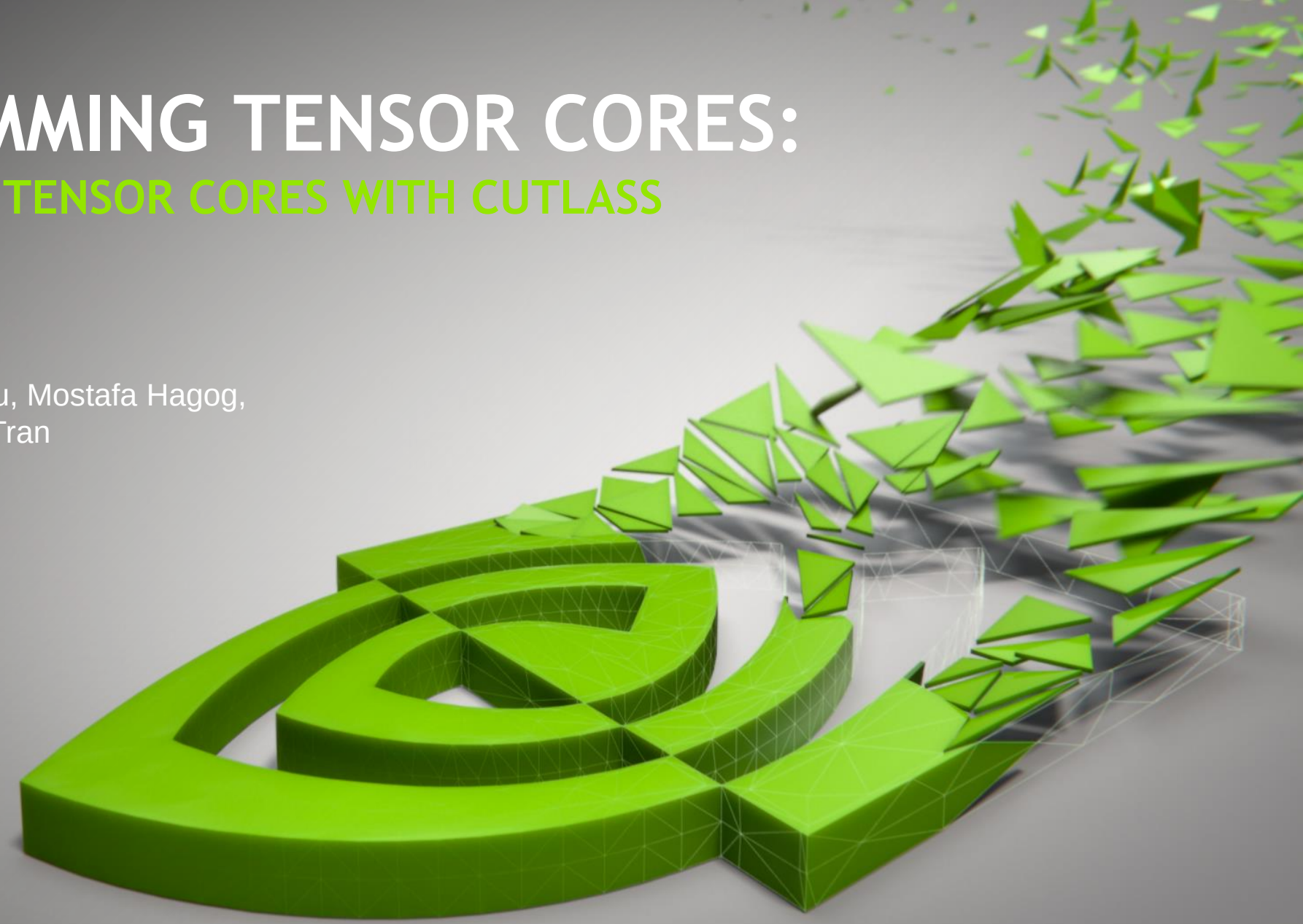
PROGRAMMING TENSOR CORES:

NATIVE VOLTA TENSOR CORES WITH CUTLASS

Andrew Kerr, Timmy Liu, Mostafa Hagog,
Julien Demouth, John Tran



NVIDIA

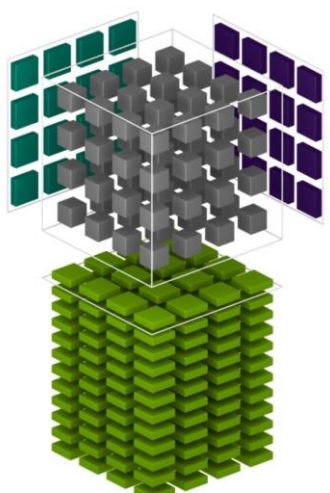
March 20, 2019

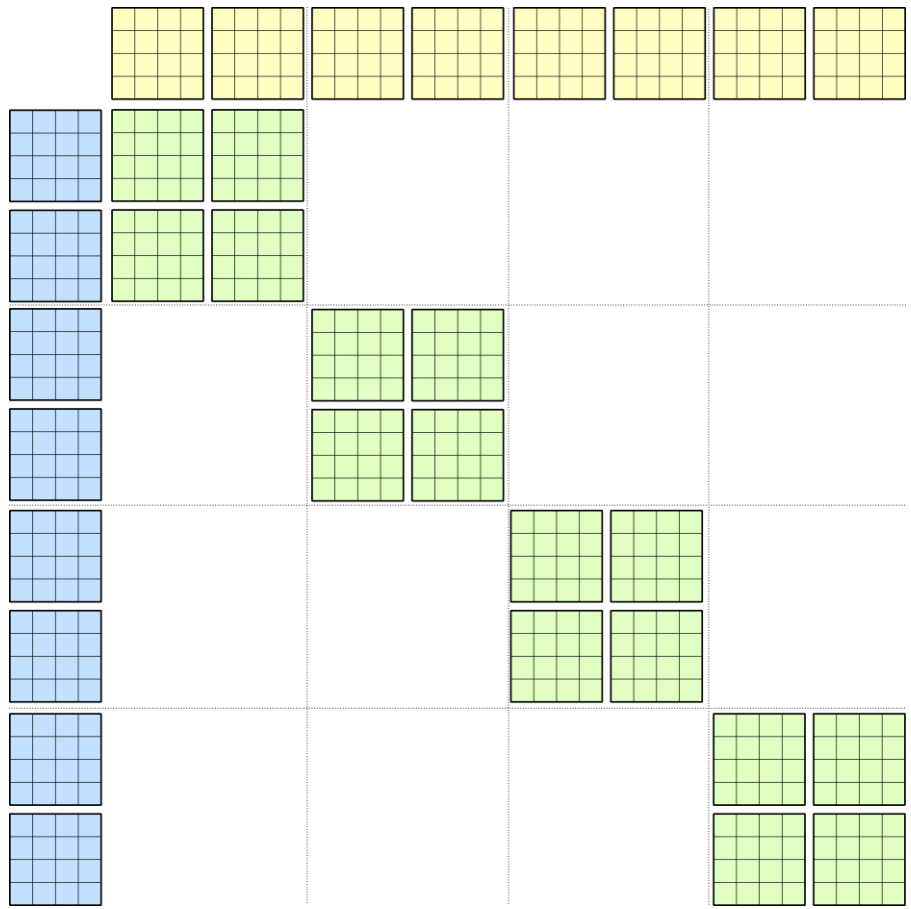
PROGRAMMING TENSOR CORES IN CUDA

mma.sync (new instruction in CUDA 10.1)

Feeding the Data Path

CUTLASS 1.3 – Native Volta Tensor Cores GEMM
(March 20, 2019)

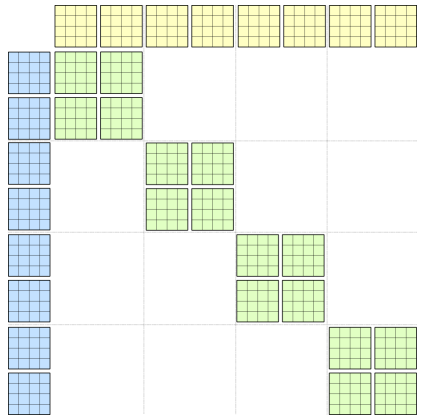




TENSOR CORES

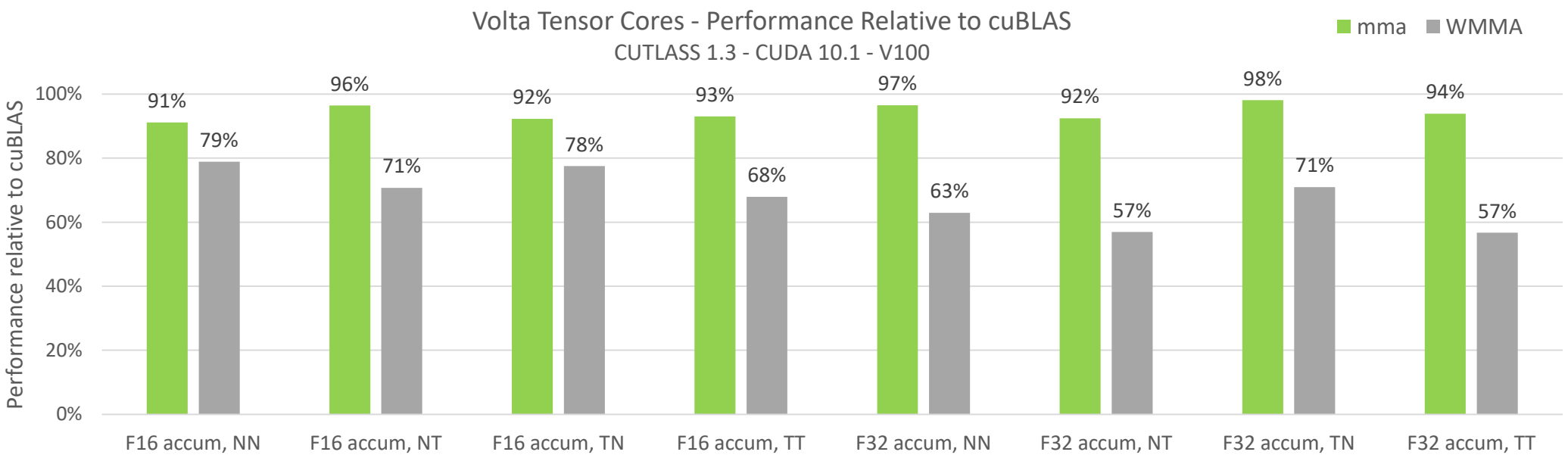
Tensor Cores

- **8x speedup** for mixed-precision matrix multiply
- Programmable via WMMA API (CUDA 9)



Direct access to Volta Tensor Cores: **mma.sync** (new instruction in CUDA 10.1)

- Maximum efficiency on Volta SM Architecture
- New in CUTLASS 1.3



https://github.com/NVIDIA/cutlass

TENSOR CORES

This talk is about Volta Tensor Cores.

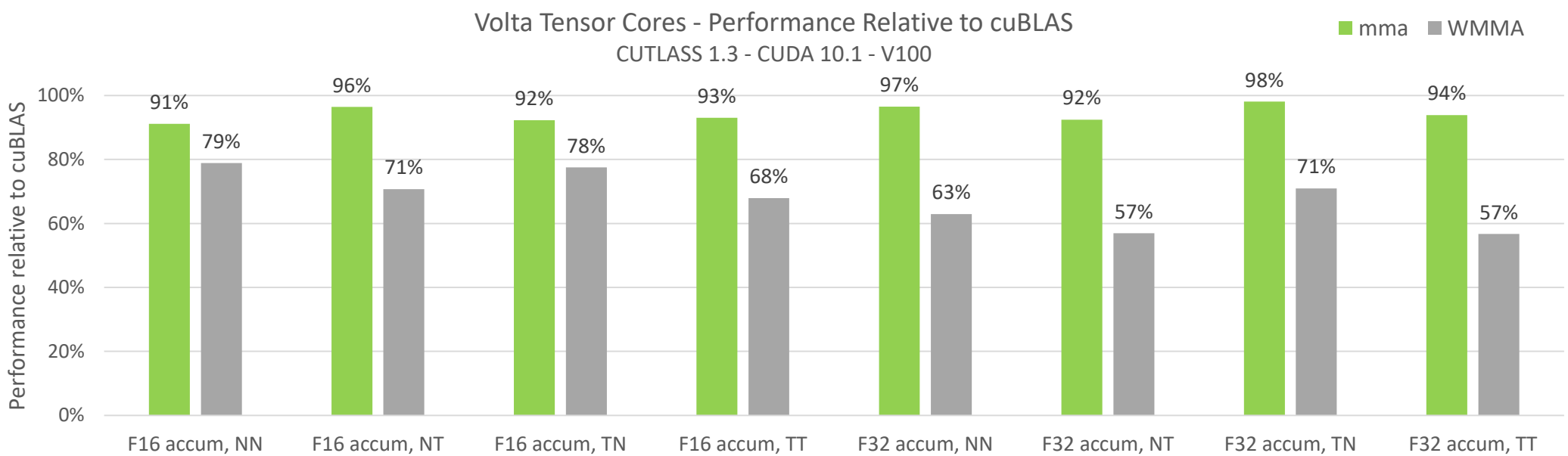
Warp-synchronous Matrix Multiply Accumulate

(WMMA API)

portable abstraction layer for Tensor Cores

mma.sync

Direct access to Volta Tensor Cores



https://github.com/NVIDIA/cutlass

VOLTA MMA.SYNC

VOLTA MMA.SYNC

Warp-scoped matrix multiply instruction

mma.sync: new instruction in CUDA 10.1

- Directly targets Volta Tensor Cores

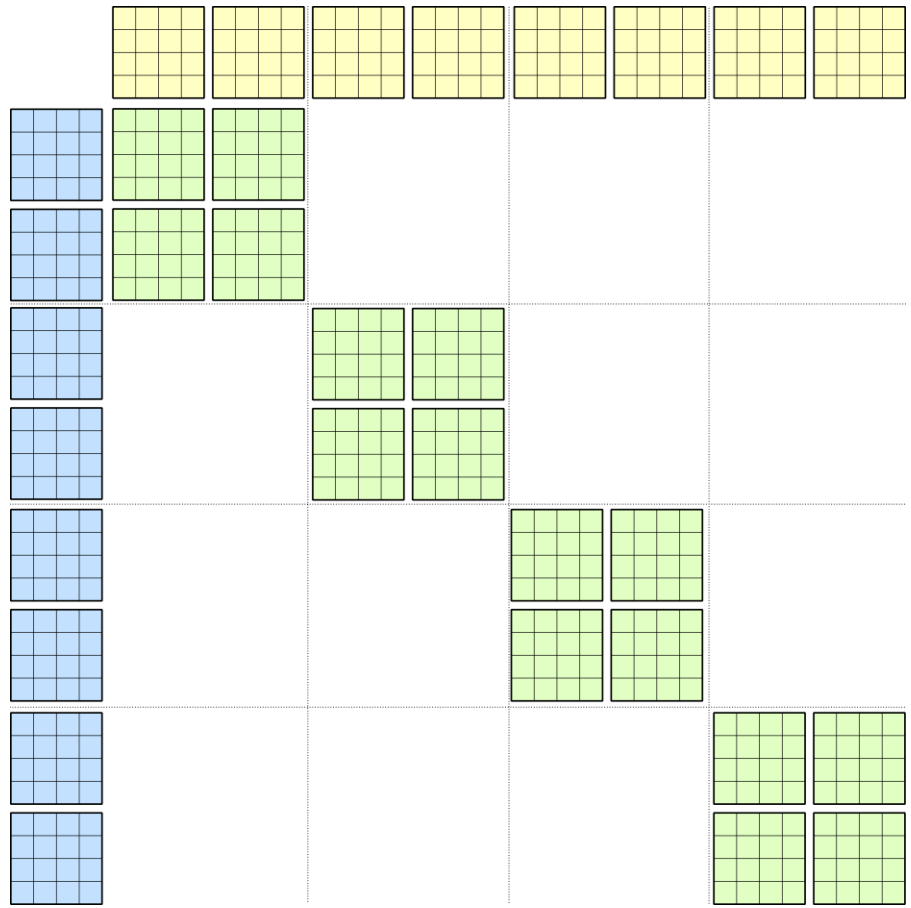
Matrix multiply-accumulate

D = A * B + C

- A, B: half
- C, D: float or half

Warp-synchronous:

- Four independent **8-by-8-by-4** matrix multiply-accumulate operations



VOLTA MMA.SYNC

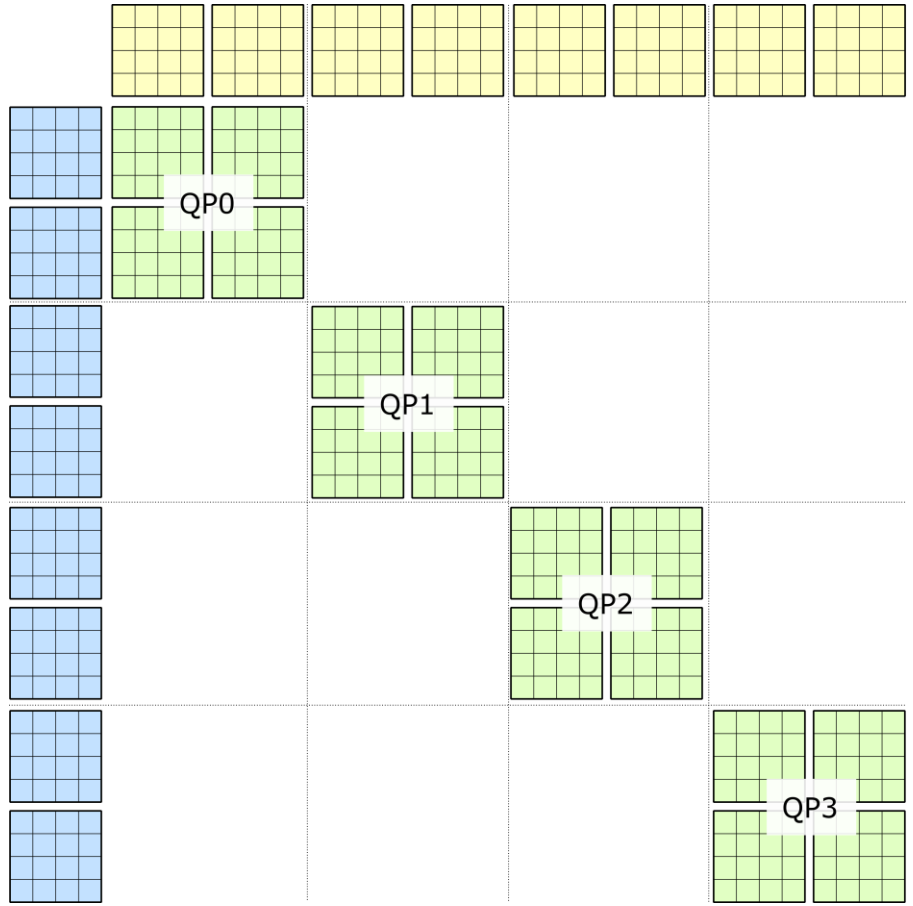
Warp-scoped matrix multiply instruction

Warp is partitioned into Quad Pairs

- QP0: `T0..T3` `T16..T19`
- QP1: `T4..T7` `T20..T23`
- QP2: `T8..T11` `T24..T27`
- QP3: `T12..T15` `T28..T31`

(eight threads each)

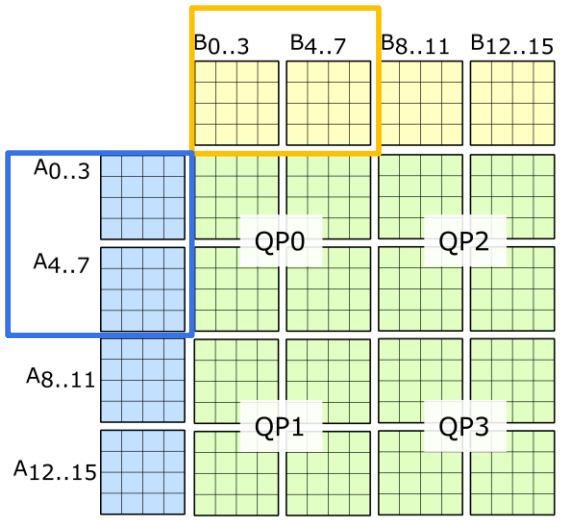
Each Quad Pair performs one **8-by-8-by-4** matrix multiply



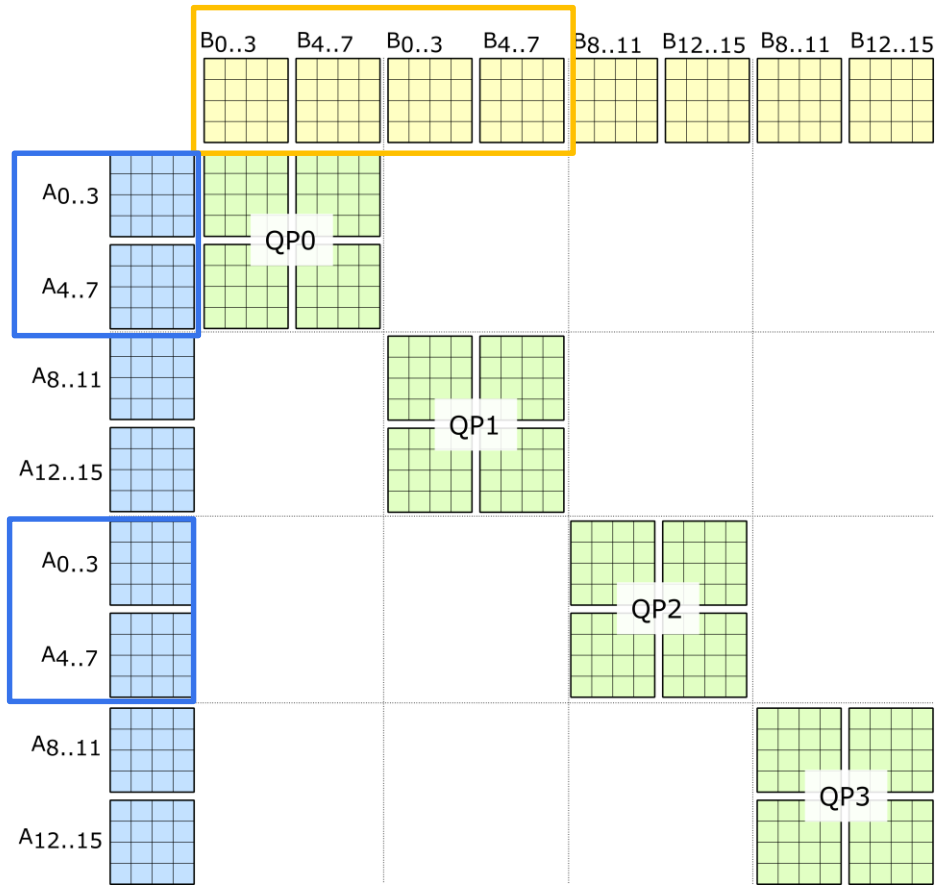
COMPOSING MATRIX MULTIPLIES

Replicate data to compute warp-wide 16-by-16-by-4 matrix product

- $A_{0..7}$: QP0,QP2 $A_{8..15}$: QP1, QP3
- $B_{0..7}$: QP0,QP1 $B_{8..15}$: QP2, QP3



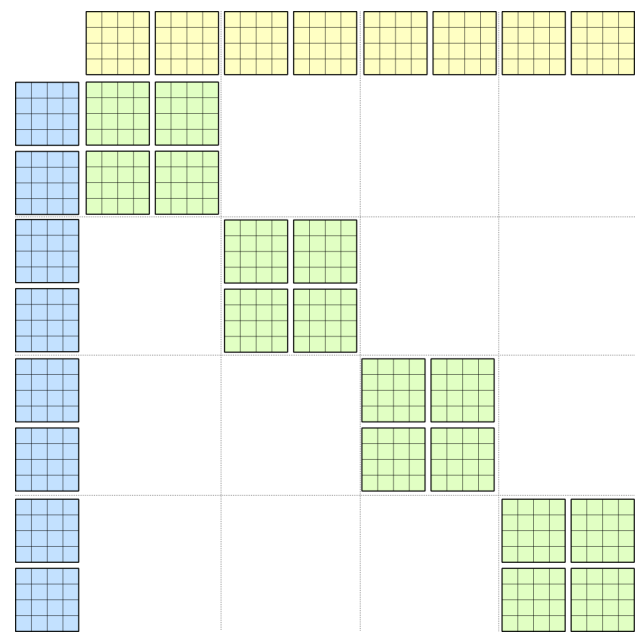
1 x mma.sync: 16-by-16-by-4



VOLTA MMA.SYNC D = A * B + C

PTX Syntax

```
mma.sync.aligned.m8n8k4.alayout.blayout.dtype.f16.f16.ctype   d, a, b, c;

.alayout = {.row, .col};
.blayout = {.row, .col};
.ctype   = {.f16, .f32};
.dtype   = {.f16, .f32};
```



```
d: 8 x .dtype
a: 4 x .f16
b: 4 x .f16
c: 8 x .ctype
```

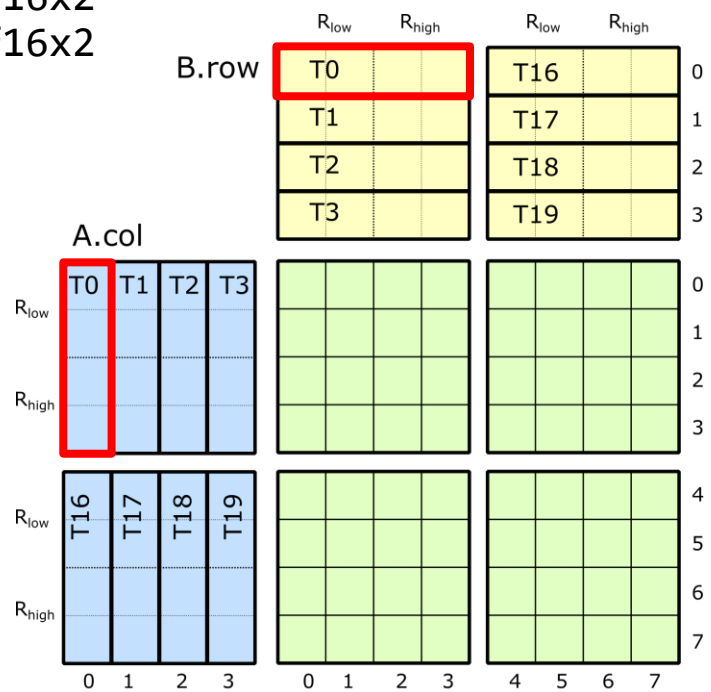
Note: .f16 elements must be packed into .f16x2

https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#warp-level-matrix-instructions-mma

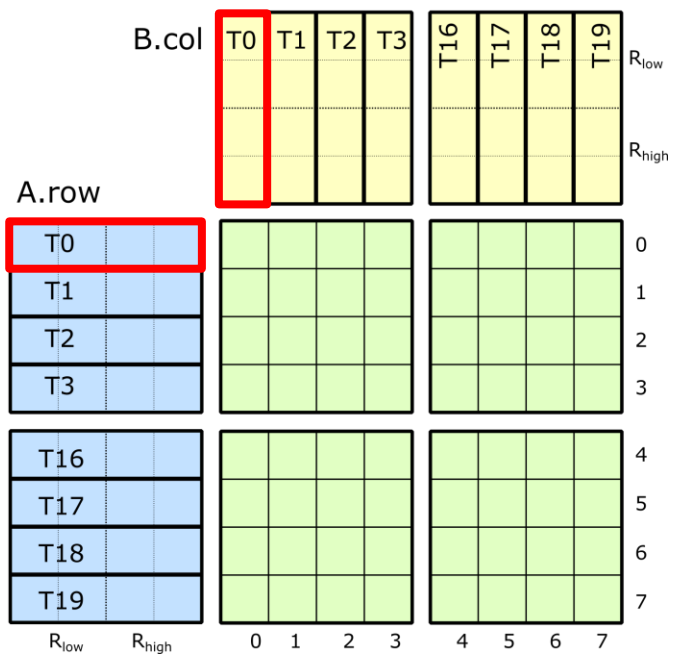
THREAD-DATA MAPPING - F16 MULTIPLICANDS

Distributed among threads in quad pair (QP0 shown)

```
mma.sync.aligned.m8n8k4.alayout.blayout.dtype.f16.f16.ctype   d, a, b, c;

.alayout = {.row, .col};
.blayout = {.row, .col};

a: 2 x .f16x2
b: 2 x .f16x2
```



COL-ROW (“NT”)

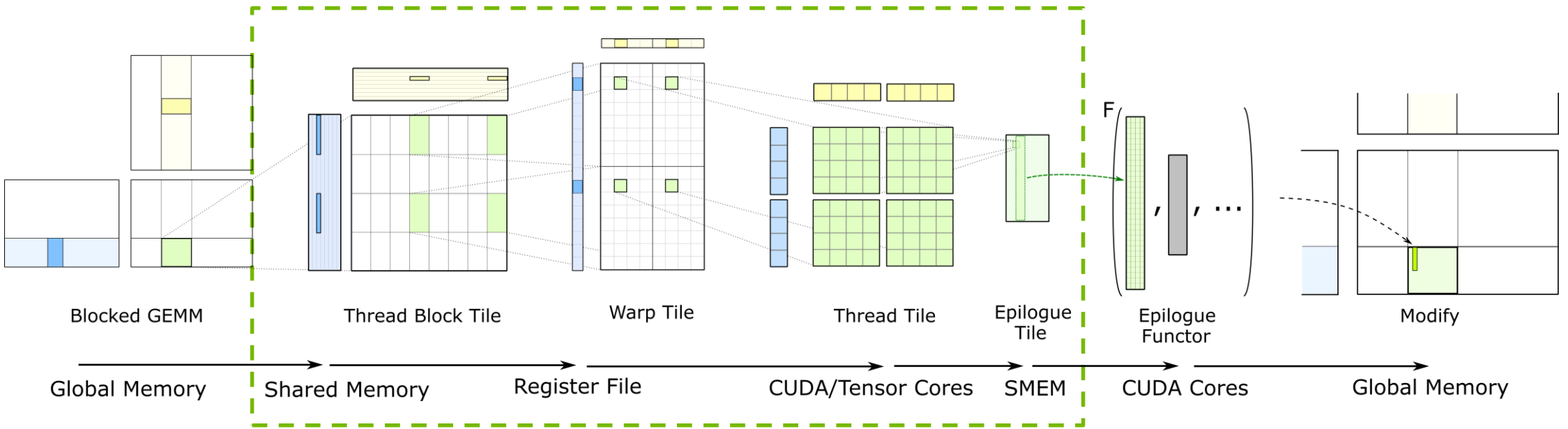


ROW-COL (“TN”)

FEEDING THE DATA PATH

FEEDING THE DATA PATH

Efficiently storing and loading through shared memory



See CUTLASS GTC 2018 talk for more details about this model.

CONFLICT-FREE ACCESS TO SHARED MEMORY

Efficiently storing and loading through shared memory

Bank conflicts between threads in the same phase

4B words are accessed in 1 phase

8B words are accessed in 2 phases:

- Process addresses of the **first 16** threads in a warp
- Process addresses of the **second 16** threads in a warp

16B words are accessed in 4 phases:

128 bit access size

- Each phase processes **8** consecutive threads of a warp

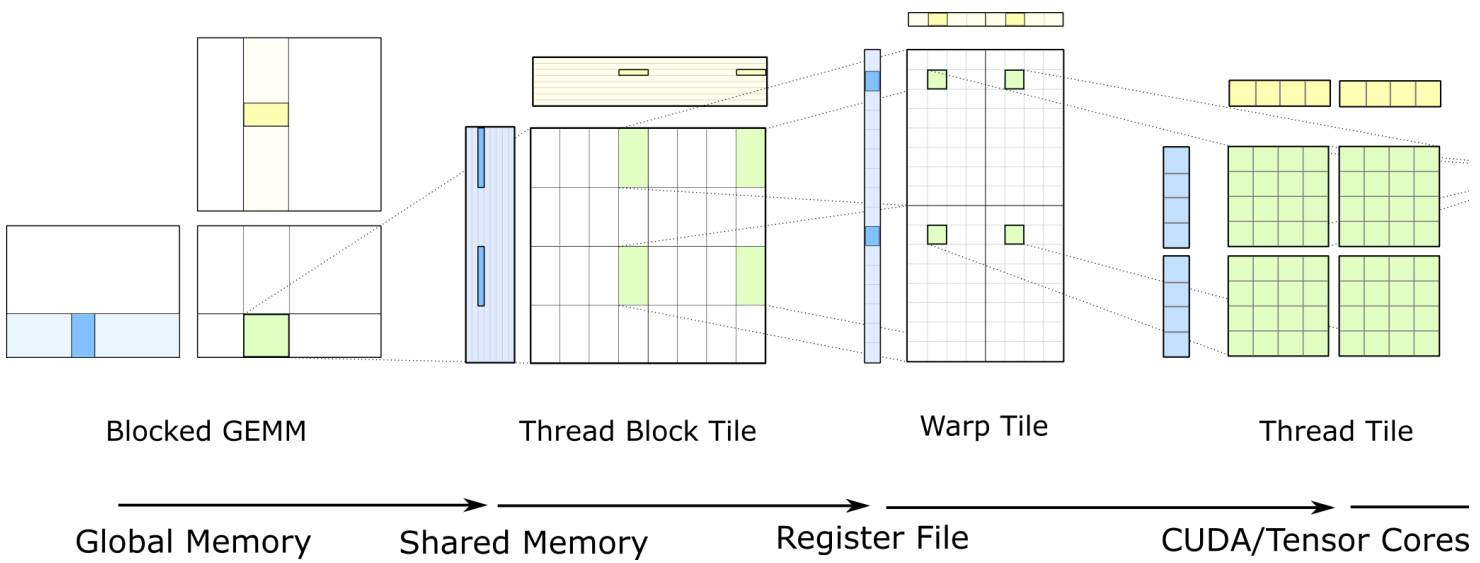
Slide borrowed from: Guillaume Thomas-Collignon and Paulius Micikevicius. "Volta Architecture and performance optimization." GTC 2018.
http://on-demand.gputechconf.com/gtc/2018/presentation/s81006-volta-architecture-and-performance-optimization.pdf

FEEDING THE DATA PATH

Efficiently storing and loading through shared memory

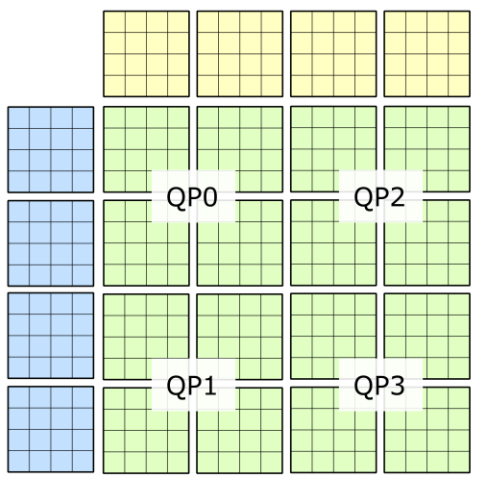
Must move data from shared memory to registers as efficiently as possible

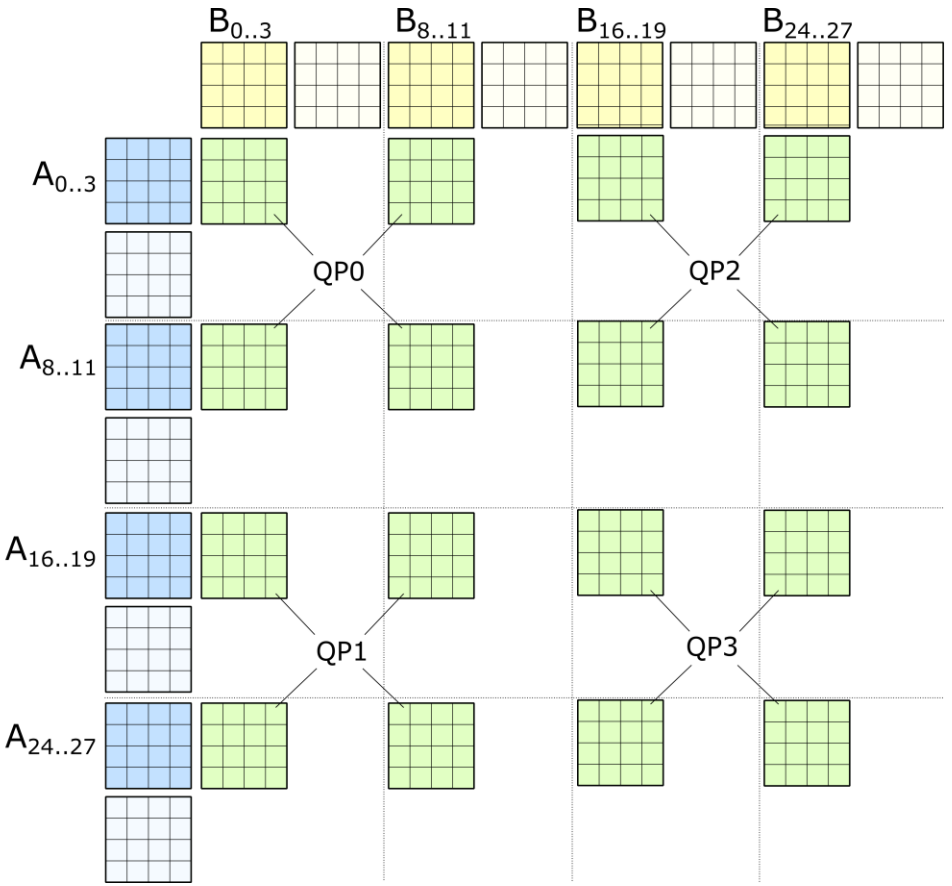
- 128 bit access size
- Conflict-free Shared Memory stores
- Conflict-free Shared Memory loads



MMA.SYNC GEMM: SPATIALLY INTERLEAVED

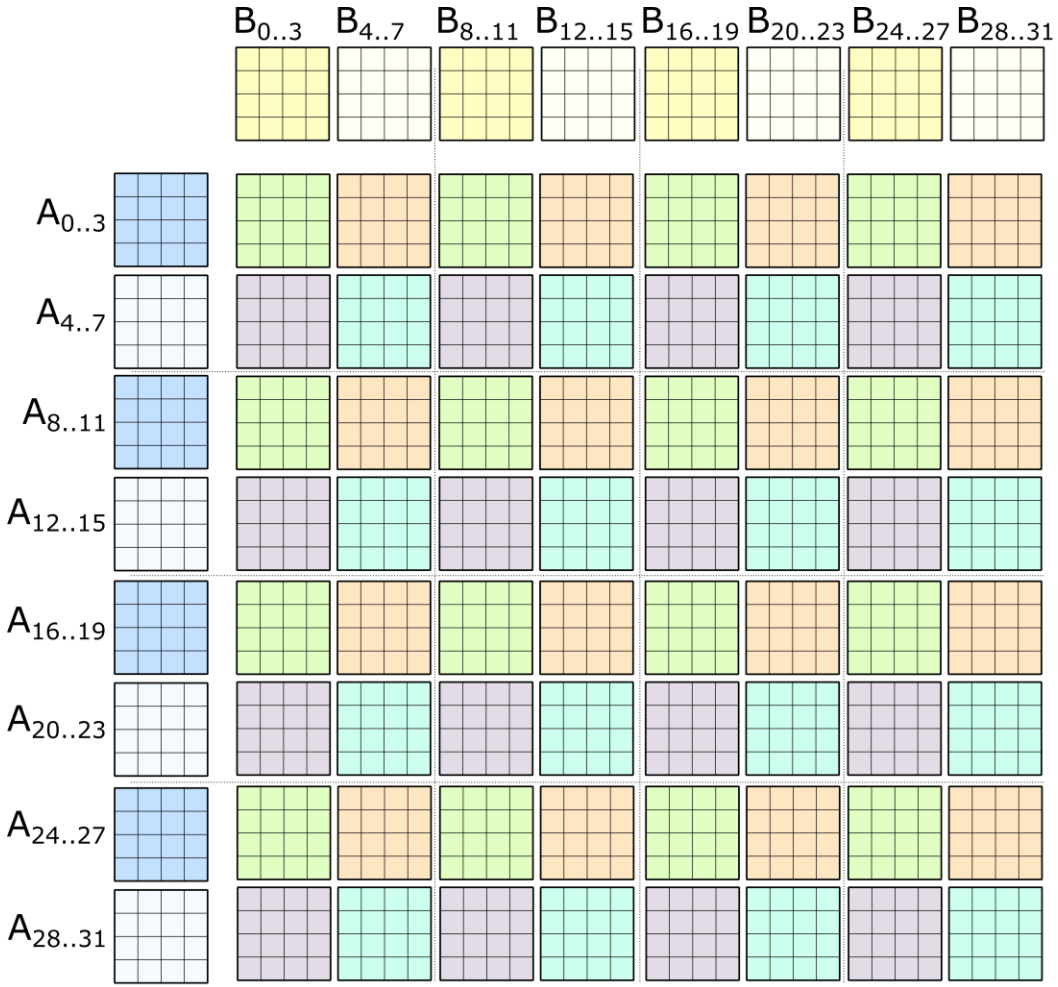
Accumulator tiles may not be contiguous

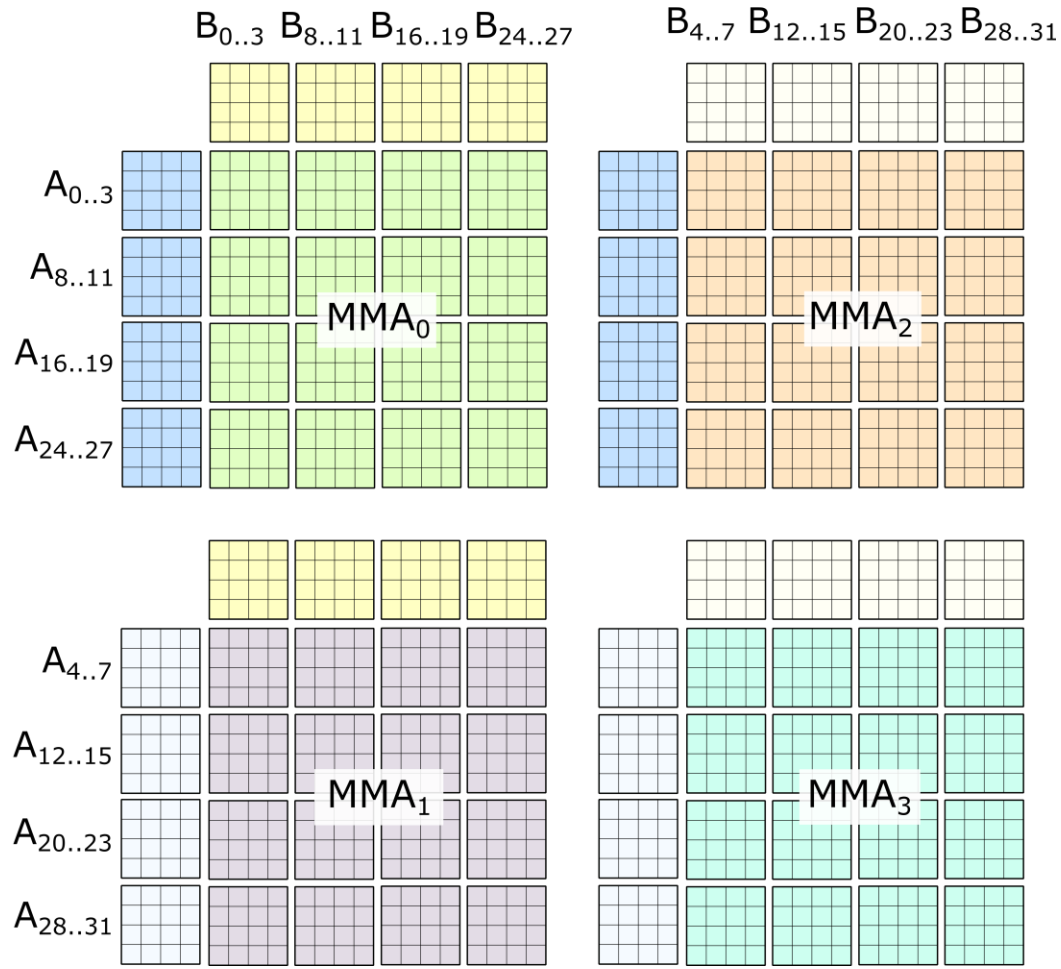




1 x mma.sync: 16-by-16-by-4

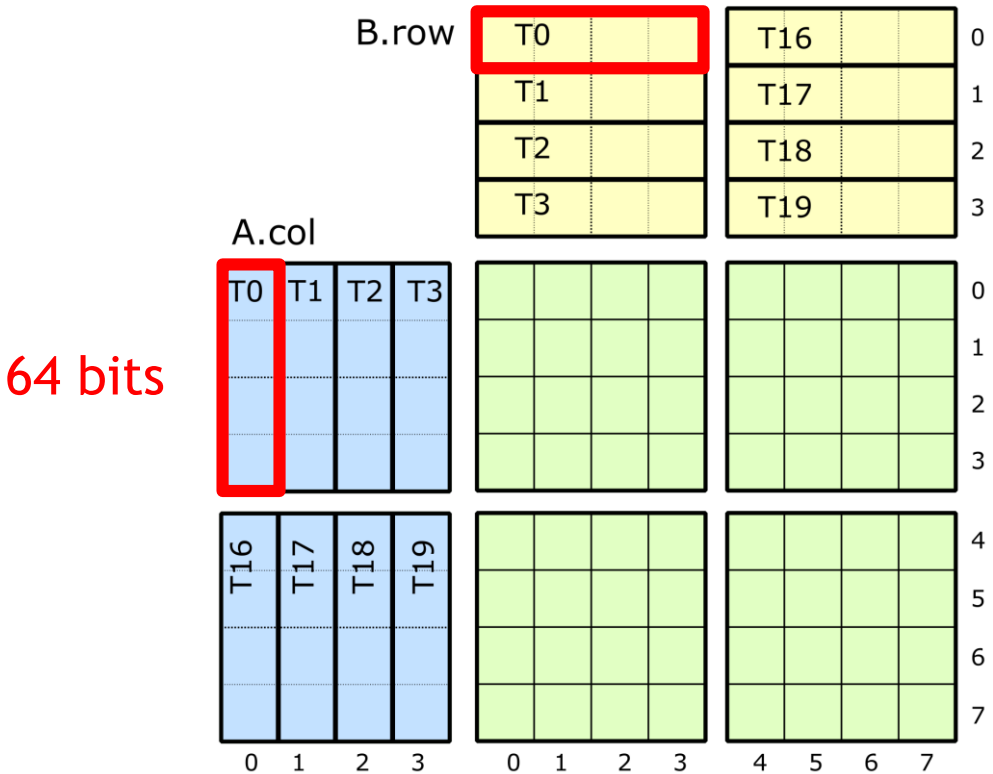
MMA.SYNC GEMM: SPATIALLY INTERLEAVED



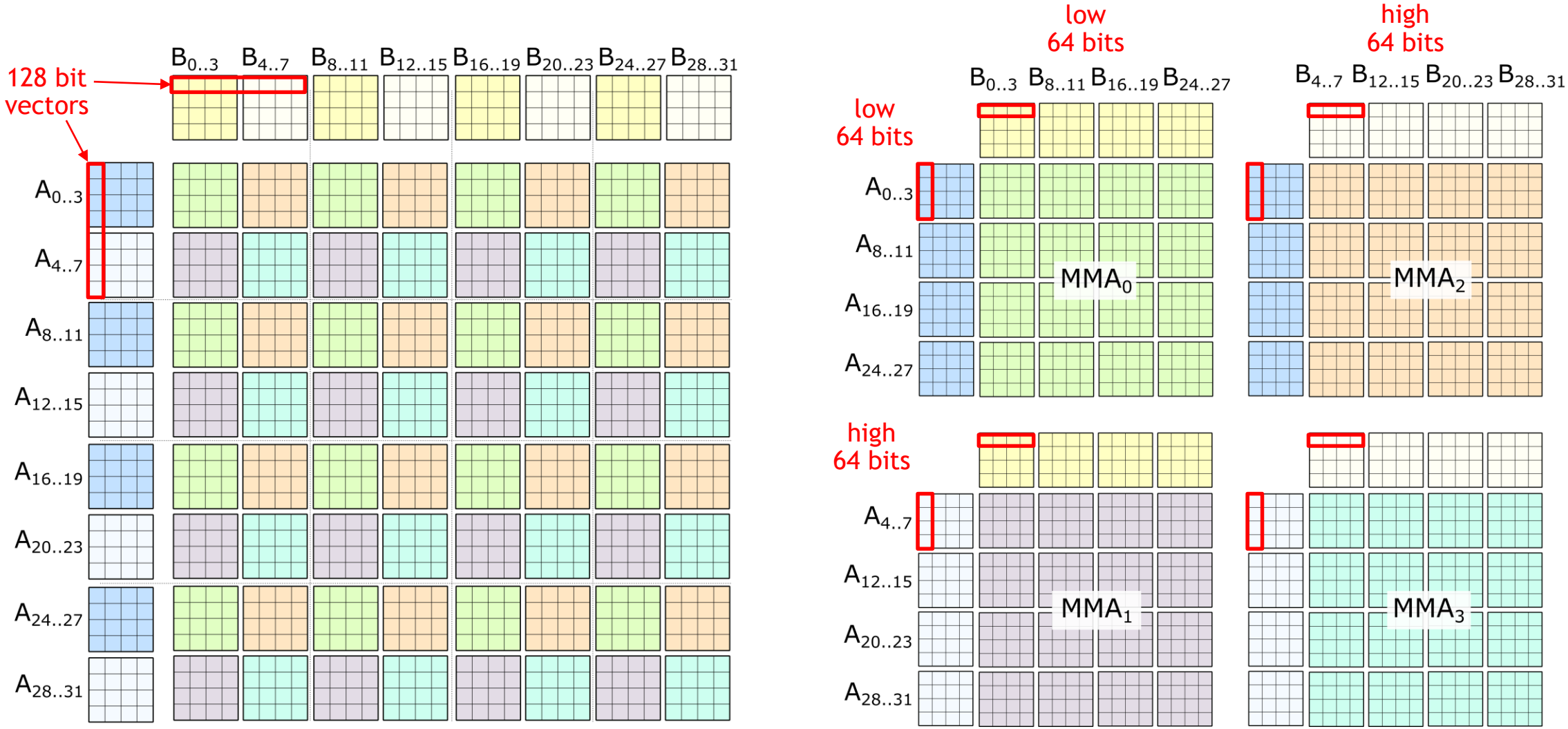


4 x mma.sync: 32-by-32-by-4 (spatially interleaved)

THREAD-DATA MAPPING - F16 MULTIPLICANDS



SPATIALLY INTERLEAVED: 128 BIT ACCESSES



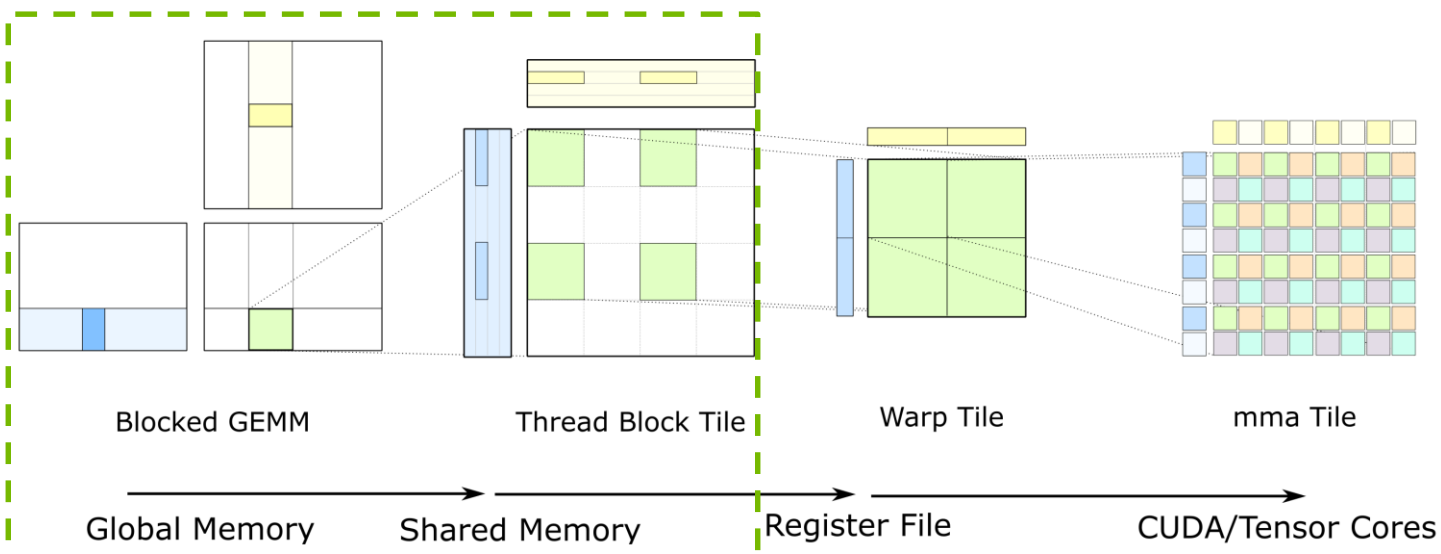
4 x mma.sync: 32-by-32-by-4 (spatially interleaved)

FEEDING THE DATA PATH

Efficiently storing and loading through shared memory

Must move data from shared memory to registers as efficiently as possible

- 128 bit access size
- **Conflict-free Shared Memory stores**
- Conflict-free Shared Memory loads

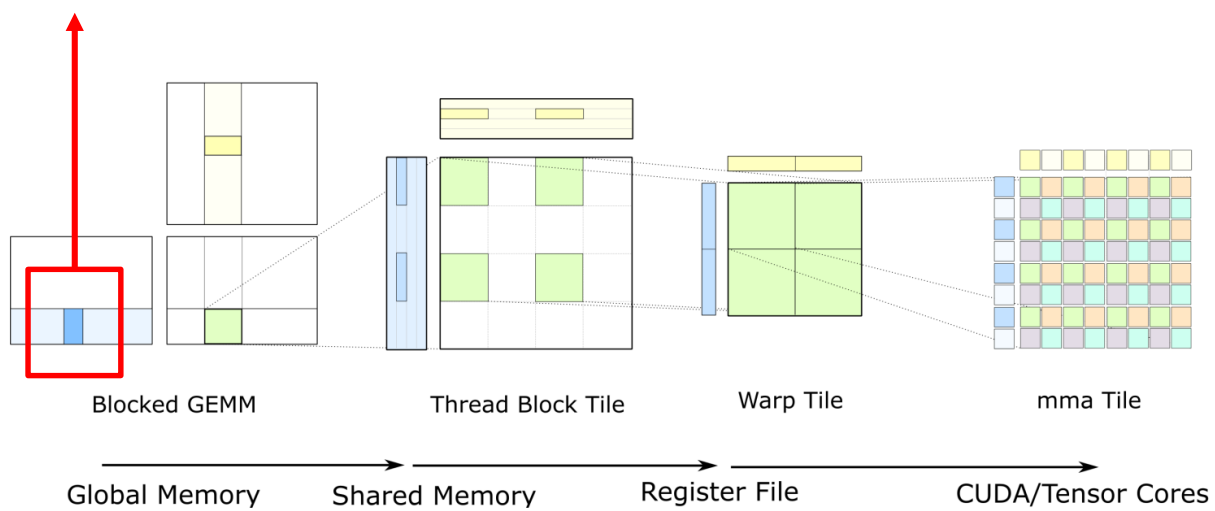


GLOBAL MEMORY (CANONICAL)

Striped over
8 x 4 threads

$A_{0..7, 0}$	$A_{8..15, 0}$	$A_{16..23, 0}$	$A_{24..31, 0}$	$A_{32..39, 0}$	$A_{40..47, 0}$	$A_{48..55, 0}$	$A_{56..63, 0}$
$A_{0..7, 1}$	$A_{8..15, 1}$	$A_{16..23, 1}$	$A_{24..31, 1}$	$A_{32..39, 1}$	$A_{40..47, 1}$	$A_{48..55, 1}$	$A_{56..63, 1}$
$A_{0..7, 2}$	$A_{8..15, 2}$	$A_{16..23, 2}$	$A_{24..31, 2}$	$A_{32..39, 2}$	$A_{40..47, 2}$	$A_{48..55, 2}$	$A_{56..63, 2}$
$A_{0..7, 3}$	$A_{8..15, 3}$	$A_{16..23, 3}$	$A_{24..31, 3}$	$A_{32..39, 3}$	$A_{40..47, 3}$	$A_{48..55, 3}$	$A_{56..63, 3}$

GMEM

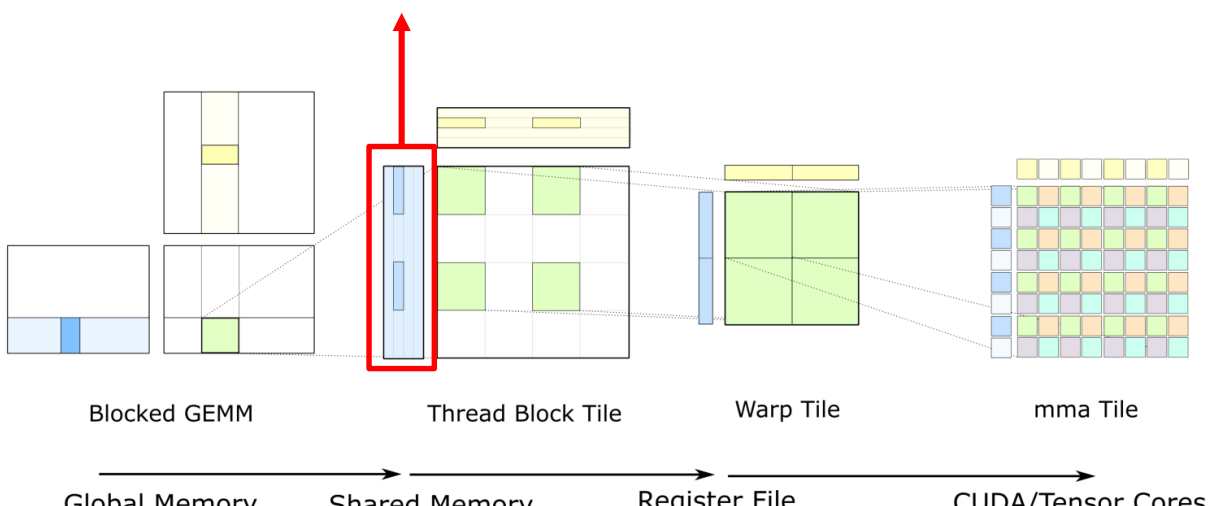


SHARED MEMORY (PERMUTED)

Permuted layout

$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$

SMEM



PERMUTED SHARED MEMORY TILES

Load
(128 bits per thread)

Store
(128 bits per thread)

Global Memory (column-major)

$A_{0..7, 0}$	$A_{8..15, 0}$	$A_{16..23, 0}$	$A_{24..31, 0}$	$A_{32..39, 0}$	$A_{40..47, 0}$	$A_{48..55, 0}$	$A_{56..63, 0}$
$A_{0..7, 1}$	$A_{8..15, 1}$	$A_{16..23, 1}$	$A_{24..31, 1}$	$A_{32..39, 1}$	$A_{40..47, 1}$	$A_{48..55, 1}$	$A_{56..63, 1}$
$A_{0..7, 2}$	$A_{8..15, 2}$	$A_{16..23, 2}$	$A_{24..31, 2}$	$A_{32..39, 2}$	$A_{40..47, 2}$	$A_{48..55, 2}$	$A_{56..63, 2}$
$A_{0..7, 3}$	$A_{8..15, 3}$	$A_{16..23, 3}$	$A_{24..31, 3}$	$A_{32..39, 3}$	$A_{40..47, 3}$	$A_{48..55, 3}$	$A_{56..63, 3}$

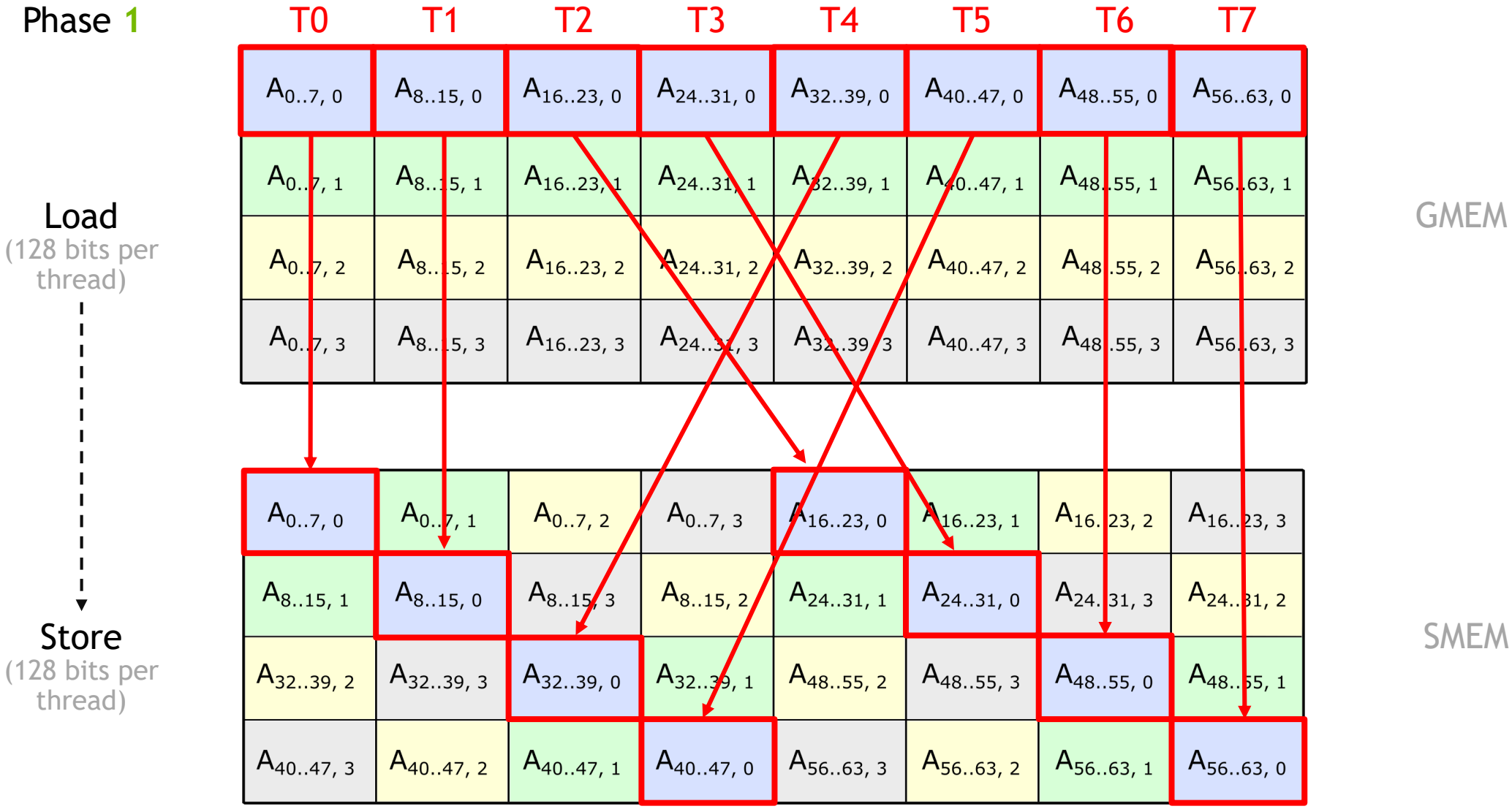
GMEM

Shared Memory (permuted)

$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$

SMEM

PERMUTED SHARED MEMORY TILES



PERMUTED SHARED MEMORY TILES

Phase 2

T8	T9	T10	T11	T12	T13	T14	T15
$A_{0..7, 0}$	$A_{8..15, 0}$	$A_{16..23, 0}$	$A_{24..31, 0}$	$A_{32..39, 0}$	$A_{40..47, 0}$	$A_{48..55, 0}$	$A_{56..63, 0}$
$A_{0..7, 1}$	$A_{8..15, 1}$	$A_{16..23, 1}$	$A_{24..31, 1}$	$A_{32..39, 1}$	$A_{40..47, 1}$	$A_{48..55, 1}$	$A_{56..63, 1}$
$A_{0..7, 2}$	$A_{8..15, 2}$	$A_{16..23, 2}$	$A_{24..31, 2}$	$A_{32..39, 2}$	$A_{40..47, 2}$	$A_{48..55, 2}$	$A_{56..63, 2}$
$A_{0..7, 3}$	$A_{8..15, 3}$	$A_{16..23, 3}$	$A_{24..31, 3}$	$A_{32..39, 3}$	$A_{40..47, 3}$	$A_{48..55, 3}$	$A_{56..63, 3}$

GMEM

Load
(128 bits per thread)

Store
(128 bits per thread)

$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$

SMEM

PERMUTED SHARED MEMORY TILES

Phase 3

	T16	T17	T18	T19	T20	T21	T22	T23
	$A_{0..7, 0}$	$A_{8..15, 0}$	$A_{16..23, 0}$	$A_{24..31, 0}$	$A_{32..39, 0}$	$A_{40..47, 0}$	$A_{48..55, 0}$	$A_{56..63, 0}$
	$A_{0..7, 1}$	$A_{8..15, 1}$	$A_{16..23, 1}$	$A_{24..31, 1}$	$A_{32..39, 1}$	$A_{40..47, 1}$	$A_{48..55, 1}$	$A_{56..63, 1}$
	$A_{0..7, 2}$	$A_{8..15, 2}$	$A_{16..23, 2}$	$A_{24..31, 2}$	$A_{32..39, 2}$	$A_{40..47, 2}$	$A_{48..55, 2}$	$A_{56..63, 2}$
	$A_{0..7, 3}$	$A_{8..15, 3}$	$A_{16..23, 3}$	$A_{24..31, 3}$	$A_{32..39, 3}$	$A_{40..47, 3}$	$A_{48..55, 3}$	$A_{56..63, 3}$

Load
(128 bits per thread)

GMEM

Store
(128 bits per thread)

$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$

SMEM

PERMUTED SHARED MEMORY TILES

Phase 4

T24	T25	T26	T27	T28	T29	T30	T31
$A_{0..7, 0}$	$A_{8..15, 0}$	$A_{16..23, 0}$	$A_{24..31, 0}$	$A_{32..39, 0}$	$A_{40..47, 0}$	$A_{48..55, 0}$	$A_{56..63, 0}$
$A_{0..7, 1}$	$A_{8..15, 1}$	$A_{16..23, 1}$	$A_{24..31, 1}$	$A_{32..39, 1}$	$A_{40..47, 1}$	$A_{48..55, 1}$	$A_{56..63, 1}$
$A_{0..7, 2}$	$A_{8..15, 2}$	$A_{16..23, 2}$	$A_{24..31, 2}$	$A_{32..39, 2}$	$A_{40..47, 2}$	$A_{48..55, 2}$	$A_{56..63, 2}$
$A_{0..7, 3}$	$A_{8..15, 3}$	$A_{16..23, 3}$	$A_{24..31, 3}$	$A_{32..39, 3}$	$A_{40..47, 3}$	$A_{48..55, 3}$	$A_{56..63, 3}$

Load
(128 bits per thread)

GMEM

Store
(128 bits per thread)

$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$

SMEM

POINTER OFFSETS FOR PERMUTED SHARED MEMORY

Global Memory (column-major)

$A_{0..7, 0}$	$A_{8..15, 0}$	$A_{16..23, 0}$	$A_{24..31, 0}$	$A_{32..39, 0}$	$A_{40..47, 0}$	$A_{48..55, 0}$	$A_{56..63, 0}$
$A_{0..7, 1}$	$A_{8..15, 1}$	$A_{16..23, 1}$	$A_{24..31, 1}$	$A_{32..39, 1}$	$A_{40..47, 1}$	$A_{48..55, 1}$	$A_{56..63, 1}$
$A_{0..7, 2}$	$A_{8..15, 2}$	$A_{16..23, 2}$	$A_{24..31, 2}$	$A_{32..39, 2}$	$A_{40..47, 2}$	$A_{48..55, 2}$	$A_{56..63, 2}$
$A_{0..7, 3}$	$A_{8..15, 3}$	$A_{16..23, 3}$	$A_{24..31, 3}$	$A_{32..39, 3}$	$A_{40..47, 3}$	$A_{48..55, 3}$	$A_{56..63, 3}$

```
int lane = threadIdx.x % 32;

int c = lane % 8;
int s = lane / 8;

int gmem_offset = c + s * lda;
```

Shared Memory (permuted)

$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$

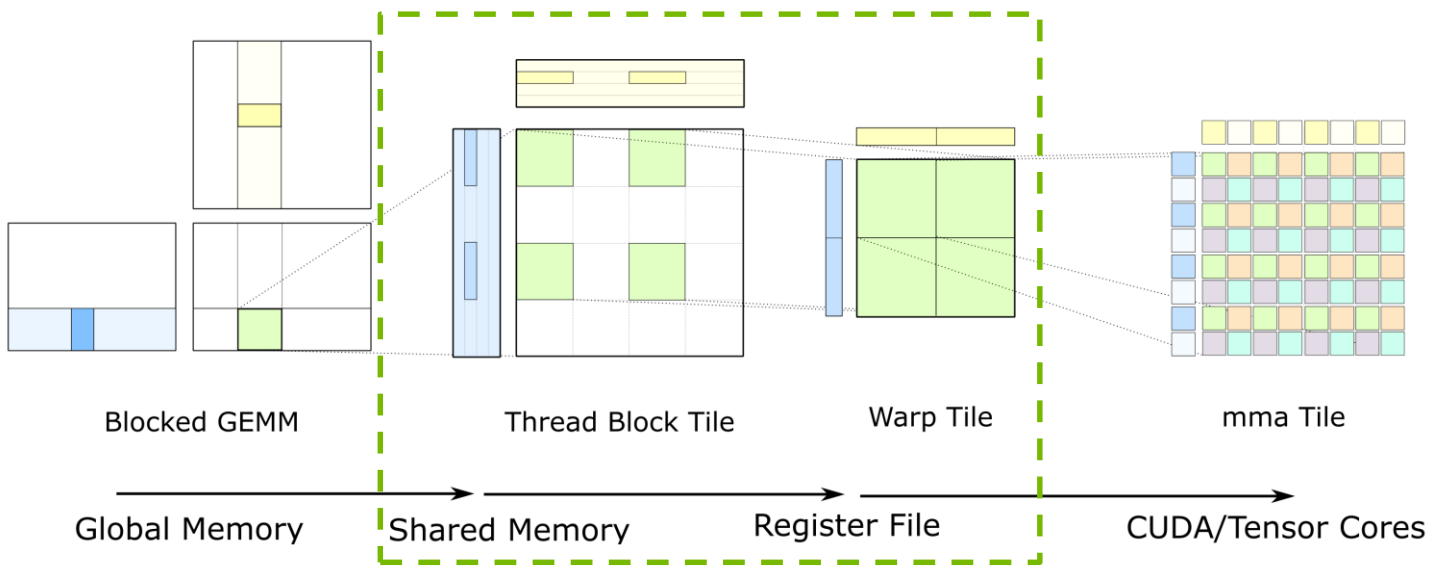
```
int lane = threadIdx.x % 32;

int c = lane % 8;
int s = lane / 8;

int smem_row = (c & 1) | ((c >> 1) & 2);
int bank = ((c << 1) & 4) | s ^ smem_row;

int smem_offset = smem_row * ldm_smem + bank;
```

FEEDING THE DATA PATH

Efficiently storing and loading through shared memory

Must move data from shared memory to registers as efficiently as possible

- 128 bit access size
- Conflict-free Shared Memory stores
- **Conflict-free Shared Memory loads**

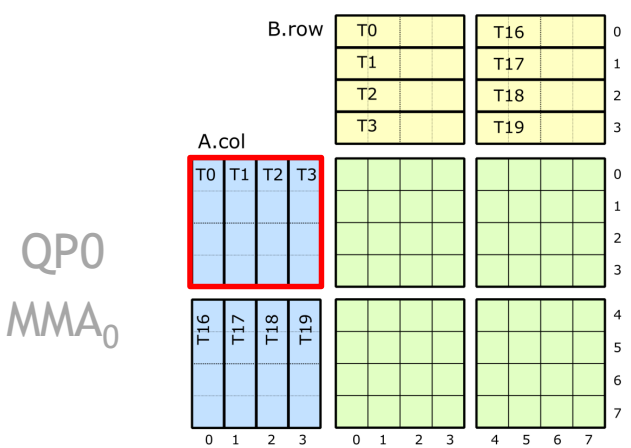


CONFLICT-FREE SHARED MEMORY LOADS

Phase 1 QP0

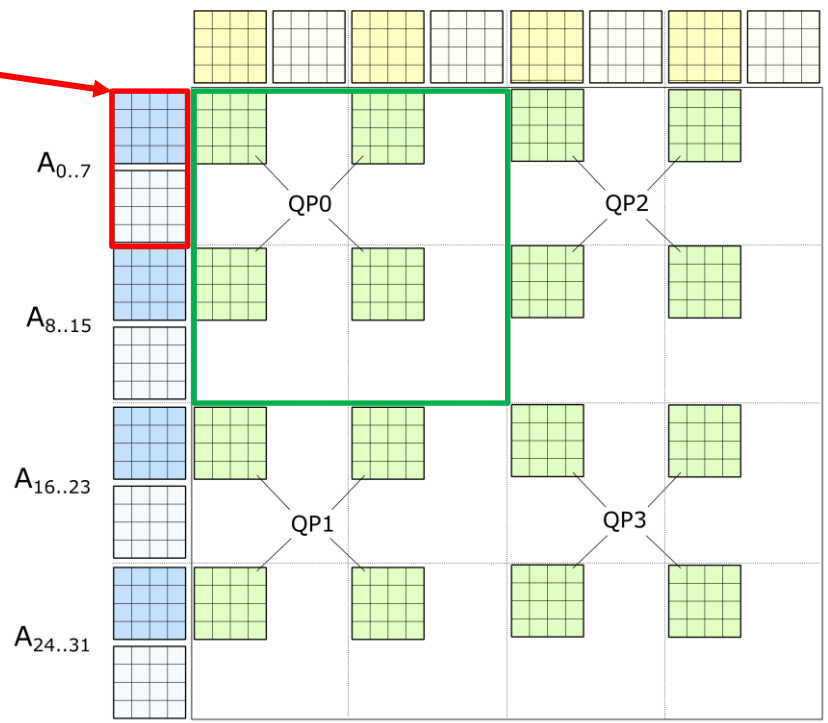
T0 T1 T2 T3

$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$



QP0

MMA_0



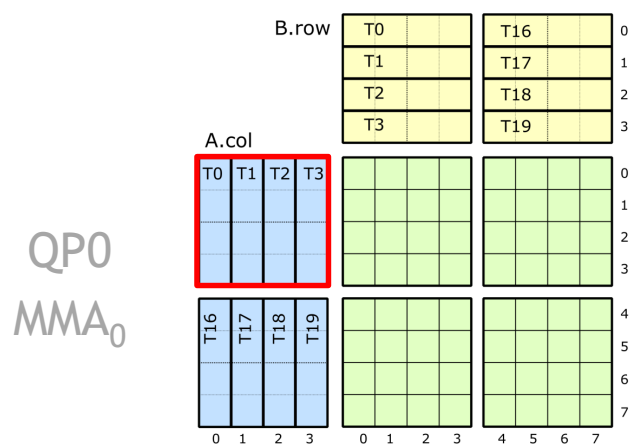
CONFLICT-FREE SHARED MEMORY LOADS

Phase 1

QP0

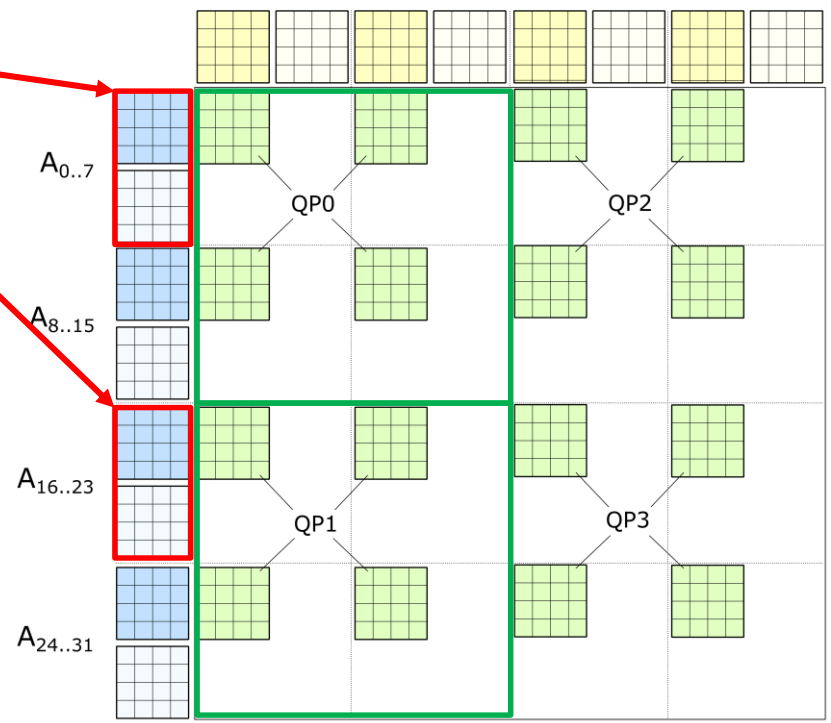
QP1

T0	T1	T2	T3	T4	T5	T6	T7
$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$



QP0

MMA_0



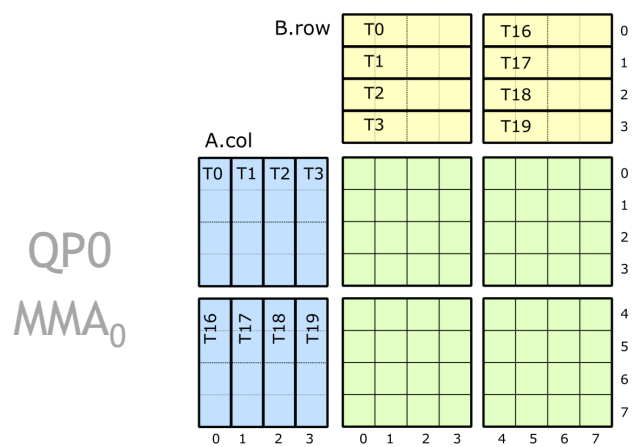
CONFLICT-FREE SHARED MEMORY LOADS

Phase 2

QP2

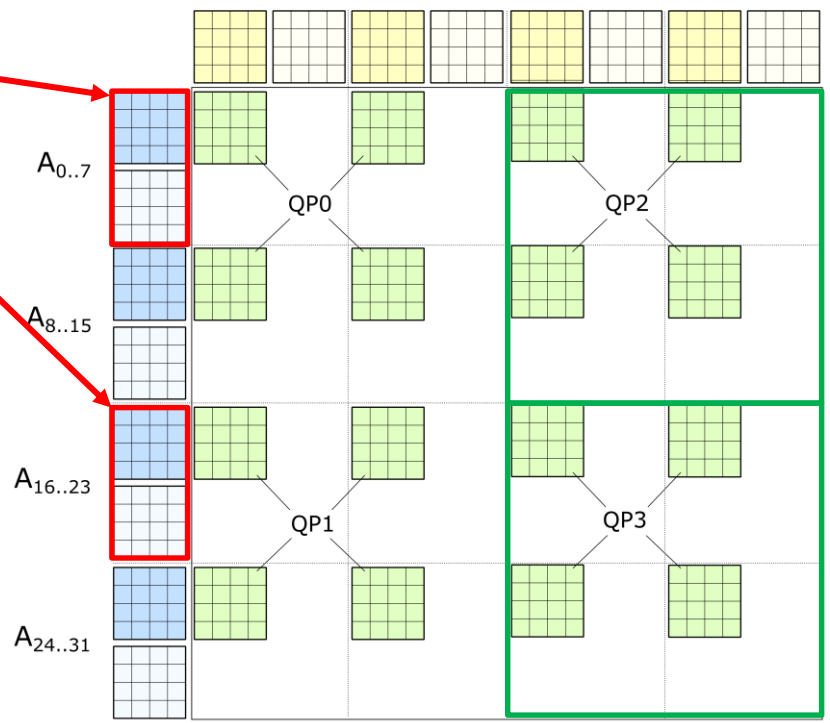
QP3

T8	T9	T10	T11	T12	T13	T14	T15
$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$



QP0

MMA_0



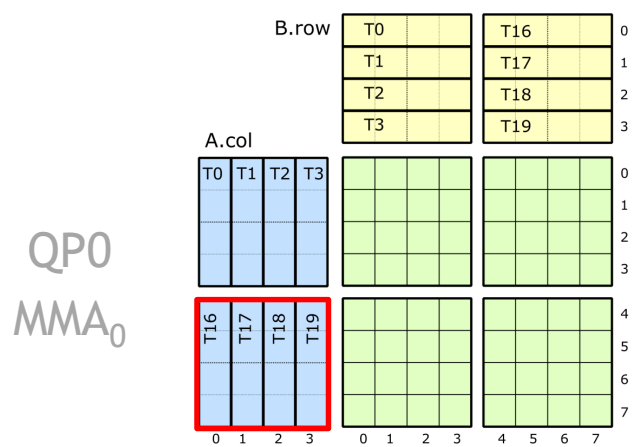
CONFLICT-FREE SHARED MEMORY LOADS

Phase 3

QP0

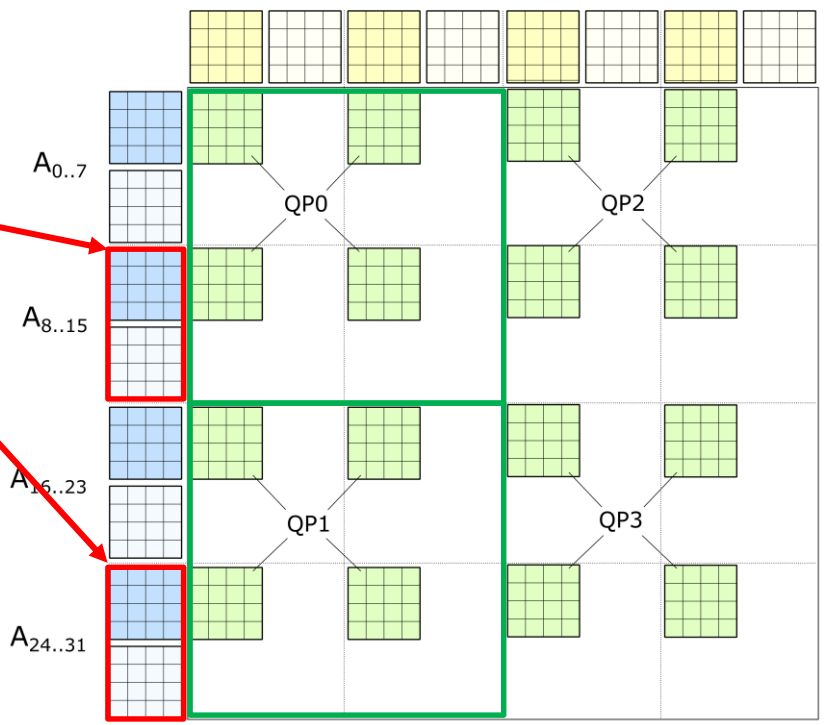
QP1

T17	T16	T19	T18	T21	T20	T23	T22
$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$



QP0

MMA_0



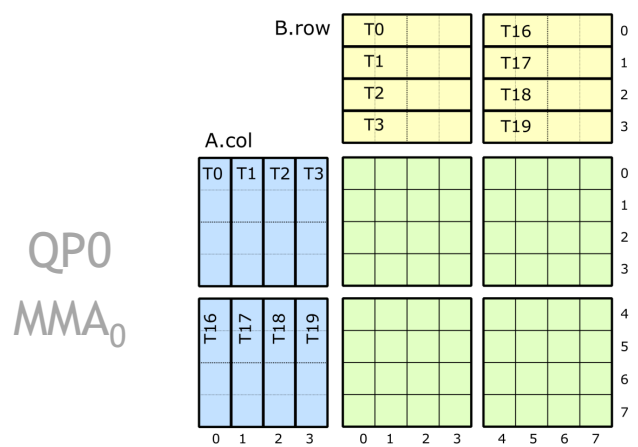
CONFLICT-FREE SHARED MEMORY LOADS

Phase 4

QP2

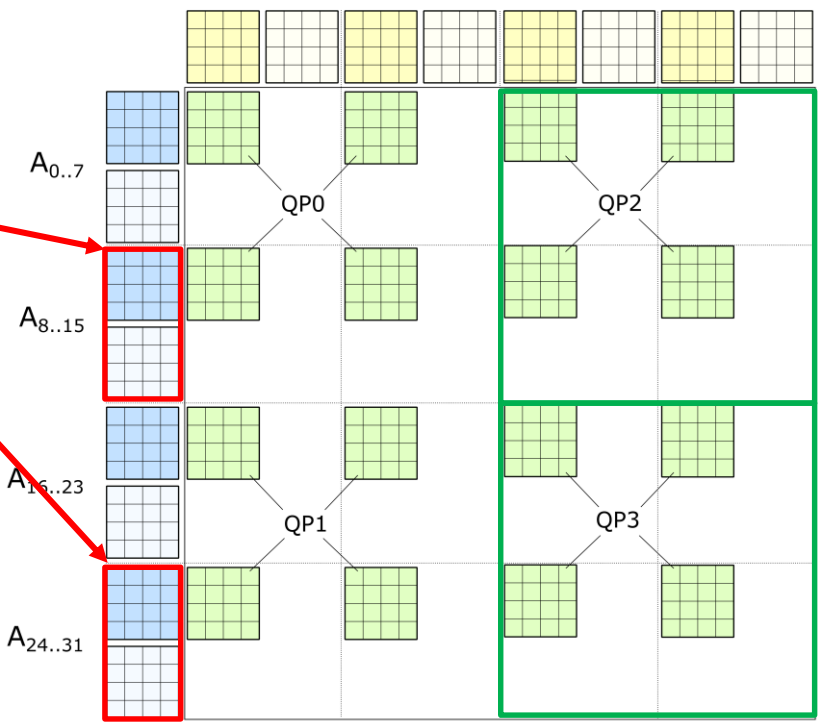
QP3

T25	T24	T27	T26	T29	T28	T31	T30
$A_{0..7, 0}$	$A_{0..7, 1}$	$A_{0..7, 2}$	$A_{0..7, 3}$	$A_{16..23, 0}$	$A_{16..23, 1}$	$A_{16..23, 2}$	$A_{16..23, 3}$
$A_{8..15, 1}$	$A_{8..15, 0}$	$A_{8..15, 3}$	$A_{8..15, 2}$	$A_{24..31, 1}$	$A_{24..31, 0}$	$A_{24..31, 3}$	$A_{24..31, 2}$
$A_{32..39, 2}$	$A_{32..39, 3}$	$A_{32..39, 0}$	$A_{32..39, 1}$	$A_{48..55, 2}$	$A_{48..55, 3}$	$A_{48..55, 0}$	$A_{48..55, 1}$
$A_{40..47, 3}$	$A_{40..47, 2}$	$A_{40..47, 1}$	$A_{40..47, 0}$	$A_{56..63, 3}$	$A_{56..63, 2}$	$A_{56..63, 1}$	$A_{56..63, 0}$



QP0

MMA_0

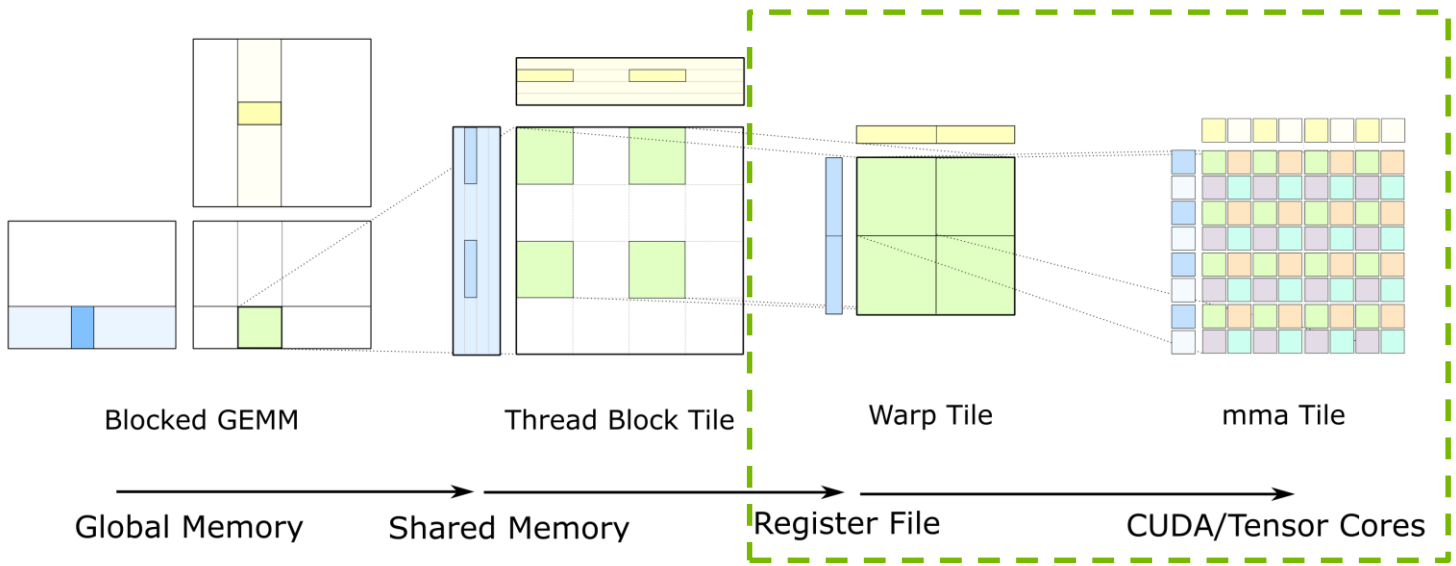


FEEDING THE DATA PATH

Efficiently storing and loading through shared memory

Must move data from shared memory to registers as efficiently as possible

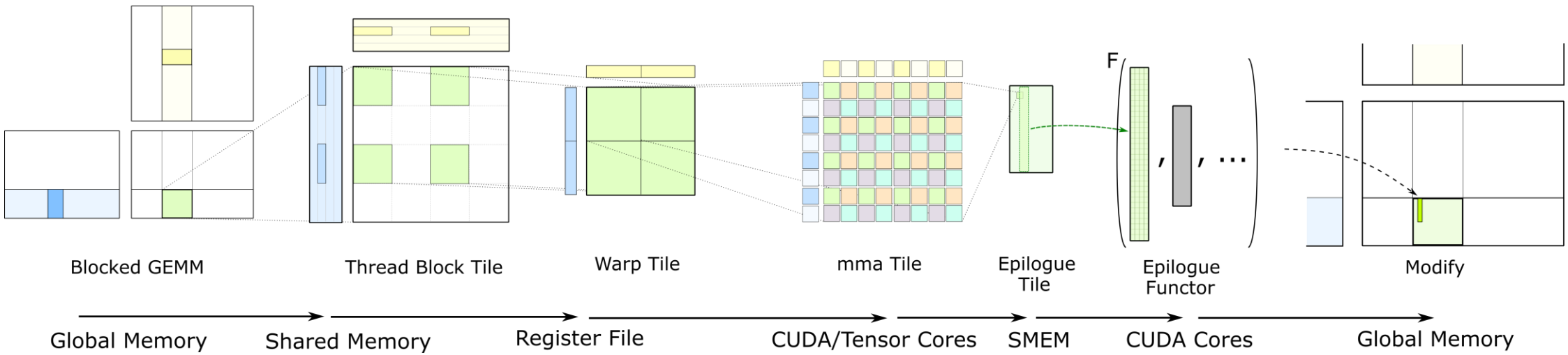
- **128 bit access size**
- **Conflict-free Shared Memory stores**
- **Conflict-free Shared Memory loads**



CUTLASS 1.3

CUTLASS

CUDA C++ Template Library for Deep Learning



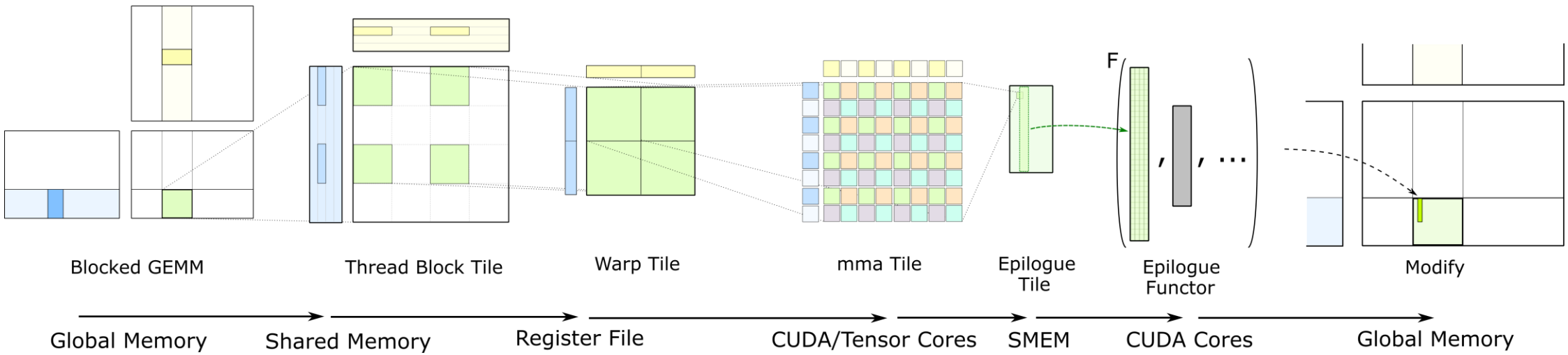
CUTLASS template library for GEMM computations

- Blocked structure to maximize data reuse
- Software pipelined to hide latency
- Conflict-free Shared Memory access to maximize data throughput

See CUTLASS GTC 2018 talk.

CUTLASS 1.3

Reusable components targeting Volta Tensor Cores



GlobalLoadIterator
Transformer
SharedStoreIterator

SharedTileLoadIterator

MatrixMultiply
mma.sync

Transformer
SharedStoreIterator
SharedLoaditerator

Functor

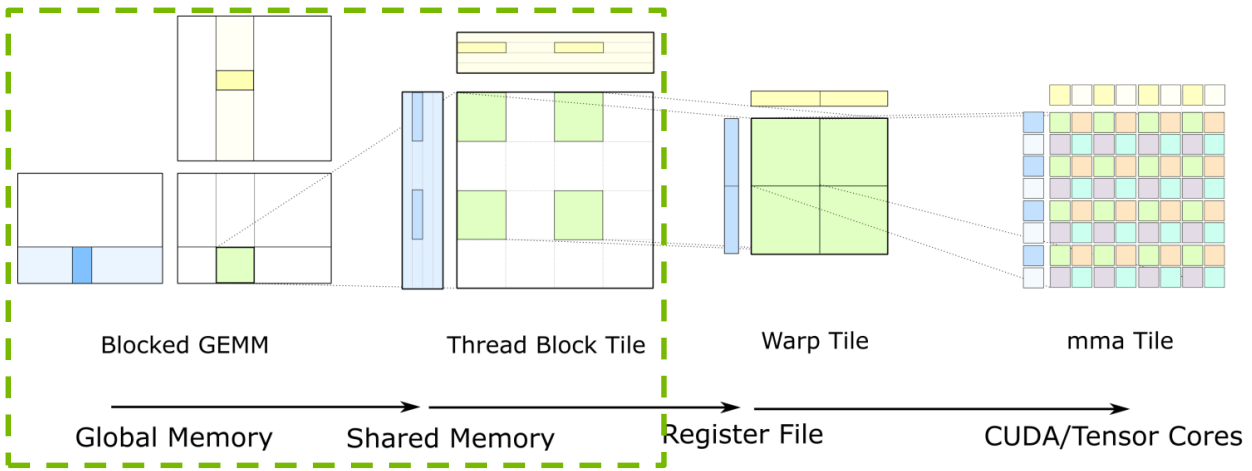
GlobalLoadIterator
GlobalStoreIterator

GlobalLoadStream

Warp Matrix Multiply

Epilogue

STORING TO SHARED MEMORY



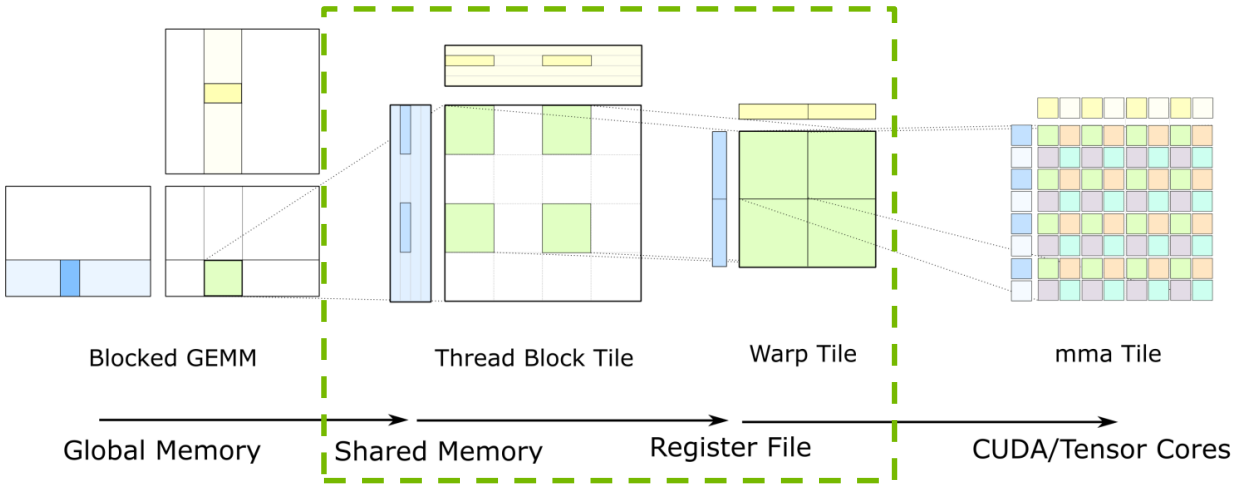
cutlass/gemm/volta884_multiplicand.h

```
// Defines iterators for loading and storing multiplicands
template <
  /// Identifies multiplicand of GEMM (A or B)
  GemmOperand::Kind Operand,
  /// Specifies layout of data in source memory
  MatrixLayout::Kind Layout,
  /// Specifies threadblock tile shape
  typename Tile,
  /// Specifies warp tile shape
  typename WarpTile,
  /// Specifies the number of participating warps
  int WarpCount,
  /// Specifies the delta between warp tiles
  typename WarpDelta
>
struct Volta884Multiplicand {

  //
  // Thread-block load iterator (canonical matrix layout)
  //
  typedef ...  LoadIterator;

  //
  // Thread-block store iterator (permuted SMEM layout)
  //
  typedef ...  StoreIterator;

  //
  // Warp-level load iterator
  //
  typedef ... WarpLoadIterator;
};
```

CUTLASS Tile Iterators to transform:

- Global Memory: Canonical matrix layout ➔ Shared Memory: permuted shared memory layout

LOADING FROM SHARED MEMORY



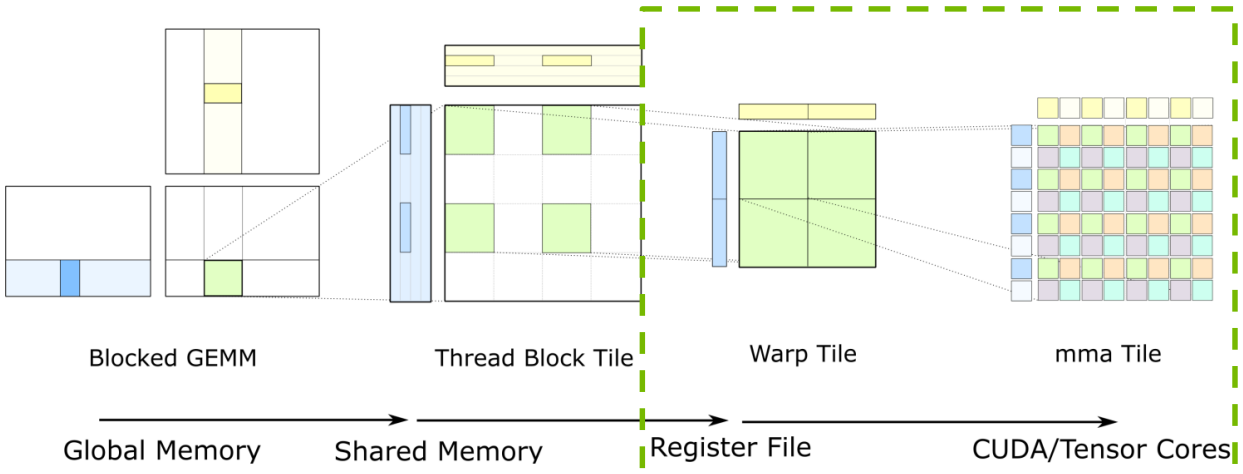
cutlass/gemm/volta884_multiplicand.h

```
// Defines iterators for loading and storing multiplicands
template <
  /// Identifies multiplicand of GEMM (A or B)
  GemmOperand::Kind Operand,
  /// Specifies layout of data in source memory
  MatrixLayout::Kind Layout,
  /// Specifies threadblock tile shape
  typename Tile,
  /// Specifies warp tile shape
  typename WarpTile,
  /// Specifies the number of participating warps
  int WarpCount,
  /// Specifies the delta between warp tiles
  typename WarpDelta
>
struct Volta884Multiplicand {

  //
  // Thread-block load iterator (canonical matrix layout)
  //
  typedef ...  LoadIterator;

  //
  // Thread-block store iterator (permuted SMEM layout)
  //
  typedef ...  StoreIterator;

  //
  // Warp-level load iterator
  //
  typedef ... WarpLoadIterator;
};
```

CUTLASS Tile Iterators to transform:

- Shared Memory: permuted shared memory layout ➔ Register File: mma.sync thread-data mapping

EXECUTING MMA.SYNC



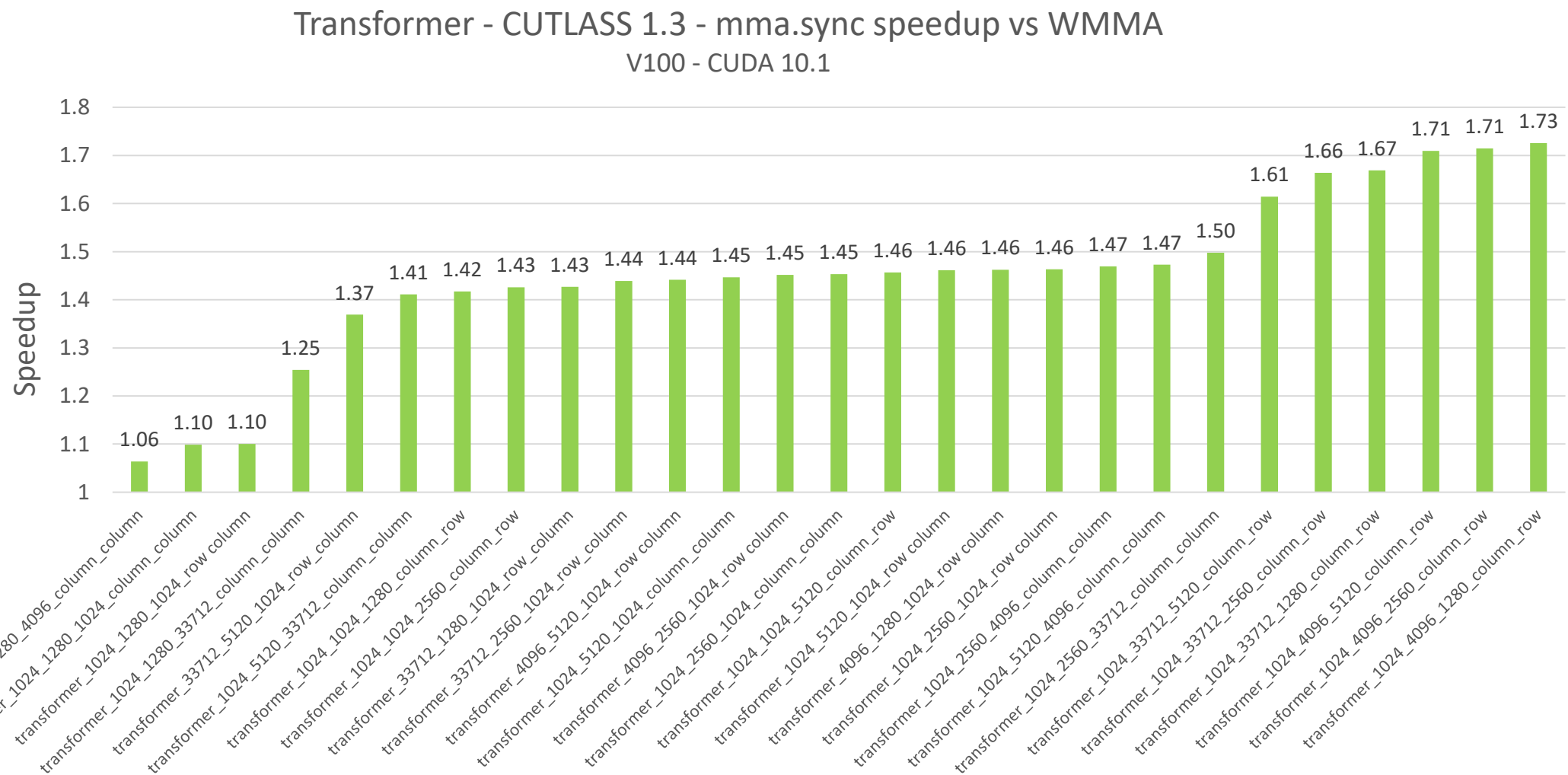
cutlass/gemm/volta884_multiply_add.h

```
template <
  /// Shape of a warp-level GEMM (K-by-N-by-M)
  typename WarpGemmShape_,
  /// Layout of A multiplicand
  MatrixLayout::Kind LayoutA,
  /// Data type of A multiplicand
  typename ScalarA,
  /// Layout of B multiplicand
  MatrixLayout::Kind LayoutB,
  /// Data type of A multiplicand
  typename ScalarB,
  /// Data type of accumulators
  typename ScalarC,
  /// Whether infinite results are saturated to +-MAX_FLOAT
  bool SatFinite = false
>
struct Volta884MultiplyAdd {

  //
  // Multiply : d = (-)a*b + c.
  //
  CUTLASS_DEVICE
  void multiply_add(
    FragmentA const& A,
    FragmentB const& B,
    Accumulators const& C,
    Accumulators& D,
    bool negate = false) {

    ...
  }
};
```

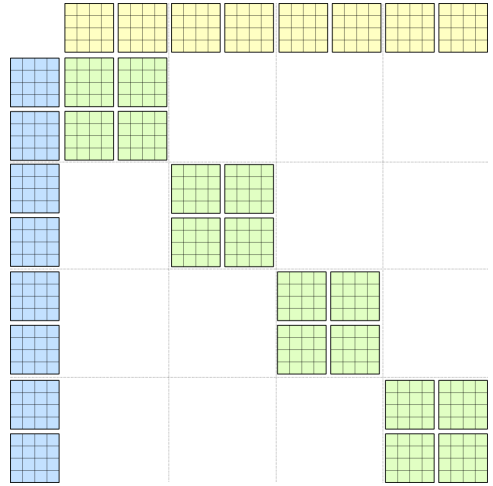
CUTLASS Warp-scoped matrix multiply

- Register File: mma.sync thread-data mapping ➔ Tensor Cores: mma.sync

SPEEDUP RELATIVE TO WMMA



CONCLUSION



Volta Tensor Cores directly programmable in CUDA 10.1

- Complements WMMA API
- Direct access: mma.sync instruction for Volta Architecture

CUTLASS 1.3 (March 2019)

- CUDA C++ Template Library for Deep Learning
- Reusable components:
  - mma.sync for Volta Tensor Cores
  - Storing and loading from permuted shared memory
  - Efficient epilogue for updating output matrix
- New kernels:
  - Real- and complex-valued mixed precision GEMMs targeting Tensor Cores
  - Parallelized reductions for mma.sync GEMM (first added in CUTLASS 1.2)

https://github.com/NVIDIA/cutlass

REFERENCES

CUTLASS source code: https://github.com/NVIDIA/cutlass

Volta Tensor Cores in CUDA

- mma.sync: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#warp-level-matrix-instructions-mma
- Matrix fragments: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#warp-level-matrix-fragment-mma

GEMM resources

- CUTLASS Parallel for All blog post
- GTC 2018 CUTLASS talk [video recording]

QUESTIONS?

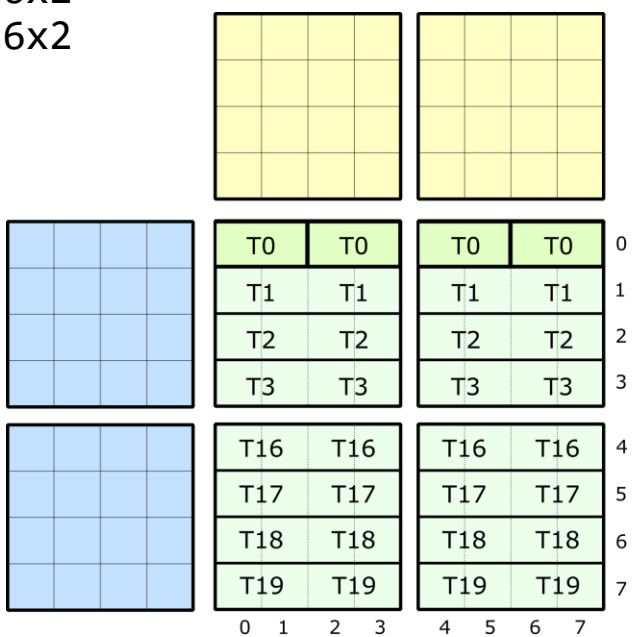
EXTRA MATERIAL

THREAD-DATA MAPPING – F16 ACCUMULATION

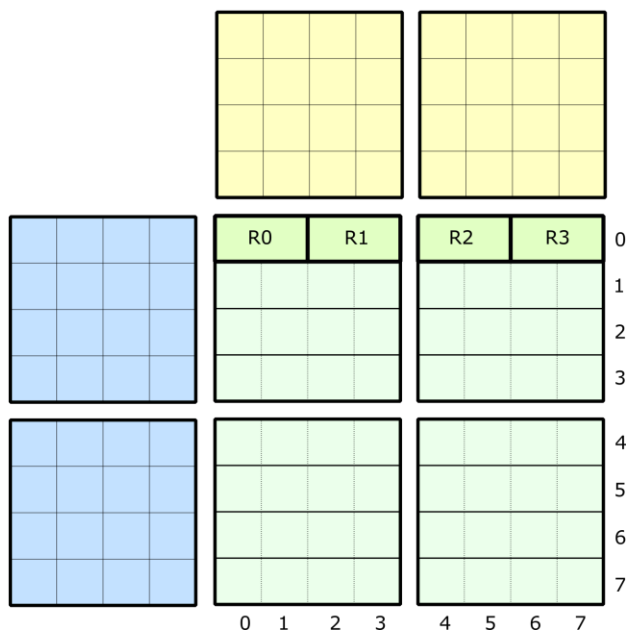
Accumulators distributed among threads (QP0 shown)

```
mma.sync.aligned.m8n8k4.alayout.blayout.dtype.f16.f16.ctype   d, a, b, c;

.ctype   = {.f16, .f32};
.dtype   = {.f16, .f32};
```

d: 4 x .f16x2
c: 4 x .f16x2



Quad Pair 0

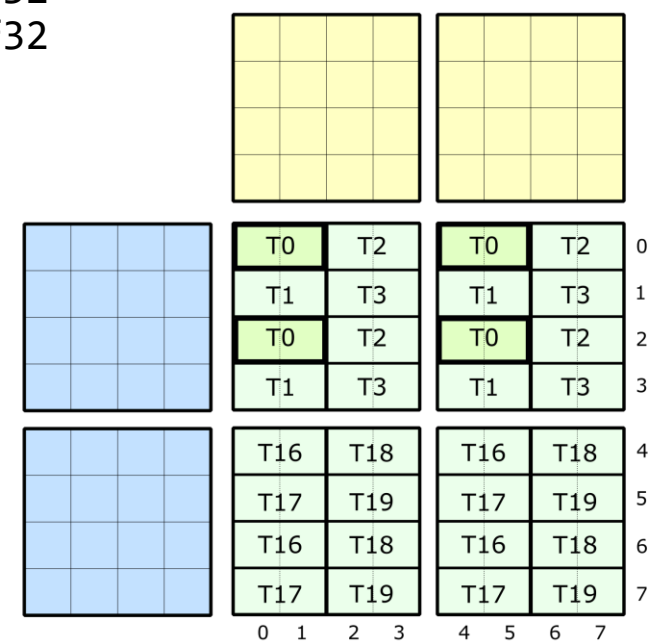


Thread 0

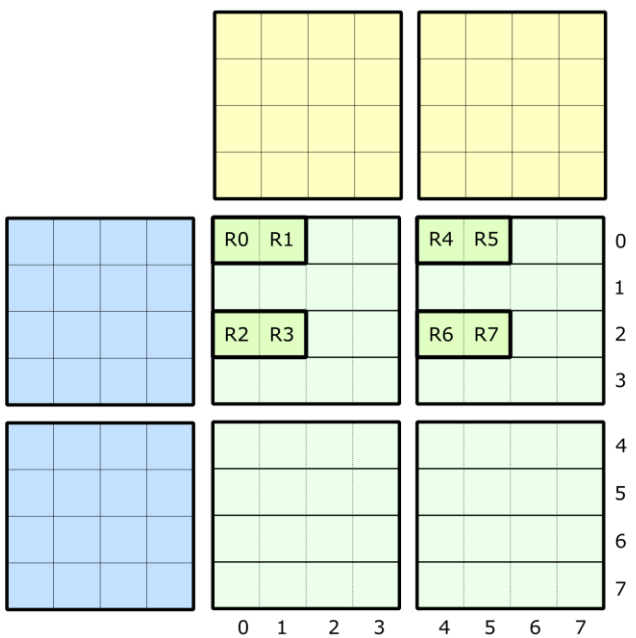
THREAD-DATA MAPPING – F32 ACCUMULATION

Accumulators distributed among threads (QP0 shown)

```
mma.sync.aligned.m8n8k4.alayout.blayout.dtype.f16.f16.ctype   d, a, b, c;

.ctype   = {.f16, .f32};
.dtype   = {.f16, .f32};

d: 8 x .f32
c: 8 x .f32
```



Quad Pair 0



Thread 0