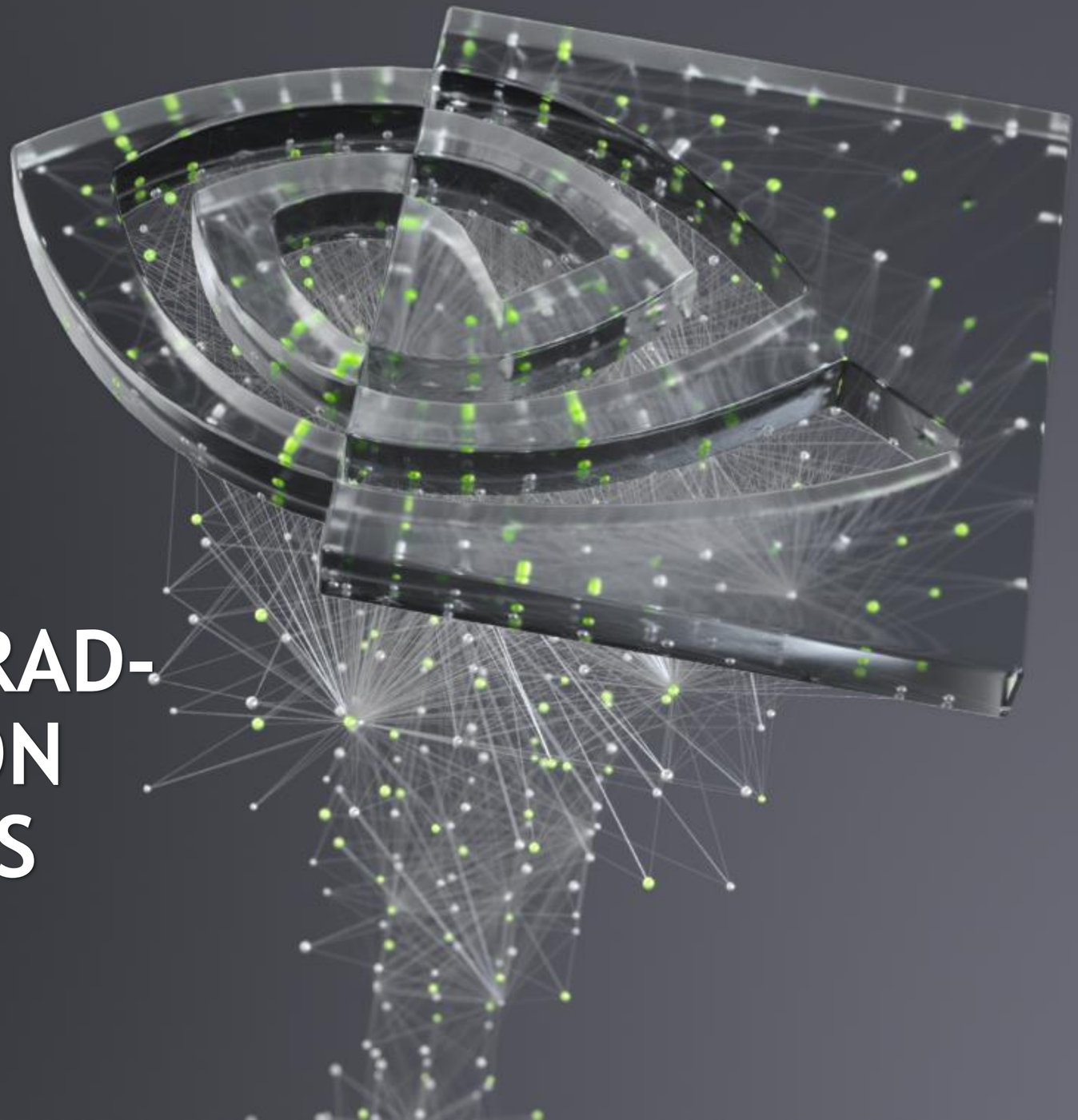


OPTIMIZING WINOGRAD-BASED CONVOLUTION WITH TENSOR CORES

Junhong Liu, Dongxu Yang and Junjie Lai

08/12/2021

ICPP21

AGENDA

Background

NVIDIA Tensor Cores and Winograd convolution

Our Winograd Convolution Methods

Input transformation and batched matrix multiplication

Evaluation

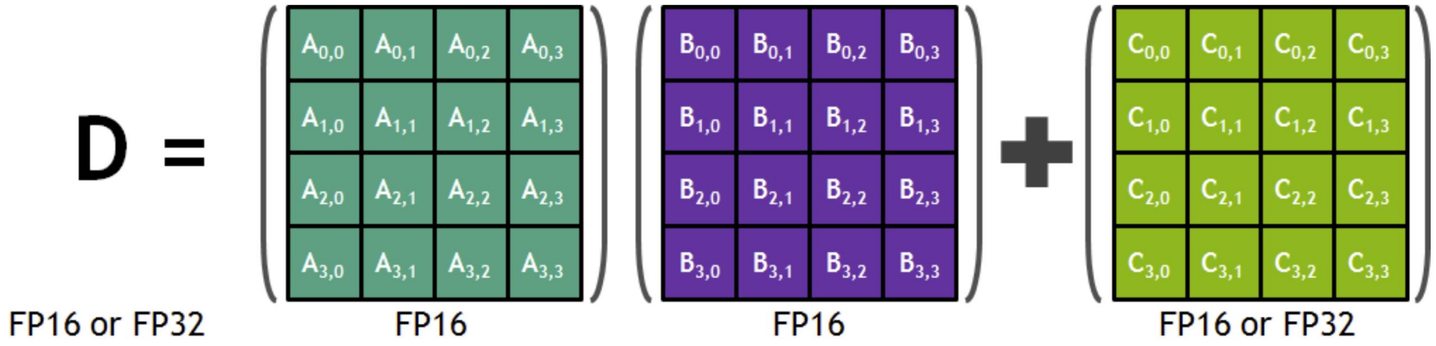
Accuracy and performance results

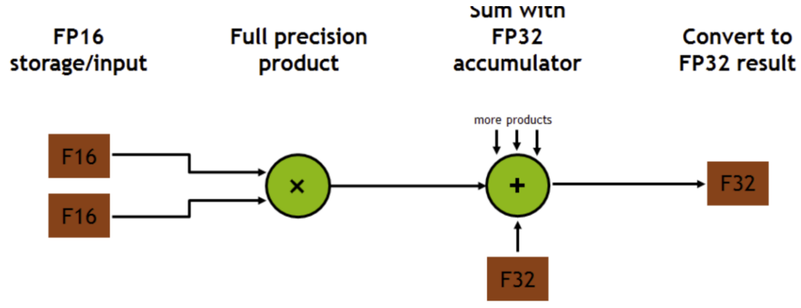
BACKGROUND

BACKGROUND

What are GPU Tensor Cores?

- **Specialized hardware execution units**
  - Tensor Cores perform **matrix instructions**: multiplication between matrix of elements at a time
- **Tensor Cores for 16-bit formats**
  - Products are computed **without loss of precision**, accumulated in FP32
  - For Ampere architecture **16×8×8, and 16×8×16 ($m \times n \times k$) matrix shapes** are supported for FP16 matrix instructions
  - Accelerate **math- and memory-limited operations**





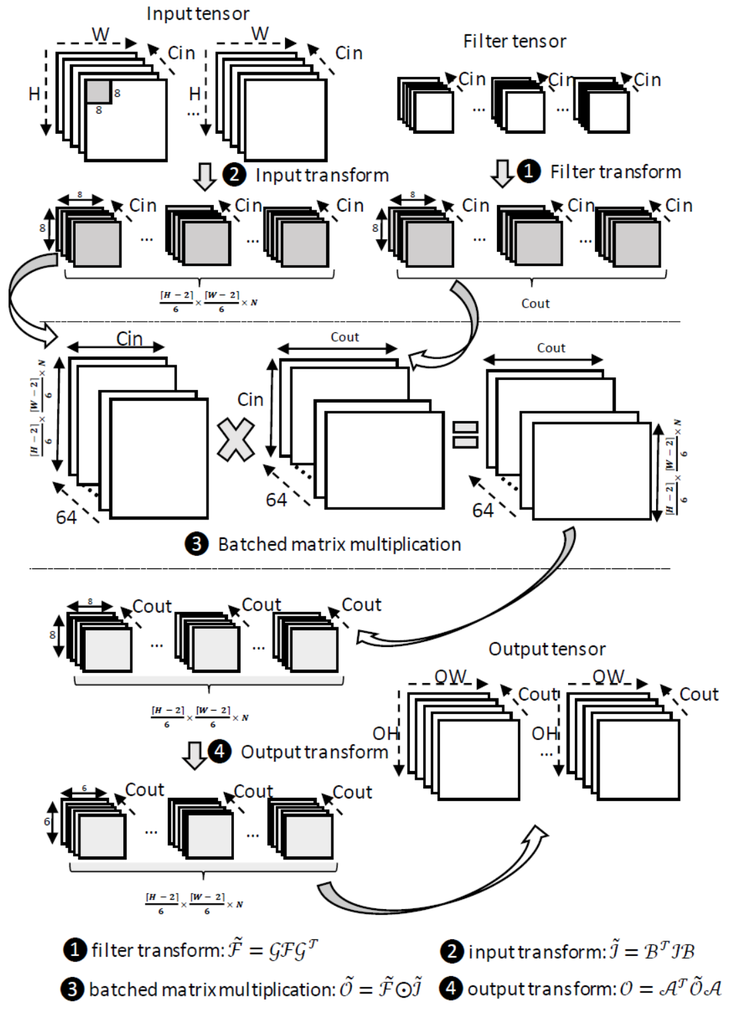
BACKGROUND

Basics of Winograd convolution

Winograd convolution: $F(m \times m, r \times r)$, $m \times$ m is output tile size; r × r is filter size; (m + r - 1) × (m + r - 1) is input tile size

1. Filter transformation: $\widetilde{\mathcal{F}} = \mathcal{G}\mathcal{F}\mathcal{G}^T$
2. Input transformation: $\widetilde{\mathcal{I}} = \mathcal{B}^T\mathcal{I}\mathcal{B}$
3. Element-wise multiplication: $\widetilde{\mathcal{O}} = \widetilde{\mathcal{F}} \odot \widetilde{\mathcal{I}}$
4. Output transformation: $\mathcal{O} = \mathcal{A}^T\widetilde{\mathcal{O}}\mathcal{A}$

$$\mathcal{O} = \overset{④}{\mathcal{A}^T}[(\overset{①}{\mathcal{G}\mathcal{F}\mathcal{G}^T}) \overset{③}{\odot} (\overset{②}{\mathcal{B}^T\mathcal{I}\mathcal{B}})]\mathcal{A}$$



F(6x6, 3x3) Winograd convolution

MOTIVATION

- Existing Winograd convolution implementations limited to **small tiles**
  - F (2 × 2, 3 × 3) and F (4 × 4, 3 × 3)
  - Smaller reduction of arithmetic complexity than F(6x6, 3x3)
- Existing Winograd convolution implementations mostly limited to **single precision data**
  - Consumes more memory and computes slower than that of half precision data
  - Do not use Tensor Cores for all of four parts of Winograd convolution
- We focus on **mixed precision *F(6x6, 3x3)*** Winograd convolution using **Tensor Cores**
  - Pro: **larger reduction of arithmetic complexity**, up to 5.06x reduction
  - Con: **lower numeric precision**

OUR METHODS

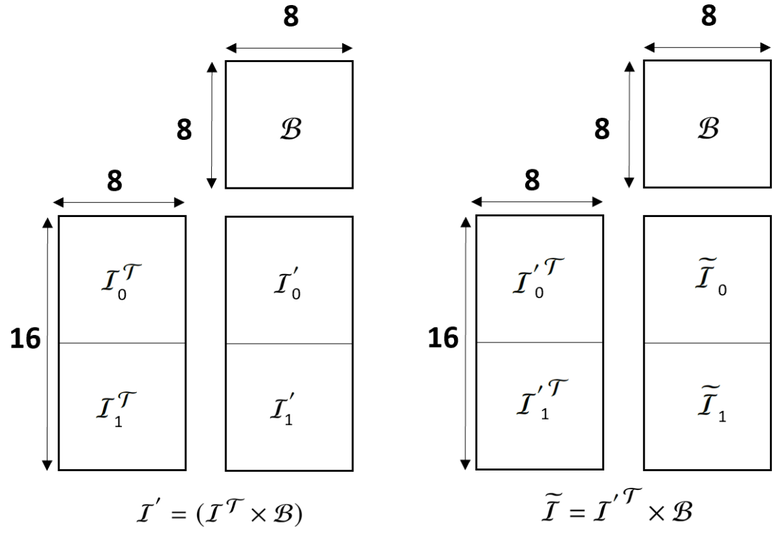
INPUT TRANSFORMATION

F(6x6, 3x3) Winograd convolution

- Input transformation computing : $\widetilde{\mathcal{I}} = \mathcal{B}^{\mathcal{T}} \mathcal{I} \mathcal{B}$
  - $\mathcal{B}$ is transformation matrix with fixed 8x8 size for *F(6x6, 3x3)*
  - $\mathcal{I}$ is an input tensor tile with shape of 8x8 for *F(6x6, 3x3)*
- 16×8×8 ($m \times n \times k$) matrix shapes supported for Tensor Cores on Ampere GPU
  - To efficiently use Tensor Cores:

$$\widetilde{\mathcal{I}} = \mathcal{B}^{\mathcal{T}} \mathcal{I} \mathcal{B} \rightarrow \widetilde{\mathcal{I}} = (\mathcal{I}^{\mathcal{T}} \mathcal{B})^{\mathcal{T}} \times \mathcal{B}$$

$$\rightarrow \mathcal{I}' = (\mathcal{I}^{\mathcal{T}} \times \mathcal{B}) \quad (\mathcal{I}')^{\mathcal{T}} \quad \widetilde{\mathcal{I}} = \mathcal{I}'^{\mathcal{T}} \times \mathcal{B}$$



INPUT TRANSFORMATION

F(6x6, 3x3) Winograd convolution

- The process of input transformation $\widetilde{\mathcal{I}} = (\mathcal{I}^{\mathcal{T}}\mathcal{B})^{\mathcal{T}} \times \mathcal{B}$ is divided into 4 steps:
  - **Load** input tensor from global memory to registers
    - **Coalesced memory access** of global memory
    - Use **vector loads** to **increase bandwidth utilization** and **decrease instructions**
  - **Reorganize register data layout** using **shuffle instructions**
    - As elements in each thread is not right data for Tensor Cores matrix instructions
  - Perform the input transformation **using Tensor Cores:** $\mathcal{I}' = (\mathcal{I}^{\mathcal{T}} \times \mathcal{B})$ $\widetilde{\mathcal{I}} = \mathcal{I}'^{\mathcal{T}} \times \mathcal{B}$
    - Matrix transpose using shuffle instructions: $(\mathcal{I}')^{\mathcal{T}}$
  - **Store** the results back to global memory

ELEMENT WISE MULTIPLICATION

F(6x6, 3x3) Winograd convolution

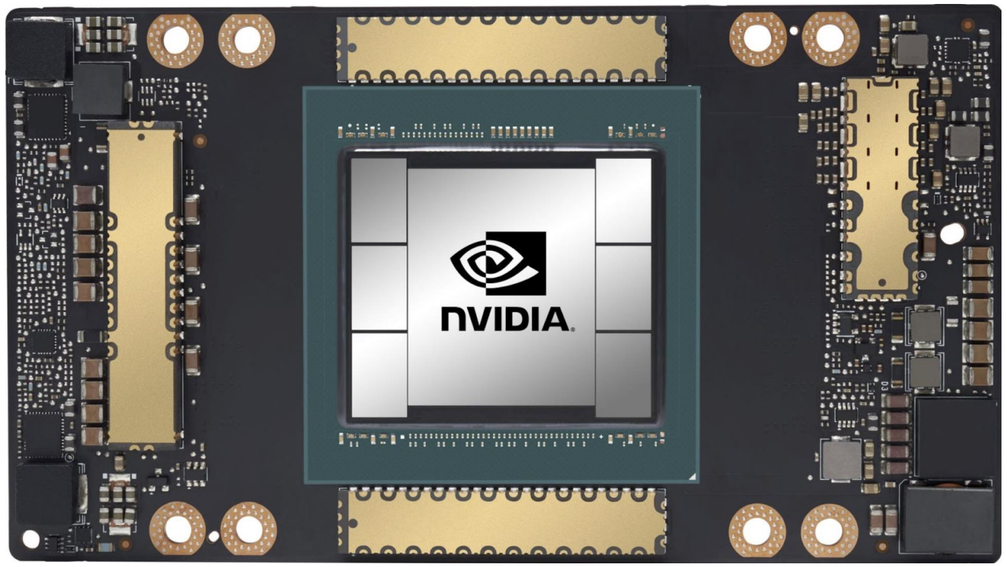
- The element wise multiplication can be converted to batched **matrix multiplication** (GEMM)
  - Using Tensor Cores to implement batched GEMM
- On Ampere architecture, the data in global memory can be loaded to shared memory directly without using intermediate registers, using $memcpy\_async$ API
  - Load and compute are pipelined to hide memory latency

EVALUATIONS

EVALUATIONS

Experiment setup

- Hardware: NVIDIA Ampere A100
- Software: NVCC 11.2
- Baseline: cuDNN 8.1.0
- Network: VGG & FusionNet



ACCURACY

Network	$Cin \times H \times W$	Cout	$F(6\times6,3\times3)$ FP32		cuDNN FP16		$F(2\times2,3\times3)$ FP16		$F(4\times4,3\times3)$ FP16		Ours $F(6\times6,3\times3)$	
			max	average	max	average	max	average	max	average	max	average
VGG	$128\times56\times56$	256	1.7E-03	3.2E-05	3.2E-00	6.4E-02	6.5E-02	7.9E-03	1.0E-00	3.3E-02	1.5E-00	7.8E-02
	$256\times56\times56$	256	2.2E-03	6.3E-05	5.9E-00	1.2E-01	9.2E-02	1.1E-02	1.4E-00	4.7E-02	2.1E-00	1.1E-01
	$256\times28\times28$	512	2.0E-03	6.2E-05	5.5E-00	1.2E-01	8.5E-02	1.1E-02	1.2E-00	4.5E-02	2.1E-00	1.1E-01
	$512\times28\times28$	512	4.2E-03	1.2E-04	11.8E-00	2.4E-01	1.2E-01	1.6E-02	1.9E-00	6.4E-02	3.1E-00	1.5E-01
	$512\times14\times14$	512	4.0E-03	1.2E-04	11.0E-00	2.4E-01	1.2E-01	1.6E-02	1.6E-00	6.8E-02	2.9E-00	1.5E-01
FusionNet	$256\times160\times160$	256	2.2E-03	6.3E-05	3.3E-00	9.1E-02	8.3E-02	1.1E-02	1.2E-00	4.8E-02	1.9E-00	1.1E-01
	$512\times80\times80$	512	3.9E-03	1.2E-04	7.1E-00	1.7E-01	1.2E-01	1.6E-02	1.6E-00	6.7E-02	2.4E-00	1.5E-01
	$1024\times40\times40$	1024	7.3E-03	2.4E-04	12.0E-00	3.4E-01	1.6E-01	2.2E-02	2.1E-00	9.3E-02	4.0E-00	2.1E-01

Maximum and average element error on different network layers on A100. Ground truth was computed by direct convolution using single precision

FP32 VGG	FP16 VGG	Ours VGG
71.22%	71.23%	71.24%

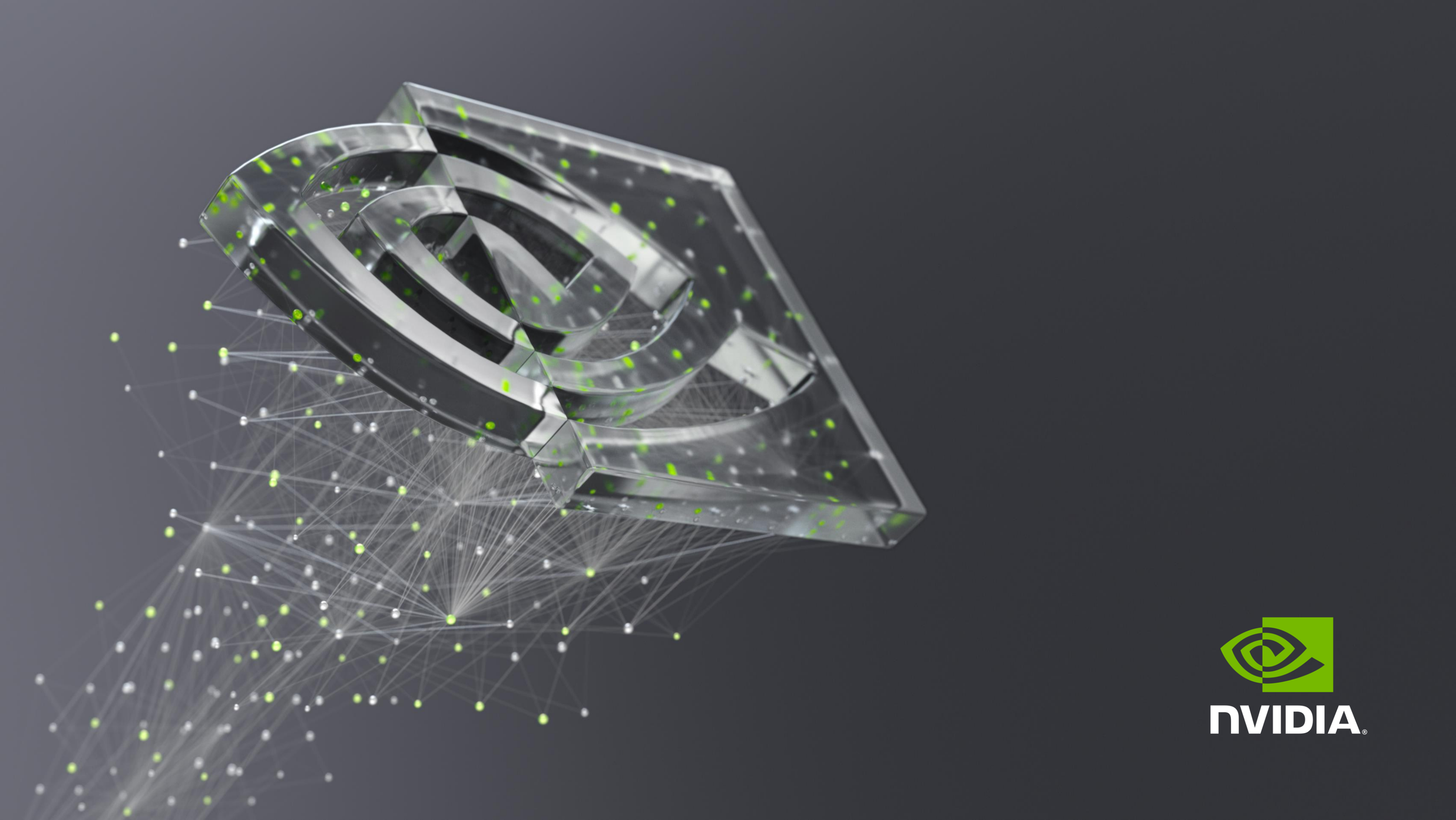
Accuracy of VGG using different data types

PERFORMANCE

$Cin \times H \times W$	Cout	Ours		cuDNN Winograd		cuDNN GEMM convolution		Speedup	
		usec	TFLOPs	usec	TFLOPs	usec	TFLOPs	cuDNN Winograd	cuDNN GEMM
$128 \times 56 \times 56$	256	1319.88	179.38	1554.08	152.35	1562.79	151.50	1.17x	1.18x
$256 \times 56 \times 56$	256	1812.67	261.23	2246.04	210.82	2667.40	177.52	1.24x	1.47x
$256 \times 28 \times 28$	512	745.53	317.57	913.05	259.31	1287.63	183.87	1.22x	1.73x
$512 \times 28 \times 28$	512	1085.47	436.24	1383.48	342.27	2619.22	180.79	1.27x	2.41x
$512 \times 14 \times 14$	512	388.28	304.88	489.68	241.75	615.12	192.45	1.26x	1.58x
$256 \times 160 \times 160$	256	121.12	249.33	1902.62	15.87	214.03	141.10	15.71x	1.77x
$512 \times 80 \times 80$	512	108.86	277.41	1146.47	26.34	227.63	132.67	10.53x	2.09x
$1024 \times 40 \times 40$	1024	179.81	167.95	984.79	30.67	258.11	117.00	5.48x	1.44x

CONCLUSIONS

- We propose an optimized mixed precision F (6×6, 3×3) Winograd convolution implementation on NVIDIA Ampere GPUs using Tensor Cores.
- Our experiments show that the accuracy of mixed precision F (6 × 6, 3 × 3) Winograd convolution is sufficient to infer the convolutional neural networks.
- Our method achieves up to 15.71x and 2.41x speedup on NVIDIA Ampere A100, compared with the state of the art Winograd based convolution and GEMM based convolution in cuDNN 8.1.0, respectively.


NVIDIA