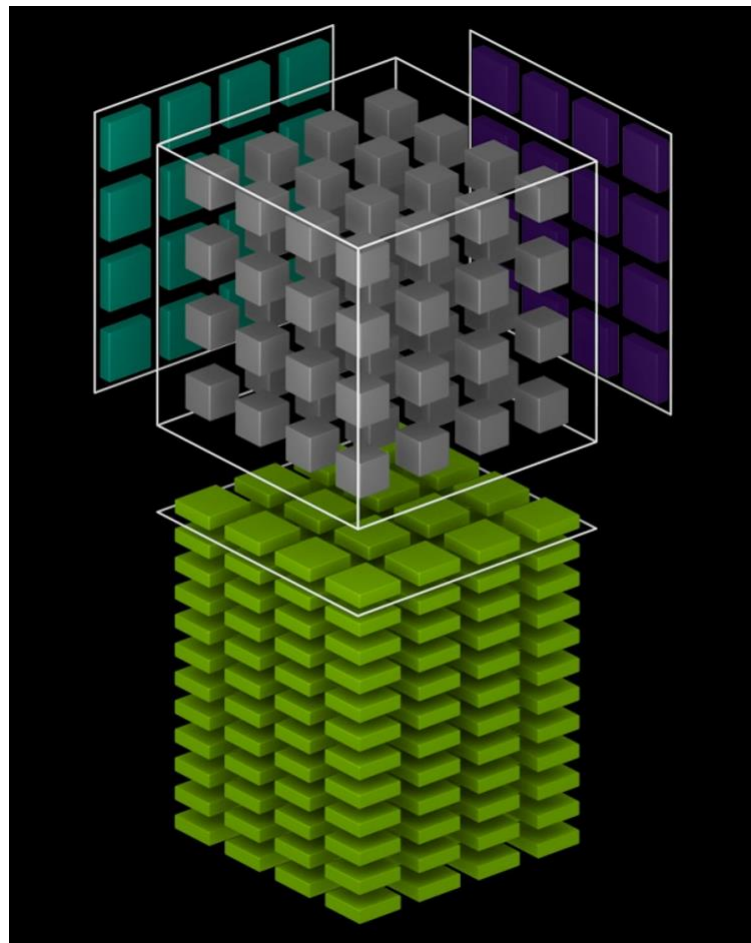




USING VOLTA TENSOR CORES

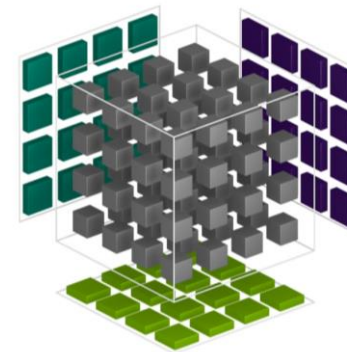
Jeff Larkin, December 04, 2018

VOLTA TENSOR CORE



TENSOR CORE

Mixed Precision Matrix Math
4x4 matrices



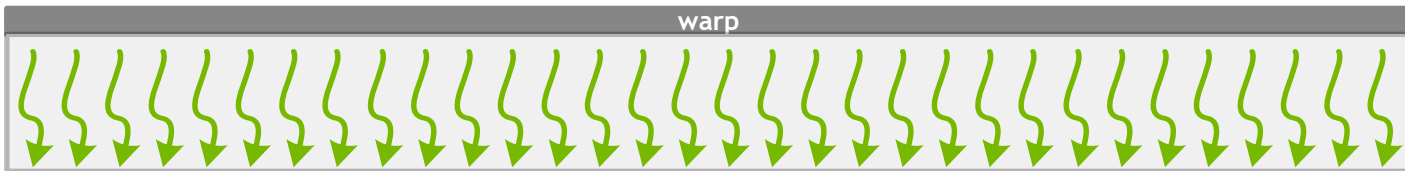
$$\mathbf{D} = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 FP16 FP16 FP16 or FP32

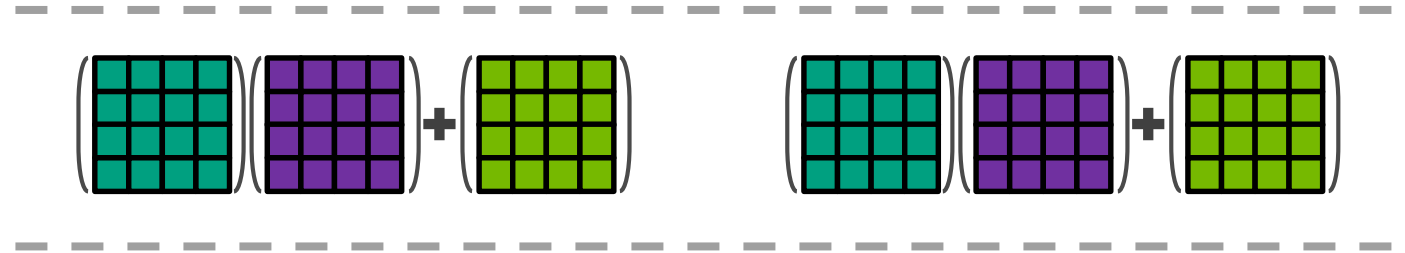
$$\mathbf{D} = \mathbf{AB} + \mathbf{C}$$

TENSOR CORE COORDINATION

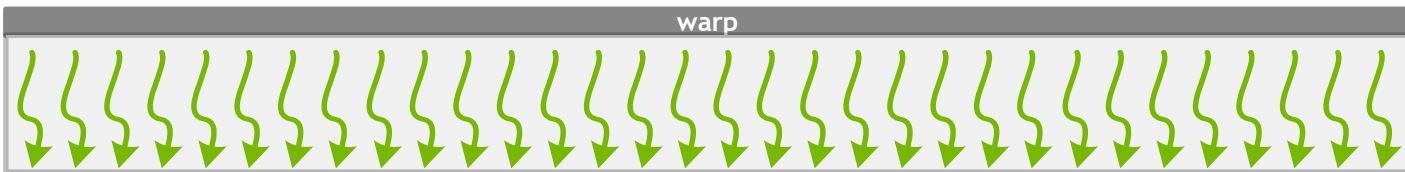
Full Warp 16x16 Matrix Math



Warp-synchronizing operation for cooperative matrix math

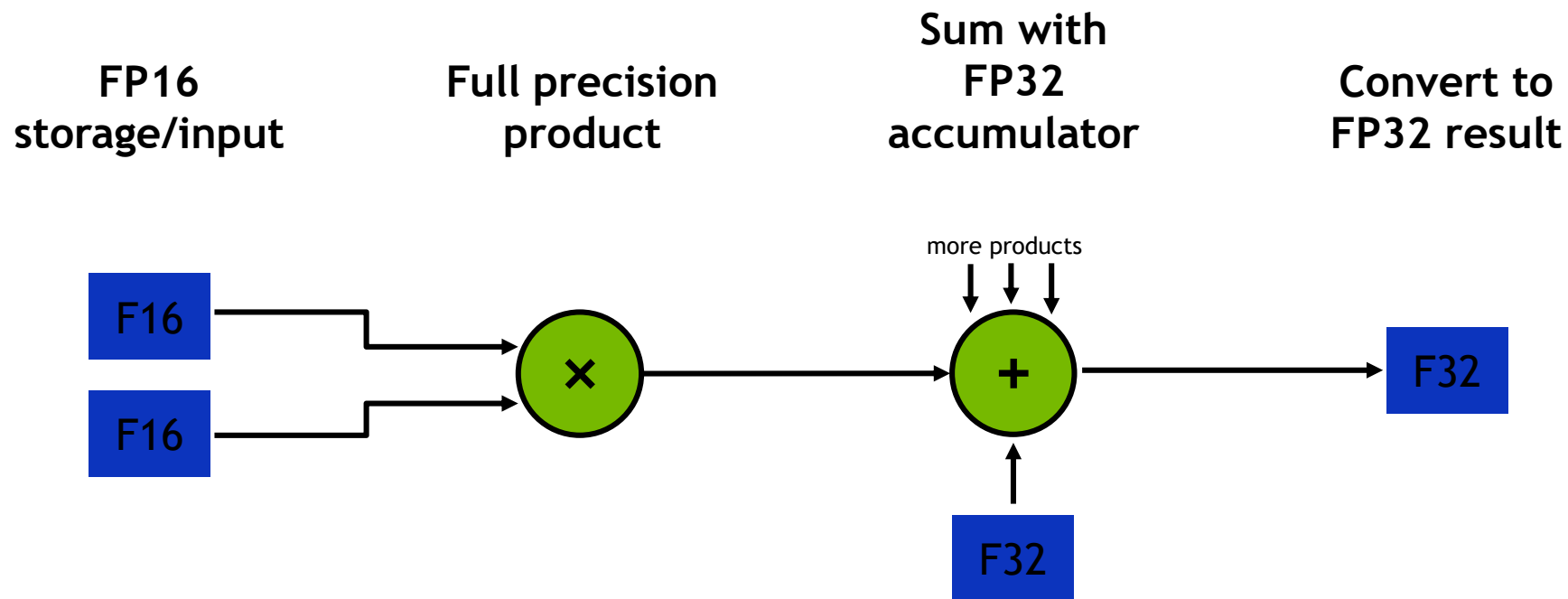


Aggregate Matrix Multiply and Accumulate for **16x16** matrices



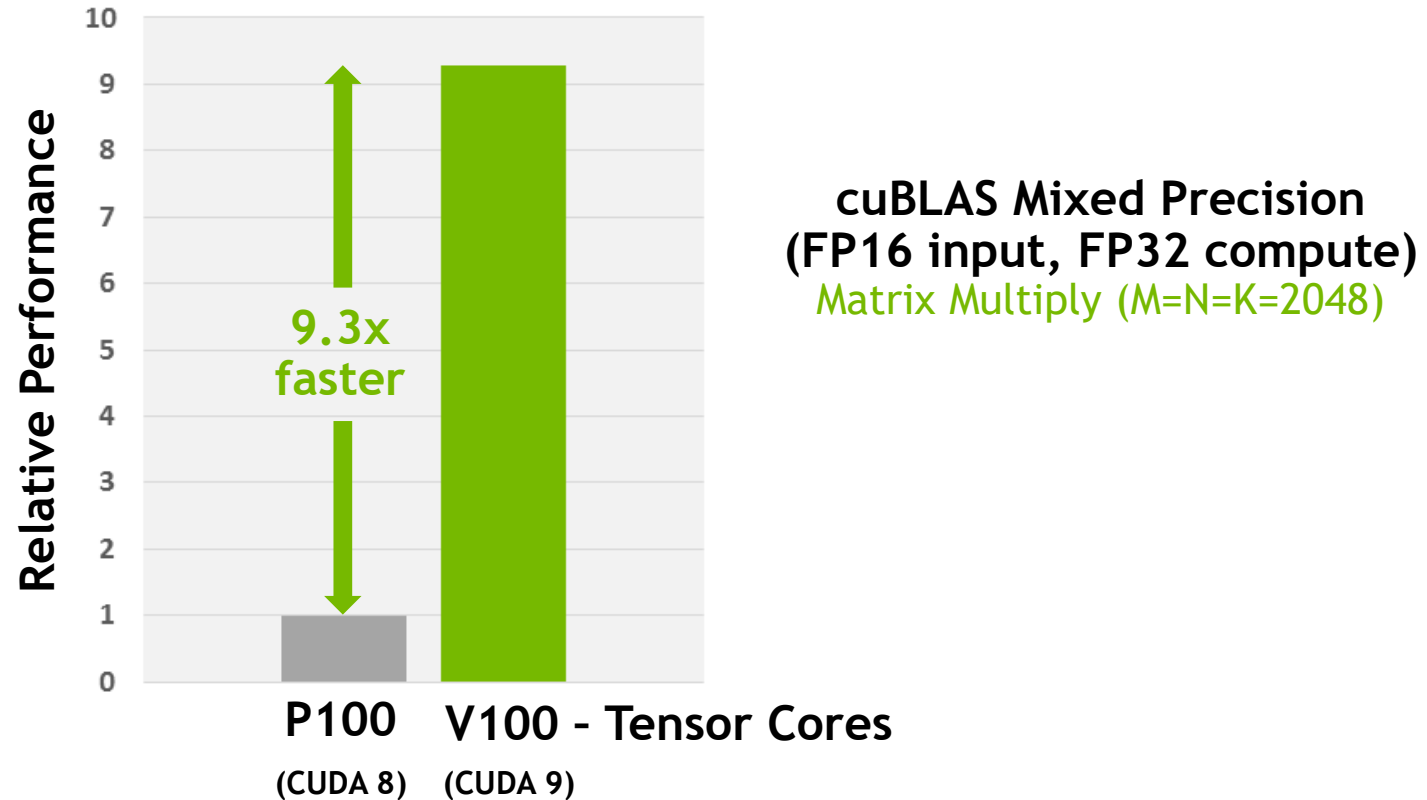
Result distributed across warp

VOLTA TENSOR OPERATION



Also supports FP16 accumulator mode for inferencing

A GIANT LEAP FOR DEEP LEARNING



USING TENSOR CORES



NVIDIA cuDNN, cuBLAS, TensorRT

Volta Optimized
Frameworks and Libraries

```
__device__ void tensor_op_16_16_16(  
    float *d, half *a, half *b, float *c)  
{  
    wmma::fragment<matrix_a, ...> Amat;  
    wmma::fragment<matrix_b, ...> Bmat;  
    wmma::fragment<matrix_c, ...> Cmat;  
  
    wmma::load_matrix_sync(Amat, a, 16);  
    wmma::load_matrix_sync(Bmat, b, 16);  
    wmma::fill_fragment(Cmat, 0.0f);  
  
    wmma::mma_sync(Cmat, Amat, Bmat, Cmat);  
  
    wmma::store_matrix_sync(d, Cmat, 16,  
        wmma::row_major);  
}
```

CUDA C++

Warp-Level Matrix Operations

TENSOR CORES FROM CUBLAS

CUBLAS provides an extended BLAS interface for mixed precisions

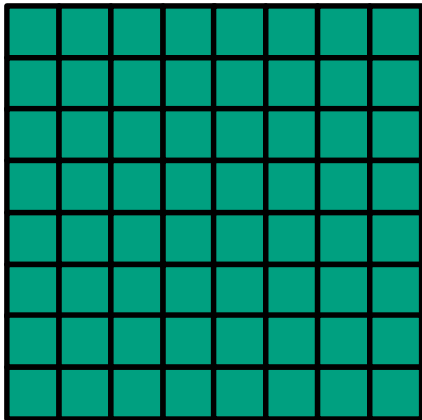
`cublasGemmEx()`, `cublasGemmBatchedEx()`, and `cublasGemmStridedBatchedEx()` all accept FP16 data types for Tensor Core use.

Some solvers are also available in an Ex form.

See <https://docs.nvidia.com/cuda/cublas/index.html#cublas-GemmEx>

CUDA TENSOR CORE PROGRAMMING

New WMMA datatypes



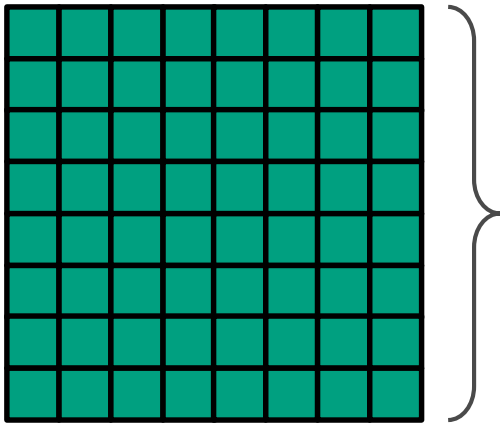
Per-Thread fragments to hold components of matrices for use with Tensor Cores

```
wmma::fragment<matrix_a, ...> Amat;
```

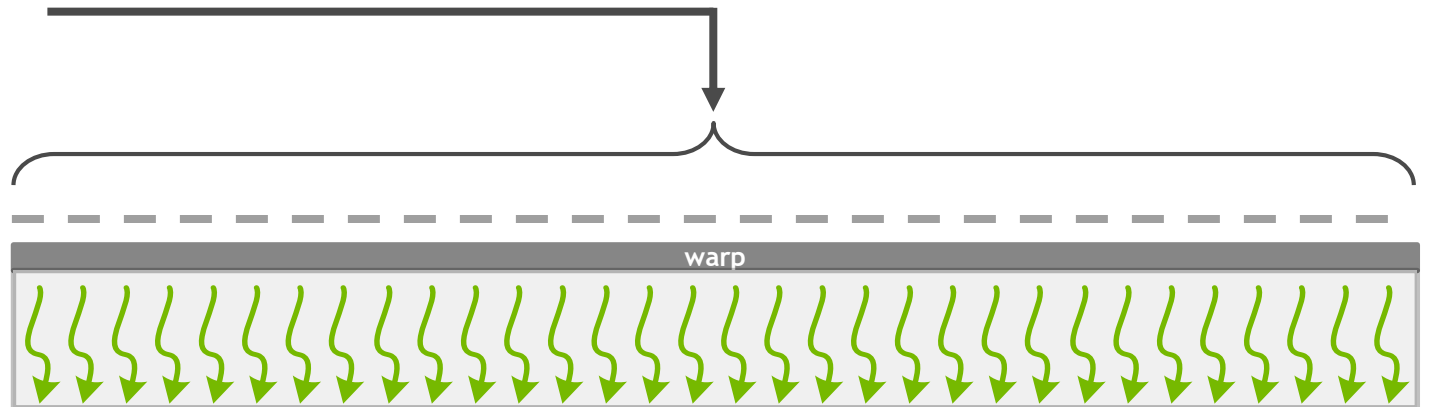
CUDA TENSOR CORE PROGRAMMING

New WMMA load and store operations

Warp-level operation to fetch components of matrices into fragments



```
wmma::load_matrix_sync(Amat, a, stride);
```



CUDA TENSOR CORE PROGRAMMING

New WMMA Matrix Multiply and Accumulate Operation

Warp-level operation to perform matrix multiply and accumulate

```
wmma::mma_sync(Dmat, Amat, Bmat, Cmat);
```

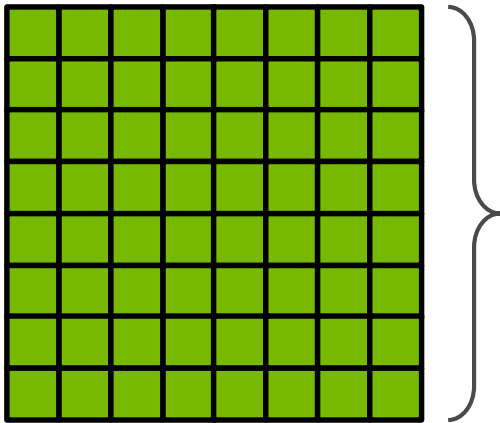
$$D = \begin{pmatrix} \text{Teal Grid} \end{pmatrix} + \begin{pmatrix} \text{Purple Grid} \end{pmatrix} + \begin{pmatrix} \text{Green Grid} \end{pmatrix}$$

The diagram illustrates the matrix multiply and accumulate operation. It shows a large matrix D on the left, followed by an equals sign. To the right of the equals sign are three matrices enclosed in large parentheses. The first matrix is a teal grid, the second is a purple grid, and the third is a green grid. A plus sign is placed between the purple and green grids, indicating that the result of the matrix multiplication of the teal and purple grids is accumulated into the green grid.

CUDA TENSOR CORE PROGRAMMING

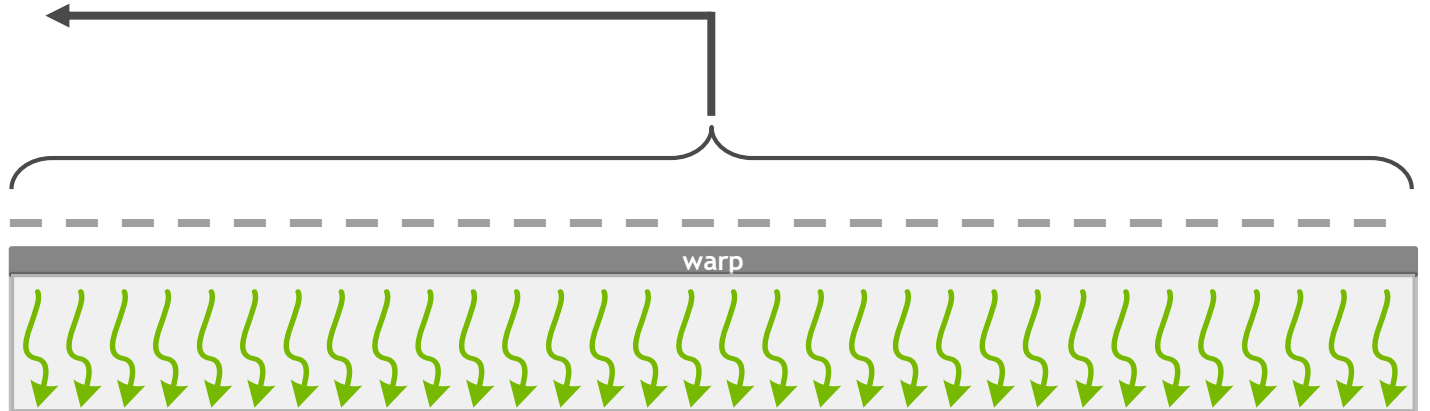
New WMMA load and store operations

Warp-level operation to fetch components of matrices into fragments



Result

```
wmma::store_matrix_sync(d, Dmat, stride);
```



TENSOR CORE EXAMPLE

Create Fragments

Initialize Fragments

Perform MatMul

Store Results

```
__device__ void tensor_op_16_16_16(
    float *d, half *a, half *b, float *c)
{
    wmma::fragment<matrix_a, ...> Amat;
    wmma::fragment<matrix_b, ...> Bmat;
    wmma::fragment<matrix_c, ...> Cmat;

    wmma::load_matrix_sync(Amat, a, 16);
    wmma::load_matrix_sync(Bmat, b, 16);
    wmma::fill_fragment(Cmat, 0.0f);

    wmma::mma_sync(Cmat, Amat, Bmat, Cmat);

    wmma::store_matrix_sync(d, Cmat, 16,
        wmma::row_major);
}
```

CUDA C++

Warp-Level Matrix Operations