CUTLASS: A Performant, Flexible, and Portable Way to Target Hopper Tensor Cores

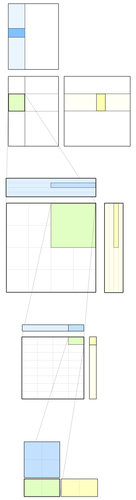
Vijay Thakkar, Jack Kosaian
NVIDIA GTC 2024 | 2024/03/19

CUTLASS

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

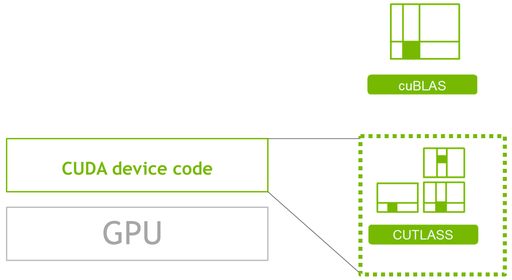
CUTLASS: tensor computations at all scopes and scales, decomposed into their “moving parts”

Device	{ GEMM, Convolution, Reductions , BLAS3 } x { all data types } x { SIMT, Tensor Cores } x { all architectures }
Kernel	GEMM, Batched GEMM, Convolution, Reduction, Fused output operations, Fused input operations
Collective	CUTLASS **temporal micro-kernels** (async producer/consumer pipelines orchestrating spatial micro-kernels)
Atom	CuTe **spatial micro-kernels** (Tiled MMA / Copy)
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)



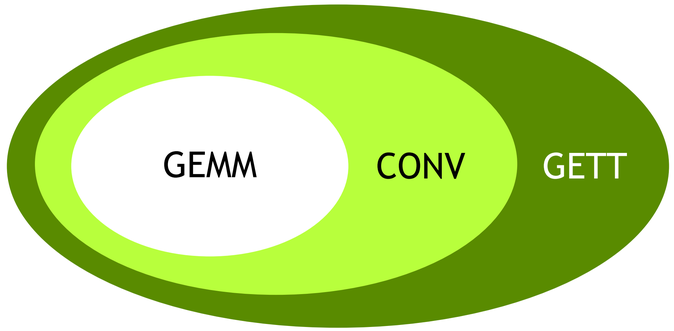
Open source: https://github.com/NVIDIA/cutlass

- 4.4K stars, 2.5M clones/month, 100+ contributors, and many active users
- Latest revision: CUTLASS 3.5
- Documentation: https://github.com/NVIDIA/cutlass#documentation
- Presented: GTC’18, GTC’19, GTC’20, GTC’21, GTC'22 , GTC’22, GTC’23



What's new since GTC'23?

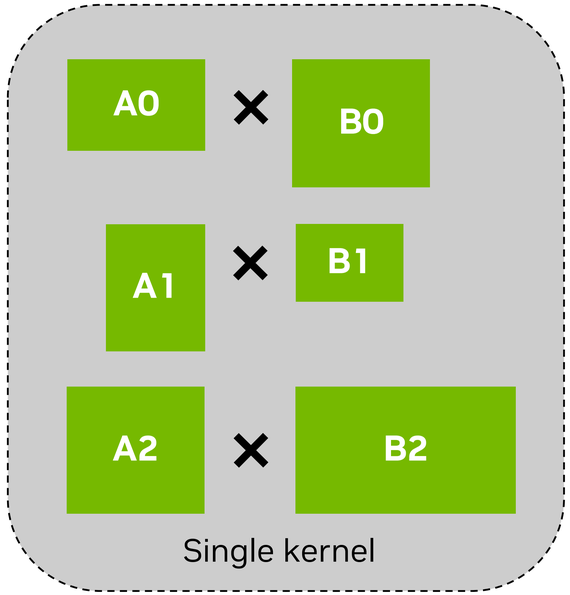
Convolutions in CUTLASS 3



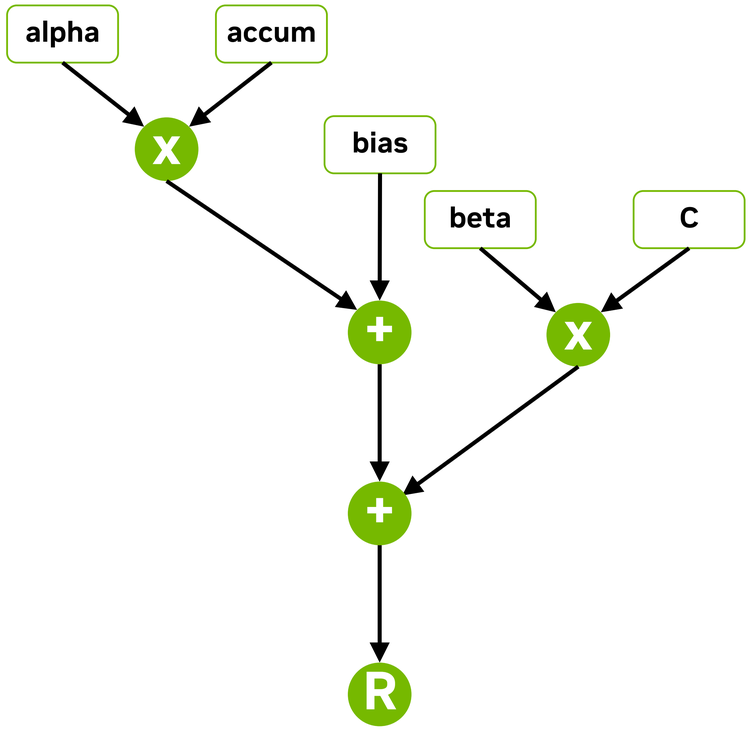
Features for LLMs

Mixed-input GEMM

Grouped GEMM

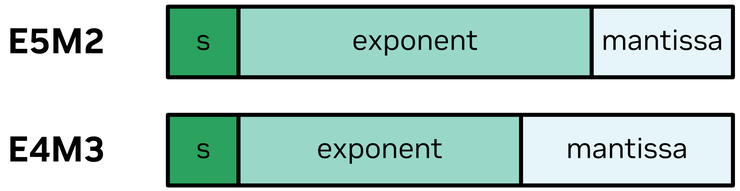


Epilogue Fusions via Visitor Tree



What *else* is new since GTC'23?

FP8 GEMMs and convolutions for Hopper and Ada



Tile schedulers for composable load balancing

```
using GemmKernel = cutlass::gemm::kernel::GemmUniversal<
  ProblemShape,
  CollectivMainloop,
  CollectiveEpilogue,
  cutlass::gemm::StreamKScheduler
>;
```

Improved Python interface support and addition of PyPI wheel

```
pip install nvidia-cutlass
```

More on GitHub

- Expanded compiler support
- Narrow-alignment GEMMs
- Improved documentation

Agenda

- Convolutions in CUTLASS 3
- Features for LLMs
- Epilogue Visitor Tree
- Conclusion

CUTLASS 3 Conceptual Hierarchy

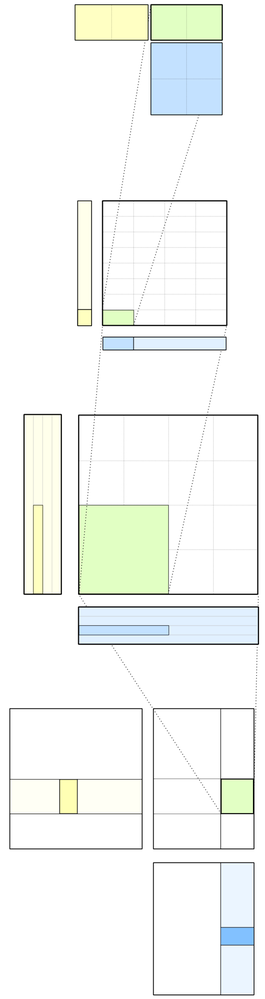
GEMMs and Convolutions

- **Atom layer:** Architecture instructions and associated meta-information
  - Smallest set of threads and values that must participate in an architecture accelerated specified math/copy op
- **Tiled MMA/Copy**: Spatial Microkernel layer
  - **Describes the complete spatial tiling of a math/copy operation**

CuTe

CUTLASS

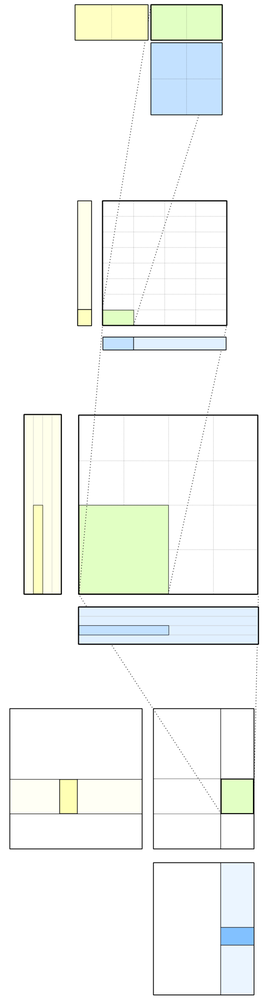
- **Collective layer:** Temporal Microkernel layer
  - **Describes the complete temporal tiling of a math/copy operation computing one output tile**
  - Mainloops that orchestrate the copy/math micro-kernels with arch specific synchronization
- **Kernel layer**: Outermost loops around collectives
  - Conceptually: A collection of all threadblock/clusters in the grid
  - Responsible for load balancing across tiles, thread marshalling, grid planning, and arguments construction
- **Device layer:** host side setup and interface



CUTLASS 3 API Entry Points

Single points of entry at each abstraction layer

- Spatial microkernels: `cute::Tiled{Mma|Copy}<>`
  - Robust representation power across a wide range GPU architectures
- Temporal Microkernels: `collective::Collective{Mma|Conv|Epilogue}<>`
  - Dispatched against by policies that also define the set of kernel schedules they can be composed with
- Kernel layer: `kernel::{Gemm|Conv}Universal<>`
  - Treats GEMM as a composition of a collective mainloop and a collective epilogue
  - Each new kernel schedule is a specialization dispatched against with schedule tags
- Device layer: `device::{Gemm|Conv}UniversalAdapter<>`
  - Can be used with 2.x or 3.x API kernels
  - A stateless handle to a kernel type
- Static asserts everywhere to guard against invalid compositions or incorrect layouts



Convolution Collectives

Primary new addition to the 3.x API for convolution

- API similar to that of GEMM collectives
  - Dispatched to via mainloop policies
  - Computes a single output tile
  - Composes with kernel schedules via kernel policies
- Almost all conv specific changes are limited to the mainloop
  - Implements fprop/dgrad/wgrad
  - Maps to cute strides for tensors
  - Constructs TMA
- Epilogue collectives are composable with both GEMM and Conv
  - Compose with existing custom epilogues out of the box!

```
template <
  class DispatchPolicy,
  class TileShape,
  class ElementA,
  class ElementB,
  class TiledMma,
  class TileTraitsA,
  class TileTraitsB
>
struct CollectiveConv;
```

Convolutions provide a familiar builder API

cutlass::**conv**::collective::CollectiveBuilder<>

"I just want a Hopper Fprop collective for NWC layout tensors (1-D Fprop)":

```
using CollectiveOp = typename conv::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    conv::Operator::kFprop,
    half_t, layout::TensorNWC, 8,
    half_t, layout::TensorNWC, 8,
    float,
    Shape<_64,_128,Shape<_64>>, Shape<_2,_1,_1>,
    conv::collective::StageCountAuto,
    conv::collective::KernelScheduleAuto
  >::CollectiveOp;
```

"I want a Hopper Dgrad collective on NDHWC layout tensors (3-D Dgrad)":

```
using CollectiveOp = typename conv::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    conv::Operator::kDgrad,
    half_t, layout::TensorNDHWC, 8,
    half_t, layout::TensorNDHWC, 8,
    float,
    Shape<_64,_128,Shape<_64>>, Shape<_2,_1,_1>,
    conv::collective::StageCount<5>,
    conv::collective::KernelScheduleAuto
  >::CollectiveOp;
```

```
template <
  class ArchTag,
  class OpClass,
  conv::Operator,
  class ElementA,
  class GmemLayoutA,
  int AlignmentA,
  class ElementB,
  class GmemLayoutB,
  int AlignmentB,
  class ElementAccumulator,
  class TileShape_MNK,
  class ClusterShape_MNK,
  class StageCountType,
  class KernelScheduleType,
  class Enable = void
>
struct CollectiveBuilder;
```

Builders do the heavy lifting …

```
using CollectiveOp = typename conv::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    conv::Operator::kDgrad,
    half_t, layout::TensorNDHWC, 8,
    half_t, layout::TensorNDHWC, 8,
    float,
    Shape<_64,_128,Shape<_64>>, Shape<_2,_1,_1>,
    conv::collective::StageCountAuto,
    conv::collective::KernelScheduleAuto
  >::CollectiveOp;
```

Automatically maps down to the best mainloop config:
mainloop type, stage count, GMMA instruction, TMA instruction, smem layouts, kernel schedule …

```
cutlass::conv::collective::CollectiveConv<
    conv::MainloopSm90TmaGmmaWarpSpecializedImplicitGemm<
        conv::Operator::kDgrad, 8, 3, Shape<_2,_1,_1>,
        cutlass::conv::KernelImplicitTmaWarpSpecializedSm90, 1>,
        Shape<_64,_128,Shape<_64>>,
    cutlass::half_t,
    cutlass::half_t,
    TiledMMA<
        MMA_Atom<SM90_64x128x16_F32F16F16_SS<GMMA::Major::K, GMMA::Major::MN>>,
        Layout<Shape<_1,_1,_1>>>,
    cutlass::conv::collective::detail::Sm90ImplicitGemmTileTraits<
        SM90_TMA_LOAD_IM2COL,
        ComposedLayout<Swizzle<3, 4, 3>, smem_ptr_flag_bits<16>,
           Layout<Shape<_64,_64,_8>, Stride<_64,_1,_4096>>>,
        void>,
    cutlass::conv::collective::detail::Sm90ImplicitGemmTileTraits<
        SM90_TMA_LOAD_MULTICAST,
        ComposedLayout<Swizzle<3, 4, 3>, smem_ptr_flag_bits<16>,
           Layout<Shape<Shape<_64,_2>,_64,_8>, Stride<Stride<_1,_4096>,_64,_8192>>>,
        void>>
```

CUTLASS 3.5 Conv API

Familiar kernel and device layer APIs

```
// Build the epilogue type
using CollectiveEpilogue = typename epilogue::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    TileShapeMNK, ClusterShapeMNK,
    epilogue::collective::EpilogueTileAuto,
    ElementAcc, ElementCompute,
    half_t, layout::TensorKCS, 8,
    half_t, layout::TensorKCS, 8,
    epilogue::collective::EpilogueScheduleAuto
  >::CollectiveOp;

// Build the mainloop type
using CollectiveMainloop = typename conv::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    conv::Operator::kWgrad,
    ElementAcc, layout::TensorNWC, 8,
    ElementFlt, layout::TensorNWC, 8,
    ElementAcc,
    TileShapeMNK, ClusterShapeMNK,
    conv::collective::StageCountAutoCarveout<carveout>,
    conv::collective::KernelScheduleAuto
  >::CollectiveOp;

// Compose both at the kernel layer, problem shape type inferred by the mainloop
using ConvKernel = conv::kernel::ConvUniversal<
    CollectiveMainloop,
    CollectiveEpilogue
  >;

// Device layer handle to the kernel
using Conv = conv::device::ConvUniversalAdapter<ConvKernel>;
```

Rank Agnostic Conv Problem Shape

Rank agnostic user facing argument for all CUTLASS 3.x CONVs

- GEMM kernel allows users to provide arbitrary `cute::Shape` as problem shape type
  - Builder allows for arbitrary strides to compose GETT kernels
- Convolutions do not need such generality
  - Parametrization limited to spatial dimensions and algorithm
- Rank agnostic implementation requires not having to “name” modes
- Im2Col transform also an implementation detail
  - User provides asymmetric padding, dilations, and traversal strides
- Solution: a new problem shape type for N-dimensional convolution problems
  - Rank and implementation choice agnostic
  - Inferred by the conv mainloop based on the spatial dim and conv operation type

```
template <
  conv::Operator ConvOp_,
  int NumSpatialDimensions
>
struct ConvProblemShape;
```

```
ConvProblemShape{
  conv::Mode::kCrossCorrelation,
  {   1,    1,   8,  8, 64}, // ndhwc
  {8000, 8000, 800, 80,  1}, // stride (ndhwc)
  {  64,    1,   1,  1,  4}, // ktrsc
  {  64,   64,  64, 64,  1}, // stride (ktrsc)
  {8000, 8000, 800, 80,  1}, // stride (nzpqk)
  {          0,   0,  0    }, // padding lower (dhw)
  {          0,   0,  0    }, // padding upper (dhw)
  {          1,   1,  1    }, // t-strides (dhw)
  {          1,   1,  1    }, // dilation (dhw)
};
```

Mapping to kernel facing strides

How do we map conv layout tags to CuTe strides?

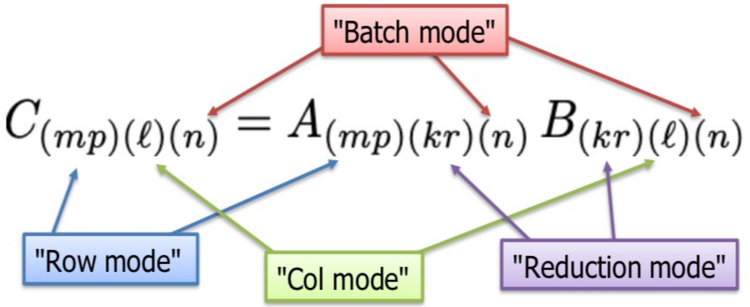
```
// CUTLASS 2.x GEMM kernel layout tags
using StridesA = cutlass::layout::RowMajor(ldA);

using StridesB = cutlass::layout::ColumnMajor(ldB);

Using StridesC = cutlass::layout::RowMajor(ldC);
```

```
// CUTLASS 2.x CONV kernel layout tags
using StridesAct = cutlass::layout::TensorNHWC(ldN, ldH, ldW);

using StridesFlt = cutlass::layout::TensorNHWC(ldN, ldH, ldW);

Using StridesOut = cutlass::layout::TensorNHWC(ldN, ldH, ldW);
```

Power Of Hierarchical Layouts

CUTLASS 3.x GEMMs are just tensor contractions (GETTs) in disguise

```
// Stride for A multi-modes, major mode in contraction dims
using RowModeStridesA = Stride<int64_t, int64_t, int64_t, int64_t>;
using RedModeStridesA = Stride< Int<1>, int64_t, int64_t>;
using BatModeStridesA = Stride<int64_t, int64_t, int64_t, int64_t>;


// Stride for B multi-modes, major mode in column dims
using ColModeStridesB = Stride< Int<1>, int64_t, int64_t, int64_t>;
using RedModeStridesB = Stride<int64_t, int64_t, int64_t>;
using BatModeStridesB = Stride<int64_t, int64_t, int64_t, int64_t>;


// Stride for C multi-modes
using RowModeStridesC = Stride<int64_t, int64_t, int64_t, int64_t>;
using ColModeStridesC = Stride<int64_t, int64_t, int64_t, int64_t>;
using BatModeStridesC = Stride<int64_t, int64_t, int64_t, int64_t>;
```



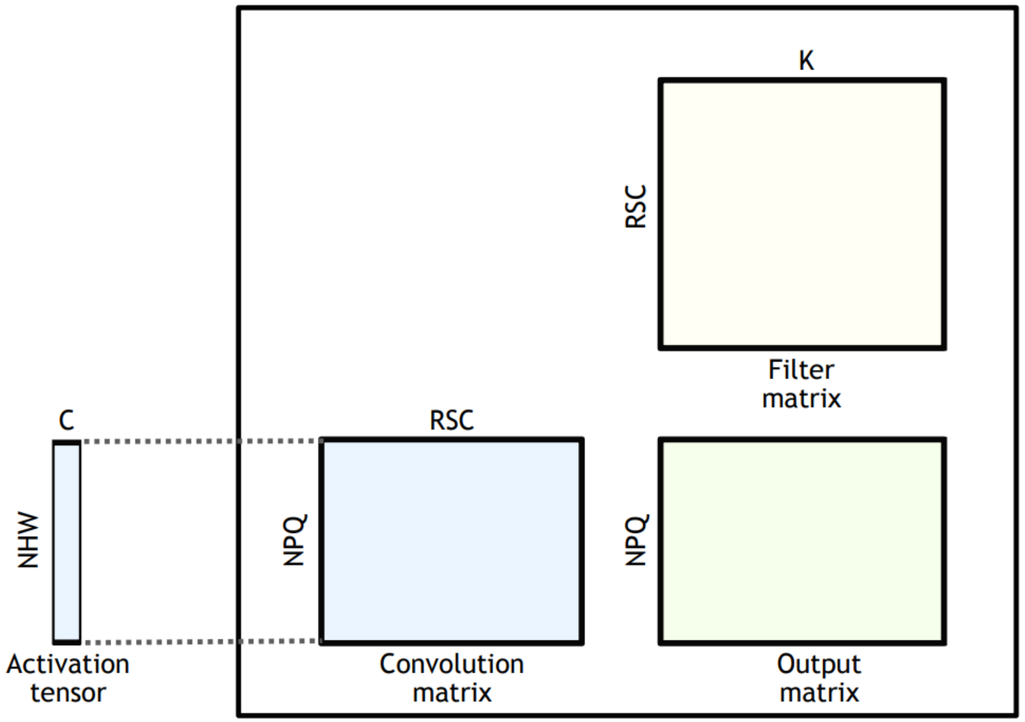

```
// Compose full tensor strides from components, following mode order convention
using StrideA = Stride<RowModeStridesA, RedModeStridesA, BatModeStridesA>;
using StrideB = Stride<ColModeStridesB, RedModeStridesB, BatModeStridesB>;
using StrideC = Stride<RowModeStridesC, ColModeStridesC, BatModeStridesC>;
```

GETTs Are All You Need

Convolutions are just special case of tensor contractions (GETT)

- Im2Col transforms the activation tensor shapes and strides
- Generalizes to any filter, dilation, traversal stride, padding
  - All factor into ZPQ shape
  - Filter shape becomes the activation TRS shape (expanding the domain)
  - Traversal strides factor into ZPQ strides
  - Dilations factor into TRS strides
- After transform
  - M multi-mode is logically consistent with the output M mode
  - K multi-mode is logically consistent with the filter K mode
- We can turn any CONV problem into a GETT!

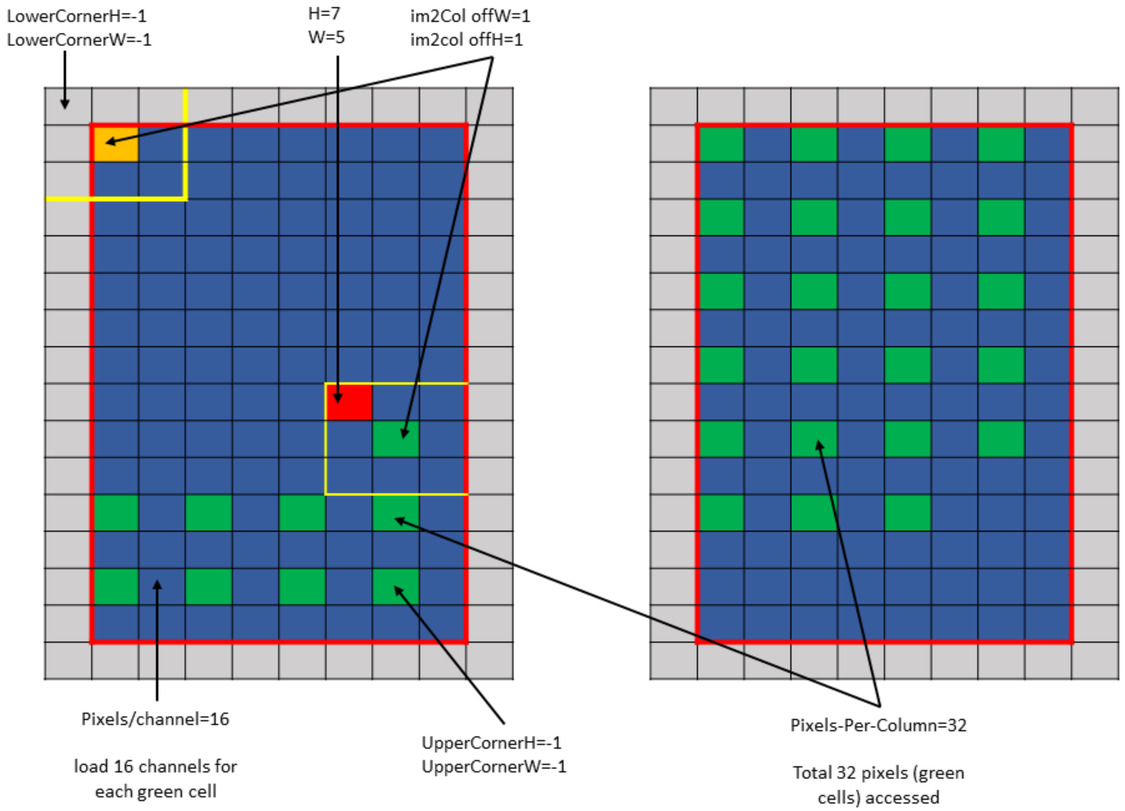
```
Conv inputs:
A tensor (Act): ((N,(D,H,W)), (C,(1,1,1)))
B tensor (Flt): ( K ,         (C,(T,R,S)))
C tensor (Out): ((N,(Z,P,Q)),  K)

Have Act: ((N,(D,H,W)), (C,(1,1,1)))
// Im2Col transform ->
Want Act: ((N,(Z,P,Q)), (C,(T,R,S)))
```



Hopper TMA Im2Col

Fast and simple data movement for convolutions

- SIMT kernels with Im2Col transform are difficult to implement and optimize
  - Replication of input activation tensor in the new expanded contraction mode
  - Accounting for halo loads and out of bounds (OOB) values for padding
  - Complex predication for OOB reads along the contraction mode
- Hopper TMA Im2Col makes this easy!
- Im2Col TMA performs
  - Im2Col transform on the tensor strides
  - Predicates OOB reads accounting for padding
  - Reduces issue overhead and register pressure

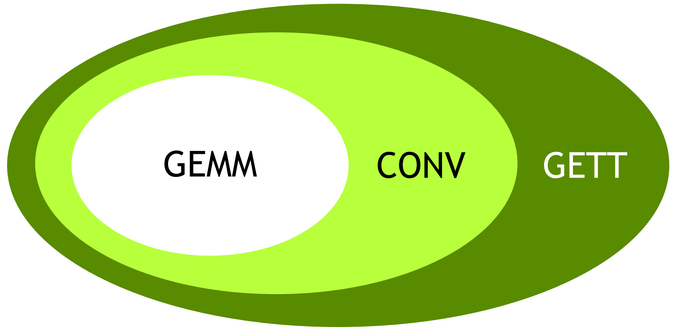


Example of using TMA im2col to load two images with non-trivial traversal strides

CUTLASS 3.5: Convolutions Support

For Hopper using TMA Im2Col and WGMMA

- CUTLASS 3.5 release includes support for convolutions natively in 3.x API
  - Beta release, feedback welcome!
- 1,2, and 3 spatial dimensions in a rank-agnostic manner
- Fprop, Dgrad, and Wgrad algorithms
- Asymmetric padding, dilations, and traversal strides
- Initial profiler integration
- Roadmap for future:
  - Performance
  - Strided dGrad
  - Extend profiler coverage for 1D conv

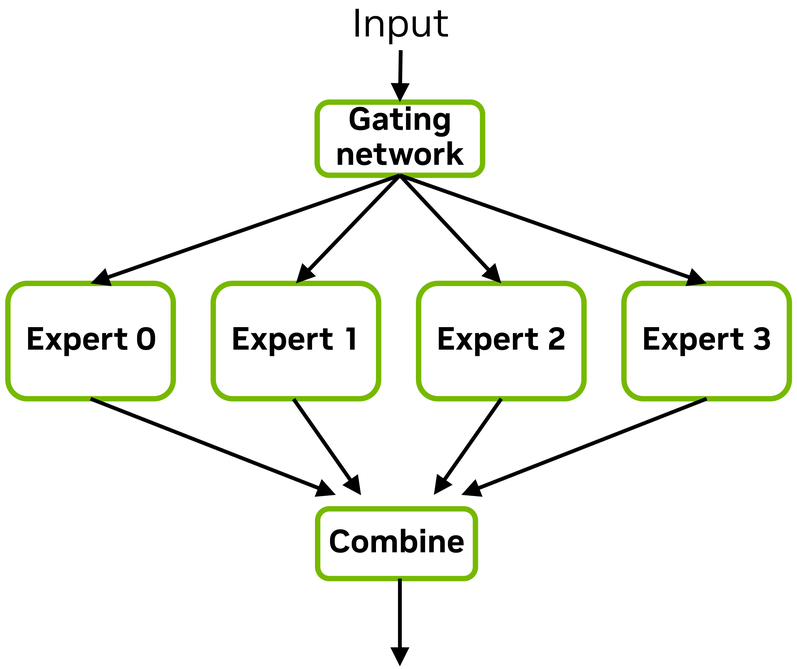


Agenda

- Convolutions in CUTLASS 3
- **Features for LLMs**
- Epilogue Visitor Tree
- Conclusion

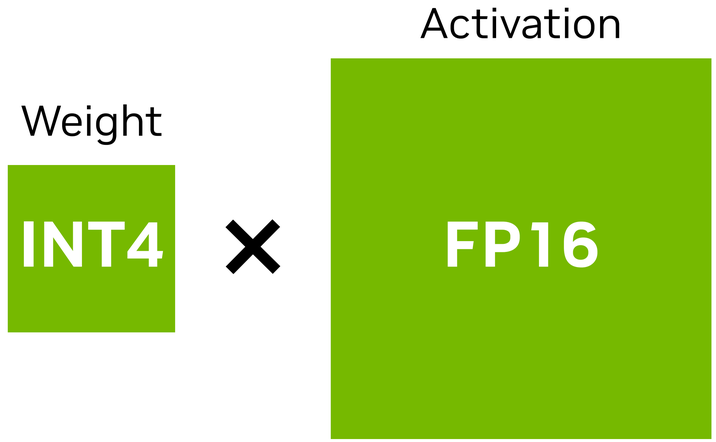
Training and deploying LLMs demands optimizations atop GEMM

Mixture of experts



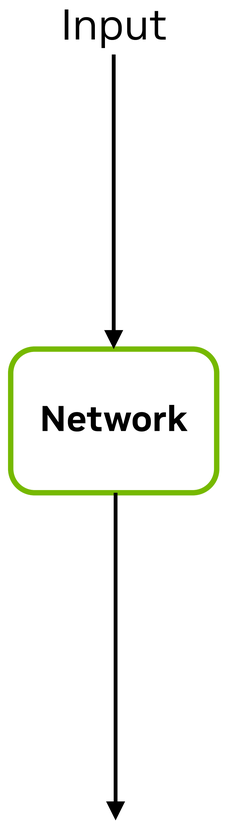
Grouped GEMM for Hopper in CUTLASS 3.4

Weight quantization

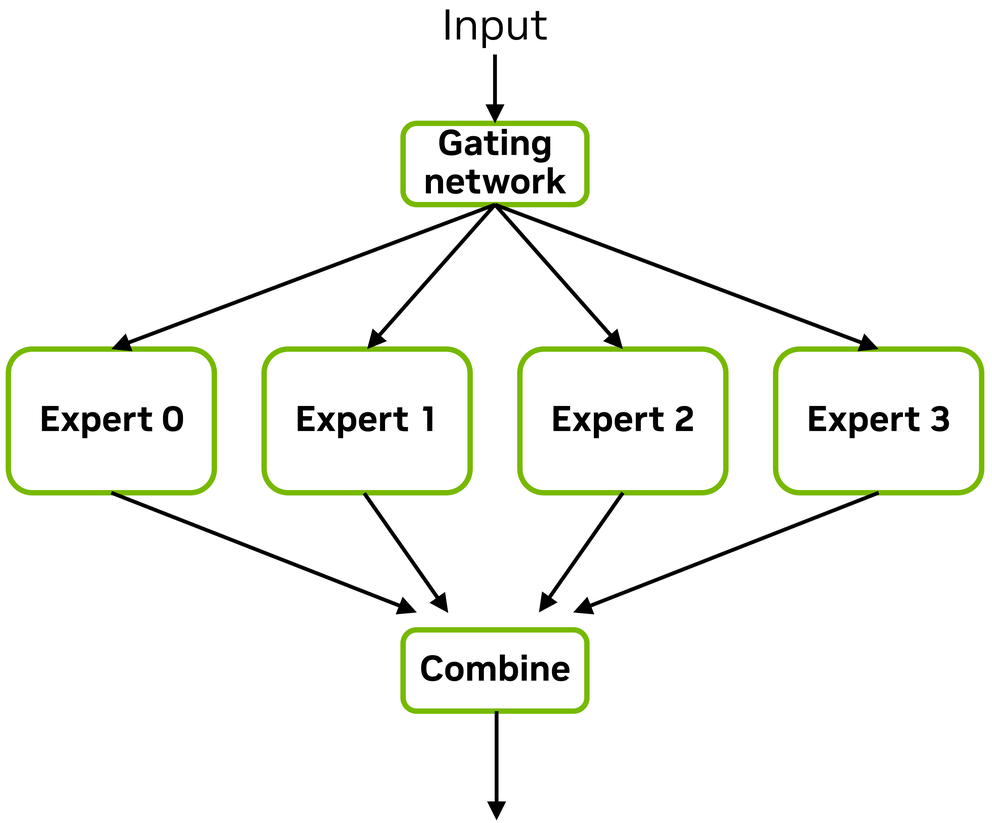


Mixed-input GEMM for Hopper and Ampere in CUTLASS 3.3

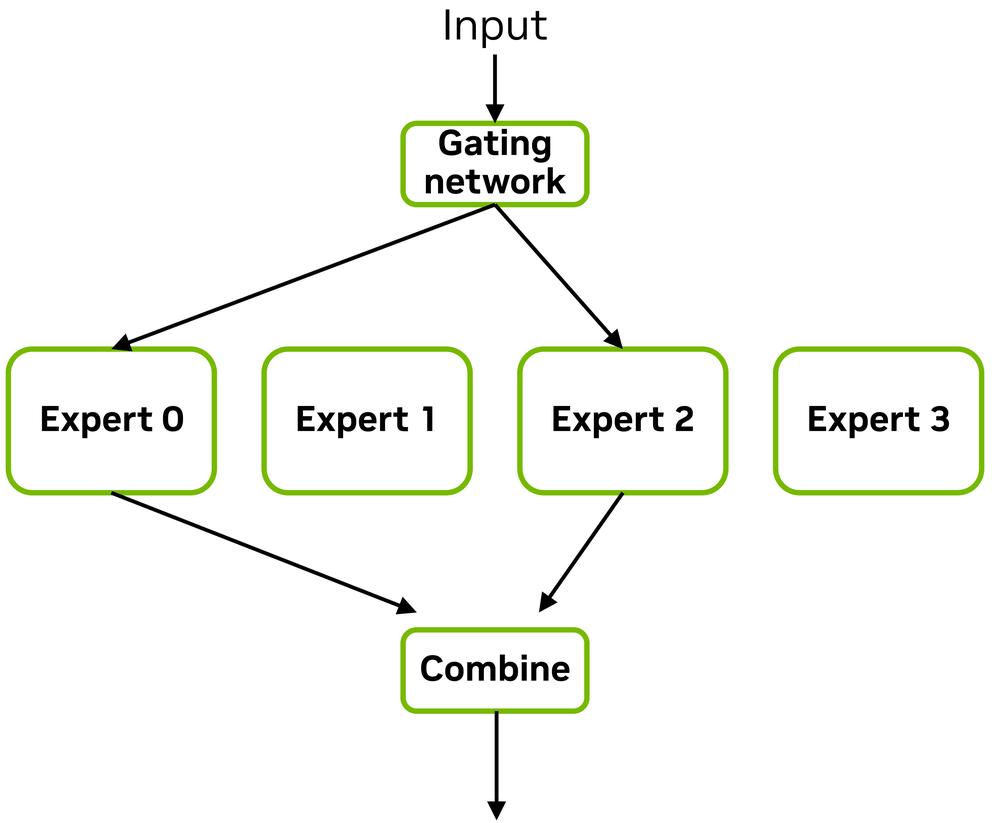
Mixture of experts (MoE) for compute-efficient LLMs



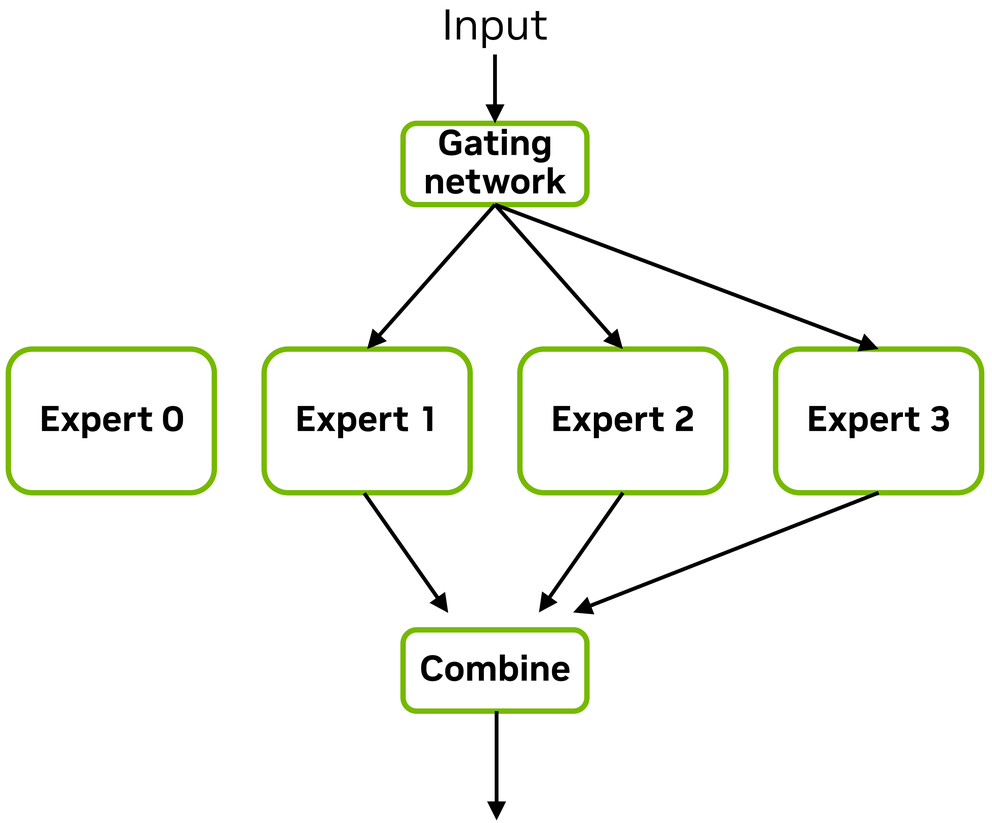
Mixture of experts (MoE) for compute-efficient LLMs



Mixture of experts (MoE) for compute-efficient LLMs



Mixture of experts (MoE) for compute-efficient LLMs



Mixture of experts (MoE) for compute-efficient LLMs

Desired operation:

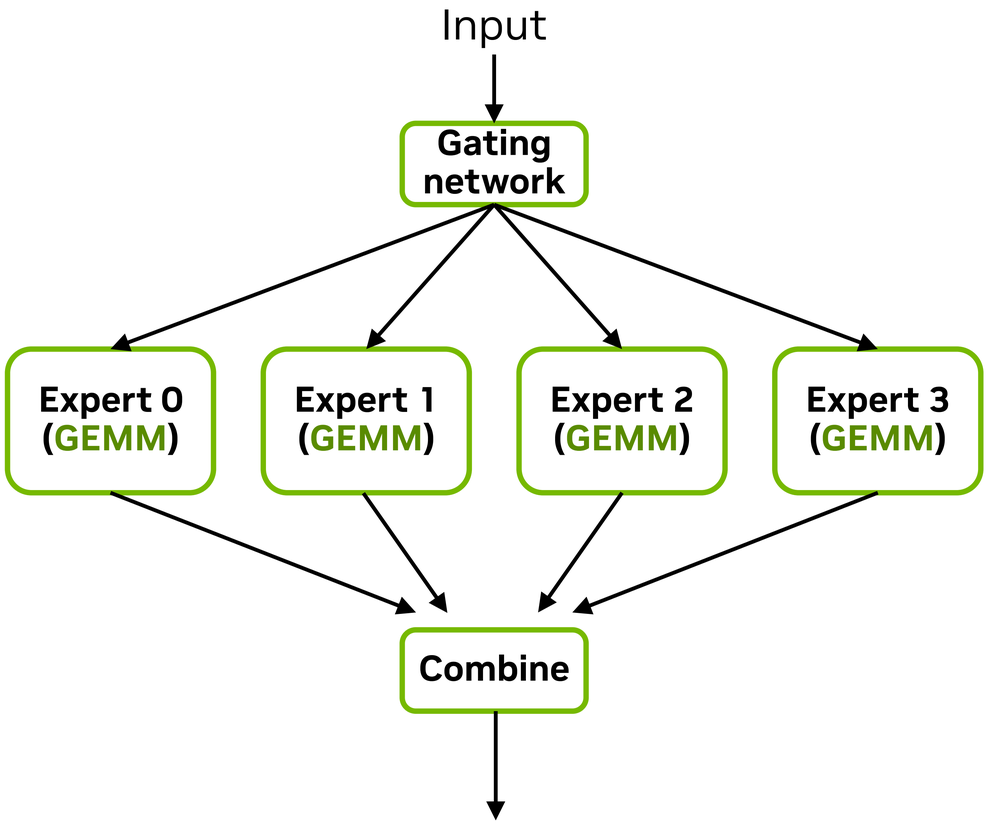
- Run multiple GEMMs in concurrently
- Each GEMM of potentially different size
- Not all GEMMs computed for each input

Batched GEMM? ✗

- Requires all GEMMs to have same size

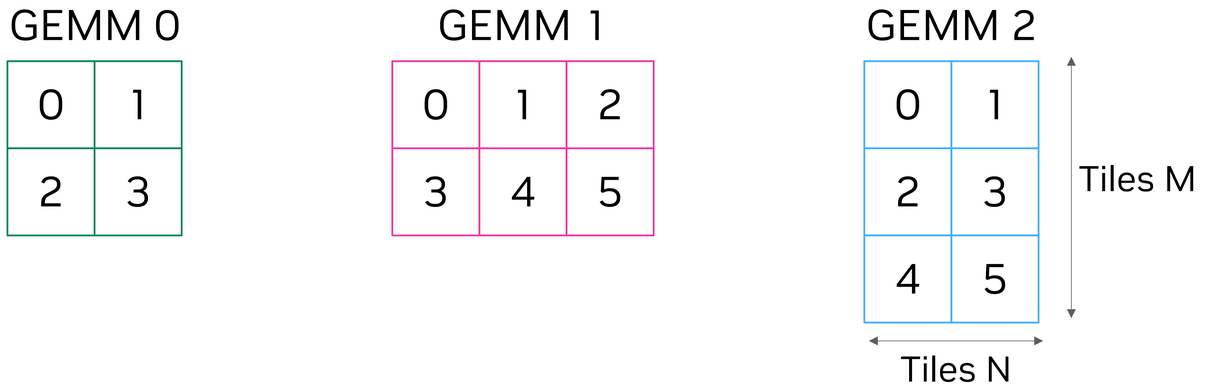
Grouped GEMM ✓

- Computes multiple GEMMs in single kernel
- Each GEMM can have different size and strides
- GEMM count and sizes do not need to be known at compile time

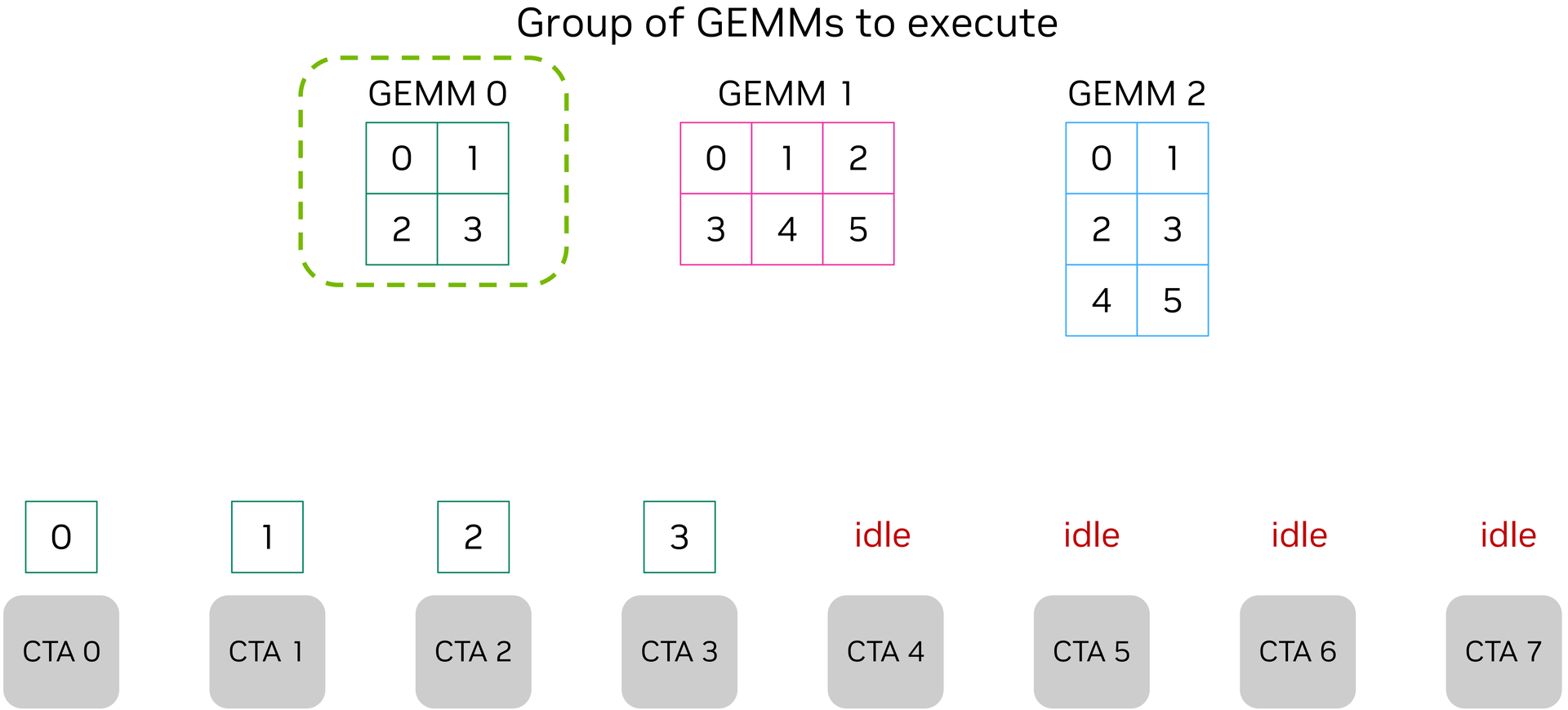


Grouped GEMM background

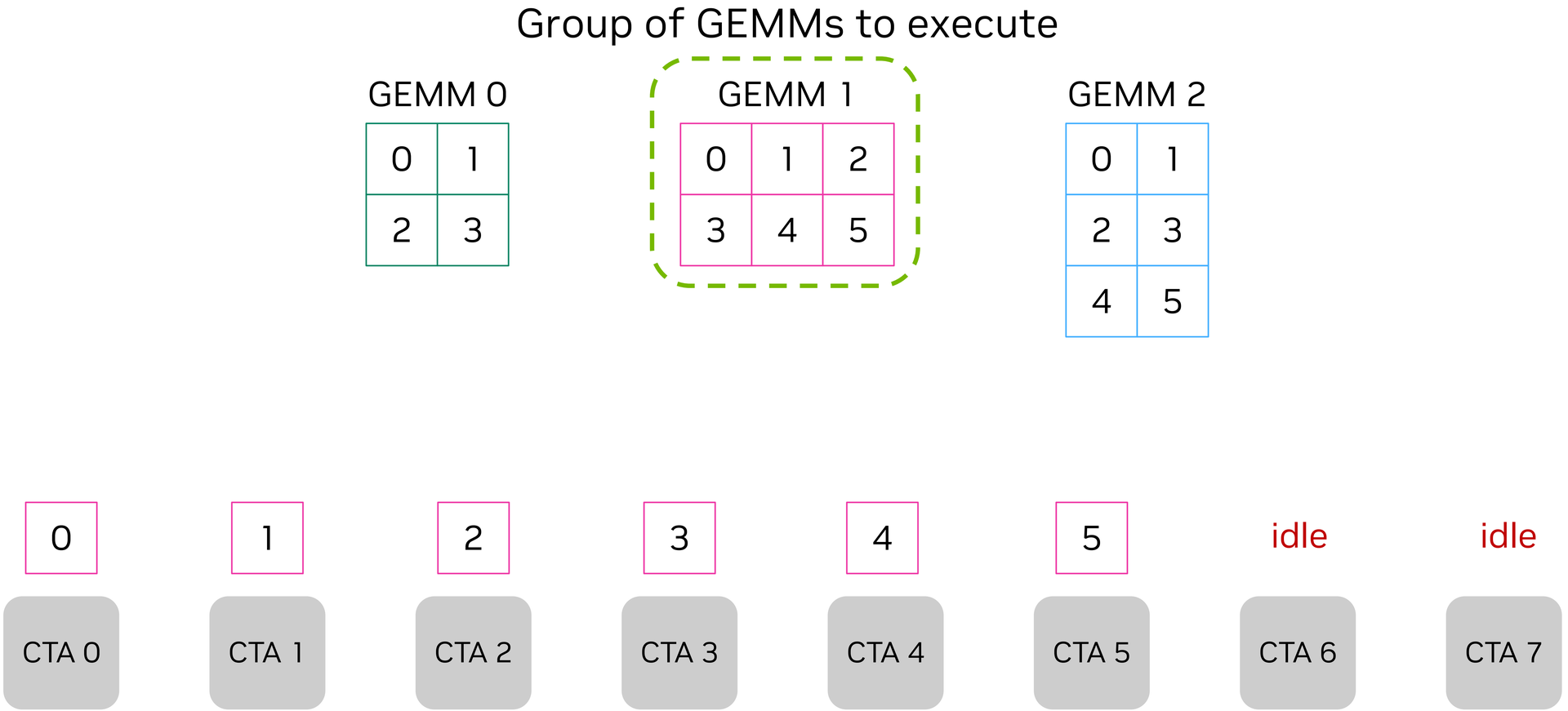
Group of GEMMs to execute



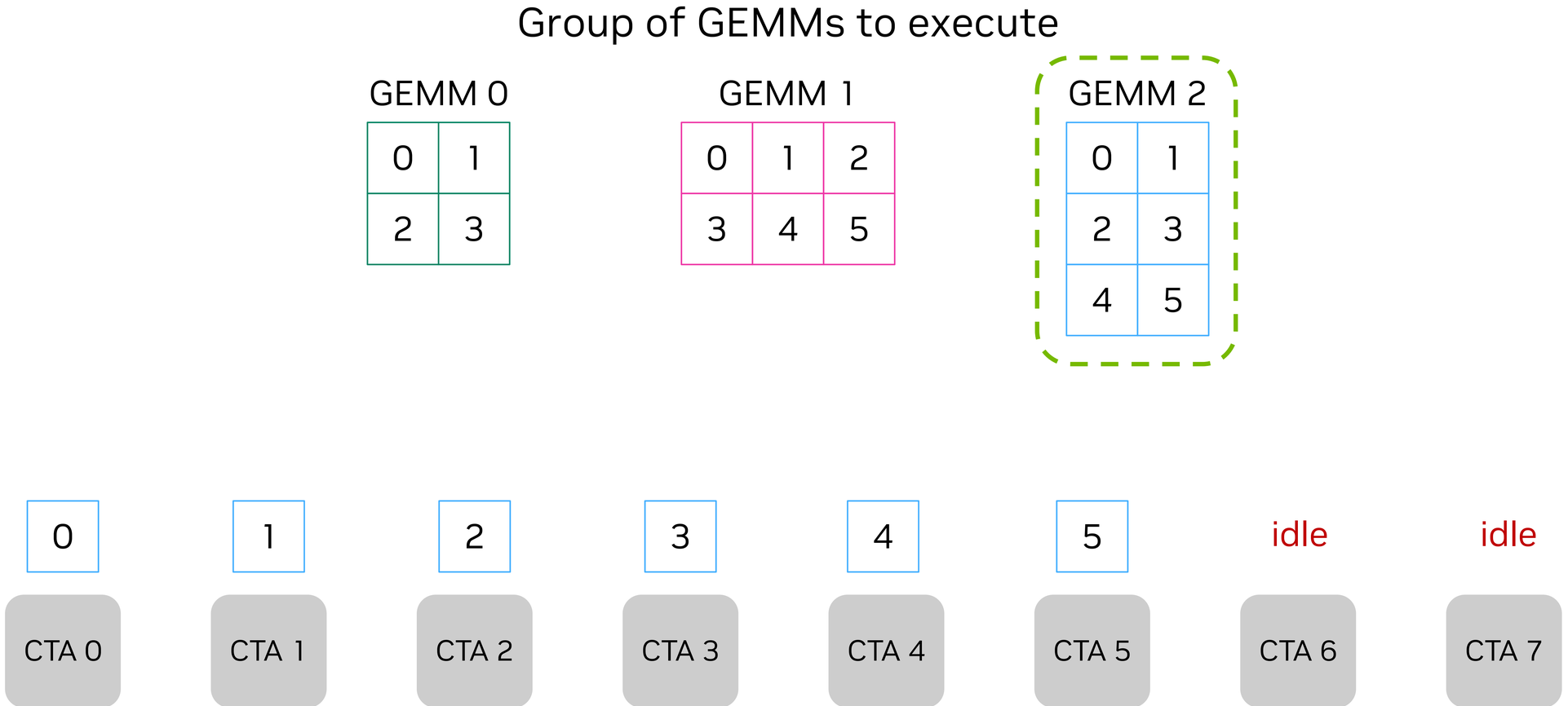
Serial execution of group of GEMMs



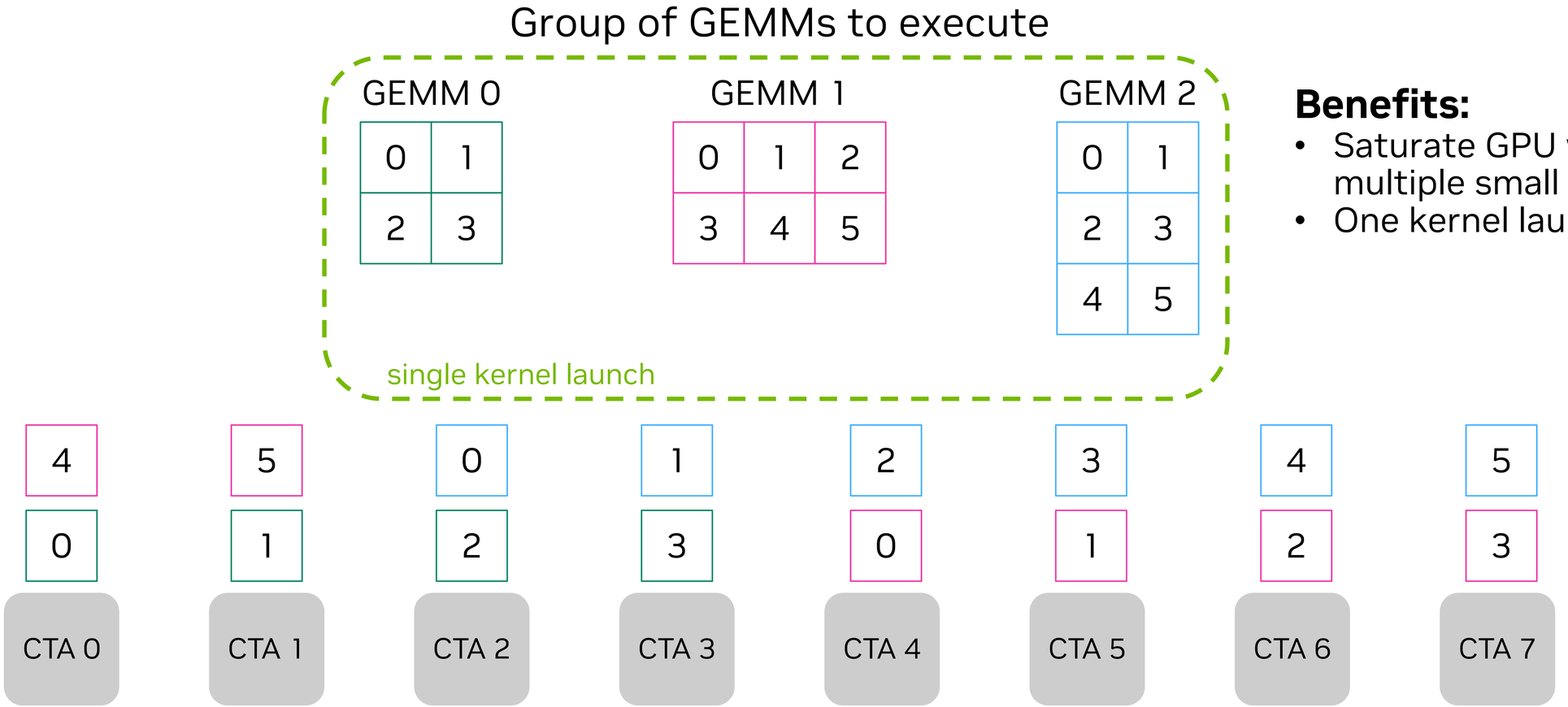
Serial execution of group of GEMMs



Serial execution of group of GEMMs



Grouped GEMM



Benefits:

- Saturate GPU with multiple small problems
- One kernel launch

Using Grouped GEMM for Hopper

```
using CollectiveEpilogue =
  typename cutlass::epilogue::collective::CollectiveBuilder<
    cutlass::arch::Sm90, cutlass::arch::OpClassTensorOp,
    TileShape, ClusterShape, EpiTileShape,
    ElementAccumulator, ElementCompute,
    ElementC, LayoutC*, AlignmentC,
    ElementD, LayoutD*, AlignmentD,
    cutlass::epilogue::collective::EpilogueScheduleAuto,
    EpilogueOperation
  >::CollectiveOp;

using CollectiveMainloop =
  typename cutlass::gemm::collective::CollectiveBuilder<
    cutlass::arch::Sm90, cutlass::arch::OpClassTensorOp,
    ElementA, LayoutA*, AlignmentA,
    ElementB, LayoutB*, AlignmentB,
    ElementAccumulator,
    TileShape, ClusterShape,
    cutlass::gemm::collective::StageCountAutoCarveout<
      sizeof(typename CollectiveEpilogue::SharedStorage))>,
    cutlass::gemm::collective::KernelScheduleAuto
>::CollectiveOp;
```

Grouped GEMM enhancements for Hopper: modifiable TMA descriptors

- Hopper's Tensor Memory Accelerator (TMA) requires using descriptors for performing copies
  - Single GEMM: one descriptor per operand (e.g., A, B) throughout lifetime of kernel
  - Grouped GEMM: need different descriptor fields for each GEMM in the group

- Single GEMM: construct descriptors on host and copied to device through launch parameters

- Grouped GEMM: tensormap.replace PTX introduced in CUDA 12.3 to update TMA descriptors on device
  - Placeholder TMA descriptors for A and B are constructed on host and passed through kernel launch parameters
  - Each CTA creates a copy of placeholder TMA descriptors in global memory
  - Each time a new problem is encountered, address, shape, and strides of descriptor are updated

Current grouped GEMM support in CUTLASS

- Ampere support since CUTLASS 2.8 (optimizations starting in 2.10)
- Hopper beta version added in CUTLASS 3.4 (optimizations expected in future releases)
- Current adoption:
  - TRT-LLM for Mixtral 8x7B
  - ByteDance ByteTransformer
  - PyTorch Geometric
- Try it out for yourself:

  examples/24_gemm_grouped
  examples/57_hopper_grouped_gemm
  examples/python/02_pytorch_extension_grouped_gemm

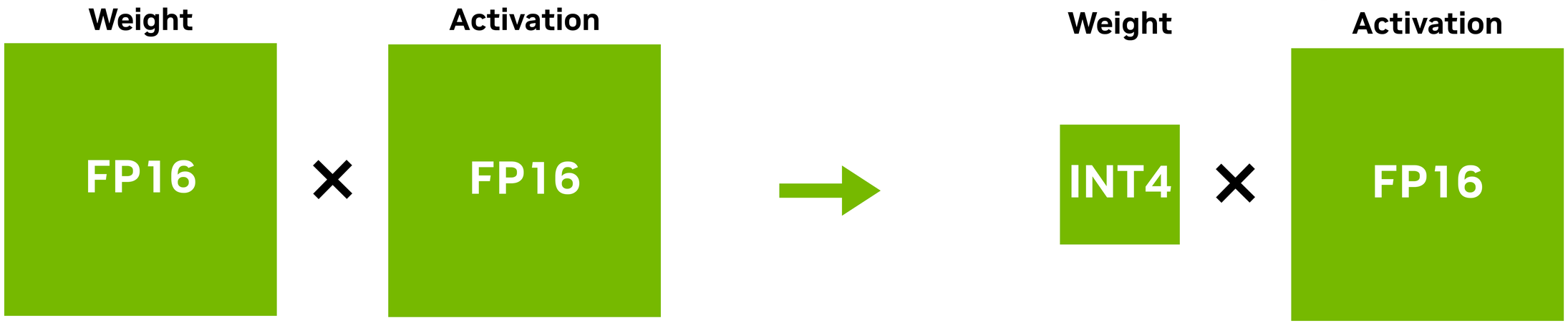
Mixed-input GEMM for LLM quantization

Unquantized GEMM

Weight: FP16 × Activation: FP16

Weight quantization (Mixed-input)

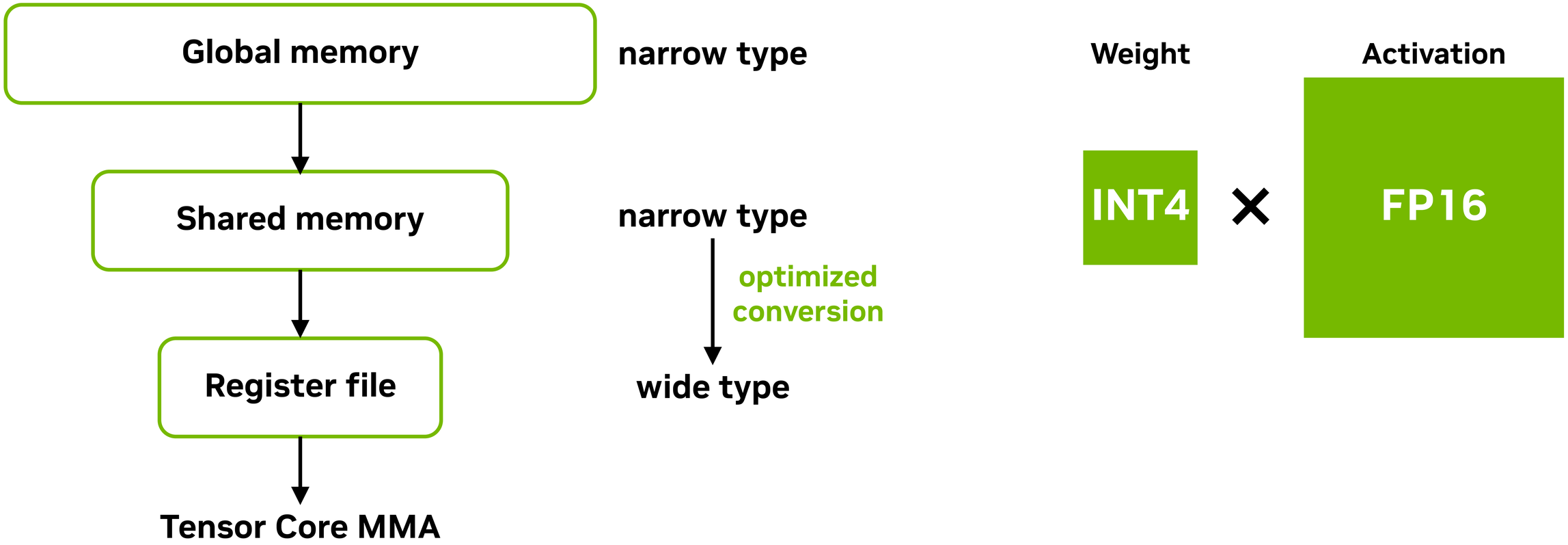
Weight: INT4 × Activation: FP16



Convert “narrow” type to “wide” type before performing Tensor Core operation.

Significant reduction in memory footprint and bandwidth

Mixed-input GEMM saves global and shared memory



CUTLASS support for mixed-input GEMMs

```
using CollectiveMainloop =
  typename cutlass::gemm::collective::CollectiveBuilder<
    cutlass::arch::Sm90, cutlass::arch::OpClassTensorOp,
    cutlass::half_t, LayoutA, AlignmentA,
    cutlass::half_t, LayoutB, AlignmentB,
    ElementAccumulator,
    TileShape, ClusterShape,
    StageCount,
    cutlass::gemm::collective::KernelScheduleAuto
  >::CollectiveOp;
```

CUTLASS support for mixed-input GEMMs

```
                using CollectiveMainloop =
                  typename cutlass::gemm::collective::CollectiveBuilder<
                    cutlass::arch::Sm90, cutlass::arch::OpClassTensorOp,
cutlass::int4b_t    ~~cutlass::half_t~~, LayoutA, AlignmentA,
                    cutlass::half_t, LayoutB, AlignmentB,
                    ElementAccumulator,
                    TileShape, ClusterShape,
                    StageCount,
                    cutlass::gemm::collective::KernelScheduleAuto
                  >::CollectiveOp;
```

Support for a variety of mixed-input combinations

```
using CollectiveMainloop =
  typename cutlass::gemm::collective::CollectiveBuilder<
    cutlass::arch::Sm90, cutlass::arch::OpClassTensorOp,
    ElementA, LayoutA, AlignmentA,
    ElementB, LayoutB, AlignmentB,
    ElementAccumulator,
    TileShape, ClusterShape,
    StageCount,
    KernelScheduleType
  >::CollectiveOp;
```

Requirements:

- Tensor Core instruction exists for wider data type
- Operands meet TMA requirements

FP16 X INT8
FP16 X INT4
BF16 X INT8
FP8 X INT4
INT8 X FP16
INT8 X BF16
INT4 X FP8
★ **INT2 X FP16**
★ **INT1 X FP8**
(and more)

Current mixed-input GEMM support in CUTLASS

- Hopper support added in CUTLASS 3.3
  - Currently used in TRT-LLM

Generative AI / LLMs

English

NVIDIA TensorRT-LLM Enhancements Deliver Massive Large Language Model Speedups on NVIDIA H200

Dec 04, 2023

+14 Like Discuss (0)

By Ashraf Eassa, Dave Salvator and Nick Comly

link

- Ampere support contributed in CUTLASS 3.3
  - Thanks to Manish Gupta of Google

BLOG ›

Mixed-input matrix multiplication performance optimizations

FRIDAY, JANUARY 26, 2024

Posted by Manish Gupta, Staff Software Engineer, Google Research

link

- Try it out for yourself:

  examples/55_hopper_mixed_dtype_gemm

Agenda

- Convolutions in CUTLASS 3
- Features for LLMs
- **Epilogue Visitor Tree**
- Conclusion

Common workloads invoke a variety of epilogues

GEMM + ReLU

GEMM + Bias Add

GEMM + Bias Add + GELU

GEMM + GELU + Residual

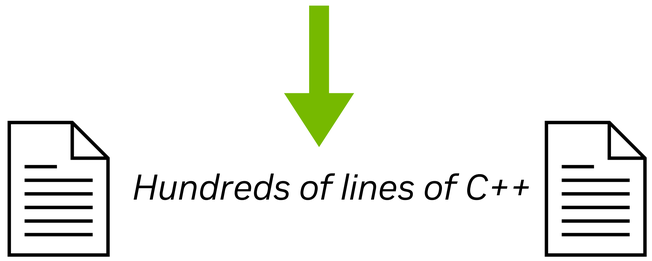
GEMM + Bias Add + ReLU + Row reduction

GEMM + ...

- Popular deep learning workloads invoke many operations after GEMM
- Fusing “epilogue” to GEMM is critical for achieving high performance

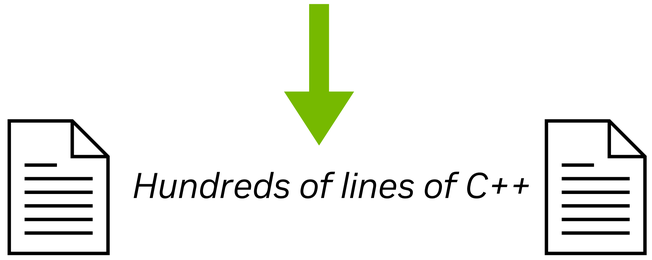
Writing a new kernel for each fusion is cumbersome

GEMM + My Custom Fusion



epilogue_with_myfusion.h gemm_with_myfusion.h

GEMM + My Custom Fusion 2



epilogue_with_myfusion2.h gemm_with_myfusion2.h

- Prior workflow:
  - Determine where to place customized logic
  - Copy existing GEMM and epilogue files
  - Repeat for each fusion you want to add
- Results in high development effort and code duplication

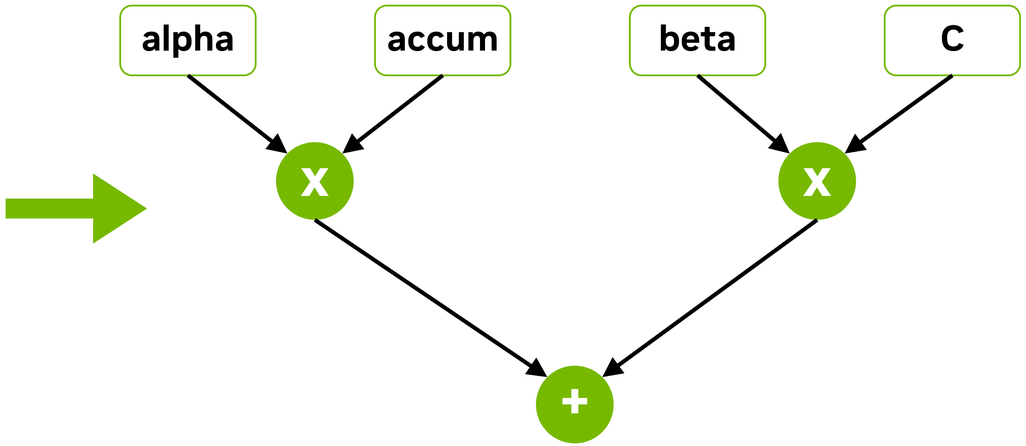
Epilogue Visitor Tree (EVT): building blocks for composing fused epilogues

- Set of primitive nodes that can be composed together to build complex epilogues

Load	Compute	Store
Accumulator Aux tensor Row broadcast Column broadcast Scalar broadcast	Compute (elementwise, binary, ternary)	Aux tensor Row reduction Column reduction Scalar reduction

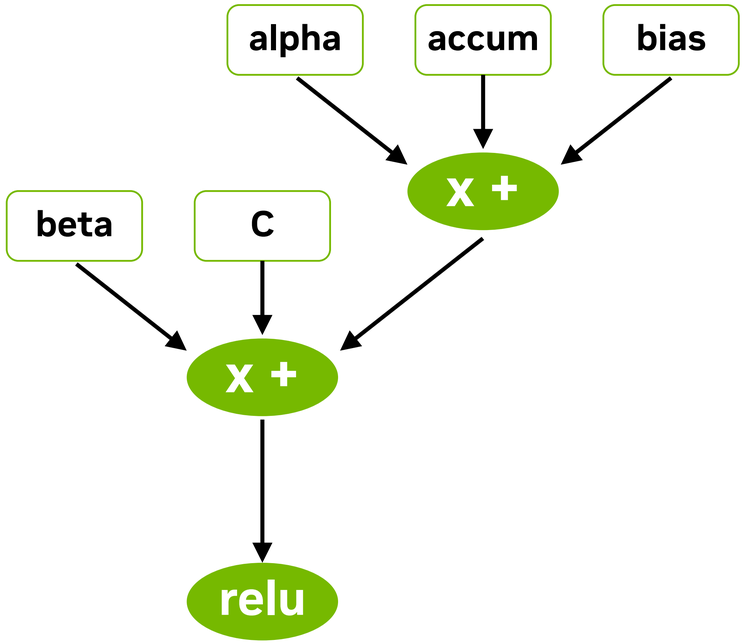
- Compose together nodes in a tree or DAG

(alpha * accumulators) + (beta * C)



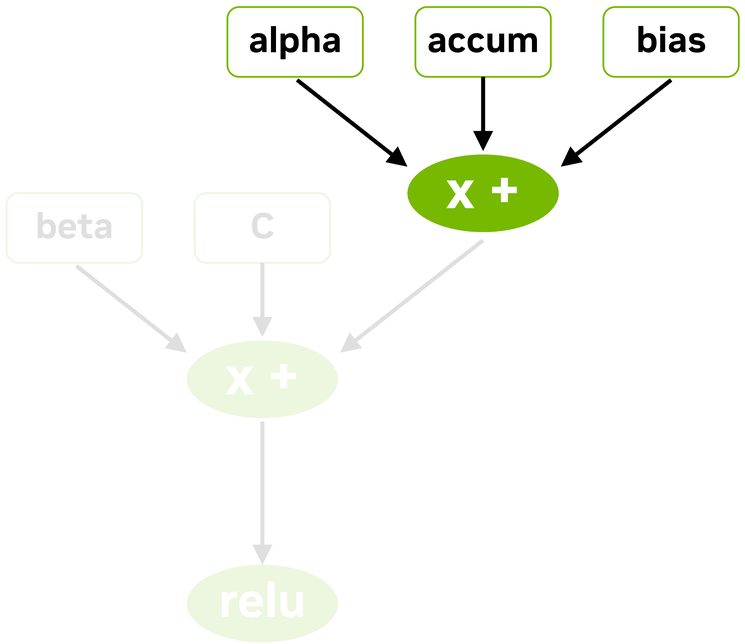
Adding an epilogue visitor tree to a GEMM in CUTLASS 3.x (C++)

ReLU((alpha * accumulators) + bias + (beta * C))



Adding an epilogue visitor tree to a GEMM in CUTLASS 3.x (C++)

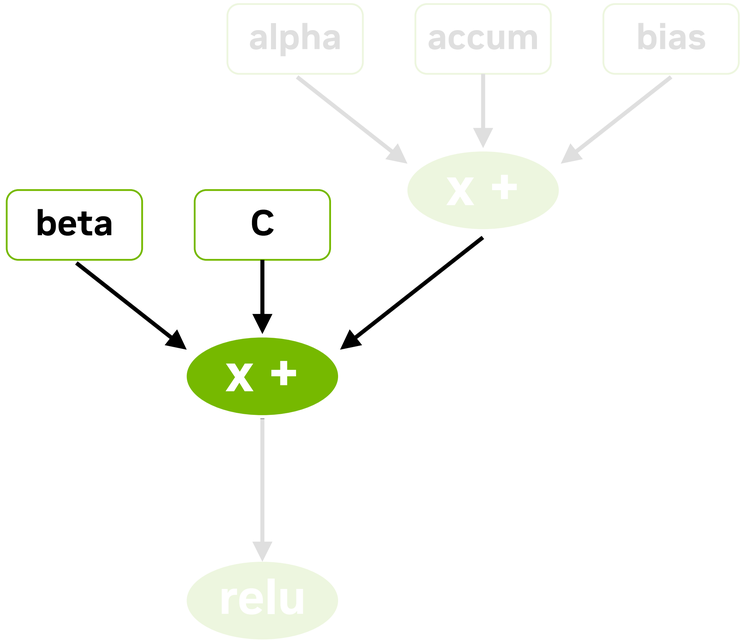
ReLU(**(alpha * accumulators) + bias** + (beta * C))




```
using Alpha = Sm90ScalarBroadcast<ElementScalar>;
using Accum = Sm90AccFetch;
using Bias  = Sm90ColBroadcast<
    0, TileShape, ElementBias, Stride<_1,_0,int>, AlignmentBias>;
using MultiplyAdd = Sm90Compute<
    multiply_add, ElementCompute, ElementCompute, RoundStyle>;
using EVTCompute0 = Sm90EVT<MultiplyAdd, Alpha, Accum, Bias>;
```

Adding an epilogue visitor tree to a GEMM in CUTLASS 3.x (C++)

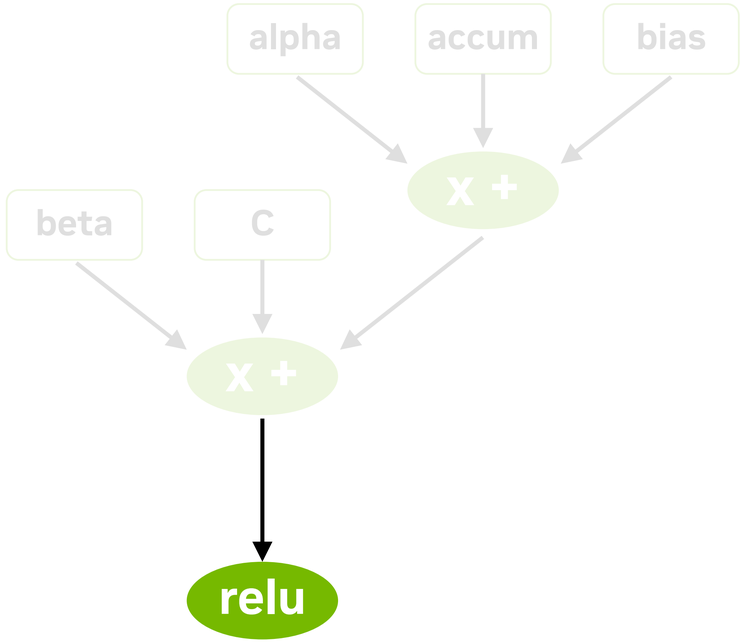
ReLU((alpha * accumulators) + bias **+ (beta * C)**)




```
using Alpha = Sm90ScalarBroadcast<ElementScalar>;
using Accum = Sm90AccFetch;
using Bias  = Sm90ColBroadcast<
    0, TileShape, ElementBias, Stride<_1,_0,int>, AlignmentBias>;
using MultiplyAdd = Sm90Compute<
    multiply_add, ElementCompute, ElementCompute, RoundStyle>;
using EVTCompute0 = Sm90EVT<MultiplyAdd, Alpha, Accum, Bias>;


using Beta = Sm90ScalarBroadcast<ElementScalar>;
using C    = Sm90SrcFetch<ElementSource>;
using EVTCompute1 = Sm90EVT<MultiplyAdd, Beta, C, EVTCompute0>;
```

Adding an epilogue visitor tree to a GEMM in CUTLASS 3.x (C++)

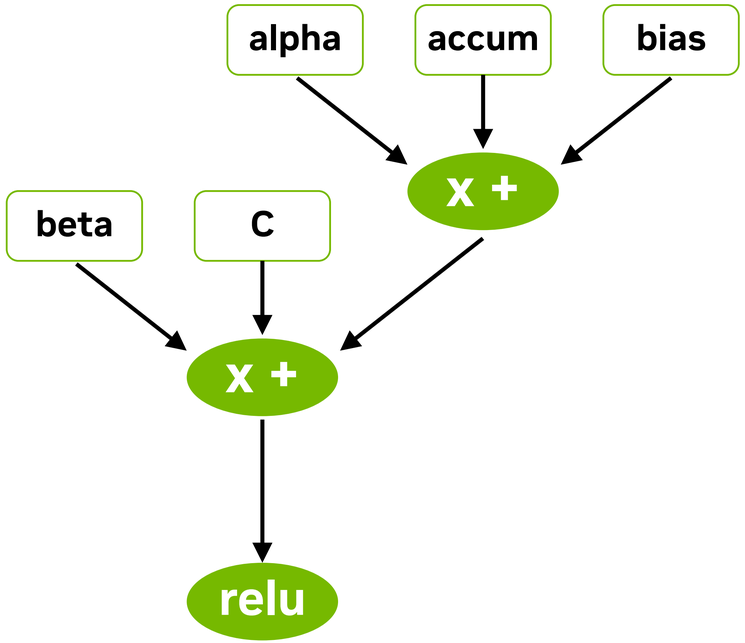
ReLU((alpha * accumulators) + bias + (beta * C) **)**




```
using Alpha = Sm90ScalarBroadcast<ElementScalar>;
using Accum = Sm90AccFetch;
using Bias  = Sm90ColBroadcast<
    0, TileShape, ElementBias, Stride<_1,_0,int>, AlignmentBias>;
using MultiplyAdd = Sm90Compute<
    multiply_add, ElementCompute, ElementCompute, RoundStyle>;
using EVTCompute0 = Sm90EVT<MultiplyAdd, Alpha, Accum, Bias>;


using Beta = Sm90ScalarBroadcast<ElementScalar>;
using C    = Sm90SrcFetch<ElementSource>;
using EVTCompute1 = Sm90EVT<MultiplyAdd, Beta, C, EVTCompute0>;


using ReLUAct = Sm90Compute<
    ReLu, ElementOutput, ElementCompute, RoundStyle>;
using EVTOutput = Sm90EVT<ReLUAct, EVTCompute1>;
```

Adding an epilogue visitor tree to a GEMM in CUTLASS 3.x (C++)

ReLU((alpha * accumulators) + bias + (beta * C))




```
using EVTOutput = Sm90EVT<ReLUAct, EVTCompute1>;

using CollectiveEpilogue = typename
cutlass::epilogue::collective::CollectiveBuilder<
      cutlass::arch::Sm90, cutlass::arch::OpClassTensorOp,
      TileShape, ClusterShape,
      cutlass::epilogue::collective::EpilogueTileAuto,
      ElementAccumulator, ElementCompute,
      ElementC, LayoutC, AlignmentC,
      ElementD, LayoutD, AlignmentD,
      EpilogueScheduleType,
      EVTOutput
>::CollectiveOp;
```

Pre-baked aliases for common patterns

```
using Alpha = Sm90ScalarBroadcast<ElementScalar>;
using Accum = Sm90AccFetch;
using Bias  = Sm90ColBroadcast<
    0, TileShape, ElementBias, Stride<_1,_0,int>, AlignmentBias>;
using MultiplyAdd = Sm90Compute<
    multiply_add, ElementCompute, ElementCompute, RoundStyle>;
using EVTCompute0 = Sm90EVT<MultiplyAdd, Alpha, Accum, Bias>;


using Beta = Sm90ScalarBroadcast<ElementScalar>;
using C    = Sm90SrcFetch<ElementSource>;
using EVTCompute1 = Sm90EVT<MultiplyAdd, Beta, C, EVTCompute0>;


using ReLUAct = Sm90Compute<ReLu, ElementOutput, ElementCompute, RoundStyle>;
using EVTOutput = Sm90EVT<ReLUAct, EVTCompute1>;
```

```
using EVTOutput = Sm90LinCombPerRowBiasEltAct<
    TileShape, ReLu, ElementOutput, ElementCompute>;
```

Epilogue visitor tree in Python

1. Declare a basic GEMM

```
import cutlass
plan = cutlass.op.Gemm(
  element=torch.float32,
  layout=cutlass.LayoutType.RowMajor)
```

2. Define an epilogue as a Python function

```
def my_epilogue(accum, alpha, C, beta, bias):
  D = relu(alpha * accum + beta * C + bias)
  return D
```

3. Define types and shapes of each EVT operand/output

```
empty_mn = torch.empty(size=(m, n), dtype=torch.float32)
empty_bias = torch.empty(size=(m, 1), dtype=torch.float32)
examples_inputs = {
  "accum": empty_mn, "C": empty_mn, "D": empty_mn,
  "alpha": 1.0, "beta": 1.0, "bias": empty_bias,
}
```

4. Construct the EVT and assign it to the GEMM

```
plan.epilogue_visitor = cutlass.epilogue.trace(
  my_epilogue, examples_inputs)
```

5. Compile and run the kernel

```
A, B, C, D, bias = ...
visitor_args = {
    "alpha": 2.0, "beta": 0.0,
    "C": C, "D": D, "bias": bias
}
plan.run(A, B, C, D, visitor_args=visitor_args)
```

EVT handles writing optimized epilogue loops for you

Consumer store warpgroup pseudocode

```
for (int i = 0; i < NumEpiSubtiles; ++i) {
  Tensor tRS_rAcc_frg_mn = tRS_rAcc_frg(_,i);

  copy(tiled_s2r, ...); // Copy C from smem to registers



  // Compute epilogue on accumulator fragments
  tRS_rD_frg(epi_v) = compute(tRS_rAcc_frg_mn, ...);



  copy(tiled_r2s, ...); // Copy results from register to smem



  copy(params.tma_store_d, ...); // Copy D smem to gmem


}
```

EVT handles writing optimized epilogue loops for you

Consumer store warpgroup pseudocode

```
callbacks.begin();
for (int i = 0; i < NumEpiSubtiles; ++i) {
  Tensor tRS_rAcc_frg_mn = tRS_rAcc_frg(_,i);

  copy(tiled_s2r, ...); // Copy C from smem to registers

  callbacks.previsit(...);

  // Compute epilogue on accumulator fragments
  tRS_rD_frg(epi_v) = callbacks.visit(tRS_rAcc_frg_mn, ...);

  callbacks.reduce(...);  // Smem reduction callback entry

  copy(tiled_r2s, ...); // Copy results from register to smem

  callbacks.postreduce(...);

  copy(params.tma_store_d, ...); // Copy D smem to gmem

  callbacks.tma_store(..);
}
callbacks.end();
```

Epilogue Visitor Tree support in CUTLASS

- Available in both C++ and Python interfaces
- Support Hopper, Ada, and Ampere
- Support for sparse GEMMs added for Ampere
  - Thanks to open-source contributor Aleksandar Samardžić
  - Support for sparse GEMMs in Hopper coming soon
- Try it out for yourself:

  examples/49_collective_builder

  examples/python/04_epilogue_visitor.ipynb

Agenda

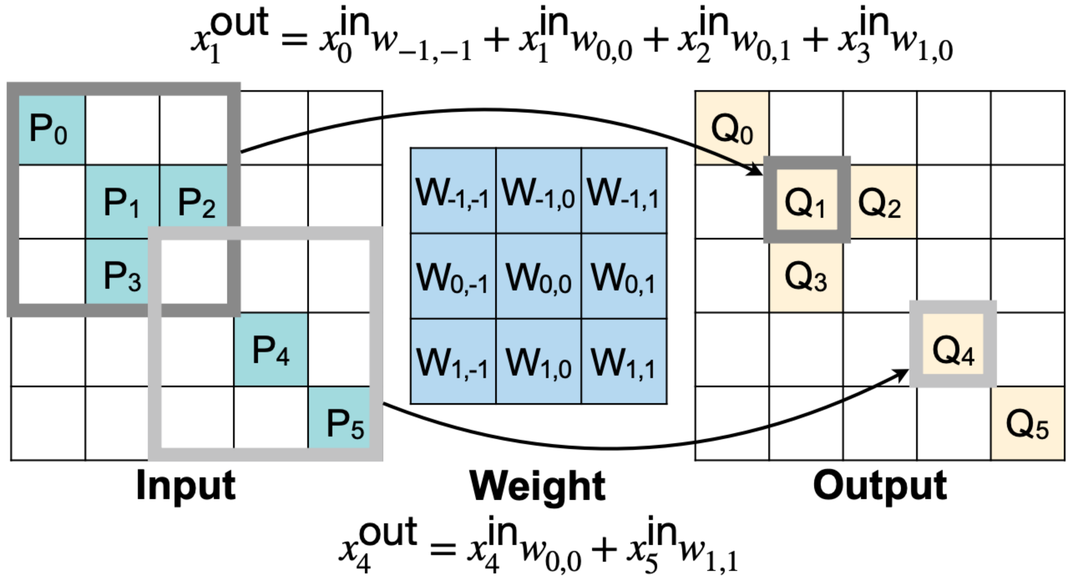
- Convolutions in CUTLASS 3
- Features for LLMs
- Epilogue Visitor Tree
- Conclusion

One More Thing …

Ampere Gather/Scatter Convolution

A tutorial on writing custom CUDA kernels using CUTLASS

- Problem statement: need spatially sparse convolutions
- Gather along NDHW dimensions
- Scatter along NZPQ dimensions
- Fully dense channel dims KC and filter tensor KTRSC
- Useful for convolutions on point-cloud data



Spatially sparse convolution

Figure credit: TorchSparse++

Step 0: Kernel API

A tutorial on writing custom CUDA kernels using CUTLASS

- Set up device code to accept CuTe tensors natively

```
//
// Conv functor
//
template <class EngineFlt,
          class EngineAct, class LayoutAct,
          class EngineOut, class LayoutOut>
void __device__
operator()(cute::Tensor<EngineFlt, LayoutFlt> mFlt, // ( K,         (C,T,R,S)) := MxK
           cute::Tensor<EngineAct, LayoutAct> mAct, // ((N,Z,P,Q), (C,T,R,S)) := NxK
           cute::Tensor<EngineOut, LayoutOut> mOut, // ( K,         (N,Z,P,Q)) := MxN
           char* smem_buf) const {
```

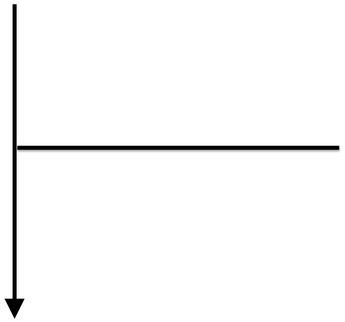
Step 1: Ampere Dense 3D conv

Composition power of CUTLASS 3.x API

- Use CuTe layout algebra to transform input tensors to im2col domain

```
// im2col transformed activation layout: ((nzpq), (ctrs)) => idx
Layout xformed_act_layout = make_layout(
  make_shape (make_shape (      N,     Z,   P, Q), make_shape ( C,       T,   R, S)),
  make_stride(make_stride(D*H*W*C, H*W*C, W*C, C), make_stride(_1{}, H*W*C, W*C, C)));

Tensor gAct = make_tensor(ptr, xformed_act_layout);
```

- Invoke existing GEMM collective on the logical MNK problem tiles



```
using CollectiveMainloop = typename gemm::collective::CollectiveMma<
  cutlass::gemm::MainloopSm80CpAsyncUnpredicated<3>,
  Shape<TileM,TileN,TileK>,
  ElementFlt, Underscore,
  ElementAct, Underscore,
  TiledMma, …>;
```

```
CollectiveMainloop collective_mma;
collective_mma(
  accum,
  gAct, gFlt,
  accum,
  k_tile_iter, k_tile_count,
  Underscore{}, threadIdx.x, smem_buf);
```

Step 2: Representing Gather/Scatter Tensors

Representation power of CuTe layouts

- Layouts are just functions: (domain coords) -> (codomain)
- Construct a layout that maps logical input coordinates to gather index pair

```
// ((nzpq), (ctrs)) => (idx_buffer_idx, dense_offset)
auto EG = E<0>{}; // Gather basis     (1,0) (idx_buffer_idx)
auto EC = E<1>{}; // Contiguous basis (0,1) (dense_offset)
auto xformed_act_logical_inner = make_layout(
  make_shape (make_shape (       N,      Z,    P,  Q), make_shape ( C,      T,    R,  S)),
  make_stride(make_stride(D*H*W*EG, H*W*EG, W*EG, EG), make_stride(EC, H*W*EG, W*EG, EG)));
```

- Construct a layout that maps gather indices to a linear offset

```
// (idx_buffer_idx, dense_offset) => idx

auto xformed_act_gather_outer = make_layout(
  make_shape(_1{},_1{}),
  make_stride(CustomStride{IndexedGather{gather_idx_buf}, C}, _1{}));
```

- Compose them together to end up with a gather indirection!

```
// ((nzpq), (ctrs)) => idx
auto xformed_act_composed_layout = composition(
  xformed_act_gather_outer,
  make_arithmetic_tuple(_0{}, _0{}),
  xformed_act_logical_inner);
```

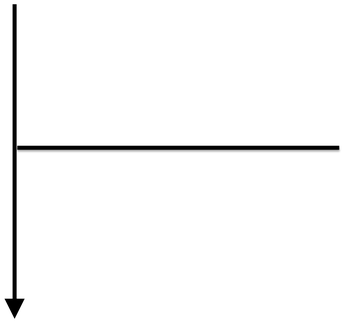
Step 3: Compose with existing collective

Composition power of CUTLASS collectives

- Replace the affine layout with composed gather layout

```
// ((nzpq), (ctrs)) => idx
auto xformed_act_composed_layout = composition(
  xformed_act_gather_outer,
  make_arithmetic_tuple(_0{}, _0{}),
  xformed_act_logical_inner);

Tensor gAct = make_tensor(ptr, xformed_act_composed_layout);
```

- Invoke existing GEMM collective on the logical MNK problem tiles



```
using CollectiveMainloop = typename gemm::collective::CollectiveMma<
  cutlass::gemm::MainloopSm80CpAsyncUnpredicated<3>,
  Shape<TileM,TileN,TileK>,
  ElementFlt, Underscore,
  ElementAct, Underscore,
  TiledMma, …>;
```

```
CollectiveMainloop collective_mma;
collective_mma(
  accum,
  gAct, gFlt,
  accum,
  k_tile_iter, k_tile_count,
  Underscore{}, threadIdx.x, smem_buf);
```

Step 4: Optimization

Exploit domain specific information

- Assume problem shapes other than image count are known at compile time

- Fully static global memory layouts for filter tensor:

```
auto GmemLayoutFlt = make_ordered_layout(
  Shape< K, Shape< C, T, R, S>>{},
  Order<_4, Order<_0,_3,_2,_1>>{});
```

- Only image count (N) dynamic
  - Fully static activation strides
  - Almost fully static activation shape
  - Fully static padding, dilation, tstrides

- CuTe layout representation eliminates all runtime index computation
- Eliminates the need for
  - complex OOB predication
  - Delta tables
- ~100 lines of code to achieve SOTA perf

```
template <class EngineFlt, class TensorActivation, class TensorOutput>
  void __device__
  operator()(cute::Tensor<EngineFlt, GmemLayoutFlt> mFlt, // ( K,        (C,T,R,S))
             TensorActivation                        mAct, // ((N,Z,P,Q), (C,T,R,S))
             TensorOutput                            mOut, // ( K,        (N,Z,P,Q))
             char* smem_buf) const {
    using namespace cute;
    using CollectiveMainloop = typename cutlass::gemm::collective::CollectiveMma<
        cutlass::gemm::MainloopSm80CpAsyncUnpredicated<3>,
        Shape<TileM,TileN,TileK>,
        ElementFlt, Underscore,
        ElementAct, Underscore,
        TiledMma,
        GmemTiledCopyFlt, SmemLayoutAtomFlt, SmemCopyAtomFlt, cute::identity,
        GmemTiledCopyAct, SmemLayoutAtomAct, SmemCopyAtomAct, cute::identity>;

    TiledMma tiled_mma;
    Tensor accum = partition_fragment_C(tiled_mma, TilerOut{});
    clear(accum);

    // Set up tensors
    Tensor gA_mk = local_tile(mFlt, TilerFlt{}, make_coord(_,_));
    Tensor gB_nk = local_tile(mAct, TilerAct{}, make_coord(_,_));
    Tensor gC_mn = local_tile(mOut, TilerOut{}, make_coord(_,_));

    // Compute m_coord and n_coord with their post-tiled shapes
    auto m_coord = idx2crd(int(blockIdx.y), shape<2>(gA_mk));
    auto n_coord = idx2crd(int(blockIdx.x), shape<2>(gB_nk));
    Tensor gA = gA_mk(_,_,m_coord,_);
    Tensor gB = gB_nk(_,_,n_coord,_);
    Tensor gC = gC_mn(_,_,m_coord,n_coord);

    auto k_tile_iter = cute::make_coord_iterator(size<2>(gA));
    int k_tile_count = size<2>(gA);

    CollectiveMainloop collective_mma;
    collective_mma(
      accum,
      gA, gB,
      accum,
      k_tile_iter, k_tile_count,
      Underscore{}, // no residue since we do not support predication
      threadIdx.x,
      smem_buf);

    // Epilogue
    SharedStorage& storage = *reinterpret_cast<SharedStorage*>(smem_buf);
    Tensor sC = make_tensor(make_smem_ptr(&storage.epilogue.sCMatrix[0]), SmemLayoutOut{});

    auto smem_tiled_copy_C = make_tiled_copy_C(SmemCopyAtomOut{}, tiled_mma);
    auto smem_thr_copy_C = smem_tiled_copy_C.get_slice(threadIdx.x);
    auto tCrC = smem_thr_copy_C.retile_S(accum);
    auto tCsC = smem_thr_copy_C.partition_D(sC);
    copy(smem_tiled_copy_C, tCrC, tCsC);

    __syncthreads();

    GmemTiledCopyOut gmem_tiled_copy_C;
    auto gmem_thr_copy_C = gmem_tiled_copy_C.get_slice(threadIdx.x);
    auto tDsC = gmem_thr_copy_C.partition_S(sC);
    auto tDgC = gmem_thr_copy_C.partition_D(gC);
    copy(gmem_tiled_copy_C, tDsC, tDgC);
  }
```

Try it out for yourself!

Case studies in custom CUTLASS 3.x fusions

- Tri Dao's Flash Attention V2:
  - https://github.com/Dao-AILab/flash-attention/

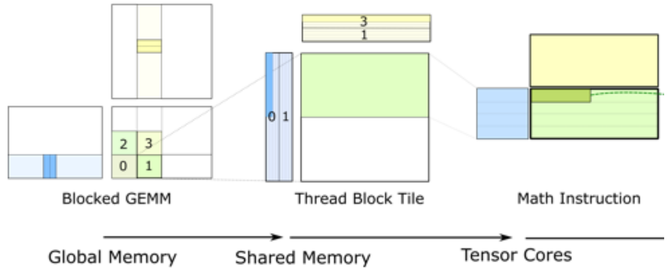
- Colfax research's FP8 implementation of FA-V2:
  - https://github.com/ColfaxResearch/cutlass-kernels/tree/master/src/fmha

examples/59_ampere_gather_scatter_conv

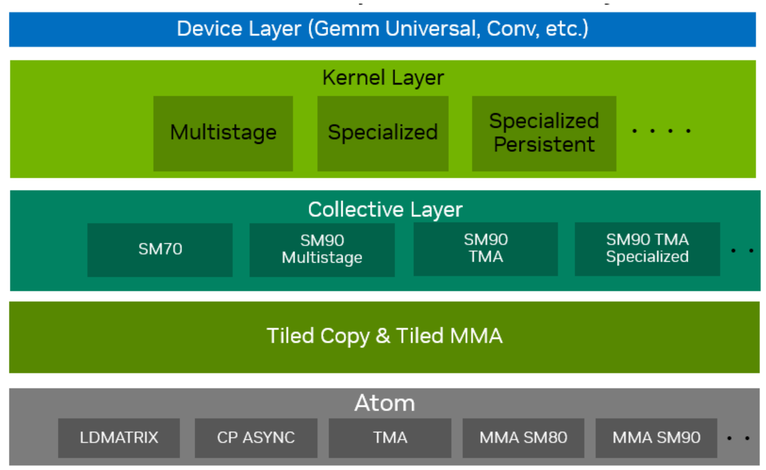
Conclusion

CUTLASS

- **CUTLASS 3.5 and CuTe**
  - Continuing to provide flexible abstractions for users to compose customized kernels and collectives
  - Introducing a generic approach to convolutions
  - Implement new kernels targeting LLMs
  - Promote flexible kernel fusion via the epilogue visitor tree abstraction



- **CUTLASS Roadmap**
  - CUTLASS 3.5 was released March 2024
  - CUTLASS 3.6 scheduled for summer 2024
    - Hopper sparsity support



- **Need help or have questions?**
  - Raise an issue: https://github.com/NVIDIA/cutlass/issues
  - Contact Matthew Nicely (mnicely@nvidia.com)

https://github.com/NVIDIA/cutlass/

Roadmap

Subject to change

Q1 - 2024	Q2 - 2024	Q3 - 2024	Q4 - 2024
• Ada FP8 support • Grouped GEMM (Hopper) • Mixed Input GEMMs • PyPI Wheels • Ptr-Array support	• CONV (Fprop, D/Wgrad) • Grouped GEMM optimizations	• CONV optimizations • Sparsity Support w/ Python interface • More EVT support • Separable/Depthwise CONV	• Documentation Refresh • Conda packaging

Acknowledgements

CUTLASS GitHub Community

4.3K stars, 2.5M clones/month, 100+ contributors, and many active users.
Many contributions and PRs from outside of NVIDIA
Integrated into PyTorch, Oneflow, TVM, PaddlePaddle, AITemplate, PyG, and 300 others GitHub projects

CUTLASS Product Management

Matthew Nicely (mnicely@nvidia.com)

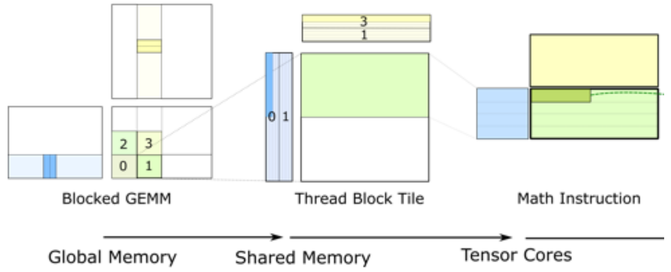
CUTLASS and CuTe Developers

See CONTRIBUTORS.md

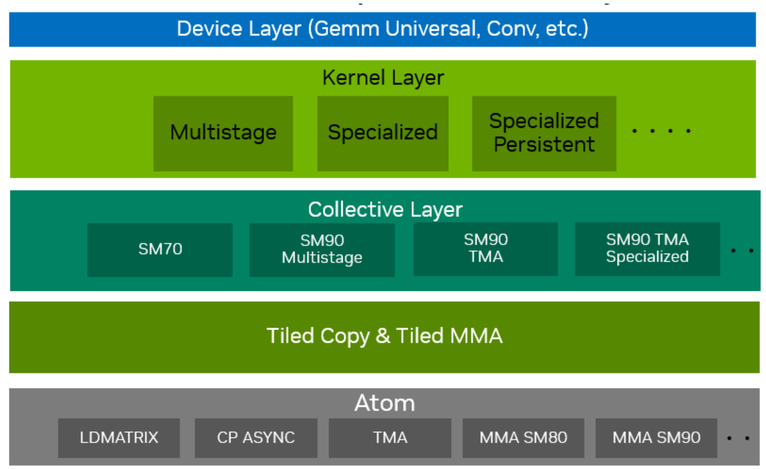
Conclusion

CUTLASS

- **CUTLASS 3.5 and CuTe**
  - Continuing to provide flexible abstractions for users to compose customized kernels and collectives
  - Introducing a generic approach to convolutions
  - Implement new kernels targeting LLMs
  - Promote flexible kernel fusion via the epilogue visitor tree abstraction



- **CUTLASS Roadmap**
  - CUTLASS 3.5 was released March 2024
  - CUTLASS 3.6 scheduled for summer 2024
    - Hopper sparsity support



- **Need help or have questions?**
  - Raise an issue: https://github.com/NVIDIA/cutlass/issues
  - Contact Matthew Nicely (mnicely@nvidia.com)

https://github.com/NVIDIA/cutlass/