Interval Computations
No 4, 1994

Towards a User Transparent Interval Arithmetic

Marc Daumas, Christophe Mazenc, and Jean-Michel Muller

The interval arithmetic provides a tool powerful enough to validate many scientific computations. Yet no implementation standard is available to the programmers. Testing a code involves getting acquainted with an evolving non standard interval arithmetic package, modifying the code and running the new source with the correct extensions. We propose and investigate a self contained easy to use interval arithmetic package that can eventually be put in place of the standard floating point arithmetic. The program produces some guaranteed bounds on the result of the theoretical real value of the numerical approximation without even recompiling the source code.

На пути к <прозрачной> для пользователя интервальной арифметике

М. Дома, К. Мазенк, Ж.-М. Мюллер

Интервальная арифметика представляет собой мощный инструмент, во многих случаях позволяющий убедиться в достоверности научных вычислений. К сожалению, программисты до сих пор не имеют стандарта на реализацию этого инструмента. Для тестирования программы часто требуется знакомство с новой реализацией интервальной арифметики, модификация исходного текста и прогон измененной программы с подключением соответствующих средств расширения. Нами предлагается и исследуется законченный, простой в использовании пакет интервальной арифметики, способный в конечном итоге заменить стандартную арифметику с плавающей точкой. Программа вычисляет некоторые гарантированные границы для результата теоретического вещественного значения численного приближения даже без повторной компиляции исходной программы.

1 Introduction

The test of any proper scientific code and its validation usually take an important share of the task of the software development. The techniques commonly accepted include some numerical error analysis and/or the use of the interval arithmetic. Yet testing a code with an interval arithmetic tool requires the programmer's deep knowledge of the different packages available. Because no standard is commonly accepted for interval arithmetic each package offers some different solutions to many problems that are related but not totally identical.

In Section 1, we propose a number format for the interval arithmetic. The actual set of the representable intervals follows from the machine encoding. Our defined *compression scheme*, presented in the Section 2, holds the place of the floating point rounding mechanism. In the last section, Section 3, we review the features of the user transparent interval floating point arithmetic.

The data format specified in this work is compatible with the IEEE standard type hierarchy: A double precision floating point interval is represented by an eight byte word. An interval is first represented by a floating point number called the origin. Integrated to the IEEE standard philosophy, the shape of the interval depends on the active rounding mode. For the Round to the Nearest mode, the origin is the center of the interval. In a directed rounding mode, the canonical bound (upper or lower bound) of the interval is used as origin.

We have added two fields to the standard floating point encoding: the width of the interval and a slash field. In [9], Matula *et al* have proposed to code the two component of a fractional numbers on one single field with a shifting separator. The position of the separator is coded in a small additional field: the slash field. We will see in the last section that coding both the mantissa of the origin and the width of the interval on two separate fixed size fields appears quite restrictive: The adapting format exhibits a behavior very close to the error propagation in floating point operations.

The format handles all the usual arithmetic operations on the intervals by computing the exact interval image of its real or interval operands and *compressing* the result to the desired fixed-length interval format; the same type of mechanism is used in the IEEE standard to round the floating point number to the fixed-length representation. The external floating point operands supposed *exact* are supported by setting the error field length to 0.

2 Data format

Checking a scientific code with an existing interval arithmetic package is in the best case a long painstaking process. Assuming the programmer is not sufficiently aware of the extended list of different tools provided to help him (including [2]) this process might also be error prone. The external data format presented here to store an interval is meant to allow a non specialist user to switch from the IEEE standard [6] floating point operation to the floating point interval arithmetic in a matter of seconds. Integrated into a chip the process would not even require recompiling the source code.

2.1 Bit encoding

The proposed length of the interval format is identical to the the length of the corresponding IEEE format: A double precision interval is represented by 8 bytes (64 bits). No extra space will be required in memory to switch from one representation (floating point) to the other one (interval). Five fields are involved in the definition of a floating point interval (see Figure 1). Some of these fields are common with the IEEE floating point standard, although their lengths may differ. The position of the different fields is well suited so a programmer may even decide to read an interval as a floating point number. In the last section, we present the meaning of such a conversion, along with the actual cost of the proposed encoding.

- The first field contains the *sign* bit s. Like the IEEE standard floating point numbers, a floating point interval is stored with a separate sign information and absolute value.

- The *exponent* field e is kept unchanged from the IEEE standard—same size, same encoding, same signification except for the floating point storage of the value 0. This point is detailed latter in this section.

- The *fraction* f of the mantissa $m = 1.f$ is stored with the bits not allocated for the dynamic error field δ.

- The new *error* field δ stores the relative width of the interval. The exact interpretation of δ depends on the rounding mode and the value of the other fields.

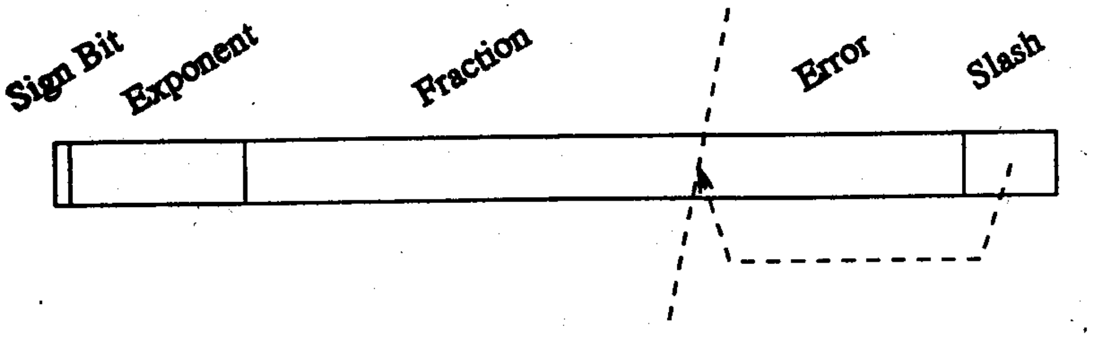

Figure 1: Bit coding map of a floating point interval

Precision	Length (bytes)	s	e	(f, δ) (bits)	l_m
Single	4	1	8	18	5
Double	8	1	11	46	6
Quad	16	1	15	106	7

Figure 2: Proposed standard floating point interval hierarchy

- The *slash* field l_m counts the number of bits allocated to the mantiss. To limit any possible value for the couple (m, δ) in the 52 bits originally available for the fraction in the IEEE standard double precision format, the field l_m is 6 bits wide. The bits are automatically allocated and reclaimed from the fields that store f and δ. The value L is the total length of the mantissa field and the error field.

2.2 Floating point interval origin

An interval is based on its floating point origin x_0. The active rounding mode and the value of the error field δ define the position and the width of the interval compared to its origin x_0 (see Figure 3). The origin x_0 uses only three of the fields of the floating point. The mantissa m of x_0 is formed with the fraction f; the value of m is represented by the fractional number below.

$$m = 1.f.$$

The standard manipulations on m involve the quantity of one *unit in the last place* (ulp), and the strict usage of the ulp-rounding mechanism. We

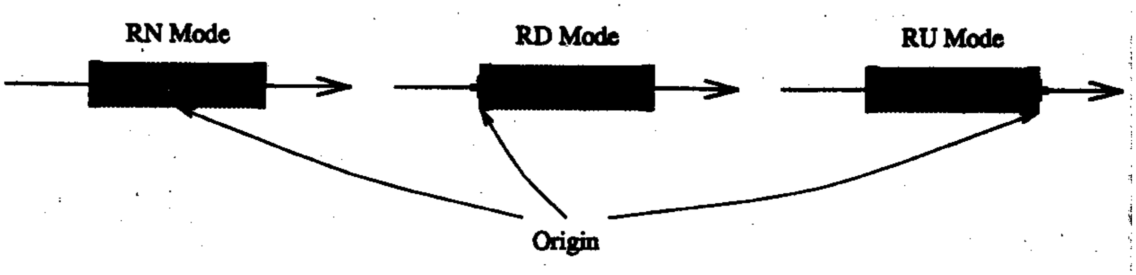

Figure 3: Position of the origin of the interval

adopt in this work the notation proposed in [5]: The *integer mantissa* M of x_0 is coded with an integer number from the concatenation on words $\oplus$, as follows.

$$M = 1 \oplus f.$$

The two representations m and M of the mantissa of x_0 are equivalent. However in the second case, an ulp is exactly one unity, independently from the length of M. With the common floating point arithmetic, the conceptual distinction between m and M is not so important since the length of the mantissa is constant. However, for the proposed floating point interval arithmetic, the length of the mantissa is variable and depends on the relative size of the interval and more directly on the value of l_m. Since the number is normalized, the mantissa satisfies

$$2^{l_m} \leq M \leq 2^{l_m+1} - 1.$$

The field l_m stores the length of the mantissa.

$$l_m = \begin{cases} \lfloor \log_2 M \rfloor + 1 & M \neq 0 \\ 0 & M = 0 \end{cases}.$$

According to the proposed format, the length l_δ of the error field δ is conjugated to l_m. The combined length L of both fields depends on the precision format as detailed in Figure 4.

$$l_\delta = L - l_m \qquad \text{or} \qquad l_m + l_\delta = L$$

The interpretation of x_0 is defined bellow using both conventions. We shall call the integer mantissa M the *mantissa* of x_0. Still f represents the

Working Precision	Value of L
Single	19
Double	47
Quad	107

Figure 4: Conjugate value of l_δ and l_m

Rounding Mode	Interval
RD	$[x_0, x_0 + \Delta]$
RU	$[x_0 - \Delta, x_0]$
RN	$[x_0 - \Delta/2, x_0 + \Delta/2]$

Figure 5: The floating point interval related to the active rounding mode

fractional part of the fractional number $1.f$, thus the integer f is called the *fraction* of x_0.

$$\begin{aligned} x_0 &= (-1)^s \times m \times 2^e & (m = 1.f) \\ &= (-1)^s \times M \times 2^{e-l_m-1} & (M = 1 \oplus f). \end{aligned}$$

2.3 Interval interpretation

The absolute error is expressed according to the exponent field. We define the quantity Δ.

$$\Delta = \delta \times 2^{e-l_m-1}.$$

The IEEE standard specifies four rounding modes. The two letter notation of these modes are RN (Round to Nearest—even), RU (Round Up towards $+\infty$), RD (Round Down towards $-\infty$) and RZ (Round towards Zero). The last rounding mode (RZ) is not adapted to the interval computation, we will focus on the three modes RN, RU and RD. The interval I relative to any rounding mode is detailed in Figure 5. The semantic inherent to the rounding mechanism in the IEEE standard arithmetic is still valid in the choice of the origin x_0 of the interval. For example, if the active mode is RD (Round Down), the floating point origin represents a lower bound on the interval.

Mantissa	Width	Error	Bit Coding
1.101...	20	...110	(1)101...110
0.000...	0	...110	(0)000...110

Figure 6: Intervals centered on zero

Intervals centered on zero. In many real applications, the intervals which have zero as origin are very important. In the RN mode, this represents a situation where too much cancellation has made impossible to deduce any information on the sign, yet the absolute value of the result is bounded. In the directed rounding modes, an interval with origin $x_0 = 0$ delivers some precious sign information.

According to the previous encoding mechanism, it is not possible to represent an interval whose origin x_0 is 0. In fact, the IEEE standard specifies that 0 is coded with both machine exponent, so called characteristic in [1], and fraction field set to zero. For the double precision arithmetic and for a non extremal value, the exponent is computed from the formula below.

$$e = \text{Characteristic} - (2^{10} - 1).$$

We can easily extend this notion to the value of zero: It is coded as $0 = 0 \times 2^{-2^{10}+1}$. With this first specification, the possible widths of an interval centered on zero is quite restricted ($\Delta < 2^{-2^{10}+46}$). The numbers of bits of the fraction field is stored as $l_m - 1$: An interval with $l_m = 0$ represents an interval where all the bits of the mantissa are lost to the error field. Even the implicit bit in the first place of the mantissa should not be considered as a 1 but as a 0. This representation introduces an interval centered on zero with any valid exponent. We present some examples in the Figure 6.

3 The compression scheme

In the previous section, we have presented the encoding of the floating point intervals. For a coherent number system, we detail the operations possible on this system. The IEEE standard definition relies heavily on the notion of rounding. Actually, the type hierarchy and the rounding mechanisms are sufficient to fully describe the IEEE standard number system. The closure and the exception handling are rather involved in the definition of the execution model.

The rounding mechanism detailed in the IEEE standard is a compression scheme: It compresses an arbitrarily long floating point number to a fixed length representation; the initial number be finite like any value $1 + 2^{-k}$ or even non finite, for example $\frac{1}{3}$.

With the compressing scheme of a floating point intervals, any mathematical operation is specified as the unique compression of the smallest interval containing the result of the infinitely precise operation of the real or interval exact inputs. This definition is totally deterministic and is comparable to the IEEE standard.

3.1 Theoretical background

We are interested in the compression of an interval presented as a normalized triplet (M, δ, e) with the proposed rounding scheme and the active rounding mode Rx. For any integer i, we build the origin $x_0^{(i)}$ of the interval $I^{(i)}$:

$$\begin{array}{rcl} M^{(i)} & = & \mathrm{Rx}\left(\frac{M}{2^i}\right) \\ {l_m}^{(i)} & = & \begin{cases} \left\lfloor \log_2 M^{(i)} \right\rfloor + 1 & M^{(i)} \neq 0 \\ 0 & M^{(i)} = 0 \end{cases} \\ e^{(i)} & = & e - l_m + {l_m}^{(i)} + i \\ x_0^{(i)} & = & (-1)^s \times M^{(i)} \times 2^{e^{(i)} - {l_m}^{(i)} - 1}. \end{array}$$

Proposition 1. *For any i, and provided $\delta^{(i)}$ satisfies the following condition, $I^{(i)}$ is a minimal normalized interval with $x_0^{(i)}$ as origin containing I.*

$$R^{(i)} = \begin{cases} \left|M - M^{(i)} \times 2^i\right| & \text{if } Rx = RD \text{ or } RU \\ 2 \times \left|M - M^{(i)} \times 2^i\right| & \text{if } Rx = RN \end{cases},$$

$$\delta^{(i)} = \left\lceil \frac{\delta + R^{(i)}}{2^i} \right\rceil .$$

Proof. We have to prove both that the interval $I^{(i)} = [a^{(i)}, b^{(i)}]$ contains $I = [a, b]$ and that $I^{(i)}$ is the smallest interval with $x_0^{(i)} \equiv (M^{(i)}, e^{(i)})$ as origin containing I. Without loss of generality we may assume that $x_0 > 0$ $\big($i.e. $s \equiv 0 \pmod 2\big)$.

Round Down towards $-\infty$. The demonstration is very close from one rounding mode to the other although there are some differences; the bounds of the initial interval I are:

$$a = M \times 2^{e-l_m-1} \qquad \text{and} \qquad b = (M+\delta) \times 2^{e-l_m-1}.$$

The encoding mechanism implies that:

$$a^{(i)} = M^{(i)} \times 2^{e^{(i)}-{l_m}^{(i)}-1} \qquad \text{and} \qquad b^{(i)} = \left(M^{(i)} + \delta^{(i)}\right) \times 2^{e^{(i)}-{l_m}^{(i)}-1}.$$

From the definition of the rounding mechanism

$$M - 2^i + 1 \leq 2^i \times M^{(i)} \leq M.$$

Hence $a^{(i)} \leq a$. Substituting $e^{(i)}$ and $M^{(i)}$ by their value in the expression of $b^{(i)}$, it follows that after simplification:

$$b^{(i)} = \left(2^i M^{(i)} + 2^i \left\lceil \frac{\delta + \mathrm{R}^{(i)}}{2^i} \right\rceil\right) 2^{e-l_m-1}.$$

From the definition of $\mathrm{R}^{(i)}$

$$b^{(i)} = \left(M - \mathrm{R}^{(i)} + 2^i \left\lceil \frac{\delta + \mathrm{R}^{(i)}}{2^i} \right\rceil\right) 2^{e-l_m-1}.$$

However, $\left\lceil \frac{\delta+\mathrm{R}^{(i)}}{2^i} \right\rceil$ satisfies

$$e + \mathrm{R}^{(i)} \leq 2^i \left\lceil \frac{\delta + \mathrm{R}^{(i)}}{2^i} \right\rceil < e + \mathrm{R}^{(i)} + 2^i - 1.$$

From the first part of the last inequality, we deduce the following property and finally that $I^{(i)}$ contains I.

$$b^{(i)} \geq \left(M - \mathrm{R}^{(i)} + \delta + \mathrm{R}^{(i)}\right) 2^{e-l_m-1}.$$

We next build an interval $I'^{(i)} = [a'^{(i)}, b'^{(i)}]$ with the same origin $(M^{(i)}, {l_m}^{(i)}, e^{(i)})$ but with the error field set to $\delta'^{(i)} = \delta^{(i)} - 1$. $b'^{(i)}$ is defined as follows.

$$b^{(i)} = \left(M^{(i)} + (\delta^{(i)} - 1)\right) \times 2^{e^{(i)} - {l_m}^{(i)} - 1}.$$

With a substitution comparable to the one presented earlier,

$$b'^{(i)} = \left(M - \mathrm{R}^{(i)} + 2^i \left\lceil \frac{\delta + \mathrm{R}^{(i)}}{2^i} \right\rceil - 2^i \right) 2^{e - l_m - 1}.$$

From the second part of the inequality on $\left\lceil \frac{\delta + \mathrm{R}^{(i)}}{2^i} \right\rceil$ we deduce that

$$b^{(i)} < \left(M - \mathrm{R}^{(i)} + \delta + \mathrm{R}^{(i)}\right) 2^{e - l_m - 1}.$$

Round Up towards $+\infty$*.* The mechanism of the proof is very close to the one just presented, although it is based on $a^{(i)}$. Since $\mathrm{R}^{(i)}$ is the absolute value of the rounding error involved in the division of $M^{(i)}$, the property used for the Round Down mode on $b^{(i)}$ holds for $a^{(i)}$.

Round to Nearest. The Round to Nearest mode is something different from the two other ones. The critical bound is not know for any value of M. Depending on the sign of $M - 2^{(i)} M^{(i)}$, we must consider $a^{(i)}$ or $b^{(i)}$ to prove that the error field $\delta^{(i)}$ is minimal.

$$a = \left(M - \frac{\delta + 1}{2}\right) \times 2^{e - l_m - 1} \qquad \text{and} \qquad b = \left(M + \frac{\delta + 1}{2}\right) \times 2^{e - l_m - 1}.$$

By a substitution close to the ones used earlier, we obtain

$$a^{(i)} = \left(2^i \times M^{(i)} + 2^{i-1} \left\lceil \frac{\delta + \mathrm{R}^{(i)}}{2^i} \right\rceil\right) 2^{e - l_m - 1}$$

and

$$b^{(i)} = \left(2^i \times M^{(i)} - 2^{i-1} \left\lceil \frac{\delta + \mathrm{R}^{(i)}}{2^i} \right\rceil\right) 2^{e - l_m - 1}.$$

Without loss of generality, we suppose that $M \geq 2^i M^{(i)}$; it means that M was rounded by default and we must verify that $b^{(i)} \geq b$.

$$b^{(i)} \geq \left(M - \frac{R^{(i)}}{2} + \frac{\delta + R^{(i)}}{2}\right) 2^{e - l_m} = b.$$

Last, we verify that the interval $\left[a'^{(i)}, b'^{(i)}\right]$ whose center is $x_0^{(i)}$ and radius is $\delta^{(i)} - 1$ does not contain the interval $[a, b]$:

$$\begin{aligned} b'^{(i)} &= \left(2^i M^{(i)} + 2^{i-1}\left\lceil\frac{\delta + \mathrm{R}^{(i)}}{2^i}\right\rceil - 2^{i-1}\right)2^{e-l_m-1} \\ &< \left(M - \frac{R^{(i)}}{2} + \frac{\delta + R^{(i)}}{2} - 2^{i-1}\right)2^{e-l_m-1} \\ &< \left(M + \frac{\delta}{2}\right)2^{e-l_m-1} \quad < \quad b. \end{aligned}$$

□

3.2 Optimality

Intuitively, if there is a value of i where the lengths of $M^{(i)}$ and $\delta^{(i)}$ are optimal, this is the compressed value of I. This property still hold if no couple $(M^{(i)}, \delta^{(i)})$ exactly matches the length of the interval.

Conjecture 1. *For any length L, and given the minimal integer i such that the normalized interval $I^{(i)} \equiv (M^{(i)}, \delta^{(i)}, e^{(i)})$ whose mantissa is coded with L bits (i.e. ${l_m}^{(i)} + {l_\delta}^{(i)} \leq L$); any normalized interval $I' \equiv (M', \delta', e')$ containing I and satisfying ${l_m}' + {l_\delta}' \leq l$ is larger than $I^{(i)}$.*

This property may not hold only when the active mode is Round to Nearest and the center of the interval $M^{(i)}$ is an exact power of 2 (i.e. $M^{(i)} = \pm 2^k$).

We shall present a counter-example for the Round to Nearest mode (see Figure 7): The value of M is $1111000b$ and the value of δ is $11010b$; e can be set to 0. We are interested in having only 3 bits of mantissa. The compression scheme returns for $i = 5$ the rounded value $M^{(i)} = 100b$ and an error of $\delta = 10$ with $e^{(i)} = 1$. The solution $M = 111b$, with $e' = 0$ and not 1 induces a minimal $\delta' = 11b$. With the correct exponent range, the interval I' is smaller than $I^{(i)}$.

	Original	Rounded	Truncated
Center	1.111000e0	1.00/00000e1	1.11/1000e0
Error	11010e0	1/00000e1	11/0000e0
Upper Bound	10.010010e0	1.01/00000e1	10.01/0000e0
Lower Bound	1.011110e0	0.10/00000e1	1.00/0000e0

Figure 7: Counter-example for the Conjecture 1

3.3 Practical implementation

To develop a chip, we may have to compute both bounds of the interval; from the bounds, we deduce the normalized triplet (M, e, δ). Depending on the length of M and δ, we can deduce one unique value i_0. Let

$$k = \left\lceil \frac{l_m + l_\delta - l}{2} \right\rceil$$

$$i_0 = \begin{cases} k, & l_m > k,\ l_\delta > k \\ l_m - l, & l_\delta < k \\ l_\delta - l, & l_m < k \end{cases}$$

Either $I^{(i_0)}$ or $I^{(i_0+1)}$ is the optimal rounding interval. We have detailed the three possible rounding situations in the Figure 8. It presents in a shadow box the possible carry ripple due to the rounding of $M^{(i)}$ and $\delta^{(i)}$. This is the reason why $i_0 + 1$ might be useful; if no ripple occurs $I^{(i_0)}$ fits the format and is optimal. If the exponent $e^{(i_0)}$ of the result overflows the format, the result is stored as $\pm\infty$.

4 Interval computation

4.1 Cost of the interval arithmetic

The bits used for the slash encoding field (6 bits) are actually lost compared to standard floating point computation. However, this means a loss of precision of $2^{-44} \approx 5.68 \times 10^{-14}$ or roughly 10% of the mantissa length. On the other hand, the bits used to store the error fields δ are not really lost. In the standard floating point operation the least significant bits of the result will

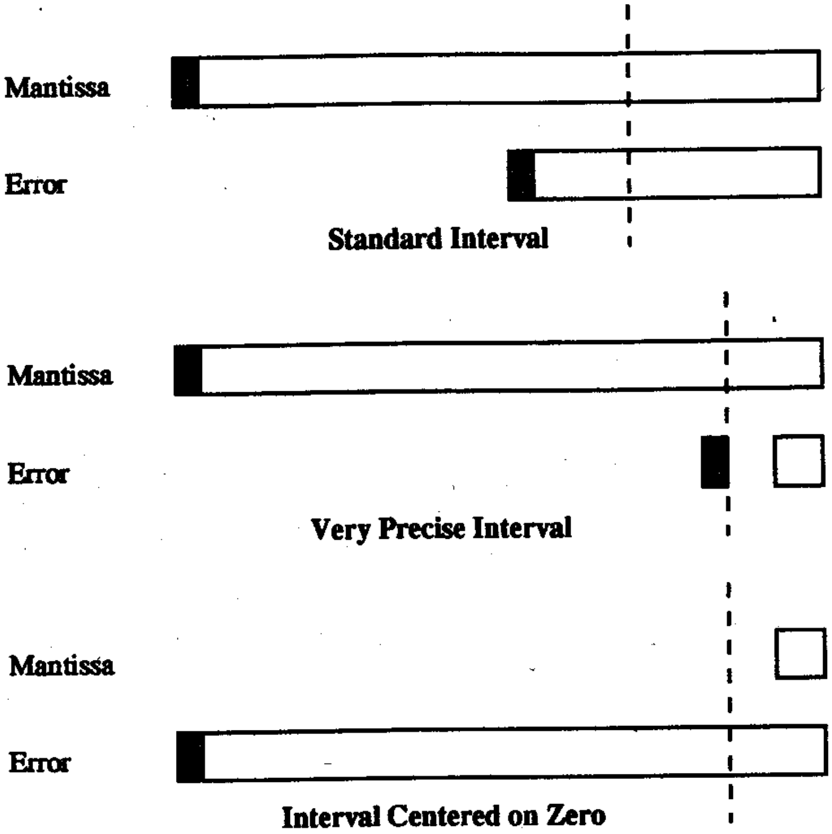

Figure 8: Compression scheme

most certainly hold some un-validated value. During a compound operation, the rounding errors are introduced into the partial results; these errors are growing and are propagated: The last bits of the results are not valid.

The encoding of the error field of the floating point interval simulates this behavior. Due to carry ripple at most one bit is lost compared to the precision we would expect from validated floating point operation.

- After a floating point addition the last bit of the result is not correct; the error field of the floating point interval is one bit wide.

- With a floating point number, if a cancellation occurs, some zeros are inserted at the end of the word to normalize it; although the floating point addition is exact, the zeros inserted cannot be validated. In the floating point interval, no zeros are inserted, but the error field grows exactly by the same number of bits as the number of zeros that should have been inserted.

The interval arithmetic is known as too pessimistic: In a common application, the errors cancel and the final error is almost always far from the maximal theoretical error. This is the reason why a programmer accepts to work with the standard un-validated arithmetic. This is inherent to interval arithmetic and not to the special encoding proposed here.

4.2 Computing with an interval

As long as the hardware support is provided to the user, the interval arithmetic can be used in any place where the standard floating point arithmetic is used. Performing a computation with the interval arithmetic rather than the floating point arithmetic does not make any difference, except for the comparisons: $a = b$, $a < b$, $a \leq b, \ldots$ When the two intervals a and b are disjoint, it is possible to answer the comparison. But as soon as the two intervals share some values as presented in Figure 9, the answer cannot be returned safely. In this situation, we shall adopt a behavior comparable to the IEEE standard.

By implementing an exception mechanism rather then a fixed process for the division by zero for example, the IEEE standard allows a programmer to change the behavior of the machine depending on the nature of his

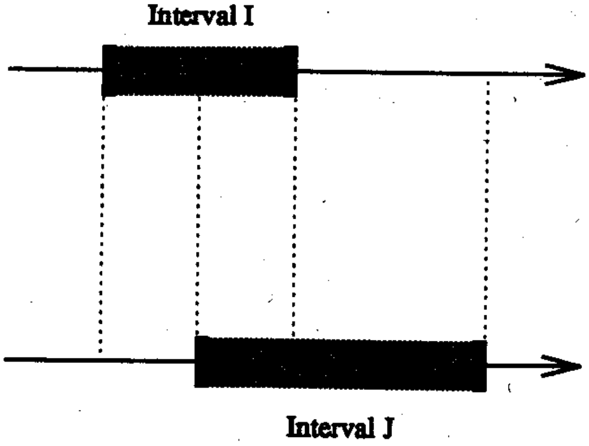

Figure 9: Comparing two intervals

computation. Whenever a dangerous comparison is intended the hardware module should only raise a user-defined exception. Here are some of the options that can be used in the case of a comparison exception.

- In the event of an exception, the computer should stop the execution and ask the operator for the *real* solution of the comparison. This approach would point sharply at the critical values.

- The comparison can be solved by comparing the origins. This allows the program to terminate without error. However, to get the certified result, a trace should be maintained. Each decision of the trace should be validated trough an experimental method.

- In the usual case, there are some valid assumption on the distribution of the values in the interval; statistically, the values close to the middle of the interval are more likely to be the correct result. Assuming the interval follows a Gaussian distribution, and if the programmer is able to evaluate the standard deviation, the computer would be able to evaluate the probability of the comparison to hold true.

- Proceeding with very high precision input, the computer should branch to a more precise process to compute both the two intervals of the comparison in order to get a better result. This is only true if the width of the intervals is due to accumulated rounding error rather than lack of precision from the input.

4.3 Floating point numbers and intervals

The proposed format for the floating point intervals is very close to the IEEE standard format for the floating point numbers. This allows the user to switch quickly form one format to the other. For example, the data structure (arrays, pointers,...) would not need any modification. After a computation, the program returns a floating point result and some indication on the precision. We should add some new functionalities to retrieve the information contained in the floating point interval:

Lower returns the lower bound of the interval, this is a floating point interval of length 0;

Upper identically returns the upper bound;

Width computes the absolute width of the interval.

We should also include conversion from floating point number to interval based on the two bounds of the interval, or on the origin and the relative or absolute error.

It is possible for an un-experimented user to use an interval in place of a floating point number without conversion. The floating point number obtained by ignoring the difference between the two formats is a valid representative of the interval.

Conclusion

We have presented an encoding system that allows any user to quickly switch from the standard floating point operation to the interval arithmetic. This operation does not even require recompiling the source code. A compression scheme applies to the high precision intervals and takes the place of the rounding scheme for the IEEE floating point operation. Hence the four arithmetic operations required by the IEEE standard for the floating point operation can be implemented with the interval arithmetic. Compared to the standard operation, the interval arithmetic only requires 6 bits out of the 53 bits of the mantissa fields in the precision. Yet at the end of a program, the user is guaranteed a mathematical bound on its result or is strongly pointed to the unstable quantities.

In the proposed format, the interpretation of an interval is changed when the active rounding mode is changed. If the user changes the rounding mode during a compound operation, the result is not valid. Yet, if the user needs to have access to the RD or the RU mode of the IEEE standard, he is performing explicit interval arithmetic, and might not switch to the transparent interval arithmetic. The interval arithmetic might require some important change in the habit of the programmers to be exploited with a large success: For example, the notion of equality should be replaced by the inclusion. With the major development of the technologies, we should not any more consider the interval arithmetic as a time and space over-expensive solution only useful for precision critical programs. This work contains some new directions to integrate the interval arithmetic into the mainstream computing. We are now investigating some optimized algorithms for the elementary functions.

References

[1] Anderson, S. F., Earle, J. G., Goldschmidt, R. E., and Powers, D. M. *The IBM system/360 model 91: floating point execution unit.* IBM Journal of Research and Development **11** (1967), pp. 34–53.

[2] Bleher, H., Kulisch, U., Metzger, M., Rump, S. M., Ullrich, C., and Walter, W. *Fortran-SC.* Computing **39** (1987), pp. 93–110.

[3] Bohlender, G. *What do we need beyond IEEE arithmetic?* In: “Computer Arithmetic and Self Validating Numerical Methods”, Academic Press, 1990, pp. 1–32.

[4] Cody, W. J. *Analysis of proposals for the floating point standard.* IEEE Computer **20** (3) (1987), pp. 63–68.

[5] Gosling, J. B. *Design of large high speed floating point arithmetic units.* Institution of Electrical Engineers, Proceeding **118** (1971), pp. 493–498.

[6] *IEEE standard for binary floating-point arithmetic.* ANSI/IEEE Std 754–1985. The Institute of Electrical Engineering and Electronics Engineers, New York, 1985.

[7] Kuck, D. J., Parker Jr., D. S., and Sameh, A. H. *Analysis of rounding methods in floating point arithmetic.* IEEE Transactions on Computers **C26** (1977), pp. 643–650.

[8] Kulisch, U. and Miranker, W. L. *Computer arithmetic in theory and practice.* Academic Press, 1981.

[9] Matula, D. W. and Kornerup, P. *Finite precision rational arithmetic: slash number systems.* IEEE Transactions on Computers **C34** (1) (1985), pp. 3–18.

Received: November 17, 1993
Revised version: July 4, 1994

Laboratoire de l'Informatique
du Parallélisme
École Normale Supérieure de Lyon
Lyon, France 69364