PROGRAMMING TENSOR CORES:
NATIVE VOLTA TENSOR CORES WITH CUTLASS

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

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

![Graph showing performance relative to cuBLAS](https://github.com/NVIDIA/cutlass)
This talk is about Volta Tensor Cores.

Warp-synchronous Matrix Multiply Accumulate (WMMA API) portable abstraction layer for Tensor Cores

Direct access to Volta Tensor Cores

Volta Tensor Cores - Performance Relative to cuBLAS
CUTLASS 1.3 - CUDA 10.1 - V100

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

\[ D = A \times B + C \]

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

Warp-synchronous:

- Four independent 8-by-8-by-4 matrix multiply-accumulate operations
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
  - $A_{0..7}$: QP1, QP3
- $B_{0..7}$: QP0, QP1
  - $B_{0..7}$: QP2, QP3

1 x mma.sync: 16-by-16-by-4
VOLTA MMA.SYNC  \[ D = A \ast B + C \]

PTX Syntax

```plaintext
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};
```

\[
\begin{array}{l}
\text{d:} \quad 8 \times \text{.dtype} \\
\text{a:} \quad 4 \times \text{.f16} \\
\text{b:} \quad 4 \times \text{.f16} \\
\text{c:} \quad 8 \times \text{.ctype}
\end{array}
\]

Note: \text{.f16} elements must be packed into \text{.f16x2}

THREAD-DATA MAPPING - F16 MULTIPICANDS

Distributed among threads in quad pair (QP0 shown)

\[
\text{mma.sync.aligned.m8n8k4.} \text{.alayout.blayout.dtype.f16.f16.ctype} \quad d, a, b, c;
\]

\[
.alayout = \{ \text{.row, .col}\};
\]

\[
.blayout = \{ \text{.row, .col}\};
\]

\[
a: 2 \times .f16x2
\]
\[
b: 2 \times .f16x2
\]
FEEDING THE DATA PATH
FEEDING THE DATA PATH
Efficiently storing and loading through shared memory

See [CUTLASS GTC 2018](https://cutlass.epPARTMENT) 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:

• Each phase processes 8 consecutive threads of a warp

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
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

COL-ROW ("NT")

64 bits
SPATIALLY INTERLEAVED: 128 BIT ACCESSES

128 bit vectors

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)**

<table>
<thead>
<tr>
<th>A_{0\ldots7}, 0</th>
<th>A_{8\ldots15}, 0</th>
<th>A_{16\ldots23}, 0</th>
<th>A_{24\ldots31}, 0</th>
<th>A_{32\ldots39}, 0</th>
<th>A_{40\ldots47}, 0</th>
<th>A_{48\ldots55}, 0</th>
<th>A_{56\ldots63}, 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_{0\ldots7}, 1</td>
<td>A_{8\ldots15}, 1</td>
<td>A_{16\ldots23}, 1</td>
<td>A_{24\ldots31}, 1</td>
<td>A_{32\ldots39}, 1</td>
<td>A_{40\ldots47}, 1</td>
<td>A_{48\ldots55}, 1</td>
<td>A_{56\ldots63}, 1</td>
</tr>
<tr>
<td>A_{0\ldots7}, 2</td>
<td>A_{8\ldots15}, 2</td>
<td>A_{16\ldots23}, 2</td>
<td>A_{24\ldots31}, 2</td>
<td>A_{32\ldots39}, 2</td>
<td>A_{40\ldots47}, 2</td>
<td>A_{48\ldots55}, 2</td>
<td>A_{56\ldots63}, 2</td>
</tr>
<tr>
<td>A_{0\ldots7}, 3</td>
<td>A_{8\ldots15}, 3</td>
<td>A_{16\ldots23}, 3</td>
<td>A_{24\ldots31}, 3</td>
<td>A_{32\ldots39}, 3</td>
<td>A_{40\ldots47}, 3</td>
<td>A_{48\ldots55}, 3</td>
<td>A_{56\ldots63}, 3</td>
</tr>
</tbody>
</table>

Striped over 8 x 4 threads

[Diagram showing the memory layout and access patterns for blocked GEMM, thread block tile, warp tile, and mma tile within the global memory.]
SHARED MEMORY (PERMUTED)

Permuted layout

\[
\begin{array}{cccccccc}
A_{0..7, 0} & A_{0..7, 1} & A_{0..7, 2} & A_{0..7, 3} & A_{16..23, 0} & A_{16..23, 1} & A_{16..23, 2} & A_{16..23, 3} \\
A_{8..15, 1} & A_{8..15, 0} & A_{8..15, 3} & A_{8..15, 2} & A_{24..31, 1} & A_{24..31, 0} & A_{24..31, 3} & A_{24..31, 2} \\
A_{32..39, 2} & A_{32..39, 3} & A_{32..39, 0} & A_{32..39, 1} & A_{48..55, 2} & A_{48..55, 3} & A_{48..55, 0} & A_{48..55, 1} \\
A_{40..47, 3} & A_{40..47, 2} & A_{40..47, 1} & A_{40..47, 0} & A_{56..63, 3} & A_{56..63, 2} & A_{56..63, 1} & A_{56..63, 0} \\
\end{array}
\]
## PERMUTED SHARED MEMORY TILES

### Global Memory (column-major)

<table>
<thead>
<tr>
<th>[A_{0..7}, 0]</th>
<th>[A_{8..15}, 0]</th>
<th>[A_{16..23}, 0]</th>
<th>[A_{24..31}, 0]</th>
<th>[A_{32..39}, 0]</th>
<th>[A_{40..47}, 0]</th>
<th>[A_{48..55}, 0]</th>
<th>[A_{56..63}, 0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A_{0..7}, 1]</td>
<td>[A_{8..15}, 1]</td>
<td>[A_{16..23}, 1]</td>
<td>[A_{24..31}, 1]</td>
<td>[A_{32..39}, 1]</td>
<td>[A_{40..47}, 1]</td>
<td>[A_{48..55}, 1]</td>
<td>[A_{56..63}, 1]</td>
</tr>
<tr>
<td>[A_{0..7}, 2]</td>
<td>[A_{8..15}, 2]</td>
<td>[A_{16..23}, 2]</td>
<td>[A_{24..31}, 2]</td>
<td>[A_{32..39}, 2]</td>
<td>[A_{40..47}, 2]</td>
<td>[A_{48..55}, 2]</td>
<td>[A_{56..63}, 2]</td>
</tr>
<tr>
<td>[A_{0..7}, 3]</td>
<td>[A_{8..15}, 3]</td>
<td>[A_{16..23}, 3]</td>
<td>[A_{24..31}, 3]</td>
<td>[A_{32..39}, 3]</td>
<td>[A_{40..47}, 3]</td>
<td>[A_{48..55}, 3]</td>
<td>[A_{56..63}, 3]</td>
</tr>
</tbody>
</table>

### Shared Memory (permuted)

<table>
<thead>
<tr>
<th>[A_{0..7}, 0]</th>
<th>[A_{0..7}, 1]</th>
<th>[A_{0..7}, 2]</th>
<th>[A_{0..7}, 3]</th>
<th>[A_{16..23}, 0]</th>
<th>[A_{16..23}, 1]</th>
<th>[A_{16..23}, 2]</th>
<th>[A_{16..23}, 3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A_{8..15}, 0]</td>
<td>[A_{8..15}, 1]</td>
<td>[A_{8..15}, 2]</td>
<td>[A_{8..15}, 3]</td>
<td>[A_{24..31}, 0]</td>
<td>[A_{24..31}, 1]</td>
<td>[A_{24..31}, 2]</td>
<td>[A_{24..31}, 3]</td>
</tr>
<tr>
<td>[A_{32..39}, 0]</td>
<td>[A_{32..39}, 1]</td>
<td>[A_{32..39}, 2]</td>
<td>[A_{32..39}, 3]</td>
<td>[A_{48..55}, 0]</td>
<td>[A_{48..55}, 1]</td>
<td>[A_{48..55}, 2]</td>
<td>[A_{48..55}, 3]</td>
</tr>
<tr>
<td>[A_{40..47}, 0]</td>
<td>[A_{40..47}, 1]</td>
<td>[A_{40..47}, 2]</td>
<td>[A_{40..47}, 3]</td>
<td>[A_{56..63}, 0]</td>
<td>[A_{56..63}, 1]</td>
<td>[A_{56..63}, 2]</td>
<td>[A_{56..63}, 3]</td>
</tr>
<tr>
<td>Phase 1</td>
<td>T0</td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td>T5</td>
<td>T6</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Load</td>
<td>(A_{0.7, 0})</td>
<td>(A_{8..15, 0})</td>
<td>(A_{16..23, 0})</td>
<td>(A_{24..31, 0})</td>
<td>(A_{32..39, 0})</td>
<td>(A_{40..47, 0})</td>
<td>(A_{48..55, 0})</td>
</tr>
<tr>
<td>Store</td>
<td>(A_{0.7, 1})</td>
<td>(A_{8..15, 1})</td>
<td>(A_{16..23, 1})</td>
<td>(A_{24..31, 1})</td>
<td>(A_{32..39, 1})</td>
<td>(A_{40..47, 1})</td>
<td>(A_{48..55, 1})</td>
</tr>
</tbody>
</table>

| Load   | \(A_{0.7, 2}\) | \(A_{8..15, 2}\) | \(A_{16..23, 2}\) | \(A_{24..31, 2}\) | \(A_{32..39, 2}\) | \(A_{40..47, 2}\) | \(A_{48..55, 2}\) | \(A_{56..63, 2}\) |
| Store  | \(A_{0.7, 3}\) | \(A_{8..15, 3}\) | \(A_{16..23, 3}\) | \(A_{24..31, 3}\) | \(A_{32..39, 3}\) | \(A_{40..47, 3}\) | \(A_{48..55, 3}\) | \(A_{56..63, 3}\) |

Load (128 bits per thread)

Store (128 bits per thread)
## Permutated Shared Memory Tiles

<table>
<thead>
<tr>
<th>Phase 2</th>
<th>T8</th>
<th>T9</th>
<th>T10</th>
<th>T11</th>
<th>T12</th>
<th>T13</th>
<th>T14</th>
<th>T15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A&lt;sub&gt;0..7, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;8..15, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;16..23, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;24..31, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;32..39, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;40..47, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;48..55, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;56..63, 0&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>A&lt;sub&gt;0..7, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;8..15, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;16..23, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;24..31, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;32..39, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;40..47, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;48..55, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;56..63, 1&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>A&lt;sub&gt;0..7, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;8..15, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;16..23, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;24..31, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;32..39, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;40..47, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;48..55, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;56..63, 2&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>A&lt;sub&gt;0..7, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;8..15, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;16..23, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;24..31, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;32..39, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;40..47, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;48..55, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;56..63, 3&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Store</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A&lt;sub&gt;0..7, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;0..7, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;0..7, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;0..7, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;16..23, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;16..23, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;16..23, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;16..23, 3&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>A&lt;sub&gt;8..15, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;8..15, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;8..15, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;8..15, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;24..31, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;24..31, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;24..31, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;24..31, 2&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>A&lt;sub&gt;32..39, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;32..39, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;32..39, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;32..39, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;48..55, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;48..55, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;48..55, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;48..55, 1&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>A&lt;sub&gt;40..47, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;40..47, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;40..47, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;40..47, 0&lt;/sub&gt;</td>
<td>A&lt;sub&gt;56..63, 3&lt;/sub&gt;</td>
<td>A&lt;sub&gt;56..63, 2&lt;/sub&gt;</td>
<td>A&lt;sub&gt;56..63, 1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;56..63, 0&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>

GMEM

SMEM
## Permutated Shared Memory Tiles

### Phase 3

<table>
<thead>
<tr>
<th></th>
<th>T16</th>
<th>T17</th>
<th>T18</th>
<th>T19</th>
<th>T20</th>
<th>T21</th>
<th>T22</th>
<th>T23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>A₀.₇, 0</td>
<td>A₈.₁₅, 0</td>
<td>A₁₆.₂₃, 0</td>
<td>A₂₄.₃₁, 0</td>
<td>A₃₂.₃₉, 0</td>
<td>A₄₀.₄₇, 0</td>
<td>A₴₈.₅₅, 0</td>
<td>A₅₆.₆₃, 0</td>
</tr>
<tr>
<td></td>
<td>A₀.₇, 1</td>
<td>A₈.₁₅, 1</td>
<td>A₁₆.₂₃, 1</td>
<td>A₂₄.₃₁, 1</td>
<td>A₃₂.₃₉, 1</td>
<td>A₴₀.₄₇, 1</td>
<td>A₴₈.₅₅, 1</td>
<td>A₅₆.₆₃, 1</td>
</tr>
<tr>
<td></td>
<td>A₀.₇, 2</td>
<td>A₈.₁₅, 2</td>
<td>A₁₆.₂₃, 2</td>
<td>A₂₄.₃₁, 2</td>
<td>A₃₂.₃₉, 2</td>
<td>A₴₀.₄₇, 2</td>
<td>A₴₈.₅₅, 2</td>
<td>A₅₆.₆₃, 2</td>
</tr>
<tr>
<td></td>
<td>A₀.₇, 3</td>
<td>A₈.₁₅, 3</td>
<td>A₁₆.₂₃, 3</td>
<td>A₂₄.₃₁, 3</td>
<td>A₃₂.₃₉, 3</td>
<td>A₴₀.₄₇, 3</td>
<td>A₴₈.₅₅, 3</td>
<td>A₅₆.₆₃, 3</td>
</tr>
</tbody>
</table>

### Store

|        | A₀.₇, 0      | A₀.₇, 1      | A₀.₇, 2      | A₀.₇, 3      | A₁₆.₂₃, 0  | A₁₆.₂₃, 1  | A₁₆.₂₃, 2  | A₁₆.₂₃, 3  |
|        | A₈.₁₅, 1    | A₈.₁₅, 0    | A₈.₁₅, 2    | A₈.₁₅, 3    | A₂₄.₃₁, 0  | A₂₄.₃₁, 1  | A₂₄.₃₁, 2  | A₂₄.₃₁, 3  |
|        | A₃₂.₃₉, 2   | A₃₂.₃₉, 3   | A₃₂.₃₉, 0   | A₃₂.₃₉, 1   | A₄₈.₅₅, 2  | A₄₈.₅₅, 3  | A₄₈.₅₅, 0  | A₄₈.₅₅, 1  |
|        | A₄₀.₄₇, 3   | A₄₀.₄₇, 2   | A₄₀.₄₇, 1   | A₄₀.₄₇, 0   | A₅₆.₆₃, 3  | A₅₆.₆₃, 2  | A₅₆.₆₃, 1  | A₅₆.₆₃, 0  |

(128 bits per thread)
### PERMUTED SHARED MEMORY TILES

**Phase 4**

<table>
<thead>
<tr>
<th></th>
<th>T24</th>
<th>T25</th>
<th>T26</th>
<th>T27</th>
<th>T28</th>
<th>T29</th>
<th>T30</th>
<th>T31</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$A_{0..7, 0}$</td>
<td>$A_{8..15, 0}$</td>
<td>$A_{16..23, 0}$</td>
<td>$A_{24..31, 0}$</td>
<td>$A_{32..39, 0}$</td>
<td>$A_{40..47, 0}$</td>
<td>$A_{48..55, 0}$</td>
<td>$A_{56..63, 0}$</td>
</tr>
<tr>
<td>A</td>
<td>$A_{0..7, 1}$</td>
<td>$A_{8..15, 1}$</td>
<td>$A_{16..23, 1}$</td>
<td>$A_{24..31, 1}$</td>
<td>$A_{32..39, 1}$</td>
<td>$A_{40..47, 1}$</td>
<td>$A_{48..55, 1}$</td>
<td>$A_{56..63, 1}$</td>
</tr>
<tr>
<td>A</td>
<td>$A_{0..7, 2}$</td>
<td>$A_{8..15, 2}$</td>
<td>$A_{16..23, 2}$</td>
<td>$A_{24..31, 2}$</td>
<td>$A_{32..39, 2}$</td>
<td>$A_{40..47, 2}$</td>
<td>$A_{48..55, 2}$</td>
<td>$A_{56..63, 2}$</td>
</tr>
<tr>
<td>A</td>
<td>$A_{0..7, 3}$</td>
<td>$A_{8..15, 3}$</td>
<td>$A_{16..23, 3}$</td>
<td>$A_{24..31, 3}$</td>
<td>$A_{32..39, 3}$</td>
<td>$A_{40..47, 3}$</td>
<td>$A_{48..55, 3}$</td>
<td>$A_{56..63, 3}$</td>
</tr>
</tbody>
</table>

**Load**

(128 bits per thread)

GMEM

**Store**

(128 bits per thread)

SMEM
POINTER OFFSETS FOR PERMUTED SHARED MEMORY

Global Memory (column-major)

<table>
<thead>
<tr>
<th>A0..7, 0</th>
<th>A8..15, 0</th>
<th>A16..23, 0</th>
<th>A24..31, 0</th>
<th>A32..39, 0</th>
<th>A40..47, 0</th>
<th>A48..55, 0</th>
<th>A56..63, 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0..7, 1</td>
<td>A8..15, 1</td>
<td>A16..23, 1</td>
<td>A24..31, 1</td>
<td>A32..39, 1</td>
<td>A40..47, 1</td>
<td>A48..55, 1</td>
<td>A56..63, 1</td>
</tr>
<tr>
<td>A0..7, 2</td>
<td>A8..15, 2</td>
<td>A16..23, 2</td>
<td>A24..31, 2</td>
<td>A32..39, 2</td>
<td>A40..47, 2</td>
<td>A48..55, 2</td>
<td>A56..63, 2</td>
</tr>
<tr>
<td>A0..7, 3</td>
<td>A8..15, 3</td>
<td>A16..23, 3</td>
<td>A24..31, 3</td>
<td>A32..39, 3</td>
<td>A40..47, 3</td>
<td>A48..55, 3</td>
<td>A56..63, 3</td>
</tr>
</tbody>
</table>

Shared Memory (permuted)

<table>
<thead>
<tr>
<th>A0..7, 0</th>
<th>A0..7, 1</th>
<th>A0..7, 2</th>
<th>A0..7, 3</th>
<th>A16..23, 0</th>
<th>A16..23, 1</th>
<th>A16..23, 2</th>
<th>A16..23, 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A8..15, 1</td>
<td>A8..15, 0</td>
<td>A8..15, 3</td>
<td>A8..15, 2</td>
<td>A24..31, 1</td>
<td>A24..31, 0</td>
<td>A24..31, 3</td>
<td>A24..31, 2</td>
</tr>
<tr>
<td>A32..39, 2</td>
<td>A32..39, 3</td>
<td>A32..39, 0</td>
<td>A32..39, 1</td>
<td>A48..55, 2</td>
<td>A48..55, 3</td>
<td>A48..55, 0</td>
<td>A48..55, 1</td>
</tr>
<tr>
<td>A40..47, 3</td>
<td>A40..47, 2</td>
<td>A40..47, 1</td>
<td>A40..47, 0</td>
<td>A56..63, 3</td>
<td>A56..63, 2</td>
<td>A56..63, 1</td>
<td>A56..63, 0</td>
</tr>
</tbody>
</table>

```
int lane = threadIdx.x % 32;
int c = lane % 8;
int s = lane / 8;
int gmem_offset = c + s * lda;
```

```
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

<table>
<thead>
<tr>
<th></th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0..7</td>
<td>0</td>
<td>A0..7</td>
<td>1</td>
<td>A0..7</td>
</tr>
<tr>
<td>A16..23</td>
<td>0</td>
<td>A16..23</td>
<td>1</td>
<td>A16..23</td>
</tr>
<tr>
<td>A8..15</td>
<td>1</td>
<td>A8..15</td>
<td>0</td>
<td>A8..15</td>
</tr>
<tr>
<td>A24..31</td>
<td>1</td>
<td>A24..31</td>
<td>0</td>
<td>A24..31</td>
</tr>
<tr>
<td>A32..39</td>
<td>2</td>
<td>A32..39</td>
<td>3</td>
<td>A32..39</td>
</tr>
<tr>
<td>A48..55</td>
<td>2</td>
<td>A48..55</td>
<td>3</td>
<td>A48..55</td>
</tr>
<tr>
<td>A56..63</td>
<td>3</td>
<td>A56..63</td>
<td>2</td>
<td>A56..63</td>
</tr>
</tbody>
</table>

QP0

MMA₀

A₀..7

QP0

QP2

A₈..₁₅

QP1

A₁₆..₂₃

QP3

A₂₄..₃₁
CONFLICT-FREE SHARED MEMORY LOADS

Phase 1

QP0

QP1

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

QP1

QP2

QP3
# CONFLICT-FREE SHARED MEMORY LOADS

## Phase 2

<table>
<thead>
<tr>
<th>T8</th>
<th>T9</th>
<th>T10</th>
<th>T11</th>
<th>T12</th>
<th>T13</th>
<th>T14</th>
<th>T15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{8..15, 1}$</td>
<td>$A_{8..15, 0}$</td>
<td>$A_{8..15, 2}$</td>
<td>$A_{24..31, 1}$</td>
<td>$A_{24..31, 0}$</td>
<td>$A_{24..31, 3}$</td>
<td>$A_{24..31, 2}$</td>
<td></td>
</tr>
<tr>
<td>$A_{32..39, 2}$</td>
<td>$A_{32..39, 3}$</td>
<td>$A_{32..39, 0}$</td>
<td>$A_{48..55, 2}$</td>
<td>$A_{48..55, 3}$</td>
<td>$A_{48..55, 0}$</td>
<td>$A_{48..55, 1}$</td>
<td></td>
</tr>
<tr>
<td>$A_{40..47, 3}$</td>
<td>$A_{40..47, 2}$</td>
<td>$A_{40..47, 1}$</td>
<td>$A_{56..63, 3}$</td>
<td>$A_{56..63, 2}$</td>
<td>$A_{56..63, 1}$</td>
<td>$A_{56..63, 0}$</td>
<td></td>
</tr>
</tbody>
</table>

### QP0

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{0..7}$</td>
<td>$A_{0..7}$</td>
<td>$A_{0..7}$</td>
<td>$A_{0..7}$</td>
</tr>
</tbody>
</table>

### QP1

<table>
<thead>
<tr>
<th>T16</th>
<th>T17</th>
<th>T18</th>
<th>T19</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{16..23}$</td>
<td>$A_{16..23}$</td>
<td>$A_{16..23}$</td>
<td>$A_{16..23}$</td>
</tr>
</tbody>
</table>

### QP2

<table>
<thead>
<tr>
<th>A.col</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
</tr>
<tr>
<td>T1</td>
</tr>
<tr>
<td>T2</td>
</tr>
<tr>
<td>T3</td>
</tr>
</tbody>
</table>

### QP3

<table>
<thead>
<tr>
<th>B.row</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
</tr>
<tr>
<td>T1</td>
</tr>
<tr>
<td>T2</td>
</tr>
<tr>
<td>T3</td>
</tr>
</tbody>
</table>

### MMA₀

- QP0
- QP1
- QP2
- QP3
CONFLICT-FREE SHARED MEMORY LOADS

Phase 3

QP0

QP1

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}, 0
A_{24..31}, 1
A_{24..31}, 2
A_{24..31}, 3
A_{32..39}, 2
A_{32..39}, 3
A_{32..39}, 0
A_{32..39}, 3
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

A_{0..7}
A_{8..15}
A_{16..23}
A_{24..31}
A_{32..39}
A_{40..47}
A_{56..63}

QP0
MMA_0

QP1
QP2
QP3
CONFLICT-FREE SHARED MEMORY LOADS

Phase 4

QP2

QP3

\[
\begin{array}{cccccccc}
A_{0.7, 0} & A_{0.7, 1} & A_{0.7, 2} & A_{0.7, 3} & A_{16..23, 0} & A_{16..23, 1} & A_{16..23, 2} & A_{16..23, 3} \\
A_{8..15, 1} & A_{8..15, 0} & A_{8..15, 3} & A_{8..15, 2} & A_{24..31, 1} & A_{24..31, 0} & A_{24..31, 3} & A_{24..31, 2} \\
A_{32..39, 2} & A_{32..39, 3} & A_{32..39, 0} & A_{32..39, 1} & A_{48..55, 2} & A_{48..55, 3} & A_{48..55, 0} & A_{48..55, 1} \\
A_{40..47, 3} & A_{40..47, 2} & A_{40..47, 1} & A_{40..47, 0} & A_{56..63, 3} & A_{56..63, 2} & A_{56..63, 1} & A_{56..63, 0} \\
\end{array}
\]
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
GlobalLoadStream

Global Memory → Shared Memory → Register File → CUDA/Tensor Cores → SMEM → CUDA Cores → Global Memory

Transformer
SharedStoreIterator
SharedLoadIterator

GlobalLoadIterator

MatrixMultiply mma.sync

Warp Matrix Multiply

Epilogue

Epilogue Functor
GlobalLoadIterator
GlobalStoreIterator
STORING TO SHARED MEMORY

CUTLASS Tile Iterators to transform:

- **Global Memory**: Canonical matrix layout ➜ **Shared Memory**: permuted shared memory layout
LOADING FROM SHARED MEMORY

CUTLASS Tile Iterators to transform:

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

```cpp
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;
};
```
**EXECUTING MMA.SYNC**

![Diagram of MMA.SYNC execution](image)

**CUTLASS Warp-scoped matrix multiply**

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

```cpp
#include <cutlass/gemm/volta884_multiply_add.h>

// Shape of a warp-level GEMM (K-by-N-by-M)
// Layout of A multiplicand
// MatrixLayout::Kind LayoutA,
// Data type of A multiplicand
// typename ScalarA,
// Layout of B multiplicand
// MatrixLayout::Kind LayoutB,
// Data type of B 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) {
        ...
    }
};
```
SPEEDUP RELATIVE TO WMMA

Transformer - CUTLASS 1.3 - mma.sync speedup vs WMMA
V100 - CUDA 10.1

Speedup
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


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 synced 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
```
Accumulators distributed among threads (QP0 shown)

\[ \text{mma.sync.aligned.m8n8k4.alayout.blayout.dtype.f16.f16.ctype} \quad \text{d, a, b, c;} \]

\[ .\text{ctype} = \{ .\text{f16, f32} \}; \]
\[ .\text{dtype} = \{ .\text{f16, f32} \}; \]

\[ \text{d: 8 x .f32} \]
\[ \text{c: 8 x .f32} \]