k, r, s, c])$$



An example of grouped convolution (groups = 4)

CUTLASS Grouped Convolution Implementation

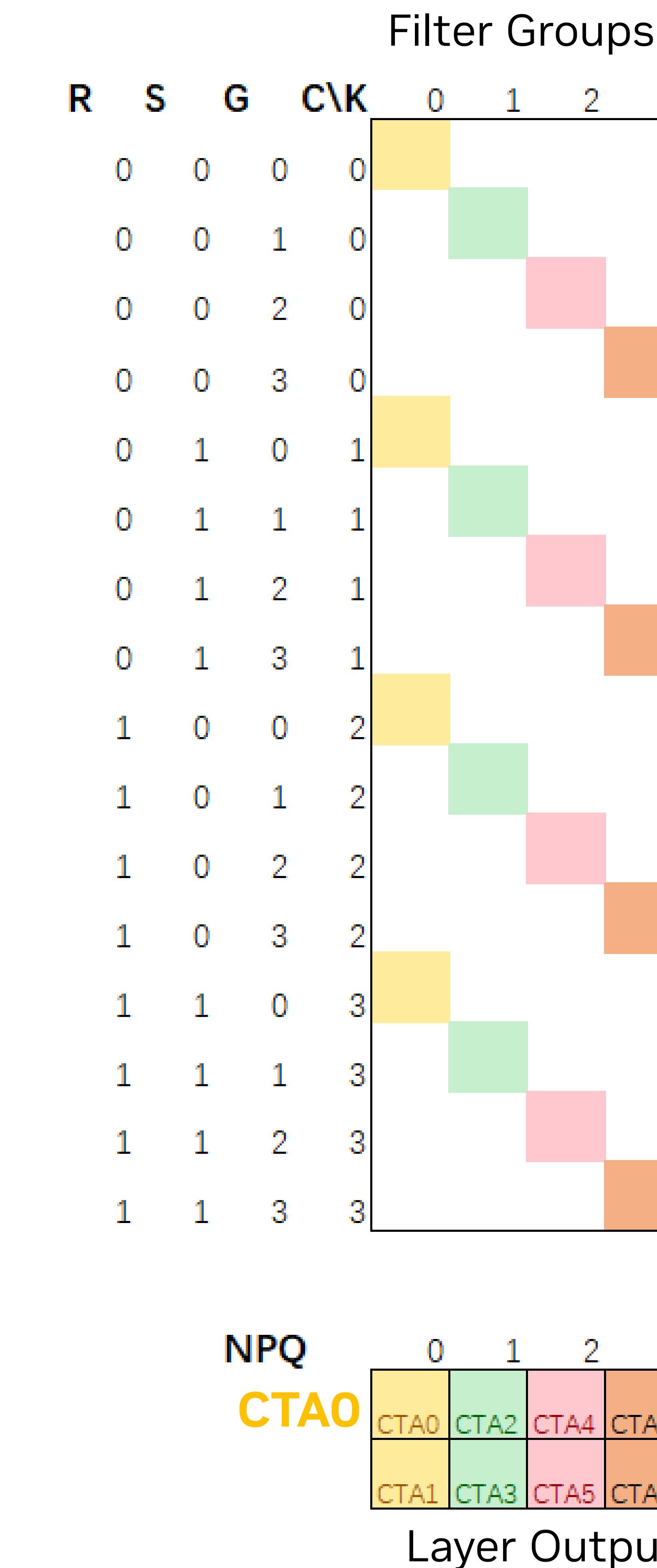
- Induces a block sparse structure in Implicit GEMM formulation
- C dimension striped over the number of groups
 - Workload within one CTA decimated by group count

**RxS: 2-by-2
Groups = 4**

	R	0	0	0	0	0	0	0	1	1	1	1	1	1	1		
	S	0	0	0	0	1	1	1	0	0	0	1	1	1	1		
	G	0	1	2	3	0	1	2	3	0	1	2	3	0	1		
NHW	C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

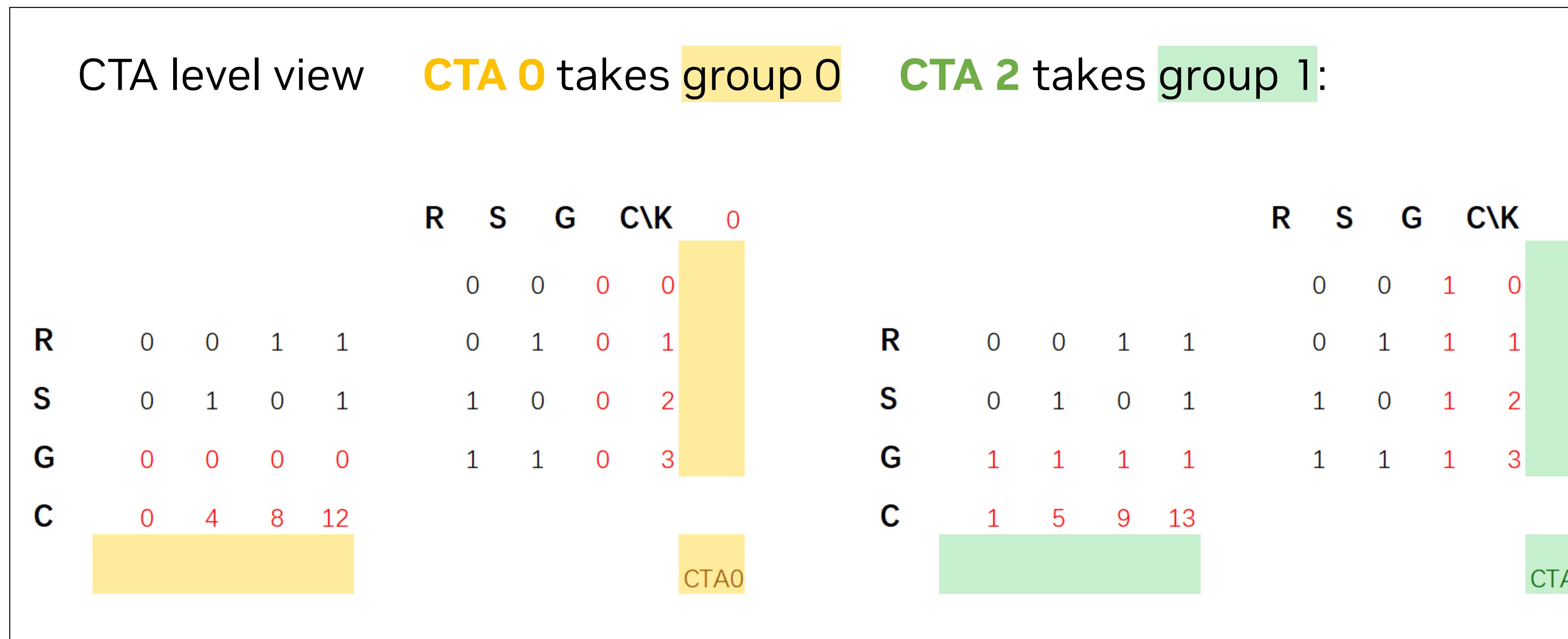
Activations

R * S * C

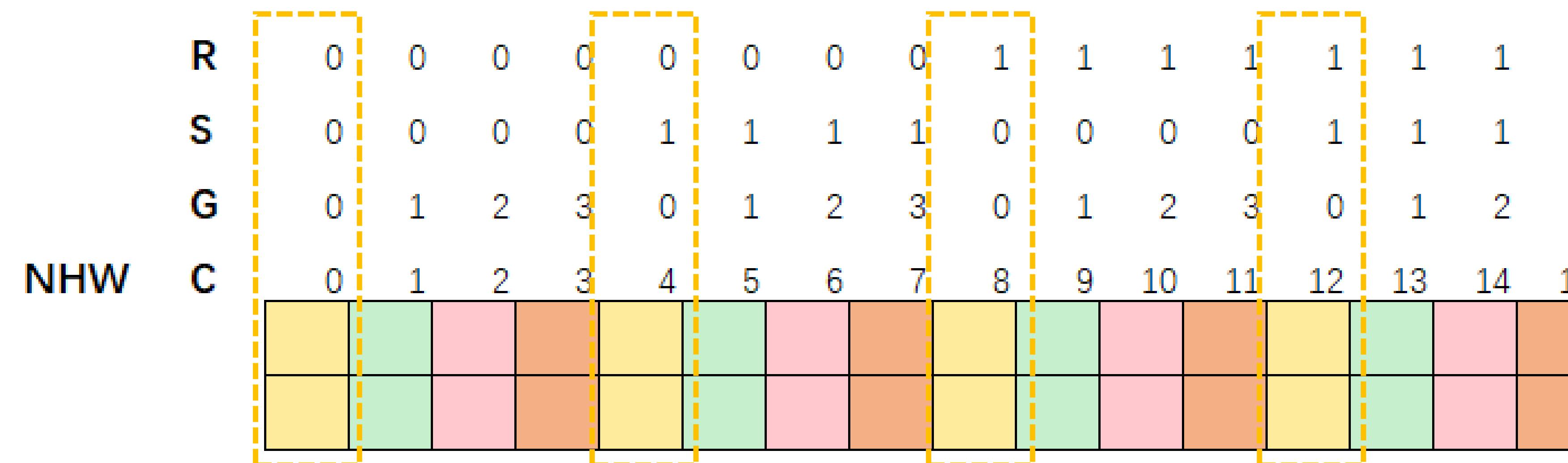


CUTLASS Grouped Convolution Implementation

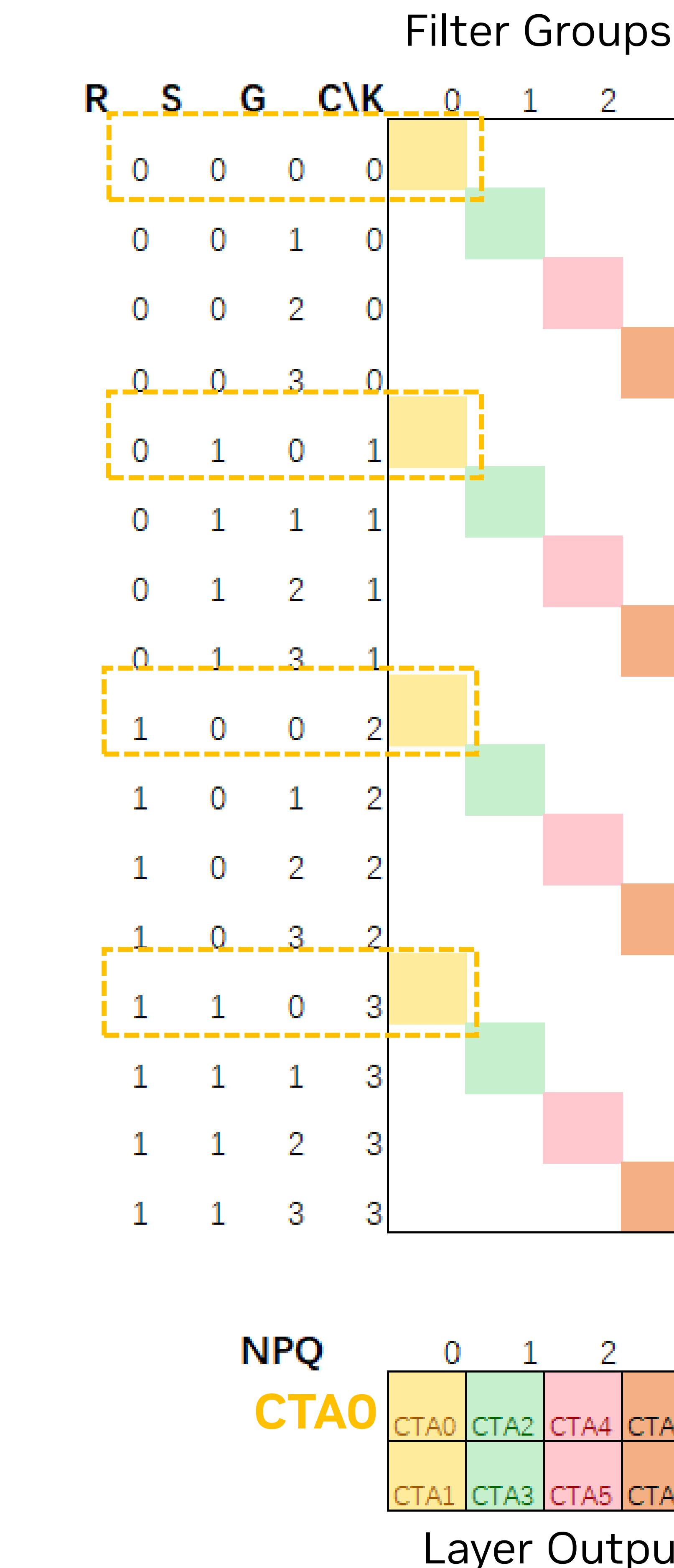
`cutlass::conv::kSingleGroup` mode: 1 CTA calculates 1 group



**RxS: 2-by-2
Groups = 4**

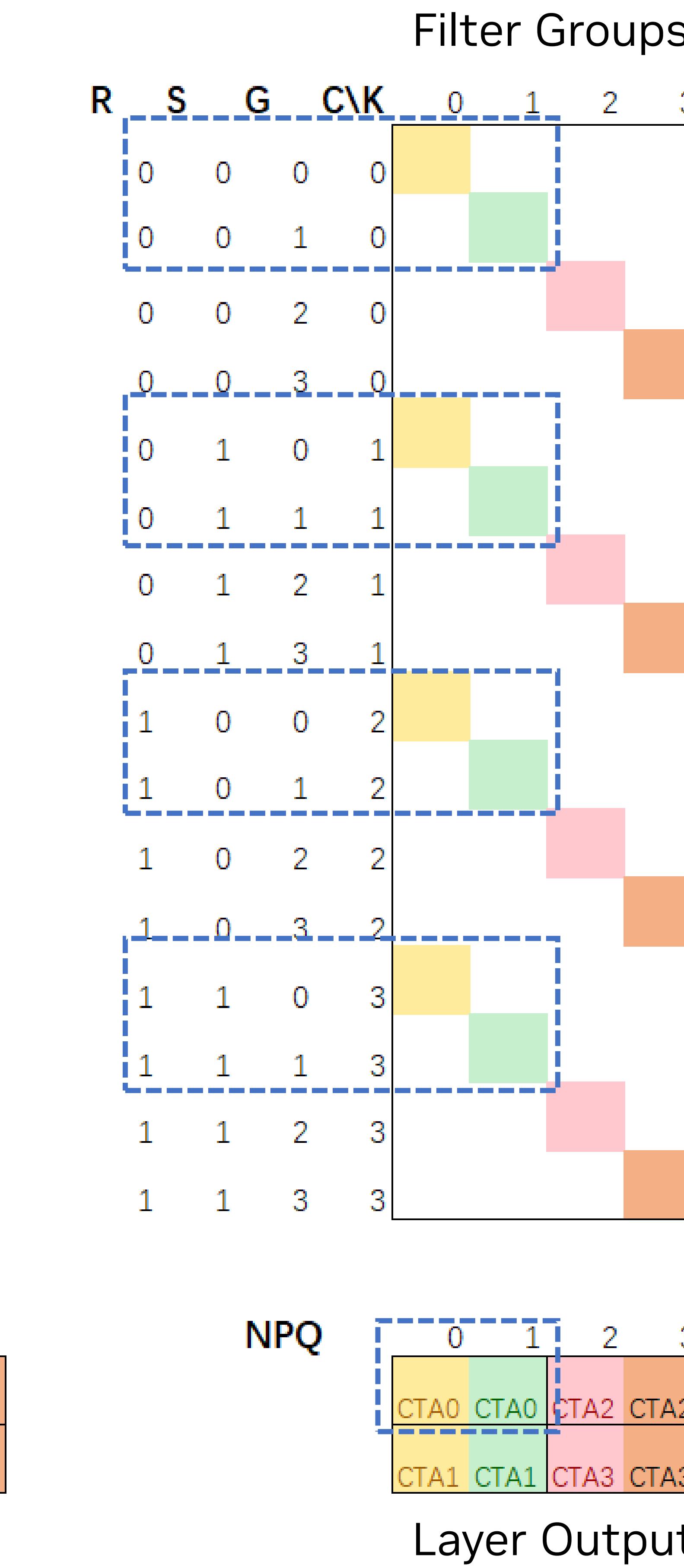
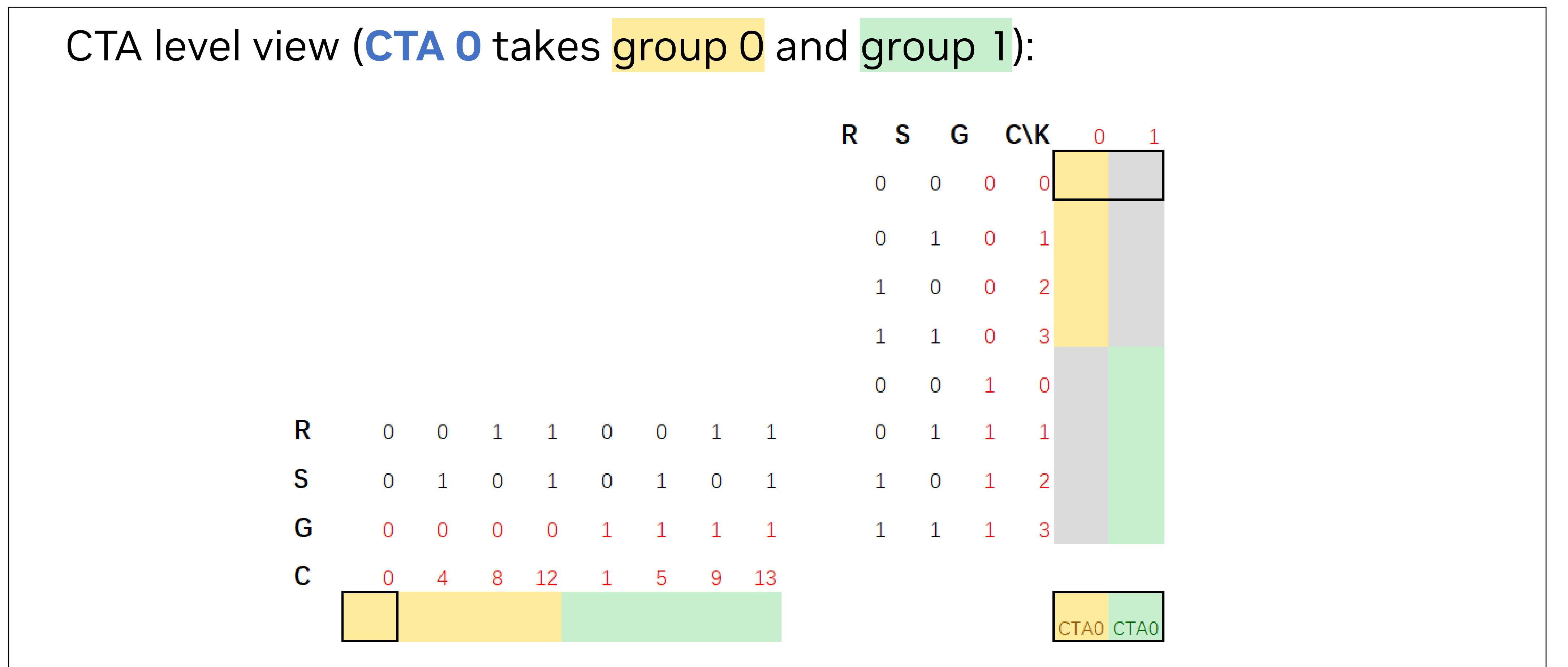


4x reduction in mainloop iteration count

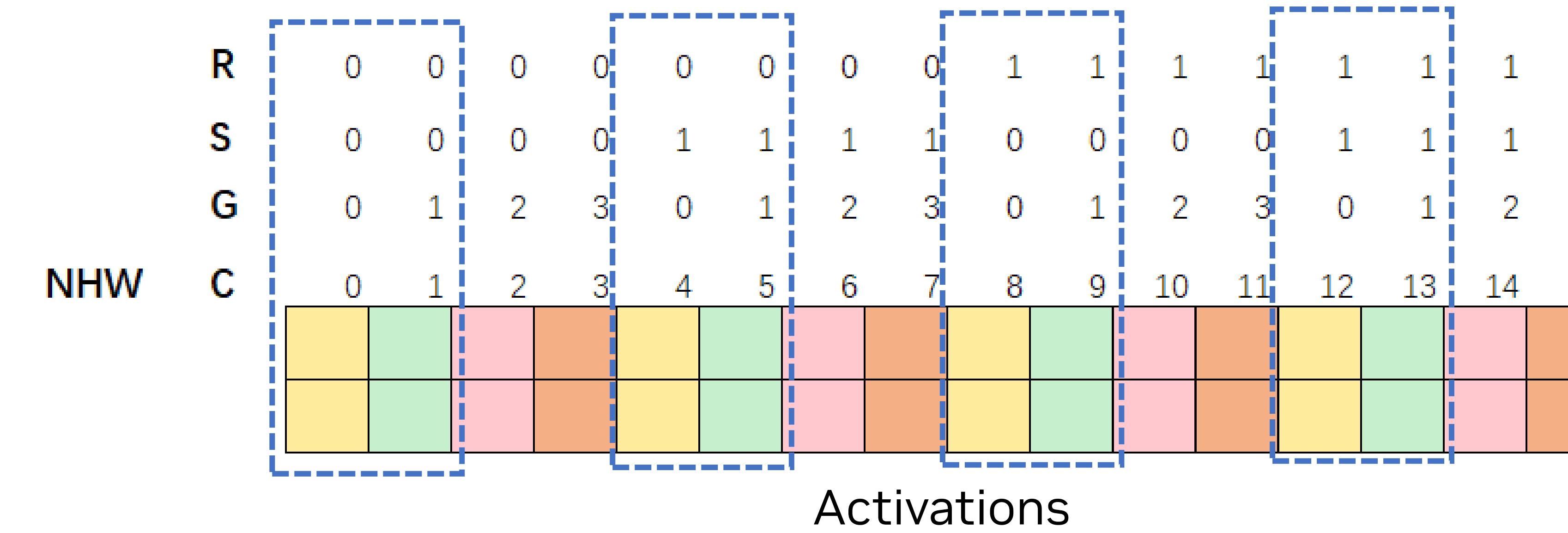


CUTLASS Grouped Convolution Implementation

`cutlass::conv::kMultipleGroup` mode: 1 CTA calculates multiple groups
this mode is for filters_per_group < CTA_tile_N



Groups = 4



2x reduction in mainloop iterations

Grouped Convolution Code Snippet

CUTLASS 2.11

CUTLASS C++ Template Instantiation
(Compile time)

```
Grouped conv kernel

using Kernel = typename cutlass::conv::kernel::DefaultConv2dGroupFprop<
    cutlass::half_t, cutlass::layout::TensorNHWC,
    cutlass::half_t, cutlass::layout::TensorNHWC,
    cutlass::half_t, cutlass::layout::TensorNHWC,
    float,
    cutlass::arch::OpClassTensorOp,
    cutlass::arch::Sm80,
    cutlass::gemm::GemmShape<128, 128, 64>,
    cutlass::gemm::GemmShape<64, 64, 64>,
    cutlass::gemm::GemmShape<16, 8, 16>,
    cutlass::epilogue::thread::LinearCombination<
        cutlass::half_t,
        128 / cutlass::sizeof_bits<cutlass::half_t>::value,
        float,
        float
    >,
    cutlass::gemm::threadblock::GemmIdentityThreadblockSwizzle<>,
    3,
    cutlass::arch::OpMultiplyAdd,
    cutlass::conv::GroupMode::kSingleGroup, CTA_tile::N == k_per_group (k / groups = 128), so we instance SingleGroup mode kernel
    cutlass::conv::IteratorAlgorithm::kAnalytic
>::Kernel;
```

Implicit GEMM Arguments Structure
(Runtime time)

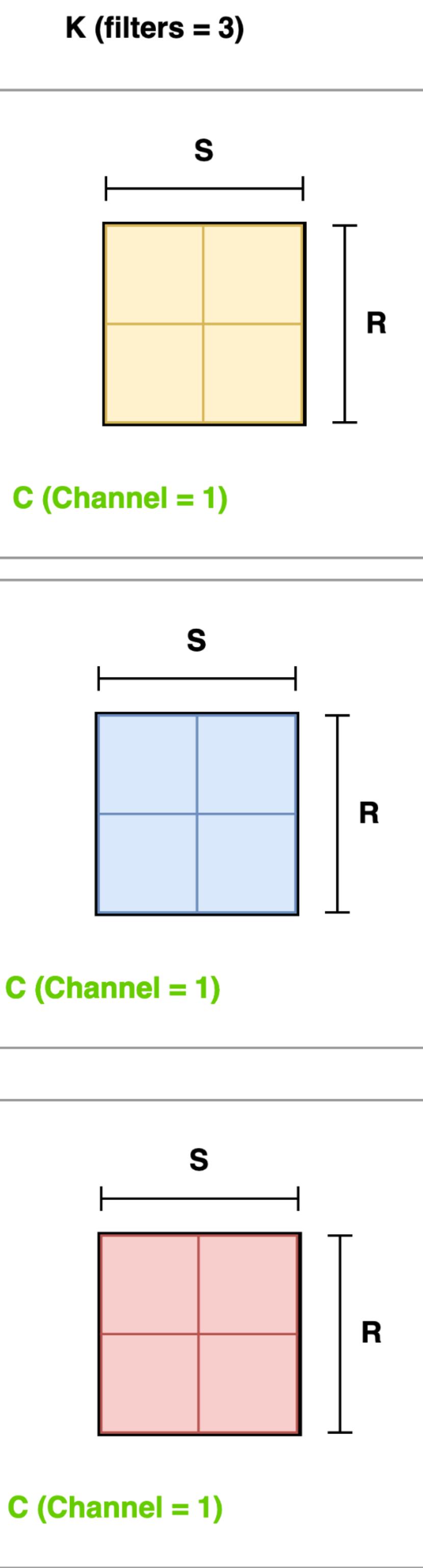
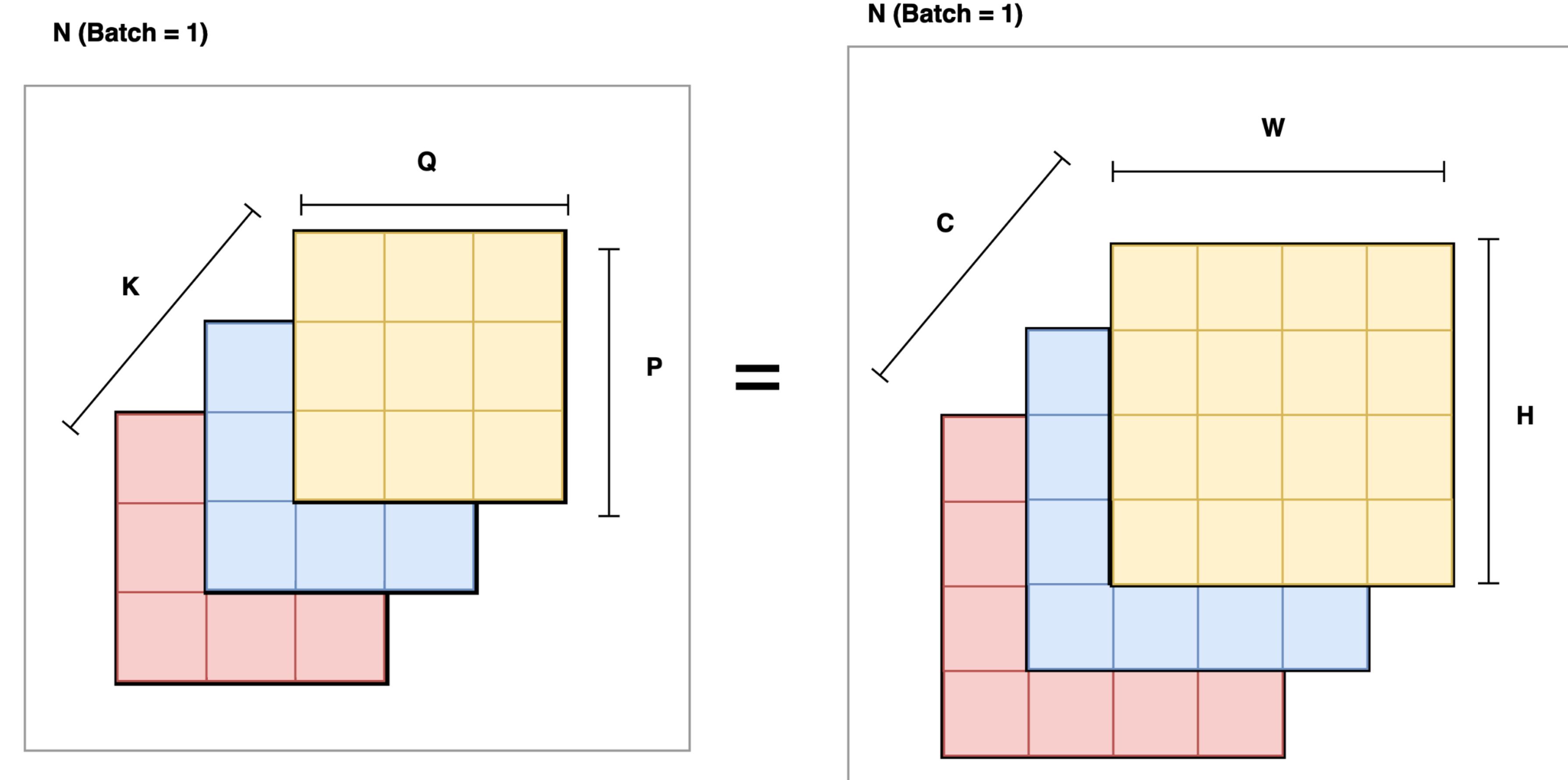
```
using Operation = cutlass::conv::device::ImplicitGemmConvolution<Kernel>;
cutlass::conv::Conv2dProblemSize problem_size(
    { 1, 8, 8, [256]}, // input size (n, h, w, c)
    {[256, 3, 3, 128]}, // filter size (k, r, s, c)
    {1, 1, 1, 1}, // padding
    {1, 1}, // stride
    {1, 1}, // dilation
    cutlass::conv::Mode::kCrossCorrelation, // mode
    1, //split_k_slices
    2 // groups
);
EXPECT_TRUE(test::conv::device::TestSpecificConv2d<Operation>(problem_size));
```

Depthwise Separable Convolution

Forward Propagation

Number of filter groups matches number of channels
`filter.channels = 1`

$$y[n, p, q, k] = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} (x[n, \bar{h}(p, r), \bar{w}(q, s), k] * w[k, r, s, 0])$$



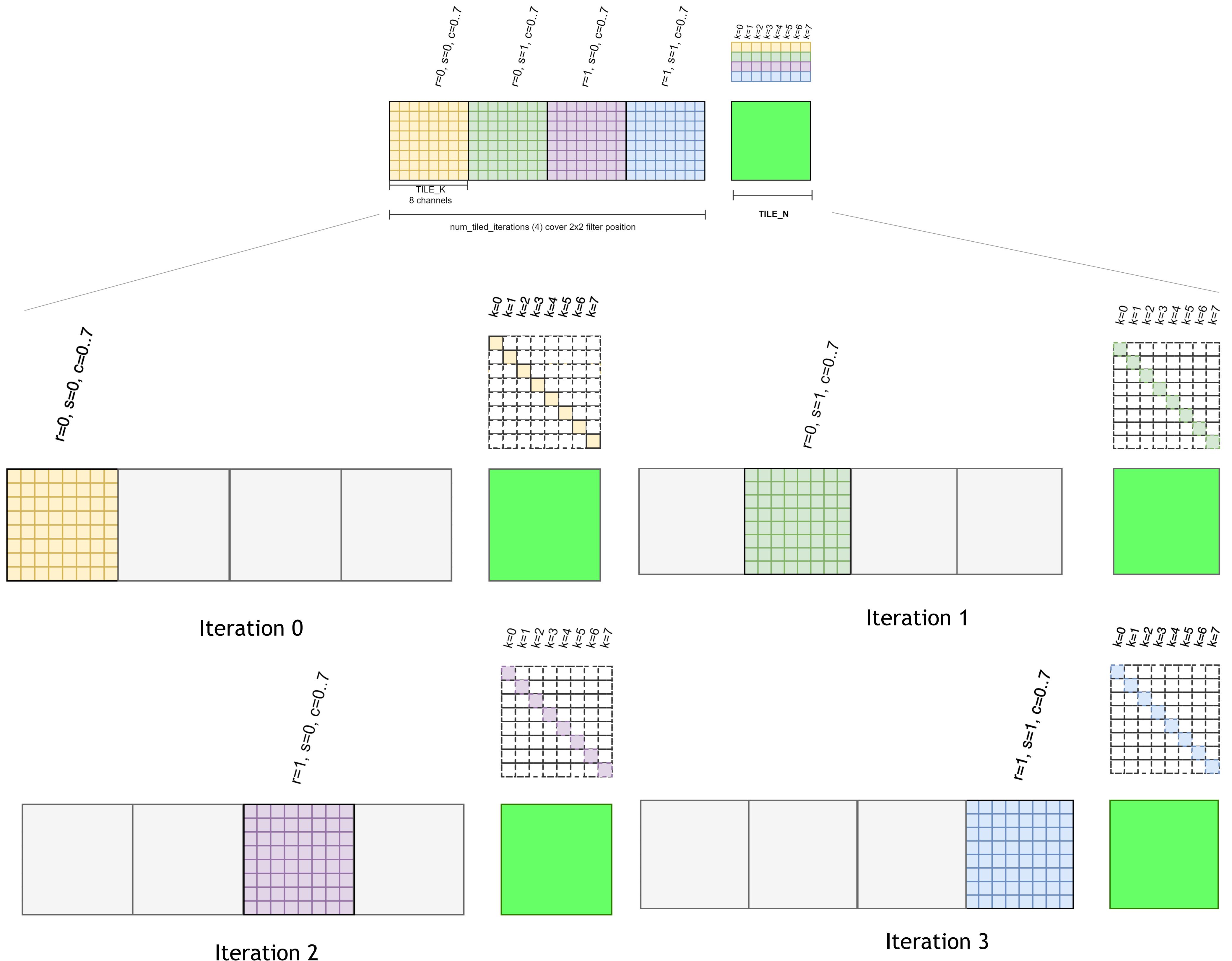
Note: $y_K = x_c = w_K$ AND $w_C = 1$

Depthwise Separable Convolution

Implicit GEMM Convolution Version

In each tiled iteration

- **Activation matrix** is constructed as implicit gemm convolution
- **Filter matrix** for computation is a diagonal-like matrix
 - a) SMEM storage is compact. Only real filter elements from GMEM are loaded and stored.
 - b) Number of LDS is reduced: SMEM data will only be loaded with elements that need to be used (colored)



Depthwise Separable Convolution Code Snippet

CUTLASS 2.11

Instance Depthwise Separable 2d fprop kernel

```
// Device-level Conv2d instance
using Conv2dFpropKernel = typename
cutlass::conv::kernel::DefaultConv2dFprop<
    ElementA,
    cutlass::layout::TensorNHWC,
    ElementB,
    cutlass::layout::TensorNHWC,
    ElementC,
    cutlass::layout::TensorNHWC,
    ElementAccumulator,
    cutlass::arch::OpClassSimt,
    cutlass::arch::Sm60,
    cutlass::gemm::GemmShape<128, 128, 8>,
    cutlass::gemm::GemmShape<64, 64, 8>,
    cutlass::gemm::GemmShape<1, 1, 1>,
    cutlass::epilogue::thread::LinearCombination<
        ElementC,
        1,
        ElementAccumulator,
        ElementCompute
    >,
    cutlass::gemm::threadblock::GemmIdentityThreadblockSwizzle<>,
    2,
    cutlass::arch::OpMultiplyAdd,
    cutlass::conv::IteratorAlgorithm::kAnalytic
>::Kernel;

using Conv2dFprop =
cutlass::conv::device::ImplicitGemmConvolution<Conv2dFpropKernel>;
```

Regular Conv2d fprop kernel

```
// Device-level depthwiseFpropKernel instance
using DepthwiseFpropKernel = typename
cutlass::conv::kernel::DefaultDepthwiseFprop<
    ElementA,
    cutlass::layout::TensorNHWC,
    ElementB,
    cutlass::layout::TensorNHWC,
    ElementC,
    cutlass::layout::TensorNHWC,
    ElementAccumulator,
    cutlass::arch::OpClassSimt,
    cutlass::arch::Sm60,
    cutlass::gemm::GemmShape<128, 128, 8>,
    cutlass::gemm::GemmShape<64, 64, 8>,
    cutlass::gemm::GemmShape<1, 1, 1>,
    cutlass::epilogue::thread::LinearCombination<
        ElementC,
        1,
        ElementAccumulator,
        ElementCompute
    >,
    cutlass::gemm::threadblock::GemmIdentityThreadblockSwizzle<>,
    2,
    cutlass::arch::OpMultiplyAdd,
    cutlass::conv::IteratorAlgorithm::kAnalytic
>::Kernel;

using DepthwiseFprop =
cutlass::conv::device::ImplicitGemmConvolution<DepthwiseFpropKernel>;
```

Depthwise Separable Conv2d fprop kernel

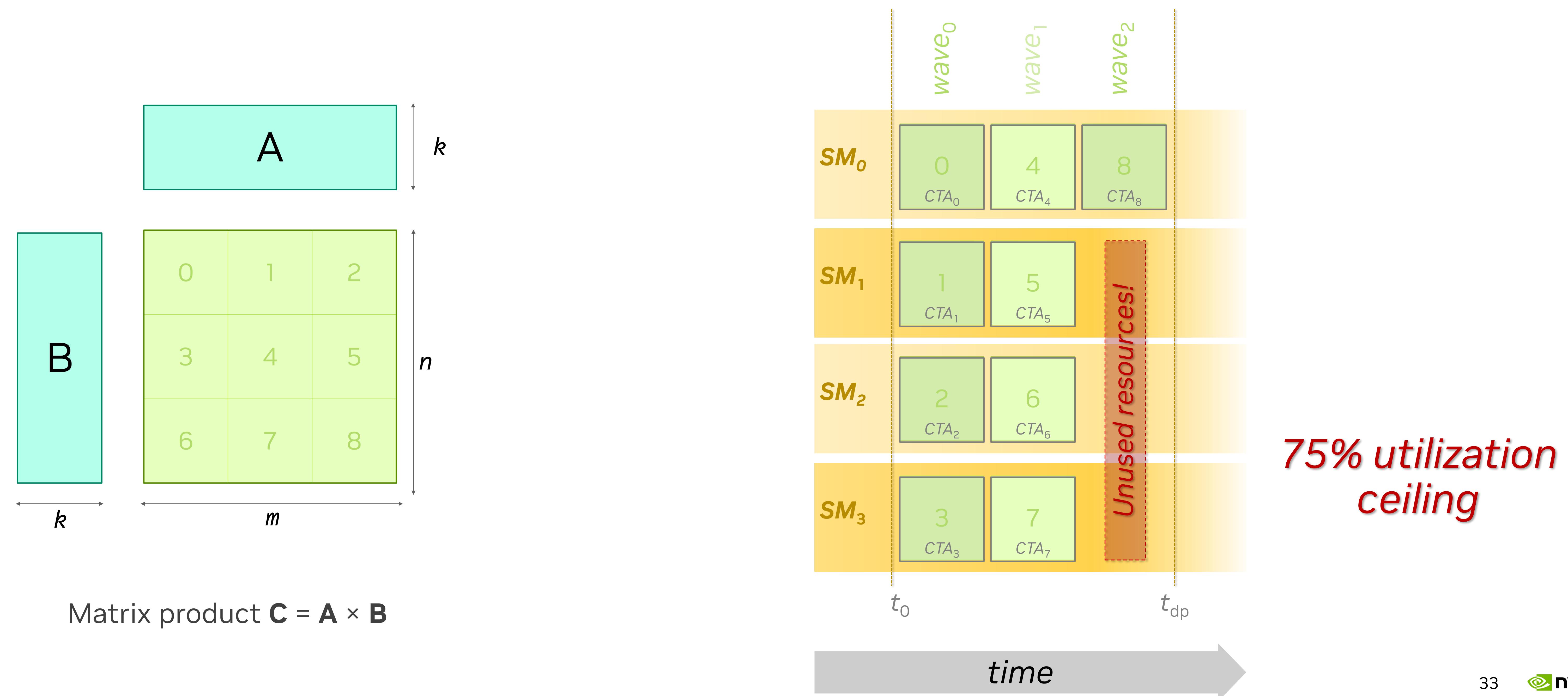


CUTLASS 2.x Enhancements

- Fused Softmax, Layer Norm, Permutation, and Multihead Attention
- Convolution: Grouped and Depthwise Separable
- Stream-K
- NVIDIA Hopper Enablement

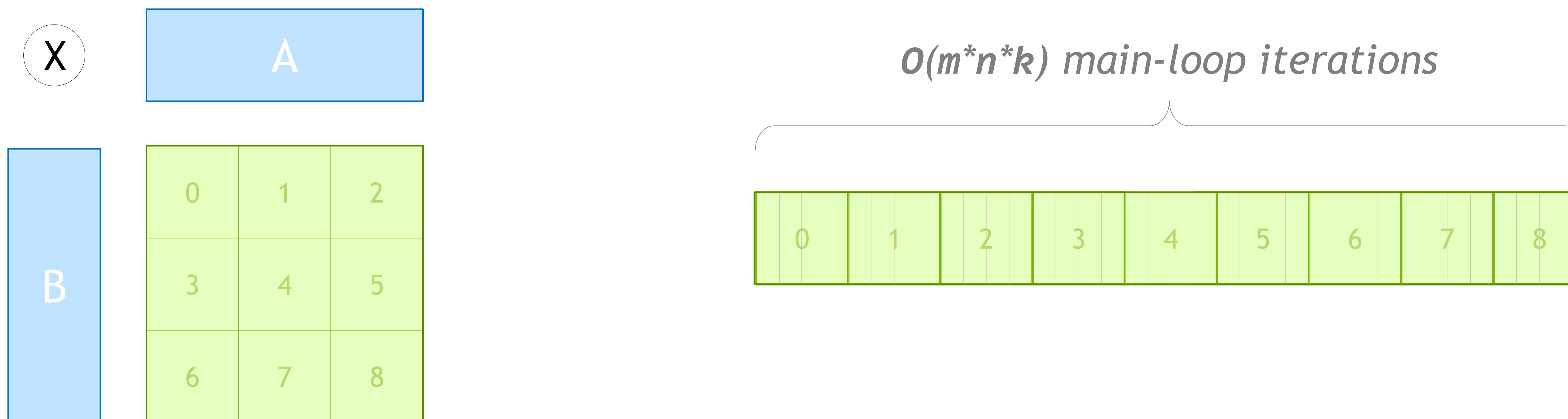
The “Classic” GEMM/CONV Parallelization:

- Output-oriented: Data parallel decomposition (one CTA per output tile)
- Suffers quantization-inefficiency: CTAs dispatch in “waves” across the chip, last wave may be partially-full
- Multiple tile sizes strategy: requires selection heuristic and may still not match GPU occupancy
- Example: 9 CTAs are scheduled in 3 waves across 4 SMs to produce nine 128x128 element tiles



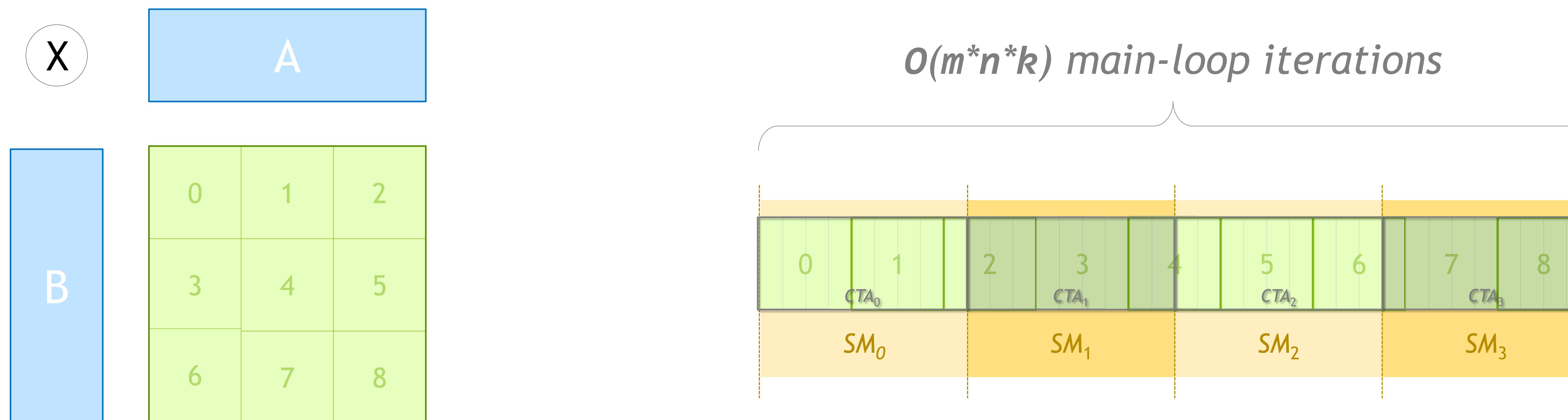
THE STREAM-K DECOMPOSITION (STK)

- Tile the output matrix into “ideal” CTA-sized tiles, but...
... consider the aggregate accumulation work of the entire GEMM computation (i.e., main-loop iterations)



THE STREAM-K DECOMPOSITION (STK)

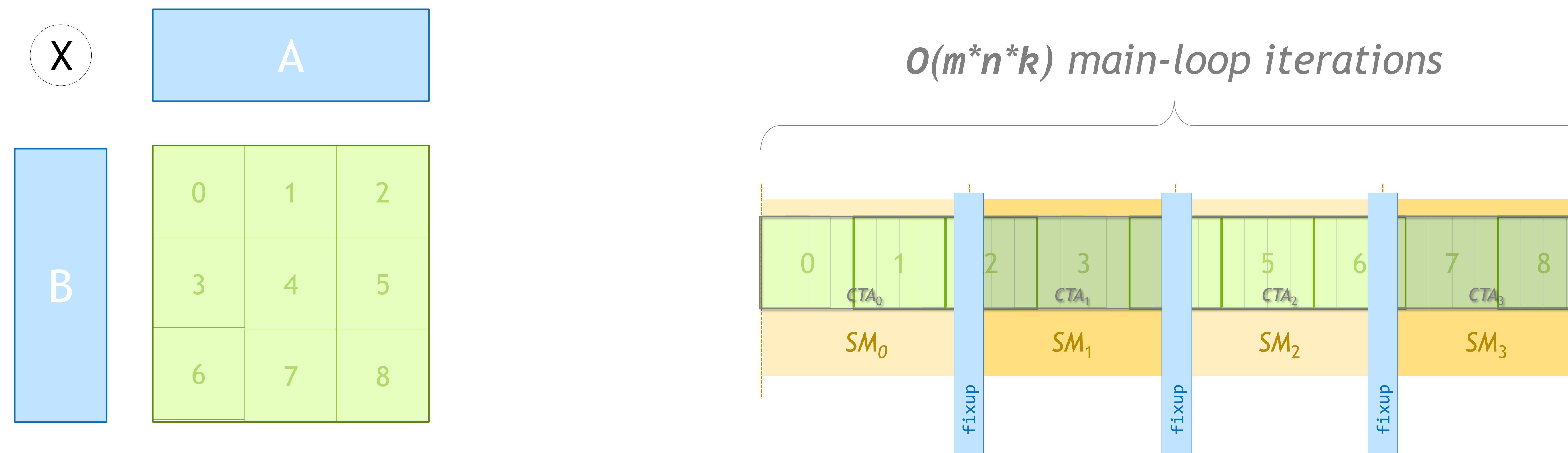
- Assuming p SMs, launch a constant number** of CTAs per SM
- Give each CTA an even-share of the aggregate main-loop iterations
- “Fixup” phase accumulates the “carry-out” partial-sums *from each CTA*
- E.g.: 36 iterations are scheduled across four 128x128 CTAs. (Fixup finishes 3 tiles whose iterations span multiple CTAs.)



(**) Whatever the SM's physical CTA-occupancy is

THE STREAM-K DECOMPOSITION (STK)

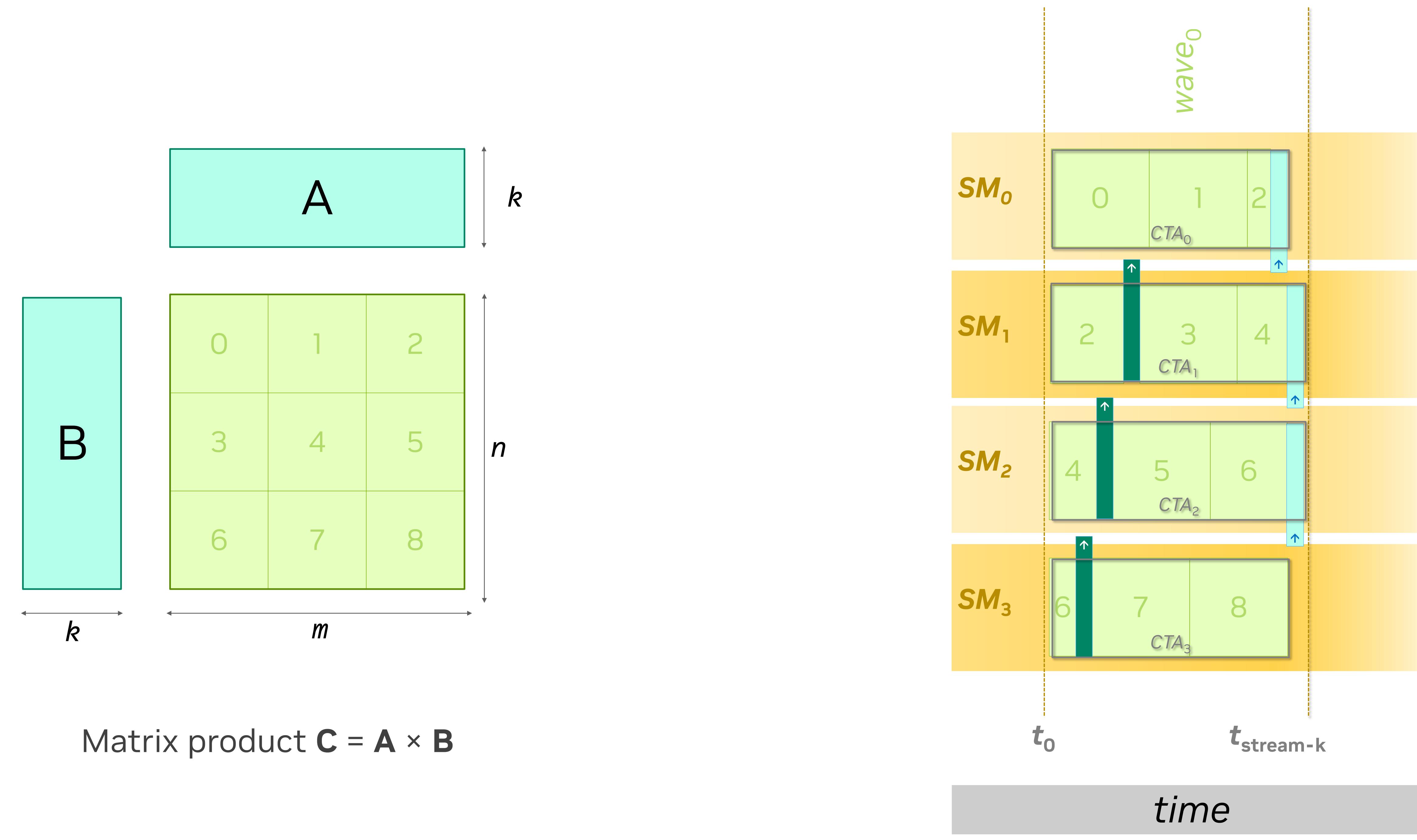
- Assuming p SMs, launch a constant number** of CTAs per SM
- Give each CTA an even-share of the aggregate main-loop iterations
- “Fixup” phase accumulates the “carry-out” partial-sums *from each CTA*
- E.g.: 36 iterations are scheduled across four 128x128 CTAs. (Fixup finishes 3 tiles whose iterations span multiple CTAs.)



(**) Whatever the SM’s physical CTA-occupancy is

“Stream-K” GEMM/CONV Parallelization

- Iteration-centric: each SM receives an even-share of the aggregate “main-loop” iterations
- Low “fixup overhead”: $O(p)$ work & storage for aggregation of partial sums across SMs
- Generalizes prior methods: Iteration-centric design can also emulate “data-parallel” and “split-k” decompositions



ADDING THE STREAM-K OUTER LOOP

Minimal extra structure around existing abstractions

```
__global__ GemmKernel(...)  
{  
    // The Stream-K persistent work-processing loop  
    while (cta_itr < cta_last_itr)  
    {  
        tile_id          = get_tile_id(itr)  
        tile_first_itr   = get_tile_itr(tile_id)  
        tile_stop_itr    = min(cta_last_itr, tile_first_itr + iters_per_tile)  
  
        // The standard tile-based GEMM main loop  
        accum = gemm_tile(  
            tile_id,  
            cta_itr - tile_first_itr,  
            tile_stop_itr - tile_first_itr);  
  
        if (!started_tile && !finished_tile) {  
            // Carry-out partial sums  
            store_partials(accum);  
            signal_done(my_bid);  
        } else {  
            if (started_tile && !finished_tile) {  
                // Carry-in sets of partial sums  
                last_bid = last_tile_bid(tile_id)  
                wait_for(my_bid + 1, last_bid)  
                accumulate_partials(accum, my_bid, last_bid)  
            }  
  
            // Produce final output tile  
            epilogue(accum, tile_id);  
        }  
    }  
}
```

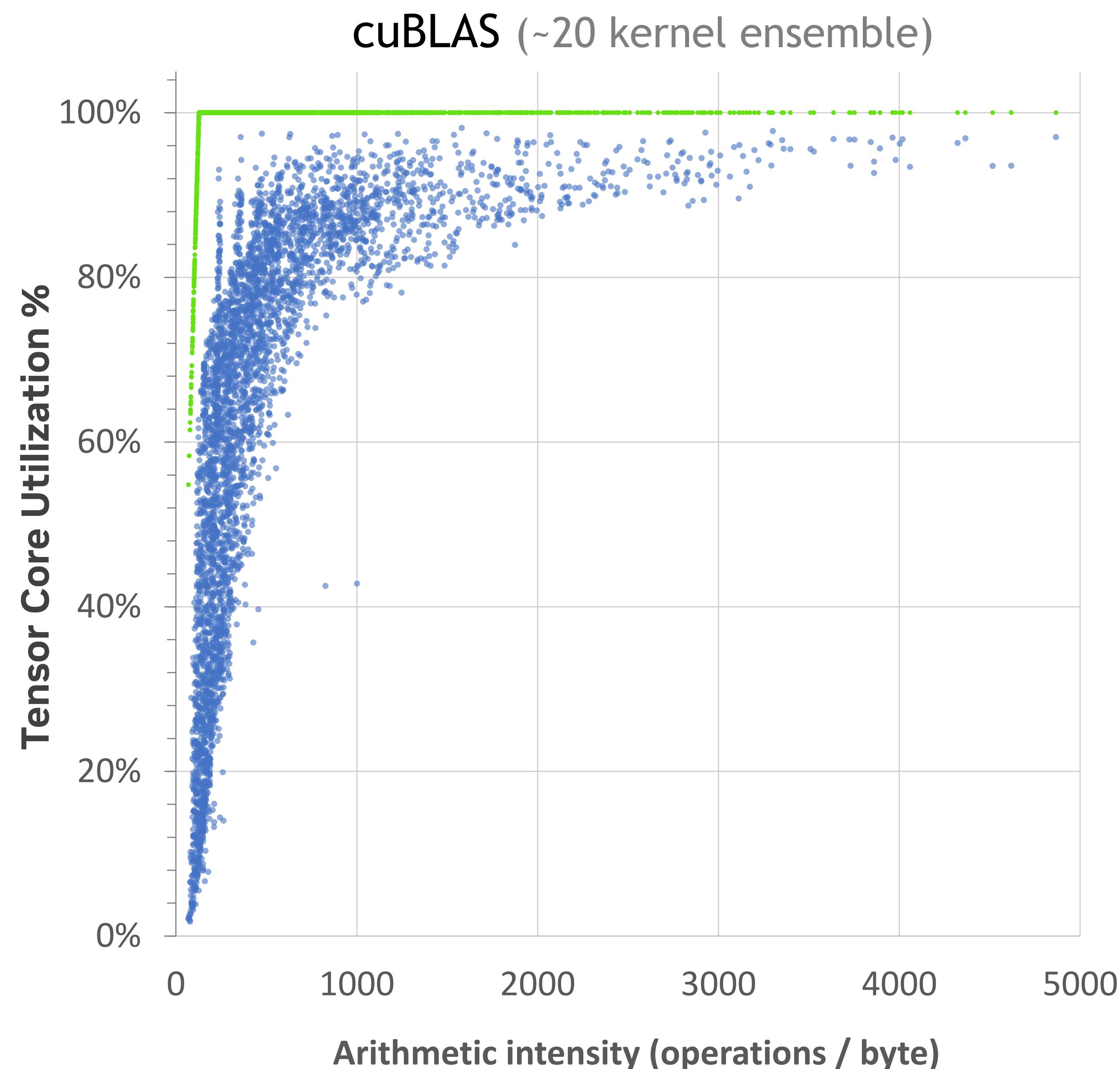
Dynamic Threadblock Scheduling logic

“Fixup” logic

Performance Improvement

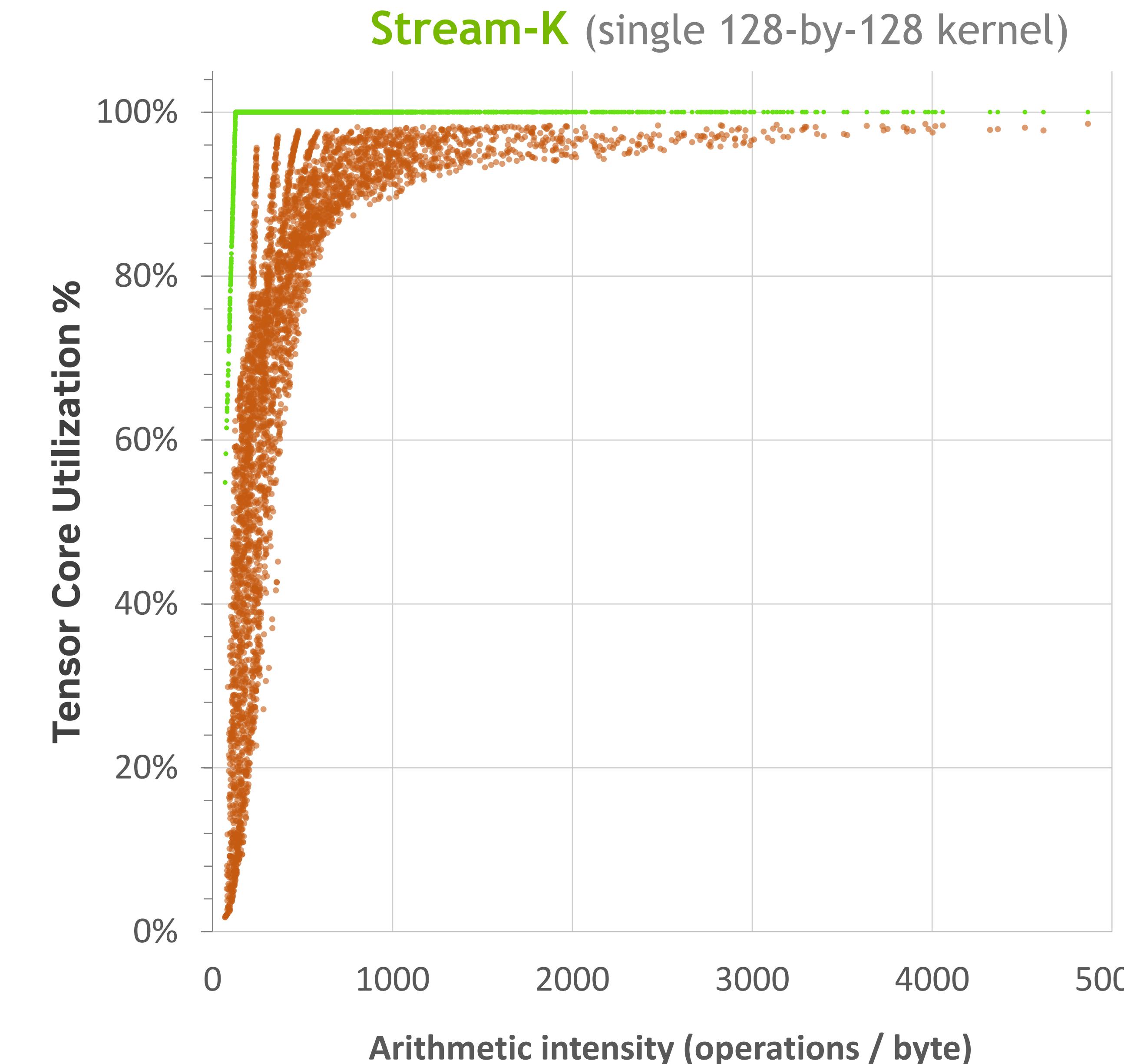
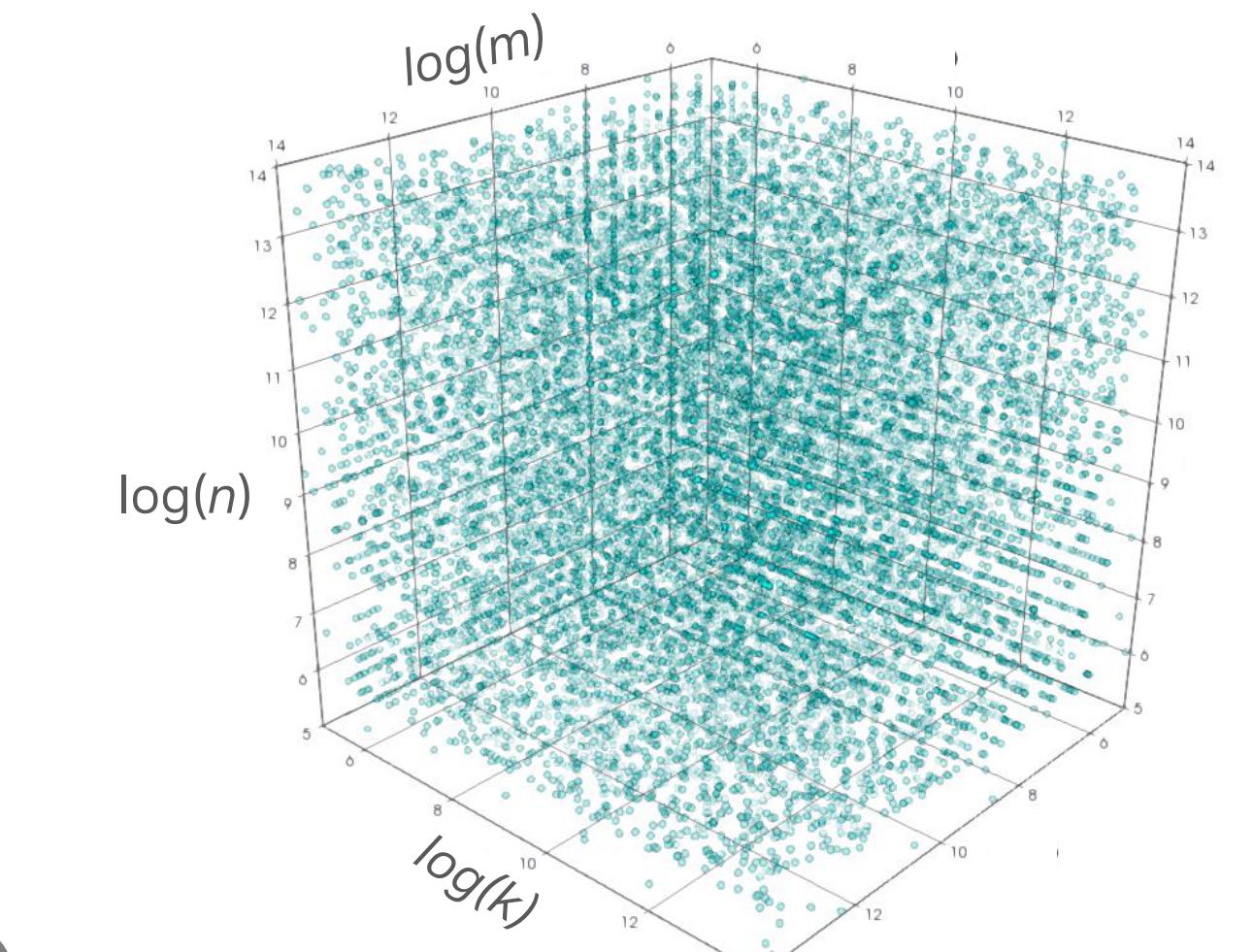
cuBLAS (ensemble) vs. Stream-K (single kernel)

- Improved *absolute* performance from strong-scaling
 - Improved performance *consistency*
- Fewer kernel specializations needed (reduced library size)



Sample study: 4,096 GEMM problems (f16.f16.f32.nt)

- NVIDIA A100
- 1.1x avg speedup
- 5.4x max speedup
- 0.6x min speedup



Using Stream-K in CUTLASS

CUTLASS 2.11

Stream-K enabled across CUTLASS GEMM computations with new threadblock rasterization function.

Extends existing GEMM interface

```
// Define a CUTLASS GEMM with Stream-K enabled
using GemmKernel =
  typename cutlass::gemm::kernel::DefaultGemmUniversal<
    cutlass::half_t, cutlass::layout::ColumnMajor,
    cutlass::ComplexTransform::kNone, 8,
    cutlass::half_t, cutlass::layout::RowMajor,
    cutlass::ComplexTransform::kNone, 8,
    cutlass::half_t, cutlass::layout::RowMajor,
    float,
    cutlass::arch::OpClassTensorOp,
    cutlass::arch::Sm80,
    cutlass::gemm::GemmShape<128, 128, 32>,
    cutlass::gemm::GemmShape<64, 64, 32>,
    cutlass::gemm::GemmShape<16, 8, 16>,
    cutlass::epilogue::thread::LinearCombination<
      cutlass::half_t,
      8,
      float,
      float
    >,
    cutlass::gemm::threadblock::ThreadblockSwizzleStreamK,
    3,
    cutlass::arch::OpMultiplyAdd
  >::GemmKernel;
```

```
using Gemm = cutlass::gemm::device::GemmUniversal<GemmKernel>;
// Construct arguments structure with additional Stream-K controls.
//
// Passing default values enables an internal heuristic.
typename Gemm::Arguments arguments{
  cutlass::GemmUniversalMode::kGemm,
  problem_size,
  batch_count,
  epilogue_params,
  ptr_A,
  ptr_B,
  ptr_C,
  ptr_D,
  batch_stride_A,
  batch_stride_B,
  batch_stride_C,
  batch_stride_D,
  lda,
  ldb,
  ldc,
  ldd,
  // number of output tiles to assign to independent, data-parallel CTAs
  // (default: -1)
  dp_tiles,
  // number of Stream-K CTAs for cooperatively processing remaining output tiles
  // (default: -1)
  streamk_blocks,
  // when the above are defaulted, the number of SMs that dispatch heuristics
  // will attempt to load-balance
  // (default: -1)
  available_SMs
};

// Launch the GEMM using Stream-K algorithm
Gemm gemm;
Status status = gemm(arguments);
```



CUTLASS 2.x Enhancements

- Fused Softmax, Layer Norm, Permutation, and Multihead Attention
- Convolution: Grouped and Depthwise Separable
- Stream-K
- NVIDIA Hopper Enablement

NVIDIA Hopper Enablement

NVIDIA H100

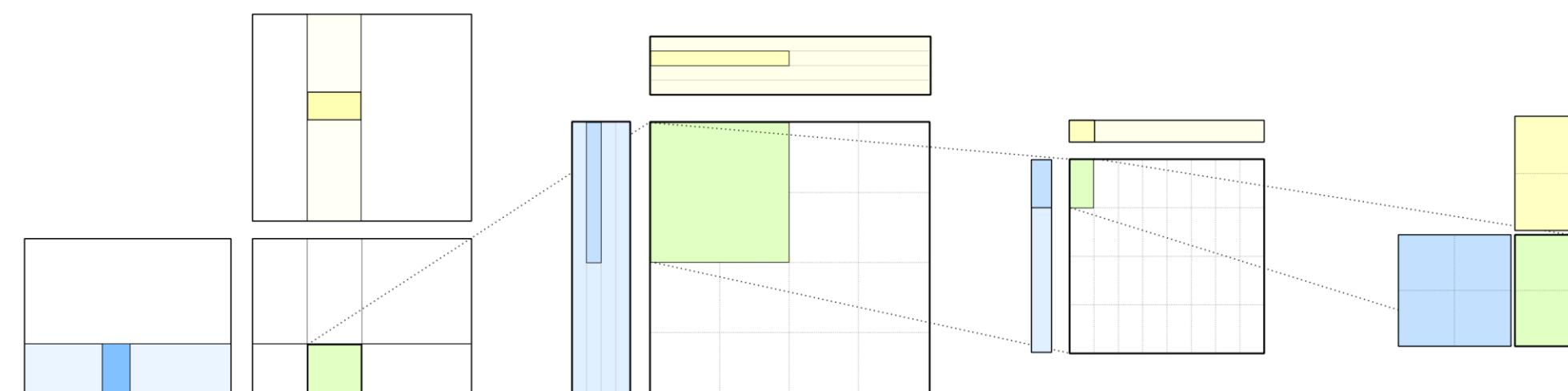
New and Faster 4th Generation Tensor Core offer Architectural Speedup

- Mixed-precision floating-point Tensor Core operations **2x** faster than NVIDIA A100
- IEEE double-precision Tensor Cores **2x** faster than NVIDIA A100

FP8 Data Types and Mode

- FP8 floating point types
- FP8 Tensor Cores are **4x** faster than FP16 Tensor Cores on NVIDIA A100

Many additional new features – see [“NVIDIA H100 Tensor Core GPU Architecture”](#) for more details



NVIDIA Hopper Enablement

CUDA 11.8 + CUTLASS 2.11

New and Faster Tensor Core Operations offer Architectural Speedup

- Mixed-precision floating-point Tensor Core operations **2x** faster than NVIDIA A100
- IEEE double-precision Tensor Cores **2x** faster than NVIDIA A100

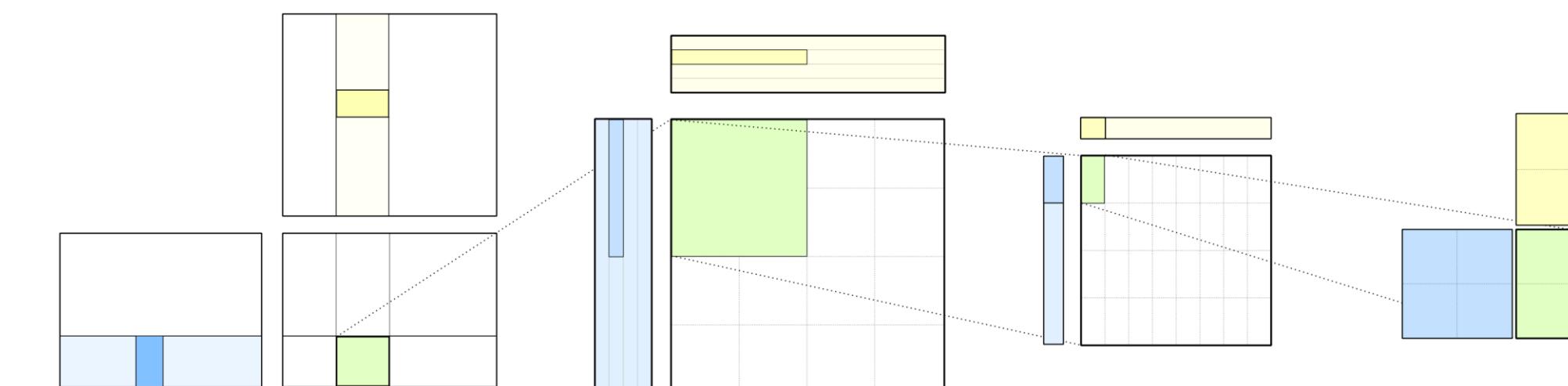
Additional Data Types and Mode

- FP8 floating point types
- FP8 Tensor Cores are **4x** faster than FP16 on NVIDIA A100

Accessible with CUDA 11.8 Toolkit

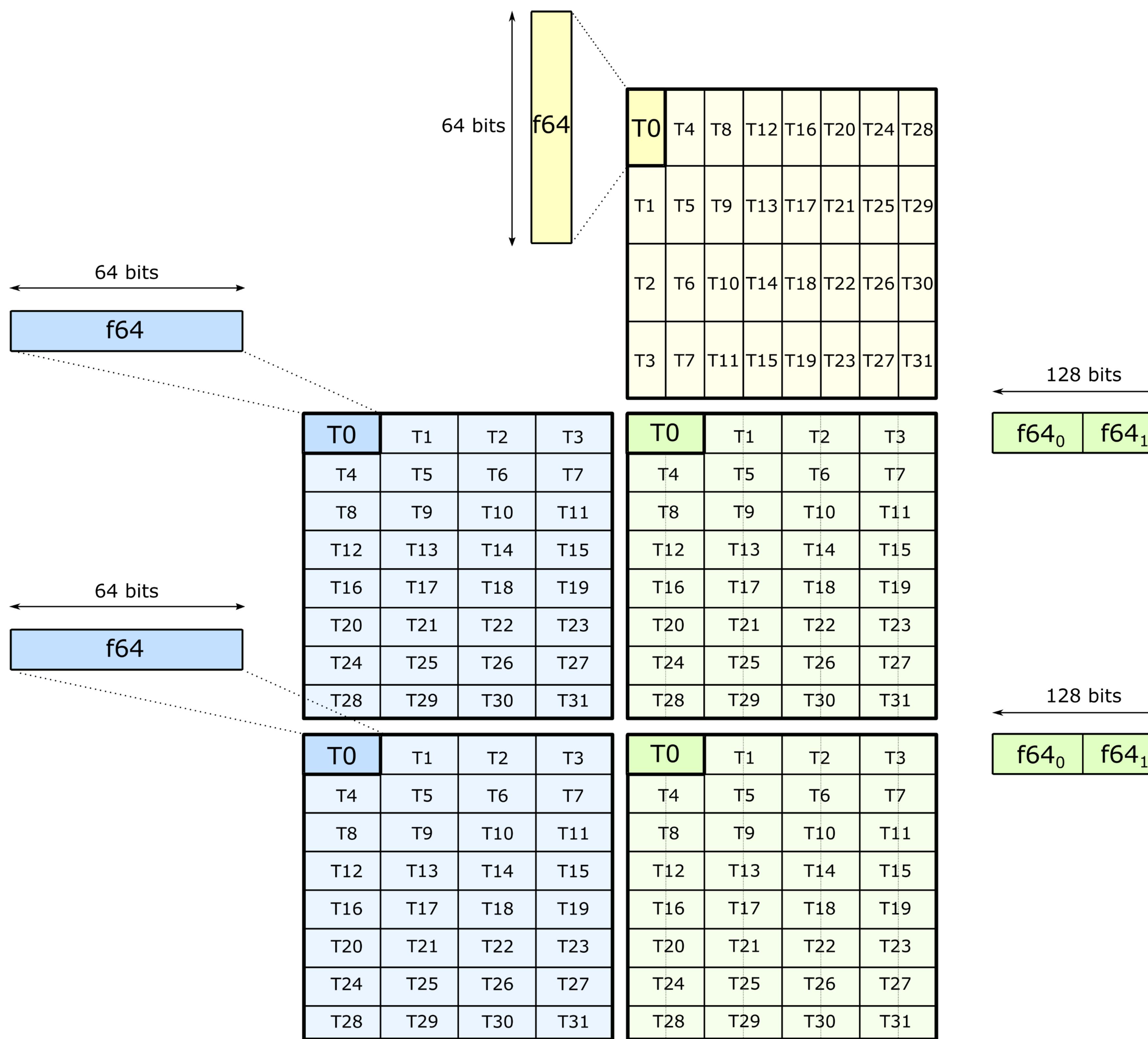
Targeted by CUTLASS 2.11

Many additional new features – see “[NVIDIA H100 Tensor Core GPU Architecture](#)” for more details



DOUBLE-PRECISION: F64 * F64 + F64

16-by-8-by-4



mma.sync.aligned (via inline PTX)

```
double      D[4];    // four 64-bit accumulators
double const A[2];  // two 64-bit elements for A operand
double const B;     // one 64-bit element for B operand
double const C[4];  // four 64-bit accumulators

// Example targets 16-by-8-by-4 Tensor Core operation

asm(
    "mma.sync.aligned.m16n8k4.row.col.f64.f64.f64.f64 "
    "%{0, %1, %2, %3}, "
    "%{4, %5}, "
    "%6,
    "%{7, %8, %9, %10};\n"
    :
    "=d"(D[0]), "=d"(D[1]), "=d"(D[2]), "=d"(D[3])
    :
    "d"(A[0]), "d"(A[1]),
    "d"(B[0]),
    "d"(C[0]), "d"(C[1]), "d"(C[2]), "d"(C[3])
);
```

2x Double Precision Tensor Cores on NVIDIA Hopper

Usage in CUTLASS 2.11

- Available in CUTLASS 2.11

Achieve optimal performance on NVIDIA Ampere and NVIDIA Hopper by varying InstructionShape argument

```
using Gemm = cutlass::gemm::device::Gemm<
    double,
    cutlass::layout::ColumnMajor,
    double,
    cutlass::layout::RowMajor,
    double,
    cutlass::layout::RowMajor,
    double,
    cutlass::arch::OpClassTensorOp,
    cutlass::arch::Sm80,>

// Threadblock shape
cutlass::gemm::GemmShape<32, 32, 16>,
// Warp shape
cutlass::gemm::GemmShape<16, 16, 16>,
// Instruction shape
cutlass::gemm::GemmShape<8, 8, 4>,
cutlass::epilogue::thread::LinearCombination<
    double,
    1,
    double,
    double
>,
cutlass::gemm::threadblock::GemmIdentityThreadblockSwizzle<>,
4
>;
```

NVIDIA Ampere (sm80)

```
using Gemm = cutlass::gemm::device::Gemm<
    double,
    cutlass::layout::ColumnMajor,
    double,
    cutlass::layout::RowMajor,
    double,
    cutlass::layout::RowMajor,
    double,
    cutlass::arch::OpClassTensorOp,
    cutlass::arch::Sm90,>

// Threadblock shape
cutlass::gemm::GemmShape<32, 32, 16>,
// Warp shape
cutlass::gemm::GemmShape<16, 16, 16>,
// Instruction shape
cutlass::gemm::GemmShape<16, 8, 4>,
cutlass::epilogue::thread::LinearCombination<
    double,
    1,
    double,
    double
>,
cutlass::gemm::threadblock::GemmIdentityThreadblockSwizzle<>,
4
>;
```

NVIDIA Hopper (sm90)

FP8 Floating-point Types

Hopper FP8 Data Types:

E4M3 and E5M2

E4M3 s e3 e2 e1 e0 m2 m1 m0

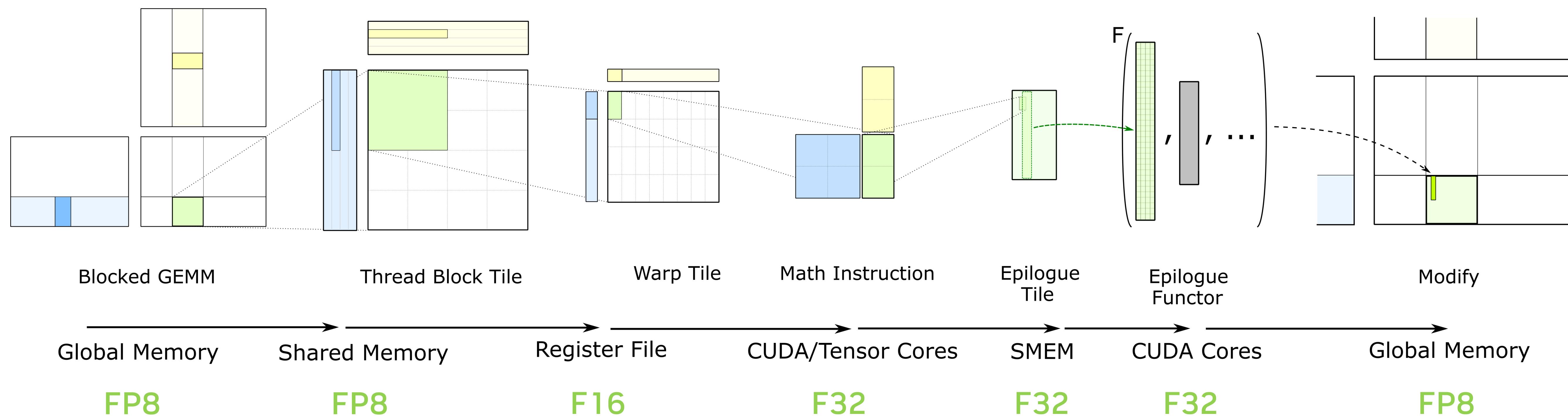
E5M2 s e4 e3 e2 e1 e0 m1 m0

CUDA 11.8 enables hardware-accelerated convert-and-pack operations

4 x FP8 \leftrightarrow 4 x F16

4 x F32 \rightarrow 4 x FP8

CUTLASS 2.11 implements FP8 “fast math” kernels: FP8 input \rightarrow F16 * F16 + F32 compute \rightarrow FP8 output





CUTLASS 3.0 Preview

- CuTe Layout Functions
- Tensor Core Programming Model
- CUTLASS 3.0 API



CUTLASS 3.0 Preview

- CuTe Layout Functions
- Tensor Core Programming Model
- CUTLASS 3.0 API

Hierarchical Layouts

Coordinates and Indices

4x4 matrix with layout

Shape: $(4, (2, 2))$

Stride: $(2, (1, 8))$

Logical 1-D
coordinate

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Logical n-D
coordinate

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



$A(I)$

$A(i, j)$

Logical h-D
coordinate

0, (0, 0)	0, (1, 0)	0, (0, 1)	0, (1, 1)
1, (0, 0)	1, (1, 0)	1, (0, 1)	1, (1, 1)
2, (0, 0)	2, (1, 0)	2, (0, 1)	2, (1, 1)
3, (0, 0)	3, (1, 0)	3, (0, 1)	3, (1, 1)



$A(i, (j_1, j_2))$

Linear 1-D
storage index

0	1	8	9
2	3	10	11
4	5	12	13
6	7	14	15



$A[k]$

Shape defines the *coordinate* mappings

$$(I) \Leftrightarrow (i, j) \Leftrightarrow (i, (j_1, j_2))$$

Stride defines the *index* mapping

$$(i, (j_1, j_2)) \Leftrightarrow [k]$$

Example Layout

More operations

```
shape(A) = ((2, (2, 2)), (2, (2, 2)))
stride(A) = ((1, (4, 16)), (2, (8, 32)))
```

i\j	0	1	2	3	4	5	6	7
0	0	2	8	10	32	34	40	42
1	1	3	9	11	33	35	41	43
2	4	6	12	14	36	38	44	46
3	5	7	13	15	37	39	45	47
4	16	18	24	25	48	50	56	58
5	17	19	26	27	49	51	57	59
6	20	22	28	30	52	54	60	62
7	21	23	29	31	53	55	61	63

```
Layout morton1 = make_layout(Shape<_2,_2>{}, Stride<_1,_2>{})
Layout morton2 = blocked_product(morton1, morton1);
Layout morton3 = blocked_product(morton1, morton2);
Tensor A = make_tensor(counting_iterator<int>{}, morton3);
```

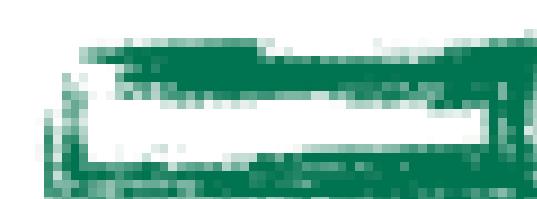
➤ Shape

size(A) = 64

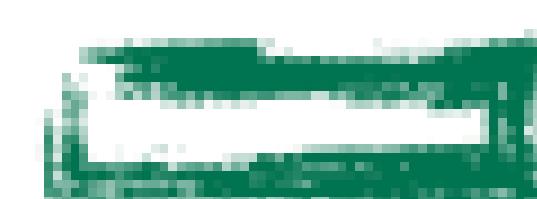
size<0>(A) = 8

size<1>(A) = 8

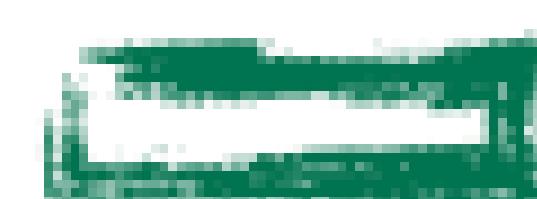
➤ Logical coordinates are 1D, 2D, hD



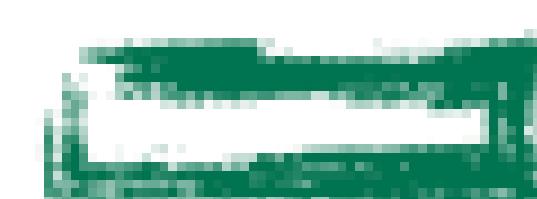
A(37) = 49



A(5, 4) = 49

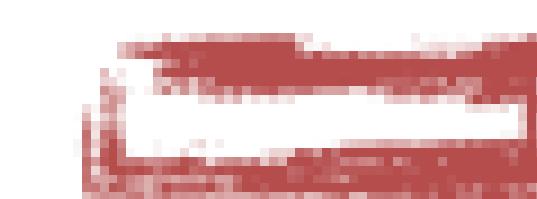


A((1, 2), (0, 2)) = 49



A((1, (0, 1)), (0, (0, 1))) = 49

➤ Slice along logical sub-boundaries



A(_ , 2) = [8; 9; 12; 13; 24; 26; 28; 29]



A((_, 1), (_, 2)) = [36, 38;

37, 39]

Layout And Tensors

Why CuTe?

- Formal algebra over **Layout**

- composition
- right_inverse
- left_inverse
- complement
- "product"
- "divide"
- "tiling"
- "partitioning"

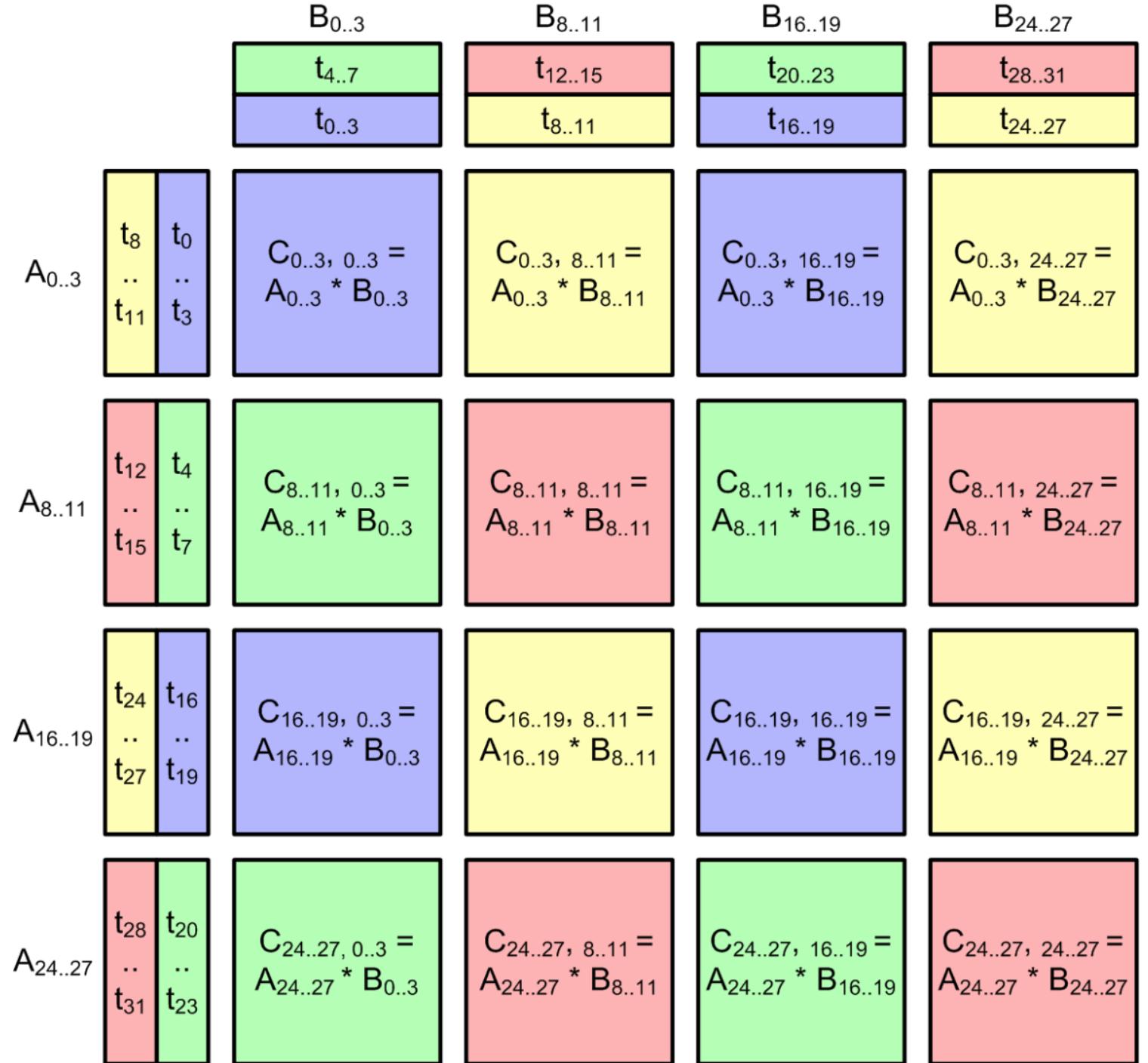
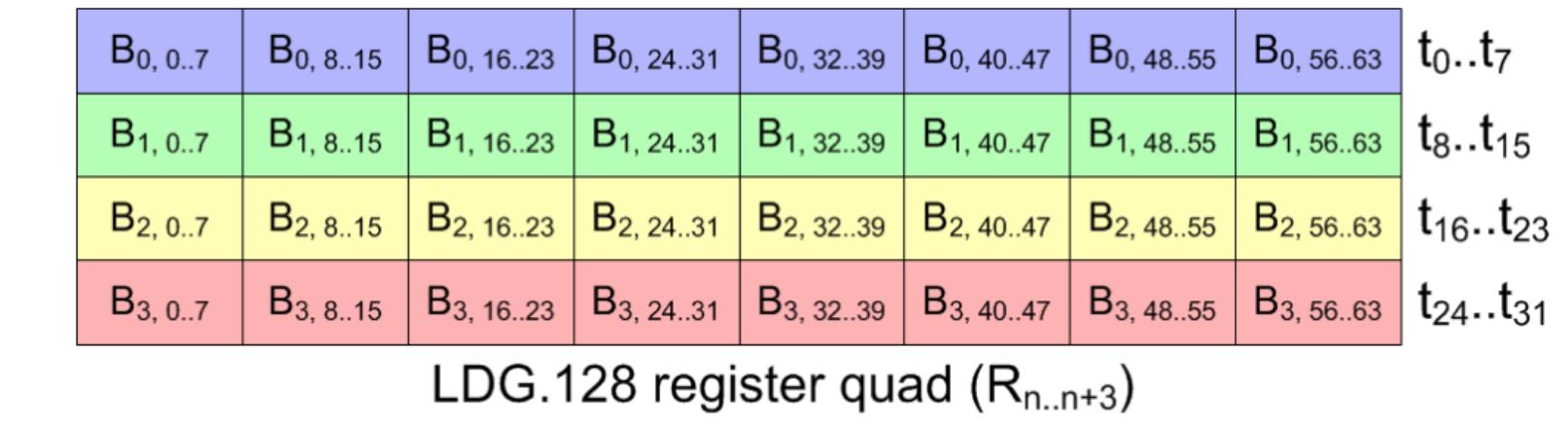
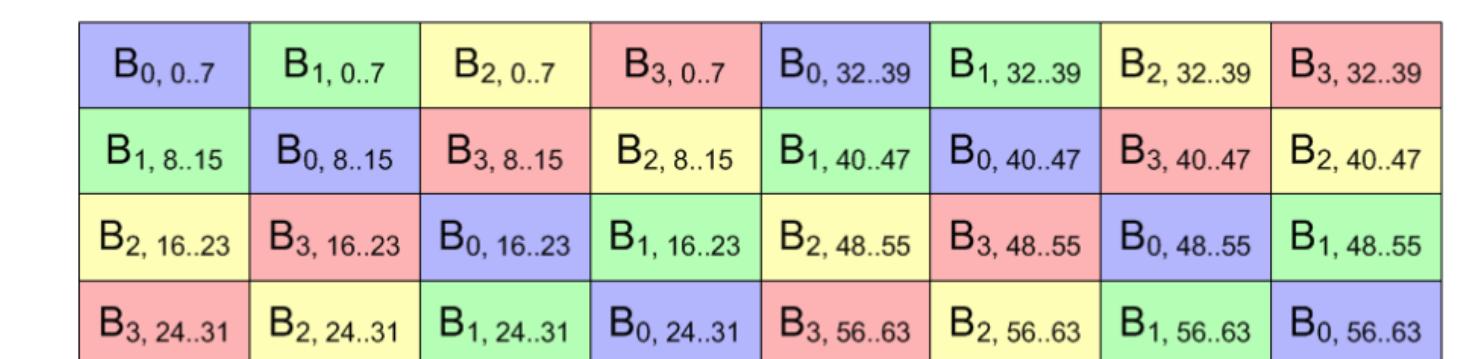


Figure 2a: Composing sparse $A[16,4] * B[4, 16] = C[16, 16]$
warp-wide multiply from four $A[8, 4] * B[4, 8] = C[8, 8]$ 8-thread multiplies.



LDG.128 register quad ($R_{n..n+3}$)



Shared memory after STS.128 w/swizzled thread id =
(tid[1:0] << 3) | (tid & 4) | (tid[4:3] ^ tid[1:0])

CUTLASS

RowMajor

ColumnMajor

RowMajorInterleaved

ColumnMajorInterleaved

PitchLinear

TensorNCHW

VoltaTensorOpMultiplicandCongruous

ColumnMajorVoltaTensorOpMultiplicandCongruous

RowMajorVoltaTensorOpMultiplicandCongruous

VoltaTensorOpMultiplicandBCongruous

ColumnMajorVoltaTensorOpMultiplicandBCongruous

RowMajorVoltaTensorOpMultiplicandBCongruous

VoltaTensorOpMultiplicandCrosswise

ColumnMajorVoltaTensorOpMultiplicandCrosswise

RowMajorVoltaTensorOpMultiplicandCrosswise

Many, many more ...

CuTe



Layout<Shape, Stride>

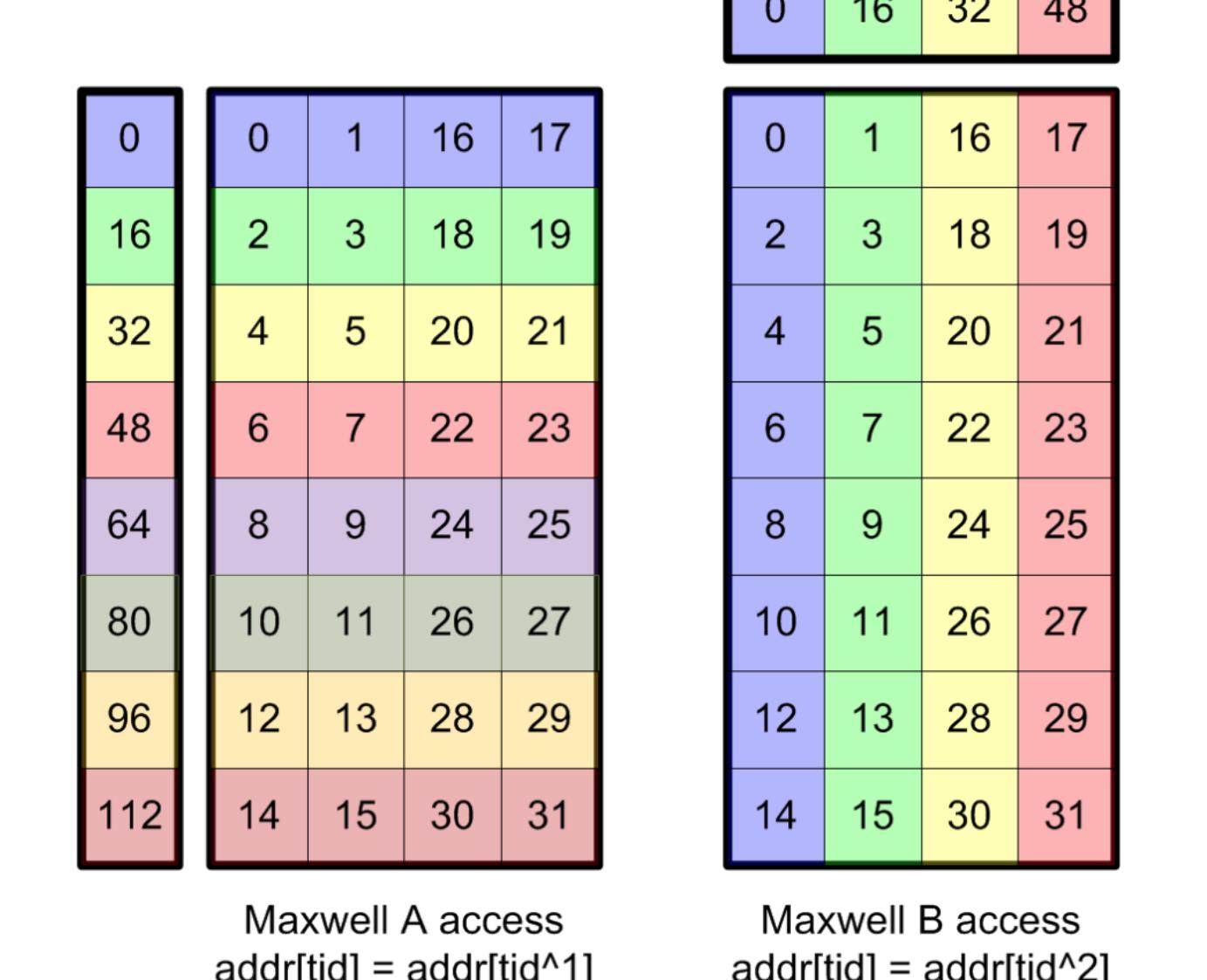
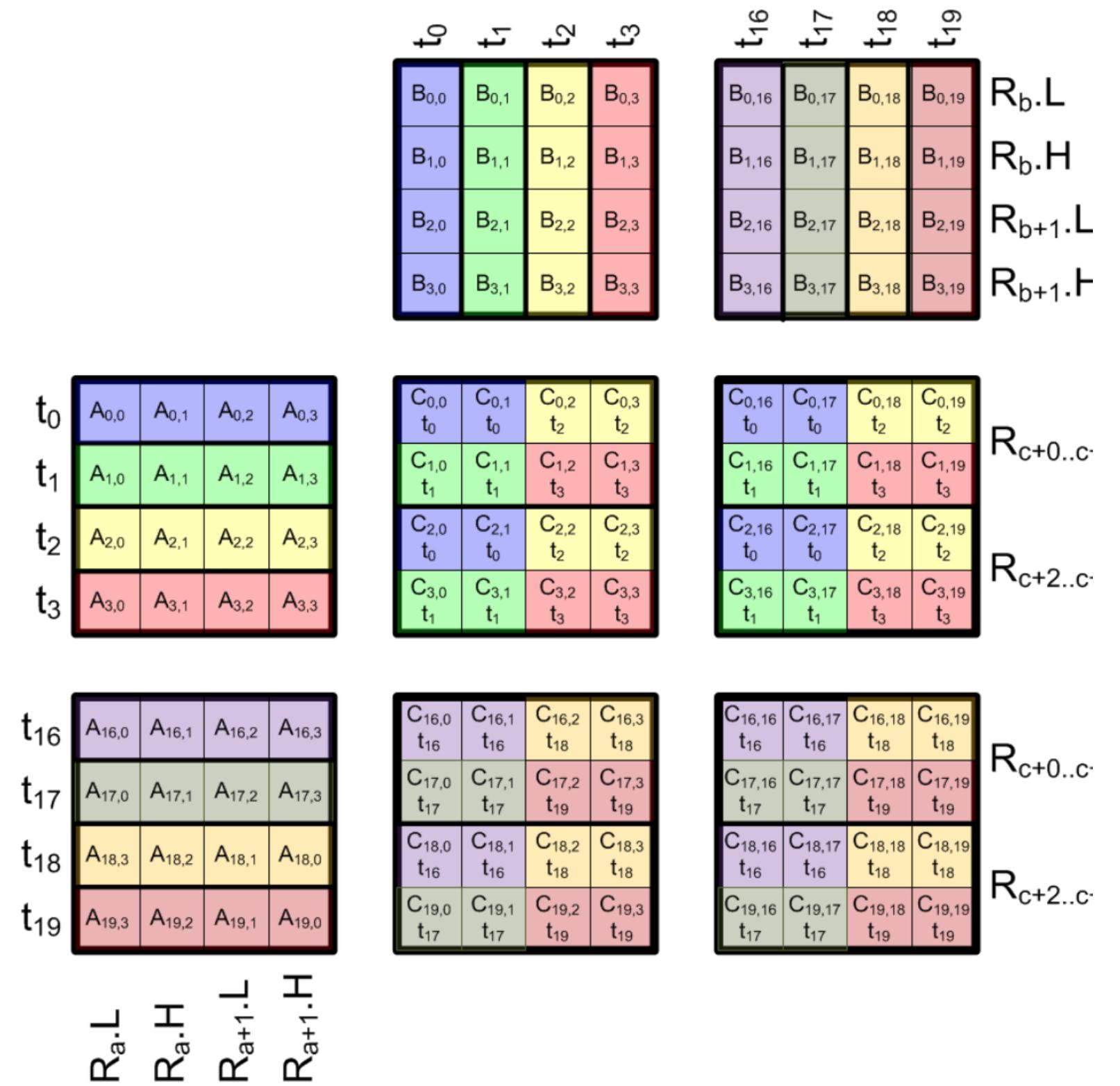


Figure 3b: Maxwell thread assignments for LDS.U matching



CUTLASS 3.0 Preview

- CuTe Layout Functions
- **Tensor Core Programming Model**
- CUTLASS 3.0 API

MMA_Op and MMA_Traits

PTX and Meta-info

```
struct SM90_16x8x4_F64F64F64F64_TN
{
    using DRegisters = double[4];
    using ARegisters = double[2];
    using BRegisters = double[1];
    using CRegisters = double[4];

    CUDA_HOST_DEVICE static void
    fma(double      & d0, double      & d1,
        double      & d2, double      & d3,
        double const& a0,
        double const& b0,
        double const& c0, double const& c1,
        double const& c2, double const& c3)
    {
        asm volatile(
            "mma.sync.aligned.m16n8k4.row.col.f64.f64.f64.f64"
            "{%0, %1, %2, %3 },"
            "{%4, %5},"
            "{%6},"
            "{%7, %8, %9, %10};\n"
            : "=d"(d0), "=d"(d1), "=d"(d2), "=d"(d3)
            : "d"(a0), "d"(a1),
              "d"(b0),
              "d"(c0), "d"(c1), "d"(c2), "d"(c3));
    }
};
```

MMA_Op
Raw PTX

```
template <>
struct MMA_Traits<SM90_16x8x4_F64F64F64F64_TN>
{
    using ElementDVal = double;
    using ElementAVal = double;
    using ElementBVal = double;
    using ElementCVal = double;

    using Shape_MNK = Shape<_16,_8,_4>;
    // 32 threads, one warp
    using ThrID      = Layout<_32>;
    // (T32,V8) -> (M16,K4)
    using ALLayout = Layout<Shape<Shape<_4,_8>, _2>,
                           Stride<Stride<_16,_1>, _8>>;
    // (T32,V4) -> (N8,K4)
    using BLLayout = Layout<Shape<Shape<_4,_8>, _1>,
                           Stride<Stride<_8,_1>, _0>>;
    // (T32,V4) -> (M16,N8)
    using CLLayout = Layout<Shape<Shape<_4,_8>, Shape<_2,_2>>,
                           Stride<Stride<_32,_1>, Stride<_16,_8>>>;
};
```

MMA_Traits
PTX meta-info

MMA_Atom

The basic building block

MMA_Op
Raw PTX

MMA_Traits
PTX meta-info

MMA_Atom
Checked call interfaces
Fragment generation

```
MMA_Atom mma = MMA_Atom<SM90_16x8x4_F64F64F64F64_TN>{};  
print_latex(mma);
```

0 1 2 3

0	T0 V0	T1 V0	T2 V0	T3 V0
1	T4 V0	T5 V0	T6 V0	T7 V0
2	T8 V0	T9 V0	T10 V0	T11 V0
3	T12 V0	T13 V0	T14 V0	T15 V0
4	T16 V0	T17 V0	T18 V0	T19 V0
5	T20 V0	T21 V0	T22 V0	T23 V0
6	T24 V0	T25 V0	T26 V0	T27 V0
7	T28 V0	T29 V0	T30 V0	T31 V0
8	T0 V1	T1 V1	T2 V1	T3 V1
9	T4 V1	T5 V1	T6 V1	T7 V1
10	T8 V1	T9 V1	T10 V1	T11 V1
11	T12 V1	T13 V1	T14 V1	T15 V1
12	T16 V1	T17 V1	T18 V1	T19 V1
13	T20 V1	T21 V1	T22 V1	T23 V1
14	T24 V1	T25 V1	T26 V1	T27 V1
15	T28 V1	T29 V1	T30 V1	T31 V1

0	T0 V0	T4 V0	T8 V0	T12 V0	T16 V0	T20 V0	T24 V0	T28 V0
1	T1 V0	T5 V0	T9 V0	T13 V0	T17 V0	T21 V0	T25 V0	T29 V0
2	T2 V0	T6 V0	T10 V0	T14 V0	T18 V0	T22 V0	T26 V0	T30 V0
3	T3 V0	T7 V0	T11 V0	T15 V0	T19 V0	T23 V0	T27 V0	T31 V0
4	T0 V0	T0 V1	T1 V0	T1 V1	T2 V0	T2 V1	T3 V0	T3 V1
5	T4 V0	T4 V1	T5 V0	T5 V1	T6 V0	T6 V1	T7 V0	T7 V1
6	T8 V0	T8 V1	T9 V0	T9 V1	T10 V0	T10 V1	T11 V0	T11 V1
7	T12 V0	T12 V1	T13 V0	T13 V1	T14 V0	T14 V1	T15 V0	T15 V1
8	T16 V0	T16 V1	T17 V0	T17 V1	T18 V0	T18 V1	T19 V0	T19 V1
9	T20 V0	T20 V1	T21 V0	T21 V1	T22 V0	T22 V1	T23 V0	T23 V1
10	T24 V0	T24 V1	T25 V0	T25 V1	T26 V0	T26 V1	T27 V0	T27 V1
11	T28 V0	T28 V1	T29 V0	T29 V1	T30 V0	T30 V1	T31 V0	T31 V1
12	T0 V2	T0 V3	T1 V2	T1 V3	T2 V2	T2 V3	T3 V2	T3 V3
13	T4 V2	T4 V3	T5 V2	T5 V3	T6 V2	T6 V3	T7 V2	T7 V3
14	T8 V2	T8 V3	T9 V2	T9 V3	T10 V2	T10 V3	T11 V2	T11 V3
15	T12 V2	T12 V3	T13 V2	T13 V3	T14 V2	T14 V3	T15 V2	T15 V3
16	T16 V2	T16 V3	T17 V2	T17 V3	T18 V2	T18 V3	T19 V2	T19 V3
17	T20 V2	T20 V3	T21 V2	T21 V3	T22 V2	T22 V3	T23 V2	T23 V3
18	T24 V2	T24 V3	T25 V2	T25 V3	T26 V2	T26 V3	T27 V2	T27 V3
19	T28 V2	T28 V3	T29 V2	T29 V3	T30 V2	T30 V3	T31 V2	T31 V3

Tiled_MMA

Construct larger operations



MMA_Atom
Checked call interfaces
Fragment generation

TiledMMA
Layout of MMA_Atoms
Partition utilities

```

Tiled_MMA mma = make_tiled_mma(SM90_16x8x4_F64F64F64F64_TN{},
                                Layout<Shape<_2,_2>>{}); // 2x2 warps
print_latex(mma);
  
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0_V0	T4_V0	T8_V0	T12_V0	T16_V0	T20_V0	T24_V0	T28_V0	T64_V0	T68_V0	T72_V0	T76_V0	T80_V0	T84_V0	T88_V0	T92_V0
1	T1_V0	T5_V0	T9_V0	T13_V0	T17_V0	T21_V0	T25_V0	T29_V0	T65_V0	T69_V0	T73_V0	T77_V0	T81_V0	T85_V0	T89_V0	T93_V0
2	T2_V0	T6_V0	T10_V0	T14_V0	T18_V0	T22_V0	T26_V0	T30_V0	T66_V0	T70_V0	T74_V0	T78_V0	T82_V0	T86_V0	T90_V0	T94_V0
3	T3_V0	T7_V0	T11_V0	T15_V0	T19_V0	T23_V0	T27_V0	T31_V0	T67_V0	T71_V0	T75_V0	T79_V0	T83_V0	T87_V0	T91_V0	T95_V0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0_V0	T1_V0	T2_V0	T3_V0	T4_V0	T5_V0	T6_V0	T7_V0	T8_V0	T9_V0	T10_V0	T11_V0	T12_V0	T13_V0	T14_V0	T15_V0
1	T4_V0	T5_V0	T6_V0	T7_V0	T8_V0	T9_V0	T10_V0	T11_V0	T12_V0	T13_V0	T14_V0	T15_V0	T16_V0	T17_V0	T18_V0	T19_V0
2	T8_V0	T9_V0	T10_V0	T11_V0	T12_V0	T13_V0	T14_V0	T15_V0	T16_V0	T17_V0	T18_V0	T19_V0	T20_V0	T21_V0	T22_V0	T23_V0
3	T12_V0	T13_V0	T14_V0	T15_V0	T16_V0	T17_V0	T18_V0	T19_V0	T20_V0	T21_V0	T22_V0	T23_V0	T24_V0	T25_V0	T26_V0	T27_V0
4	T16_V0	T17_V0	T18_V0	T19_V0	T20_V0	T21_V0	T22_V0	T23_V0	T24_V0	T25_V0	T26_V0	T27_V0	T28_V0	T29_V0	T30_V0	T31_V0
5	T20_V0	T21_V0	T22_V0	T23_V0	T24_V0	T25_V0	T26_V0	T27_V0	T28_V0	T29_V0	T30_V0	T31_V0	T32_V0	T33_V0	T34_V0	T35_V0
6	T24_V0	T25_V0	T26_V0	T27_V0	T28_V0	T29_V0	T30_V0	T31_V0	T32_V0	T33_V0	T34_V0	T35_V0	T36_V0	T37_V0	T38_V0	T39_V0
7	T28_V0	T29_V0	T30_V0	T31_V0	T20_V0	T21_V0	T22_V0	T23_V0	T24_V0	T25_V0	T26_V0	T27_V0	T28_V0	T29_V0	T20_V0	T21_V0
8	T0_V1	T1_V1	T2_V1	T3_V1	T4_V1	T5_V1	T6_V1	T7_V1	T8_V1	T9_V1	T10_V1	T11_V1	T12_V1	T13_V1	T14_V1	T15_V1
9	T4_V1	T5_V1	T6_V1	T7_V1	T8_V1	T9_V1	T10_V1	T11_V1	T12_V1	T13_V1	T14_V1	T15_V1	T16_V1	T17_V1	T18_V1	T19_V1
10	T8_V1	T9_V1	T10_V1	T11_V1	T12_V1	T13_V1	T14_V1	T15_V1	T16_V1	T17_V1	T18_V1	T19_V1	T20_V1	T21_V1	T22_V1	T23_V1
11	T12_V1	T13_V1	T14_V1	T15_V1	T16_V1	T17_V1	T18_V1	T19_V1	T20_V1	T21_V1	T22_V1	T23_V1	T24_V1	T25_V1	T26_V1	T27_V1
12	T16_V1	T17_V1	T18_V1	T19_V1	T20_V1	T21_V1	T22_V1	T23_V1	T24_V1	T25_V1	T26_V1	T27_V1	T28_V1	T29_V1	T30_V1	T31_V1
13	T20_V1	T21_V1	T22_V1	T23_V1	T24_V1	T25_V1	T26_V1	T27_V1	T28_V1	T29_V1	T30_V1	T31_V1	T32_V1	T33_V1	T34_V1	T35_V1
14	T24_V1	T25_V1	T26_V1	T27_V1	T28_V1	T29_V1	T30_V1	T31_V1	T32_V1	T33_V1	T34_V1	T35_V1	T36_V1	T37_V1	T38_V1	T39_V1
15	T28_V1	T29_V1	T30_V1	T31_V1	T32_V1	T33_V1	T34_V1	T35_V1	T36_V1	T37_V1	T38_V1	T39_V1	T40_V1	T41_V1	T42_V1	T43_V1
16	T32_V0	T33_V0	T34_V0	T35_V0	T36_V0	T37_V0	T38_V0	T39_V0	T40_V0	T41_V0	T42_V0	T43_V0	T44_V0	T45_V0	T46_V0	T47_V0
17	T36_V0	T37_V0	T38_V0	T39_V0	T40_V0	T41_V0	T42_V0	T43_V0	T44_V0	T45_V0	T46_V0	T47_V0	T48_V0	T49_V0	T50_V0	T51_V0
18	T40_V0	T41_V0	T42_V0	T43_V0	T44_V0	T45_V0	T46_V0	T47_V0	T48_V0	T49_V0	T50_V0	T51_V0	T52_V0	T53_V0	T54_V0	T55_V0
19	T44_V0	T45_V0	T46_V0	T47_V0	T48_V0	T49_V0	T50_V0	T51_V0	T52_V0	T53_V0	T54_V0	T55_V0	T56_V0	T57_V0	T58_V0	T59_V0
20	T48_V0	T49_V0	T50_V0	T51_V0	T52_V0	T53_V0	T54_V0	T55_V0	T56_V0	T57_V0	T58_V0	T59_V0	T60_V0	T61_V0	T62_V0	T63_V0
21	T52_V0	T53_V0	T54_V0	T55_V0	T56_V0	T57_V0	T58_V0	T59_V0	T60_V0	T61_V0	T62_V0	T63_V0	T64_V0	T65_V0	T66_V0	T67_V0
22	T56_V0	T57_V0	T58_V0	T59_V0	T60_V0	T61_V0	T62_V0	T63_V0	T64_V0	T65_V0	T66_V0	T67_V0	T68_V0	T69_V0	T70_V0	T71_V0
23	T60_V0	T61_V0	T62_V0	T63_V0	T64_V0	T65_V0	T66_V0	T67_V0	T68_V0	T69_V0	T70_V0	T71_V0	T72_V0	T73_V0	T74_V0	T75_V0
24	T32_V1	T33_V1	T34_V1	T35_V1	T36_V1	T37_V1	T38_V1	T39_V1	T40_V1	T41_V1	T42_V1	T43_V1	T44_V1	T45_V1	T46_V1	T47_V1
25	T36_V1	T37_V1	T38_V1	T39_V1	T40_V1	T41_V1	T42_V1	T43_V1	T44_V1	T45_V1	T46_V1	T47_V1	T48_V1	T49_V1	T50_V1	T51_V1
26	T40_V1	T41_V1	T42_V1	T43_V1	T44_V1	T45_V1	T46_V1	T47_V1	T48_V1	T49_V1	T50_V1	T51_V1	T52_V1	T53_V1	T54_V1	T55_V1
27	T44_V1	T45_V1	T46_V1	T47_V1	T48_V1	T49_V1	T50_V1	T51_V1	T52_V1	T53_V1	T54_V1	T55_V1	T56_V1	T57_V1	T58_V1	T59_V1
28	T48_V1	T49_V1	T50_V1	T51_V1	T52_V1	T53_V1	T54_V1	T55_V1	T56_V1	T57_V1	T58_V1	T59_V1	T60_V1	T61_V1	T62_V1	T63_V1
29	T52_V1	T53_V1	T54_V1	T55_V1	T56_V1	T57_V1	T58_V1	T59_V1	T60_V1	T61_V1	T62_V1	T63_V1	T64_V1	T65_V1	T66_V1	T67_V1
30	T56_V1	T57_V1	T58_V1	T59_V1	T60_V1	T61_V1	T62_V1	T63_V1	T64_V1	T65_V1	T66_V1	T67_V1	T68_V1	T69_V1	T70_V1	T71_V1
31	T60_V1	T61_V1	T62_V1	T63_V1	T64_V1	T65_V1	T66_V1	T67_V1	T68_V1	T69_V1	T70_V1	T71_V1	T72_V1	T73_V1	T74_V1	T75_V1

Tiled_MMA

Construct larger operations



```

Tiled_MMA mma = make_tiled_mma(SM90_16x8x4_F64F64F64F64_TN{},
                                Layout<Shape<_2,_2>>{},           // 2x2 warps
                                Layout<Shape<_1,_1>>{},           // 1x1 vals
                                tuple<Layout<_2,_8>>{});        // M-perm
  
```

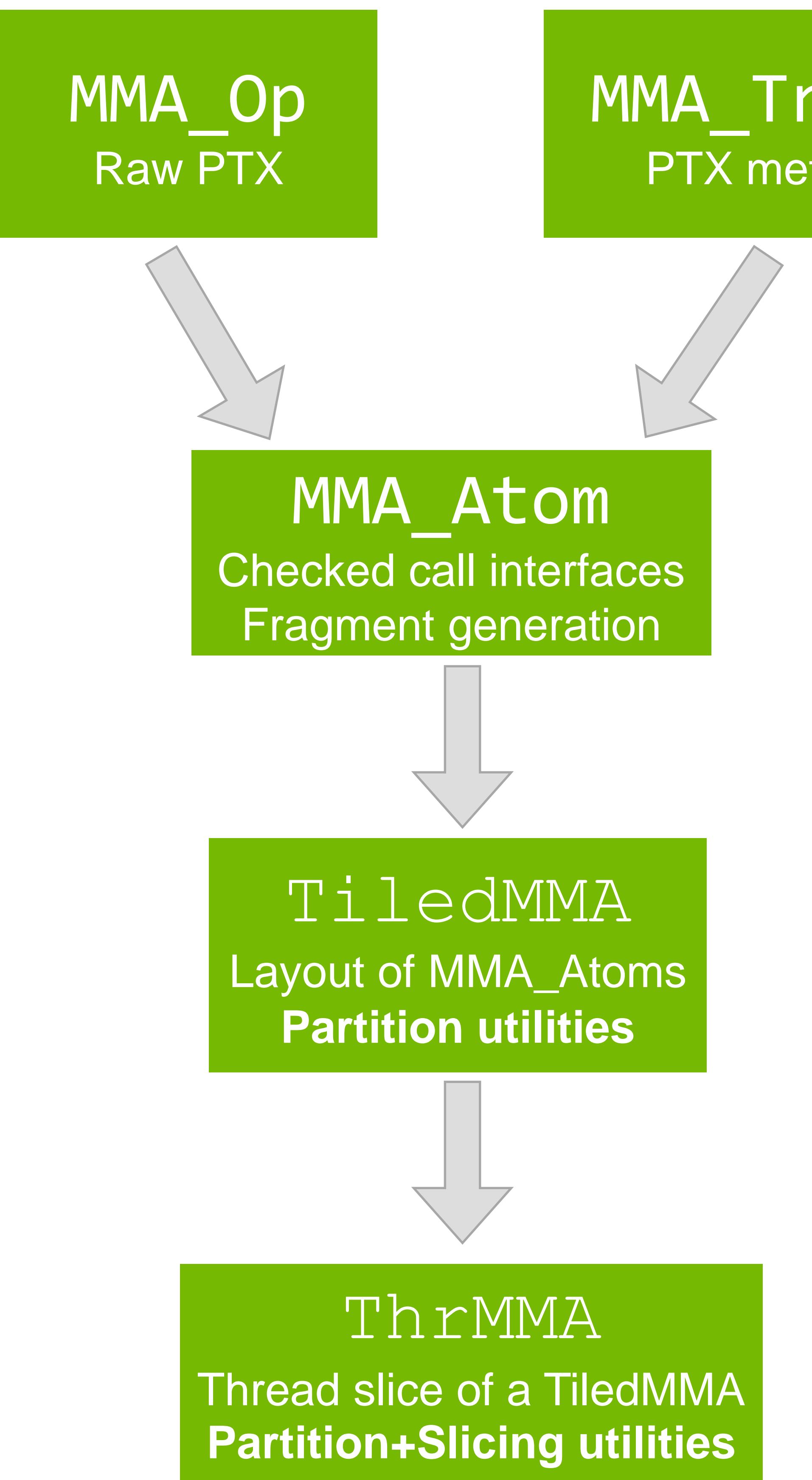
`print_latex(mma);`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0 V0	T4 V0	T8 V0	T12 V0	T16 V0	T20 V0	T24 V0	T28 V0	T64 V0	T68 V0	T72 V0	T76 V0	T80 V0	T84 V0	T88 V0	T92 V0
1	T1 V0	T5 V0	T9 V0	T13 V0	T17 V0	T21 V0	T25 V0	T29 V0	T65 V0	T69 V0	T73 V0	T77 V0	T81 V0	T85 V0	T89 V0	T93 V0
2	T2 V0	T6 V0	T10 V0	T14 V0	T18 V0	T22 V0	T26 V0	T30 V0	T66 V0	T70 V0	T74 V0	T78 V0	T82 V0	T86 V0	T90 V0	T94 V0
3	T3 V0	T7 V0	T11 V0	T15 V0	T19 V0	T23 V0	T27 V0	T31 V0	T67 V0	T71 V0	T75 V0	T79 V0	T83 V0	T87 V0	T91 V0	T95 V0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0 V0	T1 V0	T2 V0	T3 V0	T4 V0	T5 V0	T6 V0	T7 V0	T8 V0	T9 V0	T10 V0	T11 V0	T12 V0	T13 V0	T14 V0	T15 V0
1	T0 V2	T1 V1	T2 V1	T3 V1	T4 V0	T5 V1	T6 V0	T7 V0	T8 V1	T9 V1	T10 V0	T11 V0	T12 V0	T13 V0	T14 V0	T15 V0
2	T4 V0	T5 V0	T6 V0	T7 V0	T4 V1	T5 V1	T6 V1	T7 V1	T4 V2	T5 V2	T6 V2	T7 V2	T4 V3	T5 V3	T6 V3	T7 V3
3	T8 V0	T9 V0	T10 V0	T11 V0	T8 V1	T9 V1	T10 V1	T11 V1	T8 V2	T9 V2	T10 V2	T11 V2	T8 V3	T9 V3	T10 V3	T11 V3
4	T12 V0	T13 V0	T14 V0	T15 V0	T12 V1	T13 V1	T14 V1	T15 V1	T12 V2	T13 V2	T14 V2	T15 V2	T12 V3	T13 V3	T14 V3	T15 V3
5	T12 V1	T13 V1	T14 V1	T15 V1	T16 V0	T17 V0	T18 V0	T19 V0	T16 V1	T17 V1	T18 V1	T19 V1	T16 V2	T17 V2	T18 V2	T19 V2
6	T16 V0	T17 V0	T18 V0	T19 V0	T20 V0	T21 V0	T22 V0	T23 V0	T20 V1	T21 V1	T22 V1	T23 V1	T20 V2	T21 V2	T22 V2	T23 V2
7	T20 V1	T21 V1	T22 V1	T23 V1	T24 V0	T25 V0	T26 V0	T27 V0	T24 V1	T25 V1	T26 V1	T27 V1	T24 V2	T25 V2	T26 V2	T27 V2
8	T24 V1	T25 V1	T26 V1	T27 V1	T28 V0	T29 V0	T30 V0	T31 V0	T28 V1	T29 V1	T30 V1	T31 V1	T28 V2	T29 V2	T30 V2	T31 V2
9	T28 V1	T29 V1	T30 V1	T31 V1	T32 V1	T33 V1	T34 V1	T35 V1	T32 V2	T33 V2	T34 V2	T35 V2	T32 V3	T33 V3	T34 V3	T35 V3
10	T32 V0	T33 V0	T34 V0	T35 V0	T36 V0	T37 V0	T38 V0	T39 V0	T32 V1	T33 V1	T34 V1	T35 V1	T32 V2	T33 V2	T34 V2	T35 V2
11	T36 V0	T37 V0	T38 V0	T39 V0	T36 V1	T37 V1	T38 V1	T39 V1	T36 V2	T37 V2	T38 V2	T39 V2	T36 V3	T37 V3	T38 V3	T39 V3
12	T36 V1	T37 V1	T38 V1	T39 V1	T36 V2	T37 V2	T38 V2	T39 V2	T36 V3	T37 V3	T38 V3	T39 V3	T36 V4	T37 V4	T38 V4	T39 V4
13	T36 V2	T37 V2	T38 V2	T39 V2	T36 V3	T37 V3	T38 V3	T39 V3	T36 V4	T37 V4	T38 V4	T39 V4	T36 V5	T37 V5	T38 V5	T39 V5
14	T36 V3	T37 V3	T38 V3	T39 V3	T36 V4	T37 V4	T38 V4	T39 V4	T36 V5	T37 V5	T38 V5	T39 V5	T36 V6	T37 V6	T38 V6	T39 V6
15	T36 V4	T37 V4	T38 V4	T39 V4	T36 V5	T37 V5	T38 V5	T39 V5	T36 V6	T37 V6	T38 V6	T39 V6	T36 V7	T37 V7	T38 V7	T39 V7
16	T36 V5	T37 V5	T38 V5	T39 V5	T36 V6	T37 V6	T38 V6	T39 V6	T36 V7	T37 V7	T38 V7	T39 V7	T36 V8	T37 V8	T38 V8	T39 V8
17	T36 V6	T37 V6	T38 V6	T39 V6	T36 V7	T37 V7	T38 V7	T39 V7	T36 V8	T37 V8	T38 V8	T39 V8	T36 V9	T37 V9	T38 V9	T39 V9
18	T36 V7	T37 V7	T38 V7	T39 V7	T36 V8	T37 V8	T38 V8	T39 V8	T36 V9	T37 V9	T38 V9	T39 V9	T36 V10	T37 V10	T38 V10	T39 V10
19	T36 V8	T37 V8	T38 V8	T39 V8	T36 V9	T37 V9	T38 V9	T39 V9	T36 V10	T37 V10	T38 V10	T39 V10	T36 V11	T37 V11	T38 V11	T39 V11
20	T36 V9	T37 V9	T38 V9	T39 V9	T36 V10	T37 V10	T38 V10	T39 V10	T36 V11	T37 V11	T38 V11	T39 V11	T36 V12	T37 V12	T38 V12	T39 V12
21	T36 V10	T37 V10	T38 V10	T39 V10	T36 V11	T37 V11	T38 V11	T39 V11	T36 V12	T37 V12	T38 V12	T39 V12	T36 V13	T37 V13	T38 V13	T39 V13
22	T36 V11	T37 V11	T38 V11	T39 V11	T36 V12	T37 V12	T38 V12	T39 V12	T36 V13	T37 V13	T38 V13	T39 V13	T36 V14	T37 V14	T38 V14	T39 V14
23	T36 V12	T37 V12	T38 V12	T39 V12	T36 V13	T37 V13	T38 V13	T39 V13	T36 V14	T37 V14	T38 V14	T39 V14	T36 V15	T37 V15	T38 V15	T39 V15
24	T36 V13	T37 V13	T38 V13	T39 V13	T36 V14	T37 V14	T38 V14	T39 V14	T36 V15	T37 V15	T38 V15	T39 V15	T36 V16	T37 V16	T38 V16	T39 V16
25	T36 V14	T37 V14	T38 V14	T39 V14	T36 V15	T37 V15	T38 V15	T39 V15	T36 V16	T37 V16	T38 V16	T39 V16	T36 V17	T37 V17	T38 V17	T39 V17
26	T36 V15	T37 V15	T38 V15	T39 V15	T36 V16	T37 V16	T38 V16	T39 V16	T36 V17	T37 V17	T38 V17	T39 V17	T36 V18	T37 V18	T38 V18	T39 V18
27	T36 V16	T37 V16	T38 V16	T39 V16	T36 V17	T37 V17	T38 V17	T39 V17	T36 V18	T37 V18	T38 V18	T39 V18	T36 V19	T37 V19	T38 V19	T39 V19
28	T36 V17	T37 V17	T38 V17	T39 V17	T36 V18	T37 V18	T38 V18	T39 V18	T36 V19	T37 V19	T38 V19	T39 V19	T36 V20	T37 V20	T38 V20	T39 V20
29	T36 V18	T37 V18	T38 V18	T39 V18	T36 V19	T37 V19	T38 V19	T39 V19	T36 V20	T37 V20	T38 V20	T39 V20	T36 V21			

Thread slice of Tiled_MMA

A thread's view

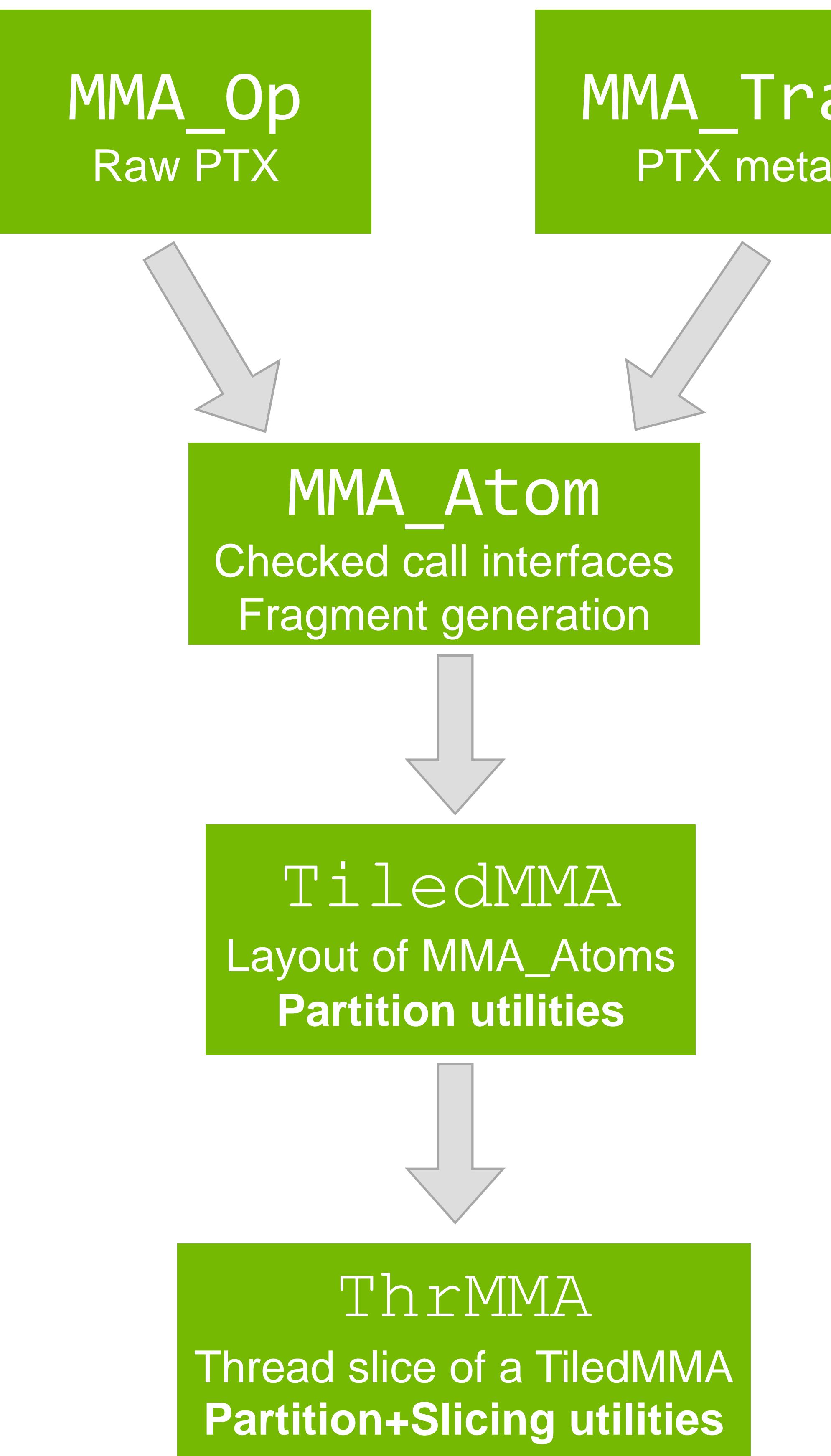


`ThrMMA thr_mma = tiled_mma.get_slice(threadIdx.x);`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
0	T0 V0	T4 V0	T8 V0	T12 V0	T16 V0	T20 V0	T24 V0	T28 V0	T64 V0	T68 V0	T72 V0	T76 V0	T80 V0	T84 V0	T88 V0	T92 V0		
1	T1 V0	T5 V0	T9 V0	T13 V0	T17 V0	T21 V0	T25 V0	T29 V0	T65 V0	T69 V0	T73 V0	T77 V0	T81 V0	T85 V0	T89 V0	T93 V0		
2	T2 V0	T6 V0	T10 V0	T14 V0	T18 V0	T22 V0	T26 V0	T30 V0	T66 V0	T70 V0	T74 V0	T78 V0	T82 V0	T86 V0	T90 V0	T94 V0		
3	T3 V0	T7 V0	T11 V0	T15 V0	T19 V0	T23 V0	T27 V0	T31 V0	T67 V0	T71 V0	T75 V0	T79 V0	T83 V0	T87 V0	T91 V0	T95 V0		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
0	T0 V0	T1 V0	T2 V0	T3 V0	T0 V0	T1 V1	T1 V0	T2 V1	T2 V0	T3 V0	T3 V1	T64 V0	T64 V1	T65 V0	T65 V1	T66 V0	T66 V1	
1	T0 V1	T1 V1	T2 V1	T3 V1	T0 V2	T0 V3	T1 V3	T2 V2	T2 V3	T3 V3	T64 V2	T64 V3	T65 V2	T65 V3	T66 V2	T66 V3	T67 V2	T67 V3
2	T4 V0	T5 V0	T6 V0	T7 V0	T4 V1	T4 V2	T5 V1	T5 V2	T6 V1	T6 V2	T7 V1	T7 V2	T68 V0	T68 V1	T69 V0	T69 V1	T70 V0	T70 V1
3	T4 V1	T5 V1	T6 V1	T7 V1	T8 V0	T9 V0	T10 V0	T11 V0	T8 V1	T9 V1	T10 V0	T11 V0	T11 V1	T12 V0	T12 V1	T13 V0	T14 V1	
4	T8 V0	T9 V0	T10 V0	T11 V0	T8 V1	T9 V1	T10 V1	T11 V1	T8 V2	T9 V2	T10 V2	T11 V2	T11 V3	T12 V0	T12 V1	T13 V0	T14 V1	
5	T8 V1	T9 V1	T10 V1	T11 V1	T12 V0	T13 V0	T14 V0	T15 V0	T12 V1	T13 V1	T14 V1	T15 V1	T15 V2	T16 V0	T16 V1	T17 V0	T18 V1	
6	T12 V0	T13 V0	T14 V0	T15 V0	T12 V1	T13 V1	T14 V1	T15 V1	T12 V2	T13 V2	T14 V2	T15 V2	T15 V3	T16 V0	T16 V1	T17 V0	T18 V1	
7	T12 V1	T13 V1	T14 V1	T15 V1	T12 V2	T13 V2	T14 V2	T15 V2	T12 V3	T13 V3	T14 V3	T15 V3	T15 V4	T16 V0	T16 V1	T17 V0	T18 V1	
8	T16 V0	T17 V0	T18 V0	T19 V0	T16 V1	T17 V1	T18 V1	T19 V1	T16 V2	T17 V2	T18 V2	T19 V2	T19 V3	T20 V0	T20 V1	T21 V0	T22 V1	
9	T16 V1	T17 V1	T18 V1	T19 V1	T20 V0	T21 V0	T22 V0	T23 V0	T16 V2	T17 V2	T18 V2	T19 V2	T19 V3	T20 V1	T20 V2	T21 V1	T22 V2	
10	T20 V0	T21 V0	T22 V0	T23 V0	T20 V1	T21 V1	T22 V1	T23 V1	T20 V2	T21 V2	T22 V2	T23 V2	T23 V3	T24 V0	T24 V1	T25 V0	T26 V1	
11	T20 V1	T21 V1	T22 V1	T23 V1	T20 V2	T21 V2	T22 V2	T23 V2	T20 V3	T21 V3	T22 V3	T23 V3	T23 V4	T24 V1	T24 V2	T25 V1	T26 V2	
12	T24 V0	T25 V0	T26 V0	T27 V0	T24 V1	T25 V1	T26 V1	T27 V1	T24 V2	T25 V2	T26 V2	T27 V2	T27 V3	T28 V0	T28 V1	T29 V0	T29 V1	
13	T24 V1	T25 V1	T26 V1	T27 V1	T24 V2	T25 V2	T26 V2	T27 V2	T24 V3	T25 V3	T26 V3	T27 V3	T27 V4	T28 V1	T28 V2	T29 V1	T29 V2	
14	T28 V0	T29 V0	T30 V0	T31 V0	T28 V1	T29 V1	T30 V1	T31 V1	T28 V2	T29 V2	T30 V2	T31 V2	T31 V3	T32 V0	T32 V1	T33 V0	T33 V1	
15	T28 V1	T29 V1	T30 V1	T31 V1	T28 V2	T29 V2	T30 V2	T31 V2	T28 V3	T29 V3	T30 V3	T31 V3	T31 V4	T32 V0	T32 V1	T33 V0	T33 V1	
16	T32 V0	T33 V0	T34 V0	T35 V0	T32 V1	T33 V1	T34 V1	T35 V1	T32 V2	T33 V2	T34 V2	T35 V2	T35 V3	T36 V0	T36 V1	T37 V0	T37 V1	
17	T32 V1	T33 V1	T34 V1	T35 V1	T32 V2	T33 V2	T34 V2	T35 V2	T32 V3	T33 V3	T34 V3	T35 V3	T35 V4	T36 V1	T36 V2	T37 V1	T37 V2	
18	T36 V0	T37 V0	T38 V0	T39 V0	T36 V1	T37 V1	T38 V1	T39 V1	T36 V2	T37 V2	T38 V2	T39 V2	T39 V3	T40 V0	T40 V1	T41 V0	T41 V1	
19	T36 V1	T37 V1	T38 V1	T39 V1	T36 V2	T37 V2	T38 V2	T39 V2	T36 V3	T37 V3	T38 V3	T39 V3	T39 V4	T40 V1	T40 V2	T41 V1	T41 V2	
20	T40 V0	T41 V0	T42 V0	T43 V0	T40 V1	T41 V1	T42 V1	T43 V1	T40 V2	T41 V2	T42 V2	T43 V2	T43 V3	T44 V0	T44 V1	T45 V0	T45 V1	
21	T40 V1	T41 V1	T42 V1	T43 V1	T40 V2	T41 V2	T42 V2	T43 V2	T40 V3	T41 V3	T42 V3	T43 V3	T43 V4	T44 V1	T44 V2	T45 V1	T45 V2	
22	T44 V0	T45 V0	T46 V0	T47 V0	T44 V1	T45 V1	T46 V1	T47 V1	T44 V2	T45 V2	T46 V2	T47 V2	T47 V3	T48 V0	T48 V1	T49 V0	T49 V1	
23	T44 V1	T45 V1	T46 V1	T47 V1	T44 V2	T45 V2	T46 V2	T47 V2	T44 V3	T45 V3	T46 V3	T47 V3	T47 V4	T48 V1	T48 V2	T49 V1	T49 V2	
24	T48 V0	T49 V0	T50 V0	T51 V0	T48 V1	T49 V1	T50 V1	T51 V1	T48 V2	T49 V2	T50 V2	T51 V2	T51 V3	T52 V0	T52 V1	T53 V0	T53 V1	
25	T48 V1	T49 V1	T50 V1	T51 V1	T48 V2	T49 V2	T50 V2	T51 V2	T48 V3	T49 V3	T50 V3	T51 V3	T51 V4	T52 V1	T52 V2	T53 V1	T53 V2	
26	T52 V0	T53 V0	T54 V0	T55 V0	T52 V1	T53 V1	T54 V1	T55 V1	T52 V2	T53 V2	T54 V2	T55 V2	T55 V3	T56 V0	T56 V1	T57 V0	T57 V1	
27	T52 V1	T53 V1	T54 V1	T55 V1	T52 V2	T53 V2	T54 V2	T55 V2	T52 V3	T53 V3	T54 V3	T55 V3	T55 V4	T56 V1	T56 V2	T57 V1	T57 V2	
28	T56 V0	T57 V0	T58 V0	T59 V0	T56 V1	T57 V1	T58 V1	T59 V1	T56 V2	T57 V2	T58 V2	T59 V2	T59 V3	T60 V0	T60 V1	T61 V0	T61 V1	
29	T56 V1	T57 V1	T58 V1	T59 V1	T56 V2	T57 V2	T58 V2	T59 V2	T56 V3	T57 V3	T58 V3	T59 V3	T59 V4	T60 V1	T60 V2	T61 V1	T61 V2	
30	T60 V0	T61 V0	T62 V0	T63 V0	T60 V1	T61 V1	T62 V1	T63 V1	T60 V2	T61 V2	T62 V2	T63 V2	T63 V3	T64 V0	T64 V1	T65 V0	T65 V1	
31	T60 V1	T61 V1	T62 V1	T63 V1	T60 V2	T61 V2	T62 V2	T63 V2</td										

Thread slice of Tiled_MMA

A thread's view



```

Tensor A = make_tensor(ptrA, Shape<_64,_16>{}); // 64x16
Tensor B = make_tensor(ptrB, Shape<_32,_16>{}); // 32x16
Tensor C = make_tensor(ptrC, Shape<_64,_32>{}); // 64x32

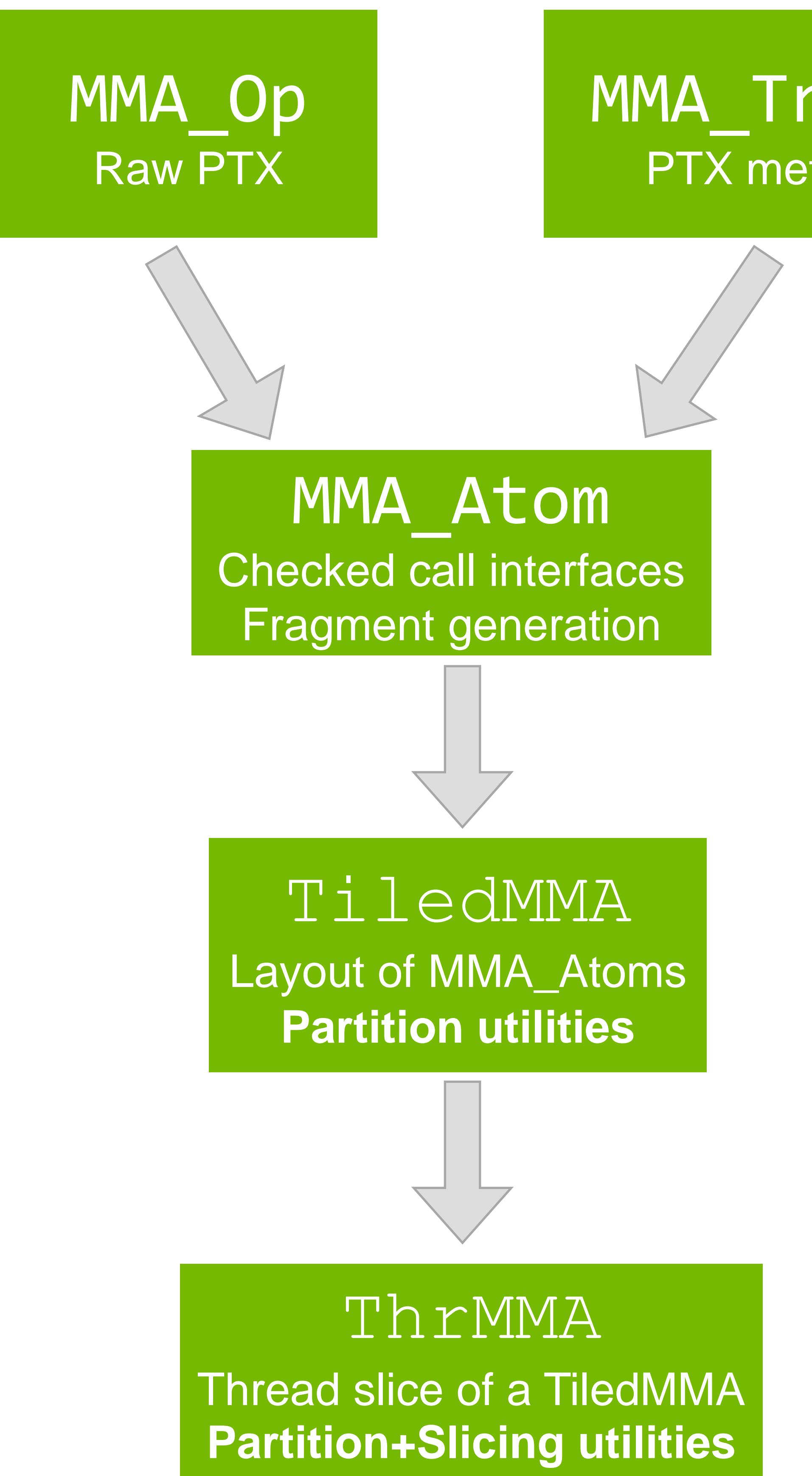
ThrMMA thr_mma = mma.get_slice(threadIdx.x);

gemm(thr_mma, A, B, C);
  
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0 V0	T4 V0	T8 V0	T12 V0	T16 V0	T20 V0	T24 V0	T28 V0	T64 V0	T68 V0	T72 V0	T76 V0	T80 V0	T84 V0	T88 V0	T92 V0
1	T1 V0	T5 V0	T9 V0	T13 V0	T17 V0	T21 V0	T25 V0	T29 V0	T65 V0	T69 V0	T73 V0	T77 V0	T81 V0	T85 V0	T89 V0	T93 V0
2	T2 V0	T6 V0	T10 V0	T14 V0	T18 V0	T22 V0	T26 V0	T30 V0	T66 V0	T70 V0	T74 V0	T78 V0	T82 V0	T86 V0	T90 V0	T94 V0
3	T3 V0	T7 V0	T11 V0	T15 V0	T19 V0	T23 V0	T27 V0	T31 V0	T67 V0	T71 V0	T75 V0	T79 V0	T83 V0	T87 V0	T91 V0	T95 V0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0 V0	T1 V0	T2 V0	T3 V0	T4 V0	T5 V0	T6 V0	T7 V0	T8 V0	T9 V0	T10 V0	T11 V0	T12 V0	T13 V0	T14 V0	T15 V0
1	T0 V2	T1 V3	T2 V2	T3 V1	T4 V1	T5 V0	T6 V0	T7 V0	T8 V0	T9 V0	T10 V0	T11 V0	T12 V0	T13 V0	T14 V0	T15 V0
2	T4 V0	T5 V0	T6 V0	T7 V0	T4 V2	T5 V3	T6 V2	T7 V3	T7 V0	T68 V1	T69 V0	T69 V1	T70 V0	T70 V1	T71 V0	T71 V1
3	T4 V1	T5 V1	T6 V1	T7 V1	T8 V0	T9 V0	T10 V0	T11 V0	T11 V0	T72 V0	T72 V1	T73 V0	T73 V1	T74 V0	T74 V1	T75 V0
4	T8 V0	T9 V0	T10 V0	T11 V0	T8 V1	T9 V1	T10 V1	T11 V0	T11 V0	T72 V1	T73 V0	T73 V1	T74 V0	T74 V1	T75 V0	T75 V1
5	T8 V1	T9 V1	T10 V1	T11 V1	T8 V2	T9 V2	T10 V2	T11 V2	T11 V3	T72 V3	T73 V2	T73 V3	T74 V2	T74 V3	T75 V2	T75 V3
6	T12 V0	T13 V0	T14 V0	T15 V0	T12 V1	T13 V1	T13 V0	T14 V1	T14 V0	T15 V0	T15 V1	T15 V0	T16 V0	T17 V0	T17 V1	T18 V0
7	T12 V1	T13 V1	T14 V1	T15 V1	T12 V2	T13 V3	T14 V2	T15 V2	T15 V3	T76 V2	T76 V3	T77 V2	T77 V3	T78 V2	T78 V3	T79 V2
8	T16 V0	T17 V0	T18 V0	T19 V0	T16 V1	T17 V1	T17 V0	T18 V1	T18 V0	T19 V0	T19 V1	T19 V0	T20 V0	T21 V0	T22 V0	T23 V0
9	T16 V1	T17 V1	T18 V1	T19 V1	T16 V2	T17 V3	T17 V2	T18 V3	T18 V2	T19 V3	T19 V2	T19 V3	T20 V1	T21 V1	T22 V1	T23 V1
10	T20 V0	T21 V0	T22 V0	T23 V0	T20 V1	T21 V1	T22 V1	T23 V0	T23 V0	T24 V0	T24 V1	T24 V0	T25 V0	T26 V0	T27 V0	T28 V0
11	T20 V1	T21 V1	T22 V1	T23 V1	T20 V2	T21 V3	T22 V2	T23 V3	T23 V2	T24 V2	T25 V3	T25 V2	T26 V1	T27 V1	T27 V2	T28 V1
12	T24 V0	T25 V0	T26 V0	T27 V0	T24 V1	T25 V1	T25 V0	T26 V1	T26 V0	T27 V0	T27 V1	T28 V0	T29 V0	T29 V1	T29 V0	T29 V1
13	T24 V1	T25 V1	T26 V1	T27 V1	T24 V2	T25 V3	T25 V2	T26 V3	T26 V2	T27 V3	T27 V2	T28 V2	T28 V3	T28 V2	T29 V2	T29 V3
14	T28 V0	T29 V0	T30 V0	T31 V0	T28 V1	T29 V1	T29 V0	T30 V1	T30 V0	T31 V0	T31 V1	T31 V0	T32 V0	T32 V1	T32 V0	T33 V0
15	T28 V1	T29 V1	T30 V1	T31 V1	T28 V2	T29 V3	T29 V2	T30 V3	T30 V2	T31 V3	T31 V2	T32 V2	T32 V3	T32 V2	T33 V2	T33 V3
16	T32 V0	T33 V0	T34 V0	T35 V0	T32 V1	T33 V1	T34 V1	T35 V0	T35 V0	T33 V0	T34 V0	T35 V0	T32 V1	T33 V1	T34 V1	T35 V0
17	T32 V1	T33 V1	T34 V1	T35 V1	T32 V2	T33 V3	T33 V2	T34 V3	T34 V2	T35 V3	T35 V2	T36 V3	T36 V2	T37 V2	T37 V3	T38 V2
18	T36 V0	T37 V0	T38 V0	T39 V0	T36 V1	T37 V1	T38 V1	T39 V0	T39 V0	T38 V1	T39 V1	T39 V0	T40 V0	T41 V0	T42 V0	T43 V0
19	T36 V1	T37 V1	T38 V1	T39 V1	T36 V2	T37 V3	T37 V2	T38 V3	T38 V2	T39 V3	T39 V2	T40 V1	T41 V1	T42 V1	T43 V1	T44 V0
20	T40 V0	T41 V0	T42 V0	T43 V0	T40 V1	T41 V1	T42 V1	T43 V0	T43 V0	T42 V1	T43 V1	T44 V0	T45 V0	T46 V0	T47 V0	T48 V0
21	T40 V1	T41 V1	T42 V1	T43 V1	T40 V2	T41 V3	T41 V2	T42 V3	T42 V2	T43 V3	T43 V2	T44 V1	T45 V1	T46 V1	T47 V1	T48 V1
22	T44 V0	T45 V0	T46 V0	T47 V0	T44 V1	T45 V1	T46 V1	T47 V0	T47 V0	T48 V0	T49 V0	T49 V1	T50 V0	T51 V0	T51 V1	T52 V0
23	T44 V1	T45 V1	T46 V1	T47 V1	T44 V2	T45 V3	T45 V2	T46 V3	T46 V2	T47 V3	T47 V2	T48 V1	T49 V1	T50 V1	T51 V0	T52 V1
24	T48 V0	T49 V0	T50 V0	T51 V0	T48 V1	T49 V1	T49 V0	T50 V1	T50 V0	T51 V0	T51 V1	T52 V0	T53 V0	T54 V0	T55 V0	T56 V0
25	T48 V1	T49 V1	T50 V1	T51 V1	T48 V2	T49 V2	T49 V1	T50 V2	T50 V1	T51 V2	T51 V1	T52 V1	T53 V1	T54 V1	T55 V1	T56 V1
26	T52 V0	T53 V0	T54 V0	T55 V0	T52 V1	T53 V1	T54 V1	T55 V0	T55 V0	T54 V1	T55 V1	T56 V0	T57 V0	T58 V0	T59 V0	T60 V0
27	T52 V1	T53 V1	T54 V1	T55 V1	T52 V2	T53 V3	T53 V2	T54 V3	T54 V2	T55 V3	T55 V2	T56 V1	T57 V1	T58 V1	T59 V0	T60 V1
28	T56 V0	T57 V0	T58 V0	T59 V0	T56 V1	T57 V1	T58 V1	T59 V0	T59 V0	T58 V1	T59 V1	T56 V0	T57 V0	T58 V0	T59 V1	T60 V0
29	T56 V1	T57 V1	T58 V1	T59 V1	T56 V2	T57 V3	T57 V2	T58 V3	T58 V2	T59 V3	T59 V2	T56 V1	T57 V1	T58 V1	T59 V0	T60 V1
30	T60 V0	T61 V0	T62 V0	T63 V0	T60 V1	T61 V1	T62 V1	T63 V0	T63 V0	T62 V1	T63 V1	T60 V0	T61 V0	T62 V0	T63 V1	T64 V0
31	T60 V1	T61 V1	T62 V1	T63 V1	T60 V2	T61 V3	T61 V2	T62 V3	T62 V2	T63 V3	T63 V2	T64 V1	T65 V1	T66 V0	T67 V1	T68 V0

Thread slice of Tiled_MMA

A thread's view



```

Tensor A = make_tensor(ptrA, Shape<_64,_16>{}); // 64x16
Tensor B = make_tensor(ptrB, Shape<_32,_16>{}); // 32x16
Tensor C = make_tensor(ptrC, Shape<_64,_32>{}); // 64x32

ThrMMA thr_mma = mma.get_slice(threadIdx.x);
Tensor thr_A = thr_mma.partition_A(A); // 2xMxK
Tensor thr_B = thr_mma.partition_B(B); // 1xNxK
Tensor thr_C = thr_mma.partition_C(C);

Tensor rA = make_tensor<double>(shape(thr_A));
Tensor rB = make_tensor<double>(shape(thr_B));
Tensor rC = make_tensor<double>(shape(thr_C));

copy(thr_A, rA);
copy(thr_B, rB);
clear(rC);

for (int m = 0; m < size<1>(rC); ++m) {
    for (int n = 0; n < size<2>(rC); ++n) {
        for (int k = 0; k < size<2>(rA); ++k) {
            mma.call(rA(_,m,k), rB(_,n,k), rC(_,m,n));
        }
    }
}

copy(rC, thr_C);
  
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	T0 V0	T4 V0	T8 V0	T12 V0	T16 V0	T20 V0	T24 V0	T28 V0	T64 V0	T68 V0	T72 V0	T76 V0	T80 V0	T84 V0	T88 V0	T92 V0	
1	T1 V0	T5 V0	T9 V0	T13 V0	T17 V0	T21 V0	T25 V0	T29 V0	T65 V0	T69 V0	T73 V0	T77 V0	T81 V0	T85 V0	T89 V0	T93 V0	
2	T2 V0	T6 V0	T10 V0	T14 V0	T18 V0	T22 V0	T26 V0	T30 V0	T66 V0	T70 V0	T74 V0	T78 V0	T82 V0	T86 V0	T90 V0	T94 V0	
3	T3 V0	T7 V0	T11 V0	T15 V0	T19 V0	T23 V0	T27 V0	T31 V0	T67 V0	T71 V0	T75 V0	T79 V0	T83 V0	T87 V0	T91 V0	T95 V0	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	T0 V0	T1 V0	T2 V0	T3 V0	T4 V0	T5 V0	T6 V0	T7 V0	T8 V0	T9 V0	T10 V0	T11 V0	T12 V0	T13 V0	T14 V0	T15 V0	
1	T0 V2	T1 V3	T2 V2	T3 V1	T4 V1	T5 V0	T6 V0	T7 V0	T8 V1	T9 V1	T10 V0	T11 V0	T12 V0	T13 V0	T14 V0	T15 V0	
2	T4 V0	T5 V0	T6 V0	T7 V0	T4 V2	T5 V3	T6 V2	T7 V3	T7 V1	T68 V0	T68 V1	T69 V0	T69 V1	T70 V0	T70 V1	T71 V0	T71 V1
3	T4 V1	T5 V1	T6 V1	T7 V1	T8 V0	T9 V0	T10 V0	T11 V0	T11 V1	T72 V0	T72 V1	T73 V0	T73 V1	T74 V0	T74 V1	T75 V0	T75 V1
4	T8 V0	T9 V0	T10 V0	T11 V0	T8 V1	T9 V1	T10 V1	T11 V1	T11 V2	T72 V1	T72 V2	T73 V1	T73 V2	T74 V0	T74 V1	T75 V0	T75 V1
5	T8 V1	T9 V1	T10 V1	T11 V1	T8 V2	T9 V2	T10 V2	T11 V2	T11 V3	T72 V2	T72 V3	T73 V2	T73 V3	T74 V1	T74 V2	T75 V1	T75 V2
6	T12 V0	T13 V0	T14 V0	T15 V0	T12 V1	T13 V1	T14 V1	T15 V1	T15 V0	T76 V0	T76 V1	T77 V0	T77 V1	T78 V0	T78 V1	T79 V0	T79 V1
7	T12 V1	T13 V1	T14 V1	T15 V1	T12 V2	T13 V2	T14 V2	T15 V2	T15 V1	T76 V2	T76 V3	T77 V2	T77 V3	T78 V2	T78 V3	T79 V2	T79 V3
8	T16 V0	T17 V0	T18 V0	T19 V0	T16 V1	T17 V1	T18 V1	T19 V1	T19 V0	T80 V0	T80 V1	T81 V0	T81 V1	T82 V0	T82 V1	T83 V0	T83 V1
9	T16 V1	T17 V1	T18 V1	T19 V1	T16 V2	T17 V3	T18 V2	T19 V3	T19 V2	T80 V2	T80 V3	T81 V2	T81 V3	T82 V1	T82 V2	T83 V1	T83 V2
10	T20 V0	T21 V0	T22 V0	T23 V0	T20 V1	T21 V1	T22 V1	T23 V1	T23 V0	T84 V0	T84 V1	T85 V0	T85 V1	T86 V0	T86 V1	T87 V0	T87 V1
11	T20 V1	T21 V1	T22 V1	T23 V1	T20 V2	T21 V3	T22 V2	T23 V3	T23 V2	T84 V2	T84 V3	T85 V2	T85 V3	T86 V2	T86 V3	T87 V2	T87 V3
12	T24 V0	T25 V0	T26 V0	T27 V0	T24 V1	T25 V1	T26 V1	T27 V1	T27 V0	T88 V0	T88 V1	T89 V0	T89 V1	T90 V0	T90 V1	T91 V0	T91 V1
13	T24 V1	T25 V1	T26 V1	T27 V1	T24 V2	T25 V2	T26 V2	T27 V2	T27 V1	T88 V1	T88 V2	T89 V1	T89 V2	T90 V0	T90 V1	T91 V0	T91 V1
14	T28 V0	T29 V0	T30 V0	T31 V0	T28 V1	T29 V1	T30 V1	T31 V1	T31 V0	T90 V0	T90 V1	T91 V0	T91 V1	T92 V0	T92 V1	T93 V0	T93 V1
15	T28 V1	T29 V1	T30 V1	T31 V1	T28 V2	T29 V3	T30 V2	T31 V3	T31 V2	T92 V0	T92 V1	T93 V0	T93 V1	T94 V0	T94 V1	T95 V0	T95 V1
16	T32 V0	T33 V0	T34 V0	T35 V0	T32 V1	T33 V1	T34 V1	T35 V1	T35 V0	T96 V0	T96 V1	T97 V0	T97 V1	T98 V0	T98 V1	T99 V0	T99 V1
17	T32 V1	T33 V1	T34 V1	T35 V1	T32 V2	T33 V3	T34 V2	T35 V3	T35 V2	T96 V3	T96 V4	T97 V3	T97 V4	T98 V3	T98 V4	T99 V3	T99 V4
18	T36 V0	T37 V0	T38 V0	T39 V0	T36 V1	T37 V1	T38 V1	T39 V1	T39 V0	T100 V0	T100 V1	T101 V0	T101 V1	T102 V0	T102 V1	T103 V0	T103 V1
19	T36 V1	T37 V1	T38 V1	T39 V1	T36 V2	T37 V2	T38 V2	T39 V2	T39 V1	T100 V1	T100 V2	T101 V1	T101 V2	T102 V1	T102 V2	T103 V1	T103 V2
20	T40 V0	T41 V0	T42 V0	T43 V0	T40 V1	T41 V1	T42 V1	T43 V1	T43 V0	T104 V0	T104 V1	T105 V0	T105 V1	T106 V0	T106 V1	T107 V0	T107 V1
21	T40 V1	T41 V1	T42 V1	T43 V1	T40 V2	T41 V3	T42 V2	T43 V3	T43 V2	T104 V1	T104 V2	T105 V1	T105 V2	T106 V1	T106 V2	T107 V1	T107 V2
22	T44 V0	T45 V0	T46 V0	T47 V0	T44 V1	T45 V1	T46 V1	T47 V1	T47 V0	T108 V0	T108 V1	T109 V0	T109 V1	T110 V0	T110 V1	T111 V0	T111 V1
23	T44 V1	T45 V1	T46 V1	T47 V1	T44 V2	T45 V3	T46 V2	T47 V3	T47 V2	T108 V1	T108 V2	T109 V1	T109 V2	T110 V0	T110 V1	T111 V0	T111 V1
24	T48 V0	T49 V0	T50 V0	T51 V0	T48 V1	T49 V1	T50 V1	T51 V1	T51 V0	T112 V0	T112 V1	T113 V0	T113 V1	T114 V0	T114 V1	T115 V0	T115 V1
25	T48 V1	T49 V1	T50 V1	T51 V1	T48 V2	T49 V2	T50 V2	T51 V2	T51 V1	T112 V1	T112 V2	T113 V1	T113 V2	T114 V0	T114 V1	T115 V0	T115 V1
26	T52 V0	T53 V0	T54 V0	T55 V0	T52 V1	T53 V1	T54 V1	T55 V1	T55 V0	T116 V0	T116 V1	T117 V0	T117 V1	T118 V0	T118 V1	T119 V0	T119 V1
27	T52 V1	T53 V1	T54 V1	T55 V1	T52 V2	T53 V3	T54 V2	T55 V3	T55 V2	T116 V1	T116 V2	T117 V1	T117 V2	T118 V0	T118 V1	T119 V0	T119 V1
28	T56 V0	T57 V0	T58 V0	T59 V0	T56 V1	T57 V1	T58 V1	T59 V1	T59 V0	T120 V0	T120 V1	T121 V0	T121 V1	T122 V0	T122 V1	T123 V0	T123 V1
29	T56 V1	T57 V1	T58 V1	T59 V1	T56 V2	T57 V3	T58 V2	T59 V3	T59 V2	T120 V1	T120 V2	T121 V1	T121				

CUTE and Copy

Layouts of COPYs

```

Copy_Atom copy_atom = Copy_Atom<SM75_U32x4_LDSM_N>{};
TiledCopy tiled_copy = make_tiled_copy_A(copy_atom, tiled_mma);

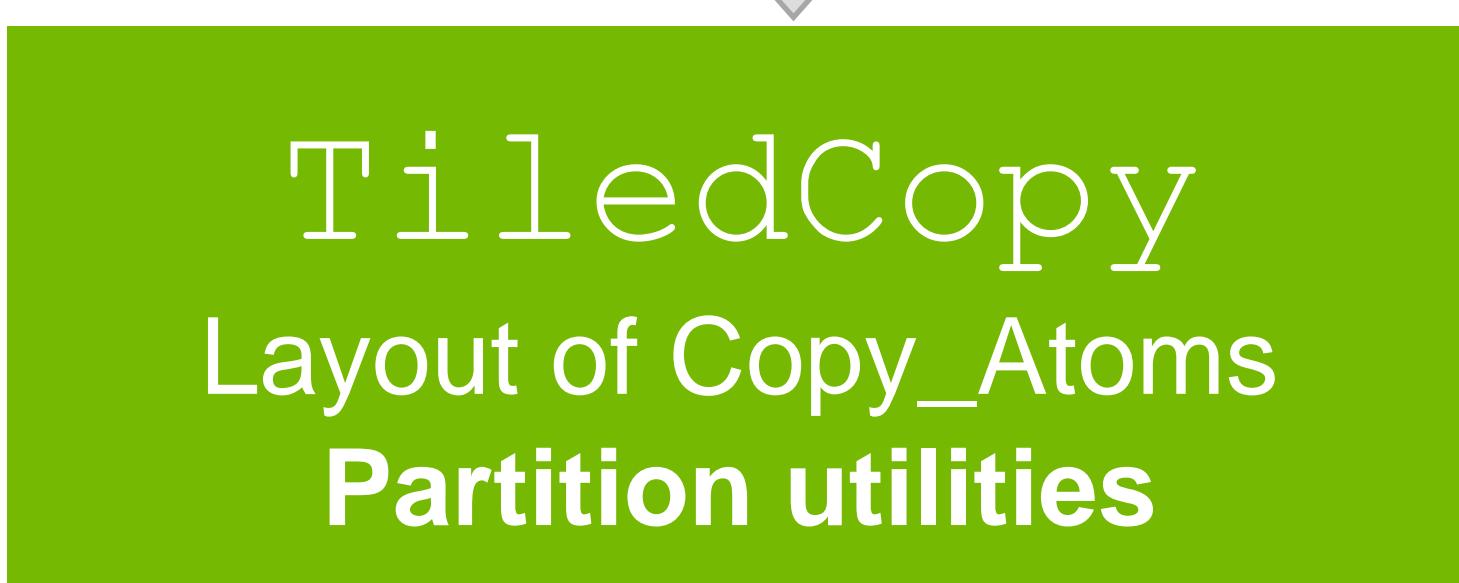
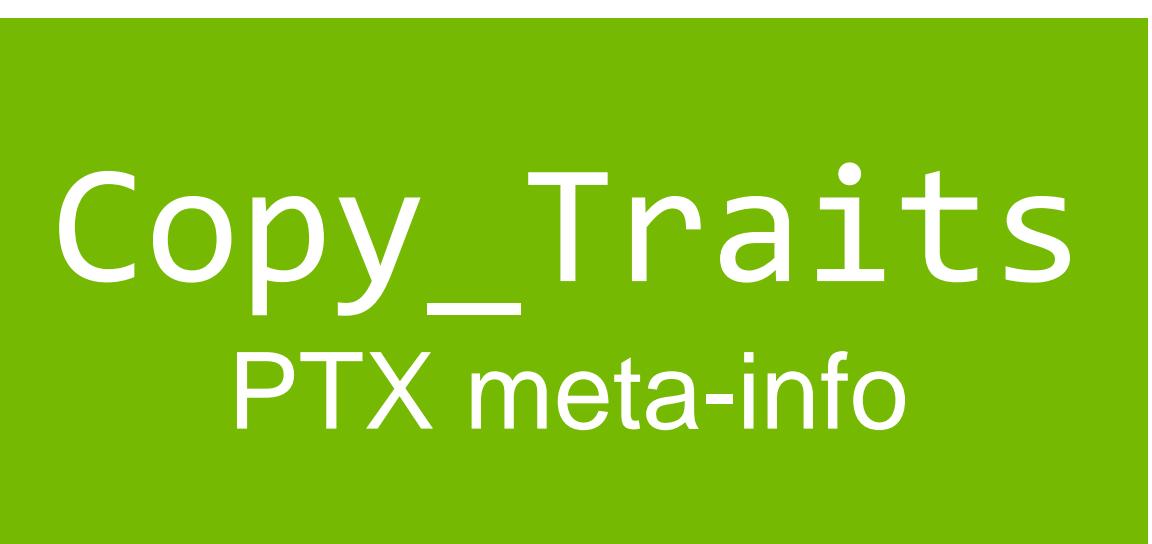
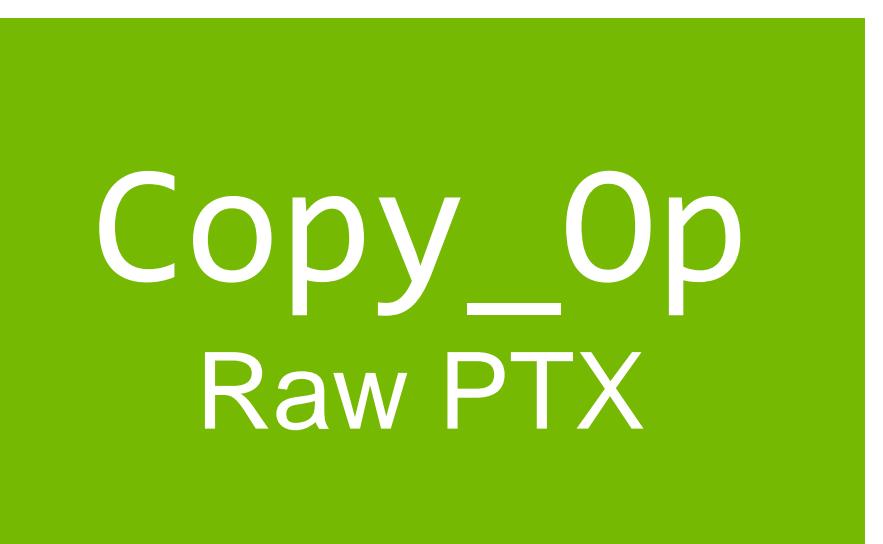
ThrCopy thr_copy = tiled_copy.get_slice(thread_idx);
Tensor thr_A = thr_copy.partition_S(A);
Tensor cpy_rA = thr_copy.retile_D(rA);

// Perform a (V,M) -> (V,M) copy via LDSM
copy(copy_atom, thr_A(_,_), cpy_rA(_,_));

```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0 V0	T1 V0	T2 V0	T3 V0	T4 V0	T5 V0	T6 V0	T7 V0	T16 V0	T17 V0	T18 V0	T19 V0	T20 V0	T21 V0	T22 V0	T23 V0
1	T0 V1	T1 V1	T2 V1	T3 V1	T4 V1	T5 V1	T6 V1	T7 V1	T16 V1	T17 V1	T18 V1	T19 V1	T20 V1	T21 V1	T22 V1	T23 V1
2	T0 V2	T1 V2	T2 V2	T3 V2	T4 V2	T5 V2	T6 V2	T7 V2	T16 V2	T17 V2	T18 V2	T19 V2	T20 V2	T21 V2	T22 V2	T23 V2
3	T0 V3	T1 V3	T2 V3	T3 V3	T4 V3	T5 V3	T6 V3	T7 V3	T16 V3	T17 V3	T18 V3	T19 V3	T20 V3	T21 V3	T22 V3	T23 V3
4	T0 V4	T1 V4	T2 V4	T3 V4	T4 V4	T5 V4	T6 V4	T7 V4	T16 V4	T17 V4	T18 V4	T19 V4	T20 V4	T21 V4	T22 V4	T23 V4
5	T0 V5	T1 V5	T2 V5	T3 V5	T4 V5	T5 V5	T6 V5	T7 V5	T16 V5	T17 V5	T18 V5	T19 V5	T20 V5	T21 V5	T22 V5	T23 V5
6	T0 V6	T1 V6	T2 V6	T3 V6	T4 V6	T5 V6	T6 V6	T7 V6	T16 V6	T17 V6	T18 V6	T19 V6	T20 V6	T21 V6	T22 V6	T23 V6
7	T0 V7	T1 V7	T2 V7	T3 V7	T4 V7	T5 V7	T6 V7	T7 V7	T16 V7	T17 V7	T18 V7	T19 V7	T20 V7	T21 V7	T22 V7	T23 V7
8	T8 V0	T9 V0	T10 V0	T11 V0	T12 V0	T13 V0	T14 V0	T15 V0	T24 V0	T25 V0	T26 V0	T27 V0	T28 V0	T29 V0	T30 V0	T31 V0
9	T8 V1	T9 V1	T10 V1	T11 V1	T12 V1	T13 V1	T14 V1	T15 V1	T24 V1	T25 V1	T26 V1	T27 V1	T28 V1	T29 V1	T30 V1	T31 V1
10	T8 V2	T9 V2	T10 V2	T11 V2	T12 V2	T13 V2	T14 V2	T15 V2	T24 V2	T25 V2	T26 V2	T27 V2	T28 V2	T29 V2	T30 V2	T31 V2
11	T8 V3	T9 V3	T10 V3	T11 V3	T12 V3	T13 V3	T14 V3	T15 V3	T24 V3	T25 V3	T26 V3	T27 V3	T28 V3	T29 V3	T30 V3	T31 V3
12	T8 V4	T9 V4	T10 V4	T11 V4	T12 V4	T13 V4	T14 V4	T15 V4	T24 V4	T25 V4	T26 V4	T27 V4	T28 V4	T29 V4	T30 V4	T31 V4
13	T8 V5	T9 V5	T10 V5	T11 V5	T12 V5	T13 V5	T14 V5	T15 V5	T24 V5	T25 V5	T26 V5	T27 V5	T28 V5	T29 V5	T30 V5	T31 V5
14	T8 V6	T9 V6	T10 V6	T11 V6	T12 V6	T13 V6	T14 V6	T15 V6	T24 V6	T25 V6	T26 V6	T27 V6	T28 V6	T29 V6	T30 V6	T31 V6
15	T8 V7	T9 V7	T10 V7	T11 V7	T12 V7	T13 V7	T14 V7	T15 V7	T24 V7	T25 V7	T26 V7	T27 V7	T28 V7	T29 V7	T30 V7	T31 V7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0 V0	T1 V1	T2 V0	T3 V1	T4 V0	T5 V1	T6 V0	T7 V1	T16 V0	T17 V1	T18 V0	T19 V1	T20 V0	T21 V1	T22 V0	T23 V1
1	T4 V0	T4 V1	T5 V0	T5 V1	T6 V0	T6 V1	T7 V0	T7 V1	T4 V4	T4 V5	T5 V4	T5 V5	T6 V4	T6 V5	T7 V4	T7 V5
2	T8 V0	T8 V1	T9 V0	T9 V1	T10 V0	T10 V1	T11 V0	T11 V1	T8 V4	T8 V5	T9 V4	T9 V5	T10 V4	T10 V5	T11 V4	T11 V5
3	T12 V0	T12 V1	T13 V0	T13 V1	T14 V0	T14 V1	T15 V0	T15 V1	T12 V4	T12 V5	T13 V4	T13 V5	T14 V4	T14 V5	T15 V4	T15 V5
4	T16 V0	T16 V1	T17 V0	T17 V1	T18 V0	T18 V1	T19 V0	T19 V1	T16 V4	T16 V5	T17 V4	T17 V5	T18 V4	T18 V5	T19 V4	T19 V5
5	T20 V0	T20 V1	T21 V0	T21 V1	T22 V0	T22 V1	T23 V0	T23 V1	T20 V4	T20 V5	T21 V4	T21 V5	T22 V4	T22 V5	T23 V4	T23 V5
6	T24 V0	T24 V1	T25 V0	T25 V1	T26 V0	T26 V1	T27 V0	T27 V1	T24 V4	T24 V5	T25 V4	T25 V5	T26 V4	T26 V5	T27 V4	T27 V5
7	T28 V0	T28 V1	T29 V0	T29 V1	T30 V0	T30 V1	T31 V0	T31 V1	T28 V4	T28 V5	T29 V4	T29 V5	T30 V4	T30 V5	T31 V4	T31 V5
8	T0 V2	T0 V3	T1 V2	T1 V3	T2 V2	T2 V3	T3 V2	T3 V3	T0 V6	T0 V7	T1 V6	T1 V7	T2 V6	T2 V7	T3 V6	T3 V7
9	T4 V2	T4 V3	T5 V2	T5 V3	T6 V2	T6 V3	T7 V2	T7 V3	T4 V6	T4 V7	T5 V6	T5 V7	T6 V6	T6 V7	T7 V6	T7 V7
10	T8 V2	T8 V3	T9 V2	T9 V3	T10 V2	T10 V3	T11 V2	T11 V3	T8 V6	T8 V7	T9 V6	T9 V7	T10 V6	T10 V7	T11 V6	T11 V7
11	T12 V2	T12 V3	T13 V2	T13 V3	T14 V2	T14 V3	T15 V2	T15 V3	T12 V6	T12 V7	T13 V6	T13 V7	T14 V6	T14 V7	T15 V6	T15 V7
12	T16 V2	T16 V3	T17 V2	T17 V3	T18 V2	T18 V3	T19 V2	T19 V3	T16 V6	T16 V7	T17 V6	T17 V7	T18 V6	T18 V7	T19 V6	T19 V7
13	T20 V2	T20 V3	T21 V2	T21 V3	T22 V2	T22 V3	T23 V2	T23 V3	T20 V6	T20 V7	T21 V6	T21 V7	T22 V6	T22 V7	T23 V6	T23 V7
14	T24 V2	T24 V3	T25 V2	T25 V3	T26 V2	T26 V3	T27 V2	T27 V3	T24 V6	T24 V7	T25 V6	T25 V7	T26 V6	T26 V7	T27 V6	T27 V7
15	T28 V2	T28 V3	T29 V2	T29 V3	T30 V2	T30 V3	T31 V2	T31 V3	T28 V6	T28 V7	T29 V6	T29 V7	T30 V6	T30 V7	T31 V6	T31 V7





CUTLASS 3.0 Preview

- CuTe Layout Functions
- Tensor Core Programming Model
- **CUTLASS 3.0 API**

CUTLASS 3.0: Device, Kernel, Mainloop API

CUDA C++ Template Library for Deep Learning and High Performance Computing

CUTLASS 3.0 Computation hierarchy:

Device	Universal, kernel agnostic host interface for parameter construction, and kernel launch.
Kernel	Entry point for a grid of CTAs that may or may not be organized in a cluster. Composition point for fusing back-to-back GEMMs, epilogues, ...
Collective	Number of threads that cooperate. Pipelined Matrix Multiply, epilogues, accelerated synchronization, thread block clusters Composition point for mainloop fusion, epilogue bias fusion etc ...
TiledMMA / TiledCopy	Layout of copy or math atoms across the collective.
Atom	Smallest operation (math/copy) issued and managed by one or more threads FFMA, LDS, mma.sync, LDSM, 3xTF32, Complex MMAs ... Composed PTX instructions and PTX meta-information.

- Customize any layer in the hierarchy without while preserving composability with other layers.
 - Extreme focus on developer productivity.
- Static checking at every layer to ensure layout compatibilities
 - If it compiles, it will be correct – actionable static assert messages otherwise
- Optimization through careful layout construction
 - CUTLASS 3.0 will provide default layouts for common use cases

CUTLASS 3.0: 2.x Device Compatibility API

`cutlass::gemm::device::Gemm<>`

- Backwards compatible template interface with CUTLASS 2.x
- Default GEMM configurations adapt basic GEMM kernel configuration to 3.0 API
 - Also provide examples of 3.0 kernel recipes
- More familiar OOTB kernel usability for simple use cases
 - Only data types and layouts respected
 - Instruction shape, warp shape, tile shape, operator class, deprecated and unused
- CUTLASS 3.0 also supports a bespoke GemmUniversal interface:

```
template <typename GemmKernel_>
class GemmUniversal {
public:
    using GemmKernel = GemmKernel_;
    using KernelPolicy = typename GemmKernel::KernelPolicy;
    . . .
};
```

`DefaultGemmConfiguration<>` provides aliases for 3.0 API recipes

```
template<
    class OperatorClass, class ArchTag,
    class ElementA, class LayoutA,
    class ElementB, class LayoutB,
    class ElementC, class LayoutC,
    class ElementAccumulator>
struct DefaultGemmConfiguration;
```

Compatibility `cutlass::gemm::device::Gemm<>`
Assembles 3.0 Kernels with this recipe while preserving 2.x Device API compatibility

```
// Convert cutlass::Layout::XXX to cute::Stride
using StrideA = typename detail::TagToStrideA<LayoutA>::type;
using StrideB = typename detail::TagToStrideB<LayoutB>::type;
using StrideC = typename detail::TagToStrideC<LayoutC>::type;

// Tag dispatch policy for kernel and mainloops
using KernelPolicy = typename Config::KernelPolicy;

// Grabs the right atom from the configuration
using TiledMma = typename Config::TiledMma;

// Tiled Copy for gmem->smem copy
using GmemTiledCopyA = typename Config::GmemTiledCopyA;
using GmemTiledCopyB = typename Config::GmemTiledCopyB;

// Shared memory Layouts of A and B, including the pipeline mode (if any)
using SmemLayoutA = typename Config::SmemLayoutA;
using SmemLayoutB = typename Config::SmemLayoutB;

// Tiled Copy for smem->rmem copy
using SmemCopyAtomA = typename Config::SmemCopyAtomA;
using SmemCopyAtomB = typename Config::SmemCopyAtomB;
```

CUTLASS 3.0: Mainloop API

`cutlass::gemm::collective::Gemm<>`
`cutlass::epilogue::collective::Epilgoue<>`

- **Collective:**

Largest number of threads in a grid that can leverage hardware features for accelerated communication and synchronization

- CTA/CGA wide operand and work ownership
- Mainloop constructs like stream K, mainloop fusion etc ...
- Selection via dispatch policies
 - Easy to write custom mainloops and compose with kernel layer

```
// 2 stage pipeline through 1 stage in smem, 1 in rmem
template<
    bool Predicated_
>
struct MainloopSm70TwoStage {
    constexpr static int Stages = 2;
    constexpr static bool Predicated = Predicated_;
};

// n-buffer pipeline in smem, accessed async with LDGSTS
template<
    int Stages_,
    bool Predicated_
>
struct MainloopSm80Ldgsts {
    constexpr static int Stages = Stages_;
    constexpr static bool Predicated = Predicated_;
};
```

```
template <
    class DispatchPolicy,
    class TileShape,
    class ElementA,
    class SmemLayoutA,
    class ElementB,
    class SmemLayoutB,
    class ElementC,
    class ArchTag,
    class TiledMma,
    class GmemCopyAtomA,
    class SmemCopyAtomA,
    class GmemCopyAtomB,
    class SmemCopyAtomB
>
struct CollectiveMma;

template <
    class ThreadEpilogueOp_,
    class SmemLayout_,
    class CopyAtomR2S_,
    class TiledCopyS2R_,
    class CopyAtomR2G_
>
class CollectiveEpilogue;
```

Collective Mainloop Setup

Ampere Tensor Cores + LDGSTS + LDSM mainloop

```

Tensor sA = make_tensor(make_smem_ptr(shared_storage.smem_a.data()), SmemLayoutA{}); // (BLK_M,BLK_K,PIPE)

// Partition the copying of A and B tiles across the threads
GmemTiledCopyA gmem_tiled_copy_A;
auto gmem_thr_copy_A = gmem_tiled_copy_A.get_slice(thread_idx);
Tensor tAgA = gmem_thr_copy_A.partition_S(gA); // (ACPY,ACPY_M,ACPY_K,k)
Tensor tAsA = gmem_thr_copy_A.partition_D(sA); // (ACPY,ACPY_M,ACPY_K,PIPE)

// Tile MMA compute thread partitions and allocate accumulators
TiledMma tiled_mma;
auto thr_mma = tiled_mma.get_thread_slice(thread_idx);
Tensor tCrA = thr_mma.partition_fragment_A(sA(_,_,_0)); // (MMA,MMA_M,MMA_K)

// Retile smem->rmem copies with LDSM TiledCopy
auto smem_tiled_copy_A = make_tiled_copy_A(SmemCopyAtomA{}, tiled_mma);
auto smem_thr_copy_A = smem_tiled_copy_A.get_thread_slice(thread_idx);
Tensor tCsA = smem_thr_copy_A.partition_S(sA); // (CPY,CPY_M,CPY_K,PIPE)
Tensor tCrA_copy_view = smem_thr_copy_A.retile_D(tCrA); // (CPY,CPY_M,CPY_K)
CUTE_STATIC_ASSERT_V(size<1>(tCsA) == size<1>(tCrA_copy_view));
CUTE_STATIC_ASSERT_V(size<2>(tCsA) == size<2>(tCrA_copy_view));

// Copy of the k-tile iterator in gmem that runs ahead for cp.async
auto k_tile_iter_read = k_tile_iter;

// Start async Loads for all pipes but the last
CUTLASS_PRAGMA_UNROLL
for (int k_pipe = 0; k_pipe < DispatchPolicy::Stages-1; ++k_pipe) {
    int k_tile_read = (k_tile_iter_read < k_tile_iter_end) ? *k_tile_iter_read : 0;
    copy(gmem_tiled_copy_A, tAgA(_,_,_0,k_tile_read), tAsA(_,_,_0,k_pipe));
    copy(gmem_tiled_copy_B, tBgB(_,_,_0,k_tile_read), tBsB(_,_,_0,k_pipe));
    copy_ldgsts_fence();
    ++k_tile_iter_read;
}

// PREFETCH register pipeline
if (K_BLOCK_MAX > size<2>(tCrA)) {
    // Wait until our first prefetched tile is loaded in
    copy_ldgsts_wait<DispatchPolicy::Stages-2>();
    __syncthreads();

    // Prefetch the first rmem from the first k-tile
    copy(smem_tiled_copy_A, tCsA_p(_,_0,Int<0>{}), tCrA_copy_view(_,_0,Int<0>{}));
    copy(smem_tiled_copy_B, tCsB_p(_,_0,Int<0>{}), tCrB_copy_view(_,_0,Int<0>{}));
}

```

- Partition gmem tensor with **GmemTiledCopy**
- LDGSTS global memory atom with vectorization information
- Partition tiled MMA and allocate thread fragments
- Partition smem tensor with **SmemTiledCopy**
 - LDSM copy atom with source and destination layout static check
 - Retile destination MMA rmem tensor
- Issue prologue gmem reads
- Prefetch 0th K_BLOCK registers

Collective Mainloop Pipeline

Ampere Tensor Cores + LDGSTS + LDSM mainloop

```
CUTLASS_PRAGMA_NO_UNROLL
for ( ; k_tile_iter < k_tile_iter_end; ++k_tile_iter)
{
    // Pipeline the outer products with a static for loop
    for_each(make_int_sequence<K_BLOCK_MAX>{}, [&] (auto k_block)
    {
        if (k_block == K_BLOCK_MAX - 1) {
            tCsA_p = tCsA(_,_,_ ,k_pipe_read);
            tCsB_p = tCsB(_,_,_ ,k_pipe_read);

            // Commit the smem for k_pipe_read
            copy_ldgsts_wait<DispatchPolicy::Stages-2>();
            __syncthreads();
        }

        // Load A, B smem->rmem for k_block+1
        auto k_block_next = (k_block + Int<1>{}) % K_BLOCK_MAX; // static
        copy(smem_tiled_copy_A, tCsA_p(_,_,_ ,k_block_next), tCrA_copy_view(_,_,_ ,k_block_next));
        copy(smem_tiled_copy_B, tCsB_p(_,_,_ ,k_block_next), tCrB_copy_view(_,_,_ ,k_block_next));

        // Copy gmem to smem before computing gemm on each k-pipe
        if (k_block == 0) {
            int k_tile_read = (k_tile_iter_read < k_tile_iter_end) ? *k_tile_iter_read : 0;
            copy(gmem_tiled_copy_A, tAgA(_,_,_ ,k_tile_read), tAsA(_,_,_ ,k_pipe_write));
            copy(gmem_tiled_copy_B, tBgB(_,_,_ ,k_tile_read), tBsB(_,_,_ ,k_pipe_write));
            copy_ldgsts_fence();

            ++k_tile_iter_read;

            // Advance the pipe
            k_pipe_write = k_pipe_read;
            ++k_pipe_read;
            k_pipe_read = (k_pipe_read == DispatchPolicy::Stages) ? 0 : k_pipe_read;
        }

        // Transform before compute
        cute::transform(tCrA(_,_,_ ,k_block), transform_A);
        cute::transform(tCrB(_,_,_ ,k_block), transform_B);

        // Thread-Level register gemm for k_block
        cute::gemm(tiled_mma, accum, tCrA(_,_,_ ,k_block), tCrB(_,_,_ ,k_block), src_accum);
    });
}
```

- Pipelined mainloop body
 - N stages in smem fetched async with LDGSTS
 - Vectorized, bank conflict free smem loads with LDSM
- No iterators, only `cute::Tensor`
 - Except for `k_tiles` to enable GETT/streamK style iteration strategies
- Compare with [gemm/threadblock/mma_multistage.h](#)



Conclusion

- Roadmap
- CUTLASS Python
- CUTLASS 2.x Enhancements
- CUTLASS 3.0 Preview

Conclusion

CUTLASS

- **CUTLASS Roadmap**

- CUTLASS 2.10 was released August 2022
- CUTLASS 2.11 available October 2022
- CUTLASS 3.0 Preview to become available November 2022

- **CUTLASS Python**

- Defines a Python API and programming model for constructing CUTLASS kernels
- Interoperable with Numpy

- **CUTLASS 2.x Enhancements**

- Fused Softmax, Layernorm, and Multihead Attention kernels offer significant speedups
- Grouped and Depthwise Separable Convolution reduces time complexity of certain convolution workloads
- Stream-K dynamic scheduling algorithm achieves load balance and reduces kernel selection burden
- NVIDIA Hopper Architecture enablement accelerates FP64 and supports FP8

- **CUTLASS 3.0 Preview**

- New programming model for Tensor Cores
- Based on CuTe layout algebra and metaprogramming abstractions
- Implements optimal computations using 4th generation Tensor Cores in H100
- To be released November 2022 to coincide with CUDA 12.0

