Numerical validation of half precision simulations

Fabienne Jézéquel, Sara Sadat Hoseininasab, Thibault Hilaire

Numerical validation of half precision simulations

Fabienne Jézéquel[1,2], Sara sadat Hoseininasab[1], and Thibault Hilaire[1]

[1] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, 75005 Paris, France
[2] Université Panthéon-Assas, 12 place du Panthéon, 75231 Paris CEDEX 05, France
{Fabienne.Jezequel, Sara.Hoseininasab, Thibault.Hilaire}@lip6.fr

Abstract. In this article, we show how to control the numerical quality of half precision computations using stochastic arithmetic. The CADNA library that is used to estimate rounding errors and detect numerical instabilities in floating-point codes has been improved and extended to support half precision. A performance gain up to a factor 20 has been observed w.r.t. accuracy estimation in arbitrary precision. Furthermore we present how to generate codes mixing half, single, and double precision with a requested accuracy on results providing a user-defined numerical quality of the code. Control of robustness and floating-point auto-tuning taking into account half precision have been successfully performed on various numerical simulations, in particular a control application.

Keywords: numerical quality, numerical validation, rounding errors, auto-tuning, discrete stochastic arithmetic, floating-point arithmetic, half precision, mixed precision,

1 Introduction

Among the different tests that must be performed to validate a code, numerical validation is crucial because rounding errors generated by its floating-point operations can invalidate its results. Indeed controlling the robustness of a simulation code includes the analysis of its numerical quality. Numerical simulations tend to be carried out in reduced precision (e.g. single or half precision instead of double precision) or in mixed precision. Since reducing the precision has an impact on performance, memory usage, and energy consumption, using half precision can be particularly advantageous [1, 5]. However, because of the low mantissa-length and the limited range of that format, the accuracy of half precision computation should be carefully controlled in order to ensure code quality. We respond to this challenging issue with the following contributions.

– We show how to estimate rounding errors and detect numerical instabilities in any half precision computation. The related software development has been integrated in the CADNA library [2].

- We have extended the PROMISE software [4] which performs floating-point auto-tuning such that it can automatically generate codes mixing half, single and double precision.
- Thanks to the new version of CADNA, the numerical quality of various simulations in half precision has been analysed, in particular, a control application and a Conjugate Gradient solver. Furthermore versions of these codes in mixed precision (including half precision) have been provided by the new version of PROMISE.

The remaining part of this paper is organized as follows. Section 2 reminds the principles of the rounding error estimation using the Discrete Stochastic Arithmetic and the floating-point auto-tuning. Section 3 shows how floating-point computation can be carried out in half precision. Sections 4 and 5 detail the extension of the accuracy estimation and auto-tuning to half precision arithmetic. Finally, Section 6 presents some numerical experiments, and conclusions are given in Section 7.

2 Control of numerical quality and floating-point auto-tuning

2.1 Principles of DSA

Discrete Stochastic Arithmetic (DSA) [9] is an automatic method for rounding error analysis based on a probabilistic approach. DSA can provide an estimation of the numerical quality of results, especially in large scale applications. DSA allows one to estimate the number of correct digits in computed results by executing the user programs several times using a random rounding mode: either rounding to $+\infty$ or to $-\infty$ with the same probability. Therefore, the computer's deterministic arithmetic is replaced by a stochastic arithmetic, where each arithmetic operation is performed N times before the next one is executed. DSA supplies us with N samples $R_1, \cdots, R_N$ of each computed result R and the number of correct digits in R is estimated using a statistical test.

2.2 Accuracy estimation by CADNA & SAM

DSA is implemented, on the one hand, in the CADNA[3] library [2] that can be used to control the accuracy of programs in single, double and/or quadruple precision, and, on the other hand, in the SAM[4] library [3] that estimates rounding errors in arbitrary precision programs. Thanks to three executions of the user program with the random rounding mode, CADNA and SAM estimate, with the probability 95%, the number of correct digits of any computed result. Their codes are based on new numerical types: the stochastic types. Each stochastic variable contains three values of the corresponding numerical type and an integer

[3] http://cadna.lip6.fr
[4] http://www-pequan.lip6.fr/~jezequel/SAM

to store its number of correct digits. Arithmetic operators, comparison operators, all the mathematical functions are overloaded for these stochastic types. Therefore the use of CADNA or SAM in a program requires only a few modifications: essentially changes in the declarations of variables and in input/output statements. CADNA and SAM can detect numerical instabilities which occur during the execution of the code. Such instabilities are usually generated by numerical noise, *i.e.* a result having no correct digits.

2.3 Floating-point auto-tuning: the PROMISE software

The PROMISE[5] tool [4], based on CADNA, aims at reducing in numerical programs the number of double precision variable declarations in favor of single precision ones. From an original program and a requested accuracy on the result, PROMISE provides a transformed program having a maximum number of variables declared with a lower precision and computing a result that satisfies the accuracy constraint. The search for a suitable type configuration is performed with a reasonable complexity thanks to the Delta Debug algorithm [10] based on a hypothesis-trial-result loop.

3 Half precision computation

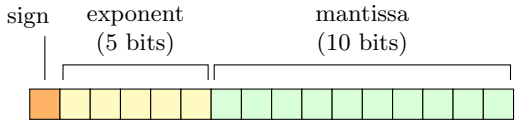


Fig. 1: *binary16* format

Half precision is referred to as *binary16* in the IEEE 754-2008 standard [6]. As shown in Figure 1, a *binary16* floating-point number consists of a sign bit, a 5-bit long exponent, and a 10-bit long mantissa. Although 10 bits are explicitly stored to represent the mantissa, its precision is actually 11 bits. Because $\log_{10}(2^{11}) \approx 3.3$, the accuracy of a *binary16* number is at most 3 decimal digits. In *binary16*, the minimum strictly positive (subnormal) value is $2^{-24} \approx 5.9610^{-8}$ and the maximum representable value is 65504. The *binary16* format is increasingly supported by hardware. It is available for instance on ARM CPUs from version 8.2, on NVIDIA Pascal and Volta GPUs, on AMD Radeon Instinct MI25 GPU. *binary16* computation can also be emulated, for instance using the FlexFloat[6] library [8] that supports multiple floating-point formats or the IEEE 754-based half-precision floating-point library developed by C. Rau[7]. In this paper, we focus on the *binary16* format, however, another 16-bit floating-point format called *bfloat16* is available, for instance on Google Cloud TPUs and ARM

[5] http://promise.lip6.fr
[6] https://github.com/oprecomp/flexfloat
[7] http://half.sourceforge.net

NEON CPUs. Compared to *binary16* numbers, *bfloat16* numbers benefit from a wider range thanks to their 8 bit-exponent, but have a lower significand precision (8 bits instead of 11).

4 Controlling the numerical quality of half precision computation

4.1 Extension of CADNA to half precision

CADNA has been extended to control the robustness of codes with half precision declarations. On processors that do not support half precision, CADNA relies on the half precision floating-point library developed by C. Rau already mentioned in Sec. 3. This half precision library has been improved to enable computation mixing of half, single and double precision. CADNA has also been extended to be used on the ARM v8.2 processor that supports half precision computation.

A new stochastic type, `half_st`, associated with emulated or native half precision has been introduced in CADNA. A `half_st` variable consists in three half precision floating-point values and an integer to store the accuracy. Relational and arithmetic operations, mathematics and intrinsic functions have been redefined for this new type. With CADNA, for performance reasons, the rounding mode is changed in an implicit way [2] as

- $a \oplus_{+\infty} b = -(-a \oplus_{-\infty} -b)$ (similarly for $\ominus$)
- $a \otimes_{+\infty} b = -(a \otimes_{-\infty} -b)$ (similarly for $\oslash$)

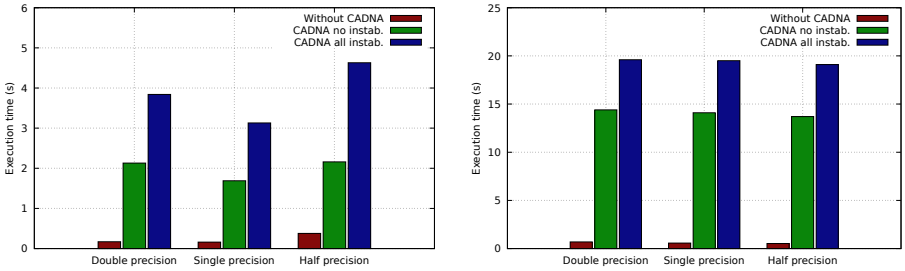
where $\oplus_{+\infty}$ and $\otimes_{+\infty}$ (resp. $\oplus_{-\infty}$ and $\otimes_{-\infty}$) are the floating-point operations rounded towards $+\infty$ (resp. $-\infty$). The results of each rounding mode can be obtained from computation made in the other rounding mode. With native and emulated half precision, operations involving `half_st` variables are performed with the random rounding mode thanks to the aforementioned arithmetic properties. The rounding mode is set once to rounding to $+\infty$ in a CADNA initialization function, such that operations involving stochastic variables based on native types can be performed with the random rounding mode, without any explicit change of the rounding mode. In the library developed by C. Rau, half precision operations can be performed with any rounding mode of the IEEE standard. The rounding mode, which is by default rounding to nearest, has been set to rounding to $+\infty$ thanks to a parameter change.

Furthermore, because of the limited range of half precision numbers, the detection of new kinds of instabilities can been enabled: overflow and underflow caused by half precision computation. Codes including half precision can be controlled by this new CADNA version using clang++ or g++ on processors supporting half precision or thanks to emulated half precision.

4.2 Performance tests

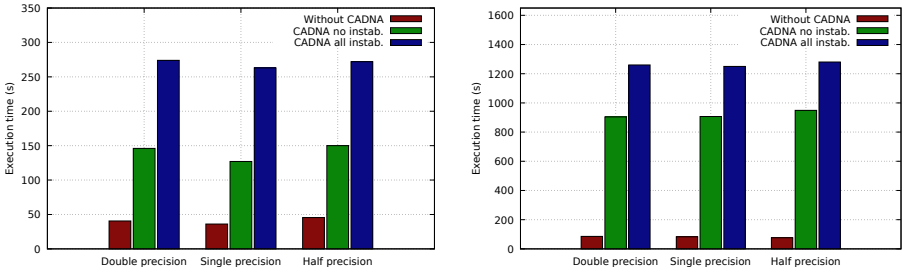
Performance tests have been carried out using g++ 7.4.0 on a 2.5 GHz Intel Core i7 processor and using g++ 7.5.0 on a 2.26 GHz ARM v8.2 processor, both

with 4 MB cache memory. The latter is used in an Nvidia Jetson AGX Xavier module designed for embedded computation. Figures 2 and 3 present execution times for matrix multiplication on Intel Core i7 processor where half precision is emulated and on ARM v8.2 processor using native half precision. With CADNA computation has been carried out with no instability detection, and also with the detection of all kinds of numerical instabilities. Our aim here is to show the feasibility of numerical validation in half precision and to compare the cost of CADNA in half, single and double precision. The matrix multiplication is performed on one core using a non-optimized algorithm based on 3 nested loops.



(a) on Intel Core i7 processor (b) on ARM v8.2 processor

Fig. 2: Performance of the multiplication of matrices of size 500



(a) on Intel Core i7 processor (b) on ARM v8.2 processor

Fig. 3: Performance of the multiplication of matrices of size 2000

In the CADNA codes, a matrix is a stochastic array. The matrix values are accessed from a structure that implies indirect memory access. With CADNA, accessing matrix values is expensive, especially for small problems. Therefore as the matrix size increases, the cost of CADNA w.r.t. classic computation decreases. The higher execution times and the higher cost of CADNA on ARM v8.2 processor than on Intel Core i7 processor can be explained by the higher cost of memory access. One can observe that the cost of CADNA is not particularly higher in half precision than in single or double precision. The cost of CADNA in half precision w.r.t single or double precision is even significantly lower when small matrices are multiplied on Intel Core i7 processor. In that case, because of

half precision emulation, the ratio computation/memory access is more favorable to CADNA in half precision than in other precisions.

The detection of all kinds of numerical instabilities mainly requires tests and accuracy estimation, no extra access to matrix values. Its cost w.r.t. no instability detection is about 2 on Intel Core i7 processor and 1.4 on ARM v8.2 processor. Furthermore, it has been observed that the cost of underflow/overflow detection in half precision is a factor between 1.3 and 1.6 on Intel Core i7 processor and approximately 1.2 on ARM v8.2 processor.

The performance of CADNA and SAM for the numerical validation of half precision codes have been compared on Intel Core i7 processor where CADNA uses emulated half precision. For different matrix sizes, Figure 4 presents the execution times of matrix multiplication computed with CADNA in half precision and with SAM using 11-bit mantissa length variables. A log scale is used for the y-axis to improve the readability of the results obtained. Two instability detection levels have been chosen: no instability detection and the detection of all kinds of numerical instabilities.

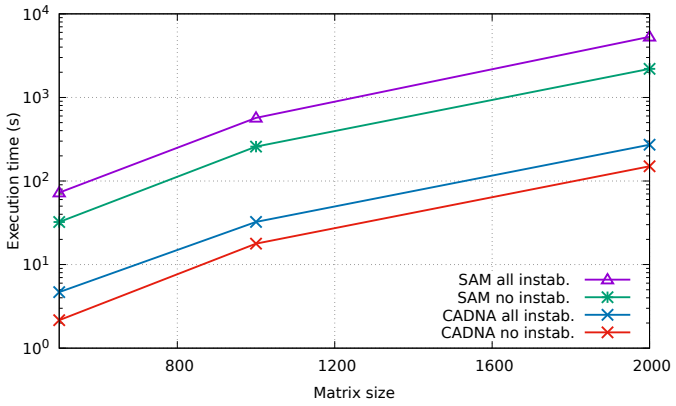


Fig. 4: Execution time of matrix multiplications in half precision with CADNA and SAM

One can observe that the cost of SAM w.r.t. CADNA is a factor between 13 and 20. Both with CADNA and SAM the detection of all instabilities costs about 2 w.r.t. no instability detection. As a remark, CADNA has an extra feature compared to SAM, it enables one to detect underflows and overflows in operations involving half precision variables.

5 Floating-point auto-tuning using half precision

A new version of PROMISE has been developed in order to generate mixed precision codes including half precision. This new version also benefits from various

improvements. It is more user-friendly and has better performance. Indeed the search for a valid type configuration can be accelerated thanks to information provided by the user, such as variables that should have the same type.

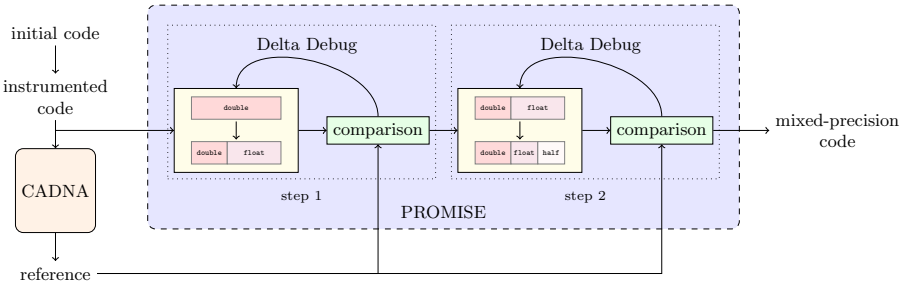


Fig. 5: PROMISE dataflow

PROMISE is based on the Delta-Debug algorithm [10], initially proposed for debugging programs using a scientific approach of hypothesis-trial-result loop. The approach is here used to test the numerical quality of the code with various precisions.

From a C/C++ code having a set of variables to be considered (with simply changing their type to a generic PROMISE-adapted type), PROMISE uses Discrete Stochastic Arithmetic (with CADNA) to determine a reference result. This reference is then used to verify the numerical quality of subsequent results obtained with several mixed-precision codes. The search for a valid type configuration relies on the Delta-Debug algorithm, as shown in Figure 5.

If PROMISE is run in half, single and double precision, the Delta Debug algorithm is executed twice. First, PROMISE determines which variables should stay in double and which ones can be relaxed to single precision. Then, from the single precision variables, PROMISE determines which ones can be relaxed to half precision.

6 Numerical experiments

The numerical experiments described in this section have been carried out using g++ 7.4.0 on a 2.5 GHz Intel Core i7 processor with 4 MB cache memory where half precision is emulated.

6.1 Determinant of Hilbert's matrix

The determinant of Hilbert's matrix of size n defined by $H_{i,j} = \frac{1}{i+j-1}$ for $i = 1, \cdots, n, j = 1, \cdots, n$ is computed using Gaussian elimination without pivoting. The determinant D is the product of the computed pivots: $D = \prod_{i=1}^{n} p_i$. Table 1 presents the results obtained with and without CADNA in double and half

precision for Hilbert's matrix of size 3. The digits in common with the exact results are displayed in bold.

	without CADNA		with CADNA	
	double	half	double	half
p_1	**1.00000000000000**e+00	**1.000000e**+00	**1.00000000000000**e+00	**1.00**e+00
p_2	**8.33333333333333**e-02	**8.3**25195e-02	**8.33333333333333**e-02	**8.3**e-02
p_3	**5.55555555555553**e-03	**5**.310059e-03	**5.555555555555**e-03	**5**.e-03
D	**4.62962962962960**e-04	**4**.420280e-04	**4.6296296296296**e-04	**4**.e-04

Table 1: Determinant of Hilbert's matrix of size 3 with and without CADNA in half and double precision

Without CADNA, results computed in half precision are casted to single precision to be printed. CADNA displays only the digits not affected by rounding errors: 14 or 15 digits in double precision and 1 to 3 digits in half precision. No numerical instability, nor underflow, nor overflow is detected. The digits displayed by CADNA in double and half precision are those of the exact results.

6.2 Control application

This example is a multi-input, multi-output controller from the automotive context. It is part of an active controller of vehicle longitudinal oscillations [7]. It is a linear State-Space system of the form

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases}$$

where $u(k) \in \mathbb{R}^5$ is the vector of inputs at time k, $y(k) \in \mathbb{R}^7$ the vector of outputs at time k and $x(k) \in \mathbb{R}^4$ the state-vector containing the internal state of the controller. A, B, C and D are small dense matrices with double precision coefficients (they define the behavior of the controller).

The associated code is executed for 100 iterations. At each iteration, 7 elements of an array are computed. When the computation is completely carried out in half precision, the number of correct digits estimated by CADNA varies from 0 to 3 and its mean value is about 2.0. Table 2 presents the results provided by PROMISE with three precisions (half, single, double): it shows the number of requested digits, the number of executions performed, the number of variables of each precision in the code provided by PROMISE, the execution time of PROMISE. A mixed precision type configuration that includes half precision is found if 1 or 2 correct digits are requested in all the results. If 3 correct digits are requested, a single precision version of the code is provided. Then, the number of double precision variables increases, as the requested accuracy increases.

# req. digits	# exec	# half-# single-# double	time (s)
1-2	58	6-12-0	58.16
3	52	0-18-0	51.47
4	55	0-15-3	47.53
5	62	0-11-7	50.92
6	67	0-9-9	53.76
7	66	0-7-11	50.89
8	63	0-4-14	47.36
9-11	52	0-1-17	38.10

Table 2: PROMISE results for the signal processing application

6.3 Conjugate Gradient code

In this experiment, the sequential version of a CG (Conjugate Gradient) code from the Seoul National University NPB (NAS Parallel Benchmarks) suite[8] is analyzed using PROMISE. The code solves a linear system with a matrix of size 7,000 with 8 non-zero values per row by performing 15 CG iterations. Table 3 presents the results provided by PROMISE, the number of requested digits on the solution vector varying from 1 to 12. Indeed, because of rounding errors, the maximal accuracy is 12 correct digits. Whatever the requested accuracy from 1 to 12 correct digits, PROMISE always provides in less than 5 minutes a mixed-precision configuration that includes half precision variables. Out of the 3^{25} (more than 847 billions) possible different configurations, PROMISE considers in the worst case only 100 possibilities.

# req. digits	# exec	# half-# single-# double	time (s)
1	44	19-6-0	212.71
2	55	18-7-0	235.07
3	53	17-8-0	241.90
4	69	14-11-0	209.08
5	67	12-13-0	197.04
6-7	74	12-13-0	204.96
8	100	10-13-2	256.29
9	89	11-9-5	225.77
10	89	12-5-8	219.10
11	94	9-9-7	233.45
12	82	11-3-11	207.51

Table 3: PROMISE results for the Conjugate Gradient code

[8] http://aces.snu.ac.kr/software/snu-npb

7 Conclusion & perspectives

User-friendly and efficient tools for the numerical validation of half precision codes and floating-point auto-tuning including half precision have been presented. They can ensure that a given code can be tuned to use low precision types while providing a user-defined numerical code quality. This work can be extended to other floating-point formats, such as *bfloat16* available for instance on Google Cloud TPUs and ARM NEON CPUs. Since CADNA and PROMISE have been successfully used for the numerical validation of real-life applications in single or in double precision, we plan to control the numerical quality of large scale applications in half precision, such as deep learning codes.

Acknowledgments

The authors gratefully acknowledge the financial support under the scope of the COMET program within the K2 Center "Integrated Computational Material, Process and Product Engineering (IC-MPPE)" (Project No 859480). This program is supported by the Austrian Federal Ministries for Transport, Innovation and Technology (BMVIT) and for Digital and Economic Affairs (BMDW), represented by the Austrian research funding association (FFG), and the federal states of Styria, Upper Austria and Tyrol.

References

1. Carson, E., Higham, N.J.: Accelerating the solution of linear systems by iterative refinement in three precisions. SIAM J. Sci. Comput. 40(2), A817–A847 (2018)
2. Eberhart, P., Brajard, J., Fortin, P., Jézéquel, F.: High performance numerical validation using stochastic arithmetic. Reliable Computing 21, 35–52 (2015)
3. Graillat, S., Jézéquel, F., Wang, S., Zhu, Y.: Stochastic arithmetic in multiprecision. Mathematics in Computer Science 5(4), 359–375 (2011)
4. Graillat, S., Jézéquel, F., Picot, R., Févotte, F., Lathuilière, B.: Auto-tuning for floating-point precision with discrete stochastic arithmetic. Journal of Computational Science 36, 101017 (2019)
5. Haidar, A., Abdelfattah, A., Zounon, M., Wu, P., Pranesh, S., Tomov, S., Dongarra, J.: The design of fast and energy-efficient linear solvers: On the potential of half-precision arithmetic and iterative refinement techniques. vol. 10860, p. 586–600. Springer, Wuxi, China (Jun 2018)
6. IEEE Computer Society: IEEE Standard for Floating-Point Arithmetic. IEEE Standard 754-2008 (Aug 2008)
7. Lefebvre, D., Chevrel, P., Richard, S.: An H_∞ based control design methodology dedicated to the active control of longitudinal oscillations. IEEE Trans. on Control Systems Technology 11(6), 948–956 (November 2003)
8. Tagliavini, G., Marongiu, A., Benini, L.: Flexfloat: A software library for transprecision computing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39(1), 145–156 (2020)
9. Vignes, J.: Discrete Stochastic Arithmetic for validating results of numerical software. Numerical Algorithms 37(1–4), 377–390 (Dec 2004)
10. Zeller, A.: Why Programs Fail. Morgan Kaufmann, Boston, second edn. (2009)