



# Benchmarking the NVIDIA V100 GPU and Tensor Cores

Matt Martineau<sup>(✉)</sup>, Patrick Atkinson, and Simon McIntosh-Smith

HPC Group, University of Bristol, Bristol, UK  
{m.martineau,p.atkinson,cssnmis}@bristol.ac.uk

**Abstract.** The V100 GPU is the newest server-grade GPU produced by NVIDIA and introduces a number of new hardware and API features. This paper details the results of benchmarking the V100 GPU and demonstrates that it is a significant generational improvement, increasing memory bandwidth, cache bandwidth, and reducing latency. A major new addition is the Tensor core units, which have been marketed as deep learning acceleration features that enable the computation of a  $4 \times 4 \times 4$  half precision matrix-multiply-accumulate operation in a single clock cycle. This paper confirms that the Tensor cores offer considerable performance gains for half precision general matrix multiplication; however, programming them requires fine control of the memory hierarchy that is typically unnecessary for other applications.

## 1 Introduction

To fit within the anticipated power budgets for future supercomputing architectures, it is possible that clusters targeting exascale and beyond will be comprised of diverse heterogeneous architectures, including both CPUs and accelerator devices such as GPUs. Server-grade processors are constantly evolving in terms of core counts, vector widths, and memory architectures, in response to the needs of modern applications. Given the increasing complexity of heterogeneous devices, it is becoming more difficult to develop and optimise scientific applications that can exploit available supercomputing resources. The core aim of this paper is to uncover key architectural changes of the NVIDIA V100 GPU compared to its predecessors, and discuss the implications on performance.

Renewed investment in the machine learning space means that many areas of architecture design are focusing on the technological improvements that can also benefit low precision, approximate computation. A recent example of technological innovation targeting machine learning is the inclusion of Tensor cores in the new NVIDIA Volta V100 GPUs, a principal focus of this research. Two of the largest supercomputers in the world, Sierra and Summit [1], use dual-socketed POWER9 CPUs and NVIDIA Volta GPUs, supporting a peak performance of 72 and 122 PetaFLOP/s respectively. Through the use of micro-benchmarks and applications this paper will demonstrate that the V100 GPUs are a significant improvement over previous generations, offering impressive performance for current scientific workloads.

## 2 Contributions

The following contributions are presented in the research:

- Benchmarking and analysis of many characteristics of the V100 GPUs compared to the previous generation of server-grade GPUs (Table 1).
- Analysis and evaluation of the Tensor cores, through the optimisation of a general matrix multiplication benchmark.
- A discussion regarding the applicability of Tensor cores to HPC.

## 3 Background

NVIDIA GPUs are throughput computing devices that support execution of thousands of parallel threads on lightweight processing elements called Streaming Multiprocessors (SMs). The GPU schedules multiple 32-wide units of execution, named warps, on the SMs.

**Table 1.** Hardware details of NVIDIA Tesla P100 and V100 devices

Device	Tesla P100	Tesla V100
SMs	56	80
FP32 cores/SM	64	64
FP64 cores/SM	32	32
Tensor cores/SM	-	8
GPU clock	1.189 GHz	1.245 GHz
Shared memory/SM	64 KB	96 KB
L2 cache size	4096 KB	6144 KB

The NVIDIA P100 GPU, presented in Fig. 1a, is the most powerful GPU in widespread use at the time of writing. The GPU introduces hardware double precision atomic instructions, and high bandwidth memory quoted to offer a theoretical 732 GB/s at peak. The NVIDIA V100 GPU (Fig. 1b) is the newest hardware from NVIDIA and introduces a number of features including:

- **Tensor cores** - Deep-learning focused cores that perform fast matrix-multiply-accumulate (MMA) operations.
- **Individual program counters per thread** - Numerous changes to synchronisation have been enabled as a consequence, including intra and inter GPU synchronisation that was not previously possible.
- **Increased memory bandwidth** - The high bandwidth memory (HBM2) has been optimised for a higher theoretical peak memory bandwidth.

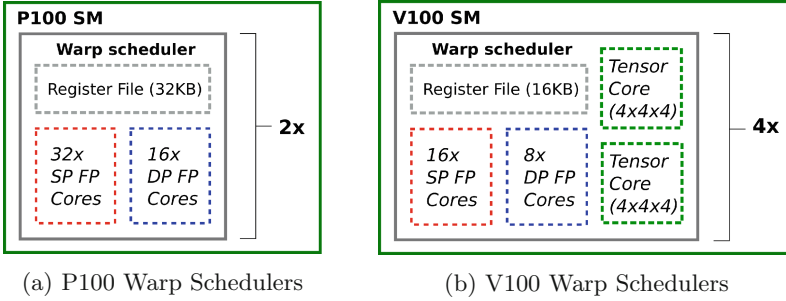


Fig. 1. Balance of warp schedulers within each SM of P100 and V100 GPUs.

### 3.1 NVIDIA Volta GPU Tensor Cores

The available literature is not clear on the exact mechanisms that are used to fulfil a  $16 \times 16 \times 16$  matrix multiplication using Tensor cores. In an NVIDIA V100 GPU, the SMs contain 8 Tensor cores, with each supporting a  $4 \times 4 \times 4$  matrix-multiply-accumulate ( $A * B + C = D$ , or  $A * B + C = C$ ) operation per clock cycle.

The matrix multiplication step is performed in half precision, while the accumulation can be performed in either half or single precision. Although each of the SMs possesses 8 of the  $4 \times 4 \times 4$  Tensor core units, the PTX ISA currently only supports warp-level operations of size  $16 \times 16 \times 16$ ,  $32 \times 8 \times 16$ , and  $8 \times 32 \times 16$  [10], exposed through the Warp Matrix-Multiply-Accumulate (WMMA) interface. This reduces the number of use cases for Tensor cores, as access to the individual  $4 \times 4 \times 4$  units could allow for thread-level optimisation of routines, with the small blocking factor being far more useful to the general case.

## 4 Benchmarking

In this section we will benchmark the P100 and V100 GPUs to compare for generational improvements. All benchmarks are compiled with CUDA 9.0.

### 4.1 FLOP/s

Considering the NVIDIA V100 GPU with core clock speeds of 1.245 GHz, we can calculate the maximum FLOP/s achievable through single precision Fused-Multiply-Adds (FMAs):  $2 \text{ FLOPs} \times 64 \text{ threads} \times 80 \text{ SMs} \times 1.245 \text{ GHz} = 12.7 \text{ TFLOP/s}$ . Further to this, each of the 8 Tensor cores processes a  $4 \times 4 \times 4$  MMA operation in a single cycle, performing 64 FMAs:  $8 \text{ Tensor Cores} \times 128 \text{ FLOPs} \times 80 \text{ SMs} \times 1.245 \text{ GHz} = 102 \text{ TFLOP/s}$  for a mixed half and single precision operation. We observed a maximum of 25 TFLOP/s in half precision with FMAs, and 99 TFLOP/s using the MMA instructions.

### 4.2 Memory Bandwidth

The peak global memory bandwidth has been increased between the P100 and V100 by 1.23x; further, the achievable proportion of peak increases from roughly 80% on the P100 to around 93% on the V100. This is an important optimisation for many scientific workloads [7]. The number of warps in a thread-block required to saturate memory bandwidth has increased on the V100 to 4 warps from 3 warps on the P100 in single precision, or 8 warps from 5 warps in half precision (Fig. 2).

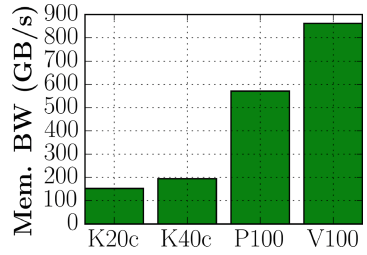


Fig. 2. NVIDIA GPU memory bandwidth.

### 4.3 Cache Bandwidth

The P100 GPU showed significant improvements to cache bandwidth over its predecessors [12] and through benchmarking we observed that the V100 continues this trend of increased memory bandwidth at the L1 cache level. To measure and compare the cache memory bandwidths between the P100 and the V100 we used the method outlined in [3], that is, we ran the STREAM Triad benchmark [9] multiple times over the same data-set to ensure cache-residency. The benchmark increasingly allocates more memory per CPU core or CUDA thread-block, which shows the bandwidth of each level of cache as the array saturates the available capacity, eventually spilling accesses into main memory.

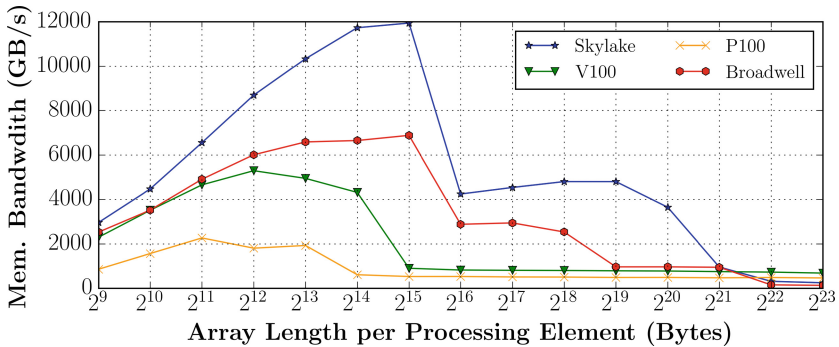


Fig. 3. Aggregate memory bandwidth as data per processing element doubles.

Figure 3 shows the aggregate bandwidth for several modern HPC processors. The L1 cache performance of the V100 GPU is 2.57x higher than the L1 cache performance of the P100, partly due to the increased number of SMs in the V100 increasing the aggregate result. However, when observing the memory bandwidth per SM, rather than the aggregate, the performance increase is 1.86x,

suggesting there has been significant improvements made to the L1 cache. This performance increase has come from a change in hardware design as the Volta architecture features a unified shared memory/L1/texture cache, whereas the Pascal architecture has separate L1/texture and shared memory caches.

Figure 3 also compares the cache bandwidth to CPU architectures, 56 core Skylake, and 44 core Broadwell. The Xeon Skylake architecture outperforms the previous generation of CPUs (Broadwell) by roughly 1.7x, mainly due to the increase in vector width from 256-bit to 512-bit. The GPUs lag behind the CPU architectures in terms of unmanaged L1 cache performance; however, the shared-memory performance of the V100 and P100 architectures were observed to be 11.0 TB/s and 6.3 TB/s respectively, hence are comparable to the cache bandwidth of current CPU architectures.

### 4.4 Latency

Figure 4 shows the memory access latency of both the P100 and V100 GPUs alongside the performance of the Skylake CPU.

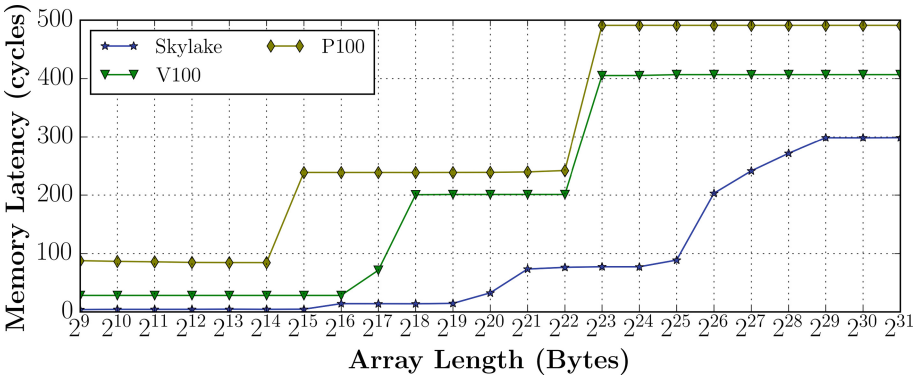


Fig. 4. Latency of memory accesses as array size is doubled.

For all levels of the memory hierarchy lower per-cycle latency is observed for the CPUs, compared to the GPUs. Further, the clock speed of the NVIDIA GPUs is around 3x lower than the CPU when executing on a single core without AVX, meaning that the CPUs move data significantly faster through the memory hierarchy. The GPU manages many more active threads and in-flight memory access requests than the CPU, in order to amortise the high latency for individual accesses. In spite of this, there has been a significant generational improvement in the L1 cache latency, as a consequence of the L1 and shared memories being combined in the new architecture.

### 4.5 Tensor Core Analysis

Throughout the paper we will refer to *tiles*, which represents a complete  $16 \times 16 \times 16$  matrix-multiply-accumulation. Each of the 80 SMs in the V100 GPU contain 8 Tensor cores, where pairs of cores are situated on each of the 4 warp schedulers in an SM (Fig. 1b). Each of the cores is capable of performing a single sub-tile  $4 \times 4 \times 4$  MMA instruction per cycle; so to perform an entire MMA for a  $16 \times 16 \times 16$  tile, a total of 64 individual sub-tile MMAs are required. As a consequence, when a warp requests a  $16 \times 16 \times 16$  MMA, there will be at least a 32 cycle latency for the two cores on a warp scheduler to fulfil the entire request.

In Fig. 5, we show the performance of increasing the number of warps resident on the GPU, for a benchmark kernel that performs a matrix multiplication without moving any data from global or shared memory. The performance scales linearly as the number of warps resident on each SM is increased from 1 to 4, and at 5 warps the performance plateaus. The results show that optimal MMA throughput is achieved when 8 warps are active on an SM.

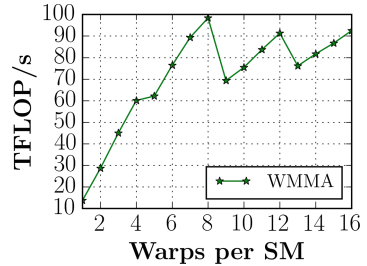


Fig. 5. Performance of MMA kernel as number of warps per thread block is increased.

### 4.6 Application Performance

The results of executing three optimised test applications written in CUDA can be seen in Fig. 6; *flow* is a 2D explicit hydrodynamics application, *hot* is a 2D implicit heat diffusion solver that uses the Conjugate Gradient method, and *neutral* is a 2D Monte Carlo neutral particle transport solver [7, 8].

The applications are intended to represent the performance profiles of important HPC applications, and the results show a significant uplift in performance between the generations of hardware. The performance differences observed for both *flow* and *hot* track the memory bandwidth improvements of the architecture, as expected. The *neutral* application suffers from issues with memory latency due to poor reuse, and the performance observed on the V100 GPU is indicative of improved memory latency hiding in the new architecture, as well as improvements to compute throughput and memory bandwidth.

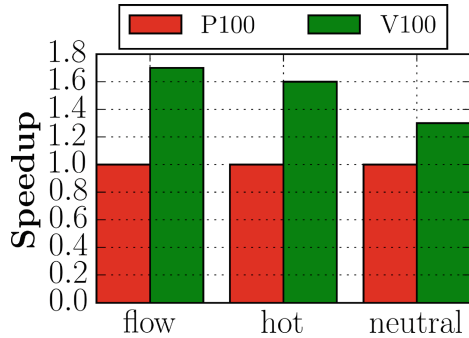


Fig. 6. Speedup observed for applications taken from the *arch* project.

### 4.7 cuBLAS Performance

Figure 7 presents the achieved FLOP/s for the varying precision cuBLAS routines, as the matrix size is increased. The half and single precision routines improve in performance, and asymptote upon reaching 8192<sup>2</sup>. In contrast, for the Tensor core implementation we observe a sudden drop in performance once the matrix size exceeds 8192<sup>2</sup>.

We compared the performance of calculating a single 16384<sup>2</sup> Tensor core MMA versus an equivalent calculation using 4 MMAs where the sub-matrices were 16384 wide on the k-th dimension and 8192 on the adjoining dimension. We observed a 1.35x improvement in performance, demonstrating that, in the short term, blocking cuBLAS for matrix sizes above 2<sup>13</sup> is necessary even when the whole matrix is resident in high bandwidth memory.

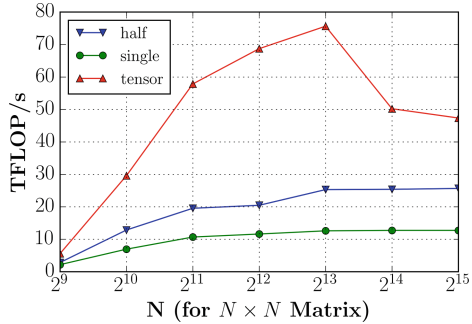


Fig. 7. cuBLAS Performance for increasing matrix sizes.

## 5 Tensor Core Accelerated MMA

In a real application, the Tensor cores will more than likely be leveraged through the cuBLAS and cuDNN interfaces; however, an aim of this research is to uncover the current state of direct programming of the cores, and so we present the efforts taken to develop an optimised matrix multiplication as a canonical example.

### 5.1 Parallelisation and Decomposition for Tensor Cores

Our parallelisation strategy was influenced by the NVIDIA CUDA matrix multiplication sample, using 128 x 128 sub-matrices, containing 64 16 x 16 x 16 MMA tiles, enabling coalescence and saturation of shared memory. We chose a block size of 256 threads, where 8 warps co-operate in performing the instructions to complete the blocked 128 x 128 MMA.

The WMMA API is a significant departure from the conventional CUDA APIs, as the developer is expected to program operations at the warp level, rather than the thread level. As such, each CUDA block is responsible for performing 64 16 x 16 x 16 MMAs per sub-matrix along the k-th dimension, with each warp performing a single MMA at a time. Prior to performing the MMA operation, tiles are loaded from shared memory into WMMA fragments, which are groups of 16 x 16 registers, declared as in Listing 1.1.

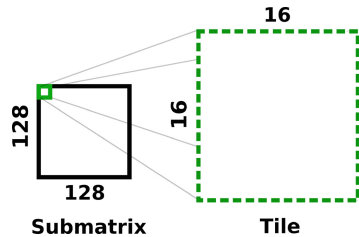


Fig. 8. Decompose 128 x 128 sub-matrices into 16 x 16 tiles.

**Listing 1.1.** WMMA fragments (`nvcuda::wmma` excluded for brevity).

```
fragment<matrix_a, 16, 16, 16, half, row_major> a;
fragment<accumulator, 16, 16, 16, float> c;
```

Our initial implementation allocated each of the warps within a CUDA block all 8 of the tiles in a row of the sub-matrix C. Each warp can fetch a single  $16 \times 16$  fragment of A and multiply that by a whole row of B, achieving perfect reuse of A but no reuse of B. The result is that every warp is required to load the entirety of B from shared memory for every processed sub-matrix, with warps storing the results in an array of fragments for each tile in each row of the sub-matrix C.

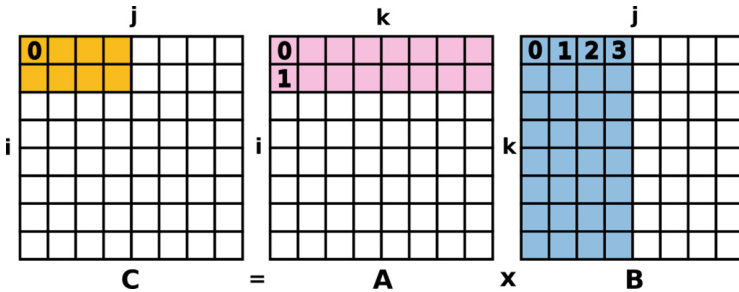
**Listing 1.2.** WMMA load A and perform MMA sync.

```
load_matrix_sync(a, &Aidx, shared_tile_lda);
mma_sync(c[cidx], a, b[bidx], c[cidx]);
```

In the main computational loop of the matrix multiplication, the A and B fragments were loaded and the MMA operation performed using the API calls in Listing 1.2.

### 5.2 Tiling and Register Optimisation

The scheme described in the previous section is inefficient in its management of fragments, introducing more shared memory requests than were actually required. It was possible to adjust the partitioning to increase the reuse of B and reduce shared memory accesses in favour of increasing register utilisation.



**Fig. 9.** The calculation of the first element of C, by multiplying the first element of A with the first chunk row of B, and then repeating the calculation for the first element of the second row of A.

Figure 9 depicts an optimisation where each warp is given a chunk of C, shaded orange, meaning the warp is only responsible for loading half of sub-matrix B per sub-matrix C. The result of this optimisation was a roughly 1.17x speedup for  $N = 16384^2$ .

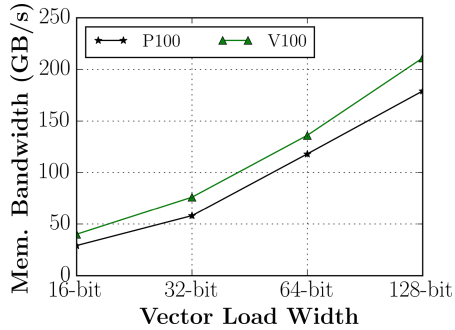


### 5.3 Vector Loads

The matrix multiplication CUDA sample for the WMMA API adapted all reads from global memory to 128-bit wide loads; as such, each of the threads was responsible for reading 8 of the 16-bit floating-point numbers at once from global memory, and populating shared memory with them. As such, each warp reads 2 rows of a  $128 \times 128$  sub-matrix.

Figure 10 shows that the memory bandwidth utilisation of a single warp resident on an SM greatly increases as the global load width is increased. This trend is true even if we have a single warp per warp scheduler on an SM, for 4 per SM on Volta, where the achieved memory bandwidth is 3.8x higher for 128-bit loads than scalar 16-bit loads.

Through benchmarking we determined that, when the number of blocks issued for a particular kernel significantly over-subscribed the number of available SMs and warp schedulers, a single warp per block could saturate memory bandwidth using 128-bit vector loads, with half precision computation. For 64-bit vector loads, two warps were required to saturate memory bandwidth.



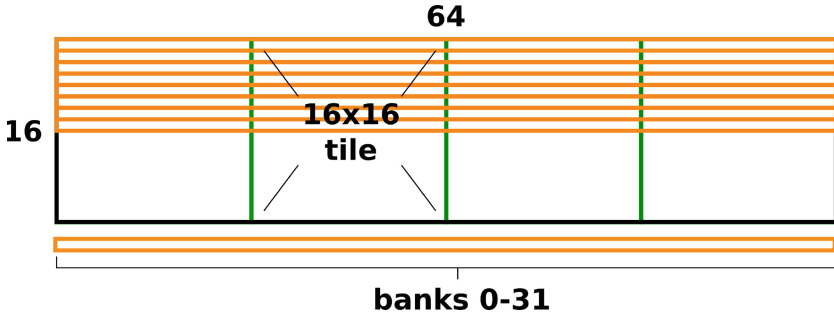
**Fig. 10.** Memory bandwidth of vector loads from global memory (16 to 128-bits) for one warp resident on each SM.

### 5.4 Shared Memory Optimisation

The shared memory on a V100 GPU, as with previous compute capabilities, is organised into 32 32-bit banks, which allow a warp to read 64 half precision values in a single cycle, two from each bank. If two threads access elements in the same bank, the latency of the resulting memory operation will increase to two cycles, a two-way bank conflict.

In Fig. 8, we show a section of a sub-matrix, where each individual tile stored in shared memory is read using a single WMMA API call. We can see in Fig. 11 that the individual banks, the orange bars, can contain 64 half precision elements, and so are distributed across an entire row of the half sub-matrix, or twice per row of the full sub-matrix. The banks line up perfectly so that the beginning of each row of the tile coincides with the beginning of the shared memory banks. As such, the 16 16-bit elements of each row of a tile reside within the same 8 32-bit banks of shared memory, meaning that, if a tile is allocated contiguously, warp-level accesses to that tile inherently lead to banks conflicts.

The exact manner in which the WMMA API moves data into fragments is not known; however, it is expected that the API uses all threads in a half-warp to access a number of rows in the tile. In theory, each thread could access 2

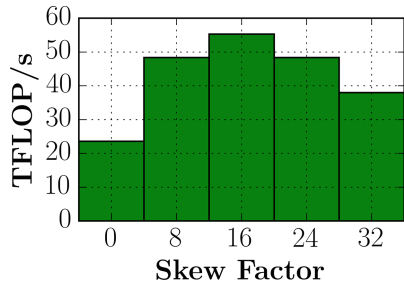


**Fig. 11.** A representation of the distribution of elements of the sub-matrix stored in the banks shared memory.

elements of each row of the tile, so the 16 threads would access 2 rows of the tile at once, resulting in a two-way bank conflict.

Padding, known as a skew factor, can be added to the leading dimension of the sub-matrix to avoid the data lining up. It is clear that a skew of 16 elements (16-bit elements of the sub-matrix) results in banks 0–7 containing row 0 of the tile, and banks 8–15 containing row 1, and so on. This ensures that rows 0–3 can be read in a single cycle by a single warp.

Given that the V100 allows the user to allocate up to 96 KB of shared memory per SM, and both A and B are 32 KB, there is enough space to pad both of the arrays in shared memory. In support of the analytic choice of 16, we demonstrate empirically that it is the best skew in Fig. 12.



**Fig. 12.** Tuning skew factor for shared memory accesses where the sub-matrix size is  $128 \times 128$ , and the tile size is  $16 \times 16 \times 16$ .

While the preceding analysis appears specific to the optimisation of general matrix multiplications, or our particular choice of block size, it is important to recognise the generality of this particular issue. We anticipate that, in the majority of use cases, the utilisation of Tensor cores will require memory to be read in from global memory and reused within the available shared memory; as such, the fact that the tiles of the sub-matrices are  $16 \times 16$  is expected to result in bank conflicts in general usage.

It is important that programmers are aware that bank conflicts are not fixed by the WMMA API. The 2.5x difference in performance observed between the 0 skew and 16 skew cases in Fig. 12 demonstrates that avoiding bank conflicts is an important part of programming with Tensor cores. Furthermore, due to the dramatic jump in computational throughput introduced by the Tensor cores, it is likely that tuning the use of shared memory, and potentially optimising for all levels of cache, will be essential for maximum performance.

## 6 Future Work

We expect that the majority of future research articles will focus on machine learning use cases for the Tensor cores, but we are hopeful that researchers will continue to investigate the potential for using them within HPC. As some of the largest pre-exascale supercomputers include V100 GPUs, it is important to discover use-cases or adaptations to existing algorithms that could leverage the Tensor cores. It will be important future work to understand the implication of attempting to include the WMMA API calls into code that is otherwise thread-level, as the shift in paradigm may make expressing some algorithms more challenging.

## 7 Related Work

An NVIDIA PARALLEL FORALL blog by Appleyard et al. [2] presents details about the Tensor hardware and API, and evaluates the performance of Tensor core accelerated routines in cuBLAS and cuDNN. The article by Harris [4] discussed mixed precision programming introduced in CUDA 8.0, providing technical details and demonstrating performance improvements for a number of use cases and libraries.

Through micro-benchmarking, Jia et al. [5] uncovered many of the architectural details of the Volta architecture such as the memory bandwidth/latency, warp scheduling policy, and instruction latencies. Markidis et al. [6] evaluated the Tensor core units in terms of programmability, performance, and precision, finding similar performance to the results presented here and exploring in detail the consequences of the loss in precision.

Reguly et al. [11] benchmarked the POWER8 GPUs in comparison to other hardware for a number of applications and benchmarks. Trott [12] performed an evaluation of the P100 GPU soon after release, and demonstrated that there were significant improvements in cache bandwidths and atomic performance.

## 8 Conclusion

The NVIDIA V100 GPU has been proven to be a significant improvement over its recent predecessor, the P100 GPU, introducing many new programmability improvements and offering exceptionally high memory bandwidth. Through benchmarking we observed that these memory bandwidth improvements have come at all levels of the memory hierarchy, but most notably the L1 cache, which offers considerably higher performance than the P100. We also observed gains in performance provided by the Tensor cores but that performance is now strongly coupled to the performance of the GPU memory hierarchy.

The Tensor cores can shift the balance of computational power within the devices to such an extent that it will be essential to carefully consider all levels of cache to successfully exploit the devices for computationally-bound algorithms.

This has important consequences for the future of HPC on those devices, as a significant amount of research is being conducted into the adaptation of algorithms into compute bound variations to support future scaling.

The organisation of the WMMA API to issue instructions at the warp rather than the thread level means that the individual  $4 \times 4 \times 4$  Tensor cores cannot be directly programmed, and the developer will only have access to  $16 \times 16 \times 16$  MMA instructions. In spite of this, the Tensor cores allow for a huge increase in compute performance when executing specific deep learning workloads, whilst other architectural changes bring benefits to HPC workloads.

## References

1. Ang, J., Cook, J., Domino, S.P., Glass, M.W., Voskuilen, G.R.: Exascale co-design progress and accomplishments. *New Front. High Perform. Comput. Big Data* **30**, 3 (2017)
2. Appleyard, J., Yokim, S.: Programming Tensor Cores in CUDA 9 (2017)
3. Deakin, T., Price, J., McIntosh-Smith, S.: Portable methods for measuring cache hierarchy performance. In: *IEEE/ACM Super Computing* (2017)
4. Harris, M.: Mixed-Precision Programming with CUDA 8 (2017)
5. Jia, Z., Maggioni, M., Staiger, B., Scarpazza, D.P.: Dissecting the NVIDIA Volta GPU architecture via microbenchmarking, April 2018
6. Markidis, S., Chien, S.W.D., Laure, E., Peng, I.B., Vetter, J.S.: NVIDIA Tensor Core Programmability, Performance & Precision. *CoRR abs/1803.04014* (2018). <http://arxiv.org/abs/1803.04014>
7. Martineau, M., McIntosh-Smith, S.: The arch project: physics mini-apps for algorithmic exploration and evaluating programming environments on HPC architectures. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 850–857, September 2017
8. Martineau, M., McIntosh-Smith, S.: Exploring on-node parallelism with neutral, a Monte Carlo neutral particle transport mini-app. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE (2017)
9. McCalpin, J.: Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA)*, pp. 19–25 (1995)
10. NVIDIA Corporation: Parallel Thread Execution ISA Version 6.1 (2017)
11. Reguly, I.Z., Keita, A.K., Giles, M.B.: Benchmarking the IBM Power8 processor. In: *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*, pp. 61–69. IBM Corp. (2015)
12. Trott, C.R.: Early Experience with P100 on POWER8. Technical report, Sandia National Lab. (SNL-NM), Albuquerque, NM (United States) (2016)