HAL
open science

Shadow computation with BFloat16 to compute numerical accuracy

David Defour, Pablo de Oliveira Castro, Matei Istoan, Eric Petit

To cite this version:

David Defour, Pablo de Oliveira Castro, Matei Istoan, Eric Petit. Shadow computation with BFloat16 to compute numerical accuracy. 2021. hal-03159965v1

HAL Id: hal-03159965
https://hal.science/hal-03159965v1

Preprint submitted on 4 Mar 2021 (v1), last revised 26 May 2021 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Shadow computation with BFloat16 to compute numerical accuracy

David Defour, Pablo de Oliveira Castro, Matei Iştoan, Eric Petit

***Abstract*—In this article, we propose to exploit the new computational capability offered by the Bfloat16 representation format to perform shadow computations and compute estimations of the relative error. We demonstrate and evaluate the assumptions under which shadow computation is valid for the summation problem.**

I. Introduction

With the finite precision arithmetic available in today's processors, results are subject to numerical errors which can come from rounding errors or cancellations. It is difficult to estimate the numerical error of a computation without further investigation, or, when comparing two results, find the one which exhibits the smallest relative error.

Meanwhile, we are witnessing a generalization in the support of low-precision floating-point formats in order to address new use-cases, such as neural networks. Among them, Bfloat16 (or BF16)leads to a tremendous increase in processing power. As of 2019, this format is available on Intel AI processors (AVX-512 with BF16 extension and AMX), Google's TPU and in the ARMv8.6-A.

Vendors usually provide BF16 units with peak FLOPS 8-32x higher than Binary32. Furthermore, the Intel AMX implementation [1] can be used concurrently with the AVX FP32 units, which means that with careful software design, leveraging data prefetching and fast conversion, a shadow BF16 execution can have virtually no overhead. The BF16 FMA operation uses BF16 inputs and a Binary32 accumulator that can be saved to memory in either Binary32 or BF16. It has been demonstrated that the 32-bit accumulator is a low-overhead, critical feature to handle AI workloads [12], by avoiding absorption (swamping effect [20]).

In this article, we are interested, regarding Binary32 and Bfloat16 computation for the summation problem, in:

- Predict a result's accuracy and detect potentially faulty execution using estimators.
- Given two computed results in Binary32, determine the one with the smallest relative error.

II. Bibliography and related works

The study of error and in particular of round-off error due to the use of floating-point arithmetic is an important field of numerical analysis. The error analysis can be done *a priori* using a mathematical proof or *a posteriori*. In the latter case, the analysis is done automatically for a given set of input data, taking into account uncertainty and rounding error along with the main computation. All methods belonging to this class of analysis involve a computational and memory overhead, slowing the observed computation.

Among the available methods, automatic differentiation can evaluate numerically the derivative of a sequence of arithmetic operations and functions in a computer program. Automatic differentiation has been implemented through libraries for many programming languages [2]. However, tools based on automatic differentiation face drawbacks in terms of difficulty of implementation when applied to complex programs, and in terms of computational and memory overhead.

Stochastic arithmetic estimates the error distribution with a stochastic simulation of round-off effects, either applying random rounding at each step of the chain of computation [19] or through Monte-Carlo arithmetic [16]. Stochastic arithmetic has been successfully applied to localize numerical errors in post-mortem numerical analysis but is impractical for online error estimation.

Interval arithmetic is used to model uncertainty on input data, along with round-off error, in order to guarantee the results [13]. Interval arithmetic can provide a contained memory and computational overhead, but suffers from the dependency problem leading to large uninformative interval.

Error analysis on a sequence of operations can be done by devoting extra memory and computational resources to recompute with extra-accuracy, either by using dedicated arithmetic (ex: MPFR[5], double-double[8]), or compensation [6].

For each of the previously mentioned methods, beside the fact that the overhead in terms of resources can be large, none of them is able to provide numerical information, for a cost that is lower than the original computation.

The application range of BF16 has been limited to specific fields, such as AI, due to the limited accuracy of the format. Recently, Henry et al.[7] leveraged BF16 to speed-up computations by splitting Binary32 operations into sub-operations and provide an accuracy analysis. They also propose to use iterative refinement with BF16 as a pre-conditioner to accelerate convergence in a LU solver.

The concept of replicating a sequence of computations, also called shadow replication, has been used in the context of fault tolerance for green cloud computing [15]. The replication can be limited to intermediate data in the context of shadow memory, which is used, for example, for numerical debugging purposes [14].

III. Background

Our objective consists of using the extra computational capabilities offered by BF16, available in today's processors,

to estimate the numerical error of a sequence of operations. In this section we present characteristics of the BF16 format and how errors can be bounded.

A. Bfloat16

Bfloat16 corresponds to a 16-bit shortened version of the IEEE-754 32-bit single-precision floating-point format (i.e. Binary32). The 16 least-significant bits of Binary32 correspond solely to mantissa bits, which allows for fast conversion to and from Bfloat16. The description of the bit fields of both formats, Bfloat16 and Binary32, are given in Figure 1.

As we can observe, the difference between both formats lies in the fraction field, which is 7-bits long, as opposed to 23-bits in Binary32. This means that the representation range of both formats is almost equivalent.

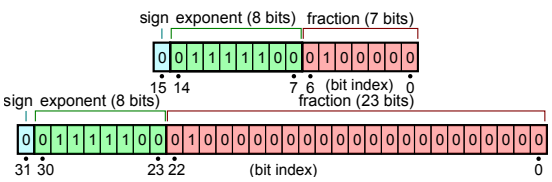

Fig. 1: The Bfloat16 and Binary32 format

B. Rounding error analysis

We denote by $\triangle_f(x)$ and $\bigcirc_f(x)$ the rounding away from 0 and the rounding to nearest of $x \in \mathbb{R}$, respectively, in the format f. Among the considered formats, in this article we use Binary64, Binary32 and Bfloat16, which are represented in the rest of the paper by $b64$, $b32$, $bf16$.

In the interval of representable numbers, the output of floating-point operations is impacted by rounding, absorption and catastrophic cancellation errors. In the case of the sum $S_n = \sum_{i=1}^{n} x_i$ of a vector x_i of floating-point numbers, the real absolute error depends on the order of the partial summations. However, there exist metrics which are independent of the order. In rounding to nearest, the approximation error of the true sum S_n is:

$$\widehat{S_n} = \bigcirc_f(S_n) = S_n(1+\epsilon)$$

with $|\epsilon| \leq u_f$ and u_f the machine precision ($u_f = 2^{-53}; 2^{-24}; 2^{-8}$ for $b64$, $b32$ and $bf16$ respectively) can be bounded [9], independently of the order, for all summation methods without overflow as follow:

$$|\widehat{S_n} - S_n| \leq (n-1)u_f \sum_{i=1}^{n} |x_i| + O(u_f^2) \tag{1}$$

Rump introduced the unit in the first place concept (*ufp*) to perform elegant error analysis, such as in [17]. He proposed algorithms to compute estimations of the error for the summation and dot product problems. It consists in computing an approximation of the exact sum S_n, as follows:

$\widehat{s_1} = x_1; \widehat{T_1} = |x_1|$
for $\{k = 2; k \leq n; k++\}$ **do**
  $\widehat{s_k} = \bigcirc_f(\widehat{s_{k-1}} + x_k)$
  $\widehat{T_k} = \bigcirc_f(\widehat{T_{k-1}} + |x_k|)$
end for

The computed sum $\widehat{T_n}$ corresponds to the accumulation of the absolute value performed at each step of the algorithm. The only requirement is that both $\widehat{s_k}$ and $\widehat{T_k}$ use the same format. It can be used to derive a computable absolute error bound:

$$\begin{aligned} |\widehat{s_n} - \sum_{i=1}^{n} x_i| &\leq (n-1) \cdot u_f \cdot ufp(\widehat{T_n}) \\ &\leq (n-1) \cdot u_f \cdot \widehat{T_n} \end{aligned}$$

This bound is known to be a pessimistic upper bound, especially when dealing with random data. Except for very particular cases, the real bound is lower [9]. In [10], using probabilistic assumptions about the rounding error, they proposed a relaxed bound which grows as $\sqrt{n-1}u_f$ instead of $(n-1)u_f$ for the worst case. In [11], the authors extended their work by proposing a bound depending on the data range distribution independently of n.

IV. Proposed strategies

Relative error, or approximation error is a useful metric to estimate the number of correct digits in a result. The question which we address is: how can we provide a computable relative error for a small overhead?

A. Reference relative error

In order to establish a fair comparison, we have to set a reference value, that we name E_{ref}. Computing the relative error would require, in the case of the summation problem, computing the reference value using a long accumulator in order to avoid rounding errors, such as the one proposed in [4]. However, for problems of reasonable size ($N < 2^{20}$) and condition number ($C < 2^{30}$), computing the reference value using Binary64 can be considered enough:

$$E_{ref} = \frac{|\widehat{s_{b32}} - \widehat{s_{b64}}|}{|\widehat{s_{b64}}|} \tag{2}$$

with $\widehat{s_f}$ corresponding to the computed sum in format f with rounding to nearest mode.

B. Mixed bound

The problem with equation 2 is that the real value ($\widehat{s_{b64}}$) has to be computed very accurately, requiring extra time and memory compared to the main computation. The relative error corresponds to the absolute value divided by the real value. We propose to use a computable bound for the absolute value as described in Section III-B.

Rump's estimator ($\widehat{T_n}$) is only valid when both $\widehat{T_n}$ and $\widehat{s_n}$ are computed with identical floating-point formats and rounding modes. It is straightforward to use BF16 to compute $\widehat{T_n}$, and only requires few modifications. The proof of Rump's estimator is based on the monotonicity of rounding ([17], equation (3.9)) which states that if $|x| \leq X$ and $|y| \leq Y$ then $|\bigcirc_f (x+y)| \leq \bigcirc_f(X+Y)$ for $x, y, X, Y \in \mathbb{R}$.

We propose to alter Rump's algorithm, to replace the computation of $\widehat{T_n}$ by $\widehat{B_n}$ computed as follow:

$$B_k = \widehat{B_{(k-1)}} + \triangle_{bf16}(|p_k|); \widehat{B_k} = \bigcirc_{b32}(B_k); \quad (3)$$

This corresponds to loading BF16 approximations of $|p_k|$, using rounding away from zero, accumulated in a Binary32 accumulator using rounding to nearest. This corresponds to what is usually available in BF16 hardware.

Theorem IV.1. *If $|x| \leq X$ and $|y| \leq Y$, then $|\bigcirc_{b32}(x+y)| \leq \bigcirc_{b32}(\triangle_{bf16}(X) + \triangle_{bf16}(Y))$ for $x, y, X, Y \in \mathbb{R}$.*

Proof. Rounding away from zero lets us have $|x| \leq \triangle_{bf16}(|x|)$.

Upscaling does not involve rounding error, therefore $\bigcirc_{b32}(\triangle_{bf16}(x)) = \triangle_{bf16}(x)$.

Thanks to the monotonicity of rounding, we get $|\bigcirc_{b32}(x+y)| \leq \bigcirc_{b32}(\triangle_{bf16}(X) + \triangle_{bf16}(Y))$. □

Note that it is possible to consider performing everything in BF16 with rounding away from zero (including the accumulation), based on the fact that double rounding issues do not occur with directed rounding. This leads to $|\bigcirc_{b32}(x+y)| \leq \bigcirc_{b32}(\triangle_{bf16}(X) + \triangle_{bf16}(Y)) \leq \triangle_{bf16}(\triangle_{bf16}(X) + \triangle_{bf16}(Y))$. However, an inevitable drift from the real value happens, as it is usually observed with interval arithmetic, but emphasized here by the poor accuracy provided by BF16.

We propose the following mixed bound E_{mixed}, based on Rump's estimator for the absolute value, using BF16 input values:

$$E_{mixed} = (n-1) \cdot u_{b32} \cdot \frac{\widehat{B_{bf16}}}{|\widehat{s_{b64}}|} \quad (4)$$

with $\widehat{s_{b64}}$ computed in rounding to nearest and $\widehat{B_{bf16}}$ computed as in equation 3.

C. Fully computed bound

Equation 4 cannot be considered a fully computable bound as it requires $|\widehat{s_{b64}}|$. We have to bound $|\widehat{s_{b64}}|^{-1}$ using the computed quantities ($\widehat{s_{b32}}$ and $\widehat{B_{bf16}}$). One may use the fact that $|S| \geq |\widehat{s_{b32}}| - (n-1)u_{b32}\widehat{B_{bf16}}$ to propose the following *computed* estimator:

$$E_{comp} = (n-1) \cdot u_{b32} \cdot \frac{\widehat{B_{bf16}}}{|\widehat{s_{b32}}| - (n-1)u_{32}\widehat{B_{bf16}}} \quad (5)$$

Note that this estimator is valid only if

$$|\widehat{s_{b32}}| > (n-1)u_{b32}\widehat{B_{bf16}}$$

D. Approximate estimator number 1

Equation 5 is a valid upper bound corresponding to the worst case scenario. As for other estimators we can consider a more plausible one. In particular, we can use the following overestimation $|S| \geq |\widehat{s_{b32}}|$ to propose the following E_{approx} estimator:

$$E_{approx} = (n-1) \cdot u_{b32} \cdot \frac{\widehat{B_{bf16}}}{|\widehat{s_{b32}}|} \quad (6)$$

V. Results

This section illustrates through some experiments how BF16 shadow computation can add numerical information on a main computation done in Binary32. To do so, we compare the four estimators on the summation problem. We have executed the naive summation for 5000 vectors of 400 floating-point numbers with condition numbers ranging from 2^6 and 2^{50}. We use a random generator inspired by the GenSum function in [18]. Results are reported in Figure 2, which represents the relative error in log scale according to the condition number.

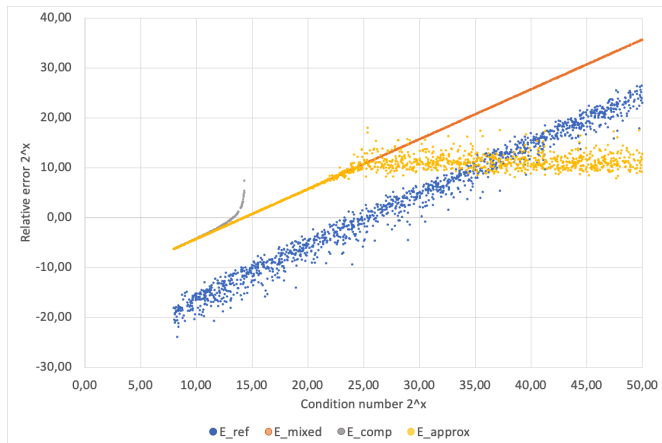

Fig. 2: Relative error for random vectors of size 400 for the four estimators according to the condition number.

We observe that all four estimators are above (overestimation) the true error (in blue) for problems of condition number less than 2^{24}. For problems of condition numbers greater than this bound, we observe that the mixed estimator follows through a straight line the real error. The true computed bound (in grey) presents a singularity for problems of condition number greater than 2^{14}. This is due to the validity condition of the estimator ($|\widehat{s_{b32}}| > (n-1)u_{b32}\widehat{B_{bf16}}$). Finally, the approximate estimator (yellow) behaves as the mixed estimator for problems of condition numbers less than 2^{24}. For problems of larger condition numbers, the estimator saturates at an error 2^{10}. Even though this estimator is wrong in this case, it still indicates that all bits are lost in the result, which is an important information.

A. Comparing numerical quality

In this section, we illustrate how the approximated estimator E_{approx} can be used to compare the numerical quality of two computed results. We have generated 10^5 random vectors of size 400. We counted the number of times the approximated estimator makes a correct prediction to determine which of two results presents the smallest relative error. Results are reported in Figure 3. It represents the percentage of correct predictions according to the condition number of both vectors, indicated in xy-coordinate.

We observe that the estimator correctly predicts which result is the most accurate as long as the difference is significant

	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49
7	55	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
9	66	58	80	94	98	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
11	100	85	56	83	95	97	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
13	100	97	88	49	81	93	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
15	100	100	97	86	60	84	95	97	98	100	100	100	100	100	100	100	100	100	100	100	100	100
17	100	100	98	98	90	59	75	96	100	100	100	100	100	100	100	100	100	100	100	100	100	100
19	100	100	100	100	95	87	63	77	93	100	100	98	100	100	100	100	100	100	100	100	100	100
21	100	100	100	100	98	98	73	56	81	96	100	97	100	100	100	100	100	100	100	100	100	100
23	100	100	100	100	98	100	91	79	60	77	77	91	90	89	81	94	88	91	87	93	89	95
25	100	100	100	100	100	100	96	98	74	23	50	45	51	45	46	58	62	55	56	58	46	45
27	100	100	100	100	100	100	98	100	87	46	19	30	43	43	59	52	54	46	43	39	48	52
29	100	100	100	100	100	100	100	100	92	46	31	1	27	35	38	52	51	49	50	55	58	55
31	100	100	100	100	100	100	100	100	96	51	48	28	5	30	36	55	58	43	46	54	45	42
33	100	100	100	100	100	100	100	100	95	64	43	45	33	3	27	40	40	55	42	46	44	50
35	100	100	100	100	100	100	100	100	94	51	54	61	46	36	0	27	37	47	40	55	59	43
37	100	100	100	100	100	100	100	100	83	48	58	43	45	46	29	14	20	43	53	52	51	54
39	100	100	100	100	100	100	100	100	86	51	42	41	54	44	43	28	3	22	42	41	48	59
41	100	100	100	100	100	100	100	100	88	58	44	53	41	48	51	52	28	6	22	43	58	53
43	100	100	100	100	100	100	100	100	88	68	45	51	54	78	59	47	51	29	5	28	43	45
45	100	100	100	100	100	100	100	100	84	70	36	47	52	46	50	60	46	44	28	8	28	51
47	100	100	100	100	100	100	100	100	89	48	44	50	55	42	52	49	49	51	47	32	7	30
49	100	100	100	100	100	100	100	100	86	52	55	52	38	56	54	46	52	52	48	45	22	3

Fig. 3: Percentage of time that BF16 shadow computation make a correct prediction when comparing the relative error of two summation problems with condition number varying between 2^6 and 2^{50}.

(diagonal) and the condition number for at least one vector is less than 2^{23} (lower right square) . The difference has to be significant between the two vectors for problem of condition number less than 2^{23} as this corresponds to a sharp estimator (yellow line in fig. 2) and a reference relative error that may be better than expected (wide set of blue dots for a given condition number). For the lower right square, this corresponds to the place where the estimator predict that all bits of accuracy are lost without being able to distinguish the "less worse" among the two.

VI. Conclusion

In this article we proposed a computable estimator for the relative error in the case of the summation problem. This illustrates how BF16 computation resources can be used to add information on numerical quality to a Binary32 computation. Through tests, we have shown that the estimator follows the optimal bound for problems of condition number less than the number of bits available in the mantissa. When the condition number is larger, the estimator remains useful by indicating that all bits of information are lost. Finally, we have shown how the estimator successfully predicts which of two computed results is the most accurate, as long as the difference is significant. As future work, we plan to investigate how the estimator can be used to help remove the rise of computer zero during pivoting in LU factorisation [3].

References

[1] Intel Architecture Instruction Set Extensions and Future Features Programming Reference - 319433-042 December 2020.

[2] Community portal for automatic differentiation, 2021.

[3] J. M. Chesneaux. The equality relations in scientific computing. *Num. Algo.*, 7:129–143, 1994.

[4] S. Collange, D. Defour, S. Graillat, and R. Iakymchuk. Numerical reproducibility for the parallel reduction on multi- and many-core architectures. *Parallel Computing*, 49:83–97, 2015.

[5] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding, 2005.

[6] S. Graillat, P. Langlois, and N. Louvet. Algorithms for accurate, validated and fast polynomial evaluation. October 01 2009.

[7] G. Henry, Ping Tak Peter Tang, and A. Heinecke. Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pages 69–76. IEEE, 2019.

[8] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating point arithmetic. In *Proc. 15th IEEE Symposium on Computer Arithmetic*, pages 155–162. IEEE Computer Society Press, Los Alamitos, CA, USA, 2001.

[9] N. Higham. The accuracy of floating point summation. *SIAM J. Sci. Comput.*, 14:783–799, 1993.

[10] N. J. Higham and T. Mary. A new approach to probabilistic rounding error analysis. *SIAM J. Sci. Comput*, 41(5):A2815–A2835, 2019.

[11] N. J. Higham and T. Mary. Sharper probabilistic backward error analysis for basic linear algebra kernels with random data. *SIAM Journal on Scientific Computing*, 42(5):A3427–A3446, 2020.

[12] D. Kalamkar et al. A Study of BFLOAT16 for Deep Learning Training, 2019.

[13] U. W. Kulisch. Mathematics and speed for interval arithmetic: A complement to ieee 1788. *ACM Trans. Math. Softw*, 45(1):5:1–5:22, 2019.

[14] M. O. Lam and B. L. Rountree. Floating-point shadow value analysis. In *ESPT@SC*, pages 18–25. IEEE, 2016.

[15] B. N. Mills, T. Znati, and R. G. Melhem. Shadow computing: An energy-aware fault tolerant computing model. In *ICNC*, pages 73–77. IEEE, 2014.

[16] D. Stott Parker. Monte Carlo arithmetic: exploiting randomness in floating-point arithmetic. Technical Report CSD 970002, Department of Computer Science, University of California, Los Angeles, Los Angeles, CA, USA, 1997.

[17] Siegfried M. Rump. Error estimation of floating-point summation and dot product. *BIT Numerical Mathematics*, 52(1):201–220, March 2012.

[18] S.M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. http://www.ti3.tuhh.de/rump/.

[19] J. Vignes. A stochastic arithmetic for reliable scientific computation. *Math. Comp. Simul.*, 35:233–261, 1993.

[20] N. Wang, J. Choi, D. Brand, C. Y. Chen, and K. Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. *NeurIPS*, 2018.