

Numerical Behavior of NVIDIA Tensor Cores

Fasi, Massimiliano and Higham,
Nicholas J. and Mikaitis, Mantas and Pranesh, Srikara

2020

MIMS EPrint: **2020.10**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: <http://eprints.maths.manchester.ac.uk/>

And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

Numerical Behavior of NVIDIA Tensor Cores

Massimiliano Fasi¹, Nicholas J. Higham¹, Mantas Mikaitis¹, and Srikara Pranesh¹

¹Department of Mathematics, The University of Manchester, Manchester, UK,

Corresponding author:

Mantas Mikaitis¹

Email address: mantas.mikaitis@manchester.ac.uk

ABSTRACT

We explore the floating-point arithmetic implemented in NVIDIA tensor cores, which are hardware accelerators for mixed-precision matrix multiplication available on the Volta, Turing, and Ampere microarchitectures. Using Volta V100 and Turing T4 graphics cards, we determine what precision is used for the intermediate results, whether subnormal numbers are supported, what rounding mode is used, in which order the operations underlying the matrix multiplication are performed, and whether partial sums are normalized. These aspects are not documented by NVIDIA, and we gain insight by running carefully designed numerical experiments on these hardware units. Knowing the answers to these questions is important if one wishes to: 1) accurately simulate NVIDIA tensor cores on conventional hardware; 2) understand the differences between results produced by code that utilizes tensor cores and code that uses only IEEE 754-compliant arithmetic operations; and 3) build hardware that computes a matrix-matrix product matching the results of the NVIDIA tensor cores. As part of this work we provide a testsuite that can be easily adapted to test the latest tensor cores available in the NVIDIA Ampere A100, once those graphics cards become easily accessible. Moreover, we identify a non-monotonicity issue that arises in floating-point multi-operand addition if the intermediate results are not normalized.

1 INTRODUCTION

One hundred and twelve of the computers in the June 2020 TOP500 list,¹ feature NVIDIA graphics processing units (GPUs) based on the Volta, Turing, and Ampere microarchitectures. A prominent feature of these microarchitectures is tensor cores, which are specialized hardware accelerators for performing the matrix multiply-accumulate

¹<https://www.top500.org/lists/top500/list/2020/06/>

Table 1. Processing units or architectures equipped with mixed-precision matrix multiply-accumulate accelerators. The notation $m \times k \times n$ refers to the matrix multiply-accumulate operation in (1) where C and D are $m \times n$ matrices, and A and B have size $m \times k$ and $k \times n$, respectively.

Year of release	Device	Matrix dimensions	Input format	Output format	Reference
2016	Google TPU v2	$128 \times 128 \times 128$	bfloat16	binary32	(Google, 2020)
2017	Google TPU v3	$128 \times 128 \times 128$	bfloat16	binary32	(Google, 2020)
2017	NVIDIA V100	$4 \times 4 \times 4$	binary16	binary32	(NVIDIA, 2017)
2018	NVIDIA T4	$4 \times 4 \times 4$	binary16	binary32	(NVIDIA, 2018)
2019	ARMv8.6-A	$2 \times 4 \times 2$	bfloat16	binary32	(Arm Limited, 2020)
2020	NVIDIA A100	$8 \times 8 \times 4$	bfloat16	binary32	(NVIDIA, 2020b)
		$8 \times 8 \times 4$	binary16	binary32	
		$4 \times 2 \times 2$	binary64	binary64	
		$4 \times 8 \times 4$	TensorFloat-32	binary64	

operation

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{binary16 or binary32}} = \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{binary16 or binary32}} + \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{binary16}} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{binary16}} \quad (1)$$

where A , B , C , and D are 4×4 matrices. The entries of A and B must be in binary16 format, whereas those of C and D can be either binary16 or binary32 floating-point numbers depending on the accumulation mode (the A100 has different matrix dimensions and numerical formats: see Table 1). The element d_{ij} can be seen as the sum of c_{ij} and the dot product between the i th row of A and the j th column of B , so that, for instance

$$d_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} + c_{11}. \quad (2)$$

Unfortunately, NVIDIA provides very little information about the numerical features of these units, and many questions naturally arise. The white paper that describes the Volta microarchitecture (NVIDIA, 2017, p. 15) states that²

Tensor Cores operate on FP16 input data with FP32 accumulation. The FP16 multiply results in a full precision product that is then accumulated using FP32 addition with the other intermediate products for a $4 \times 4 \times 4$ matrix multiply.

The official documentation (NVIDIA, 2020a) adds only a few more details:

²The IEEE 754 (IEEE, 2019) formats binary16 and binary32 are sometimes referred to as fp16 (or FP16) and fp32 (or FP32), respectively.

Element-wise multiplication of matrix A and B is performed with at least single precision. When `.ctype` or `.dtype` is `.f32`, accumulation of the intermediate values is performed with at least single precision. When both `.ctype` and `.dtype` are specified as `.f16`, the accumulation is performed with at least half precision. The accumulation order, rounding and handling of subnormal inputs is unspecified.

From a numerical point of view, many essential aspects of tensor cores are not specified. This lack of key information makes it challenging to simulate tensor cores on conventional IEEE 754-compliant systems or build hardware to emulate their behavior, and it can lead to unexpected differences between the results computed on NVIDIA devices with tensor cores enabled or disabled. The IEEE 754 standard for floating-point arithmetic provides a somewhat relaxed set of requirements for reduction operations such as multi-operand addition and dot product (IEEE, 2019, Sec. 9.4), since it does not prescribe the order in which the partial sums should be evaluated and allows the use of a higher-precision internal format. In particular, the standard does not specify: 1) whether this internal format should be normalized, as it would be if the multi-operand addition were implemented using IEEE 754 elementary arithmetic operations, 2) which rounding mode should be used, and 3) when the rounding should happen. These loose requirements can potentially cause the results computed with a given multi-operand addition unit to be significantly different from those obtained using other hardware implementations or a software implementation based on IEEE 754-compliant elementary arithmetic operations.

With matrix multiplication being ubiquitous in artificial intelligence, accelerators for mixed-precision matrix multiply-accumulate operations are becoming widely available, as Table 1 shows. Hardware vendors often design these units focusing on performance rather than numerical reliability, and this may lead to the implementation of unconventional, non-IEEE-compliant arithmetics. Some of the hardware units in Table 1, for instance, use `bfloat16`, a 16-bit format that allocates 8 bits to the significand (including the implicit bit) and 8 bits to the exponent, and does not support subnormal numbers (Intel Corporation, 2018, 2020). The NVIDIA Ampere A100 GPUs introduce a new 19-bit format called `TensorFloat-32`, which was designed to have the same dynamic range as `binary32` (8 bits) and the same precision as `binary16` (11 fraction bits, including the implicit bit) (NVIDIA, 2020b). In order to better understand the differences between the results computed using different systems, it is necessary to develop techniques to probe the numerical features of these units. This is not much different from the situation with single and double precision before the wide adoption of the IEEE 754-1985 standard.

Using idiosyncrasies of floating-point arithmetic, we design tests to better understand the numerical behavior of tensor cores, extending the testing approach recently introduced by Hickmann and Bradford (2019). Our aim is to explore the following questions about tensor cores.

- Are subnormal inputs supported or are they flushed to zero? Can tensor cores produce subnormal numbers?
- Are the multiplications in (2) exact and the additions performed in `binary32` arithmetic, resulting in four rounding errors for each element of D ? In what order are the four additions in (2) performed?

- What rounding mode is used in (2)?
- Is the result of each floating-point operation in (2) normalized, or do tensor cores only normalize the final sum? What rounding mode is used for the normalization?

The answers to these questions are of wide interest because although they were built to accelerate training of deep neural networks (NVIDIA, 2017, p. 12), accelerators are increasingly being used in general purpose scientific computing, where their fast low precision arithmetic can be exploited in mixed-precision algorithms (Abdelfattah et al., 2020), for example in iterative refinement for linear systems (Haidar et al., 2018a,b, 2020).

The results discussed here were produced by running our test suite³ on NVIDIA Tesla V100 SXM2 16GB (Volta microarchitecture) and NVIDIA Tesla T4 16GB (Turing microarchitecture) GPUs. We stress, however, that the ideas we employ are very general and can be exploited to understand the numerical features of any hardware accelerator based on operations of the form (2). Finally, it is worth noting that binary16 arithmetic in NVIDIA CUDA cores is not fully IEEE 754 compliant, as round-to-nearest is the only rounding mode implemented for elementary arithmetic operations (NVIDIA, 2020c)—we use this observation in Section 3.1.1.

2 PREVIOUS WORK

From a hardware perspective, instruction-level details, register configuration, and memory layout of the tensor cores in the NVIDIA Volta (Jia et al., 2018b; Yan et al., 2020) and Turing (Yan et al., 2020; Jia et al., 2018a) GPUs have been extensively described. Another study by Basso et al. (2020) explores the reliability of tensor cores in terms of rate of hardware errors in matrix multiplications. The main finding is that low-precision operations and usage of tensor cores increases the amount of correct data produced by the GPU, despite increasing the impact of faults due to lower precision of the data. In order to quantify the accuracy of tensor cores, Blanchard et al. (2020) provide a rounding error analysis of what they call a block fused multiply-add (FMA), a generalization of the multiply-accumulate operation in (1) in which the matrix sizes, the precisions of the arguments, and the internal precision of the accumulator are taken as parameters.

Markidis et al. (2018) discuss various aspects of tensor cores and propose a technique, called precision refinement, to enhance the accuracy of mixed-precision matrix multiplication. Improving the accuracy of tensor-core-based matrix multiplications was further explored by Mukunoki et al. (2020).

None of these sources, however, examines to what extent tensor cores conform to the IEEE 754 standard or investigates how tensor cores compare with a matrix multiply-accumulate operation based on dot products implemented in software. Hickmann and Bradford (2019) explore some details of the numerical behavior of tensor cores with the main goal of inferring the hardware-level design of these units. Our work follows a similar approach and complements their findings by supplying further insights into the subject. We show that the additions in (2) are performed starting from the operand that is largest in magnitude, that at least 2 extra bits are used for carries, and that (2)

³Available at <https://github.com/mfasi/tensor-cores-numerical-behavior>.

Table 2. Parameters of various floating-point formats: number of digits of precision including the implicit bit (p), boundary of the exponent range (e_{\min} and e_{\max}), machine epsilon (ϵ), and smallest positive representable normal (f_{\min}) and subnormal (s_{\min}) numbers.

Property	binary16	bfloat16	TensorFloat-32	binary32
p	11	8	11	24
e_{\max}	15	127	127	127
e_{\min}	-14	-126	-126	-126
ϵ	2^{-10}	2^{-7}	2^{-10}	2^{-23}
f_{\min}	2^{-14}	2^{-126}	2^{-126}	2^{-126}
s_{\min}	2^{-24}	2^{-133}	2^{-136}	2^{-149}

may be non-monotonic for certain sets of inputs. Furthermore, we consider the second generation of tensor cores, which equips the Turing T4 GPUs, and conclude that their internal accumulator has an extra bottom bit compared with the tensor cores on Volta V100 GPUs. Finally, unlike Hickmann and Bradford (2019) we make our test suite freely available, in order to guarantee reproducibility and facilitate testing other matrix multiply-accumulate units, such as the third generation tensor cores in the latest NVIDIA Ampere A100 GPUs NVIDIA (2020b).

3 RESULTS

In this section we describe our testing methods and give the results from the two NVIDIA graphics cards.

3.1 NVIDIA Volta Tensor Cores

Tensor cores can be accessed using the cuBLAS library, or the native hardware assembly instructions `HMMMA.884` (Jia et al., 2018b,a) and `HMMMA.1688` (Yan et al., 2020; Jia et al., 2018a). In our experiments, we opted for the warp-level C++ function `wmma::mma_sync()`, which performs a $16 \times 16 \times 16$ matrix multiply-accumulate operation. This is the lowest level interface to access the tensor cores in the NVIDIA CUDA programming environment. In order to use only a single tensor core, we set all but the top left 4×4 blocks to 0. We ensure that our experiments do use the tensor cores by running our test suite with the NVIDIA profiler `nvprof` and additionally observing that `HMMMA` instructions are being used in the assembly output from CUDA binary files. At software level, tensor cores can be used either in `binary16` or `binary32` mode, which defines what is the number format of D in (1). We refer to this by `binary16` or `binary32 mode` further in the paper.

The experiments in this section were run on an NVIDIA Tesla V100 SXM2 16GB (Volta microarchitecture) GPU.

3.1.1 Support for subnormal numbers

We start by investigating the support for subnormal numbers, as this knowledge will dictate what range of input values we are allowed to use in further tests.

Table 2 reports precision, exponent range, machine epsilon, and the smallest representable subnormal and normal positive values for the four floating-point formats supported by tensor cores. Based on the table we highlight two points. First, conversion from binary16 to binary32 does not result in subnormal numbers. Second, the product of two binary16 numbers requires at most 22 bits for the significand, 6 bits for the exponent and one for the sign, and thus can be represented exactly in binary32. In terms of tensor cores, there are multiple questions regarding the support of subnormal numbers.

1. Can tensor cores take binary16 subnormal numbers as inputs for A and B in (2) without flushing them to zero, use them in computation, and return binary16 or binary32 normal or subnormal answers?
2. Can tensor cores take binary32 subnormal numbers as inputs for C in (2) without flushing them to zero, use them in computation, and return subnormal binary32 answers?
3. Can tensor cores compute subnormal numbers from normal numbers and return them?

The first question can easily be answered by setting in (2) $a_{11} = 2^{-24}$, $b_{11} = 2^2$ (arbitrarily chosen), and the other elements to zero. The tensor cores return the subnormal result $a_{11}b_{11} = 2^{-22}$ in both binary16 and binary32 mode.

To clarify the second point, a similar idea can be used: setting c_{11} to the smallest positive binary32 subnormal 2^{-149} and all the elements of A and B to zero yields $d_{11} = 2^{-149}$, which confirms that the subnormal number c_{11} is not altered by the dot product in (2). We note, however, that the need for binary32 subnormals is questionable. The absolute value of the smallest nonzero value that can be produced from the multiplication of two binary16 numbers is 2^{-48} , thus c_{11} would not contribute to the sum if it were a binary32 subnormal: in binary32 arithmetic with round-to-nearest $2^{-48} + x > 2^{-48}$ only if $x > 2^{-48} \cdot 2^{-24} = 2^{-72}$, which is normal in binary32.

For the third point, we can obtain subnormal numbers as outputs in a several ways. For instance, we can set a_{11} to 2^{-14} , the smallest normal number in binary16, and b_{11} to 2^{-1} , and confirm that tensor cores return the binary16 subnormal 2^{-15} in both binary16 and binary32 modes. Another possibility is to set $a_{11} = 2^{-14}$, $b_{11} = 1$, $c_{11} = -2^{-15}$, which produces the subnormal binary16 number $d_{11} = 2^{-15}$. As mentioned above, it is not possible to obtain subnormal binary32 number from binary16 inputs in (2). In summary, these experiments demonstrate that there is full support for subnormal inputs in tensor cores.

One might wonder whether tensor cores natively support subnormals or some degree of software interaction is present. The NVIDIA profiler confirms that the experiments discussed in this section make use of the tensor cores, but we implemented an additional test to further reinforce the evidence that subnormals are supported in hardware. In Section 3.1.3 we show that tensor cores use round-toward-zero. We can use the fact that CUDA cores provide only round-to-nearest for binary16 computations to show that subnormals are in fact manipulated with tensor cores. In order to do so, we can set a_{11} and a_{12} to 1, b_{11} to the binary16 subnormal $2^{-23} + 2^{-24}$, b_{21} to 2 and the other elements of A and B to 0. Since the addition in (2) is done in binary32 arithmetic, the smallest

value that can be exactly added to $b_{21} = 2$ is 2^{-22} . Since $b_{11} = 2^{-23} + 2^{-24}$ is below this value and is $3/4$ of 2^{-22} , it will be either lost (round-toward-zero) or rounded up due to round-to-nearest if the summation is done in software to support subnormals. We found that the sum returned 2, which means that b_{11} was rounded down—further indicating that subnormals are natively supported by tensor cores.

3.1.2 Accuracy of the dot products

Our second goal is to test the accuracy of the dot product (2) with the tensor cores. The first step is to check that the multiplications of binary16 values are computed exactly, which implies that the products must be kept in some wider intermediate format and accumulated without being rounded back to binary16. Specifically we want to test that $a_{11}b_{11}$ and the other products are exact. This can be achieved by ensuring that the four multiplications produce floating-point numbers that are not representable in binary16 and checking that these are preserved and returned as binary32 entries of D .

In order to demonstrate this, we set the first row of A and the first column of B to $1 - 2^{-11}$ and c_{11} to 0. Each of the partial products evaluates exactly to

$$(1 - 2^{-11}) \cdot (1 - 2^{-11}) = 1 - 2^{-10} + 2^{-22}$$

which, if the products were stored in binary16 precision, would be rounded to $1 - 2^{-10}$ or $1 - 2^{-11}$, depending on the rounding mode. As tensor cores produce the exact binary32 answer $d_{11} = 4 \cdot (1 - 2^{-10} + 2^{-22})$, we conclude that partial products are held exactly.

Another question is whether the precision of the 5-operand addition in (2) changes in any way when binary16 is used for C and D in (1). The test is to set $a_{11} = b_{11} = a_{12} = 1 - 2^{-11}$, $b_{21} = 2^{-11}$, and the remaining elements to 0. In this test, the first product is as before $a_{11}b_{11} = 1 - 2^{-10} + 2^{-22}$ and requires precision higher than binary16 to be represented, whereas the second evaluates to $a_{12}b_{21} = 2^{-11} - 2^{-22}$. The sum of these two products is $a_{11}b_{11} + a_{12}b_{21} = 1 - 2^{-10} + 2^{-11}$, which is representable in binary16 but could not be calculated exactly by a binary16 accumulator, since the first product requires higher precision. Indeed we found that tensor cores output the exact value, confirming that the partial products are still held exactly even when C and D are in binary16 format.

A third question concerns the number of rounding errors in the 5-operand addition of the partial products. The dot product in (2) contains 4 additions: 3 additions sum the exact partial products and a fourth adds the binary32 argument c_{11} . Our expectation is that the additions are done in binary32 rather than exactly, as indicated by NVIDIA (2017, 2020a). In order to confirm this, we can set the first row of A to 1, thereby reducing (2) to

$$d_{11} = b_{11} + b_{21} + b_{31} + b_{41} + c_{11}, \quad (3)$$

and then run 5 different cases with one of the addends in (3) set to 1 and the rest set to 2^{-24} . In this test, an exact addition would return $1 + 2^{-22}$, whereas inexact binary32 arithmetic would cause 4 round-off errors when adding 2^{-24} to 1, causing the number 1 to be returned. All permutations return $d_{11} = 1$, leading to the following conclusions.

- In the worst case each element of D includes four rounding errors, which conforms to the block FMA model used by Blanchard et al. (2020, Sec. 2.1).

- The partial products in (2) are not accumulated in a fixed order, but always starting from the largest value in magnitude. This sorting is necessary in order to know which arguments require to be shifted right in the significand alignment step of a standard floating-point addition algorithm (Muller et al., 2018, Sec. 7.3), (Tenca, 2009), and is most likely done in hardware. This is in line with the literature on hardware dot products (Kim and Kim, 2009; Tao et al., 2013; Sohn and Swartzlander, 2016; Kaul et al., 2019), where either sorting or a search for the maximum exponent is performed. Furthermore, this experiment demonstrates that none of the additions is performed in parallel with the first addition to the largest magnitude value: if evaluated before arguments are shifted right relative to the largest magnitude arguments exponent, any other sum would compute $2^{-24} + 2^{-24} = 2^{-23}$, a value that then could be added exactly to the total sum.

In summary, each entry of D in (1) can have up to four rounding errors, and the 5-operand additions that compute each element are performed starting from the largest summand in absolute value.

3.1.3 Rounding modes in tensor core computations

If binary32 mode is used, only the four additions in (2) can be subject to rounding errors. The IEEE 754 standard defines four rounding modes for elementary arithmetic operations (IEEE, 2019, Sec. 4.3): round-to-nearest with even-on-ties, round-toward-zero, round-down, and round-up. In this section we use the notation defined by Muller et al. (2018, Sec. 2.2.1) and denote these four rounding operators by RN, RZ, RD, and RU, respectively.

As round-to-nearest is the default rounding direction in the IEEE 754 standard, we start by testing whether this is the rounding mode used by the tensor cores. This can be verified by setting any two partial products to values such that one of them would be rounded up only if round-to-nearest or round-up were used. If the result is rounded down we can conclude that the tensor cores implement either round-toward-zero or round-down, but neither round-to-nearest nor round-up (Figure 1b). One such test is to set in (3) $b_{11} = 2$, $b_{21} = 2^{-23} + 2^{-24}$, and the remaining entries in the first column of B to 0. Note that in binary32 arithmetic $\text{RN}(2+x) > 2$ if $x > 2 \cdot 2^{-24} = 2^{-23}$, whereas the smallest positive y such that $\text{RZ}(2+y) > 2$ is $2 \cdot 2^{-23} = 2^{-22}$. The choice $b_{21} = \frac{3}{4} \cdot 2^{-22}$ is such that $x < b_{21} < y$, thus $\text{RN}(b_{11} + b_{21}) = \text{RU}(b_{11} + b_{21}) = 2 + 2^{-22}$ while $\text{RZ}(b_{11} + b_{21}) = \text{RD}(b_{11} + b_{21}) = 2$. Running this experiment on tensor cores returns $c_{11} = 2$, suggesting that either round-to-zero or round-down is used for the additions in (2).

We can discriminate between these two rounding modes by repeating the same experiment on the negative semiaxis (Figure 1a), by changing the sign of the nonzero elements in B . This experiment produces $c_{11} = -2$, and assuming that the rounding mode does not depend on the input, we conclude that the additions in (2) are performed in round-toward-zero. We note that this rounding mode is known to be the cheapest option to implement (Santoro et al., 1989, Sec. 6.1) and is usually chosen for that reason.

When tensor cores are used in binary16 mode, the result computed in the format of the internal accumulator of the 5-operand adder has to be rounded to binary16 before being returned. In order to test the rounding mode used by this operation, we set

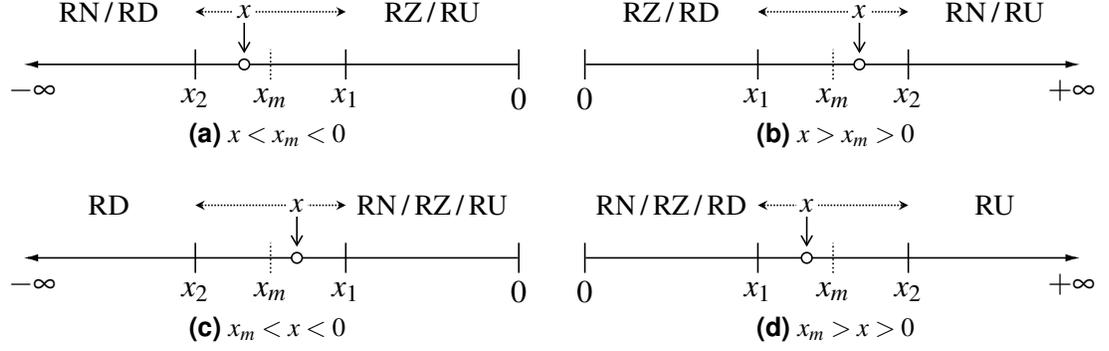


Figure 1. Demonstration of the possible IEEE 754 rounding modes for different positions of the exact value x ; x_1 and x_2 are the two floating-point numbers closest to x , and x_m is the half-way point between them. The dotted arrows surrounding x show the direction in which various rounding modes would round it.

$a_{11} = a_{12} = 2^{-24}$, $b_{11} = 2^{-1}$, $b_{21} = 2^{-2}$, and the rest of elements of A and B as well as c_{11} to 0. The exact result of the dot product in this case is $2^{-25} + 2^{-26}$, which is not representable in binary16, and therefore will cause rounding errors in the result. Note that $2^{-25} + 2^{-26} = \frac{3}{4} \cdot 2^{-24}$, therefore $\text{RN}(2^{-25} + 2^{-26}) = \text{RU}(2^{-25} + 2^{-26}) = 2^{-24}$ while $\text{RZ}(2^{-25} + 2^{-26}) = \text{RD}(2^{-25} + 2^{-26}) = 0$. The fact that tensor cores return 2^{-24} confirms that round-toward-zero is not used, thereby suggesting that this conversion is performed in software rather than inside tensor cores that use round-toward-zero as we have determined above.

We also demonstrate in Figures 1c and 1d how round-down and round-up could alternatively be determined (by choosing x such that $|x| < |x_m|$).

3.1.4 Features of the accumulator

We now discuss some tests that allowed to determine various features of the internal accumulator of the 5-operand adder calculating (2). The quotes from NVIDIA that we provided in Section 1 indicate that the internal accumulation is done in binary32 format; here we show that the format has higher precision and that the partial sums are not normalized.

Extra bits in the alignment of significands In order to compute the sum of two floating-point values, the floating-point adder matches the exponents of the two summands by shifting the significand of the number that has the smaller exponent to the right. In general this operation causes loss of information, as the least significant bits of the shifted significand are typically discarded, but it is customary to retain a few of these digits to guard the computation against numerical cancellation and to obtain correct rounding in round-to-nearest, round-up, and round-down. As tensor cores use truncation in the additions, we know that they do not require any such guard digits for rounding, and we can easily show that in fact they do not implement guard digits at all. If in (3) we set $b_{11} = 1$ and $c_{11} = -1 + 2^{-24}$, the tensor cores return $d_{11} = 2^{-23}$, which represents a relative error of $(2^{-23} - 2^{-24})/2^{-24} = 1$.

Normalization in addition When two floating-point values are added, the significand of the result may become larger than 2, in which case a normalization step (shifting the significand right by one place and increase the exponent by one) is required (Muller et al., 2018, Sec. 7.3). In an IEEE 754-compliant arithmetic, the result of each partial sum in (2) would be normalized, as floating-point adders always normalize the result in order to produce a normalized floating-point number. But tensor cores compute the whole expression (2) in hardware rather than by IEEE 754 elementary arithmetic operations, and it is natural to ask whether each partial result in (2) is normalized or only the final answer is. We can verify this by adding values chosen so to produce different results with and without normalization of the partial sums. In (3) we set $c_{11} = 1 - 2^{-24}$ and the elements of the first column of B to 2^{-24} .

Recalling that the values are accumulated on the summand of largest magnitude, we start by examining what would happen if each partial result were normalized. The exact value of the partial sum $s := c_{11} + 2^{-24}$ is 1, and the normalization step would shift the significand by one bit to the right. At this point the three remaining addends would be smaller than the least significant bit of the partial sum, thus adding them to s separately would have no effect with round-to-zero. If the partial results were not normalized, on the other hand, the partial sum of $c_{11} + 2^{-24}$ would be held with one extra bit and therefore the rest of the addends could be added into it. Running this test on tensor cores shows that only the final result of the dot product is normalized. This has probably been done in order to simplify the implementation, a choice that has also been made for example in the hardware accelerator for performing vector inner product described by Kim and Kim (2009).

Normalization in subtraction As the products in (2) can be positive as well as negative, some of the partial sums can in fact be subtractions. The significand of the difference of two-floating point numbers may be smaller than 1, in which case the result has to be normalized by shifting the significand left and decreasing the exponents accordingly until the result becomes a normal number. We can show that tensor cores do not perform this kind of normalization as follows. If in (3) we set $c_{11} = -1 + 2^{-24}$, and two of the elements of the first column of B to 1 and -2^{-24} , we have that d_{11} evaluates to 0 if the partial sums are normalized. Instead, running this experiment on tensor cores yields $d_{11} = 2^{-23}$, which can be explained as follows. When the sum is evaluated as $(1 + c_{11}) - 2^{-24}$, then the lack of guard digit implies that $1 + c_{11}$ evaluates to 2^{-23} , and if the partial results were normalized the tensor cores would return $2^{-23} - 2^{-24}$, which can be represented exactly in binary32 format's precision. If, on the other hand, the sum were evaluated as $(-2^{-24} + 1) + c_{11}$, the first sum would return 1 due to the lack of guard digit, and the lack of normalization would not have any effect in this case. We can conclude that the result of the subtraction is not normalized, as long as we assume that the summands in (3) are accumulated on the largest in magnitude in a fixed order.

Extra bits for carry out Another question concerns the number of extra bits required due to lack of normalization. If only the final result is normalized, then accumulating k addends requires $\lceil \log_2 k \rceil$ bits for the carry-out bits, and the hardware for accumulating the 5 values in (2) would internally require $\lceil \log_2 5 \rceil = 3$ extra carry-out bits at the top of the significand. We can prove that the internal accumulator of the 5-operand adder in tensor cores has at least two extra bits as follows. In (3) we take $c_{11} = 1 + 2^{-22} + 2^{-23}$,

which sets the two least significant bits of the significand to 1, and assign to the first column of B a permutation of the values 1, 1, 1, and 2^{-23} . We consider all four possible permutations of these values in the first column of B , as we assume that the addends apart from the largest in magnitude are not sorted. The main idea is to show that when 1 is accumulated three times into c_{11} the last two bits of c_{11} are not dropped as they would be in IEEE 754 floating-point arithmetic, since the sum is larger than 4 and would have to be shifted right by two positions in order to be normalized. Then, when 2^{-23} is added at the end, the carry propagates into the third bottom bit and therefore is not lost in the final normalization step. If there are 2 extra bits, then all 4 combinations of ordering the first column of B would return the exact result of $4 + 2^{-21}$. Running these tests on tensor cores, we found that all four combinations returned the exact result for this test case, thereby proving that the significand of the internal accumulator has at least 2 extra bits for carries.

It is worth noting at this point that if 1) there is no normalization, 2) the additions in (2) start with the largest value in magnitude, and 3) all of the significands of the addends are shifted right relative to the exponent of the largest value in magnitude, then the order of the rest of the addends will not impact the final result.

In the test case above, by replacing one of the 2^{-23} by 1 we can also confirm, using the methods developed in Section 3.1.3, that the rounding mode in the final normalization step (internal accumulator conversion to binary32 answer) is round-toward-zero.

Monotonicity of dot product The observation in section 3.1.4 raises one final question regarding the monotonicity of the sums in (2). The accumulation is monotonic if in floating-point arithmetic the sum $x_1 + \dots + x_n$ is no larger than $y_1 + \dots + y_n$ when $x_i \leq y_i$ for all $1 \leq i \leq n$.

We can show that the lack of normalization causes the dot product in tensor cores—and most likely in any other similar architectures in which partial sums are not normalized (Kim and Kim, 2009; Tao et al., 2013; Sohn and Swartzlander, 2016; Kaul et al., 2019)—to behave non-monotonically. Let us set in (3) all the elements in the first column of B to 2^{-24} and then c_{11} to $1 - 2^{-24}$ and 1 in turn. When $c_{11} = 1 - 2^{-24}$, the difference in exponents guarantees that the values in B are large enough to be added to c_{11} . This causes the result to become larger than 1, requiring a normalization that returns $1 - 2^{-24} + 3 \cdot 2^{-24} = 1 + 2^{-23}$. On the other hand, when $c_{11} = 1$, none of the summands in (3) is large enough to be added to c_{11} , as the elements in the first column of B are all zeroed out during the significand alignment step of each addition. This happens because the exponent of 1 is larger than that of $1 - 2^{-24}$. In summary, the following two calculations demonstrate that tensor cores can produce non-monotonic behavior:

$$\begin{aligned} d_{11} &= c_{11} + 2^{-24} + 2^{-24} + 2^{-24} + 2^{-24} = 1 + 2^{-23} \quad \text{when } c_{11} = 1 - 2^{-24}, \\ d_{11} &= c_{11} + 2^{-24} + 2^{-24} + 2^{-24} + 2^{-24} = 1 \quad \text{when } c_{11} = 1. \end{aligned}$$

3.2 NVIDIA Turing tensor cores

NVIDIA Turing T4 GPUs are equipped with the second generation of tensor cores, which adds an integer matrix multiply-accumulate operation. It is not documented whether the binary16/binary32 tensor core arithmetic in Turing chips differs from that in Volta cards, therefore it is of interest to run the test suite we designed on one of the Turing cards.

We ran all the above experiments on an NVIDIA Tesla T4 16GB (Turing microarchitecture) GPU, and noticed that some of the results were different from those obtained on a V100 GPU. We found that this is due to the presence of an additional extra bit of precision at the bottom of the significand of the internal accumulator of the 5-operand adder. This has an impact over several of the tests above: the operation $1 + 2^{-24} + 2^{-24}$, for instance, can now be performed exactly due to the extra bit. The results obtained on the V100 GPU can be replicated by means of a suitable change of the constants that are chosen depending on the number of extra bits in the accumulator. For instance, in the test for the order of operations in Section 3.1.2, the constant 2^{-24} should be replaced by 2^{-25} , which is the value of the next bit to the right. Using this approach, we found that all of the properties that we have found about the tensor cores in V100 GPUs are also present in the second version of tensor cores in the T4 GPUs, except the extra bit at the bottom of the internal storage of the 5-operand adder. Expressing as fixed-point format with integer and fraction lengths $\{I.F\}$, the V100 has a format $\{3.23\}$ (or $\{4.23\}$ if 3 extra bits for carries are present as discussed in Section 3.1.4) and the T4 has a format $\{3.24\}$ (or $\{4.24\}$) for computing the significands before the final normalization and rounding to $\{1.23\}$ (the format of the significand for binary32).

4 CONCLUSIONS

In summary, our experiments indicate that the tensor cores in the NVIDIA V100 architecture have the following numerical features.

- Subnormal numbers in binary16 and binary32 are supported (new).
- The binary16 products in (2) are computed exactly, and the results are kept in full precision and not rounded to binary16 after the multiplication (confirmed).
- The five summands in (2) are accumulated starting with the largest in absolute value (new).
- The additions in (2) are performed using binary32 arithmetic (confirmed) with round-toward-zero (new).
- Only the final result of (2) is normalized; the partial sums are not, but the accumulator uses two extra bits for carries (new).
- The dot products in tensor cores are non-monotonic: in some cases, increasing the magnitude of the inputs to (2) reduces the magnitude of the final result when the summands in (2) are all nonnegative or all nonpositive (new).

The same properties were found in the second generation tensor cores which equip the NVIDIA T4 GPU's, the main difference being one extra bit of precision in the significand of the internal accumulator of the binary32 5-operand adder.

The test suite that we have developed as part of this work can be used to test various properties of the floating-point arithmetic of future versions of tensor cores as well as similar accelerators. We aim to keep extending our test suite by adding new test cases for standard non-mixed-precision binary32 or binary64 dot product or matrix multiply units,

as well as for integer arithmetic. The new NVIDIA Turing and Ampere tensor cores, for instance, added support for 4- and 8-bit integer modes (NVIDIA, 2018, 2020b), and rounding issues become relevant when these are used to implement fixed-point arithmetic. Furthermore, the NVIDIA Ampere microarchitecture adds the bfloat16, TensorFloat-32, and binary64 formats to the tensor cores, and we aim to test these cards using the techniques we have developed here as soon as they become available.

5 ACKNOWLEDGEMENTS

The authors thank the University of Manchester for providing access to the NVIDIA V100 graphic cards through the Computational Shared Facility.

REFERENCES

- Abdelfattah, A., Anzt, H., Boman, E. G., Carson, E., Cojean, T., Dongarra, J., Gates, M., Grützmacher, T., Higham, N. J., Li, S., Lindquist, N., Liu, Y., Loe, J., Luszczek, P., Nayak, P., Pranesh, S., Rajamanickam, S., Ribizel, T., Smith, B., Swirydowicz, K., Thomas, S., Tomov, S., Tsai, Y. M., Yamazaki, I., and Yang, U. M. (2020). A survey of numerical methods utilizing mixed precision arithmetic. ArXiv:2007.06674.
- Arm Limited (2020). Arm architecture reference manual. Technical Report ARM DDI 0487F.b (ID040120).
- Basso, P. M., Dos Santos, F. F., and Rech, P. (2020). Impact of tensor cores and mixed-precision on the reliability of matrix multiplication in GPUs. *IEEE Trans. Nucl. Sci.* Early access accepted paper.
- Blanchard, P., Higham, N. J., Lopez, F., Mary, T., and Pranesh, S. (2020). Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores. *SIAM J. Sci. Comput.*, 42(3):C124–C141.
- Google (2020). System architecture. Online: <https://cloud.google.com/tpu/docs/system-architecture>. Accessed: April 15, 2020.
- Haidar, A., Abdelfattah, A., Zounon, M., Wu, P., Pranesh, S., Tomov, S., and Dongarra, J. (2018a). The design of fast and energy-efficient linear solvers: On the potential of half-precision arithmetic and iterative refinement techniques. In Shi, Y., Fu, H., Tian, Y., Krzhizhanovskaya, V. V., Lees, M. H., Dongarra, J., and Sloot, P. M. A., editors, *Computational Science—ICCS 2018*, volume 10860 of *Lect. Notes Comput. Sci.*, pages 586–600.
- Haidar, A., Bayraktar, H., Tomov, S., Dongarra, J., and Higham, N. J. (2020). Mixed-precision solution of linear systems using accelerator-based computing. Technical Report ICL-UT-20-05, Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA.
- Haidar, A., Tomov, S., Dongarra, J., and Higham, N. J. (2018b). Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18 (Dallas, TX), pages 47:1–47:11, Piscataway, NJ, USA. IEEE Press.
- Hickmann, B. and Bradford, D. (2019). Experimental analysis of matrix multiplication functional units. In *Proc. 26th IEEE Symposium on Computer Arithmetic*, pages 116–119, Kyoto, Japan.

- IEEE (2019). *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019 (revision of IEEE Std 754-2008)*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA.
- Intel Corporation (2018). BFLOAT16—hardware numerics definition. Online: <https://software.intel.com/en-us/download/bfloat16-hardware-numeric-definition>. White paper. Document number 338302-001US. Accessed: July 15, 2020.
- Intel Corporation (2020). Intel architecture instruction set extensions and future features programming reference. Online: <https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf>. Accessed: July 15, 2020.
- Jia, Z., Maggioni, M., Smith, J., and Scarpazza, D. P. (2018a). Dissecting the NVidia Turing T4 GPU via microbenchmarking. arXiv:1903.07486 [cs.DC].
- Jia, Z., Maggioni, M., Staiger, B., and Scarpazza, D. P. (2018b). Dissecting the NVIDIA Volta GPU architecture via microbenchmarking. arXiv:1804.06826 [cs.DC].
- Kaul, H., Anders, M., Mathew, S., Kim, S., and Krishnamurthy, R. (2019). Optimized fused floating-point many-term dot-product hardware for machine learning accelerators. In *Proc. 26th IEEE Symposium on Computer Arithmetic*, pages 84–87.
- Kim, D. and Kim, L. (2009). A floating-point unit for 4D vector inner product with reduced latency. *IEEE Trans. Comput.*, 58(7):890–901.
- Markidis, S., Chien, S. W. D., Laure, E., Peng, I. B., and Vetter, J. S. (2018). NVIDIA tensor core programmability, performance & precision. In *Proc. 32rd IEEE International Parallel and Distributed Processing Symposium Workshops*, pages 522–531.
- Mukunoki, D., Ozaki, K., Ogita, T., and Imamura, T. (2020). DGEMM using tensor cores, and its accurate and reproducible versions. In Sadayappan, P., Chamberlain, B. L., Juckeland, G., and Ltaief, H., editors, *Proc. of ISC High Performance: International Conference on High Performance Computing*, pages 230–248. Springer International Publishing.
- Muller, J.-M., Brunie, N., de Dinechin, F., Jeannerod, C.-P., Joldes, M., Lefèvre, V., Melquiond, G., Revol, N., and Torres, S. (2018). *Handbook of Floating-Point Arithmetic*. Birkhäuser, Cham, Switzerland, 2nd edition.
- NVIDIA (2017). NVIDIA Tesla V100 GPU architecture. Online: <https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf>. NVIDIA whitepaper WP-08608-001_v1.1. Accessed: July 15, 2020.
- NVIDIA (2018). NVIDIA Turing GPU architecture. Online: <https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf>. NVIDIA whitepaper WP-09183-001_v01. Accessed: July 15, 2020.
- NVIDIA (2020a). Multiply-and-Accumulate Instruction: mma. Online: <https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#warp-level-matrix-instructions-mma>. Accessed: April 15, 2020.
- NVIDIA (2020b). NVIDIA A100 tensor core GPU architecture. Online:

- <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf>. NVIDIA whitepaper v1.0. Accessed: July 15, 2020.
- NVIDIA (2020c). NVIDIA CUDA Math API. Online: <https://docs.nvidia.com/cuda/cuda-math-api/index.html>. Accessed: April 15, 2020.
- Santoro, M. R., Bewick, G., and Horowitz, M. A. (1989). Rounding algorithms for IEEE multipliers. In *Proc. 9th IEEE Symposium on Computer Arithmetic*, pages 176–183.
- Sohn, J. and Swartzlander, E. E. (2016). A fused floating-point four-term dot product unit. *IEEE Trans. Circuits Syst. I, Reg. Papers*, 63(3):370–378.
- Tao, Y., Deyuan, G., Xiaoya, F., and Nurmi, J. (2013). Correctly rounded architectures for floating-point multi-operand addition and dot-product computation. In *Proc. 24th International Conf. on Application-Specific Systems, Architectures and Processors*, pages 346–355.
- Tenca, A. F. (2009). Multi-operand floating-point addition. In *Proc. 19th IEEE Symposium on Computer Arithmetic*, pages 161–168.
- Yan, D., Wang, W., and Chu, X. (2020). Demystifying tensor cores to optimize half-precision matrix multiply. In *2020 IEEE International Parallel and Distributed Processing Symposium*, pages 634–643, New Orleans, LA, USA.