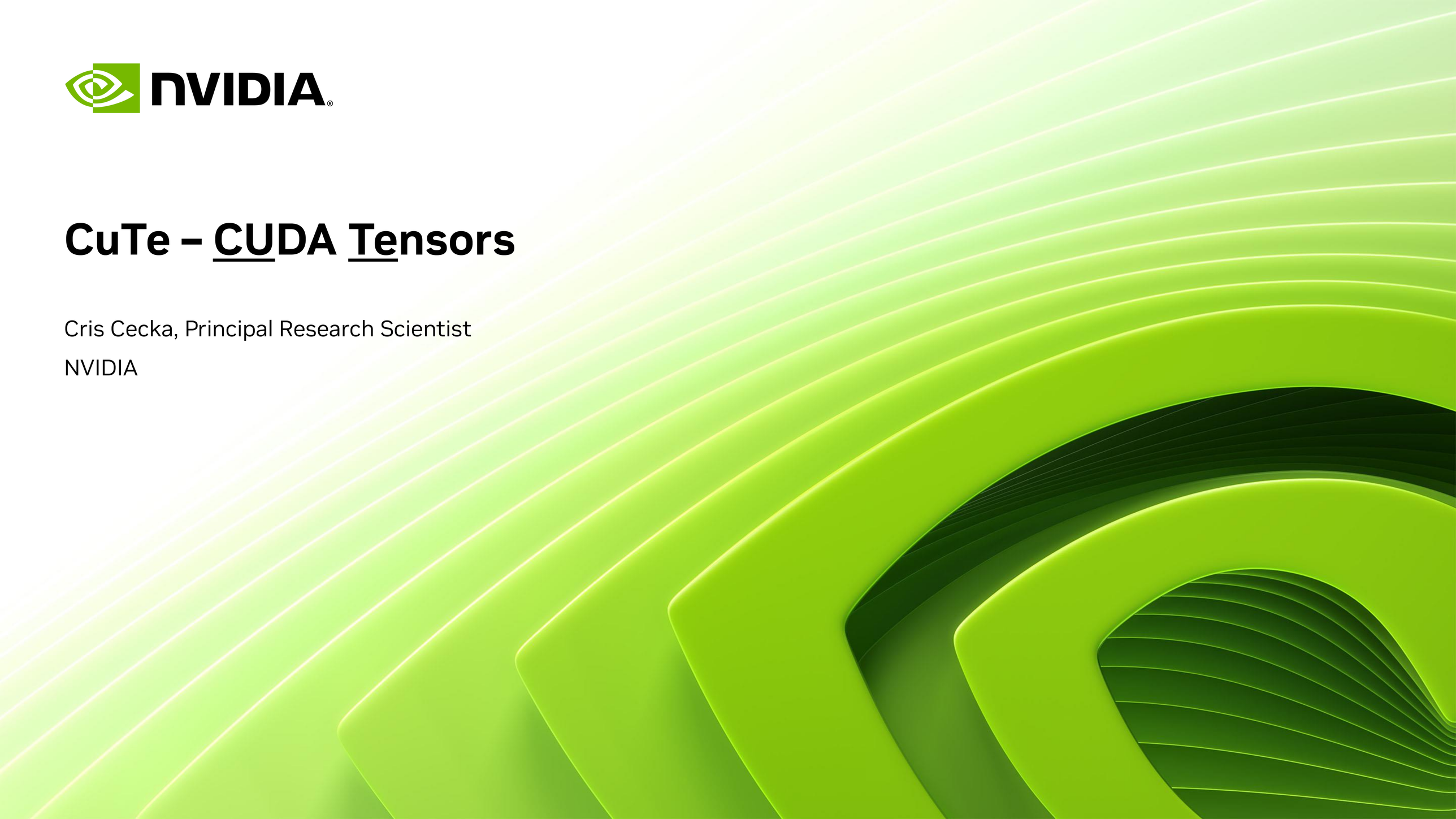


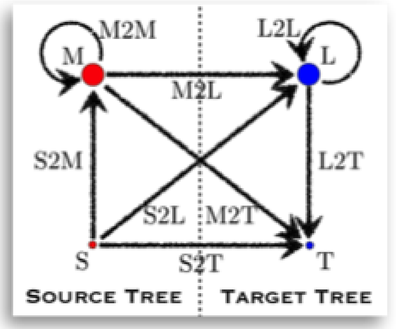
CuTe – CUDA Tensors

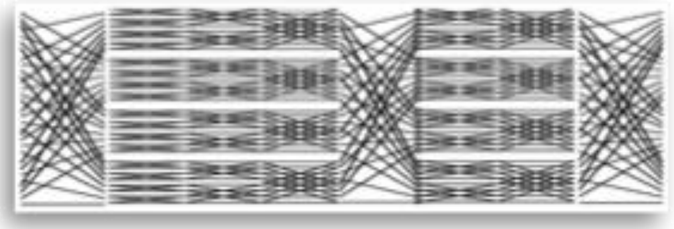
Cris Cecka, Principal Research Scientist

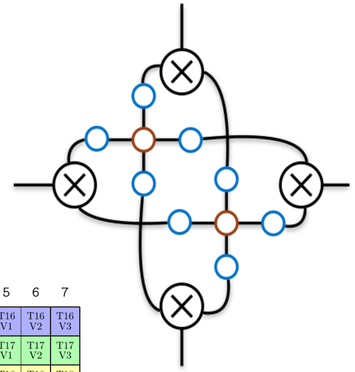
NVIDIA

CuTe History

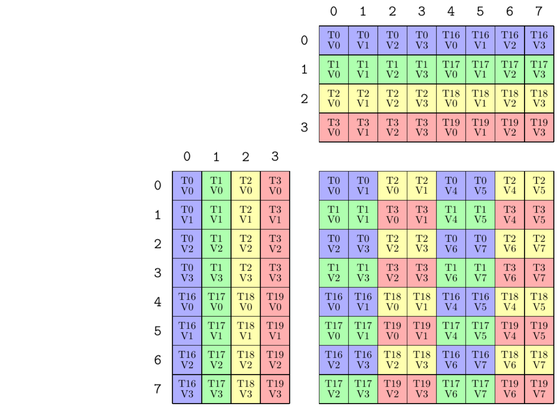
CuTe baby and CuTe maturity

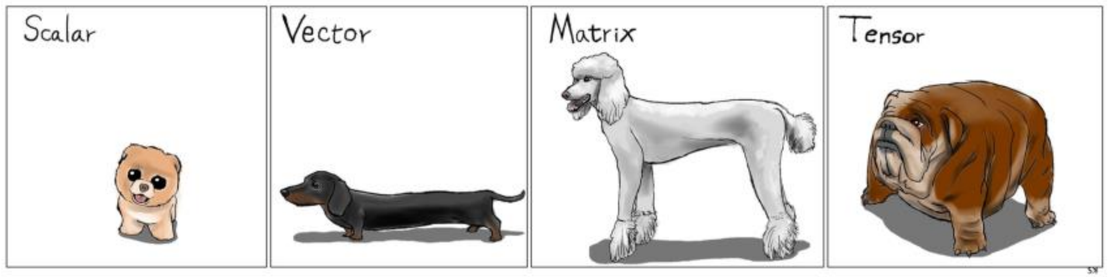






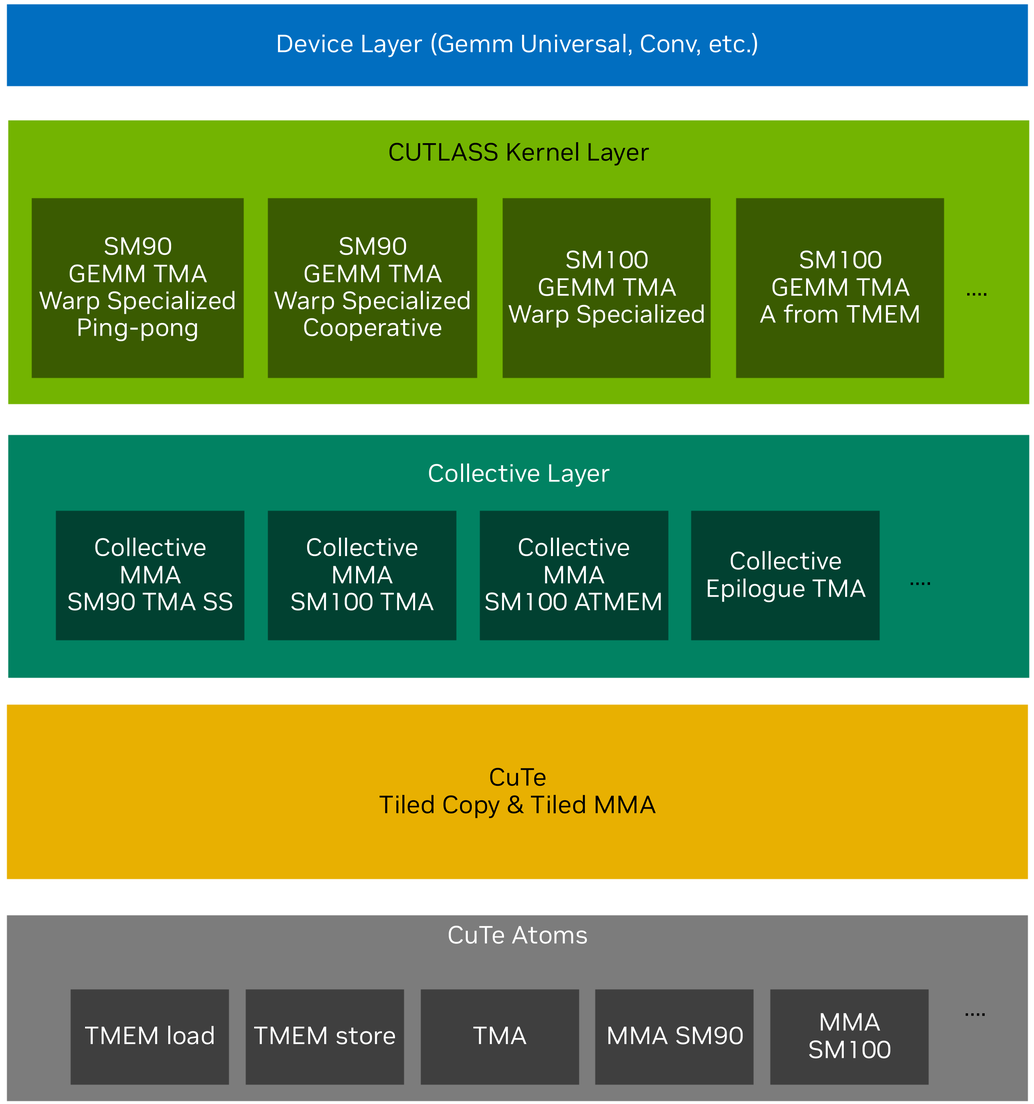
- FFT Research
  - BLAS extensions for tensor contractions
- Low-rank ML research
  - "I need more tensor contractions, tensor contractions are just GEMMs, why is this hard?"
- cuTensor collaboration – Paul Springer
  - CUTLASS 2.x pain
- "Oh, this shouldn't be hard."
  - Volta, Ampere, Hopper
  - StreamK – Mohammad Osama
  - Tensor cores
  - SoL GETT
- CUTLASS 3.x -- Vijay Thakkar
  - Hopper, Blackwell
  - TMA

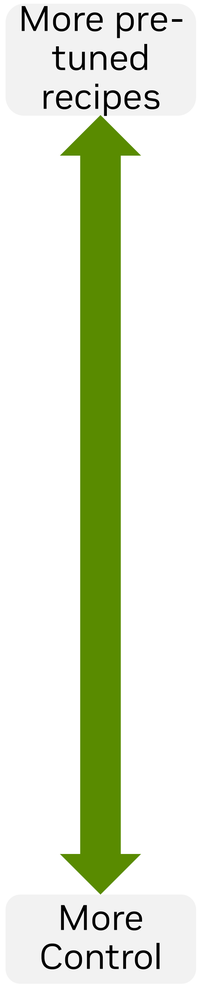




CUTLASS

Abstractions for productivity and performance at all scopes and scales





- Open source https://github.com/NVIDIA/cutlass
- Presented: GTC'18, GTC'19, GTC'20, GTC'21, GTC'22, GTC'22, GTC'23, GTC'24, GTC'25
- Multiple entry points depending on your needs

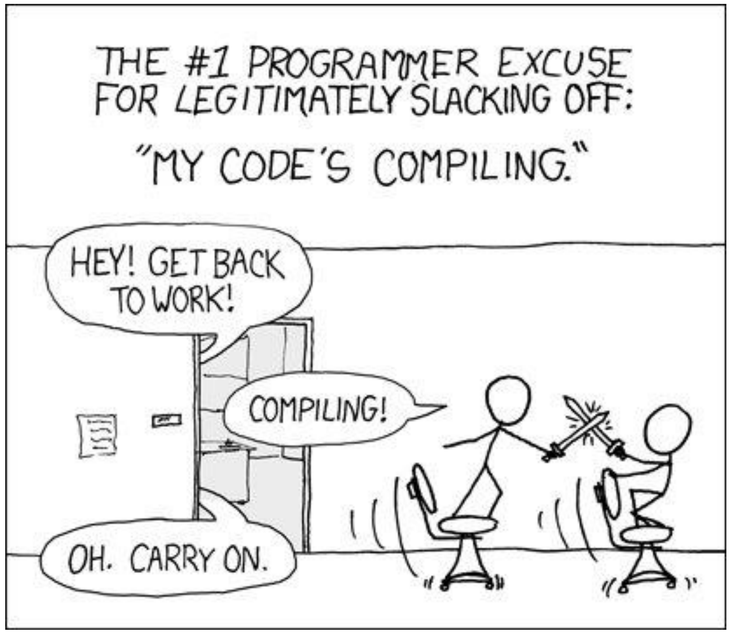
- CuTe examples
  - Bare naked GEMM+GETT kernels.
  - Using MMA, TMA, TMEM.
- CUTLASS examples
  - GETT, CONV, FastAttention, FMHA, GroupedGEMM.
  - FP4, StreamK, Epilogues, Distributed, Optimizations.

- 7500 GitHub stars.
- 180 GitHub contributors.
- 4.2M downloads per month.
- The most popular NVIDIA open source project.

Major pain points with C++

C++ templates and unfortunate consequences

- C++ templates are inconvenient
  - Additional mental load when writing compile-time logic
  - Error messages are longer than novels
- C++ templates suffer from slow compilation time
  - Front-end too generic for our purposes
  - Prevents fast iteration
  - Prohibits JIT-ting at scale and brute force auto-tuning
- The DL space fully embraces the python ecosystem regardless
  - Everybody hates writing binding code
  - Dependency on nvcc
- LLMs are likely better at generating python programs



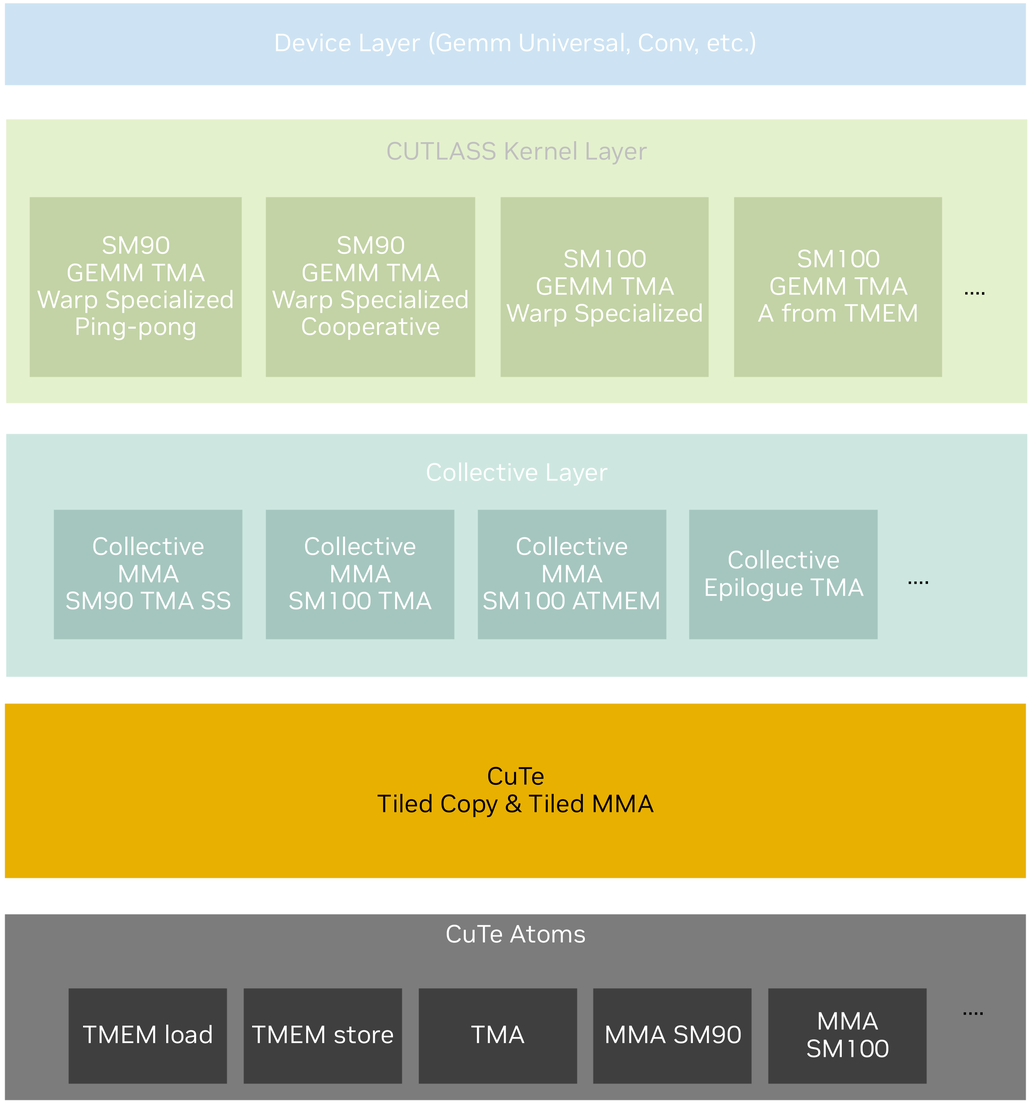
Do I have to tolerate all of this to use CUTLASS?

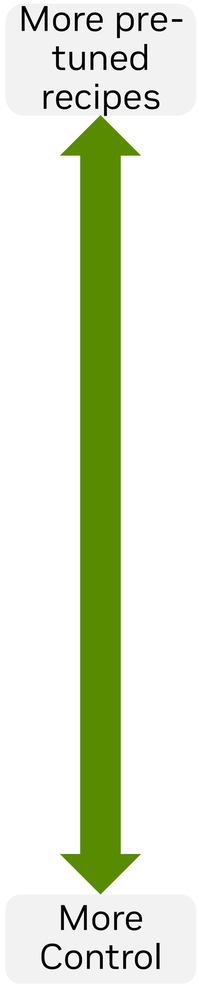
Introducing CUTLASS 4.0

Tensor core programming in Python

CUTLASS in Python

Initial launch to include CuTe





This first release makes available a ***mature*** low-level tensor programming model, giving access to tensor cores with full control.

More to come later...

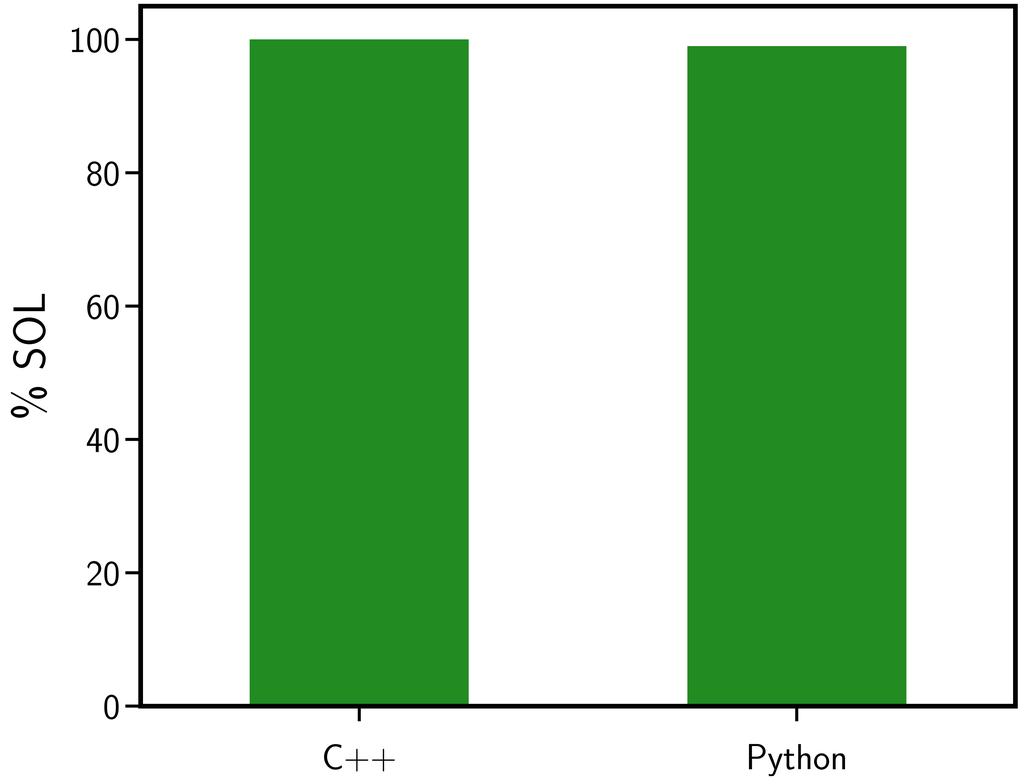
- `make_shape(Int<1>{}, Int<2>{}, x)` → `(1,2,x)`
- `make_layout`
- `make_identity_tensor`
- `zipped_divide`
- `local_tile`
- `tiled_mma.get_slice`
- `thr_mma.partition_A`
- `thr_copy.partition_S`

CUTLASS in Python

What do you get?

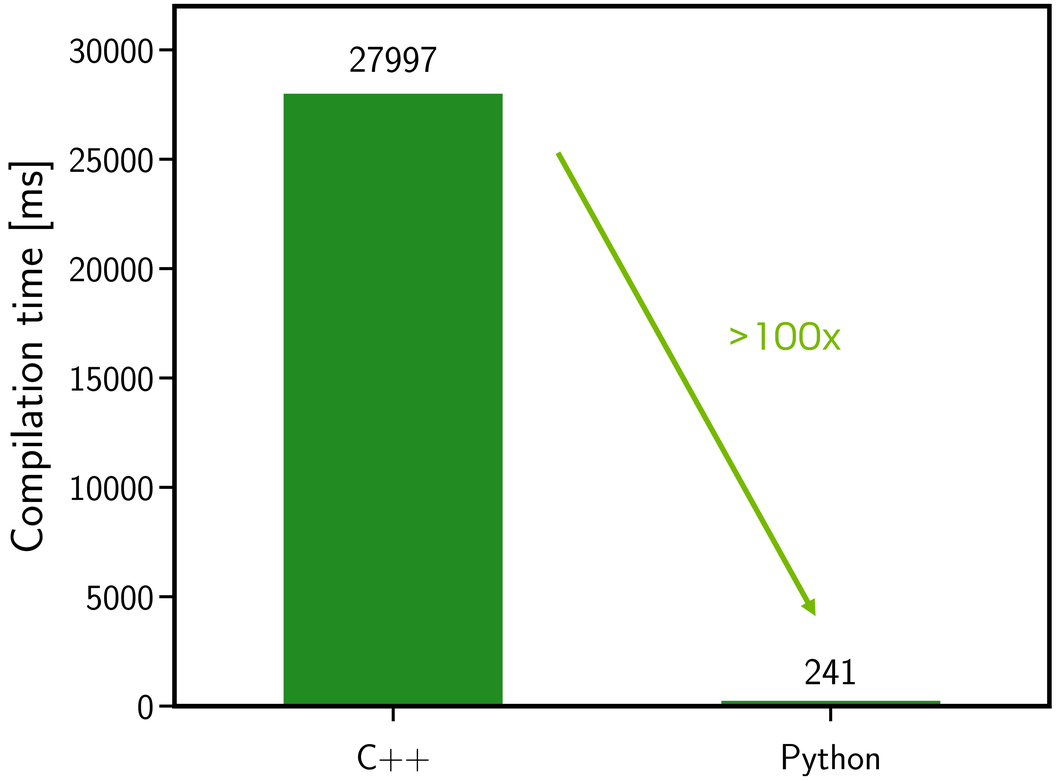
8kx8kx8k GEMM

Peak Performance!



Higher is better

Blazing Fast Compilation Time!



Lower is better

CUTLASS in Python

How will you get started?

```
pip install nvidia-cutlass-dsl
```

```python
import cutlass
import cutlass.cute as cute

@cute.kernel
def kernel():
  tidx, _, _ = cutlass.arch.thread_idx()
  if tidx == 0:
    cute.print_("Hello world")

@cute.jit
def host():
  cutlass.cuda.initialize_cuda_context()
  kernel().launch(
     grid=(1,1,1),
     block=(32,1,1))

host()
```

```
python3 hello_world.py
```

CuTe in a Slide

Representation + Algebra

- CuTe Layouts are a **representation**
  - Data layouts beyond row-major and col-major.
  - Generic algorithms like `copy`, `gemm`, `reduce`.
  - Encapsulation of metadata.

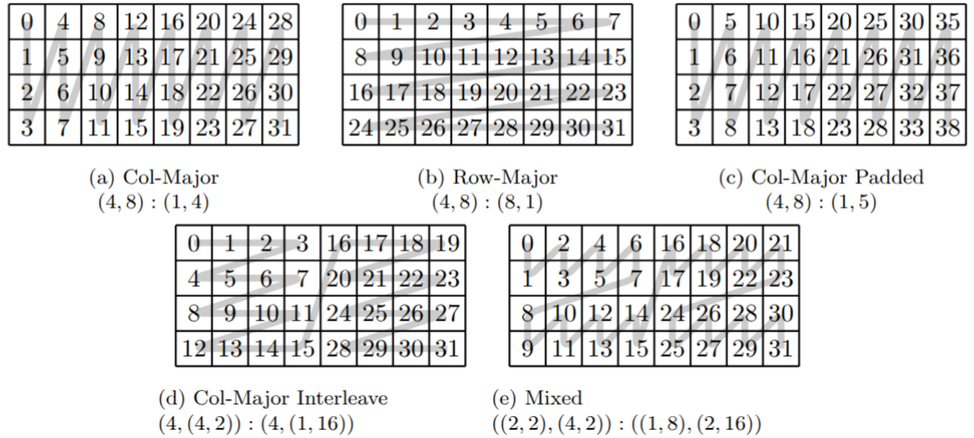


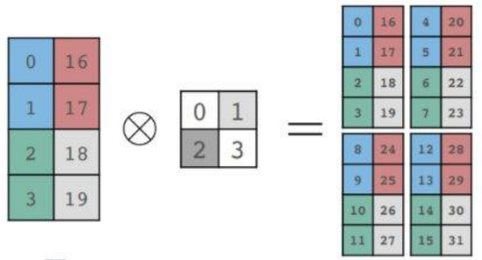
Figure 1: Examples of layouts compatible with shape $(4, 8)$ plotted with two dimensional coordinates.

- CuTe Layouts are an **algebra**
  - Combine Layouts to create new Layouts.
  - Generic tiling, partitioning.

Logical product:

$$f_A \otimes g_B = (f_A, f_A^* \circ g_B) \to (f_A, h_{B'})$$

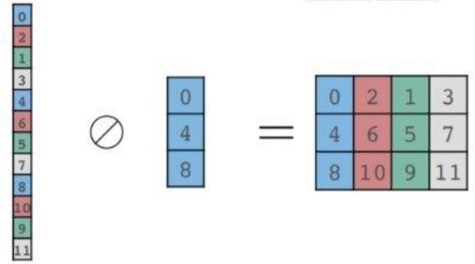
"Produce a layout where every element of layout B is a layout A."



Logical divide:

$$f_A \oslash g_B = f_A \circ (g_B, g_B^*) \to (h_{B'}, \ell_C)$$

"Produce a layout of Bs from a layout A."



Introduction

- Basics
- Motivation
- Example Layouts
- CUTLASS

Loop Philosophy

Canonical form

- Consider the loop

```
for (int i = 2; i <= 50; i += 3) {
  A[7*i + 5] = 0;
}
```

- for-start: **2**
- for-end: **50** (incl)
- for-step: **3**
- base-ptr: **A**
- affine idx transform: **7*i+5**

which can be rewritten as

```
for (int i = 0; i < 17; ++i) {
  (A+14+5)[3*7*i] = 0;
}
```

- Base-ptr: **A+19**
- Size: **17**
- Stride: **21**

Loop Philosophy

Canonical form

- Consider the loop

```
for (int j = 3; j < 43; j += 2) {
  for (int i = 4; i <= 20; i += 5) {
    A[10*i - j + 1] = 0;
  }
}
```

which can be rewritten as

```
for (int j = 0; j < 20; ++j) {
  for (int i = 0; i < 4; ++i) {
    (A+4*10-3+1)[5*10*i - 2*j] = 0;
  }
}
```

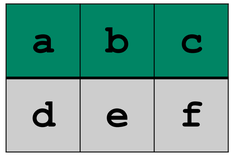
- for-start-j: **3**
- for-end-j: **43** (excl)
- for-step-j: **2**
- for-start-i: **4**
- for-end-i: **20** (incl)
- for-step-i: **5**
- base-ptr: **A**
- affine idx transform: **10*i-j+1**

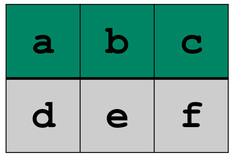
- Base-ptr: **A+38**
- Shape: **(4, 20)**
- Stride: **(50,-2)**

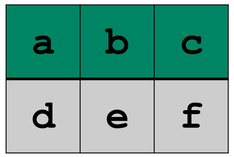
Layout Representation

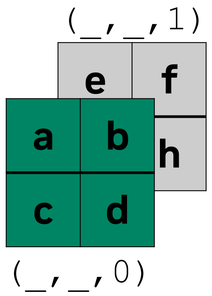
Function from Coordinate to Index

Logical









Shape: (2,3)
Stride: (1,2)

Column-major

Shape: (2,3)
Stride: (3,1)

Row-major

Shape: (2,3)
Stride: (1,4)

Padded Col-major

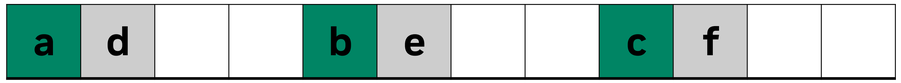
Shape: (2,2,2)
Stride: (4,1,2)

Tensor layout

Physical


```
idx = i*1 + j*2
```


```
idx = i*3 + j*1
```




```
idx = i*1 + j*4
```


```
idx = inner_product(coord,stride)
```

Introduction

Basics

```
int n_rows = 22;
int n_cols = 19;
thrust::host_vector<float> storage(n_rows * n_cols);

Layout layout = make_layout(make_shape(n_rows, n_cols));
Tensor matrix = make_tensor(storage.data(), layout);

static_assert(rank(matrix) == 2);

for (int i = 0; i < size<0>(matrix); ++i) {
  for (int j = 0; j < size<1>(matrix); ++j) {
    matrix(i,j) = i * stride<0>(matrix) + j * stride<1>(matrix);
  }
}
```

`Tensor` is a view, not a container

`rank`: number of modes

`size`: extent of a mode

`stride`: elements between consecutive elements of a mode

print(matrix)

```
ptr[32b](0x559ecbf182b0) o (22,19):(_1,22)
```

Introduction

Basics

```
auto n_rows = Int<22>{};
int  n_cols = 19;
thrust::host_vector<float> storage(n_rows * n_cols);

Layout layout = make_layout(make_shape(n_rows, n_cols));
Tensor matrix = make_tensor(storage.data(), layout);

static_assert(rank(matrix) == 2);
static_assert(size<0>(matrix) == 22);

for (int i = 0; i < size<0>(matrix); ++i) {
  for (int j = 0; j < size<1>(matrix); ++j) {
    matrix(i,j) = i * stride<0>(matrix) + j * stride<1>(matrix);
  }
}
```

Static integers as extents:
`cute::Int<N>{}`,
`cute::C<N>{}`, **or**
`N_c`

`size<0>` **now known at compile time**

print(matrix)

```
ptr[32b](0x559ecbf182b0) o (_22,19):(_1,_22)
```

Introduction

Basics

```
auto n_rows = Int<22>{};
int  n_cols = 19;
int  d_rows = 47;
auto d_cols = Int<2>{};
thrust::host_vector<float> storage(n_rows * d_row + n_cols * d_col);

Layout layout = make_layout(make_shape (n_rows, n_cols),
                            make_stride(d_rows, d_cols));
Tensor matrix = make_tensor(storage.data(), layout);

static_assert(rank(matrix) == 2);
static_assert(size<0>(matrix) == 22);
static_assert(stride<0>(matrix) == 2);

for (int i = 0; i < size<0>(matrix); ++i) {
  for (int j = 0; j < size<1>(matrix); ++j) {
    matrix(i,j) = i * stride<0>(matrix) + j * stride<1>(matrix);
  }
}
```

print(matrix)

```
ptr[32b](0x559ecbf182b0) o (_22,19):(47,_2)
```

Static integers as strides:

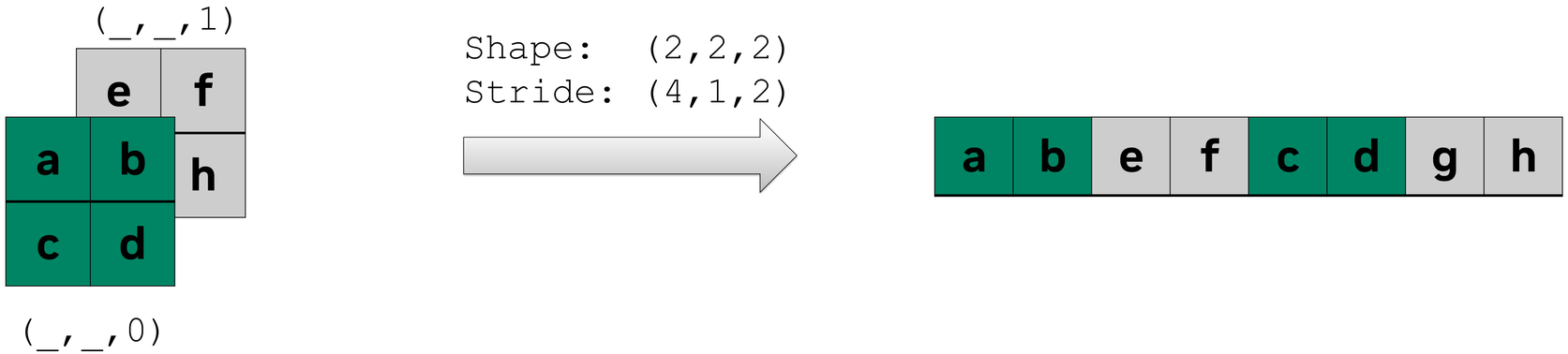
`cute::Int<2>{}`,
`cute::C<2>{}`, **or**
`2_c`

`make_stride(…)`	**-- custom strides**
`LayoutLeft{}`	**-- "Col"-major (default)**
`LayoutRight{}`	**-- "Row"-major**

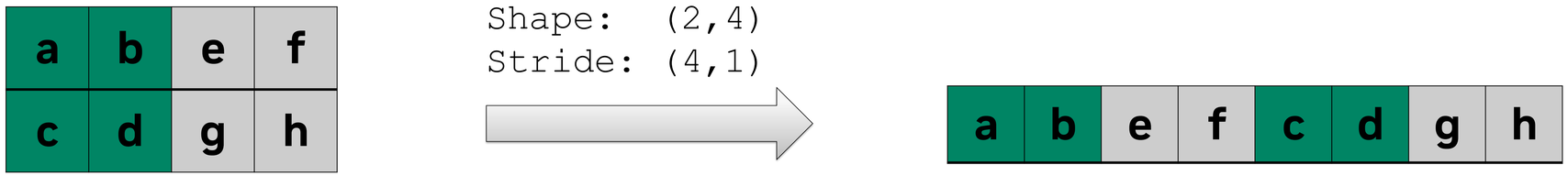
Layout Representation

Logical and Physical

Consider the 2x2x2 tensor



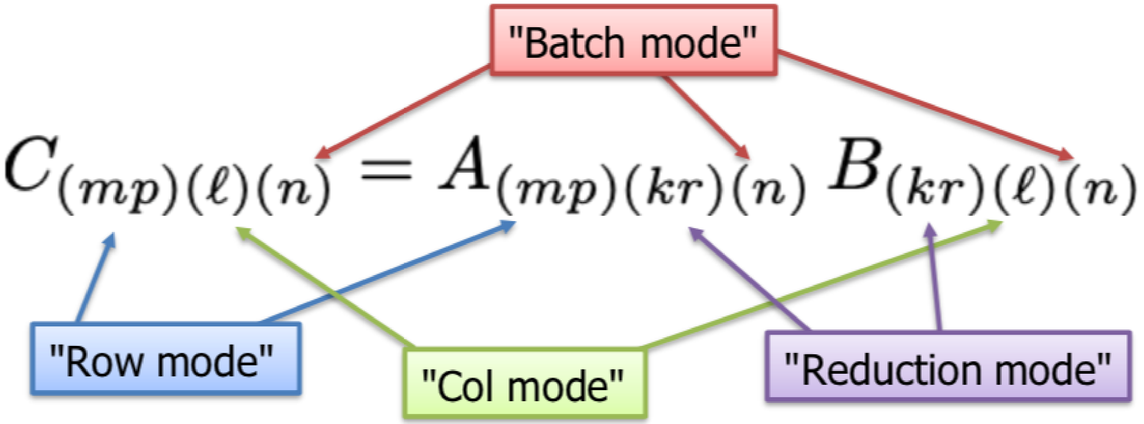
which can be folded and viewed as a 2x4 matrix



General Tensor Folding

All GETTs are GEMMs

$$C_{mn\ell p} = A_{mnkpr}\, B_{\ell nrk} \iff C_{(mp)(\ell)(n)} = A_{(mp)(kr)(n)}\, B_{(kr)(\ell)(n)}$$



All tensor contractions map to a canonical contraction: Batched-GEMM

- By grouping modes by type and defining **multi-modes**:
  - **m-modes** or “row modes” appear in C & A & !B
  - **n-modes** or “column modes” appear in C & !A & B
  - **k-modes** or “reduction modes” appear in !C & A & B
  - **p-modes** or “batch modes” appear in C & A & B

$$C_{\hat{m}\hat{n}\hat{p}} = A_{\hat{m}\hat{k}\hat{p}}\, B_{\hat{k}\hat{n}\hat{p}}$$

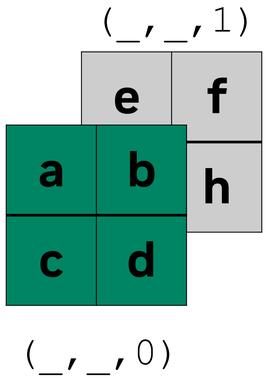
Shi, Niranjan, Anandkumar, **Cecka**. *Tensor Contractions with Extended BLAS Kernels on CPU and GPU*, IEEE 2016.

Cecka. *Low communication FMM-accelerated FFT on GPUs*, SuperComputing 2017.

Layout Representation

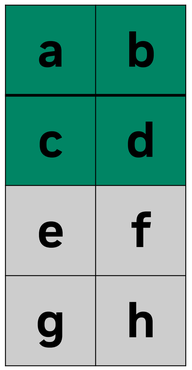
Logical and Physical

Consider the 2x2x2 tensor




```
Shape:  (2,2,2)
Stride: (4,1,2)
```

which can be folded and viewed as a 4x2 matrix…?

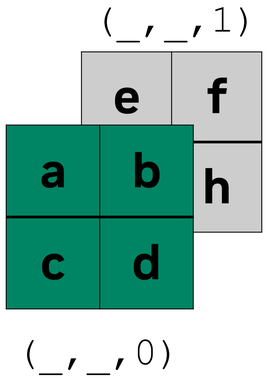



```
Shape:  (4,2)
Stride: (?,1)
```

Layout Representation

Logical and Physical

Consider the 2x2x2 tensor




```
Shape:  (2,2,2)
Stride: (4,1,2)
```

which can be folded and viewed as a 4x2 matrix with a nested layout

	0	1
(0,0)	a	b
(1,0)	c	d
(0,1)	e	f
(1,1)	g	h

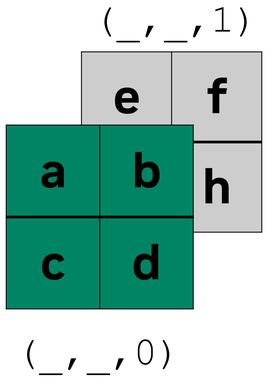
```
Shape:  ((2,2),2)
Stride: ((4,2),1)
```


```
idx = inner_product(coord,stride)
```

Layout Representation

Logical and Physical

Consider the 2x2x2 tensor




```
Shape:  (2,2,2)
Stride: (4,1,2)
```

which can be folded and viewed as a 4x2 matrix with a nested layout

	0	1
0 -> (0,0)	a	b
1 -> (1,0)	c	d
2 -> (0,1)	e	f
3 -> (1,1)	g	h

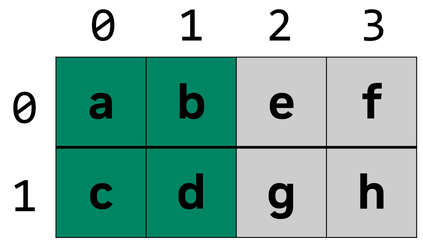
```
Shape:  ((2,2),2)
Stride: ((4,2),1)
```


```
idx = inner_product(coord,stride)
```

Layout Representation

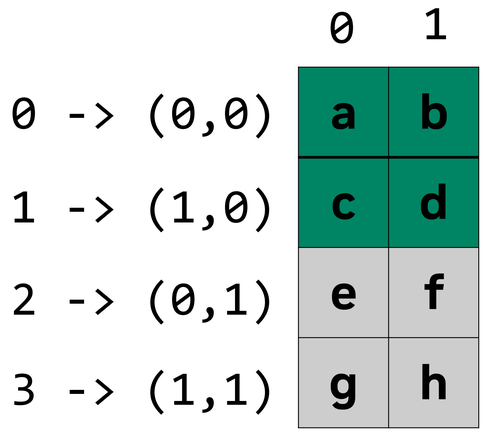
Logical and Physical

The 2x4 view




```
Shape:  (2,4)
Stride: (4,1)
```

The 4x2 view



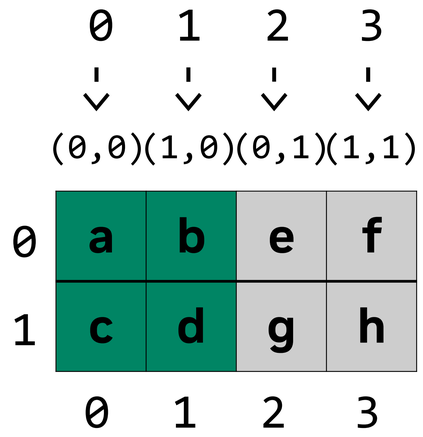

```
Shape:  ((2,2),2)
Stride: ((4,2),1)
```


```
idx = inner_product(coord,stride)
```

Layout Representation

Logical and Physical

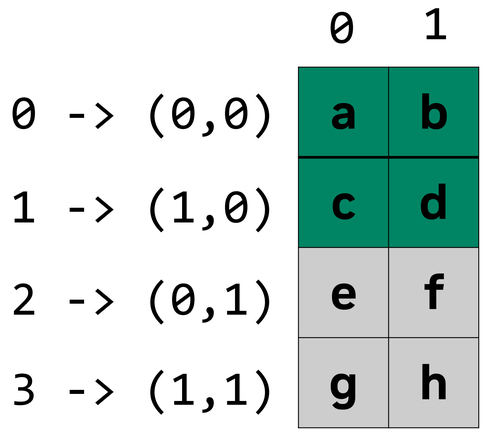
The 2x4 view




```
Shape:  (2,(2,2))
Stride: (4,(1,2))
```

```
Shape:  (2,4)
Stride: (4,1)
```

The 4x2 view




```
Shape:  ((2,2),2)
Stride: ((4,2),1)
```


```
idx = inner_product(coord,stride)
```

Types and Concepts

Representation and Algebra

Shape

- Concept **HTuple<Int>**: **Int** or **Tuple of HTuple<Int>**.

```
N   (N,M)   (N,M,P)   ((N0,N1),M)   ((N0,N1),(M0,(M10,M11,M12)))
```

Stride

- Concept **HTuple<D>**: **D** or **Tuple of HTuple<D>**.
- **D**: Anything that supports inner-product with integers.
- Congruent with the Shape.

Layout<Shape,Stride>

- Mapping between logical coordinate(s) within shape and **D**.

$$I \Longleftrightarrow (i,j) \Longleftrightarrow (i,(j_0,j_1)) \longrightarrow [k]$$

Tensor<Ptr,Layout>

- Composition of Layout with underlying storage.

```
T[], T*, smem_ptr<T>, gmem_ptr<T>, counting_iterator, transform_iterator
```

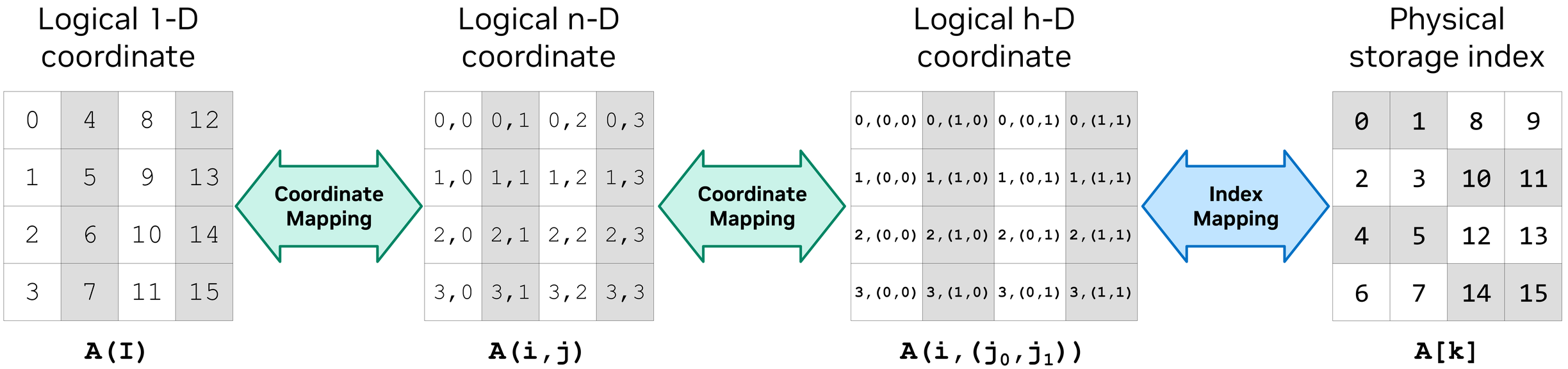
Algebra of Layouts:

- `concatenation(Layouts...)          -> Layout`
- `composition(LayoutA, LayoutB)      -> Layout`
- `right_inverse(Layout)              -> Layout`
- `left_inverse(Layout)               -> Layout`
- `complement(Layout, Shape)          -> Layout`
- `logical_product(LayoutA, LayoutB)  -> Layout`
- `logical_divide (LayoutA, LayoutB)  -> Layout`

Layout Mappings

Coordinates and Indices

```
Shape:  (4,(2,2))
Stride: (2,(1,8))
```



`Shape` defines the ***coordinate*** mappings

$$I \Leftrightarrow (i, j) \Leftrightarrow (i, (j_0, j_1))$$

`Stride` defines the ***index*** mapping

$$(i, (j_0, j_1)) \rightarrow [k]$$

Quick Layout Examples

Cover everything

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

(a) Col-Major
$(4, 8) : (1, 4)$

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

(b) Row-Major
$(4, 8) : (8, 1)$

0	5	10	15	20	25	30	35
1	6	11	16	21	26	31	36
2	7	12	17	22	27	32	37
3	8	13	18	23	28	33	38

(c) Col-Major Padded
$(4, 8) : (1, 5)$

0	1	2	3	16	17	18	19
4	5	6	7	20	21	22	23
8	9	10	11	24	25	26	27
12	13	14	15	28	29	30	31

(d) Col-Major Interleave
$(4, (4, 2)) : (4, (1, 16))$

0	2	4	6	16	18	20	21
1	3	5	7	17	19	22	23
8	10	12	14	24	26	28	30
9	11	13	15	25	27	29	31

(e) Mixed
$((2, 2), (4, 2)) : ((1, 8), (2, 16))$

Figure 1: Examples of layouts compatible with shape $(4, 8)$ plotted with two dimensional coordinates.

Hierarchical Layout

Example: Simple multi-mode

Shape: `(8,(2, 2))`

Stride: `(2,(1,16))`

i\j	0 (0,0)	1 (1,0)	2 (0,1)	3 (1,1)
0	0	1	16	17
1	2	3	18	19
2	4	5	20	21
3	6	7	22	23
4	8	9	24	25
5	10	11	26	27
6	12	13	28	29
7	14	15	30	31

```
Layout layout = make_layout(make_shape (8,make_shape (2, 2)),
                            make_stride(2,make_stride(1,16)));
Tensor A = make_tensor(counting_iterator{}, layout);
```

- Logical coordinates are 1D, 2D, hD

```
A(17)      = 18
A(1,2)     = 18
A(1,(0,1)) = 18
```

- Slice along logical sub-boundaries

```
A(3,_) = [6,7,22,23]

A(5,(_,1)) = [26,27]
```

Hierarchical Layout

Example: More multi-modes

Shape: `((2,(2, 2)),(2,(2, 2)))`

Stride: `((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	6	8	13	15	37	39	45	47
4	16	18	24	26	48	50	56	58
5	17	19	25	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{}, morton3);
```

- 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;25;28;29]
A((_,1),(_,2)) = [36,38;
                  37,39]
```

Layout Compatibility

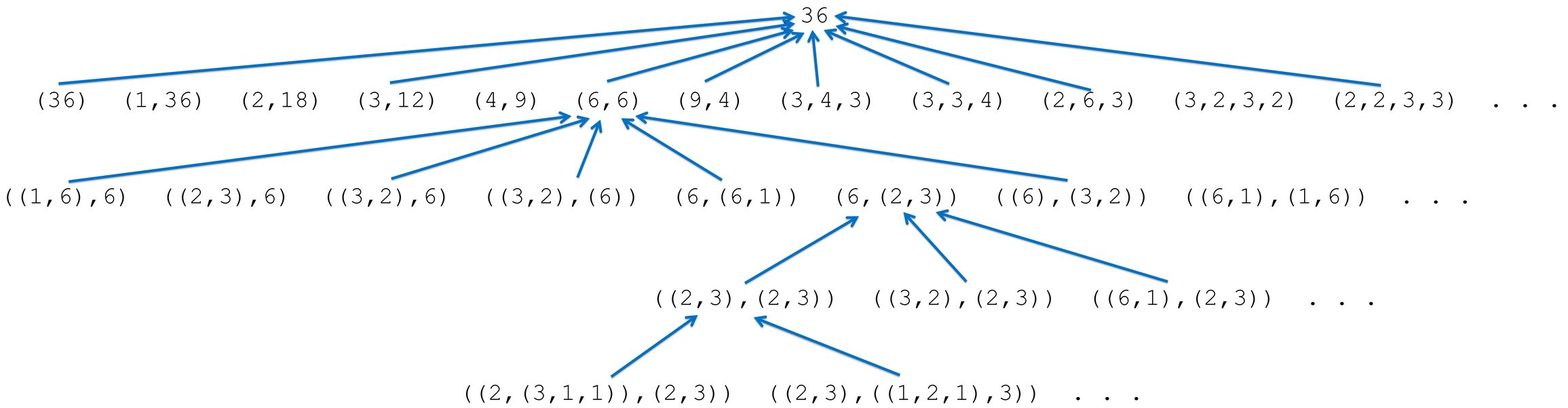
A poset of Shapes

Definition 3.4. We say that a shape A is *compatible* with shape B if

$$\mathbb{Z}(A) \subseteq \mathbb{Z}(B)$$

and write this as $A \preceq B$. If shape A is compatible with shape B, then all coordinate lists of A are also coordinate lists of B and, therefore, any coordinate of A can be used as a coordinate of B. Note that compatibility defines a partial order on the set of shapes.

Reflexivity: For all shapes A, $A \preceq A$.
Anti-symmetry: For all shapes A and B, if $A \preceq B$ and $B \preceq A$, then $A = B$.
Transitivity: For all shapes A and B and C, if $A \preceq B$ and $B \preceq C$, then $A \preceq C$.
Minimal Elements: For all integers n and all shapes S with $n = |S|$, we have $n \preceq S$.
Maximal Elements: If all elements of a shape P are prime, then there is no shape $A \neq P$ with $P \preceq A$.



CuTe CUTLASS

Why CuTe?

- Layout subsumes every iterator
  - A single impl. and vocab type
- Formal algebra to manipulate Layout
  - `concatenation`
  - `composition`
  - `complement`
  - `right_inverse`, `left_inverse`
  - “`product`”
  - “`divide`”
- Layout for both threads and data

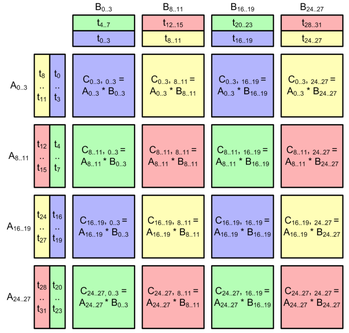


Figure 2a: Composing sparse A[16,4] * B[4, 16] = C[16, 16]
warp-wide multiply from four A[8, 4] * B[4, 8] = C[8, 8] 8-thread multiplies.

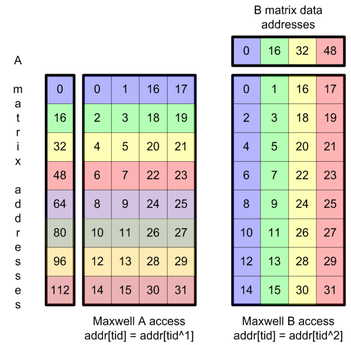
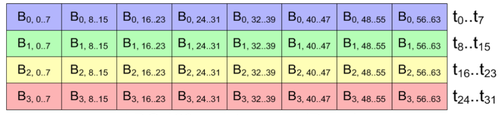
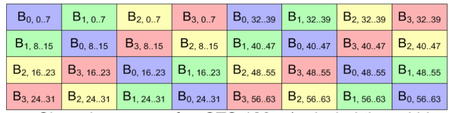


Figure 3b: Maxwell thread assignments for LDS.U matching



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])

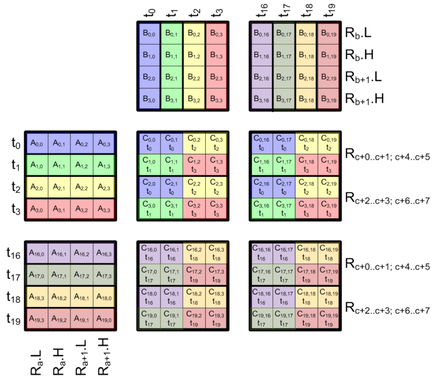
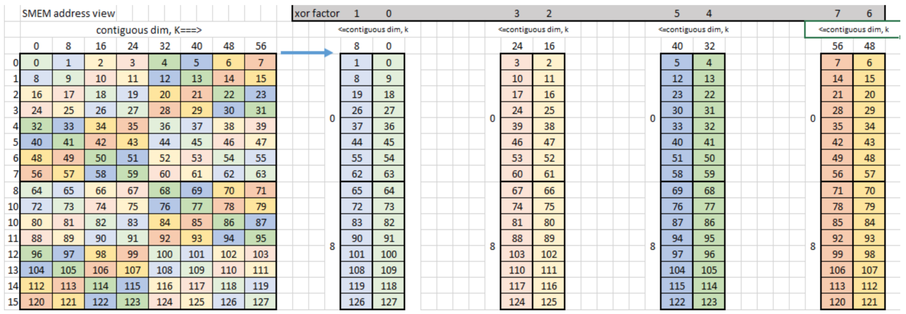


Figure 1b: HMMA.884.F32.TN data layout.
Again, ignore the apparent isomorphism
between thread numbers and matrix indices.



CUTLASS 2.x

```
RowMajor
ColMajor
RowMajorInterleaved
ColumnMajorInterleaved
PitchLinear
TensorNCHW
VoltaTensorOpMultiplicandCongruous
ColumnMajorVoltaTensorOpMultiplicandCongruous
RowMajorVoltaTensorOpMultiplicandCongruous
VoltaTensorOpMultiplicandBCongruous
ColumnMajorVoltaTensorOpMultiplicandBCongruous
RowMajorVoltaTensorOpMultiplicandBCongruous
VoltaTensorOpMultiplicandCrosswise
ColumnMajorVoltaTensorOpMultiplicandCrosswise
RowMajorVoltaTensorOpMultiplicandCrosswise
Many, many more...
```

CuTe + CUTLASS 3.x

```
Layout<Shape,Stride>
```

tridao commented 3 weeks ago — Author

Amazing! Thanks a lot for the explanation @thakkarV.
I've always found smem swizzling to be the trickiest part of writing a fast gemm. It's crazy that you can replace thousands of lines of smem swizzling code with a few lines of well-designed and composable abstractions.

I'm excited to start hacking on CuTe!

❤️ 1

Wild Indexing

Iterators vs Layouts

- CUTLASS 2.x implements custom layout functors
  - Mapping logical coord -> index
- Thread/Data layouts fixed and implicit in implementations
- Indexing is **hard** to implement, **impossible** to maintain
- Composability of layouts suffers
  - Named types for each layout
  - Rigid interfaces and specific targets

```
auto swizzle_atom    = composition(
       Swizzle<3,3,3>{},
       Layout<Shape <Shape <_2, _4, _2>,Shape <_8, _2>>,
              Stride<Stride<_8,_64,_32>,Stride<_1,_16>>>{});

auto swizzle_layout = tile_to_shape(swizzle_atom, Shape<_128,_64>{});
```

```
/// Returns the offset of a coordinate in linear memory.
CUTLASS_HOST_DEVICE
LongIndex operator()(TensorCoord const &coord) const {
    int vec_contiguous_idx = coord.contiguous() / kElementsPerAccess;
    int vec_strided_idx = coord.strided() / kFactor;

    int tile_contiguous_idx =
        vec_contiguous_idx / (TileShape::kContiguous / kFactor);
    int tile_contiguous_residual =
        vec_contiguous_idx % (TileShape::kContiguous / kFactor) +
        ((coord.strided() % kFactor) * (TileShape::kContiguous / kFactor));
    int tile_strided_residual = vec_strided_idx % TileShape::kStrided;

    int partition_contiguous_idx =
        tile_contiguous_residual / PartitionShape::kContiguous;
    int partition_strided_idx =
        tile_strided_residual / PartitionShape::kStrided;
    int partition_contiguous_residual =
        tile_contiguous_residual % PartitionShape::kContiguous;
    int partition_strided_residual =
        tile_strided_residual % PartitionShape::kStrided;

    int permuted_vec_contiguous_within_partition =
        partition_contiguous_residual ^ (partition_strided_residual % 4);
    int permuted_partition_contiguous_within_tile =
        partition_contiguous_idx ^ (partition_strided_idx % 2);

    int element_contiguous = (tile_contiguous_idx * TileShape::kContiguous +
                              permuted_partition_contiguous_within_tile *
                                  PartitionShape::kContiguous +
                              permuted_vec_contiguous_within_partition) *
                                 kElementsPerAccess +
                             (coord.contiguous() % kElementsPerAccess);

    int element_strided = vec_strided_idx;
    return element_contiguous + element_strided * stride_[0] * kFactor;
}
```

Wild Indexing

Iterators vs Layouts

- CUTLASS 2.x implements custom layout functors
  - Mapping logical coord -> index
- Thread/Data layouts fixed and implicit in implementations
- Indexing is **hard** to implement, **impossible** to maintain
- Composability of layouts suffers
  - Named types for each layout
  - Rigid interfaces and specific targets

```
auto swizzle_atom   = composition(
      Swizzle<3,3,3>{},
      Layout<Shape <Shape <_2, _4, _2>,Shape <_8, _2>>,
             Stride<Stride<_8,_64,_32>,Stride<_1,_16>>>{});

auto swizzle_layout = tile_to_shape(swizzle_atom, Shape<_128,_64>{});

print_latex(swizzle_atom);
```

```
/// Returns the offset of a coordinate in linear memory.
CUTLASS_HOST_DEVICE
LongIndex operator()(TensorCoord const &coord) const {
    int vec_contiguous_idx = coord.contiguous() / kElementsPerAccess;
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	16	17	18	19	20	21	22	23
1	8	9	10	11	12	13	14	15	24	25	26	27	28	29	30	31
2	72	73	74	75	76	77	78	79	88	89	90	91	92	93	94	95
3	64	65	66	67	68	69	70	71	80	81	82	83	84	85	86	87
4	144	145	146	147	148	149	150	151	128	129	130	131	132	133	134	135
5	152	153	154	155	156	157	158	159	136	137	138	139	140	141	142	143
6	216	217	218	219	220	221	222	223	200	201	202	203	204	205	206	207
7	208	209	210	211	212	213	214	215	192	193	194	195	196	197	198	199
8	32	33	34	35	36	37	38	39	48	49	50	51	52	53	54	55
9	40	41	42	43	44	45	46	47	56	57	58	59	60	61	62	63
10	104	105	106	107	108	109	110	111	120	121	122	123	124	125	126	127
11	96	97	98	99	100	101	102	103	112	113	114	115	116	117	118	119
12	176	177	178	179	180	181	182	183	160	161	162	163	164	165	166	167
13	184	185	186	187	188	189	190	191	168	169	170	171	172	173	174	175
14	248	249	250	251	252	253	254	255	232	233	234	235	236	237	238	239
15	240	241	242	243	244	245	246	247	224	225	226	227	228	229	230	231

```
    int element_strided = vec_strided_idx;
    return element_contiguous + element_strided * stride_[0] * kFactor;
}
```

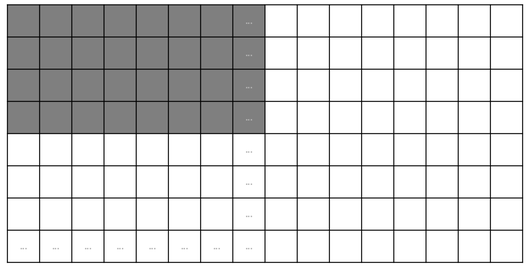
Algorithms

Generality of a Tile

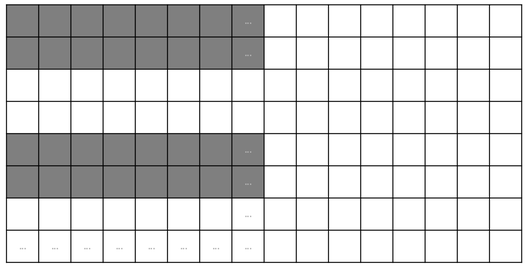
Motivation for Copy

Many ways to "tile" a *MxN tensor* into **4x8 subtensors**

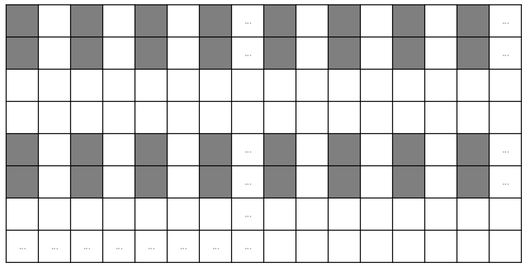
`<4,8>`
`<4:1,8:1>`



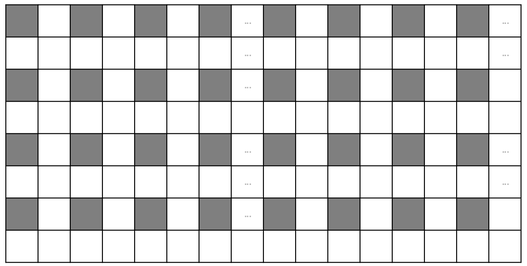
`<(2,2):(1,4),8:1>`



`<(2,2):(1,4),8:2>`



`<4:2,8:2>`



Many ways to construct a **4x8 layout**

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

(a) Col-Major
$(4, 8) : (1, 4)$

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

(b) Row-Major
$(4, 8) : (8, 1)$

0	5	10	15	20	25	30	35
1	6	11	16	21	26	31	36
2	7	12	17	22	27	32	37
3	8	13	18	23	28	33	38

(c) Col-Major Padded
$(4, 8) : (1, 5)$

0	2	4	6	16	18	20	21
1	3	5	7	17	19	22	23
8	10	12	14	24	26	28	30
9	11	13	15	25	27	29	31

(e) Mixed
$((2, 2), (4, 2)) : ((1, 8), (2, 16))$

One copy interface:

```
cute::copy(src_4x8_tensor, dst_4x8_tensor);
```

Copy

Generic Tensor Operations

```
template <class SrcEngine, class SrcLayout,
          class DstEngine, class DstLayout>
CUTE_HOST_DEVICE void
copy(Tensor<SrcEngine,SrcLayout> const& src,
     Tensor<DstEngine,DstLayout>      & dst)
{
}
```

Copy Basics

Rank-1

```
template <class SrcEngine, class SrcLayout,
          class DstEngine, class DstLayout>
CUTE_HOST_DEVICE void
copy(Tensor<SrcEngine,SrcLayout> const& src,
     Tensor<DstEngine,DstLayout>      & dst)
{
  for (int i = 0; i < size(dst); ++i) {
    dst(i) = src(i);
  }
}
```

Copy Basics

Rank-1

```
template <class SrcEngine, class SrcLayout,
          class DstEngine, class DstLayout>
CUTE_HOST_DEVICE void
copy(Tensor<SrcEngine,SrcLayout> const& src,
     Tensor<DstEngine,DstLayout>      & dst)
{
  for (int i = 0; i < size(dst); ++i) {
    dst(i) = src(i);
  }
}
```

- Copy is a rank-1 algorithm regardless of the argument tensors' rank.

- For *static shapes*, this is **optimal**
  - Very common in-kernel: rmem, smem, tmem, and tiles of gmem
  - Loop **unrolled**
  - Coordinate transform computed **statically**
  - Inner product to get physical offset often computed **statically**

Copy Basics

Rank-1

```
template <class SrcEngine, class SrcLayout,
          class DstEngine, class DstLayout>
CUTE_HOST_DEVICE void
copy(Tensor<SrcEngine,SrcLayout> const& src,
     Tensor<DstEngine,DstLayout>      & dst)
{
  for (int i = 0; i < size(dst); ++i) {
    dst(i) = src(i);
  }
}
```

Applications	Source	Destination
1D Arrays	8:1	8:1
ND Arrays	(8,2,3):(1,8,16)	(8,2,3):(1,8,16)
GATHER	(2,2,2):(42,1,128)	8:1
SCATTER	8:1	(2,2,2):(42,1,128)
BROADCAST	8:0	8:1
CONSTANT	8:0	8:0
TRANSPOSE	(8,3):(1,8)	(8,3):(3,1)

Gemm Basics

Rank-3

```
template <class AEngine, class ALayout,
          class BEngine, class BLayout,
          class CEngine, class CLayout>
CUTE_HOST_DEVICE constexpr void
gemm(Tensor<AEngine,ALayout> const& A,   // (M,K)
     Tensor<BEngine,BLayout> const& B,   // (N,K)
     Tensor<CEngine,CLayout>      & C)   // (M,N)
{
  for (int k = 0; k < size<1>(A); ++k) {
    for (int m = 0; m < size<0>(A); ++m) {
      for (int n = 0; n < size<0>(B); ++n) {
        C(m,n) += A(m,k) * B(n,k);
      }
    }
  }
}
```

Gemm Basics

Rank-3

```
template <class AEngine, class ALayout,
          class BEngine, class BLayout,
          class CEngine, class CLayout>
CUTE_HOST_DEVICE constexpr void
gemm(Tensor<AEngine,ALayout> const& A,   // (M,K)
     Tensor<BEngine,BLayout> const& B,   // (N,K)
     Tensor<CEngine,CLayout>      & C)   // (M,N)
{
  for (int k = 0; k < size<1>(A); ++k) {
    for (int m = 0; m < size<0>(A); ++m) {
      for (int n = 0; n < size<0>(B); ++n) {
        C(m,n) += A(m,k) * B(n,k);
      }
    }
```

Applications	A Layout	B Layout	C Layout
NT, TN, NN, TT GEMM	(M,K):(1,lda)	(N,K):(1,ldb)	(M,N):(1,ldc)
NTT GEMM	(M,K):(1,lda)	(N,K):(1,ldb)	(M,N):(ldc,1)
BLIS GEMM	(M,K):(dma,dka)	(N,K):(dnb,dkb)	(M,N):(dmc,dnc)
GETT	$((M_1,M_2),K)$	(N,K)	$((M_1,M_2),N)$
GETT	$(M,(K_1,K_2))$	$(N,(K_1,K_2))$	(M,N)
CONV	(K,(C,T,R,S))	((N,Z,P,Q),(C,T,R,S))	(K,(N,Z,P,Q))

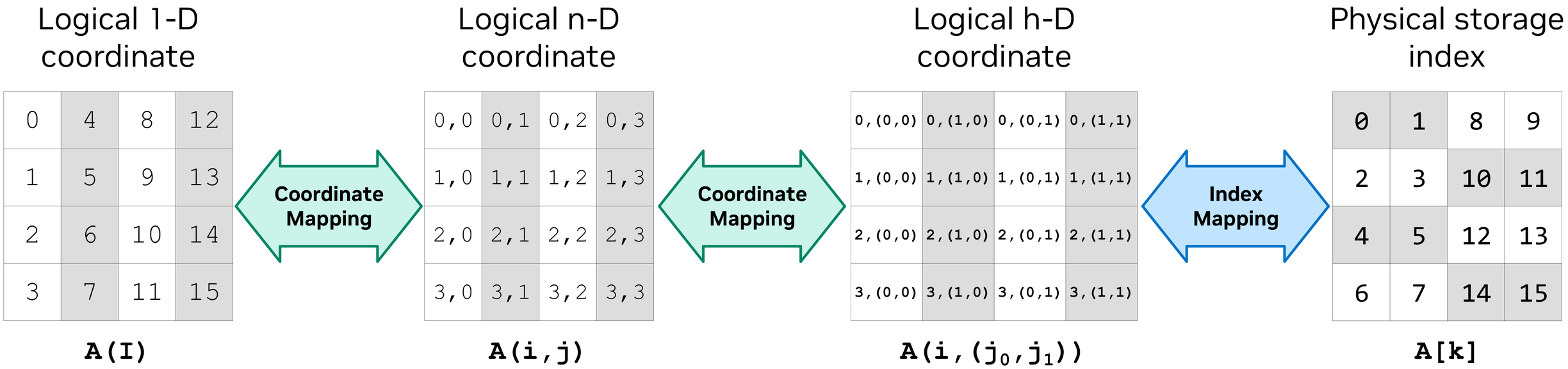
cute::composition

- Motivation
- Definition
- Applications

Layout Mappings

Coordinates and Indices

```
Shape:  (4,(2,2))
Stride: (2,(1,8))
```



`Shape` defines the ***coordinate*** mappings

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

`Stride` defines the ***index*** mapping

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

Layout Mappings

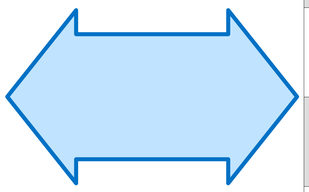
Coordinates and Indices

```
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

A(I)



Physical storage index

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

A[k]

Layout Composition

Algebra Construction

The composition of layouts $\mathbf{A}$ and $\mathbf{B}$ produces a layout $\mathbf{R}$

$$\mathbf{A} \circ \mathbf{B} \to \mathbf{R}$$

with

$$\mathbf{B} \preceq \mathbf{R}$$

such that

$$\forall c \in \mathbb{Z}(\mathbf{B}), \quad \mathbf{R}(c) = \mathbf{A}(\mathbf{B}(c))$$

- Layout **R** is *compatible* with Layout **B**.
  - All coordinates of Layout **B** can be used as coordinates of Layout **R**.
- Left and Right Identities $\mathbf{I} \circ \mathbf{B} = \mathbf{B} \qquad \mathbf{A} \circ \mathbf{I}_A = \mathbf{A}$
- Associativity $(\mathbf{A} \circ \mathbf{B}) \circ \mathbf{C} = \mathbf{A} \circ (\mathbf{B} \circ \mathbf{C})$
- Left Distributivity (B-surjective) $\mathbf{A} \circ \mathbf{B} = \mathbf{A} \circ (\mathbf{B}_1, \mathbf{B}_2) = (\mathbf{A} \circ \mathbf{B}_1, \mathbf{A} \circ \mathbf{B}_2)$

Compositional Power

Example

Compositional Power

Example

Compositional Power

Example

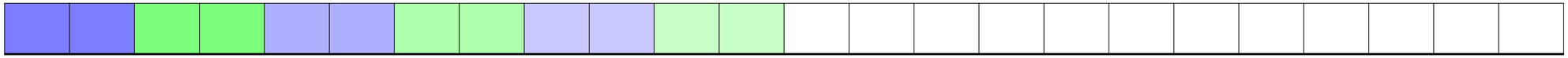
```
((2,3))
((1,4))
```

Values

0	1	4	5	8	9

Compositional Power

Example



```
((2,3))
((1,4))
```

Values

0	1	4	5	8	9
2	3	6	7	10	11

Compositional Power

Example

```
((2,3))
((1,4))
```

Values

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

Compositional Power

Example

((2,3))
((1,4))
Values

((2, 2))
((2,12))

Threads

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒

Thr **Val**
((2, 2),(2,3))
((2,12),(1,4))

A function from (thread_idx, value_idx) to 1D coord of array

Compositional Power

Example

((2,3))
((1,4))
Values

((2, 2))
((2,12))

Threads

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒

Thr Val
((2, 2),(2,3))
((2,12),(1,4))

A function from (thread_idx, value_idx) to 1D coord of array

Compositional Power

Example

((2,3))
((1,4))
Values

((2, 2))
((2,12))

Threads

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒

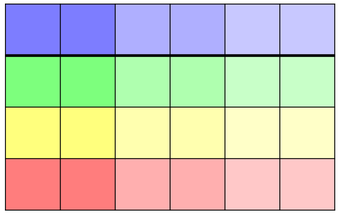
Thr Val
((2, 2),(2,3))
((2,12),(1,4))

A function from (thread_idx, value_idx) to 1D coord of array

∘

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒



Compositional Power

Example

((2,3))
((1,4))
Values

((2, 2))
((2,12))

Threads

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒

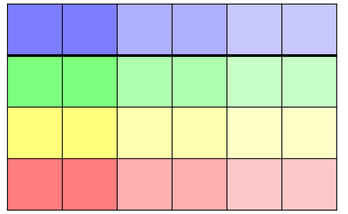
Thr **Val**
((2, 2),(2,3))
((2,12),(1,4))

A **function** from (**thread_idx**, **value_idx**) to 1D **coord** of array

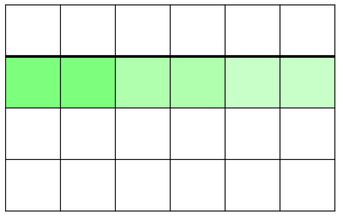
∘

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒



⇒



Compositional Power

Example

((2,3))
((1,4))
Values

Threads

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

((2, 2))
((2,12))

⇒

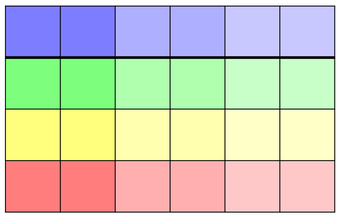
Thr Val
((2, 2),(2,3))
((2,12),(1,4))

```
Tensor input     = make_tensor(. . .);
Tensor tv_input  = composition(input, thr_val);
Tensor thr_input = tv_input(tid, _);
```

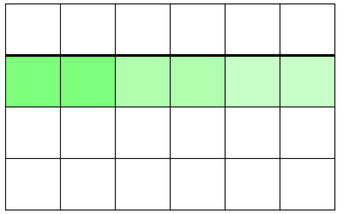
∘

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒



⇒



Compositional Power

Example

((2,3))
((1,4))
Values

((2, 2))
((2,12))

Threads

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒

Thr **Val**
((2, 2),(2,3))
((2,12),(1,4))

Moral of the story:

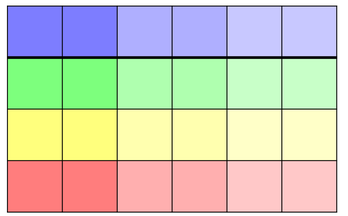
Given a layout, ***partitioning*** is functional ***composition*** followed by ***slicing.***

```
Tensor input     = make_tensor(. . .);
Tensor tv_input  = composition(input, thr_val);
Tensor thr_input = tv_input(tid, _);
```

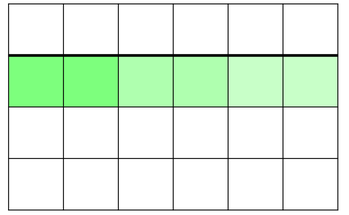
∘

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

⇒



⇒



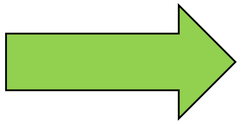
	0	1	2	3	4	5	6	7
0	a	e	i	m	q	u	y	γ
1	b	f	j	n	r	v	z	δ
2	c	g	k	o	s	w	α	ϵ
3	d	h	l	p	t	x	β	η

4x8 data
Any layout

o

	0	1	2	3
0	0	4	16	20
1	8	12	24	28
2	1	5	17	21
3	9	13	25	29
4	2	6	18	22
5	10	14	26	30
6	3	7	19	23
7	11	15	27	31

8x4 thr-val -> 4x8 1d coord
Shape ((2,4),(2, 2))
Stride ((8,1),(4,16))



	0	1	2	3
0	a	e	q	u
1	i	m	y	γ
2	b	f	r	v
3	j	n	z	δ
4	c	g	s	w
5	k	o	α	ϵ
6	d	h	t	x
7	l	p	β	η

8x4 thr-val -> 4x8 data

	0	1	2	3	4	5	6	7
0	a	e	i	m	q	u	y	γ
1	b	f	j	n	r	v	z	δ
2	c	g	k	o	s	w	α	ϵ
3	d	h	l	p	t	x	β	η

o

	0	1	2	3
0	0	4	16	20
1	8	12	24	28
2	1	5	17	21
3	9	13	25	29
4	2	6	18	22
5	10	14	26	30
6	3	7	19	23
7	11	15	27	31

4x8 data

8x4 thr-val -> 4x8 1d coord

	0	1	2	3	4	5	6	7
0	T0 V0	T0 V1	T1 V0	T1 V1	T0 V2	T0 V3	T1 V2	T1 V3
1	T2 V0	T2 V1	T3 V0	T3 V1	T2 V2	T2 V3	T3 V2	T3 V3
2	T4 V0	T4 V1	T5 V0	T5 V1	T4 V2	T4 V3	T5 V2	T5 V3
3	T6 V0	T6 V1	T7 V0	T7 V1	T6 V2	T6 V3	T7 V2	T7 V3

4x8 2d coord -> thr-val idx

Shape (4,(2,2, 2))

Stride (2,(8,1,16))

	0	1	2	3	4	5	6	7
0	a	e	i	m	q	u	y	γ
1	b	f	j	n	r	v	z	δ
2	c	g	k	o	s	w	α	ϵ
3	d	h	l	p	t	x	β	η

4x8 data

Any Layout

o

	0	1	2	3
0	0	4	16	20
1	8	12	24	28
2	1	5	17	21
3	9	13	25	29
4	2	6	18	22
5	10	14	26	30
6	3	7	19	23
7	11	15	27	31

8x4 TV → 4x8 1D coord

TV Layout

	0	1	2	3
0	a	e	q	u
1	i	m	y	γ
2	b	f	r	v
3	j	n	z	δ
4	c	g	s	w
5	k	o	α	ϵ
6	d	h	t	x
7	l	p	β	η

8x4 TV → 4x8 data

	0	1	2	3	4	5	6	7
0	T0 V0	T0 V1	T1 V0	T1 V1	T0 V2	T0 V3	T1 V2	T1 V3
1	T2 V0	T2 V1	T3 V0	T3 V1	T2 V2	T2 V3	T3 V2	T3 V3
2	T4 V0	T4 V1	T5 V0	T5 V1	T4 V2	T4 V3	T5 V2	T5 V3
3	T6 V0	T6 V1	T7 V0	T7 V1	T6 V2	T6 V3	T7 V2	T7 V3

4x8 2d coord -> thr-val idx

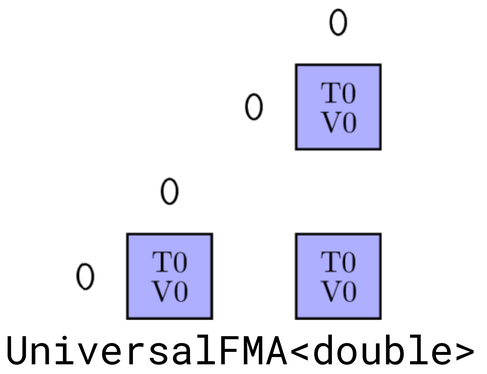
Shape (4,(2,2, 2))

Stride (2,(8,1,16))

Generic FMA

```cpp
template <class D, class A, class B, class C>
struct MMA_Traits<UniversalFMA<D,A,B,C>>
{
  using ElementDVal = D;
  using ElementAVal = A;
  using ElementBVal = B;
  using ElementCVal = C;

  // Logical shape of the MMA
  using Shape_MNK = Shape<_1,_1,_1>;

  // Logical thread id (tid) -> tidx
  using ThrID   = Layout<_1>;

  // (Logical thread id (tid), Logical value id (vid)) -> coord

  // (tid,vid) -> (m,k)
  using ALayout = Layout<Shape<_1,_1>>;
  // (tid,vid) -> (n,k)
  using BLayout = Layout<Shape<_1,_1>>;
  // (tid,vid) -> (m,n)
  using CLayout = Layout<Shape<_1,_1>>;
};
```

```cpp
template <class D, class A = D, class B = A, class C = D>
struct UniversalFMA
{
  using DRegisters = D[1];
  using ARegisters = A[1];
  using BRegisters = B[1];
  using CRegisters = C[1];

  CUTE_HOST_DEVICE static constexpr void
  fma(D      & d,
      A const& a,
      B const& b,
      C const& c)
  {
    // Forward to an ADL/cute free function for these types
    using cute::fma;
    fma(d, a, b, c);
  }
};
```

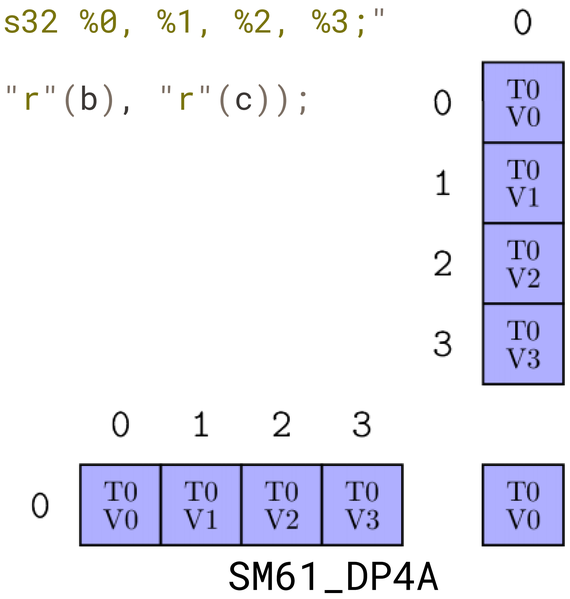


NOTE: These layouts map thread/value ID (domain) to MNK logical coordinate. We generally think of their `inverse() : MNK->thr,val`

SIMD MMA Atoms – SM61 packed int8 FMA

```cpp
template <>
struct MMA_Traits<SM61_DP4A>
{
  using ElementDVal = int32_t;
  using ElementAVal = int8_t;
  using ElementBVal = int8_t;
  using ElementCVal = int32_t;

  using ThrID   = Layout<Shape<_1>>;
  using ALayout = Layout<Shape<_1,_4>>;
  using BLayout = Layout<Shape<_1,_4>>;
  using CLayout = Layout<Shape<_1,_1>>;
};
```

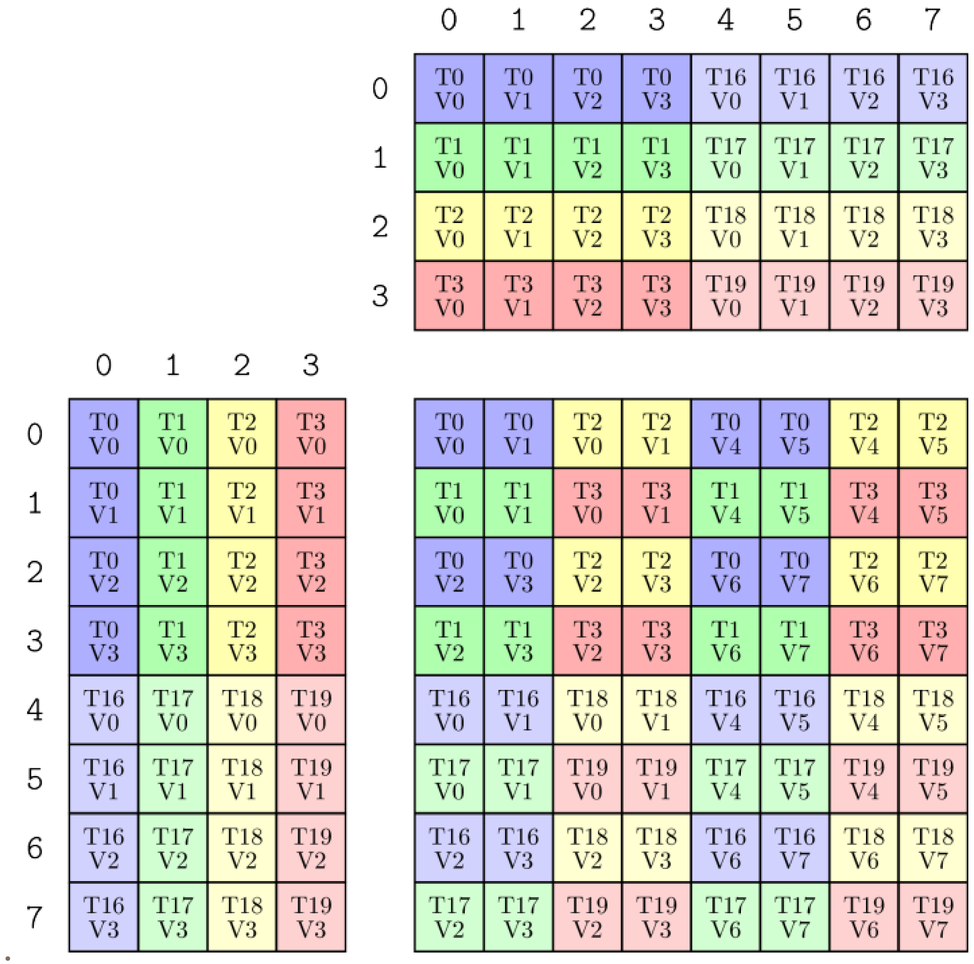
```cpp
struct SM61_DP4A
{
  using DRegisters = int32_t[1];
  using ARegisters = uint32_t[1];
  using BRegisters = uint32_t[1];
  using CRegisters = int32_t[1];

  // Arch register view
  CUTE_HOST_DEVICE static void
  fma(int32_t       & d,
      uint32_t const& a,
      uint32_t const& b,
      int32_t  const& c) {
    asm volatile("dp4a.s32.s32 %0, %1, %2, %3;"
                 : "=r"(d)
                 : "r"(a), "r"(b), "r"(c));
  }
};
```



Note the difference between architectural interface and logical element type.

Volta FP16 8x8x4 MMA

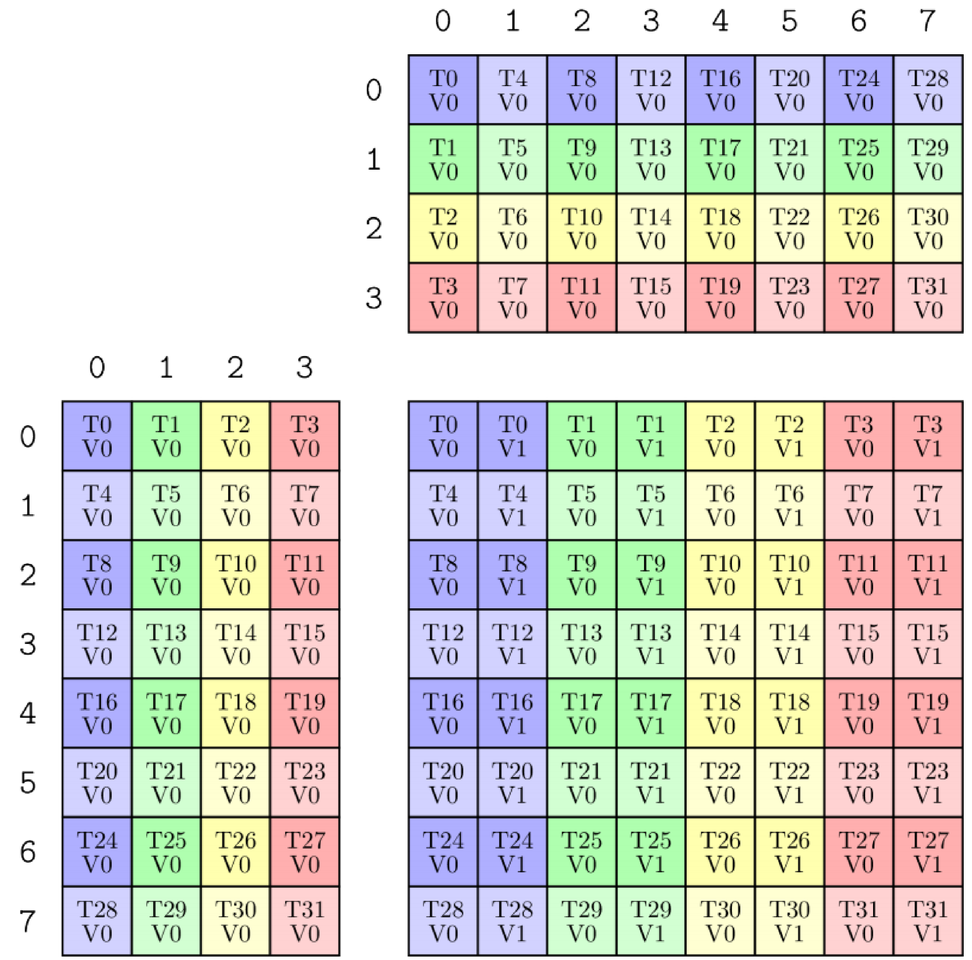
```
template <>
struct MMA_Traits<SM70_8x8x4_F32F16F16F32_NT>
{
  using ElementDVal = float;
  using ElementAVal = half_t;
  using ElementBVal = half_t;
  using ElementCVal = float;

  using Shape_MNK = Shape<_8,_8,_4>;

  using ThrID = Layout<Shape <_4, _2>,
                       Stride<_1,_16>>;

  // (T8,V4) -> (M8,K4)
  using ALayout = Layout<Shape <Shape <_4,_2>,_4>,
                         Stride<Stride<_8,_4>,_1>>;

  // (T8,V4) -> (N8,K4)
  using ALayout = Layout<Shape <Shape <_4,_2>,_4>,
                         Stride<Stride<_8,_4>,_1>>;

  // (T8,V8) -> (M8,N8)
  using CLayout = Layout<Shape <Shape <_2, _2,_2>,Shape <_2,_2, _2>>,
                         Stride<Stride<_1,_16,_4>,Stride<_8,_2,_32>>>;
};
```



SM70_8x8x4_F32F16F16F32_NT

Ampere FP64 MMA

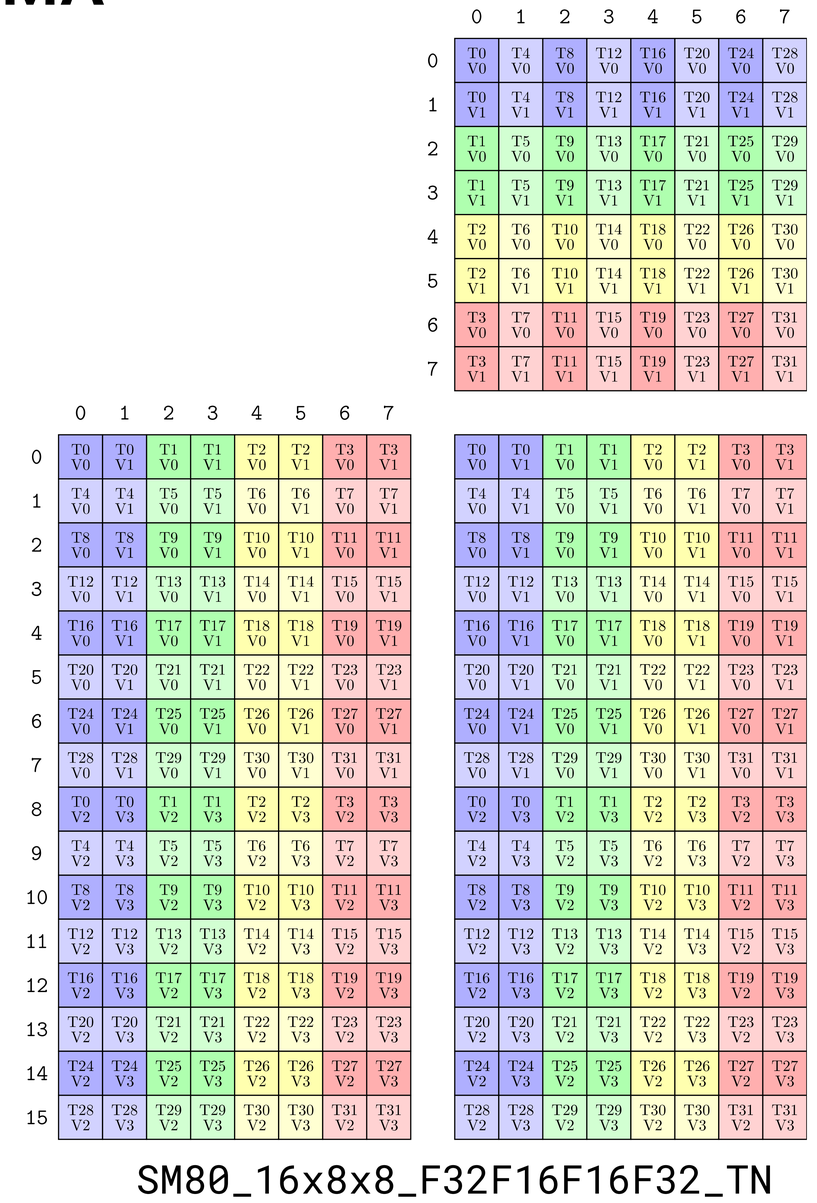
```
template <>
struct MMA_Traits<SM80_8x8x4_F64F64F64F64_TN>
{
  using ElementDVal = double;
  using ElementAVal = double;
  using ElementBVal = double;
  using ElementCVal = double;

  using Shape_MNK = Shape<_8,_8,_4>;

  using ThrID = Layout<_32>;

  // (T32,V1) -> (M8,K4)
  using ALayout = Layout<Shape <Shape < _4,_8>,_1>,
                         Stride<Stride< _8,_1>,_0>>;

  // (T32,V1) -> (N8,K4)
  using BLayout = Layout<Shape <Shape < _4,_8>,_1>,
                         Stride<Stride< _8,_1>,_0>>;

  // (T32,V2) -> (M8,N8)
  using CLayout = Layout<Shape <Shape < _4,_8>,_2>,
                         Stride<Stride<_16,_1>,_8>>;

};
```



SM80_8x8x4_F64F64F64F64_TN

Ampere/Turing FP16 MMA

```
template <>
struct MMA_Traits<SM80_16x8x8_F32F16F16F32_TN>
{
  using ElementDVal = float;
  using ElementAVal = half_t;
  using ElementBVal = half_t;
  using ElementCVal = float;

  using Shape_MNK = Shape<_16,_8,_8>;

  using ThrID = Layout<_32>;

  // (T32,V4) -> (M16,K8)
  using ALayout = Layout<Shape <Shape < _4,_8>,_1>,
                         Stride<Stride< _8,_1>,_0>>;

  // (T32,V2) -> (N8,K8)
  using BLayout = Layout<Shape <Shape < _4,_8>,_1>,
                         Stride<Stride< _8,_1>,_0>>;

  // (T32,V4) -> (M16,N8)
  using CLayout = Layout<Shape <Shape < _4,_8>,_2>,
                         Stride<Stride<_16,_1>,_8>>;

};
```

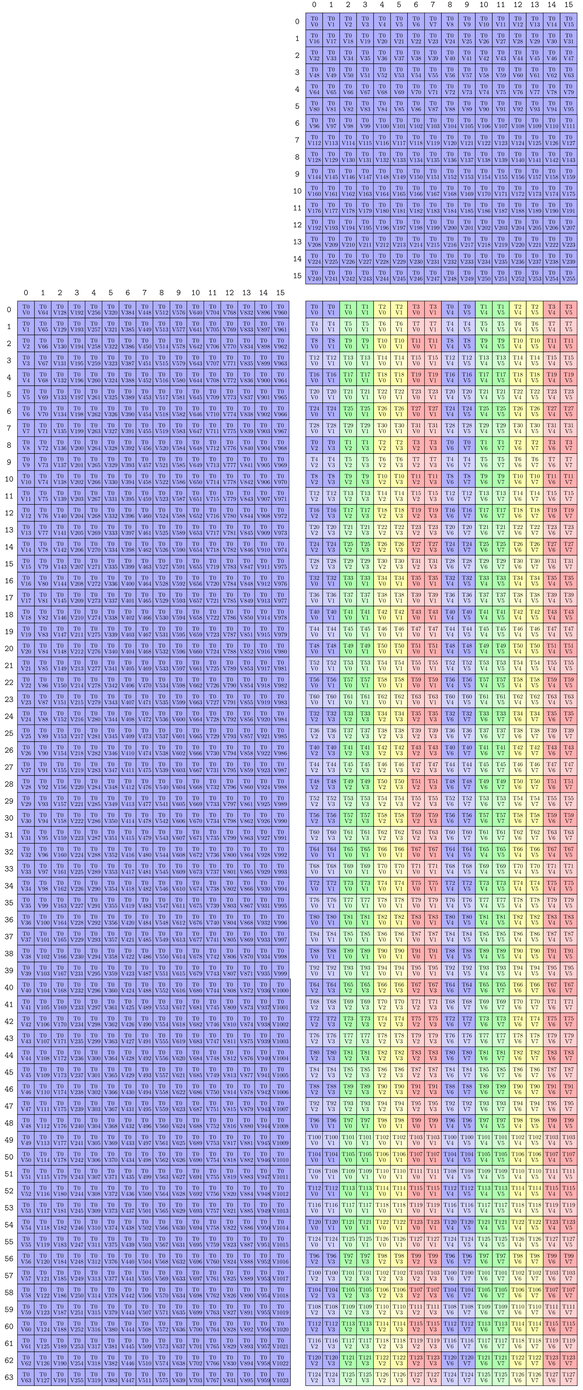


SM80_16x8x8_F32F16F16F32_TN

Hopper FP16 SS MMA

```
template <GMMA::Major tnspA, GMMA::Major tnspB>
struct MMA_Traits<SM90_64x16x16_F32F16F16_SS<tnspA, tnspB>>
{
  Using ElementDVal = float;
  using ElementAVal = half_t;
  using ElementBVal = half_t;
  using ElementCVal = float;

  using ElementAFrg = GMMA::smem_desc<tnspA>;
  using ElementBFrg = GMMA::smem_desc<tnspB>;

  using Shape_MNK = Shape<_64,_16,_16>;

  using ThrID = Layout<_128>;

  using ALayout = GMMA::ABLayout< 64, 16>;
  using BLayout = GMMA::ABLayout< 16, 16>;
  using CLayout = Layout<Shape <Shape <  _4,_8, _4>,Shape < _2,_2,  _2>>,
                         Stride<Stride<_128,_1,_16>,Stride<_64,_8,_512>>>;
};
```

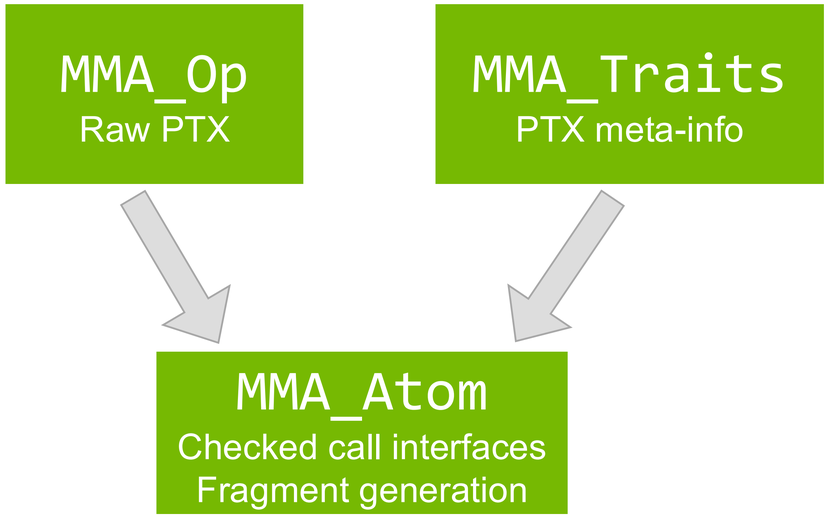
NOTE: In practice, we would not use such a small N mode for the instruction shape – Ideally, we would pick the N mode that matches our tile shape e.g. 128



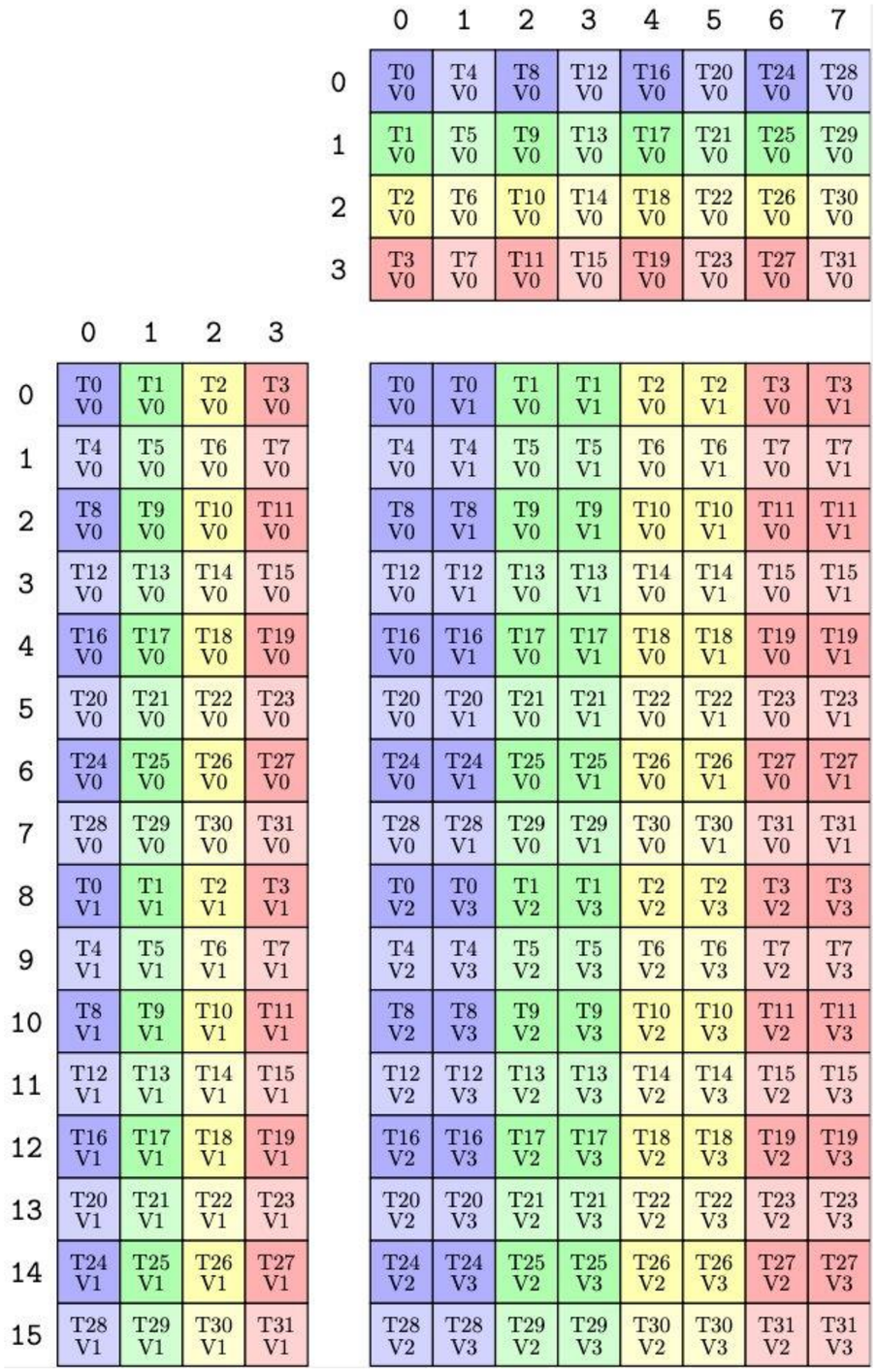
SM90_64x16x16_F32F16F16_SS

MMA_Atom

The basic building block

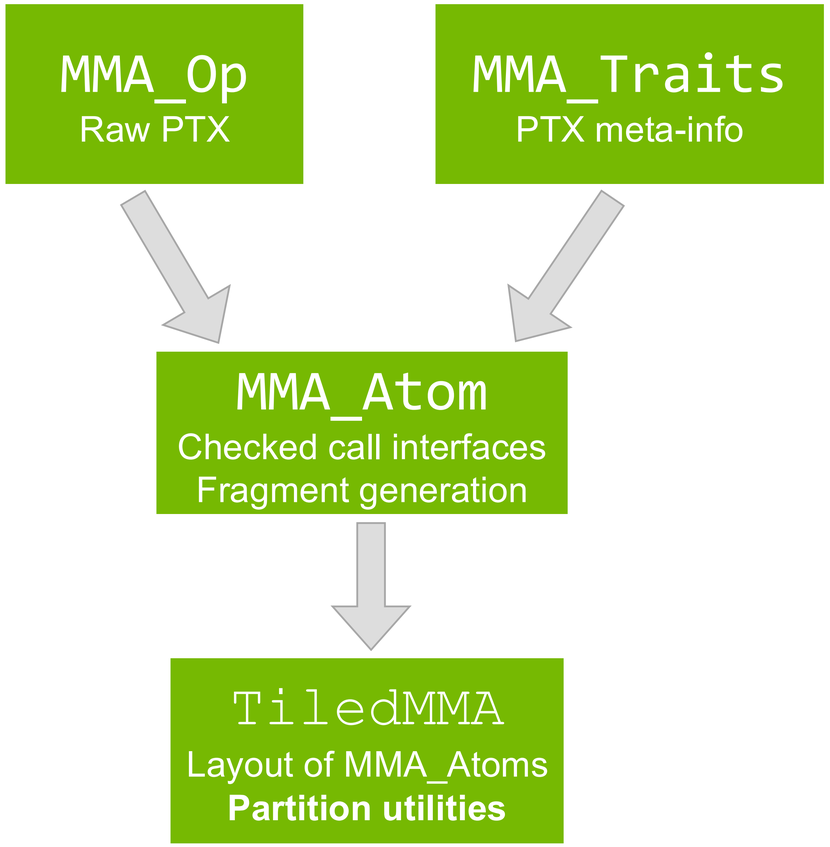



```
MMA_Atom mma = MMA_Atom<SM90_16x8x4_F64F64F64F64_TN>{};

print_latex(mma);
```

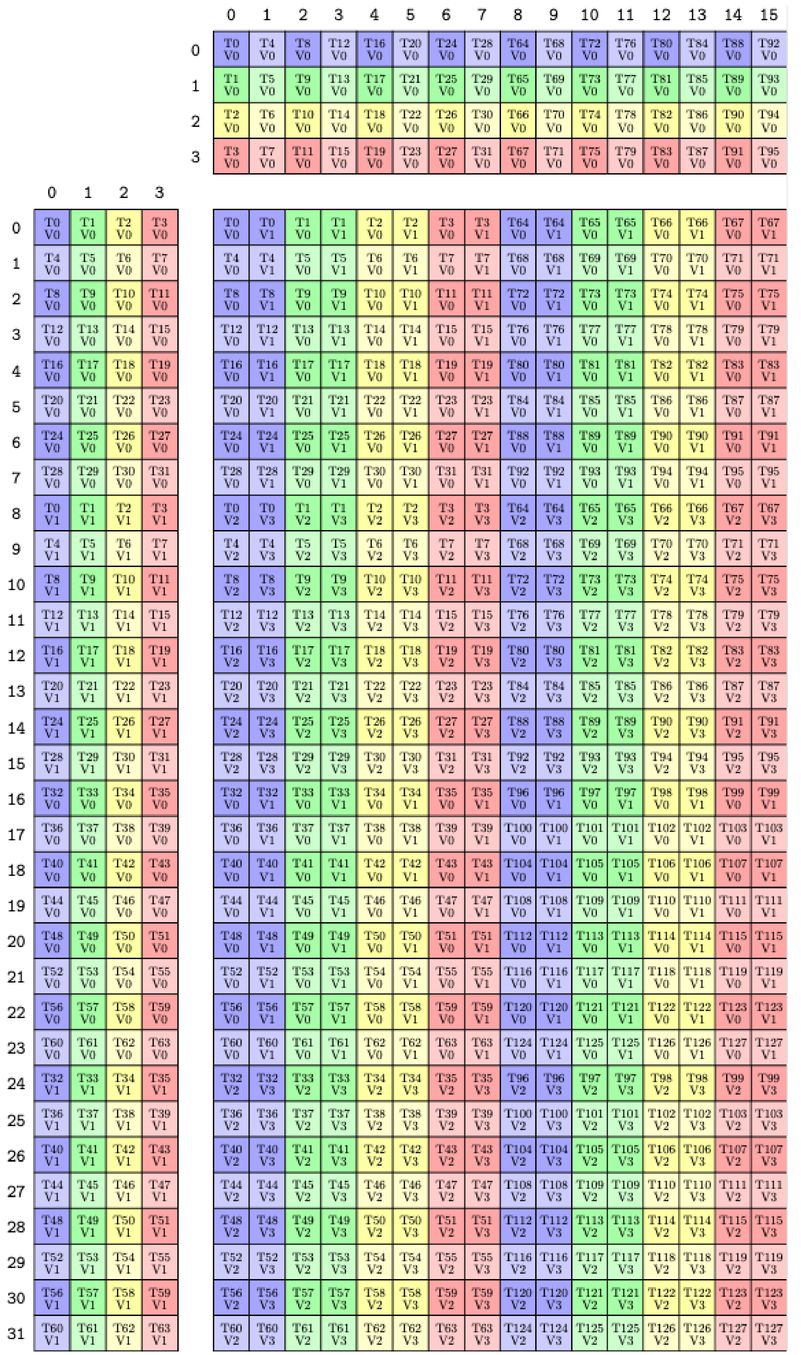


TiledMMA

Construct larger operations

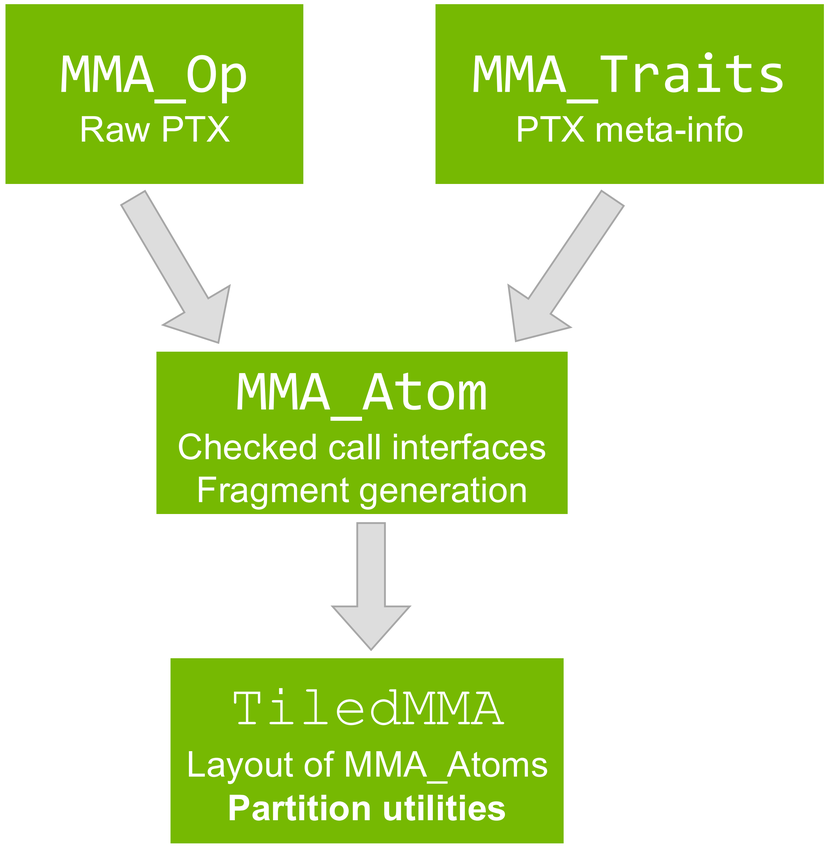



```
Tiled_MMA mma = make_tiled_mma(SM90_16x8x4_F64F64F64F64_TN{},
                               Layout<Shape<_2,_2>>{}); // 2x2 warps

print_latex(mma);
```

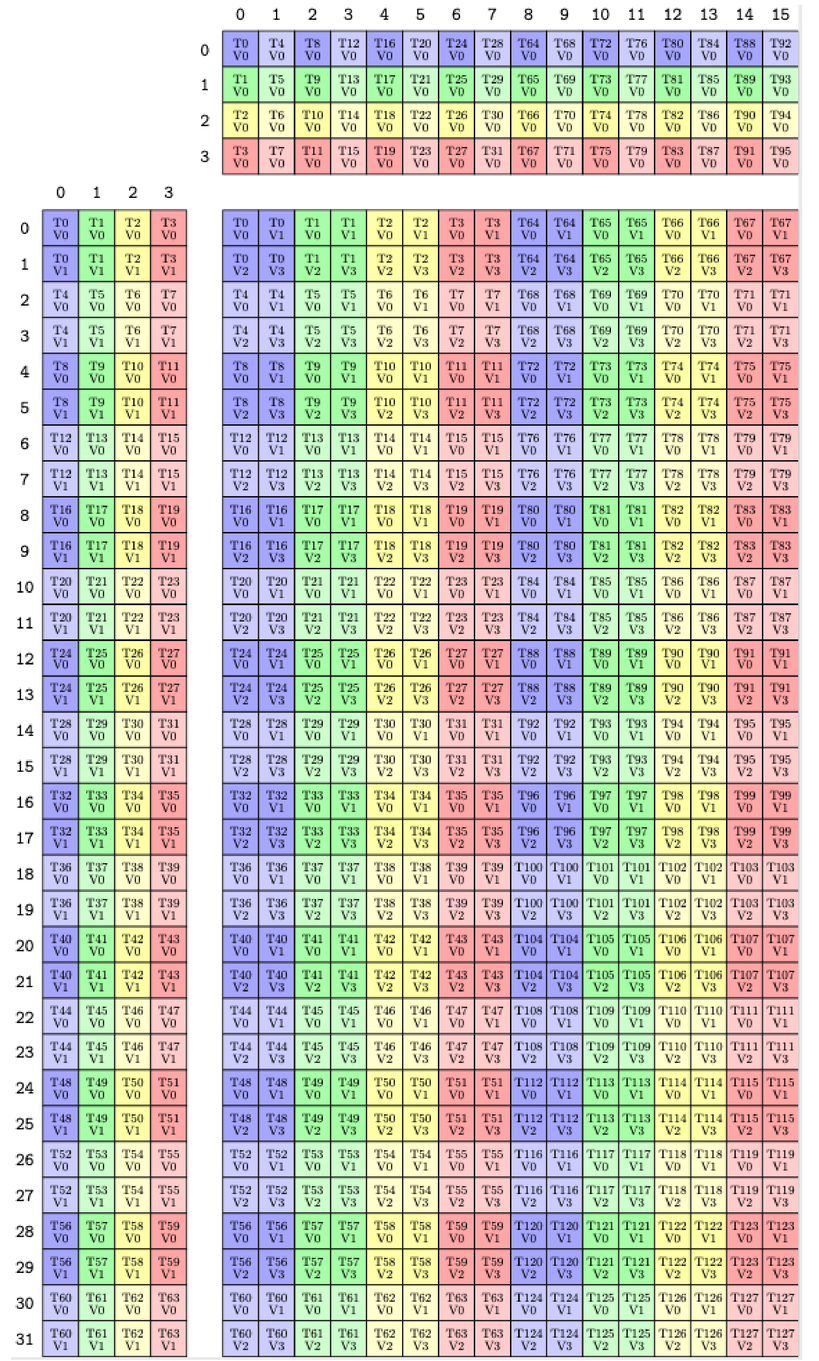


TiledMMA

Construct larger operations

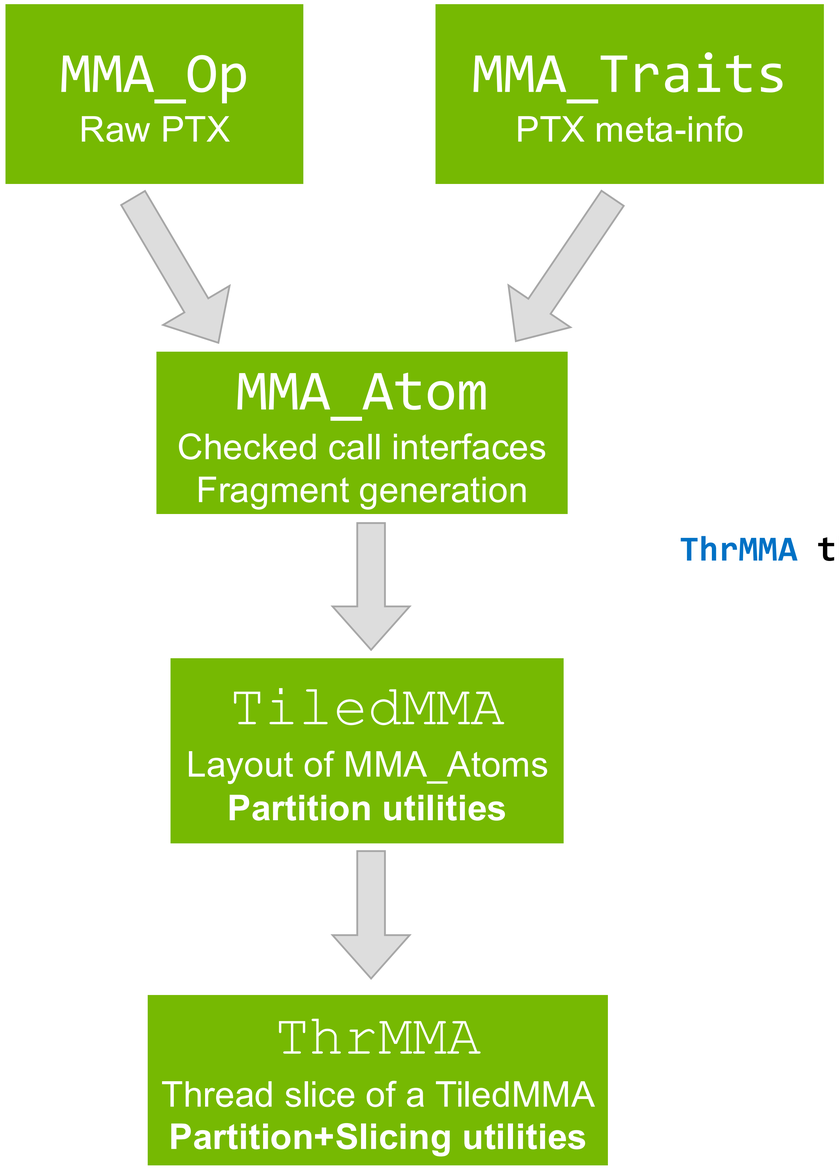



```
Tiled_MMA mma = make_tiled_mma(SM90_16x8x4_F64F64F64F64_TN{},
                               Layout<Shape<_2,_2>>{},   // 2x2 warps
                               Tile<Layout<_8,_2>>{});   // Permute M

print_latex(mma);
```

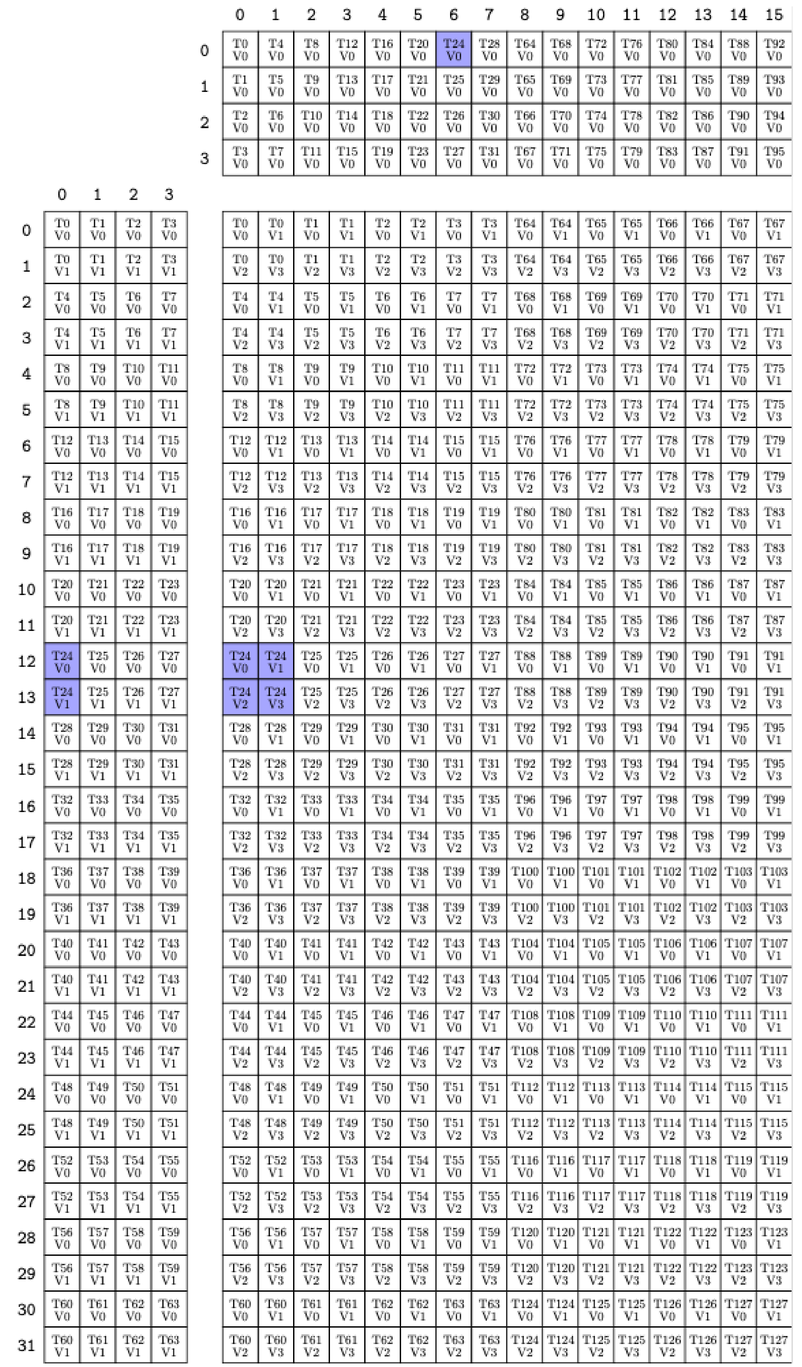


ThrMMA

A thread's view of the partition




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



ThrMMA

A thread's view of the partition

MMA_Op
Raw PTX

MMA_Traits
PTX meta-info

MMA_Atom
Checked call interfaces
Fragment generation

TiledMMA
Layout of MMA_Atoms
Partition utilities

ThrMMA
Thread slice of a TiledMMA
Partition+Slicing utilities

```
Tensor sA = make_tensor(ptrA, Shape<_64,_16>{}); // (64,16)
Tensor sB = make_tensor(ptrB, Shape<_32,_16>{}); // (32,16)
Tensor gC = make_tensor(ptrC, Shape<_64,_32>{}); // (64,32)

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

Tensor tCsA = thr_mma.partition_A(sA);              // (2,M',K')
Tensor tCsB = thr_mma.partition_B(sB);              // (1,N',K')
Tensor tCgC = thr_mma.partition_C(gC);              // (4,M',N')

Tensor tCrA = thr_mma.make_fragment_A(tCsA);        // (2,M',K')
Tensor tCrB = thr_mma.make_fragment_B(tCsB);        // (1,N',K')
Tensor tCrC = thr_mma.make_fragment_C(tCgC);        // (4,M',N')

copy(tCsA, tCrA);
copy(tCsB, rCrB);
clear(tCrC);

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(tCrA(_,m,k), tCrB(_,n,k), tCrC(_,m,n));
} } }

copy(tCrC, tCgC);
```

	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
0	T0 V0	T1 V0	T2 V0	T3 V0
1	T0 V1	T1 V1	T2 V1	T3 V1
2	T4 V0	T5 V0	T6 V0	T7 V0
3	T4 V1	T5 V1	T6 V1	T7 V1
4	T8 V0	T9 V0	T10 V0	T11 V0
5	T8 V1	T9 V1	T10 V1	T11 V1
6	T12 V0	T13 V0	T14 V0	T15 V0
7	T12 V1	T13 V1	T14 V1	T15 V1
8	T16 V0	T17 V0	T18 V0	T19 V0
9	T16 V1	T17 V1	T18 V1	T19 V1
10	T20 V0	T21 V0	T22 V0	T23 V0
11	T20 V1	T21 V1	T22 V1	T23 V1
12	T24 V0	T25 V0	T26 V0	T27 V0
13	T24 V1	T25 V1	T26 V1	T27 V1
14	T28 V0	T29 V0	T30 V0	T31 V0
15	T28 V1	T29 V1	T30 V1	T31 V1
16	T32 V0	T33 V0	T34 V0	T35 V0
17	T32 V1	T33 V1	T34 V1	T35 V1
18	T36 V0	T37 V0	T38 V0	T39 V0
19	T36 V1	T37 V1	T38 V1	T39 V1
20	T40 V0	T41 V0	T42 V0	T43 V0
21	T40 V1	T41 V1	T42 V1	T43 V1
22	T44 V0	T45 V0	T46 V0	T47 V0
23	T44 V1	T45 V1	T46 V1	T47 V1
24	T48 V0	T49 V0	T50 V0	T51 V0
25	T48 V1	T49 V1	T50 V1	T51 V1
26	T52 V0	T53 V0	T54 V0	T55 V0
27	T52 V1	T53 V1	T54 V1	T55 V1
28	T56 V0	T57 V0	T58 V0	T59 V0
29	T56 V1	T57 V1	T58 V1	T59 V1
30	T60 V0	T61 V0	T62 V0	T63 V0
31	T60 V1	T61 V1	T62 V1	T63 V1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T0 V0	T0 V1	T1 V0	T1 V1	T2 V0	T2 V1	T3 V0	T3 V1	T64 V0	T64 V1	T65 V0	T65 V1	T66 V0	T66 V1	T67 V0	T67 V1
1	T0 V2	T0 V3	T1 V2	T1 V3	T2 V2	T2 V3	T3 V2	T3 V3	T64 V2	T64 V3	T65 V2	T65 V3	T66 V2	T66 V3	T67 V2	T67 V3
2	T4 V0	T4 V1	T5 V0	T5 V1	T6 V0	T6 V1	T7 V0	T7 V1	T68 V0	T68 V1	T69 V0	T69 V1	T70 V0	T70 V1	T71 V0	T71 V1
3	T4 V2	T4 V3	T5 V2	T5 V3	T6 V2	T6 V3	T7 V2	T7 V3	T68 V2	T68 V3	T69 V2	T69 V3	T70 V2	T70 V3	T71 V2	T71 V3
4	T8 V0	T8 V1	T9 V0	T9 V1	T10 V0	T10 V1	T11 V0	T11 V1	T72 V0	T72 V1	T73 V0	T73 V1	T74 V0	T74 V1	T75 V0	T75 V1
5	T8 V2	T8 V3	T9 V2	T9 V3	T10 V2	T10 V3	T11 V2	T11 V3	T72 V2	T72 V3	T73 V2	T73 V3	T74 V2	T74 V3	T75 V2	T75 V3
6	T12 V0	T12 V1	T13 V0	T13 V1	T14 V0	T14 V1	T15 V0	T15 V1	T76 V0	T76 V1	T77 V0	T77 V1	T78 V0	T78 V1	T79 V0	T79 V1
7	T12 V2	T12 V3	T13 V2	T13 V3	T14 V2	T14 V3	T15 V2	T15 V3	T76 V2	T76 V3	T77 V2	T77 V3	T78 V2	T78 V3	T79 V2	T79 V3
8	T16 V0	T16 V1	T17 V0	T17 V1	T18 V0	T18 V1	T19 V0	T19 V1	T80 V0	T80 V1	T81 V0	T81 V1	T82 V0	T82 V1	T83 V0	T83 V1
9	T16 V2	T16 V3	T17 V2	T17 V3	T18 V2	T18 V3	T19 V2	T19 V3	T80 V2	T80 V3	T81 V2	T81 V3	T82 V2	T82 V3	T83 V2	T83 V3
10	T20 V0	T20 V1	T21 V0	T21 V1	T22 V0	T22 V1	T23 V0	T23 V1	T84 V0	T84 V1	T85 V0	T85 V1	T86 V0	T86 V1	T87 V0	T87 V1
11	T20 V2	T20 V3	T21 V2	T21 V3	T22 V2	T22 V3	T23 V2	T23 V3	T84 V2	T84 V3	T85 V2	T85 V3	T86 V2	T86 V3	T87 V2	T87 V3
12	T24 V0	T24 V1	T25 V0	T25 V1	T26 V0	T26 V1	T27 V0	T27 V1	T88 V0	T88 V1	T89 V0	T89 V1	T90 V0	T90 V1	T91 V0	T91 V1
13	T24 V2	T24 V3	T25 V2	T25 V3	T26 V2	T26 V3	T27 V2	T27 V3	T88 V2	T88 V3	T89 V2	T89 V3	T90 V2	T90 V3	T91 V2	T91 V3
14	T28 V0	T28 V1	T29 V0	T29 V1	T30 V0	T30 V1	T31 V0	T31 V1	T92 V0	T92 V1	T93 V0	T93 V1	T94 V0	T94 V1	T95 V0	T95 V1
15	T28 V2	T28 V3	T29 V2	T29 V3	T30 V2	T30 V3	T31 V2	T31 V3	T92 V2	T92 V3	T93 V2	T93 V3	T94 V2	T94 V3	T95 V2	T95 V3
16	T32 V0	T32 V1	T33 V0	T33 V1	T34 V0	T34 V1	T35 V0	T35 V1	T96 V0	T96 V1	T97 V0	T97 V1	T98 V0	T98 V1	T99 V0	T99 V1
17	T32 V2	T32 V3	T33 V2	T33 V3	T34 V2	T34 V3	T35 V2	T35 V3	T96 V2	T96 V3	T97 V2	T97 V3	T98 V2	T98 V3	T99 V2	T99 V3
18	T36 V0	T36 V1	T37 V0	T37 V1	T38 V0	T38 V1	T39 V0	T39 V1	T100 V0	T100 V1	T101 V0	T101 V1	T102 V0	T102 V1	T103 V0	T103 V1
19	T36 V2	T36 V3	T37 V2	T37 V3	T38 V2	T38 V3	T39 V2	T39 V3	T100 V2	T100 V3	T101 V2	T101 V3	T102 V2	T102 V3	T103 V2	T103 V3
20	T40 V0	T40 V1	T41 V0	T41 V1	T42 V0	T42 V1	T43 V0	T43 V1	T104 V0	T104 V1	T105 V0	T105 V1	T106 V0	T106 V1	T107 V0	T107 V1
21	T40 V2	T40 V3	T41 V2	T41 V3	T42 V2	T42 V3	T43 V2	T43 V3	T104 V2	T104 V3	T105 V2	T105 V3	T106 V2	T106 V3	T107 V2	T107 V3
22	T44 V0	T44 V1	T45 V0	T45 V1	T46 V0	T46 V1	T47 V0	T47 V1	T108 V0	T108 V1	T109 V0	T109 V1	T110 V0	T110 V1	T111 V0	T111 V1
23	T44 V2	T44 V3	T45 V2	T45 V3	T46 V2	T46 V3	T47 V2	T47 V3	T108 V2	T108 V3	T109 V2	T109 V3	T110 V2	T110 V3	T111 V2	T111 V3
24	T48 V0	T48 V1	T49 V0	T49 V1	T50 V0	T50 V1	T51 V0	T51 V1	T112 V0	T112 V1	T113 V0	T113 V1	T114 V0	T114 V1	T115 V0	T115 V1
25	T48 V2	T48 V3	T49 V2	T49 V3	T50 V2	T50 V3	T51 V2	T51 V3	T112 V2	T112 V3	T113 V2	T113 V3	T114 V2	T114 V3	T115 V2	T115 V3
26	T52 V0	T52 V1	T53 V0	T53 V1	T54 V0	T54 V1	T55 V0	T55 V1	T116 V0	T116 V1	T117 V0	T117 V1	T118 V0	T118 V1	T119 V0	T119 V1
27	T52 V2	T52 V3	T53 V2	T53 V3	T54 V2	T54 V3	T55 V2	T55 V3	T116 V2	T116 V3	T117 V2	T117 V3	T118 V2	T118 V3	T119 V2	T119 V3
28	T56 V0	T56 V1	T57 V0	T57 V1	T58 V0	T58 V1	T59 V0	T59 V1	T120 V0	T120 V1	T121 V0	T121 V1	T122 V0	T122 V1	T123 V0	T123 V1
29	T56 V2	T56 V3	T57 V2	T57 V3	T58 V2	T58 V3	T59 V2	T59 V3	T120 V2	T120 V3	T121 V2	T121 V3	T122 V2	T122 V3	T123 V2	T123 V3
30	T60 V0	T60 V1	T61 V0	T61 V1	T62 V0	T62 V1	T63 V0	T63 V1	T124 V0	T124 V1	T125 V0	T125 V1	T126 V0	T126 V1	T127 V0	T127 V1
31	T60 V2	T60 V3	T61 V2	T61 V3	T62 V2	T62 V3	T63 V2	T63 V3	T124 V2	T124 V3	T125 V2	T125 V3	T126 V2	T126 V3	T127 V2	T127 V3