ANANDTECH

Original Link: https://www.anandtech.com/show/12673/titan-v-deep-learning-deep-dive

The NVIDIA Titan V Deep Learning Deep Dive: It's All About The Tensor Cores

65
Comments

by Nate Oh on July 3, 2018 10:15 AM EST

Posted in GPUs NVIDIA Machine Learning Neural Networks GV100 Deep Learning Caffe TensorFlow Titan V

When we last discussed the NVIDIA Titan V in our preview, it was only a few weeks after its surprise launch at the 2017 Neural Information Processing Systems conference. We came away with the understanding that the Volta-based Titan V was a new breed of NVIDIA’s prosumer line of video cards, one that essentially encapsulated NVIDIA’s recent datacenter/compute achievements and how they got there. Which is to say, deep learning and neural networking has quickly become the driving force behind NVIDIA GPUs as state-of-the-art compute accelerators, now incorporating built-in hardware and software acceleration for machine learning operations. Deep learning prowess is the calling card of the Titan V and of Volta in general, and that performance is what we will be investigating today.

The most eye-catching of Volta’s new features are the new specialized processing blocks – tensor cores – but as we will see, this is very much integrated with the rest of Volta's microarchitectural improvements and surrounding software/framework support for deep learning (DL) and high performance compute (HPC). Matching up with the NVIDIA Titan V are the Titan Xp and GeForce GTX Titan X (Maxwell), with the AMD Radeon RX Vega 64 also present for some tests.

NVIDIA GPU Specification Comparison

	Titan V	Titan Xp	GTX Titan X (Maxwell)	GTX Titan
CUDA Cores	5120	3840	3072	2688
Tensor Cores	640	N/A	N/A	N/A
ROPs	96	96	96	48
Core Clock	1200MHz	1485MHz	1000MHz	837MHz
Boost Clock	1455MHz	1582MHz	1075MHz	876MHz
Memory Clock	1.7Gbps HBM2	11.4Gbps GDDR5X	7Gbps GDDR5	6Gbps GDDR5
Memory Bus Width	3072-bit	384-bit	384-bit	384-bit

Memory Bandwidth	653GB/sec	547GB/sec	336GB/sec	288GB/sec
VRAM	12GB	12GB	12GB	6GB
L2 Cache	4.5MB	3MB	3MB	1.5MB
Single Precision	13.8 TFLOPS	12.1 TFLOPS	6.6 TFLOPS	4.7 TFLOPS
Double Precision	6.9 TFLOPS (1/2 rate)	0.38 TFLOPS (1/32 rate)	0.2 TFLOPS (1/32 rate)	1.5 TFLOPS (1/3 rate)
Half Precision	27.6 TFLOPS (2x rate)	0.19 TFLOPs (1/64 rate)	N/A	N/A
Integer (INT8)	55.2 TOPS (4x rate)	48.4 TOPS (4x rate)	26.4 TOPS (4x rate)	N/A
Tensor Performance (Deep Learning)	110 TFLOPS	N/A	N/A	N/A
Other Native INT Operations	INT32, DP4A, DP2A	DP4A, DP2A	N/A	N/A
GPU	GV100 (815mm2)	GP102 (471mm2)	GM200 (601mm2)	GK110 (561mm2)
Transistor Count	21.1B	12B	8B	7.1B
TDP	250W	250W	250W	250W
Manufacturing Process	TSMC 12nm FFN	TSMC 16nm FinFET	TSMC 28nm	TSMC 28nm
Architecture	Volta	Pascal	Maxwell 2	Kepler
Launch Date	12/07/2017	04/07/2017	08/02/2016	02/21/13
Price	$2999	$1299	$999	$999

Circling back to NVIDIA's compute endeavors, with Titan V, the Titan brand became closer than ever to workstation-class compute, featuring a high-end compute-centric GPU for the first time: the gargantuan 815 mm^2 GV100. Complete with a workstation-class price tag of $3000, the Titan V doubled-down on high performance compute (HPC) and deep learning (DL) acceleration in hardware and software, while maintaining the fastest graphics performance around. Looking back, it's a far cry from the original Kepler-based GeForce GTX Titan, a jack-of-all-trades video card that acted as enthusiast flagship with full double precision (FP64) compute for prosumers. Up until Titan V, NVIDIA's Titan lineup more-or-less represented that design methodology, where a big GPU served as lynchpin for both compute and consumer lines.

NVIDIA Tesla/Titan Family Specification Comparison

	Tesla V100 (SXM2)	Tesla V100 (PCIe)	Titan V (PCIe)	Tesla P100 (SXM2)
CUDA Cores	5120	5120	5120	3584
Tensor Cores	640	640	640	N/A
Core Clock	?	?	1200MHz	1328MHz
Boost Clock	1455MHz	1370MHz	1455MHz	1480MHz
Memory Clock	1.75Gbps HBM2	1.75Gbps HBM2	1.7Gbps HBM2	1.4Gbps HBM2
Memory Bus Width	4096-bit	4096-bit	3072-bit	4096-bit
Memory Bandwidth	900GB/sec	900GB/sec	653GB/sec	720GB/sec
VRAM	16GB 32GB	16GB 32GB	12GB	16GB
ECC	Yes	Yes	No	Yes
L2 Cache	6MB	6MB	4.5MB	4MB
Half Precision	30 TFLOPS	28 TFLOPS	27.6 TFLOPS	21.2 TFLOPS
Single Precision	15 TFLOPS	14 TFLOPS	13.8 TFLOPS	10.6 TFLOPS
Double Precision	7.5 TFLOPS	7 TFLOPS	6.9 TFLOPS	5.3 TFLOPS
Tensor Performance (Deep Learning)	120 TFLOPS	112 TFLOPS	110 TFLOPS	N/A

GPU	GV100	GV100	GV100	GP100
Transistor Count	21B	21B	21.1B	15.3B
TDP	300W	250W	250W	300W
Form Factor	Mezzanine (SXM2)	PCIe	PCIe	Mezzanine (SXM2)
Cooling	Passive	Passive	Active	Passive
Manufacturing Process	TSMC 12nm FFN	TSMC 12nm FFN	TSMC 12nm FFN	TSMC 16nm FinFET
Architecture	Volta	Volta	Volta	Pascal

With Volta, there's little detail of anything other than GV100 existing, outside of Tegra Xavier’s Volta iGPU, which is also part of Drive PX Pegasus. So as it stands, Volta is only available to the broader public in the form of the Titan V, though depending on the definition of ‘broader public,’ the $9000 32GB Quadro GV100 released in March might fall under that category too.

Remaking of a Titan: Less Flagship, More Compute

Deep learning and compute aside, there are a few more factors involved in this iteration of the Titan brand. NVIDIA has less need to make a name for itself with the Titan line, of which the original GTX Titan did exactly that by invoking the NVIDIA’s K20Xs powering Oak Ridge National Laboratory’s *Titan* supercomputer, and then setting a new high in performance (and price). Nor is there any particular competitive pressure in pricing or performance – the GeForce GTX 1080 Ti has no direct competition while the Pascal-based Titan X/Xp has carved out a $1200 price bracket above the previous $1000 mark.

Meanwhile, it’s fair to assume pushing the reticle limit (815mm^2) on a new process node (12nm FFN) with new microarchitecture and additional HBM2 packaging results in poor-yielding silicon, and thus fewer options for salvage parts, especially if they needed to be validated at enterprise level (i.e. Teslas and Quadros). So a more-prosumer-than-consumer Titan V part would be the best – and only – fit, given that the gaming performance isn’t at the level of $3000. Ultimately, as we’ve discussed prior, NVIDIA seeds academics, developers, and other researchers at a lower cost-of-entry to Tesla V100s, with the feedback contributing to ecosystem support of Volta. And on that note, while Titan V’s non-ECC HBM2 and GeForce driver stack are more consumer oriented, the card still directly benefits from software support with frameworks and APIs as part of NVIDIA’s overall deep learning development efforts. Other than NVLink, Titan V’s main compute functions (FP64, FP16, tensor core) are uncrippled, which makes sense as single node Titan V’s don’t quite cannibalize sales of NVIDIA’s other compute products. If that were to change with the Quadro GV100, cryptomining will ensure that prices are kept apart.

Taking a step back, the approach with Volta doesn’t mesh with NVIDIA’s previous approaches with Pascal and others. Instead of leading with a compute-centric big die design that could naturally cascade down the consumer stack as smaller GDDR5(x) designs for enthusiast graphics, they went for a gargantuan low-yielding die with good amounts of silicon area dedicated to brand-new non-graphics functions (i.e. tensor cores). We noted that tensor cores were a calculated bet, and broadly-speaking it was the usual tradeoff between lower-margin consumer graphics performance against lower-volume compute, one that NVIDIA could easily afford. The past couple years have put NVIDIA in pole position for raw consumer graphics performance and mindshare, while years of continued involvement at the forefront of GPU accelerated deep learning have put them in prime position to implement DL-specialized hardware with corresponding software support.

And as a side note, cryptomining demand has also thrown a wrench in matters, depleting much of the current generation products for extended periods of time. In turn, the consumer market hasn't quite been saturated with current generation video cards, leaving NVIDIA in no rush to push out a new GeForce generation. Though with all the microarchitectural improvements over Pascal, I'm sure that Volta with disabled tensor cores could be levied as a very capable gaming product if necessary – the Titan V is still king of the hill – just not at the same margins as last generation. In any case, NVIDIA quarterly financials continue to cite high Pascal GeForce sales, and like all marquee silicon designers has leapfrogging design teams, the fruits of which we might just see in a few months.

Thinking Deep with GPUs

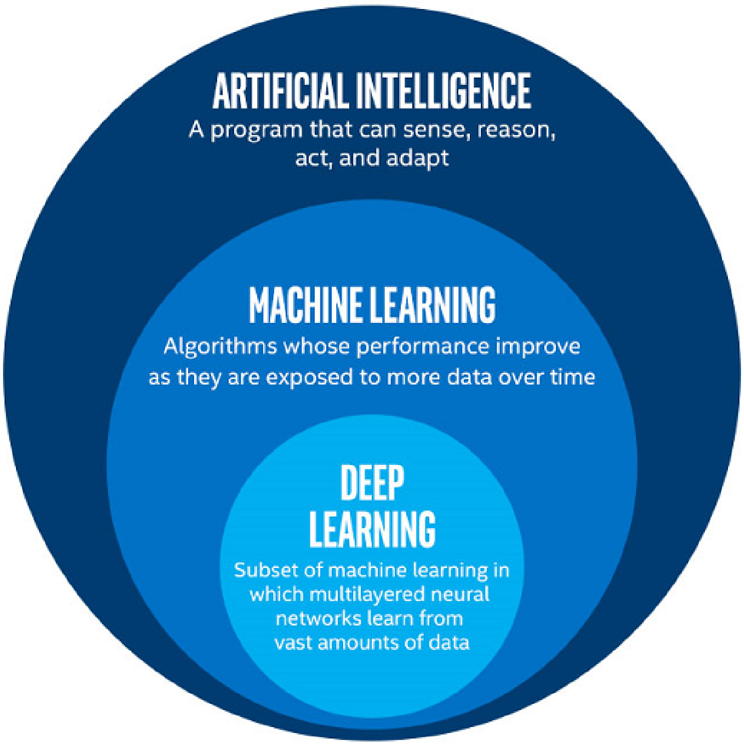
Whatever the case may be with the next generation consumer GeForce, the big picture is that both NVIDIA and AMD have publicly stated the necessity of GPU architecture bifurcation – one for HPC/ML, and one for graphics/gaming. For NVIDIA, considering that Pascal has been around for over 2 years now, Volta is conspicuously absent from recent speculation over the next GeForce generation. In looking at the Titan V today, it almost seems that NVIDIA's divergence is imminent. Even in the case of a Volta-based GeForce launch, the implementation of consumer Volta would be a very big hint at the future direction of GPUs, gaming and compute alike. At the very least, it would be a smaller design with far fewer tensor cores – NVIDIA's RTX technology all but guarantees that at least some tensor cores will show up in consumer parts – and with a GDDR controller, at which point it raises the question how much of Volta was optimized for tensor core operations.

As our first analysis of DL performance of any GPU, we have not yet determined a standard set of benchmark tests, particularly due to Volta's unique tensor cores and mixed precision capability. For this Titan V deep dive, we will be utilizing Baidu DeepBench, as well as tests from NVIDIA's Caffe2 Docker image, Stanford DAWNBench implementations, and HPE Deep Learning Benchmark Suite (DLBS).

But before we dive into the numbers, this is an opportune time to provide some context, of which there is plenty: deep learning and GPUs, the Volta microarchitecture, and the current state of benchmarking DL performance.

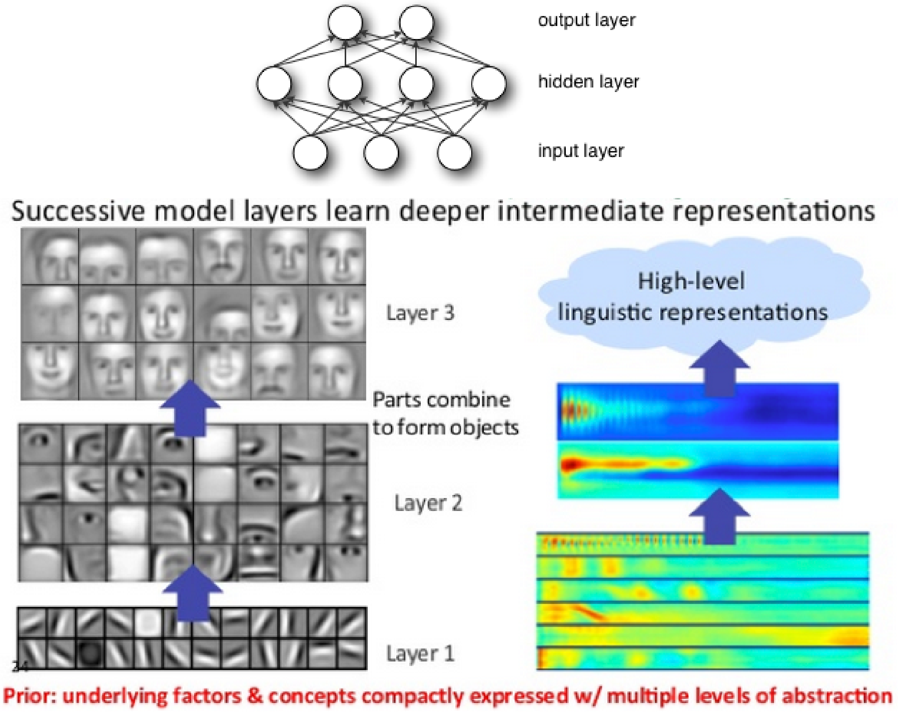
Deep Learning, GPUs, and NVIDIA: A Brief Overview

To get terminology straight, 'machine learning,' or the even more generic term 'AI' is sometimes used interchangeably for 'deep learning.' Technically, they each refer to different things, with ML being a subset of AI, and DL being a subset of ML.


ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Picture from Intel

DL acquires its name from 'deep neural networks,' which are ultimately designed to recognize patterns in data, produce a related prediction, receive feedback on the prediction's accuracy, and then adjust itself based on the feedback. When the feedback is based on an expected known output, this is 'supervised learning.' The computations occur on ‘nodes’, which are organized into ‘layers’: the original input data is first handled by the ‘input layer’ and the ‘output layer’ pushes out data that represents the model’s prediction. Any layers between those two are referred to as ‘hidden layers,’ of which deep neural networks have many hidden layers; originally, ‘deep’ meant having more than one hidden layer.


output layer
hidden layer
input layer
Successive model layers learn deeper intermediate representations
Layer 3
Parts combine to form objects
Layer 2
Layer 1
High-level linguistic representations
Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction

Images from DeepLearning4J

These hidden layers can operate in a hierarchy of increasing abstraction so that they can extract and distinguish non-linear features even from complicated input data. A standard example is in image recognition, where initial layers look for certain edges or shapes, which inform later layers that look for noses and eyes, and layers after that might look for faces. The final layers combine all this data to make a classification.

As input data progresses forward through the model, calculations include special internal parameters (weights). At the end, a loss function is produced, representing the error between the model’s prediction and the correct value. This error information is then used in running the model in reverse to calculate weight adjustments that will improve the model’s prediction. The weights are then updated. This sequence of a forward and backward pass (or backpropagation) comprises a single training iteration.

For inferencing, the process naturally excludes a backward pass and ultimately requires less computational intensity than training the model in the first place. In that sense, inferencing also has less need for higher precisions like FP32, and models can be appropriately pruned and optimized for deployment on particular devices. However, inferencing devices become much more sensitive to latency, cost, and power consumption, especially if on the edge.

Convolutional neural networks (CNNs or convnets) and recurrent neural networks are two important subtypes of (deep) neural networks, and the previous example with image recognition would be seen as a CNN. The convolutions themselves are an operation where input data and convolutional kernel are combined to form a feature map of some kind, transforming or filtering the original data to extract features. CNNs typically are ‘feedforward’, in the sense that data flows through the layers without looping. For RNNs (and variants like LSTM and GRU), there exists a separate weight that loops back to itself after every calculation, giving the net a sense of ‘memory.’ This allows the net to make time-aware predictions, useful in scenarios like text analysis, where a network would need to remember all the previous words with respect to the current one.

As much of deep learning math could be boiled down to linear algebra, certain operations can be re-written into GPU-friendlier matrix-matrix multiplications. When NVIDIA first developed and released cuDNN, one of the marquee implementations was accelerating convolutions based on lowering them into matrix multiplications. Among the cuDNN developments over the years is the 'precomputed implicit GEMM' convolution algorithm, which so happens to be the only algorithm that triggers convolution acceleration by tensor cores.

A Deep Learning Renaissance: (NVIDIA) GPUs Ascendant

Particularly for training, GPUs have become the DL accelerator-of-choice as most of these computations are essentially floating-point calculations in parallel, namely lots of matrix multiplications, with optimal performance requiring large amounts of memory bandwidth and size. These requirements neatly line up with the needs of HPC (and to a lesser extent, professional visualization), where GPUs need high precision floating point computation, large amounts of VRAM, and parallel compute capability.

Perhaps most importantly, is the underlying API and frameworks needed to utilize graphics hardware in this manner. For this, NVIDIA's CUDA had come at the right time, just as deep learning started to regain interest, and was an easy launching point for further development:

> - *Release of CUDA & cuBLAS (2006/2007), and Tesla product line (2007)*
> - *High-profile publications and achievements*
>   - *2009, "Large-scale Deep Unsupervised Learning using Graphics Processors"*
>   - *2012, AlexNet tops ILSVRC2012, running on cuda-convnet with two GTX 580s*
> - *Release of cuDNN, integrated with Caffe dev branch (2014), later integrated by other DL frameworks*

The development of CUDA and NVIDIA's compute business coincided with research advances in machine learning, which had only just re-emerged as 'deep learning' around 2006. GPU accelerated neural network models provided orders-of-magnitude speed-ups over CPUs, and in turn re-popularized deep learning into the buzzword it is today. Meanwhile, NVIDIA's graphics competitor at the time, ATI, was being acquired by AMD in 2006; OpenCL 1.0 itself only arrived in 2009, the same year AMD spun off their fabs as GlobalFoundries.

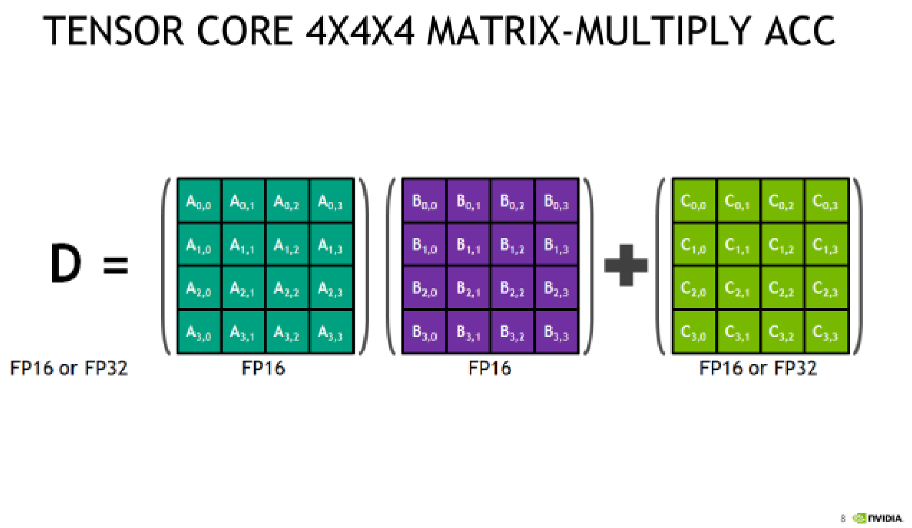
With DL researchers and academics successfully using CUDA to train neural network models faster, it was only a matter of time before NVIDIA released their cuDNN library of optimized deep learning primitives, of which there was ample precedent with the HPC-focused BLAS (Basic Linear Algebra Subroutines) and corresponding cuBLAS. So cuDNN abstracted away the need for researchers to create and optimize CUDA code for DL performance. As for AMD's equivalent to cuDNN, MIOpen was only released last year under the ROCm umbrella, though currently is only publicly enabled in Caffe.

So in that sense, NVIDIA GPUs have become the reference implementation with respect to deep learning on GPUs, though the underlying hardware of both vendors are both suitable for DL acceleration.

A Shallow Dive Into Tensor Cores

Of the several mysteries around Volta's mixed precision tensor cores, one of the more nagging ones was the capability of 4 x 4 matrix multiplication. To recap, the tensor core is a new type of processing core that performs a type of specialized matrix math, suitable for deep learning and certain types of HPC. Tensor cores perform a fused multiply add, where two 4 x 4 FP16 matrices are multiplied and then the result added to a 4 x 4 FP16 or FP32 matrix. The result is a 4 x 4 FP16 or FP32 matrix; NVIDIA refers to tensor cores as performing mixed precision math, because the inputted matrices are in half precision but the product can be in full precision. As it so happens, the math that tensor cores do is commonly found in deep learning training and inferencing.

And if this sounds familiar to a normal GPU ALU pipeline, then it should. Tensor cores, while being brand-new to the GPU space, are not all that far removed from standard ALU pipelines. The density has changed – they're now operating on sizable matricies instead of SIMD-packed scalar values – but the math has not. At the end of the day there's a relatively straightforward tradeoff here between flexibility (tensor cores would be terrible at scalar operations) and throughput, as tensor cores can pack many more operations into the same die area since they are so rigid and require a fraction of the controlling logic when that cost is divided up per ALU.

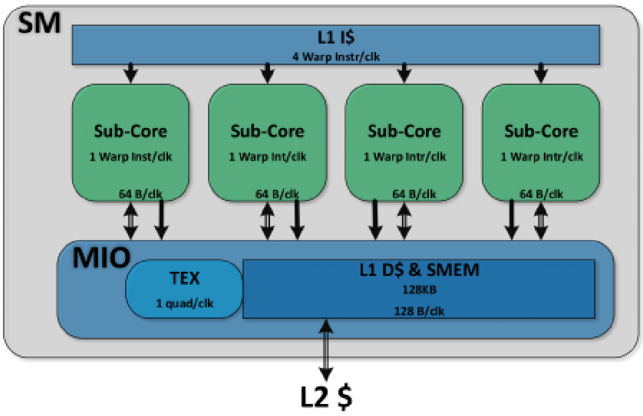

TENSOR CORE 4X4X4 MATRIX-MULTIPLY ACC

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

Consequently, while somewhat programmable, tensor cores are stuck to these types of 4 x 4 matrix multiplication-accumulation – and it's not clear how and when the accumulation step occurs. Despite being described as doing 4 x 4 matrix math, in practice, tensor core operations always seem to be working with 16 x

16 matrices, with operations being handled across two tensor cores at a time. It appears that a lot of it has to do with the other changes in Volta, and more specifically, how these tensor cores are placed in an SM. For Volta, SMs were partitioned into four processing blocks, or sub-cores.

SM MICROARCHITECTURE



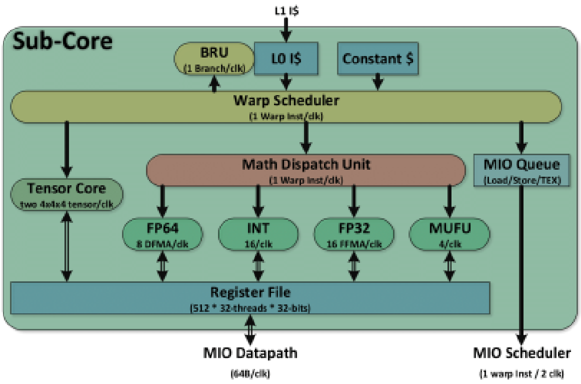
Shared L1 I$

4 Independently Scheduled Sub-Cores

Shared MIO
(TEX/L1$/SMEM)

For each sub-core, the scheduler issues one warp instruction per clock to the local branch unit (BRU), the tensor core array, math dispatch unit, or shared MIO unit. For one, this precludes issuing a combination of tensor core operations and other math simultaneously. In utilizing the two tensor cores, the warp scheduler issues matrix multiply operations directly, and after receiving the input matrices from the register, perform 4 x 4 x 4 matrix multiplies. Once the full matrix multiply is completed, the tensor cores write the resulting matrix back into the register.

SUB-CORE



Warp Scheduler

- 1 Warp instr/clk
- L0 I$, branch unit

Math Dispatch Unit

- Keeps 2+ Datapaths Busy

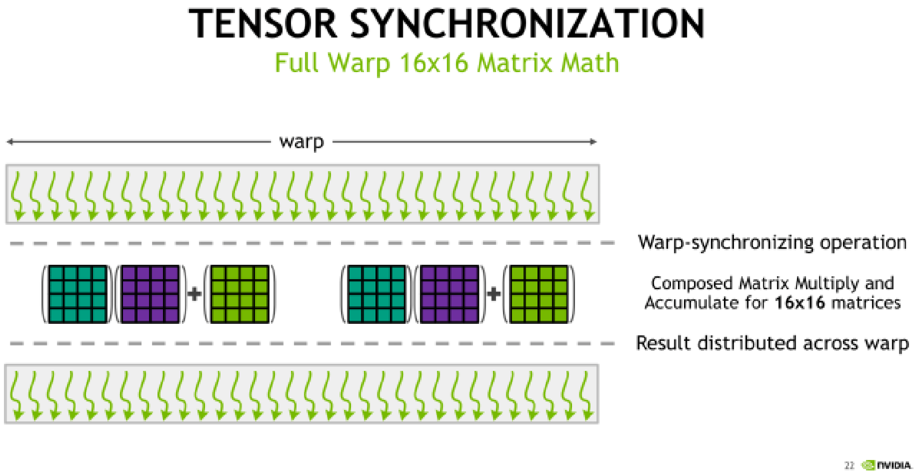
MIO Instruction Queue

- Hold for Later Scheduling

Two 4x4x4 Tensor Cores

Looking at how tensor cores execute the actual instruction, opcode HMMA, only seems to raise more questions. Even at the compiler level with NVVM IR (LLVM), there are only intrinsics for warp-level matrix operations, rather than tensor cores, and warp-level remains the only level with CUDA C++ and the PTX ISA. Loading the input matrices is in the form of each warp thread holding a fragment, whose distribution and identity is unspecified. So broadly-speaking, it follows the same pattern of thread-level tiling-based GEMM computation for standard CUDA cores, and we'll circle back on that with NVIDIA's CUTLASS library in a moment.

In general terms, though, given the A*B+C tensor core operation, fragments consist of 8 FP16x2 elements (i.e. 16 FP16 elements) for A and another 8 FP16x2 elements for B, as well as 4 FP16x2 elements for an FP16 accumulator or 8 FP32 elements for an FP32 accumulator.



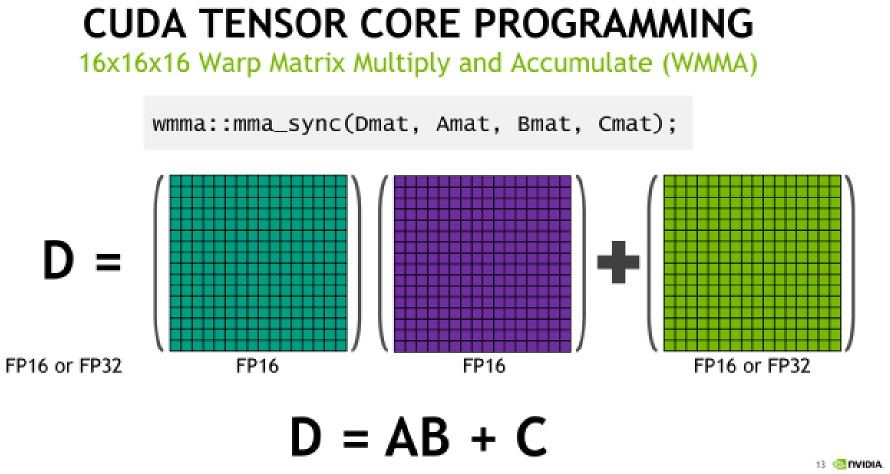
TENSOR SYNCHRONIZATION

Full Warp 16x16 Matrix Math

Warp-synchronizing operation

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

Result distributed across warp



CUDA TENSOR CORE PROGRAMMING

16x16x16 Warp Matrix Multiply and Accumulate (WMMA)

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

$$D = AB + C$$

After the matrix multiply-accumulate operation, the result is spread out in fragments in the destination registers of each thread. Requiring warp-wide unity, these low-level operations essentially fail if one of the warp threads had exited.

Low-level microbenchmarking by a team at Citadel LLC revealed a number of Volta microarchitecture details, including tensor core operations and the fragments involved, both locations in the register and identity compared to the input matrices. They observed that a sub-core proceeds to calculate the matrix-multiply in a particular patchwork pattern, with all 32 threads of the warp in action. Conceptually, the tensor cores operate on 4 x 4 submatrices to calculate the larger 16 x 16 matrix, involving Volta's cooperative groups and new scheduling model.

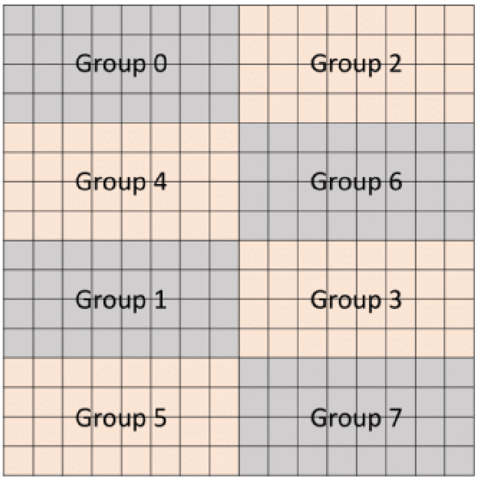


Figure 4.5: Mapping between positions in matrix C and thread group indices.

With the warp threads separated out into 8 thread groups of 4 threads, each thread group computed an 8x4 chunk serially, going through a process of 4 sets. So altogether, each thread group dealt with 1/8 of the resultant matrix.

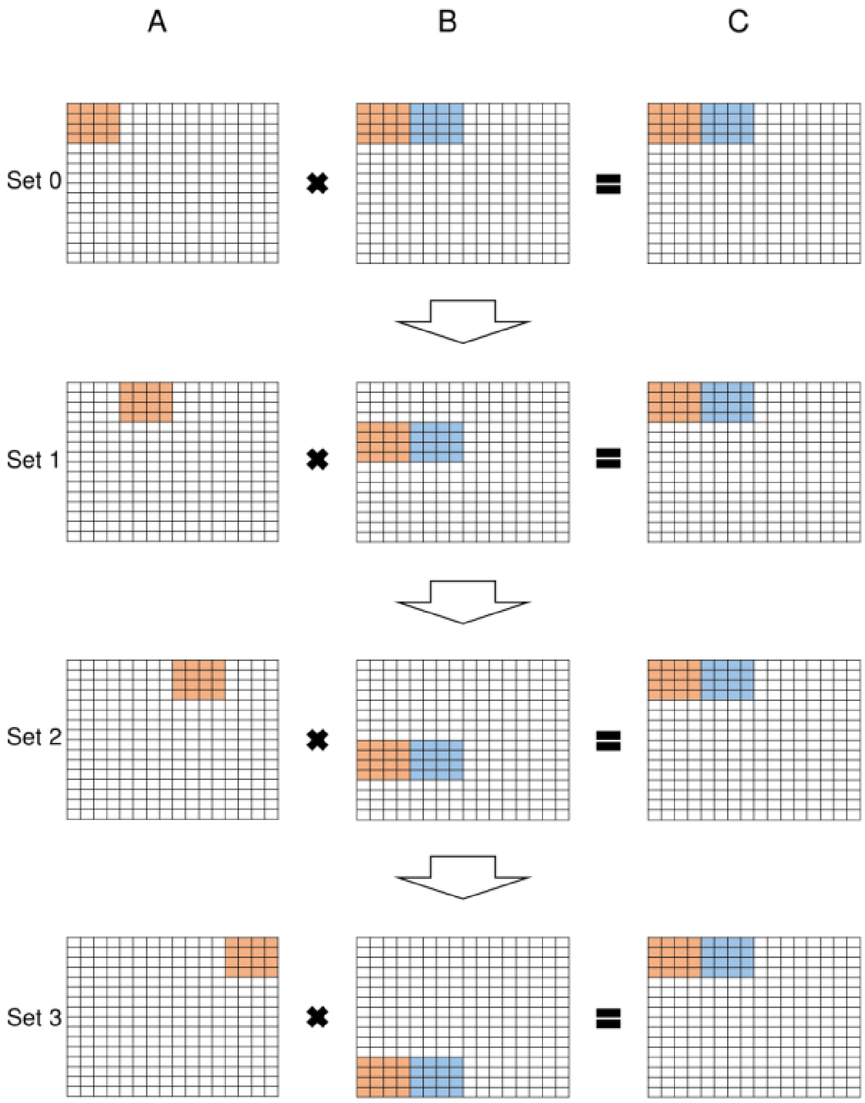


Figure 4.6: Four sets of HMMA instructions complete 4×8 results in matrix C within thread group 0. Different sets use different elements in A and B. The instructions in set 0 execute first, then the instructions in set 1, set 2 and set 3. This way, the 16 HMMA instructions can correctly compute the 4×8 elements in matrix C.

Within a set were four HMMA steps that could be done in parallel, each applying to a 4x2 subchunk. The four threads were directly linked to those matrix values in the register, so that a single Step 0 HMMA instruction could be processed by the thread group to compute the subchunk in one go.

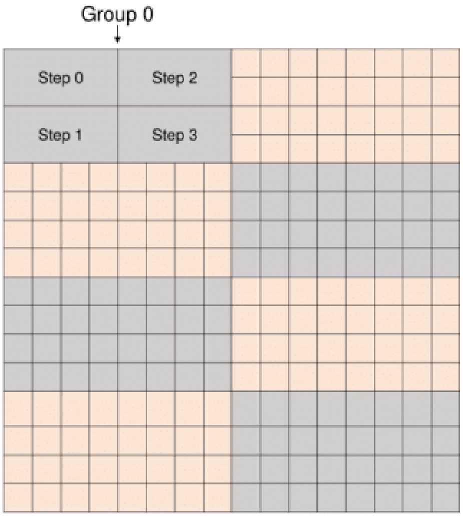
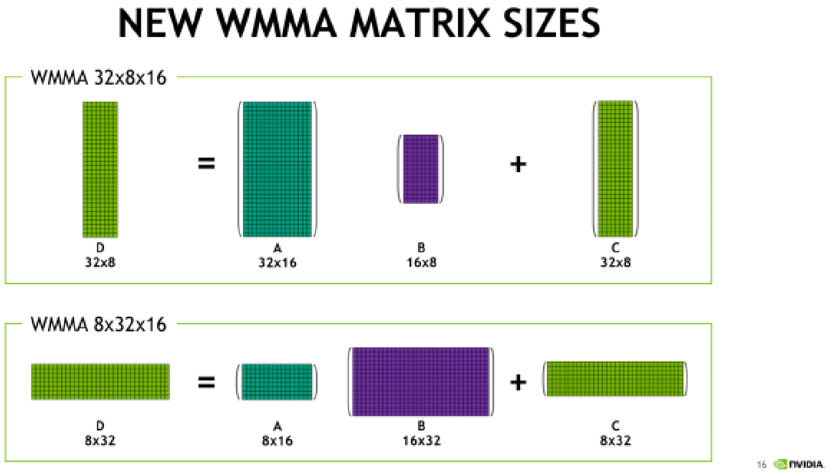


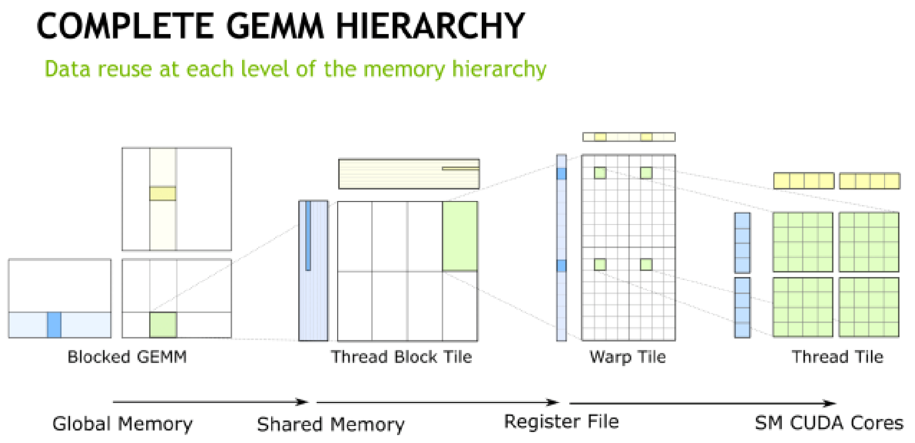
Figure 4.4: 4 steps of HMMA instructions within one set compute different elements in matrix C.

As matrix multiplication mathematically requires reuse of certain columns and rows, to permit parallel execution across all 8x4 chunk each 4 x 4 matrix is mapped to the registers of two threads. If applicable, the accumulate step sums the product with a stored accumulator; in this case of 4 x 4 submatrix operations to calculate a 16 x 16 parent matrix, this would include summing the sets as they were computed serially to form the corresponding chunk of 4 x 8 elements in the 16 x 16 matrix. Though untested by Citadel, it has been observed that FP16 HMMA instructions result in 2 steps rather than 4, relating to the smaller register space that FP16 occupies, and presumably a similar principle applies. Assuming that the sub-core was configured for peak output, it's still hard to estimate without numbers, though it seems like 'FMA ops per cycle' is in reference to the matrices' constituent values.



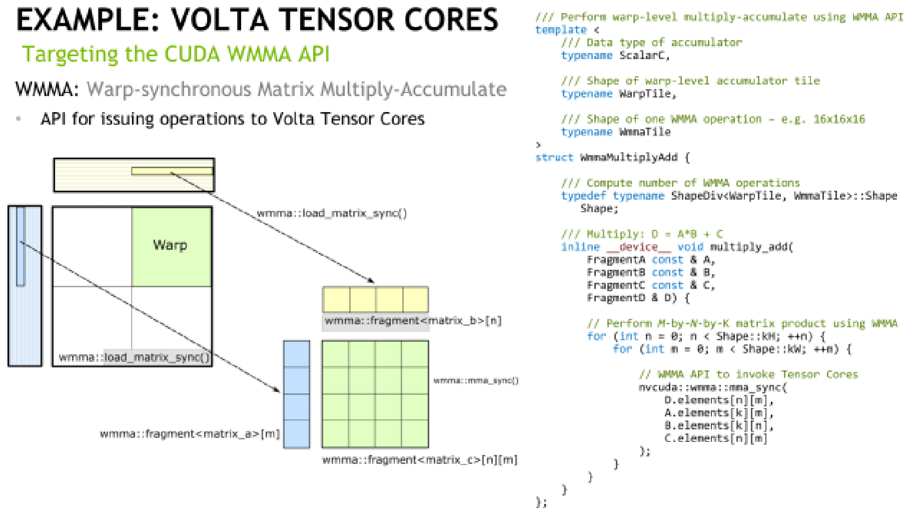
With independent thread scheduling and execution, as well as warp synchronization and warp-wide result distribution, the basic 4 x 4 x 4 tensor core operations translate into semi-programmable 16 x 16 x 16 mixed precision matrix multiply-accumulation, though with CUDA 9.1, 32 x 8 x 16 and 8 x 32 x 16 configurations are supported. For both new shapes, the multiplied matrices need the appropriate corresponding columns and rows of 16, with the end matrix being 32 x 8 or 8 x 32; this more-or-less suggests that standalone 4 x 4 x 4 matrix multiply-accumulate operations can't be easily supported. Hard-coded warp-level behavior of register fragments, implementation of the MMA instruction, or tensor core ALUs could easily result in solely warp-level tensor core matrix math. And from a practical viewpoint, power consumption would suffer due to the increased register file usage while not significantly adding to deep learning performance.

How tensor cores operate seem to be a hardware implemented step of NVIDIA's GEMM computation hierarchy, as seen in CUTLASS, their CUDA C++ template library for GEMM operations. With traditional CUDA cores, the last step requires breaking down (i.e. 'decomposing') the warp tile structure into scalar and vector elements owned by individual threads. With the WMMA API, which right now means tensor cores, all that is abstracted away, leaving only the warp-cooperative matrix fragment load/store and multiply-accumulate to deal with. The accumulation occurs in-place as an FMA-type operation.



COMPLETE GEMM HIERARCHY

Data reuse at each level of the memory hierarchy



EXAMPLE: VOLTA TENSOR CORES

Targeting the CUDA WMMA API

WMMA: Warp-synchronous Matrix Multiply-Accumulate

- API for issuing operations to Volta Tensor Cores

wmma::load_matrix_sync()
Warp
wmma::fragment<matrix_b>[n]
wmma::load_matrix_sync()
wmma::mma_sync()
wmma::fragment<matrix_a>[m]
wmma::fragment<matrix_c>[n][m]

```
/// Perform warp-level multiply-accumulate using WMMA API
template <
    /// Data type of accumulator
    typename ScalarC,

    /// Shape of warp-level accumulator tile
    typename WarpTile,

    /// Shape of one WMMA operation - e.g. 16x16x16
    typename WmmaTile
>
struct WmmaMultiplyAdd {

    /// Compute number of WMMA operations
    typedef typename ShapeDiv<WarpTile, WmmaTile>::Shape
       Shape;

    /// Multiply: D = A*B + C
    inline __device__ void multiply_add(
        FragmentA const & A,
        FragmentB const & B,
        FragmentC const & C,
        FragmentD & D) {

        // Perform M-by-N-by-K matrix product using WMMA
        for (int n = 0; n < Shape::kH; ++n) {
            for (int m = 0; m < Shape::kW; ++m) {

                // WMMA API to invoke Tensor Cores
                nvcuda::wmma::mma_sync(
                    D.elements[n][m],
                    A.elements[k][m],
                    B.elements[k][n],
                    C.elements[n][m]
                );
            }
        }
    }
};
```

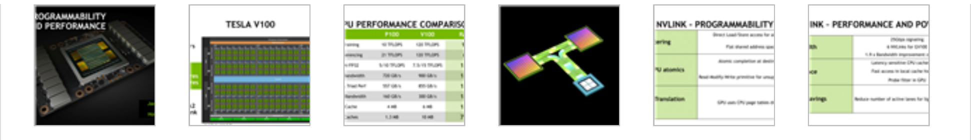
On the register level, NVIDIA themselves mentioned in their Hot Chips 2017 paper that “with three relatively small 4x4 matrices of multiply and accumulator data, 64 multiply-add operations can be performed.” And the per-thread program counter of the enhanced Volta SIMT model, something that enables tensor cores, usually requires 2 register slots per thread according to the whitepaper. There also may have been a change to the register structure; a 2-bank 64-bit configuration was reported by Citadel, though NVIDIA themselves have documented 4-bank 32-bit. The HMMA instructions themselves feature as much register reuse as possible, despite the other Volta enhancements (that we’ll touch on in a moment), so I can’t imagine registers aren’t bottlenecking tensor cores for the majority of cases.

For standalone 4 x 4 matrix multiply-accumulate, I suspect that the tensor core array was not physically designed for it in terms of registers, data paths, and scheduling, such that it is only useable with specific submatrix multiplications (though admittedly I’ve not studied linear algebra in some time).

In any case, from NVIDIA’s point-of-view, Volta isn’t a deep learning ASIC; it is still covering the GPGPU space, and so keeping to CUDA programmable tensor cores for applicability to GEMM/cuBLAS and HPC is only logical. With CUTLASS for CUDA C++, this is even more the case, as its WMMA API support is aimed at enabling tensor core GEMM operations for a broad range of applications. Fundamentally, the development of NVIDIA's deep learning hardware acceleration has much to do with the development of cuDNN (and cuBLAS, to a lesser extent) over the years.

Gallery: GTC 2018: CUTLASS Software Primitives For Dense Linear Algebra

Gallery: Hot Chips 2017: Volta Programmability and Performance



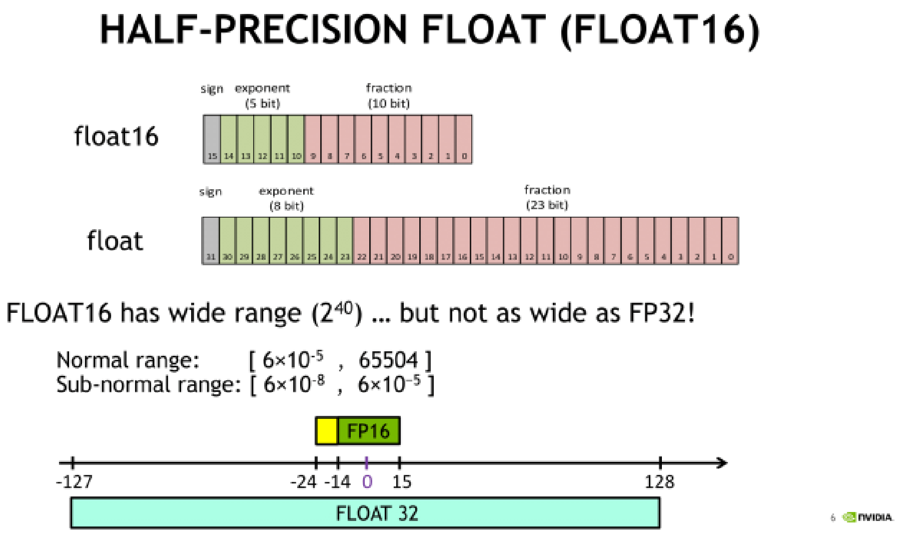
Revisiting Volta: How to Accelerate Deep Learning

While we've gone over Volta's distinguishing characteristics several times now, the marquee addition of tensor cores somewhat overshadows all the other changes that supplement or outright support tensor core usage. For one, as we've already seen, it's tightly tied into the improved SIMT model with Volta's independent thread scheduling and collective groups.

Mixed Precision: Making FP16 Work for Deep Learning

Ultimately, Volta's deep learning prowess is built on utilizing half precision (IEEE-754 FP16) rather than single precision (FP32) for deep learning training. First supported by cuDNN 3 and implemented in Tegra X1's Maxwell cores, native half precision compute was fully introduced with Pascal as "Pseudo FP16", where FP32 ALUs could instead process pairs of FP16 instructions for theoretically double FP16 throughput per clock. We've actually seen this in how tensor cores deal with matrix fragments in the register, as the two FP16 input matrices are gathered in 8 elements of FP16x2, or 16 FP16 elements.

In terms of FP32 versus FP16, because the single precision format 'describes' more data than half precision, operations are more computationally intensive and more memory storage/bandwidth is needed to house and transfer the data, in turn consuming more power. So the successful usage of lower precision in compute has been a poor man's holy grail of sorts, targeting applications where higher precision is unnecessary.

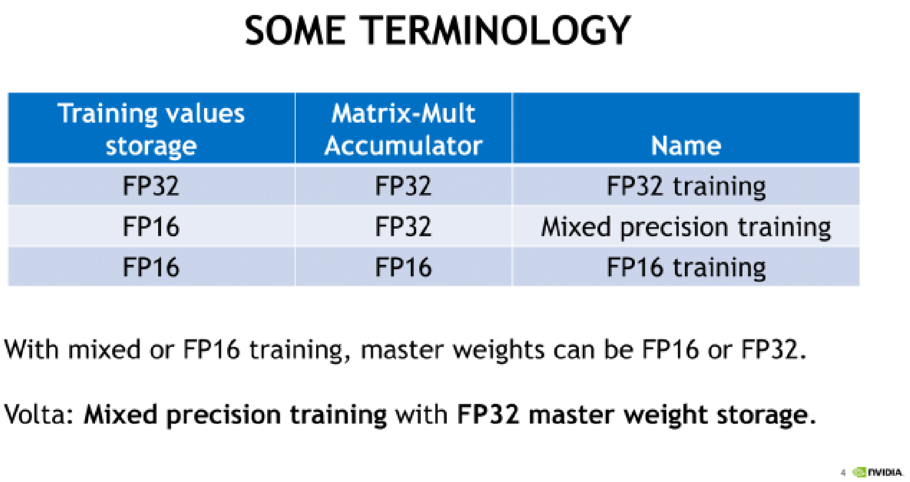


HALF-PRECISION FLOAT (FLOAT16)

FLOAT16 has wide range (2^{40}) … but not as wide as FP32!

Normal range: [6×10^{-5} , 65504]
Sub-normal range: [6×10^{-8} , 6×10^{-5}]

Aside from API/compiler/framework support, the perennial drawback for deep learning is the (unsurprising) loss of precision in using FP16 data types, where the training process would not be accurate enough and so the model cannot converge. Enter mixed precision.



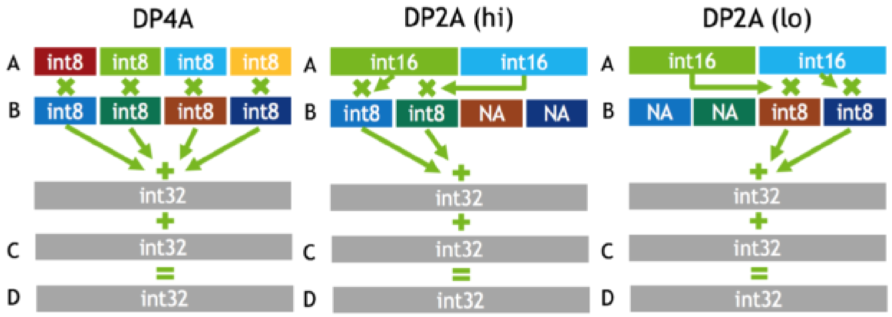
SOME TERMINOLOGY

Training values storage	Matrix-Mult Accumulator	Name
FP32	FP32	FP32 training
FP16	FP32	Mixed precision training
FP16	FP16	FP16 training

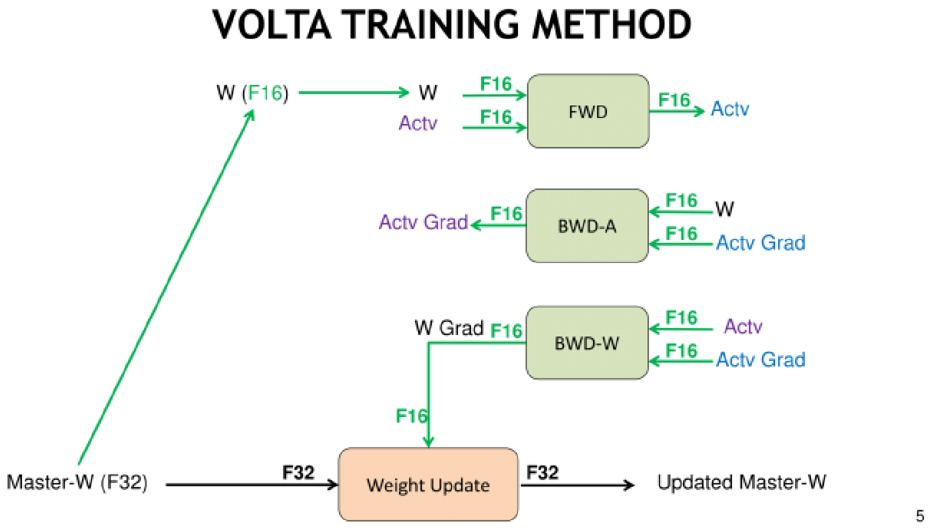
With mixed or FP16 training, master weights can be FP16 or FP32.

Volta: **Mixed precision training** with **FP32 master weight storage**.

To be fair, NVIDIA has wheeled out the 'mixed precision' term before in very similar context, in discussing Pascal's fast FP16 (for GP100) and DP4A/DP2A integer dot product operations (for GP102, GP104, and GP106 GPUs). Back then, the focus was on inference, and very much like Titan V's 'deep learning TFLOPS,' Titan X (Pascal) launched with a "44 TOPS (new deep learning inferencing instruction)." The new instructions performed integer dot products on 4-element 8-bit vectors or 2-element 8-bit/16-bit vectors, resulting in a 32-bit integer product that could be accumulated with other 32-bit integers.

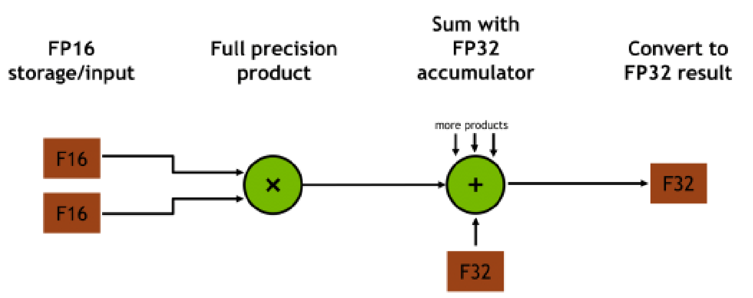


So for mixed precision in Volta, there are several more wrinkles. First is that important precision-sensitive data like master weights are stored as FP32. The second is tensor cores, where mixed precision training describes how two half precision input matrices are multiplied to get a single precision product, which is then accumulated into a single precision sum. NVIDIA has stated that the result is converted back to half precision before being written into memory, though how this happens is not exactly clear. For inferencing purposes though, the tensor core will instead accumulate the result into a half precision sum. Ultimately, when using half precision format, less data is needed in the registers and memory, which helps compensate for the data in very large matrices.



For a given training iteration, Volta mixed precision means the master weights are copied in single and half precision, and while that takes up more memory, NVIDIA believes the accuracy gains are worth it. The half precision weights are used in the ensuing computations, and when the master weights are ready to be updated with the resulting computation, the FP32 copy is used. At that last stage of an iteration, the computed weight updates are converted from FP16 to FP32 in order to update the FP32 master copy of weights, again for accuracy reasons.

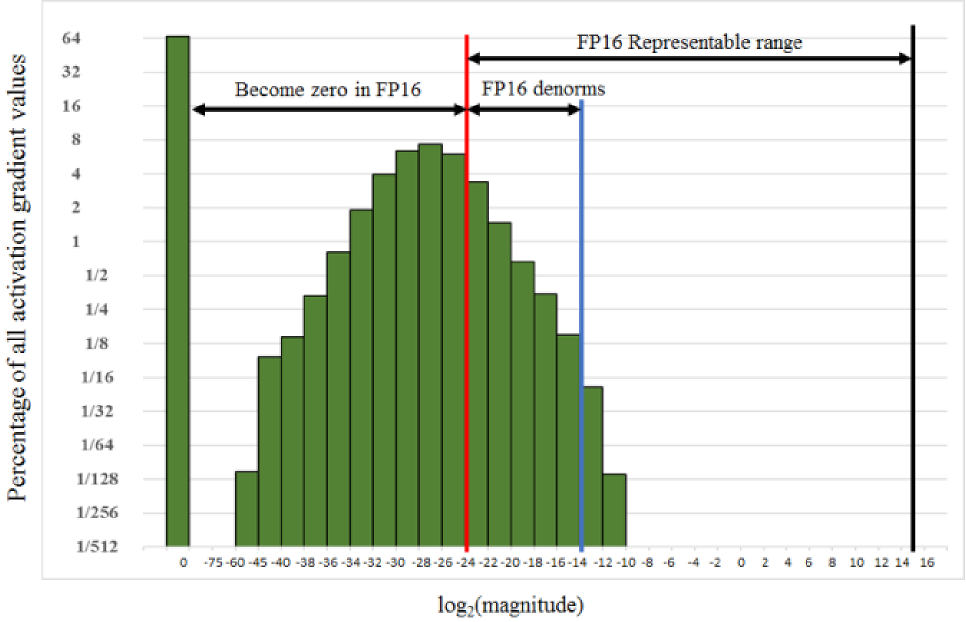
VOLTA TENSOR OPERATION



Also supports FP16 accumulator mode for inferencing

23

Recalling that FP16 does not cover the same data space as FP32, a normalization method can resolve issues where an FP32 value is outside the representable range of FP16 and thus would be converted to a zero. For example, values of many activation gradients would fall outside of the range of FP16, but because these values are clustered together, multiplying the loss with a scaling factor moves most of the values in the range of FP16. The gradients are re-scaled to the original range before weight updates are done, maintaining the original precision.

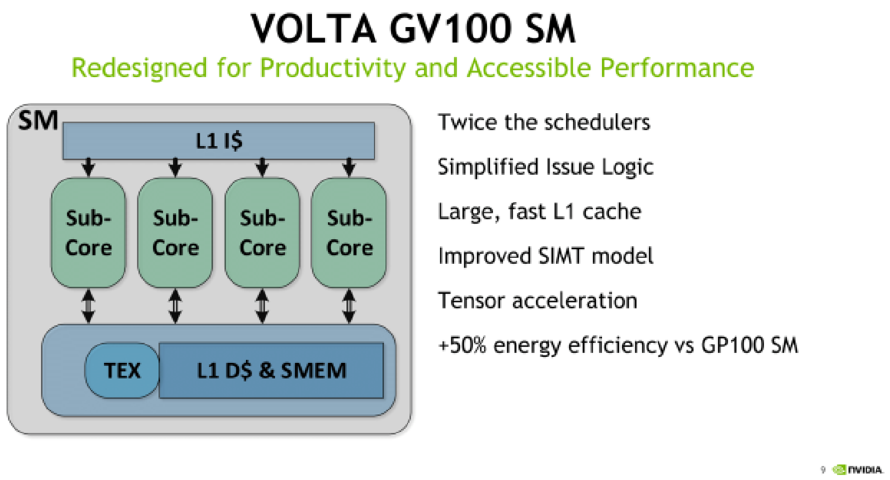


Not all math, neural networks, and layers work well with FP16 storage or math, so depending on the framework or type of neural net, FP16 will either be disabled by default or not recommended. In general, mixed precision with FP16 and tensor cores are best suited with convolution and RNN-heavy image processing and the like. For the most part, cuDNN handles a lot, and developers may only need a few pointers from NVIDIA's Mixed Precision guide. Meanwhile, cuBLAS and CUTLASS also include tensor core support. Altogether, especially with with the maturation of cuDNN it is hard to imagine tensor cores being succesful without it. Intrepid developers can continue trying to wrangle tensor cores directly in CUDA C++, PTX, and the like, though as we have seen tensor cores are, as far as generally programmable GPU blocks go, rather inflexible.

Volta and Pascal: Memory Improvements, SM Changes, and More

With mixed precision tensor cores, it would seem like the memory bandwidth issue was mitigated. As it turns out, not very much, despite the fact that Volta has received memory subsystem enhancements nearly across the board.

For one, Volta now has a 12 KiB L0 instruction cache, and while Pascal and others have had instruction buffers before, Volta's more efficient L0 is private to the sub-core SM partitions. And by that, it is private to the warp scheduler. This compensates for the larger instruction size of Volta's new ISA, and more likely than not, contributes to the framework supporting tensor core thoroughput, which uses the presumably beefy HMMA on a warp-based level. Instruction latency is also reduced from Pascal, notably with core FMAs down to 4 cycles from 6, which we previously confirmed.



VOLTA GV100 SM

Redesigned for Productivity and Accessible Performance

Twice the schedulers

Simplified Issue Logic

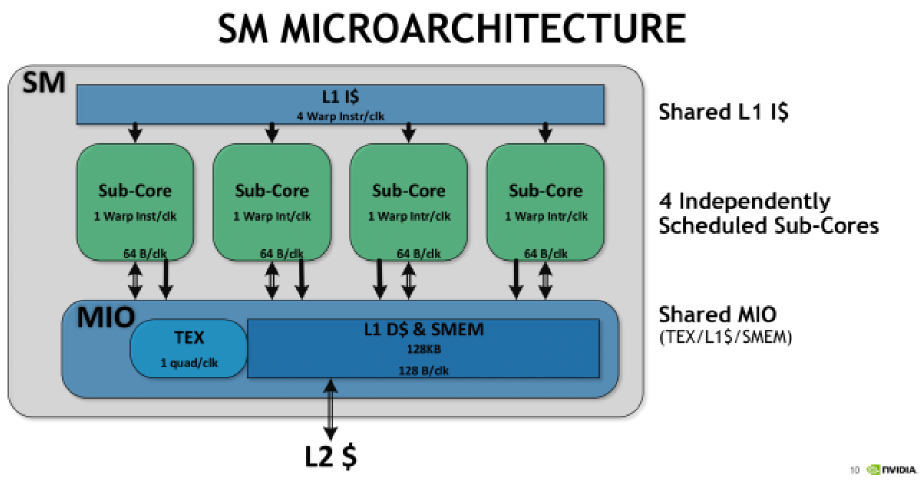
Large, fast L1 cache

Improved SIMT model

Tensor acceleration

+50% energy efficiency vs GP100 SM

With the ratio of schedulers per SM increased, the loss of the second dispatch port seems to be a tradeoff in favor of independent sub-core with separate data paths and math dispatch unit; with simultaneous FP32/INT32 execution capability, it also opens the door to other lower precision/mixed precision models. Overall, the sub-core enhancements that we detailed earlier look to optimize the tensor core array.



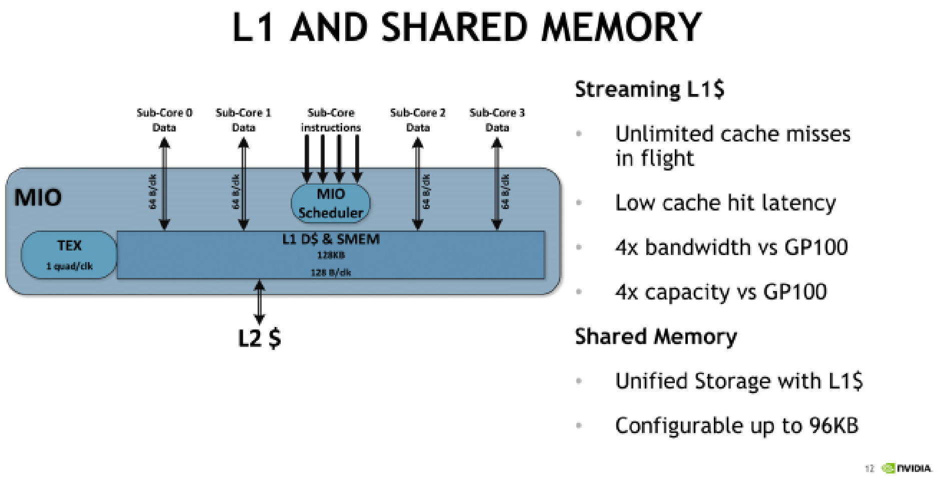
SM MICROARCHITECTURE

Shared L1 I$

4 Independently Scheduled Sub-Cores

Shared MIO (TEX/L1$/SMEM)

Another big change was merging the L1 cache and shared memory. While in the same block, the shared memory is configurable up to 96 KiB per SM. The HBM2 controller was also updated, and NVIDIA and others have noted 10 - 15% increase in efficiency.



L1 AND SHARED MEMORY

Streaming L1$

- Unlimited cache misses in flight
- Low cache hit latency
- 4x bandwidth vs GP100
- 4x capacity vs GP100

Shared Memory

- Unified Storage with L1$
- Configurable up to 96KB

Summing up the SM, Volta looks to be building around a new style of independent partition that supports tensor cores, and one leaning far more on the compute side than on gaming.

A Look at the Deep Learning Benchmark Landscape

For a new and sometimes impenetrable field like deep learning, where so much is customized to the hardware-at-hand – from frameworks and models to APIs and libraries – it's no surprise that there is little in the way of industry-accepted and publicly accessible benchmarking tools. Like HPC, much of its roots are in academic research, but deep learning's GPU-led arrival into the workstation-class hardware space is new. In a short time, we've heard of and seen deep learning datacenters, deep learning software, and basically every hardware implementation, running the gamut from CPUs, GPUs, and SoCs, to ASICs, FPGAs, and just about anything else you can fab on silicon.

So the applications of deep learning are less familiar to end-users, except when used as buzzwords to describe future products or current devices with mediocre inferencing capabilities. But ultimately, because deep learning encapsulates both training and inferencing, it has legitimate reason to include all types of hardware. That's partially what makes it so enticing, though the situation is somewhat of a chicken-and-egg scenario; cryptomining and blockchains were treated very differently before the latest surge in popularity (and infamy).

In terms of benchmarking GPUs for traditional HPC and workstation performance, there are several standardized suites (e.g. SPECviewperf, SiSoftSandra) that produce relatively consumer-accessible data, not to mention direct comparisons to real-world performance in ISV workstation software. This is not the case here.

Modern DL Benchmarking

The past couple years has seen a renewed effort to create a type of external benchmark suite, but the mainstays have been many of the reference implementations of DL frameworks like TensorFlow. With the impact of ImageNet and some of the models that have emerged from it (AlexNet, VGGNet, Inception, Resnet, to name a few), training on the Imagenet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) image dataset is considered to be an industry-standard task of sorts.

As it plays out in the media, reference models in framework repositories are often run in isolation and are offered up as raw peak throughput numbers; for image recognition, this would be 'images trained per second.' Though given the amount of configuration sometimes needed, this is understandable.

The recent releases of third-party deep learning benchmark suites very much look to solve that issue with standardization and accessible data. Of these, Fathom and TBD are more conventional benchmark suites with tests configured for specific frameworks and models, covering many of the different machine learning applications. Meanwhile, the recent Deep Learning Frameworks focuses on comparing performance for a given model and dataset across frameworks.

Comparison of Selected Deep Learning Benchmark Suites

Test Suite	Published	Collaborators
Fathom	9/2016	Harvard University C-FAR
Baidu DeepBench	9/2016	Baidu Research NVIDIA, Intel, Arm, AMD
Stanford DAWN Deep Learning Benchmark (DAWNBench)	11/2017	Stanford DAWN Project (incl. Intel, Microsoft, and Google)
HPE Deep Learning Benchmark Suite (DLBS)	11/2017	HPE
Training Benchmark for DNNs (TBD)	3/2018	University of Toronto Microsoft Research
Deep Learning Frameworks Comparison	3/2018	Microsoft Machine Learning
MLPerf	5/2018 ("Alpha")	Harvard, Stanford, Berkeley, University of Minnesota, University of Toronto Google, Baidu, Intel, AMD, and others

As for the bulk of our results today, DeepBench does not use frameworks per se, instead using low-level libraries to evaluate performance of machine learning operations across devices and machines with various preset kernels. On its own, while it does not directly implicate framework/model/application performance as other tests, instead it provides metrics that are representative of mathmatical operations and hardware capability as optimized by vendors; the binaries for each product are compiled with libraries that the hardware vendors (NVIDIA, Intel, Arm, AMD) provide and implement. This allows us to have a point-of-comparison between devices independent of frameworks and datasets.

One of the more different ones is DAWNBench, which is not so much a benchmark suite as it is a competition-like reporting of training and inference results for three datasets: ImageNet, CIFAR10, and

SQuAD. The focus here is on real-world applicable data, namely end-to-end metrics of computation time-to-accuracy and cost, as opposed to raw accuracy or thoroughput.

For HPE DLBS, as part of HPE's Deep Learning Cookbook, it is largely GPU-focused and sticks to TensorFlow, MXNet, PyTorch, and Caffe-type frameworks, and additionally includes TensorRT testing. While the implementation has well-featured multi-test batching, logging, monitoring, and reporting, it outputs purely performance and time metrics, without any end-to-end measurements of time-to-accuracy or cost.

The most recent high profile benchmark suite, MLPerf, includes researchers and engineers previously working on DAWNBench and other suites; for all intents and purposes, the DAWNBench project has now been superseded by MLPerf. Explicitly aspiring to do for machine learning what SPEC does for general-purpose compute and TPC does for database systems, MLPerf is looking to include Fathom's approach with cross-domain ML tests, as well as DAWNBench's focus on end-to-end computation time of a model above a threshold accuracy. Being so new, however, it is currently on an alpha release, and the reference benchmarks are stated as not suitable for accurate hardware comparisons. For that reason, we have not incorporated any MLPerf testing in this review.

Modern DL being such a new and rapidly changing field, new benchmarks appear quickly, perhaps even as recent as last week. And old ones drift away, like the defunct DeepMark (also created by Chintala) and BenchIP. But for MLPerf, it does seem to be building off of all the lessons learned prior.

Benchmark Accuracy and Metrics

The Deep Learning Frameworks Comparison benchmark allows us to bring up a useful point: differences between frameworks can easily lead to unintended consequences, and thus invalid benchmarks, which affected the Deep Learning Frameworks Comparison as mentioned by Yuxin Wu (creator of tensorpack for TensorFlow). And in general, most DL benchmarks are invalid or less accurate in this way – something that Soumith Chintala (creator of convnet-benchmarks and PyTorch) noted. In the end, without a background in machine learning, there is no easy way of independently validating the accuracy and scope of DL benchmarks, which the MLPerf project appears to try to address.

Another issue is the difficulty in tracking down model variants or reproducing published results; many times, benchmark implementations originate from publications, reference model implementations, or otherwise ML competitions like Kaggle.

For our purposes though, the situation is slightly different, as we are testing GPU performance rather than framework or model performance. But ultimately un-optimized benchmarks would skew GPU performance results anyhow. For these reasons, micro-benchmarks such as DeepBench and 32-bit CNN benchmarks can still be useful in comparing performance between GPUs and between hardware vendors.

Models, Frameworks, and Datasets

The other factor is the sheer amount of deep learning models, frameworks, and datasets. Fortunately, benchmark suites tend to use the same models and datasets, and with competition-style suites like DAWNBench, forgo a mandated framework or model altogether.

As far as frameworks go, essentially all modern DL frameworks support CUDA and cuDNN. For Volta, all frameworks with FP16 storage support also support tensor core acceleration; if FP16 storage is enabled, tensor core acceleration is automatically enabled as well. We will want to utilize these frameworks in order to look at tensor core performance.

Comparison of Selected Deep Learning Benchmark Frameworks

Framework	Support for cuDNN	Support for FP16 Storage	Support for Tensor Core Math
NVCaffe	Yes	Yes	Yes

Caffe2	Yes	Yes	Yes
MXNet	Yes	Yes	Yes
PyTorch	Yes	Yes	Yes
Torch	Yes	**No**	**No**
Chainer	Yes	~~**No**~~ **Yes**	~~**No**~~ **Yes**
TensorFlow	Yes	Yes	Yes
Theano	Yes	Yes	Yes
Microsoft Cognitive Toolkit (formerly CNTK)	Yes	~~**No**~~ **Yes**	~~**No**~~ **Yes**

Update (7/16/2018): Microsoft reached out to clarify that CNTK has supported FP16 and tensor cores since 2.4, which released in January 2018. The information was originally sourced to NVIDIA's Mixed Precision Training Guide, and Microsoft is working with NVIDIA to correct this. In light of this, we have found that Chainer 4 supports FP16/tensor cores to some degree since at least April 2018.

That being said, just because a framework can exploit FP16 storage and tensor cores, doesn't mean it will; the mixed precision guidelines we discussed earlier are very much applicable. A benchmark or test on a given model is not necessarily configured to utilize FP16 and tensor cores out-of-the-box, even if it is built on a compatible framework. And even if it is, the model may not converge without further modification.

In the future, we can look forward to interoperable framework formats like ONNX and NNEF as another datapoint.

The Test

For our purposes, we have utilized the full Baidu DeepBench for a single GPU, a reference benchmark from NVIDIA's Caffe2 Docker image, submissions for Stanford DAWNBench, and benchmarks from HPE DLBS. Altogether, this offers a low-level look into the Titan V, as well as real-world performance, as well as a glance at NVIDIA's TensorRT inference optimizer.

Outside of DeepBench, all tests were done in Docker images. Configuring and troubleshooting ROCm/HIP/MIOpen beyond DeepBench was beyond the scope of this article, and so the Radeon RX Vega 64 only features in the DeepBench tests.

Overview of Conducted Deep Learning Tests

Parent Suite/Test	Type	Dataset	Model	Framework	Tensor Core Aware
DeepBench Dense Matrix Multiplies	Training Inference		N/A		Yes
DeepBench Convolutions	Training Inference		N/A		Yes
DeepBench Recurrent Layers	Training Inference		N/A		Yes
DeepBench Sparse Ops	Inference		N/A		N/A
NVIDIA Caffe2 Docker ImageNet Training	Training	ILSVRC2012 (ImageNet)	ResNet-50 (CNN)	Caffe2	Yes
HPE DLBS Caffe2	Training Inference	ILSVRC2012 (ImageNet)	ResNet-50	Caffe2	Yes
HPE DLBS TensorRT	Inference	ILSVRC2012 (ImageNet)	ResNet-50	TensorRT	Yes

DAWNBench CIFAR10 Image Classification	Training	CIFAR10	Custom ResNet34 Custom ResNet18	PyTorch	No

For one, we are limited by our single-node, single-GPU configuration, as well as the need for regression testing. In that sense, multi-day training runtimes are not ideal, particularly as on older hardware this might translate into multi-week runtimes and non-convergence.

As our first foray into deep learning performance on GPUs, we do not expect this to be the most optimal test lineup, and we welcome constructive criticism on our ongoing deep learning investigations.

Software Configurations

The testbed was put in non-graphical mode when running benchmarks, so that the GPU was not additionally rendering a desktop environment. For the implementations of the two DAWNBench CIFAR10 submissions, we utilized later versions and lightly modified them for easier logging/use (models, optimizers, parameters, etc., were untouched). Docker images were pulled from NVIDIA GPU Cloud (NGC).

Deep Learning Tests Comparison

Test		Software Versions
DeepBench	**NVIDIA**	CUDA 9.1.85 CuDNN 7.1.3 NVIDIA Driver 390.30
	AMD	ROCm 1.8.118 MIOpen-HIP 1.3.0 rocBLAS 0.13.2.1
NVIDIA Caffe2 Docker ImageNet Training		NGC Docker Image: Caffe 18.04-py2
DAWNBench Image Classification Submissions		NGC Docker Image: PyTorch 18.04-py3
HPE DLBS		NGC Docker Image: Caffe2 18.04-py2 PyTorch 18.04-py3

Citations

Baidu DeepBench

Baidu Research. **DeepBench: Benchmarking Deep Learning operations on different hardware**. *https://github.com/baidu-research/DeepBench*

ImageNet (ILSVRC2012)

Olga Russakovsky and Jia Deng (equal contribution), Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. **ImageNet Large Scale Visual Recognition Challenge**. *International Journal of Computer Vision (IJCV).* **2014**, *115,* 211-252. https://arxiv.org/abs/1409.0575

Stanford DAWNBench

Cody A. Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. **DAWNBench: An End-to-End Deep Learning Benchmark and Competition**. *NIPS ML Systems Workshop 2017*. https://dawn.cs.stanford.edu/benchmark/papers/nips17-dawnbench.pdf

CIFAR10

Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. University of Toronto, 2009.

KervResNet

Chen Wang. https://github.com/wang-chen/KervNets

Basenet (ResNet18 with Modifications)

Ben Johnson. https://github.com/bkj/basenet/

Benchmarking Testbed Setup

Our hardware has been modified for deep learning workloads with a larger SSD and more RAM.

CPU:	Intel Core i7-7820X @ 4.3GHz
Motherboard:	Gigabyte X299 AORUS Gaming 7
Power Supply:	Corsair AX860i
Hard Disk:	Intel 1.1TB
Memory:	G.Skill TridentZ RGB DDR4-3200 4 x 16GB (15-15-15-35)
Case:	NZXT Phantom 630 Windowed Edition
Monitor:	LG 27UD68P-B
Video Cards:	***NVIDIA Titan V*** NVIDIA Titan Xp NVIDIA GeForce GTX Titan X (Maxwell) AMD Radeon RX Vega 64
Video Drivers:	NVIDIA: Release 390.30 for Linux x64 AMD:
OS:	Ubuntu 16.04.4 LTS

With deep learning benchmarking requiring some extra hardware, we must give thanks to the following that made this all happen.

Many Thanks To...

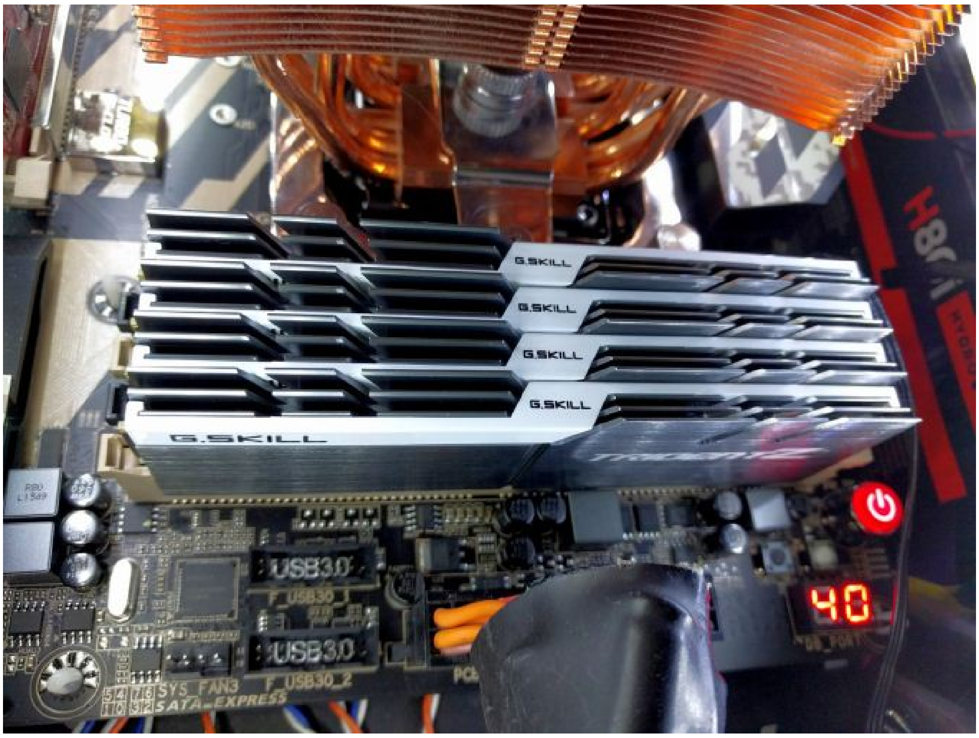
Much thanks to our patient colleagues over at Tom's Hardware, for both splitting custody of the Titan V and lending us their Titan Xp and Quadro P6000. None of this would have been possible without their support.

Buy G.SKILL 2x16G DDR4 3600 on Amazon.com

And thank you to G.Skill for providing us with a 64GB set of DDR4 memory suitable for deep learning workloads, not a small feat in these DDR4 price-inflated times. G.Skill has been a long-time supporter of AnandTech over the years, for testing beyond our CPU and motherboard memory reviews. We've reported on their high capacity and high-frequency kits, and every year at Computex G.Skill holds a world overclocking tournament with liquid nitrogen right on the show floor.

Buy G.SKILL 2x8G DDR4 3000 on Amazon.com

Further Reading: AnandTech's Memory Scaling on Haswell Review, with G.Skill DDR3-3000

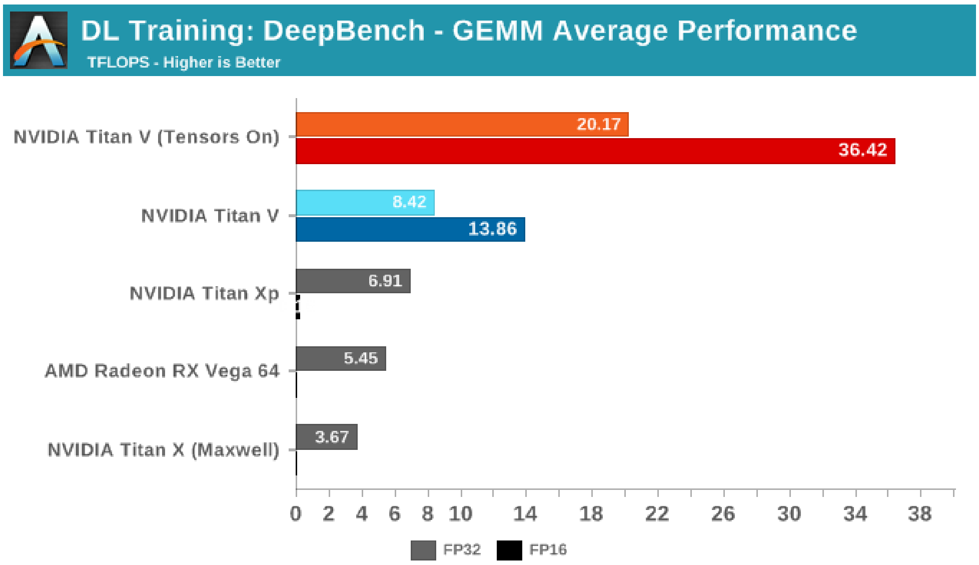


DeepBench Training: GEMM and RNN

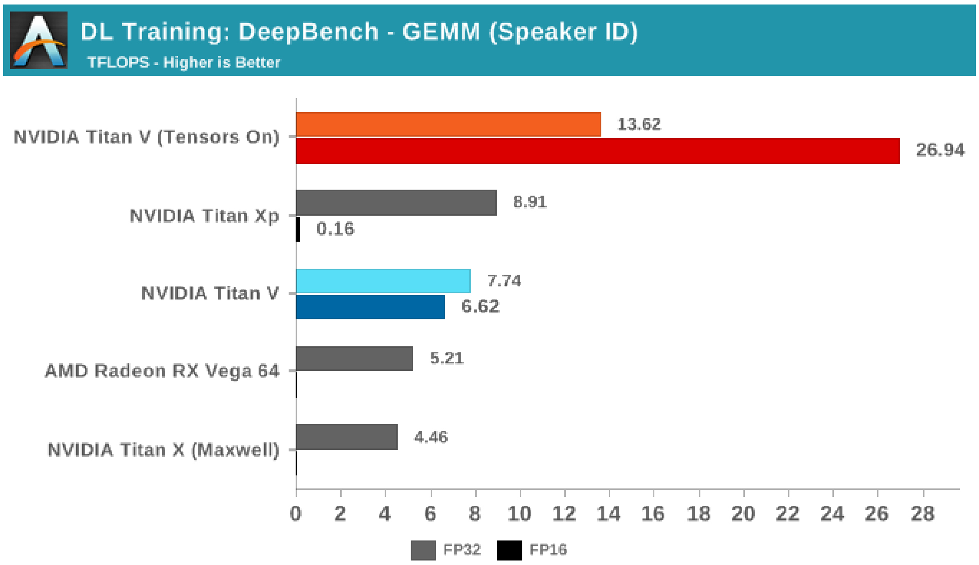
Opening up our DeepBench results are GEMM tests, which we've already seen as pure synthetic operations. Using kernels and GEMM operations used in certain deep learning applications (DeepSpeech, Speaker ID, and Language Modelling), performance here is a little more representative than running through pure matrix-matrix multiplications in cuBLAS.

To preface, the NVIDIA Titan Xp has crippled half precision, while the GeForce GTX Titan X (Maxwell) only supports single precision. According to Baidu, they test a FP32 with tensor cores mode for Volta, where 32-bit inputs undergo 16-bit multiplication and 32-bit accumulation. The specifics of this are somewhat unclear, but we've gone ahead and included those results. Otherwise, FP16 with tensor cores is the 'standard' Volta mixed precision.

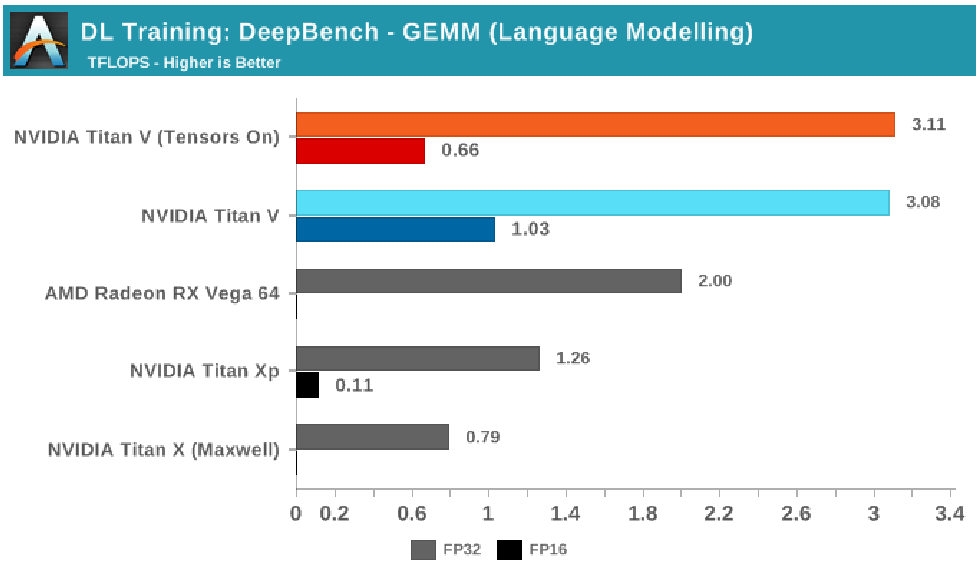
The average results of all sub-tests seem unsurprising: enabling tensor cores results in large performance increases all around. Digging into the details reveals just how specific tensor core acceleration is to certain types of matrix-matrix multiplications.



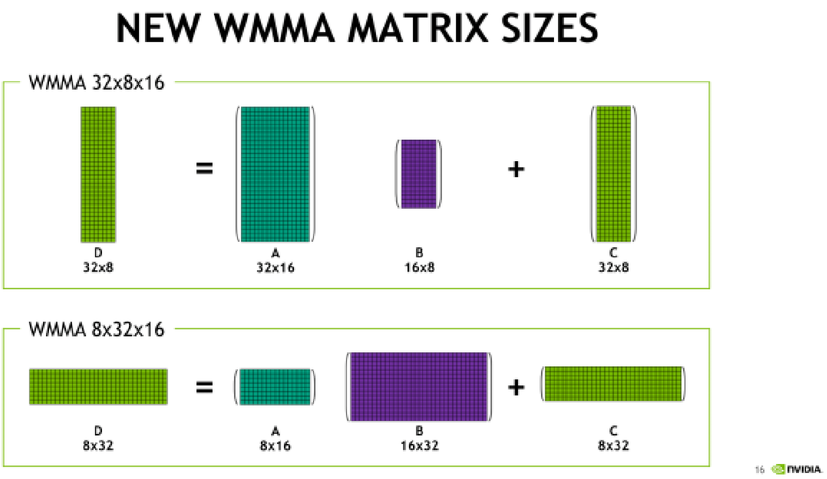
Splitting up the GEMM tests by DL application, we can start to see how tensor cores fare in ideal (and non-ideal) circumstances.



The Speaker ID GEMM workloads actually consist of only two kernels, where a difference of 10 microseconds means a difference of around 1 TFLOPS. The Titan Xp's higher performance is normal variance.

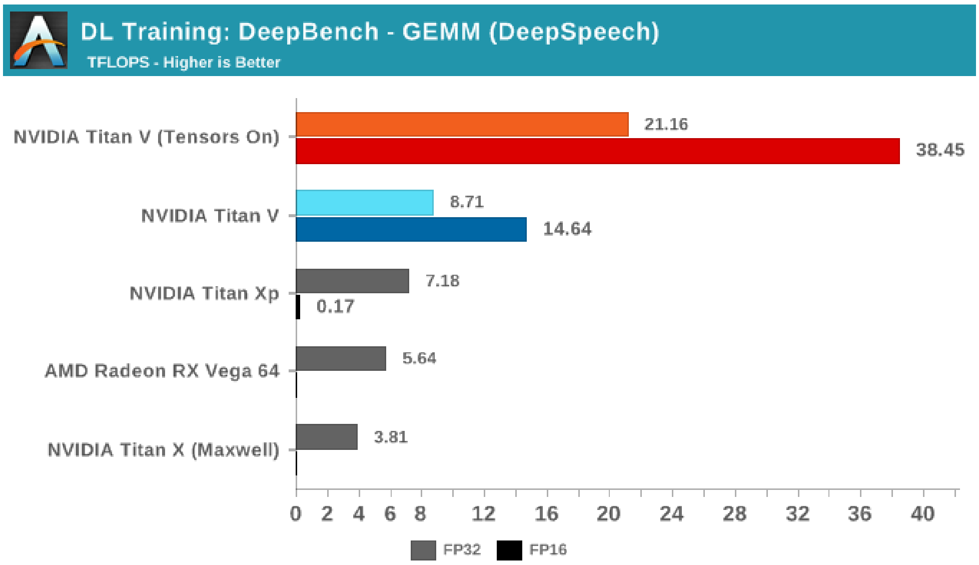


Looking into the Language Modelling kernels explains the poor performance of tensor cores. The sizes of those kernel matrices are m = 512 or 1024, n = 8 or 16, and k = 500000, and the small size of n compared to the very large k is notable. While each number is technically divisible by 8 – one of the basic requirements to qualify for tensor core acceleration – the shape of these matrices isn't a neat fit with the supported basic WMMA shapes: 16x16x16, 32x8x16, and 8x32x16. Nor does it go well with 8x8x8, if we're assuming that tensor cores truly operate on an independent 8x8x8 level.



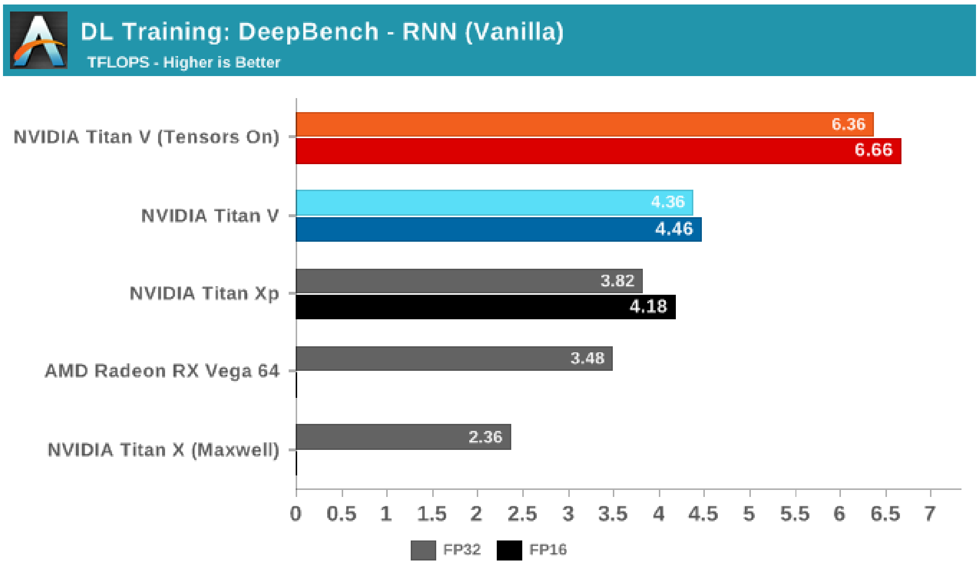
So tensor cores are being pulled into action on very lopsided matrices that can't be broken up easily as n = 8 or 16, at least, not without performance penalties.

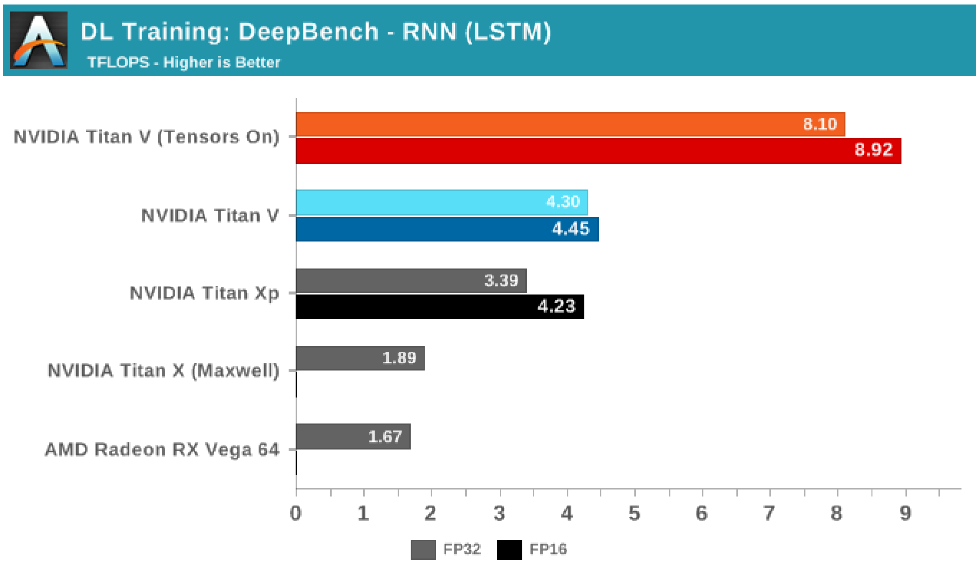
Meanwhile, the tensor cores have runaway performance on DeepSpeech kernels:

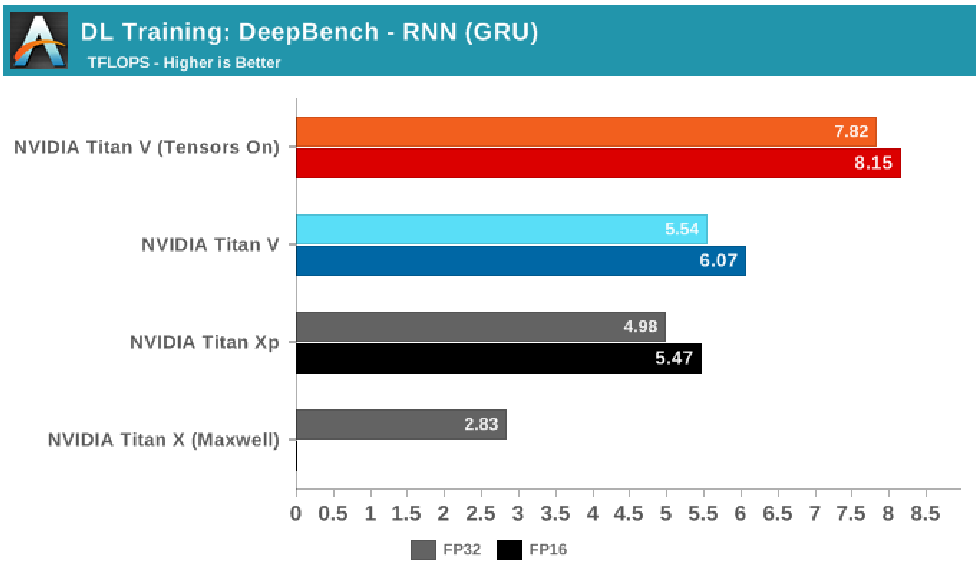


As an average, it turns out to be an impressive number of TFLOPs. Granularly, the same impact of wrong-proportioned matrices is occuring. When matrices fit the tensor core proportions, performance can jump to more than 90 TFLOPS. When they don't, and the right transpositions are not in play, then performances can drop to below 1 TFLOPS.

For the DeepBench RNN kernels, there is no drastic divergence between RNN type. But within in each RNN type, the same patterns can be seen if judging kernel-by-kernel.







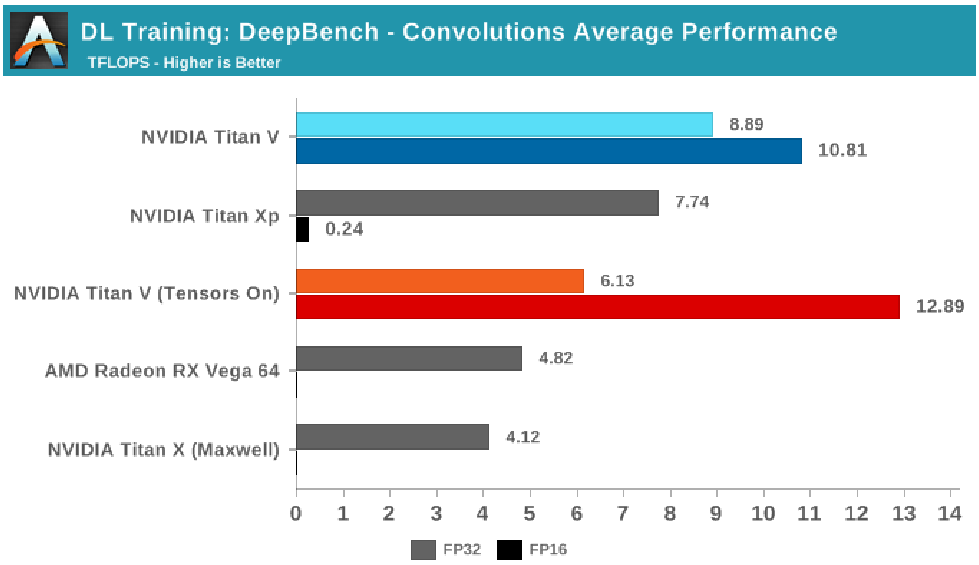
What is interesting is how close the Titan Xp can come to non-tensor core accelerated Titan V performance. We know that the Titan Xp's superior clockspeeds are doing it a favor here, but also one of Titan V's big advantages – HBM2 – is partially mitigated by its 3 controller configuration. Bandwidth-to-bandwidth, the Titan

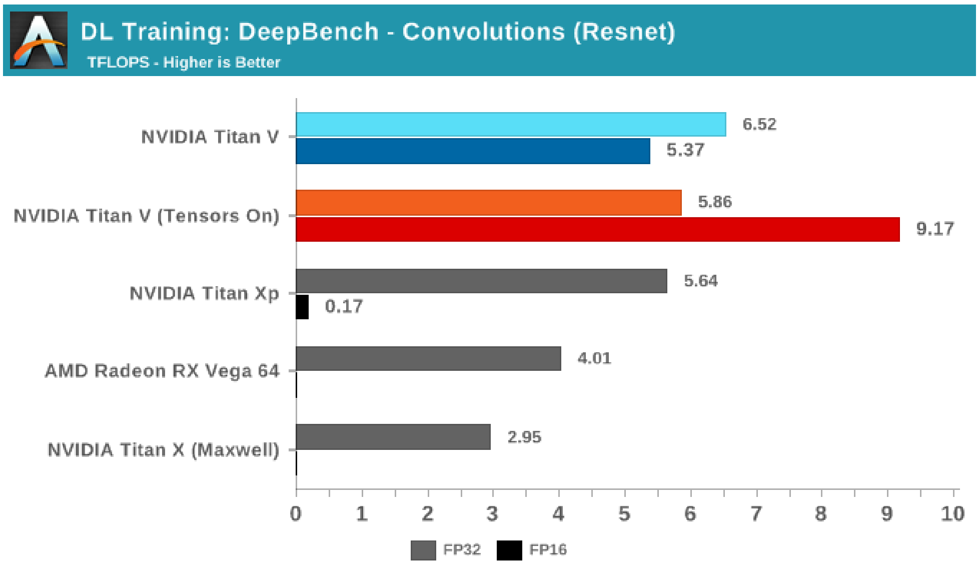
V only has theoretically around 100GB/s more, but at the same amount of VRAM; and while Volta's HBM controller efficiency has improved over Pascal P100, Titan Xp presumably features NVIDIA's 2nd generation GDDR5X controller.

DeepBench Training: Convolutions

Moving on to DeepBench's convolutions training workloads, we should see tensor cores significantly accelerate performance once again. Given that convolutional layers are essentially standard for image recognition and classification, convolutions are one of the biggest potential beneficiaries of tensor core acceleration.

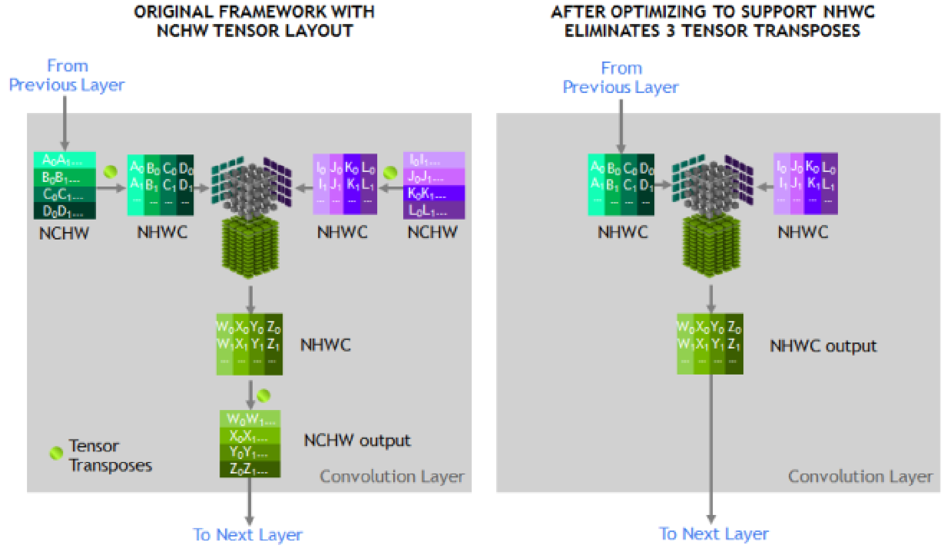
Taking the average of all tests, we again see Volta's mixed precision (FP16 with tensor cores enabled) taking the lead. Unlike with GEMM, enabling tensors on FP32 convolutions results in a tangible performance penalty.



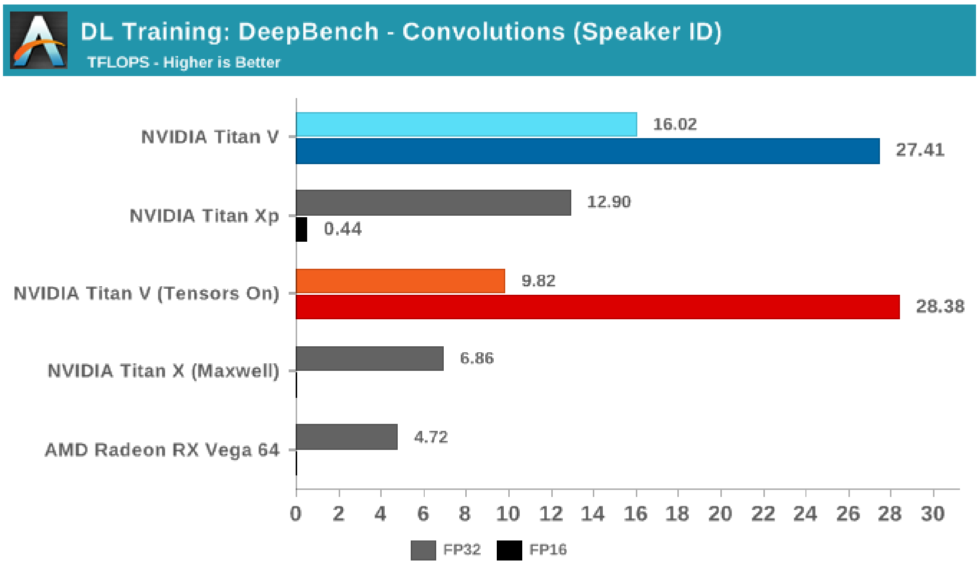


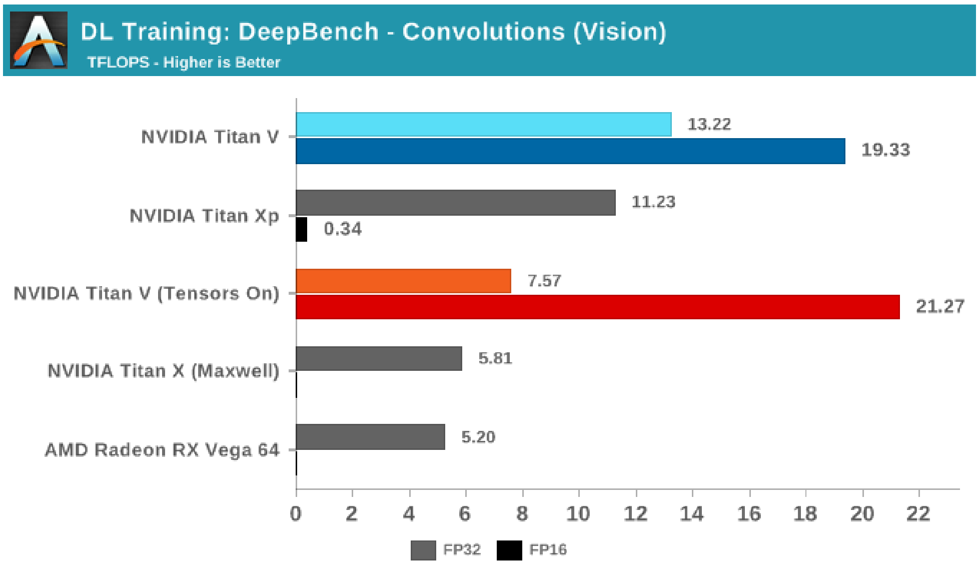
Breaking the tests out by application does not particularly clarify matters. It's only when we return to the DeepBench convolution kernels that we get a little more detail. Performance drops for both mixed precision modes when computations involve ill-matching tensor dimensions, and while standard precision modes follow a cuDNN-specified fastest forward algorithm, such as Winograd, the mixed precision modes are obliged to use implicit precomputed GEMM for all kernels.

To qualify for tensor core acceleration, both input and output channel dimensions must be a multiple of eight, and the input, filter, and output data-types must be half precision. Without going too deep into detail, the implementation of convolution acceleration with tensor cores requires tensors to be in a NHWC format (Number-Height-Width-Channel), but DeepBench, and most frameworks, expect NCHW formatted tensors. In this case, the input channels are not multiples of eight, but DeepBench does automatic padding to account for this.



The other factor is that all these NCHW kernels would require transposition to NHWC, which NVIDIA has noted takes up appreciable runtime once convolutions are accelerated. This would affect both FP32 and FP16 mixed precision modes.



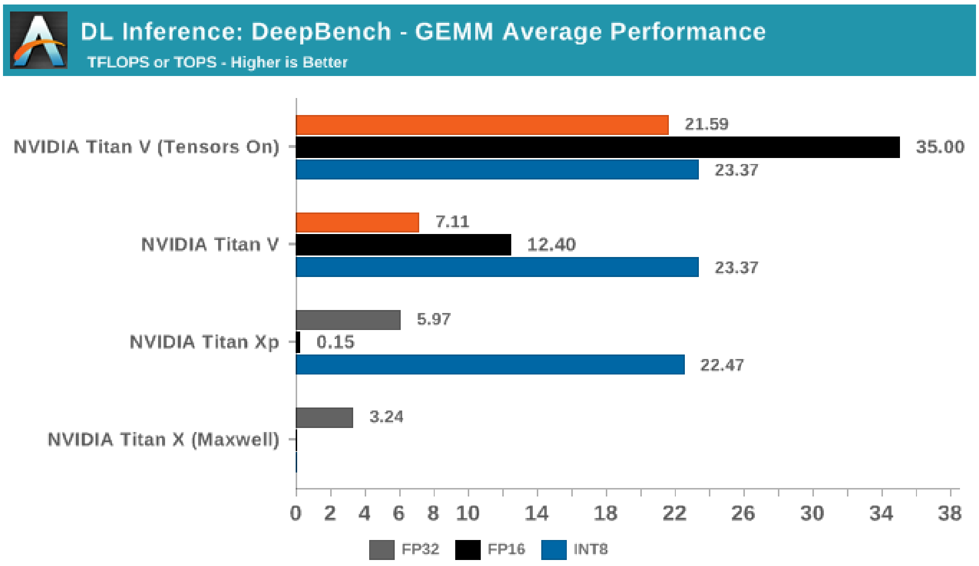


Convolutions still have to be adjusted correctly to benefit from tensor core acceleration. As DeepBench uses the NVIDIA-supplied libraries and makefiles, it's interesting that the standard behavior here would be to force tensor core use at all times.

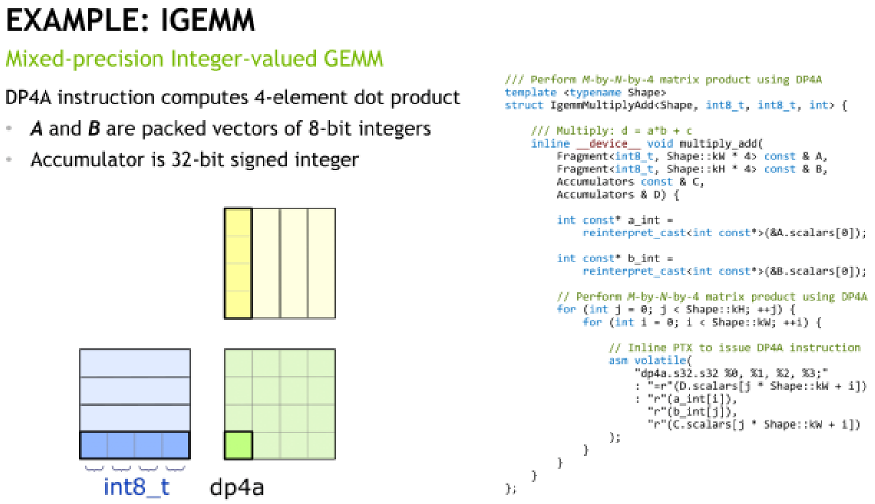
DeepBench Inference: GEMM

Shifting gears to inferencing, what DeepBench is simulating with graphics cards are more for their suitability in inference deployment servers, rather than edge devices. So of the inference kernels, none of the 'device' type ones are run.

Precision-wise, DeepBench inference tasks support INT8 on Volta and Pascal. More specifically, Baidu qualifies DeepBench GEMM and convolutions as INT8 mutiply with 32-bit accumulate.



Both Titan V and Titan Xp feature quad rate INT8 performance, and DeepBench's INT8 inferencing implementation neatly falls under the DP4A vector dot product capability introduced by Pascal. The instruction is still supported under Volta, and shows up once again as IDP and IDP4A in the instruction set.



EXAMPLE: IGEMM

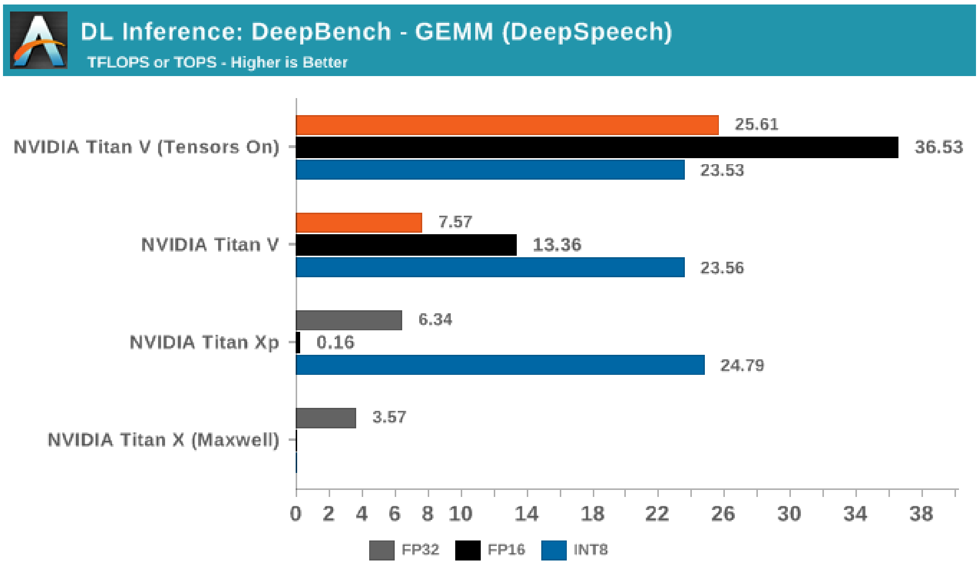
Mixed-precision Integer-valued GEMM

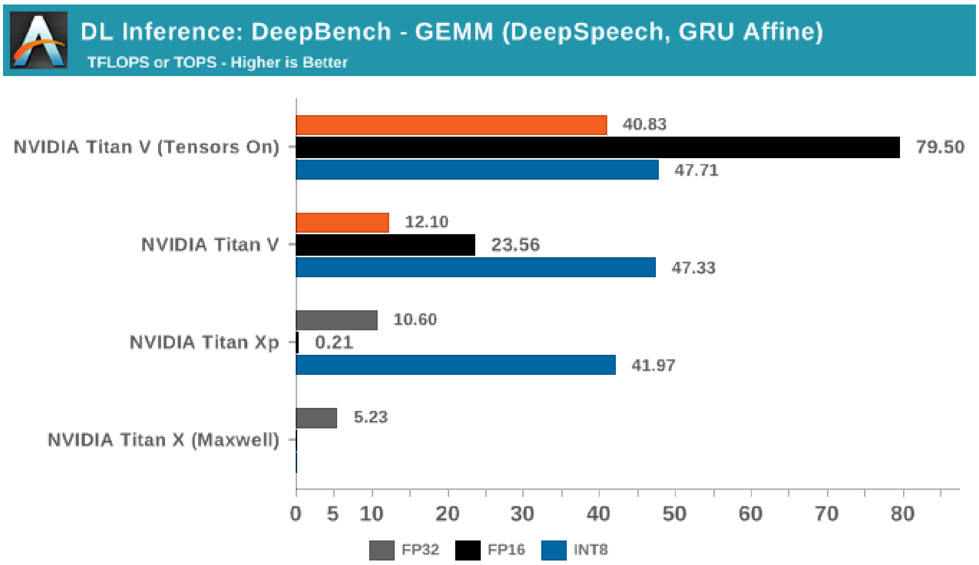
DP4A instruction computes 4-element dot product

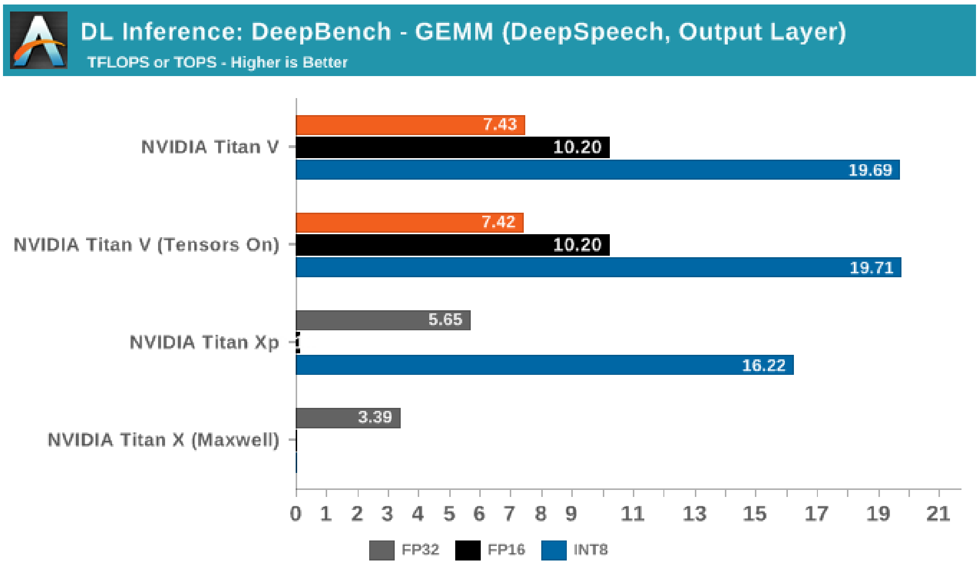
- *A* and *B* are packed vectors of 8-bit integers
- Accumulator is 32-bit signed integer

```
/// Perform M-by-N-by-4 matrix product using DP4A
template <typename Shape>
struct IgemmMultiplyAdd<Shape, int8_t, int8_t, int> {

    /// Multiply: d = a*b + c
    inline __device__ void multiply_add(
        Fragment<int8_t, Shape::kW * 4> const & A,
        Fragment<int8_t, Shape::kH * 4> const & B,
        Accumulators const & C,
        Accumulators & D) {

        int const* a_int =
            reinterpret_cast<int const*>(&A.scalars[0]);

        int const* b_int =
            reinterpret_cast<int const*>(&B.scalars[0]);

        // Perform M-by-N-by-4 matrix product using DP4A
        for (int j = 0; j < Shape::kH; ++j) {
            for (int i = 0; i < Shape::kW; ++i) {

                // Inline PTX to issue DP4A instruction
                asm volatile(
                    "dp4a.s32.s32 %0, %1, %2, %3;"
                    : "=r"(D.scalars[j * Shape::kW + i])
                    : "r"(a_int[i]),
                      "r"(b_int[j]),
                      "r"(C.scalars[j * Shape::kW + i])
                );
            }
        }
    }
};
```

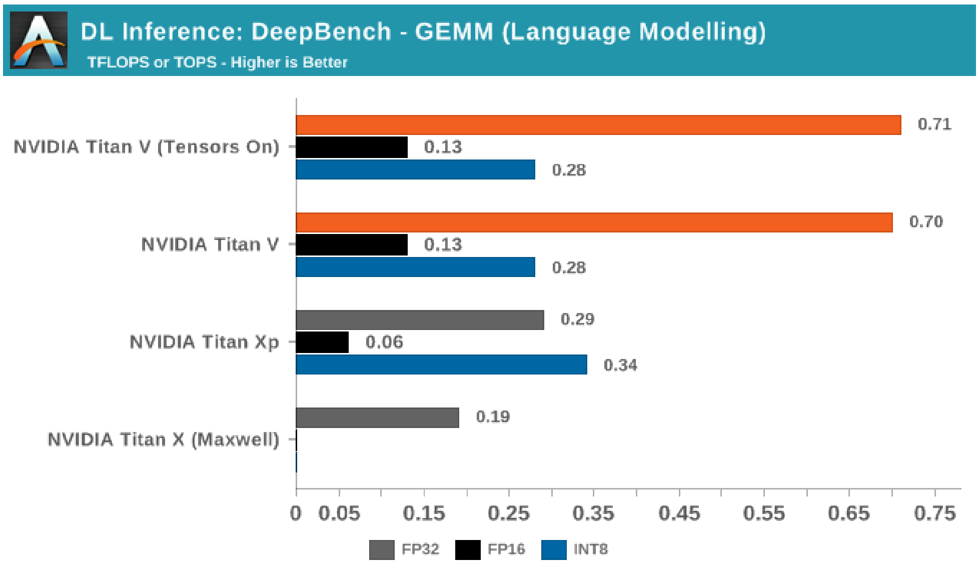
For IGEMM, as CUTLASS shows, DP4A is the bespoke operation. So we see 8-bit integer performance at a high level throughout, except in language modelling. Once again, the irregular dimensions don't lend itself to easy acceleration, if at all.

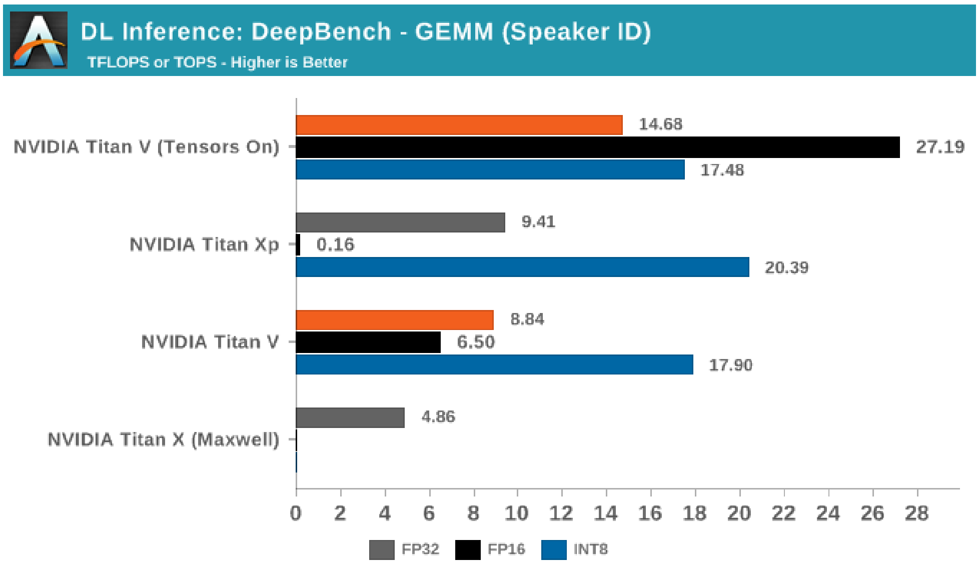






In fully-connected (or 'affine') layers, every node in that layer is connected to every node in the previous layer. What this means for a typical CNN is that the fully-connected layer is able combines all the extracted features to make a final prediction and classify the image. By the numbers, this also means large and regularly-proportioned matrices, which can equal large speedups as we see here.

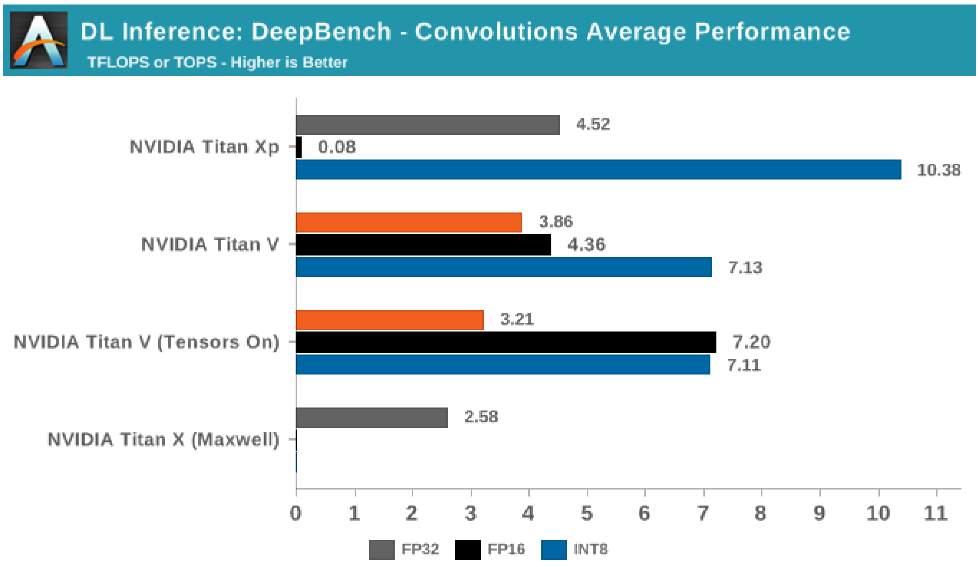




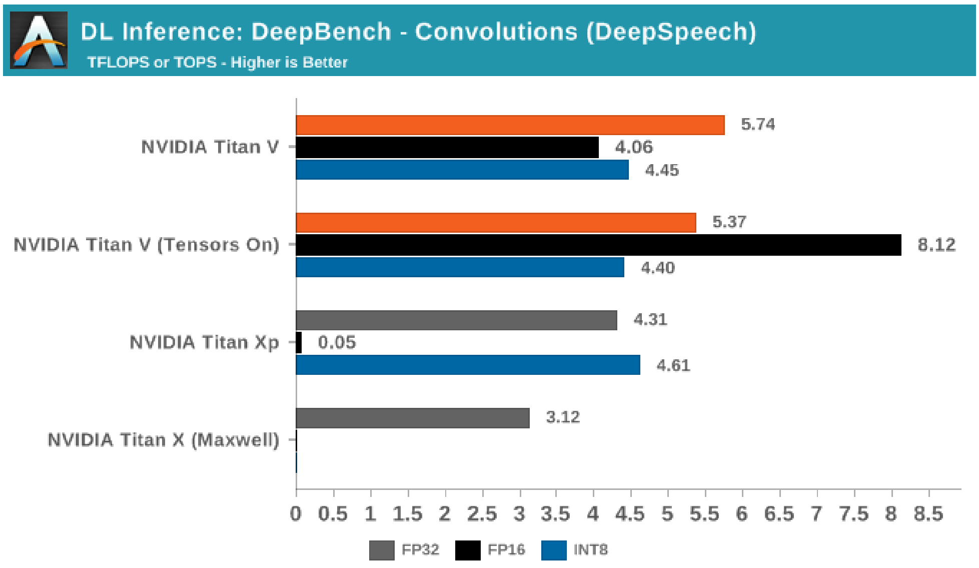
DeepBench Inference: Convolutions

Moving on to convolutions, 8-bit multiply/32-bit accumulate again makes an appearance with INT8 inferencing.

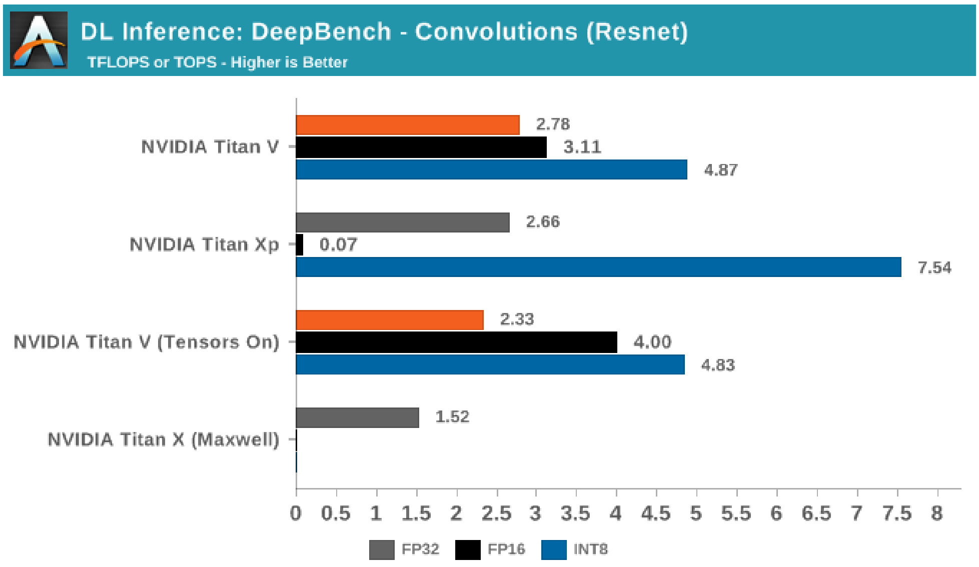
The most striking aspect in the average convolutions performance is Titan Xp's superior INT8 throughput. The numbers, being comparable to the DeepBench Titan Xp inference results, are correct. Nor is padding responsible for the disparity.



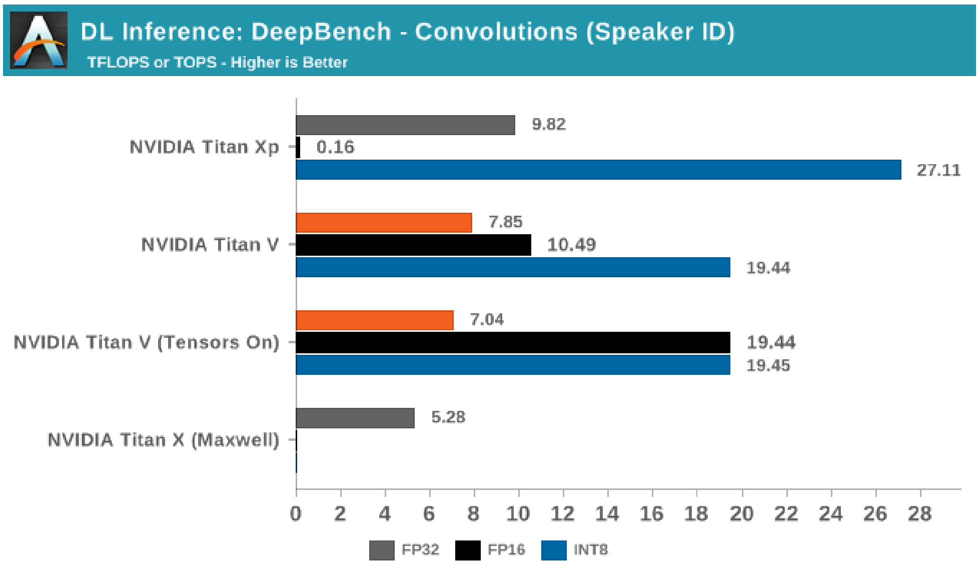
Breaking out the convolutions into application-specific workloads, we see that Resnet, Speaker ID, and Vision showcase Titan Xp's superior INT8 performance.


DL Inference: DeepBench - Convolutions (DeepSpeech)

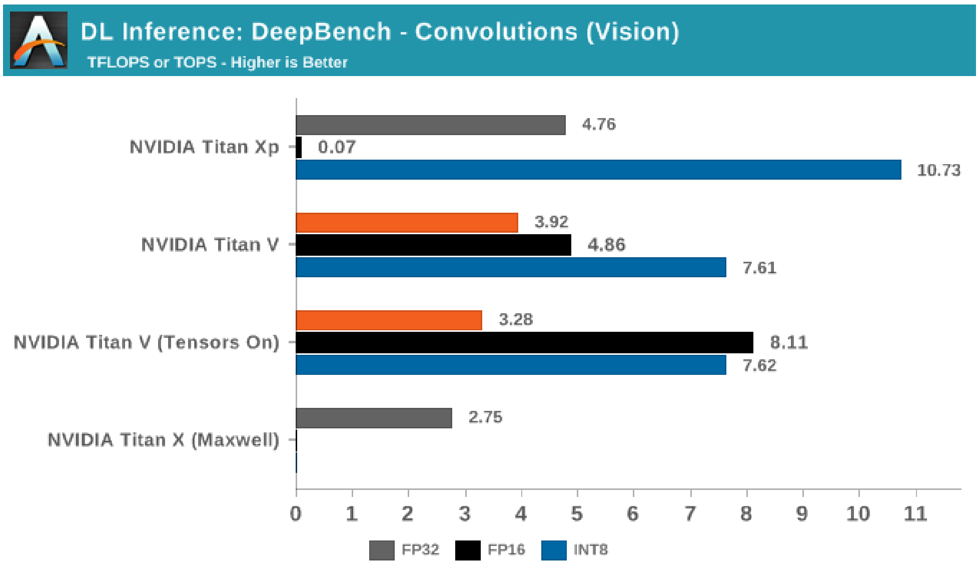
TFLOPS or TOPS - Higher is Better


DL Inference: DeepBench - Convolutions (Resnet)

TFLOPS or TOPS - Higher is Better


DL Inference: DeepBench - Convolutions (Speaker ID)

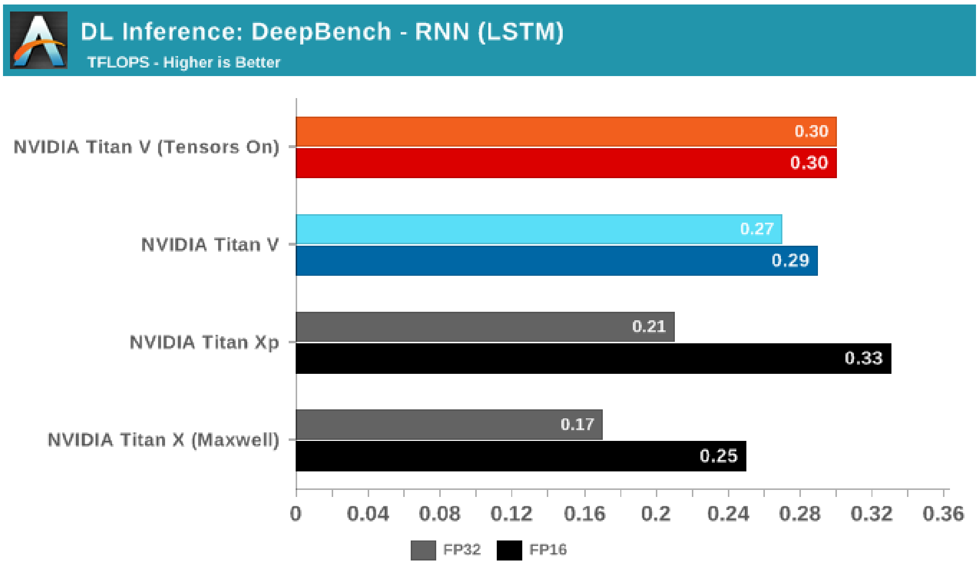
TFLOPS or TOPS - Higher is Better

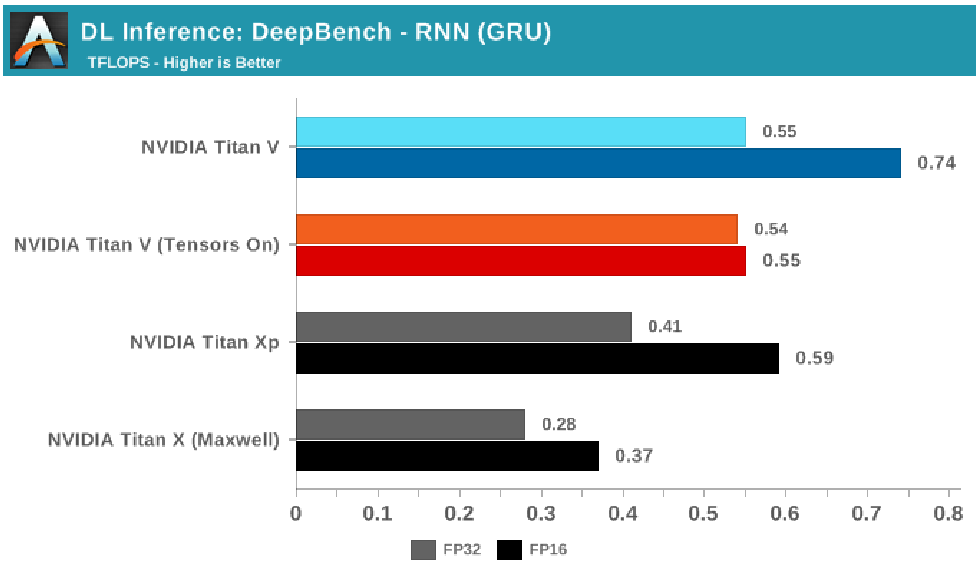


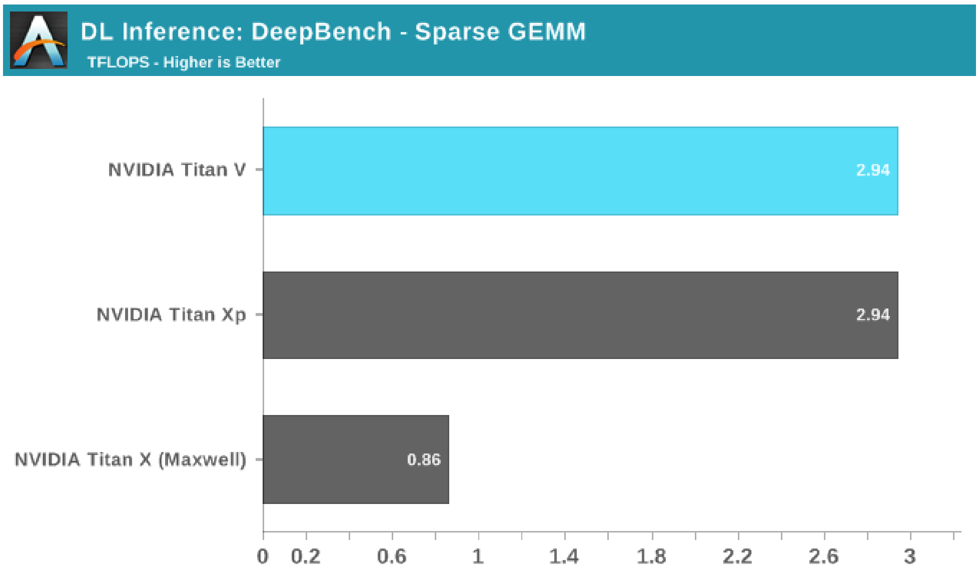
Nothing seems obvious from the kernels, but if anything, this is likely due to DP4A library/driver maturity on Pascal, compared to it's Volta implementation. There's also the chance that Volta is handling it solely through its INT cores.

DeepBench Inference: RNN and Sparse GEMM

Rounding out the last of our DeepBench inference tests are RNN and Sparse GEMM, both available in single precision only. That being said, the FP16 parameter could be selected anyway. Given the low results all around, this is more of an artifact than anything else.





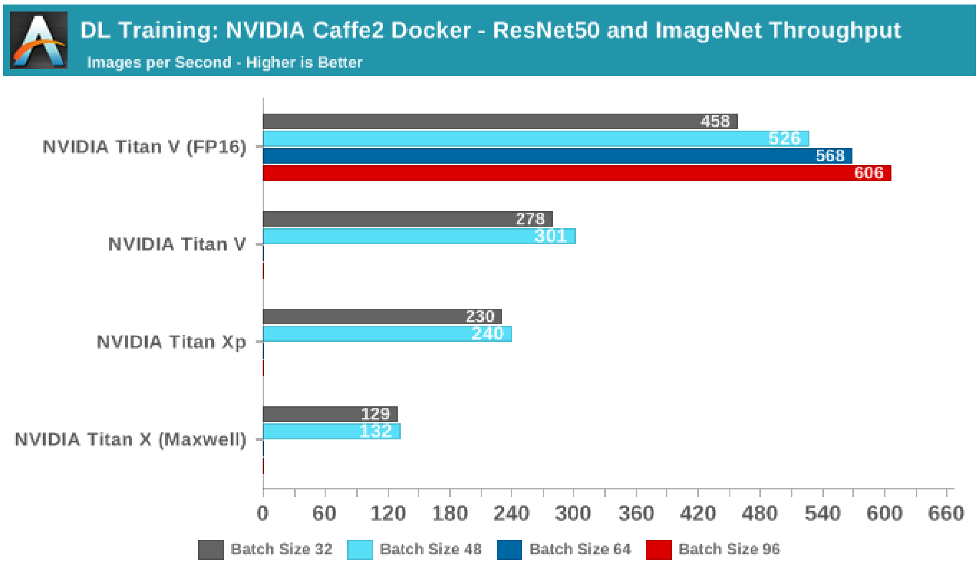


While RNNs might also be accelerated, DeepBench and NVIDIA only support single precision RNN inference at this time.

NVIDIA Caffe2 Docker: ResNet50 and ImageNet

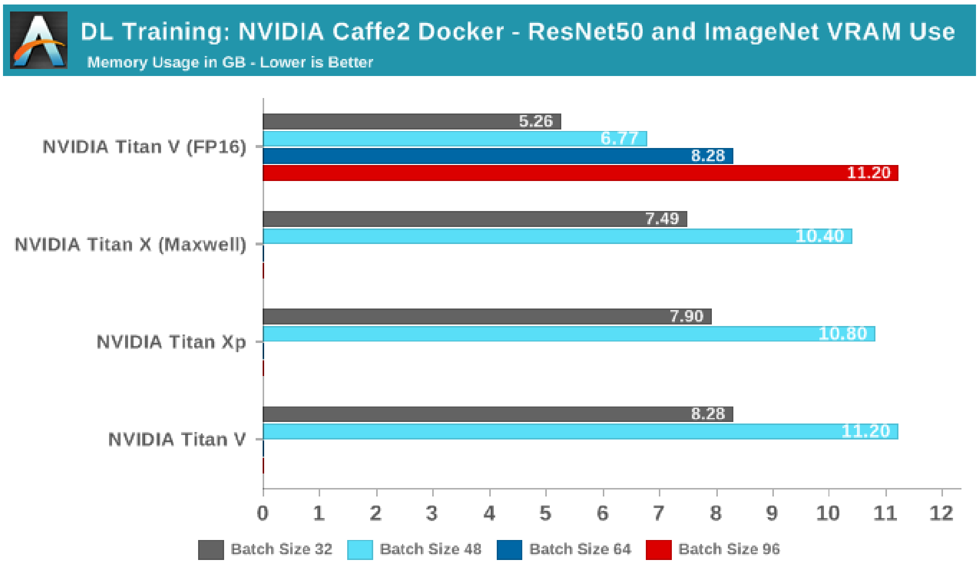
Kernels and deep learning math operations may be useful, but in the end devices are trained with real datasets. Using the standard ILSVRC 2012 pictureset, we run the standard ResNet-50 training implementation that is included in NVIDIA's Caffe2 Docker image. The model trains on ImageNet and gives us some throughput data.

While there were separate switches for FP16 and tensor cores, running FP16 mode with tensors enabled and disabled resulted in identical results for the Titan V.



No score indicates card ran out of video memory

In terms of pure throughput, the Titan V takes the lead at all batch sizes. In fact, with tensors enabled it is able to go beyond 64 batches, as opposed to the other cards, even though they all have 12 GBs of VRAM. The reasoning is that FP16 consumes less video memory.


DL Training: NVIDIA Caffe2 Docker - ResNet50 and ImageNet VRAM Use

Memory Usage in GB - Lower is Better

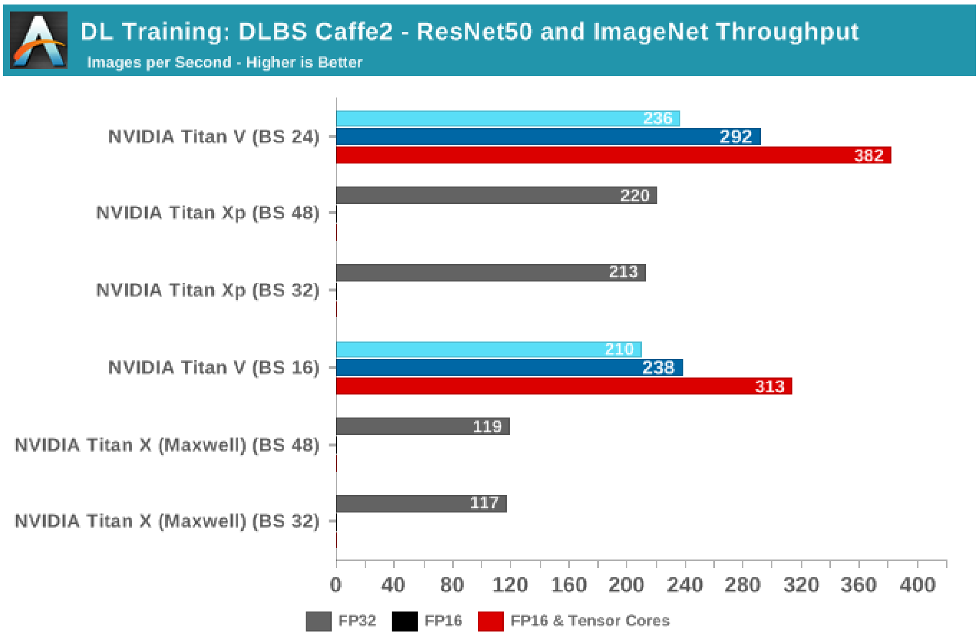
The issue with raw throughput metrics is that real-world performance for deep learning is never so simple. For one, many models might be optimized for throughput but sacrifice accuracy and/or training time. Peak or even sustained images trained per second may not be useful if the model takes an extended amount of time to converge. This is particularly relevant for Volta with FP16 storage and tensor cores, as there may be a number of necessary mitigations like loss scaling or single precision batch normalization, which wouldn't be directly accounted for in throughput metrics.

That being said, finding modern benchmarks that are Volta-aware, reasonably close to state-of-the-art, provide better metrics, go beyond CNNs for computer vision, and are accessible by non-researchers, has been a struggle. Throughput benchmarks are easier to validate and create, but in many situations they are better suited for identifying bottlenecks, platform differences, and optimization points.

HPE DLBS, Caffe2: ResNet50 and ImageNet

On that relevant note, we'll take a look at HPE Deep Learning Benchmark Suite, part of their Deep Learning Cookbook. With a different angle than usual DL benchmark suites, its wide-ranging test-running modularity lends itself to quickly diagnose issues or bottlenecks on hardware platforms, which would be useful for an organization like HPE. Focused mostly on NVIDIA GPUs running computer vision CNNs, the DLBS setup is Volta-aware and is essentially outputs throughput and time metrics only, with some advanced monitoring and visualization tools that are only somewhat user-friendly.

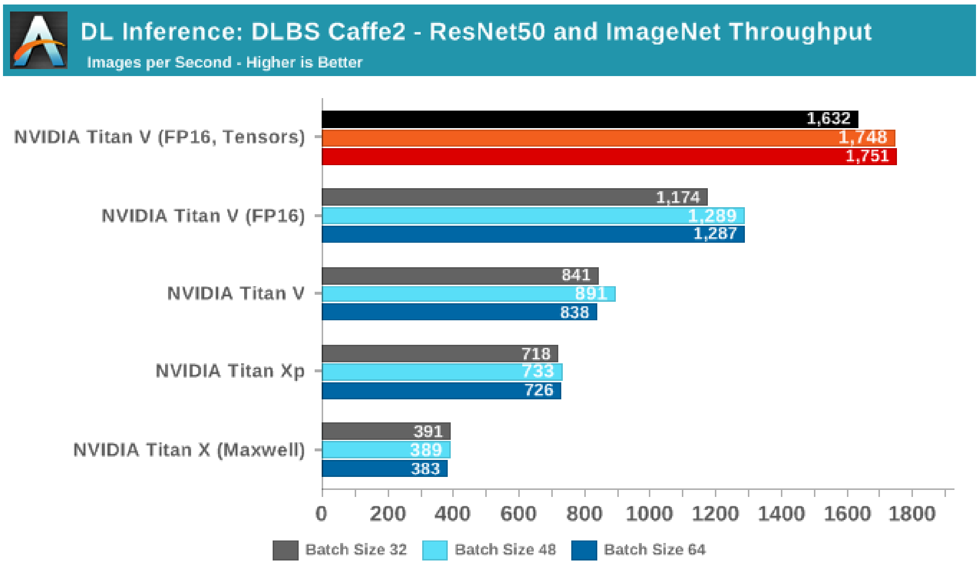
For our purposes, it allows us to corroborate the NVIDIA Caffe2 Docker benchmark, so we train and inference with a ResNet50 model on ImageNet. But as the models and implementations are different, these throughput numbers shouldn't be directly compared to NVIDIA's Caffe2 Docker test.


DL Training: DLBS Caffe2 - ResNet50 and ImageNet Throughput

Images per Second - Higher is Better

During the training benchmarking, certain batch sizes for Titan V refused to run, but generally, we see the same trend as before, with FP16 and tensor cores providing higher throughput.

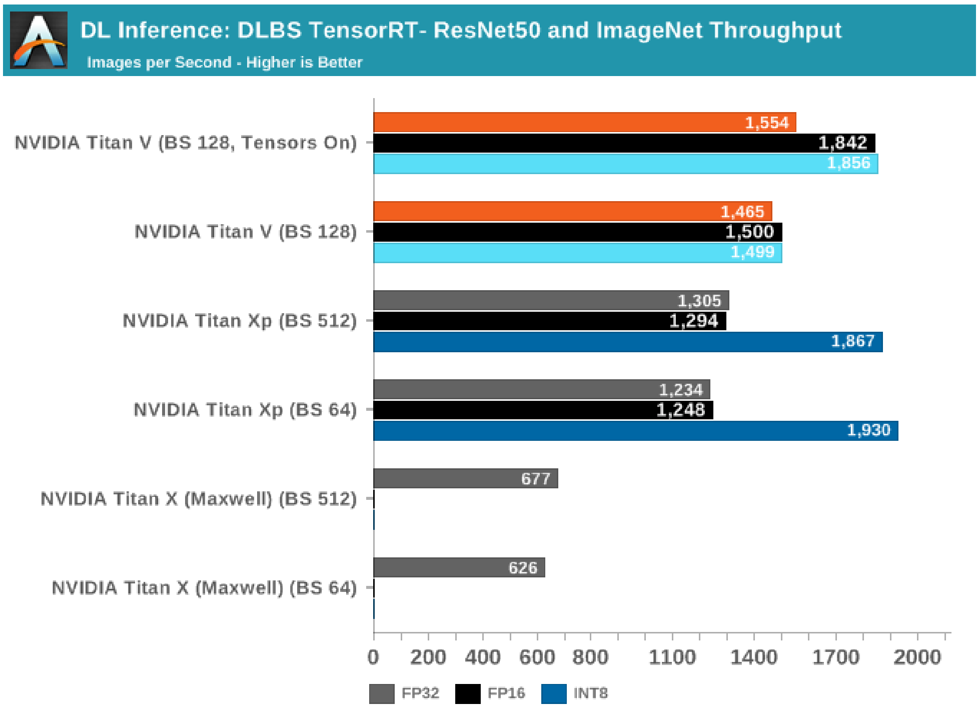
We see the same trends continue with inferencing. Unfortunately, the DLBS Caffe2 test does not seem to support INT8 inferencing.



HPE DLBS TensorRT: ResNet50 and ImageNet

The other unique aspect of HPE DLBS is the feature of a benchmark for TensorRT, NVIDIA's inference optimizing engine. In recent years, NVIDIA has pushed to integrate it with new DL features like INT8/DP4A and tensor core 16-bit accumulator mode for inferencing.

Using a Caffe model, TensorRT adjusts the model as needed for inferencing at a given precision.



In total, we ran batch sizes 64, 512, and 1024 for Titan X (Maxwell) and Titan Xp, and batch sizes 128, 256, and 640 for Titan V; the results were within 1 - 5% of the other batch sizes, so we've not included them in the graph.

The high INT8 performance of Titan Xp somewhat corroborates with the GEMM/convolution performance; both workloads seem to be utilizing DP4A. Meanwhile, it's not clear how Titan V implements DP4A. all we know is that it is supported by the Volta instruction set. And Volta does has those separate INT32 units.

DAWNBench: Image Classification (CIFAR10)

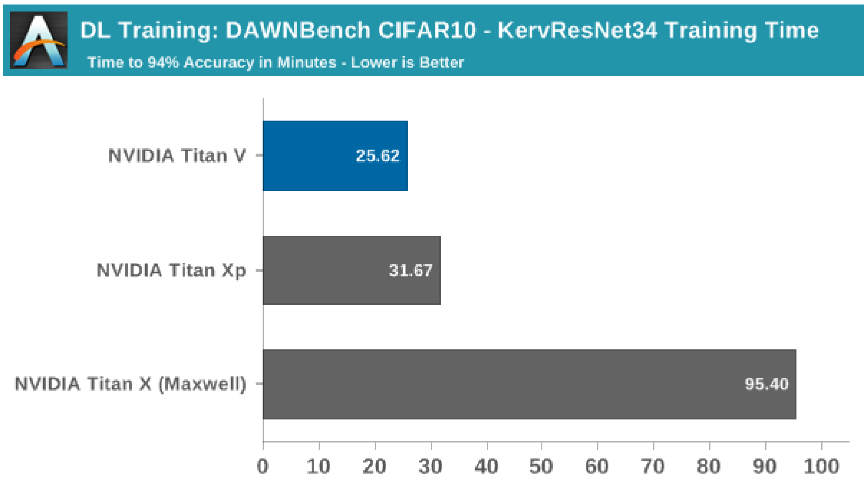
In terms of real-world applicable performance, deep learning training is better described with time-to-accuracy and cost metrics. For DAWNBench, those measurements were the stipulations for each of the three subtests. For image classification with CIFAR10, these were:

- ***Training Time:*** *Train an image classification model for the CIFAR10 dataset. Report the time needed to train a model with test set accuracy of at least 94%*
- ***Cost:*** *On public cloud infrastructure, compute the total time needed to reach a test set accuracy of 94% or greater, as outlined above. Multiply the time taken (in hours) by the cost of the instance per hour, to obtain the total cost of training the model*

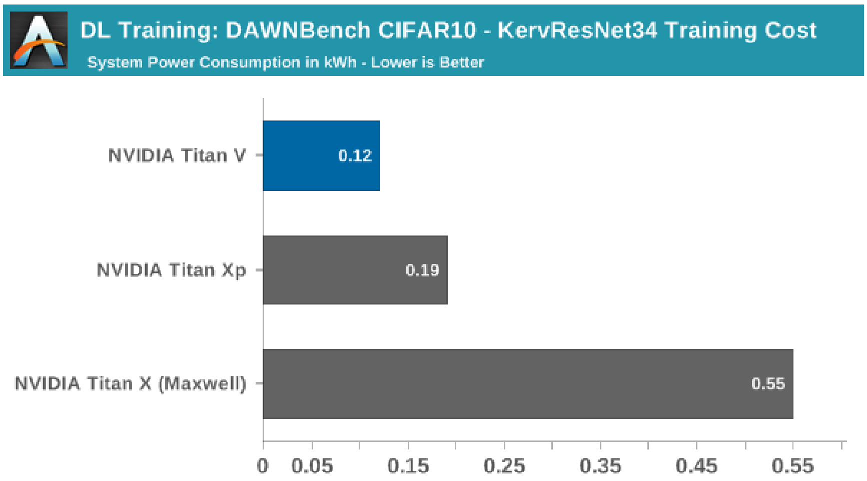
Here, we take two of the top-5 fastest CIFAR10 training implementations, both in PyTorch, and run them with our devices. The first, based on ResNet34, was created to run on a NVIDIA GeForce GTX 1080 Ti, while the second, based on ResNet18, was created to run on a single Tesla V100 (AWS p3.2large). Because these are recent top entries to DAWNBench, we can consider them to be reasonably modern, while understanding that CIFAR10 is not a hugely intensive dataset.

For our purposes, the tiny image dataset of CIFAR10 works fine as running a single-node on a dataset like ImageNet with non-professional hardware that could be old as Kepler, may result in unacceptably long training times for unlikely convergence. This is a useful result on its own but only after we have a fully detailed machine learning GPU benchmark suite.

The first implementation was designed to run on a single GTX 1080 Ti train, and the original submission noted that it took 35:37, or 35.6 minutes, to train to 94%.

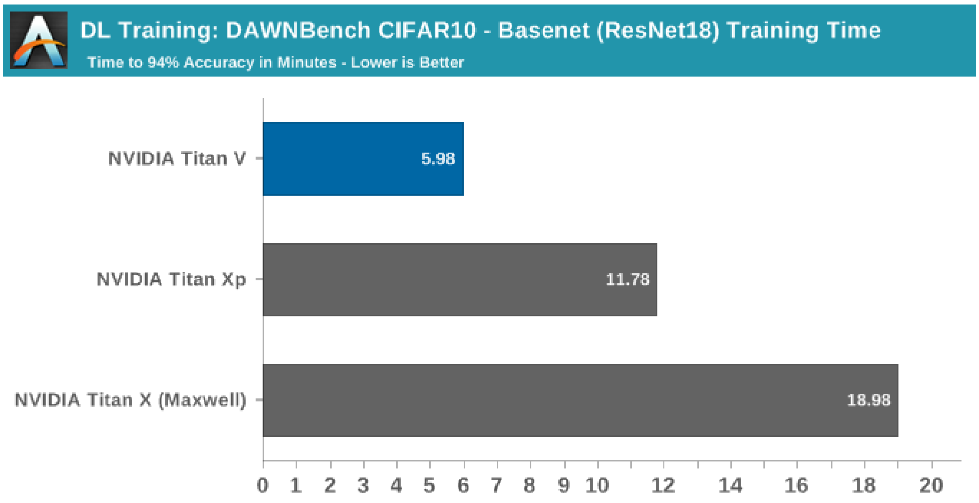


Even though we are training this on-premise, the cost metric is useful to compare against other graphics cards, though in this case we are talking about electricity differences of a few cents.

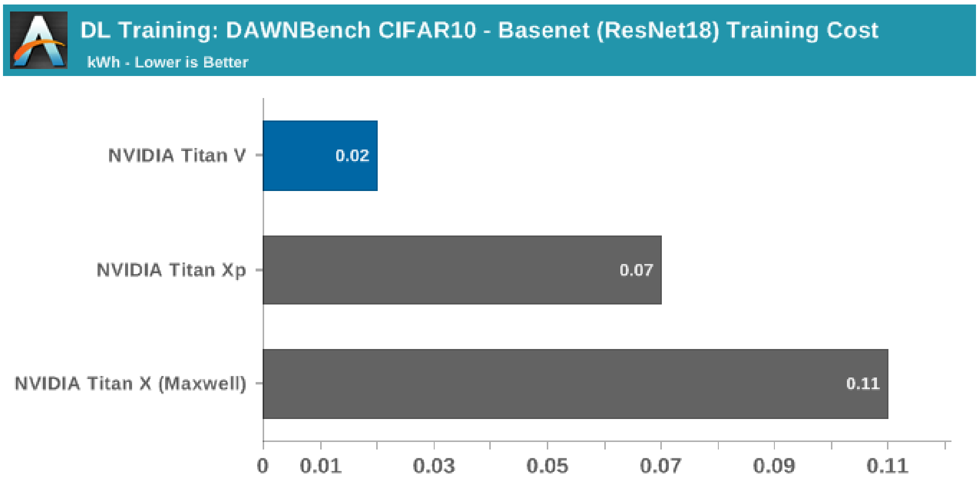


Using the Titan V in this circumstance does not take advantage of tensor cores, only its general improvements over Pascal. In this unoptimized setting, it runs in around 20% faster than the Titan Xp. Even still, the peak system consumption has dropped down by around 80W, though how much of that is due to the graphics card is unclear because of how the model deals with CPU usage and data pre-processing.

For the second implementation, the original submission reported the V100 training to 94% in 5:41, or 5.7 minutes.



Without delving into and accurately analyzing the model code, it's not clear whether it takes advantage of tensor cores or not, since the model is plenty fast on Maxwell, let alone Volta.



Final Words

As we bring this deep dive to close, I should state that DL performance benchmarking on GPUs is very much an ongoing project. For one, everything we've discussed today has been limited to image classification and CNNs, which is very standard as far as GPU deep learning goes, and very standard in its insight. It also plays to some of the traditional DL strengths of GPUs – and of tensor cores and library support for that matter – and in the end, the only featured end-to-end tests were the CIFAR10 implementations.

What we didn't cover was the plethora of failed benchmark runs from various other tests and suites. It proved impossible to find, in piecemeal or in whole, an up-to-date DL suite that ran modern workloads, provided end-to-end metrics, covered multiple ML domains, included tensor core and mixed precision support, and could be reasonably accessible and used by a non-developer.

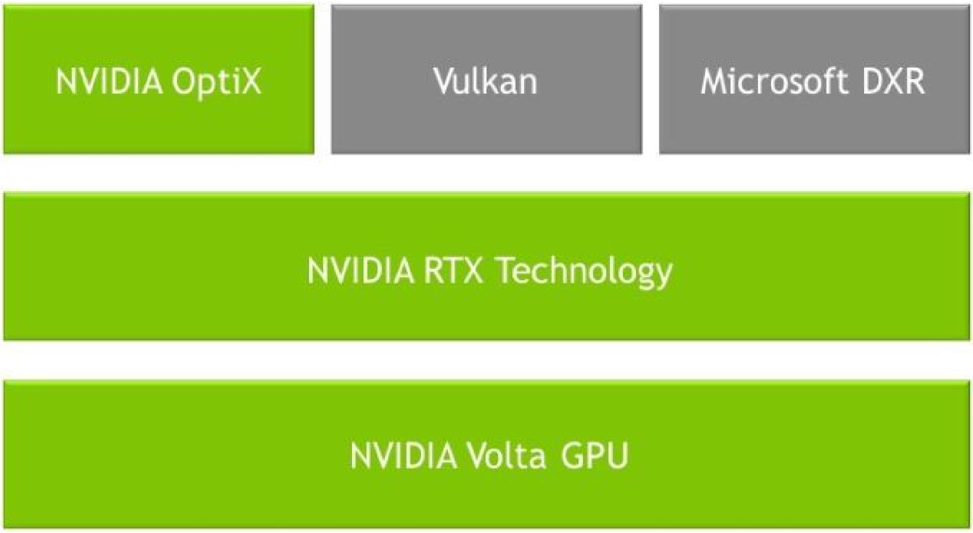
Because tensor cores are such an important part of the Titan V, it is only natural that we utilize them in our tests. And as AnandTech is not a ML research organization, developing our own robust benchmarkable model in a given framework was not an option. Similarly, we are limited in directly modifying tests and models ourselves. Reworking scripts is one thing, but once you have to go peeling through random Python code of both the models and various supporting functions for whatever framework, the task becomes a little out of scope.

Meanwhile, DAWNBench and competition-type collections were ultimately designed for researchers and developers to create their own implementations, rather than providing a generalizable benchmark. Reference implementations as maintained by the DL frameworks still would need modification to show that it could be a valid benchmark. And if it isn't obvious already, re-configuring a model for Volta-compatible mixed precision is not something that can (usually) be done on the fly.

As a whole, this actually bears relevance to the Titan V. Tensor cores and mixed precision require not just explicit support: they need development. And while the performance boost is clearly evident in particular cases, it's not there in all cases. Even when you have code and datasets that mesh well with the tensor core execution model, at the end of the day neural network processing isn't bottlenecked solely by ALUs. So while the Titan V is capable of being much more powerful than past NVIDIA GPU designs – sometimes massively so – the kinds of performance gains that NVIDIA likes to promote are understandably some of the best-case results.

What this means for anyone thinking about purchasing the Titan V for compute needs is that investing a Titan V means investing in mixed precision DL models and/or WMMA-based GEMM acceleration for HPC (with the assistance of CUTLASS and such). Titan cards have always been prosumer, but the Titan V really makes its mark as a **professional** consumer card.

Meanwhile in the context of cuDNN/cuBLAS, as well as early mixed precision with DP4A and FP16x2, tensor cores are a natural evolution in attempting to hit a bullseye with programmable hardware for DL acceleration. Of which release notes of recent cuDNN versions list various issues with tensor core math and operations, so bugfixing is very much an ongoing process. Yet tensor cores are even more limited than 'deep learning matrix-matrix multiplications,' except all the details and implementations are not relevant to end-users.



NVIDIA OptiX

Vulkan

Microsoft DXR

NVIDIA RTX Technology

NVIDIA Volta GPU

But what about mainstream consumers? What does all of this tensor core development mean for them? Despite the focus here on neural networking, the tensor cores are expected to have some relevance and use in the consumer graphics space. We know that NVIDIA's recent-ish RTX ray tracing technology uses tensor cores to apply denoising on images to make up for limited number of rays. But at the same time, that type of a use case for tensor cores implicates a very different product than the Titan V. RTX hardware functionality will be opaque to end-users, much like a supposed near-future where FP16x2/Rapid Packed Math would be used in video games. And development of tensor core utilization needs to reach that point.

One other consumer-focused role that NVIDIA will be considering for any kind of mainstream parts is whether they want to include tensor cores for development debugging purposes. This is a similar route that NVIDIA already takes today with FP64 functionality, including a handful of FP64 CUDA cores in each GPU so that developers can run FP64 code natively, just very slowly. Right now the Titan V is the only game in town if you need to debug your tensor code on real hardware, but that can certainly (and arguably is likely to) change once mainstream GPUs get a chance to play catch-up.

Meanwhile, NVIDIA's traditional hardware development cadence and redacted Hot Chips talks point to a new consumer lineup in the near future, and it will be very interesting to see what architectural features turn up. What we can say is that the Titan V definitely has the marks of NVIDIA's future GPGPU aspirations. And the Titan V is not merely a deep learning accelerator, because the tensor cores are in essence not merely deep learning accelerators. Moving forward, we're hoping that MLPerf and similar efforts make good headway, so that we can tease out a bit more secrets from GPUs.