Basic Requirements for a Future Floating-Point Arithmetic Standard[1]

GAMM Fachausschuss on
Computer Arithmetic and Scientific Computing

For elementary binary floating-point computation the IEEE standard arithmetic, adopted in 1985, is undoubtedly thorough, consistent, and well defined [1]. It has been widely accepted and can be found in almost every processor developed since 1985. This has greatly improved the portability of floating-point programs.

Dramatic advances in speed and memory size of computers have been made since 1985. Arithmetic speed has gone from megaflops to gigaflops, to teraflops, and to petaflops. This is not just a gain in speed. A qualitative difference goes with it. If the numbers a petaflops computer produces in one hour were to be printed (500 on one page and 1000 on one sheet) they would fill a pile that reaches twice from the earth to the sun. Computing that is continually and greatly speeded up calls conventional computing into question. Even with quadruple and extended precision arithmetic the computer remains an experimental tool. **A new arithmetic standard must move strongly towards more reliability in computation.** Instead of the computer being merely a fast calculating tool it must be turned into a scientific instrument of mathematics. Three simple steps in this direction would have great effect. They are both simple and practical. These requirements are backed up by extensive experience with software and hardware implementations carried out over the last three decades [2, 10, 11, 12, 15, 16, 17, 9, 20].

During the last several decades of numerical analysis, methods for very many problems have been developed which allow the computer *itself* to validate or verify its computed results. Mathematical fixed-point theorems, residual correction techniques, and interval arithmetic are basic engredients of these methods. Hardly any of these verification techniques needs higher precision floating-point arithmetic. Double precision floating-point arithmetic is the basic arithmetical tool.

Three additional arithmetical features are fundamental and necessary:
I. fast interval arithmetic,
II. a fast and exact multiply and accumulate instruction
or, what is equivalent to it, an exact scalar product, and
III. elementary functions with proven and reliable *a priori* error bounds.

I: The IEEE standard 754 seems to support interval arithmetic. It requires the basic four arithmetic operations to have rounding to nearest, towards zero, and

[1]Draft: Oct. 10, 2006.

downwards and upwards. The latter two are essential for interval arithmetic. But almost all processors that provide IEEE arithmetic separate the rounding from the basic operation, which makes interval arithmetic extremely slow.

Future processors should provide the basic arithmetic operations with the monotone downwardly and upwardly directed roundings $\bigtriangledown\!\!\!\!\!\!+$, $\bigtriangledown\!\!\!\!\!\!-$, $\bigtriangledown\!\!\!\!\!\!\times$, $\bigtriangledown\!\!\!\!\!\!/$ and $\bigtriangleup\!\!\!\!\!\!+$, $\bigtriangleup\!\!\!\!\!\!-$, $\bigtriangleup\!\!\!\!\!\!\times$, $\bigtriangleup\!\!\!\!\!\!/$ by distinct operation codes. Each of these 8 operations must be callable as a single instruction, that is, the rounding mode must be inherent in each.

It is desirable that with a new arithmetic standard, denotations for arithmetic operations with different roundings be introduced. They could be:

`+`, `-`, `*`, `/`	for operations with rounding to the nearest floating-point number,
`+>`, `->`, `*>`, `/>`	for operations with rounding upwards,
`+<`, `-<`, `*<`, `/<`	for operations with rounding downwards, and
`+|`, `-|`, `*|`, `/|`	for operations with rounding towards zero (chopping).

This would lead to much more easily readable expressions than for instance the use of operators like .addup. or .adddown. in Fortran. In languages like C++ which just provide operator overloading and do not allow user defined operators these operations would have to be called as functions or assembler instructions.

The operations with the monotone directed roundings are basic building blocks for interval arithmetic. Speed has always been an issue for computing and especially so for interval arithmetic. It is very desirable, therefore, that the case selections for interval multiplication (9 cases) and interval division (14 cases) are also supported by hardware where they can be chosen without any time penalty. See [15, 18].

II: To achieve high speed all conventional vector processors provide a multiply and accumulate instruction. It is, however, not reliably accurate. By pipelining, the accumulation (continued summation) is executed very swiftly. It is done in floating-point arithmetic using the so-called partial sum technique. This alters the sequence of the summands and causes errors beyond the usual floating-point errors [13, 14].

To obtain narrow bounds for a result, interval arithmetic has to be combined with defect correction or iterative refinement techniques. To be effective these techniques should be supported by an exact multiply and accumulate operation or, what is equivalent to this, an exact scalar product. It may be realized e.g. by accumulating complete products of double length factors into a wide fixed-point word. Fixed-point accumulation is simple, error free, and fast. The result is independent of the sequence in which the summands are added. No under- or overflow can occur during a scalar product computation, if the width of this fixed-point word is properly chosen. For the double precision data format, about 0.5 K bytes suffice for this fixed-point word. The exact scalar product eliminates many rounding errors in numerical computations and it stabilizes these computations. It is a necessary complement to floating-point and interval arithmetic.

Apparently, quadruple precision arithmetic will be part of the next IEEE arithmetic standard. But this should not be the only way to higher accuracy. If the attainable accuracy in a particular class of problems is insufficient with double precision, then it will very often be insufficient with quadruple precision as well. It is necessary, therefore, also to provide a basis for improving the accuracy rather than simply providing higher precision.

With the fast and exact multiply and accumulate operation, fast quadruple or higher precision arithmetic can also be provided easily. A multiple precision number is represented as a group of floating-point numbers. The value of the number is the sum of the components of the group. It can be represented in the wide fixed-point word. Addition and subtraction of multiple precision variables or numbers can easily be performed in this word. Multiplication of two such numbers is simply a sum of products of floating-point numbers. It also can be computed by means of the exact multiply and accumulate operation.

Software simulations for an exact scalar product are available in the XSC languages [12, 16, 17, 9], and in other more recent publications [4, 25]. A variable precision data type staggered precision is based on it. Both are heavily used in problem solving routines with automatic result verification [11, 5, 6].

Hardware support for the exact multiply and accumulate operation or the exact scalar product could considerably speed up the operation. A minimum hardware support would consist of:
a) computation of exact products of two double precision floating-point numbers (i.e., to their full length).
b) local memory on the arithmetic unit of between 1 K and 8 K bytes.

A future arithmetic standard, however, should consider full hardware support for an exact scalar product! Implementation in hardware is easy. The operation is performed by a shift of the exact products and their accumulation into a wide fixed-point register on the arithmetic unit. Fixed-point accumulation of the products is error free, and it is simpler than accumulation in floating-point arithmetic. There is a specific technique for absorbing a possible carry as soon as it arises. A very natural pipelining of the actions: *loading the data, computation of the product, shifting, and accumulation of the products* into the wide fixed-point register leads to very fast and simple circuits [2, 20]. The circuitry required for it is comparable to that for a fast multiplier with an adder tree, accepted years ago. If supported by a (conventional) vectorizing compiler the operation would boost both the speed of a computation and the accuracy of the result! Hardware implementation of a full quadruple precision arithmetic is likely to be more costly than implementation of the exact scalar product. The latter only requires fixed-point accumulation of exact products. Fast quadruple or multiple precision arithmetic and multiple precision interval arithmetic can easily be provided as a by-product.

The scalar product of two floating-point vectors is the most important and most frequently used elementary function. Since the dot product computed in double, quadruple or extended precision arithmetic can fail to produce a correct answer an error analysis is needed for all applications. This can be left to the computer. As the scalar product can always be executed exactly with moderate technical effort it should indeed always be executed exactly. Error analysis then becomes irrelevant. Furthermore, the same result is always obtained on different platforms. An exact *fused multiply and add* operation at the matrix level is inherent to it. It is fundamental to the whole of numerical analysis. In mathematics it makes a big difference whether an operation is exact for many or for all applications. Also, fast arithmetic for varying-precision formats can easily be provided with it to a certain extent.

III: All arithmetic operations, functions or routines predefined by the processor, the manufacturer or the vendor must be provided with proven and reliable *a priori* error bounds. These error bounds must be made known to the user so that he can deal with it. He should know just what he gets if an elementary function or routine is used in a program. It is the vendor's responsibility to deliver these error bounds. Error estimates or error statements based on speculation are simply not acceptable. Of course, deriving proven reliable *a priori* error bounds can take a lot of work. But this work has to be done only once. A great deal of it can be done by use of the computer and of interval arithmetic. For an individual elementary function different error bounds may be obtained for different evaluation domains. Their maximum then is a global error bound. Proven error bounds must consider all error sources, like the approximation error in a reduced range, errors resulting from range reductions, etc. [19, 23].

For the double precision format the conventional 24 elementary functions have been implemented (just using double precision arithmetic) with an error that is less than 1.5 ulp. Even for a long data type elementary functions can be implemented with an error less than a few ulp [3].

A programming language or an arithmetic standard that supports interval arithmetic also should provide elementary functions for interval data for all data formats of the standard and for dynamic precision. Of course, high accuracy is desirable. But speed also is an issue for interval arithmetic! For the double precision format the elementary functions have been implemented for interval arguments (just using double precision arithmetic) with proven reliable error bounds that differ from the best possible interval enclosure of the result by an error that is, say, less than 1.5 ulp for each of the bounds. For point intervals the computed bounds show immediately how accurately the function has been evaluated.

The vendor is responsible for the quality of the arithmetic and of the elementary functions, and he has to provide valid (guaranteed) *a priori* error bounds.

In Summary: Interval arithmetic can bring guarantees into computation while an exact *multiply and accumulate* instruction can bring high accuracy via defect correction methods and at high speed. It also is the key operation for fast multiple precision arithmetic.

Fast and accurate hardware support for **I., II.** and **III.** must be added to conventional floating-point arithmetic. All three are necessary extensions.

A computer is supposed to be a scientific instrument of mathematics. It is amazing that after 60 years of its use these elementary and most natural requirements are not yet taken as a matter of course in computers.

The requirements **I., II.** and **III.** for a future arithmetic standard are intended to give guidance to GAMM members when selecting future computers.

References

[1] American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic.* ANSI/IEEE Std. 754-1985, New York, 1985 (reprinted in SIGPLAN 22, 2, 9-25, 1987). Also adopted as IEC Standard 559:1989.

[2] Baumhof, Ch.: *Ein Vektorarithmetik-Koprozessor in VLSI-Technik zur Unterstützung des Wissenschaftlichen Rechnens.* Dissertation, Universität Karlsruhe, 1996.

[3] Defour, D., Hanrot, G., Lefvre, V., Muller, J.-M., Revol, N., Zimmermann, P.: *Proposal for a Standardization of Mathematical Function Implementation in Floating-Point Arithmetic.* In: *Numerical Algorithms*, vol. 37, 367375, 2004.

[4] Demmel, J.; Hida, Y.: *A floating-point technique for extending the available precision.* SIAM J. Sci. Comput. 25, 1214–1248, 2003.

[5] Hammer, R., et al.: *Numerical Toolbox for Verified Computing I: Basic Numerical Problems.* Springer-Verlag, Berlin/Heidelberg/New York, 1993.

Russian translation: MIR-Verlag, Moskau, 2005.

[6] Hammer, R., et al.: *C++ Toolbox for Verified Computing: Basic Numerical Problems.* Springer-Verlag, Berlin/Heidelberg/New York, 1995.

[7] Higham, N. J.: *The accuracy of floating point summation*, SIAM Journal on Scientific Computing, 14:4, 783-799, 1993.

[8] Higham, N. J.: *Accuracy and Stability of Numerical Algorithms.* Second edition, SIAM, Philadelphia, 2002.

[9] Hofschuster, W., Krämer, W.: *C-XSC 2.0: A C++ Library for Extended Scientific Computing* Preprint 2003/5, University of Wuppertal, 2003. Published in: Alt, R. et al. (Eds.): Numerical Software with Result Verification, Lecture Notes in Computer Science, Volume 2991/2004, Springer-Verlag, Heidelberg, pp. 15 - 35, 2004. See also: `http://www.math.uni-wuppertal.de/~xsc/` or `http://www.xsc.de/`

[10] *IBM System/370 RPQ. High Accuracy Arithmetic.* SA 22-7093-0, IBM Deutschland GmbH (Department 3282, Schönaicher Strasse 220, D-71032 Böblingen), 1984.

[11] *IBM High-Accuracy Arithmetic Subroutine Library (ACRITH).* IBM Deutschland GmbH (Department 3282, Schönaicher Strasse 220, D-71032 Böblingen), 3rd edition, 1986.
1. General Information Manual. GC33-6163-02.
2. Program Description and User's Guide. SC33-6164-02.
3. Reference Summary. GX33-9009-02.

[12] *ACRITH–XSC: IBM High Accuracy Arithmetic — Extended Scientific Computation.* Version 1, Release 1. IBM Deutschland GmbH (Schönaicher Strasse 220, D-71032 Böblingen), 1990.
1. General Information, GC33-6461-01.
2. Reference, SC33-6462-00.
3. Sample Programs, SC33-6463-00.
4. How To Use, SC33-6464-00.
5. Syntax Diagrams, SC33-6466-00.

[13] *IMACS-GAMM Resolution on Computer Arithmetic.* In Mathematics and Computers in Simulation 31, 297-298, 1989. In Zeitschrift für Angewandte Mathematik und Mechanik 70, No. 4, T5, 1990.

[14] *GAMM-IMACS Proposal for Accurate Floating-Point Vector Arithmetic.* GAMM, Rundbrief 2, 9-16, 1993. Mathematics and Computers in Simulation, Vol. 35, IMACS, North Holland, 1993. News of IMACS, Vol. 35, No. 4, 375-382, Oct. 1993.

[15] Kirchner, R.; Kulisch, U.: *Hardware support for interval arithmetic.* Reliable Computing, 1–16, 2006.

[16] Klatte, R., et al.: *PASCAL–XSC — Sprachbeschreibung mit Beispielen.* Springer-Verlag, Berlin/Heidelberg/New York, 1991. See also: `http://www.math.uni-wuppertal.de/~xsc/` or `http://www.xsc.de/`

English translation: *PASCAL–XSC — Language Reference with Examples.* Springer-Verlag, Berlin/Heidelberg/New York, 1992. See also: `http://www.math.uni-wuppertal.de/~xsc/` or `http://www.xsc.de/`

Russian translation: MIR-Verlag, Moskau. 1995, third edition 2006.

[17] Klatte, R., et al: *C–XSC, A C++ Class Library for Extended Scientific Computing.* Springer-Verlag, Berlin/Heidelberg/New York, 1993. See also: `http://www.math.uni-wuppertal.de/~xsc/` or `http://www.xsc.de/`

[18] Kolla, R.; Vodopivec, A.; Wolff v. Gudenberg, J.: *Splitting Double Precision FPUs for Single Precision Interval Arithmetic.* In W. Erhard et al. (edts.), ARCS'99 Workshops zur Architektur von Rechensystemen, Universität Jena, 1999, 5-16,1999.

[19] Krämer, W.: *A priori Worst Case Error Bounds for Floating-Point Computations*, IEEE Transactions on Computers, Vol. 47, No. 7, July 1998.

[20] Kulisch, U.: *Advanced Arithmetic for the Digital Computer — Design of Arithmetic Units.* Springer-Verlag, Wien, New York, 2002.

[21] Kulisch, U.: *Letters to the IEEE Computer Arithmetic Revision Group.* See: `http://www.math.uni-wuppertal.de/org/WRST/preprints/prep_06_5.pdf`

[22] Ogita, T.; Rump, S. M., and Oishi, S.: *Accurate Sum and Dot Product.* SIAM Journal on Scientific Computing (SISC), 26(6):1955-1988, 2005.

[23] Rump, S. M.: *Rigorous and portable standard functions.* BIT 41(3), 540-562, 2001.

[24] Rump, S.M.: *INTLAB - Interval Laboratory, a Matlab toolbox for verified computations, Version 5.1, 2005.* http://www.ti3.tu-harburg.de/rump/intlab/index.html.

[25] Rump, S.M.; Ogita, T.; Oishi,S: *Accurate Floating-point Summation.* Technical Report 05.12, Faculty for Information- and Communication Sciences, Hamburg University of Technology, November 13, 2005.

[26] Zielke, G.; Drygalla, V.: *Genaue Lösung linearer Gleichungssysteme.* GAMM Mitt., Ges. Angew. Math. Mech. 26, 7–107, 2003.