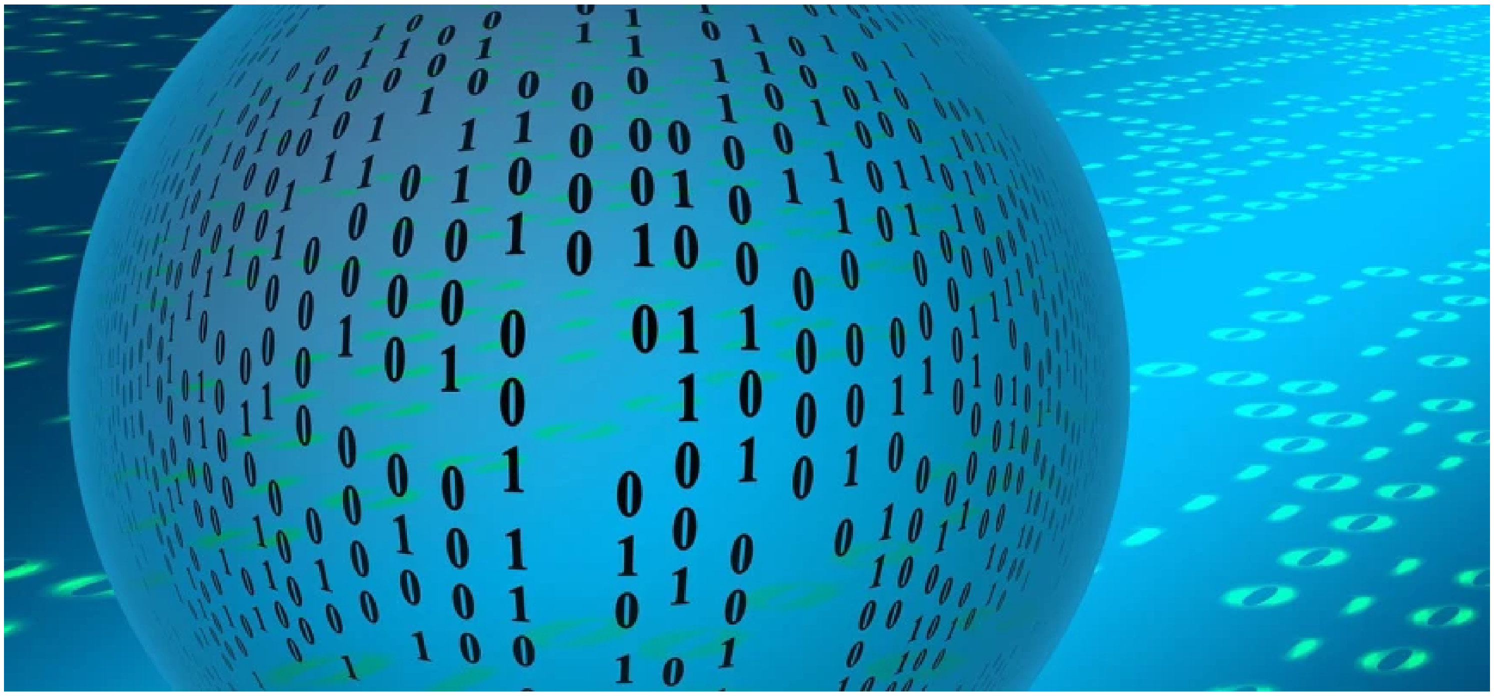
Nick Higham

Half Precision Arithmetic: fp16 Versus bfloat16



The **2008 revision (https://doi.org/10.1109/IEEESTD.2008.4610935)** of the IEEE Standard for Floating-Point Arithmetic introduced a half precision 16-bit floating point format, known as fp16, as a storage format. Various manufacturers have adopted fp16 for computation, using the obvious extension of the rules for the fp32 (single precision) and fp64 (double precision) formats. For example, fp16 is supported by the NVIDIA P100 and V100 GPUs and the AMD Radeon Instinct MI25 GPU, as well as the **A64FX Arm processor (https://www.top500.org/news/fujitsu-reveals-details-of-processor-that-will-power-post-k-supercomputer)** that will power the Fujitsu Post-K exascale computer.

Bfloat16

Fp16 has the drawback for scientific computing of having a limited range, its largest positive number being 6.55×10^4. This has led to the development of an alternative 16-bit format that trades precision for range. The **bfloat16 format (https://en.wikipedia.org/wiki/Bfloat16_floating-point_format)** is used by Google in its tensor processing units. Intel, which plans to support bfloat16 in its forthcoming Nervana Neural Network Processor, has recently (November 2018) published a **white paper (https://software.intel.com/en-us/download/bfloat16-hardware-numerics-definition)** that gives a precise definition of the format.

The allocation of bits to the exponent and significand for bfloat16, fp16, and fp32 is shown in this table, where the implicit leading bit of a normalized number is counted in the significand.

Format	Significand	Exponent
bfloat16	8 bits	8 bits
fp16	11 bits	5 bits
fp32	24 bits	8 bits

Bfloat16 has three fewer bits in the significand than fp16, but three more in the exponent. And it has the same exponent size as fp32. Consequently, converting from fp32 to bfloat16 is easy: the exponent is kept the same and the significand is rounded or truncated from 24 bits to 8; hence overflow and underflow are not possible in the conversion.

On the other hand, when we convert from fp32 to the much narrower fp16 format overflow and underflow can readily happen, necessitating the development of techniques for rescaling before conversion—see the recent EPrint **Squeezing a Matrix Into Half Precision, with an Application to Solving Linear Systems (http://eprints.maths.manchester.ac.uk/2678/)** by me and Sri Pranesh.

The drawback of bfloat16 is its lesser precision: essentially 3 significant decimal digits versus 4 for fp16. The next table shows the unit roundoff u, smallest positive (subnormal) number xmins, smallest normalized positive number xmin, and largest finite number xmax for the three formats.

	u	xmins	xmin	xmax
bfloat16	3.91e-03	(*)	1.18e-38	3.39e+38
fp16	4.88e-04	5.96e-08	6.10e-05	6.55e+04
fp32	5.96e-08	1.40e-45	1.18e-38	3.40e+38

(*) Unlike the fp16 format, Intel's bfloat16 does not support subnormal numbers. If subnormal numbers were supported in the same way as in IEEE arithmetic, xmins would be 9.18e-41.

The values in this table (and those for fp64 and fp128) are generated by the MATLAB function `float_params` that I have made available on **GitHub (https://github.com/higham/float_params)** and at **MathWorks File Exchange (https://uk.mathworks.com/matlabcentral/fileexchange/69566-float_params)**.

Harmonic Series

An interesting way to compare these different precisions is in summation of the harmonic series $1 + 1/2 + 1/3 + \cdots$. The series diverges, but when summed in the natural order in floating-point arithmetic it converges, because the partial sums grow while the addends decrease and eventually the addend is small enough that it does not change the partial sum. Here is a table showing the computed sum of the harmonic series for different precisions, along with how many terms are added before the sum becomes constant.

Arithmetic	Computed Sum	Number of terms
bfloat16	5.0625	65
fp16	7.0859	513
fp32	15.404	2097152
fp64	34.122	$2.81 \cdots \times 10^{14}$

The differences are striking! I determined the first three values in MATLAB. The fp64 value is **reported by Malone (https://www.maths.tcd.ie/pub/ims/bull71/)** based on a computation that took 24 days, and he also gives analysis to estimate the limiting sum and corresponding number of terms for fp64.

Fused Multiply-Add

The NVIDIA V100 has tensor cores that can carry out the computation D = C + A*B in one clock cycle for 4-by-4 matrices A, B, and C; this is a 4-by-4 fused multiply-add (FMA) operation. Moreover, C and D can be in fp32. The benefits that the speed and accuracy of the tensor cores can bring over plain fp16 is demonstrated in **Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers (http://dl.acm.org/citation.cfm?id=3291656.3291719)**.

Intel's bfloat16 format supports a scalar FMA d = c + a*b, where c and d are in fp32.

Conclusion

A few years ago we had just single precision and double precision arithmetic. With the introduction of fp16 and fp128 in the IEEE standard in 2008, and now bfloat16 by Google and Intel, the floating-point landscape is becoming much more interesting.
Posted on **December 3, 2018April 23, 2020** by **Nick Higham**Posted in **research**Tagged **bfloat16**, **fp16**, **IEEE_arithmetic**.

13 thoughts on "Half Precision Arithmetic: fp16 Versus bfloat16"

1. **DIMA SAYS:**
   December 3, 2018 at 10:46 pm
   numpy has had float16 for many years. (Of course, Matlab ivory towers are high, and one might have overlooked it :-))