MOTION 25

ROUNDED FLOATING-POINT ARITHMETIC

PROPOSER: ULRICH KULISCH

Abstract. The IEEE standard 754 provides floating-point operations with four different roundings. It is highly desirable making these easily accessible by the user. This proposal deals with the question how this can or should be done.

Key words: computer arithmetic, floating-point arithmetic, interval arithmetic, arithmetic standards.

The standard IEEE 754 for floating-point arithmetic [1, 2] seems to support interval arithmetic. It requires the basic four arithmetic operations with rounding to nearest, towards zero, and with rounding downwards and upwards. The latter two are needed for interval arithmetic. But almost all processors that provide IEEE 754 arithmetic separate the rounding from the operation, which proves to be a severe drawback. It makes interval arithmetic unnecessarily slow compared with conventional floating-point arithmetic.

It is desirable that with an interval arithmetic standard, denotations for arithmetic operations with the four IEEE 754 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).

Frequently used programming languages regrettably do not allow four operators for each of the operations plus, minus, multiply, and divide for floating-point numbers. This, however, does not justify separating the rounding from the operation!

Instead, a future interval arithmetic standard should require that **every future processor** shall provide the 16 operations listed above as distinct instructions. A future standard even can and should dictate names for the corresponding assembler instructions as for instance:
addp, *subp*, *mulp*, *divp*, *addn*, *subn*, *muln*, *divn*, *addz*, *subz*, *mulz*, and *divz*.
Here p stands for rounding toward positive, n for rounding toward negative, and z for rounding toward zero. With these operators interval operations can easily be programmed. They would be very fast and fully transferable from one processor to another.

Language designers will have to be convinced that this is much simpler and leads 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 be called as functions or via assembler. In such languages operations with the directed roundings are hidden within the interval operations. If operations with directed roundings are needed, interval operations can be performed instead and the upper or lower bound of the result then gives the desired answer.

References

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

[2] IEEE Floating-Point Arithmetic Standard 754, 2008.

[3] Pryce, J. D. (Ed.): *P1788, IEEE Standard for Interval Arithmetic,* http://grouper.ieee.org/groups/1788/email/pdfOWdtH2mOd9.pdf.

[4] Kirchner, R., Kulisch, U.: *Hardware support for interval arithmetic.* Reliable Computing 12:3, 225–237, 2006.

[5] Kulisch, U.,W.: *Computer Arithmetic and Validity – Theory, Implementation and Applications.* De Gruyter, Berlin, New York, 2008.

[6] Neumaier, A.: *Vienna Proposal for Interval Standardization.*

Institut für Angewandte und Numerische Mathematik, Universität Karlsruhe, D-76128 Karlsruhe GERMANY

E-mail address: `Ulrich.Kulisch@math.uka.de`