Design of a Fast Validated Dot Product Operation†

M. Daumas

Ecole Normale Supérieure de Lyon
Lyon, France 69364

D. W. Matula

Southern Methodist University
Dallas, TX 75275

Abstract

The dot product operation is very prevalent in scientific computation and has been incorporated as a primitive operation in some languages. Implementation of the dot product operation by a sequence of IEEE standard multiplications and additions does not prevent substantial error accumulation or warn about catastrophic cancellation.

The design of a double precision dot product operation is presented here where the final rounded result is validated by raising exception flags if either the result incurred catastrophic cancellation or if the result is not accurate to one unit in the last place (ulp). Our design allows that in the absence of catastrophic cancellation one ulp accuracy is guaranteed. The user can thus obtain validated results at marginal extra cost with the facility to trap to alternative routines in those cases where the results are suspicious.

Introduction

Dealing with an accumulation such as a dot product, the machine should provide enough information to fully qualify the correctness of the final sum. The existing systems do not allow such control whereas the dot product operation is a common tool in scientific computation [2].

Extending the precision to the quad format (cf. Table 1) is not acceptable to compensate for the unmonitored cancellation of leading bits set to 0 or the accumulation of round-off error since this method does not give any more information on the correction of the results. Moreover, this race for raw precision is very expensive in both time and space.

At the machine level, infinitely precise rounding is often achieved by performing internal computation with more bits than the output format. For example, the IEEE standard operations [7] are usually done with implicit infinite length [4].

Valid assumptions can only be assessed working with a multiple precision exact representation. But this precision level is not really cost efficient. Many propositions have been made to accumulate the partial sum in a multiple length representation [3, 9, 11, 14]. These different articles propose many different methods for computing error-free operations rather than coherently and reliably controlling the error. This approach is correct when the usual tools are not sufficient but this level of precision should not be used as a common tool.

Section 1 presents a well defined rounding mechanism (detailed in [5]) for vector arithmetic. Some suggestions are introduced for the analysis and the monitoring of the cancellation phenomenon in Section 2. The last section describes our self validating dot product algorithms.

1 Faithful Rounding

The IEEE standard [7] specifies four rounding modes: RN — Round to Nearest (even), RU — Round Up towards $+\infty$, RD — Round Down towards $-\infty$ and RZ — Round towards Zero). The result of an operation must be the exact rounding of the infinitely precise mathematical operation. Whereas the IEEE standard rounding is always attainable for the dot product employing the notion of a long accumulator,

	Sign	Exponent	Mantissa	Total
Single	1	8	(1) + 23	32
Double	1	11	(1) + 52	64
Quad	1	15	1 + 111	128

Table 1: Floating Point Formats

†This material is based upon work supported in part by the PRC "Architectures de Machines Nouvelles" of the french "Ministère de l'Education Nationale".

1063-6889/93 $03.00 © 1993 IEEE

We present the rounding modes corresponding to the four IEEE standard rounding modes

$$\text{RN} \longrightarrow \text{RND} \quad | \quad \text{RND}([x_1, x_2]) = \begin{cases} \text{RN}(x_1) & \text{RN}(x_1) = \text{RN}(x_2) \\ \text{RD}(x_1) & \text{RN}(x_1) \neq \text{RN}(x_2) \text{ but } \text{RD}(x_1) = \text{RD}(x_2) \\ \perp & \text{otherwise} \end{cases}$$

$$\text{RU} \longrightarrow \text{RUU} \quad | \quad \text{RUU}([x_1, x_2]) = \begin{cases} \text{RU}(x_1) & \text{RU}(x_1) = \text{RU}(x_2) \\ \text{RN}(x_1)^u & \text{RU}(x_1) \neq \text{RU}(x_2) \text{ but } \text{RN}(x_1) = \text{RN}(x_2) \\ \perp & \text{otherwise} \end{cases}$$

$$\text{RD} \longrightarrow \text{RDD} \quad | \quad \text{RDD}([x_1, x_2]) = \begin{cases} \text{RD}(x_1) & \text{RD}(x_1) = \text{RD}(x_2) \\ \text{RN}(x_1)^d & \text{RD}(x_1) \neq \text{RD}(x_2) \text{ but } \text{RN}(x_1) = \text{RN}(x_2) \\ \perp & \text{otherwise} \end{cases}$$

$$\text{RZ} \longrightarrow \text{RZZ} \quad | \quad \text{RZZ}([x_1, x_2]) = \begin{cases} \text{RZ}(x_1) & \text{RZ}(x_1) = \text{RZ}(x_2) \\ \text{RN}(x_1)^z & \text{RZ}(x_1) \neq \text{RZ}(x_2) \text{ but } \text{RN}(x_1) = \text{RN}(x_2) \\ \perp & \text{otherwise} \end{cases}$$

As a convention, v^u, v^d, v^z denote the successor, the predecessor and the next floating point value toward 0.

Figure 1: Faithful and Directed Faithful Rounding

our approach herein is more simply to obtain at least a faithfully rounded dot product.

The notion of faithful rounding was introduced in [6] and in [10] as *optimal* rounding. Our proposal, in [5], gives equivalent results on a general mechanism well suited to be a first extension of the IEEE deterministic rounding. The notation Rxy, where x and y may be one of the letters N, D, U or Z, is proposed to extend these conventions to the rounding of intervals. We should eventually propose five "standard" rounding modes RUU, RDD, RZZ, RND and RNU to correspond to the existing modes RU, RD, RZ and RN for standard arithmetic operations (see Fig 1).

Faithful Rounding The RND rounding mode is active; if the interval cannot be rounded with the primary rounding mode, Round to Nearest, the secondary mode is used to round the interval Down towards $-\infty$. An example of RND faithful rounding of an interval is presented Fig 2.

Definition 1 *The interval* $[x_1, x_2]$ *Rounds Nearest-Down (RND) to* v *if it Rounds to Nearest to* v *or else it Rounds Down to* v.

On a process that ensures faithful rounding (RND or RNU), the machine guarantees the IEEE standard

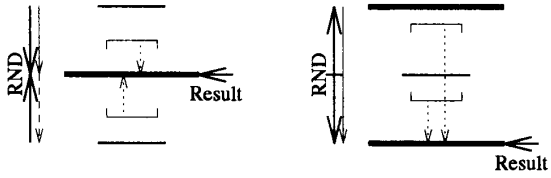

Figure 2: Rounding Nearest-Down Mode

rounding of the infinitely precise result in the primary mode (RN) or the secondary mode (RD or RU). Although the user cannot *a priori* impose which mode will be used, *a posteriori* information on the mode used is available by just checking the IEEE standard rounding flag which may be set by the computer. If the standard rounding flag is active, the primary mode was used, otherwise, the result was rounded with the secondary mode.

Directed Faithful Rounding It is first desirable to extend the notion of faithful rounding by introducing directed faithful rounding. The directed rounding modes allow interval arithmetic implementation at the user level. This is to guarantee that accumulated error during the rounding process cannot make the user lose the knowledge that the result is a directed approxima-

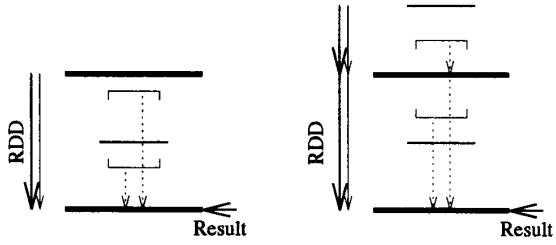

Figure 3: Rounding Down-Down Mode

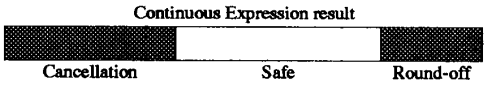

Figure 4: Loss of Precision in a Continuous Expression

tion of the infinitely precise result.

Definition 2 *The interval* $[x_1, x_2]$ *Rounds Up-Up (RUU) to* v *if it Rounds Up to* v *or else it Rounds to Nearest to* v^d.

Definition 3 *The interval* $[x_1, x_2]$ *Rounds Down-Down (RDD) to* v *if it Rounds Down to* v *or else it Rounds to Nearest to* v^u.

Fig 3 presents the RDD rounding mode. The RZZ (Round Zero-Zero) mode may be obtained from the RDD and the RUU modes.

2 Catastrophic Cancellation

The exact semantic of each separate operation does not necessarily implies the correctness of a large compound operation. During the evaluation of a continuous expression, accuracy is lost, as presented in Fig 4, from accumulated round-off error and cancellation.

Some research has been intended to track lost accuracy during compound operations: randomized mechanisms, including [13], are available to detect irrelevant results using redundant systems. This is currently a good approach to the problem, but it is not commonly available. The user must have specially built chips, and the computation must be repeated from five to ten times to get a good probability of correctness.

Simple Cancellation Lots of cancellation occur during a large computation. Detecting every cancellation of a single bit would lead to tag any result as

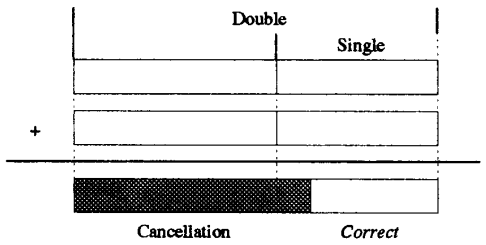

Figure 5: Catastrophic Cancellation as a Reduction of Precision from Double to Single

	Mantissa length	Catastrophic cancellation
Half-Single	11	
Single	24	13
Double	53	29
Quad	112	59

Table 2: Number of bits lost on a catastrophic cancellation

suspect. On the other hand, an operation barely cancels all the bits of a result, and a mechanism to detect such cases would be useless. The IEEE standard introduced the notion of hierarchy among the precision types (single, double, quad...). This hierarchy gives the trade-off between a too tight definition of the catastrophic cancellation and one which would never occur.

Definition 4 *A cancellation is considered as a reduction in the number of significant mantissa bits of the floating point representation. A catastrophic cancellation has occurred if enough bits have been lost in the process so that the result's mantissa is no longer than a mantissa of the next lower precision.*

The notion of catastrophic cancellation and the problem of rounding are not related. Doing the subtraction of two numbers close to each other will usually lead to a catastrophic cancellation. Although the result is the standard rounding of the infinitely precise mathematical operation on the inputs. Catastrophic cancellation measures the trust one can place on the physical interpretation of the result.

Cancellation of an Accumulation In order to get information on the validity and the precision of an accumulation, we consider an accumulation as an atomic operation and not as an ordered sequence of partial

sums. Extending the definition of catastrophic cancellation to a dot product accumulation is possible:

- The catastrophic cancellation of a single operation can be defined as a mantissa shift. In double precision, this condition is equivalent on the exponent e_1 and e_2, of the inputs x_1 and x_2 respectively, and on the exponent e_s of the sum to

$$e_s \geq \max(e_1, e_2) + 29$$

  The condition for a catastrophic cancellation of the accumulation of n partial product would be

$$e_a \geq \max(e_i)_{i=1}^{i=n} + 29$$

- The notion of cancellation is also related to the rounding error introduced by the floating point notation. A catastrophic cancellation is a cancellation that just gives one ulp precision on the next lower format. For an accumulation, the catastrophic cancellation should be equivalent to

$$e_a \geq \max(e_i)_{i=1}^{i=n} + 30 - \lceil \log n \rceil$$

However, these methods are very expensive because they require extra storage to compute the value of the highest exponent. Still, this characterization of the cancellation is minimal and order independent. Any process that implements catastrophic cancellation detection should at least detect all the cases that are described in one of these two methods.

3 Validated Dot Product

To obtain some information about the dot product we consider it as an order dependent atomic operation. There are two possibilities to handle such a case using the common instruction set available on computers and coprocessors. These two methods conceptually define the same function

$$acc : \coprod_{n \in N} \mathbf{F}^n \longrightarrow \overline{\mathbf{F}}$$

- The first method is close to the general purpose computer architecture. The dot product is obtained by issuing the following stream of operations.

```
reset_acc (X);
acc (X, x[1]);
...
acc (X, x[n]);
t = value_acc (X, n);
```

Input: A vector of floating point values $[x_i]_1^n$
Output: An approximation of $\sigma = \sum_1^n x_i$
A bound on the error δ
The level of cancellation ν

Set the two registers X^+ and X^- to 0.0
Repeat until all the vector has been read
Read a new value x_k
If $x_k \geq 0$ then $X^+ := X^+ + x_k$
else $X^- := X^- + x_k$
Return
The sum $X = X^+ + X^-$
The error bound
$\delta = (n-1)\max(\text{ulp}(X^+), \text{ulp}(X^-))$
The level of cancellation
ν: number of bit canceled
during the last operation

Figure 6: SSA — Sign Segregated Accumulation

- The second method would be implemented on some specific application oriented designs only with extra long instructions.

```
t = long_acc (x[1:n], 1, n);
```

3.1 Sign Segregated Accumulation (SSA)

Error control Our approach to attain faithful rounding involves control of the accumulated error. A floating point number v is known with an error δ given the active rounding mode Rx, if the real value x is in the interval

Round to Nearest	$x \in [v - \frac{1}{2}\delta, v + \frac{1}{2}\delta]$
Round Up towards $+\infty$	$x \in [v - \delta, v]$
Round Down towards $-\infty$	$x \in [v, v + \delta]$
Round toward Zero	$x \in [v - \delta, v + \delta]$

The RZ mode does not work correctly in regard to this specification and we have used the maximum interval.

Lemma 1 *The error on the sum of two numbers x_1 and x_2 known with respective error $\delta_1 \leq n_1 ulp(x_1)$ and $\delta_2 \leq n_2 ulp(x_2)$ is bounded by the value*

$$\delta = (n_1 + n_2 + 1)\max(ulp(x_1), ulp(x_2), ulp(x_1 + x_2))$$

The SSA Algorithm Computing the sum X^+ of the positive numbers and the sum X^- of the negative numbers (see Fig 6) ensures that the whole cancellation occurs on the last operation $X = X^+ + X^-$. This methods also guarantees that the values $\text{ulp}(X^+)$ and $\text{ulp}(X^-)$ are non decreasing, thus the accumulated error is bounded relatively to the final values of X^+ and X^-.

Theorem 1 *The error introduced by the Signed Segregated Accumulation (SSA) process is bounded by the value*

$$(n-1) \times \max\left(ulp(X^+), ulp(X^-)\right)$$

This method represents the worst case accumulation in regard to catastrophic cancellation. At the hardware level, the SSA mechanism is interesting because the selection of the register X^+ or X^- can be done in the early stages of a pipeline, and the right register would be ready on time.

Extended Type As Kahan proposed in [8], the accumulation process should take place in a type precision higher than the working type. The quad precision format represents a nice trade-off to accumulate dot product in double precision. Being a standard, this format will be accepted by manufacturers, moreover it is integrated into the IEEE philosophy, and any floating point unit that would give wider data path than the double precision operations will implement double extended operations and most certainly quad addition.

Considering the performances of the machines now available to the users, and the problem that are run on those machines, one can argue that one billion terms ($10^9 \equiv 2^{30}$) is a reasonable upper limit on the dot product operation. Thus the accumulated error cannot be bigger than 30 bits.

If no catastrophic cancellation has occurred, the last operation has canceled less than 29 bits, and thus $54 = 112-(28+30)$ bits at least of the result are available. This means one ulp precision, thus the result can be faithfully rounded.

Otherwise, a catastrophic cancellation has occurred. In this case, we strongly ask the user to check his code.

Interval Results Each function returns one unique fully specified floating point number. The output of the SSA process is the infinitely precise rounding of the sum $X^+ + X^-$. The knowledge on the accumulated error can only be used to determine the degree of precision and set the exception flags: IEEE standard rounding, faithful rounding, no guarantee.

If the result is neither the standard rounding, nor the faithful rounding of the implicit interval, the user should get the bounds of the interval $[m, M]$ within which the result is known to lie. Two more functions are supported to provide m and M when the hardware signals a possible loss of accuracy. It is the responsibility of the user to trap on non faithful rounding and to ask the floating point unit for the bounds.

Input: A vector of floating point values $[x_i]_1^n$
Output: An approximation of $\sigma = \sum_1^n x_i$
A bound on the error δ
The level of cancellation

Set one sticky accumulator X to 0.0
Repeat until all the vector has been read
Read a new value x_k
$X := X + x_k$
Return
The normalized value of X
The error bound $\delta = (n-1) \times \text{ulp}(X)$
The level of cancellation of X

Figure 7: SA — Sticky Accumulation

3.2 Sticky Accumulation (SA)

Sticky Accumulator A sticky accumulator X is a device that forces the useful property of the segregated accumulation for any floating point value t

$$\text{ulp}(X + t) \geq \max(\text{ulp}(X), \text{ulp}(t))$$

Here, X represents a word and not its real value; similarly, $X + t$ represents the word obtained by the accumulation of t to the sticky accumulator X. A quad precision floating point register (with the implicit bit made explicit) that is never normalized by shifting left is a sticky accumulator. In this case, we allow the mantissa of the accumulator to be smaller than 1.

During an addition, the only process where the exponent may decrease is the left normalization, since it is impossible, the exponent of the result has to be larger than both the exponent of the two inputs, and

so is the value of an ulp. The functionalities that are needed on a sticky accumulator are:

- Accumulate a new value.
- Get the level of cancellation (the number of leading bits set to 0).
- Get the value of an ulp (computed from the exponent of the accumulator).
- Normalize and lose the information on the sticky accumulation.

The SA Algorithm The machine computes the partial sum

$$X_m = \sum_{i=1}^{i=m} x_i$$

in quad precision with a sticky accumulator. The result is the properly normalized value of X_n in double format. During the last normalization, the system has access to the useful data to set the IEEE standard rounding flag, the faithful rounding flag and the catastrophic cancellation flag.

Theorem 2 *The error introduced by the Sticky Accumulation (SA) process is bounded by the value*

$$(n-1) \times ulp(X)$$

We present Fig 8 to illustrate the case of a decimal accumulation with three significant digits rounded to nearest. The first case is the exact sum of the inputs. This result may be produced by a fixed point process. The second sum is a standard accumulation, it yield 0 because of cancellation. The user does not have either a rough result or some information on the error. In the last case, we used sticky accumulation with one extra digit. As a result, we got better accuracy; moreover, an error bound is available through the process.

Counting With little extra hardware, the floating point unit is able to compute the IEEE standard rounding flag and the faithful rounding flag with higher accuracy. The computer may yield the correct result of an operation but still tag it as unsafe because the implicit interval has been over rated. The counting mechanism offers better control over the bound of the accumulated error.

We include in the floating point unit a counter that is connected to the sticky accumulator. This counter supports two operations : increment by one unit and shift right by s bits. Whenever X_m is shifted right by s bits to compute X_{m+1} (exponent comparison or carry ripple), the counter is also shifted by s bits. After each addition, the counter is incremented by one unit. The value of the counter at the end of the SA process bounds the error in place of $(n-1) \times \text{ulp}(X)$.

		Exact sum	Standard sum (3 digits)	Sticky accumulation (4 digits)
	458.0	458.0	458.0	458.0
+	0.2	458.2	458.0	458.2
+	600.0	1058.2	1060.0	1058.0
+	-1060.0	-1.8	0	-0002.0
Error bound		0000.0	???	0001.5

Figure 8: Decimal Example of the Three Sum Processes

Exact flag Another improvement over the standard SA algorithm is possible that does not require any extra hardware. The *exact* flag specified by the IEEE standard is active whenever the last operation has returned the exact infinitely precise result (no rounding). As long as all the sums $\{X_i\}_{i=1...m}$ have been performed exactly, there is no need to block the normalization of X_m. If a counter was implemented, it is not incremented before the m^{th} addition. The users gets in this case more precision and more control over the final result. At the end of the accumulation is possible to set the exact flag if no rounding was ever necessary.

Scalability We have presented this algorithm to implement double precision sum with one quad precision sticky accumulator. This algorithm can be extended to any length accumulator. The choice of the size of the accumulator should be free to the user. To improve the precision on non faithful result, the user should use a triple or quadruple length accumulator.

Depending on the knowledge the user has over his algorithm stability, he may also choose a long (fixed point) accumulator mechanism. However, a first pass of the SA dot product will detect the suspect cases, and the expensive process of the multiple length accumulation will be used only when it is really needed.

Conclusion

More than any other dot product algorithm, we have presented a *reliable* and efficient dot product. This algorithm is associated with an error detection mechanism, hence it is able to perform at the level of accuracy desired by the user. With almost no time penalty compared to the trivial double precision accumulation which carries no control on the result, one can efficiently monitor the error and use the relatively slow full length accumulation only in the case where the huge amount of information it carries is really necessary. Moreover on double precision operation, the algorithm guarantees, with at least a quad precision sticky accumulator, a faithfully rounded result unless a catastrophic cancellation has occurred.

References

[1] G. Bohlender, "Floating point computation of functions with maximum accuracy," *IEEE Transaction on Computer*, C26, 1977, pp. 621-632.

[2] G. Bohlender, "What do we need beyond IEEE arithmetic ?," *Computer Arithmetic and Self Validating Numerical Methods*, pp. 1-32, Academic Press, 1990.

[3] P.R. Cappello and W.L. Miranker, "Systolic super summation," *IEEE Transaction on Computer*, EC-37 (6), June 1988, pp. 657-677.

[4] J.T. Coonen, "Specification for a proposed standard for floating point arithmetic," *University of California, Berkeley*, Mem. UC8/ERL M78/72, 1978

[5] — "Rounding of floating point intervals," *Laboratoire de l'Informatique du Parallélisme* RR 93-06, March 1993

[6] T.J. Dekker, "A Floating Point Technique for Extending the Available Precision," *Numerische Mathematik*, Vol. 18, 1971, pp. 224-242.

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

[8] W. Kahan "Further Remarks on Reducing Truncation Errors," *Communication of the ACM* Vol. 8, 1965, p. 40

[9] R. Kirchner and U. Kulisch, "Arithmetic for vector processors," *8th IEEE Symposium on Computer Arithmetic*, 1987, pp. 256-269.

[10] U. Kulisch, "An axiomatic approach to rounded computations," *Numerische Mathematik*, Vol. 19, 1971, pp. 1-17

[11] A. Knöfel, "Fast hardware units for the computation of accurate dot products," *10th IEEE Symposium on Computer Arithmetic*, 1991, pp. 70-74.

[12] U. Kulisch and W.L. Miranker, "Computer arithmetic in theory and practice," Academic press, 1981.

[13] M. La Porte and J. Vignes, "Etude statistique des erreurs dans l'arithmétique des ordinateurs, application au contrôle des résultats d'algorithmes numériques," *Numerische Mathematik*, Vol. 19, 1972, pp. 400-406.

[14] M. Müller, C. Rüb and W. Rülling, "Exact accumulation of floating-point numbers," *10th IEEE Symposium on Computer Arithmetic*, 1991, pp. 64-69.

[15] M. Pichat, "Correction d'une somme en arithmétique à virgule flottante," *Numerische Mathematik*, Vol. 19, 1972, pp. 400-406.

Proofs and Examples

A.1 Cancellation

Example The rounding mode is set to Round to Nearest, and the users is using double precision arithmetic. The *physical experiment* input values are x_1 and x_2.

$$\begin{array}{rclcrcl} x_1 & = & 1+2^{-40} & \Rightarrow & \mathrm{RN}(x_1) & = & x_1 \\ x_2 & = & -1+2^{-60} & \Rightarrow & \mathrm{RN}(x_2) & = & -1 \end{array}$$

The best result an accumulation process may produce is x. However getting x as an answer, the user would assume that all the bits of x are correct in regard to the values from the experiment. The value x_0 is the one that would correspond to this assumption.

$$\begin{array}{rcll} x & = & \mathrm{RN}(\mathrm{RN}(x_1)+\mathrm{RN}(x_2)) & = & 2^{-40} \\ x_0 & = & \mathrm{RN}(x_1+x_2) & = & 2^{-40}+2^{-60} \end{array}$$

More than the difference $x - x_0$, the fact that x_1 and x_2 are very close with a relative difference of 2^{-40}

has lead to this cancellation effect. Since rounded to single precision, the result still pretends more information than it really contains, this is a catastrophic cancellation.

A.2 Sign Segregated Accumulation

Proof of the Lemma 1 We present the proof only for the Round to Nearest mode. Let t_1 and t_2 represent the true values of the inputs, $t = t_1 + t_2$ the value of the sum, $x_s = x_1 + x_2$ the temporary sum, and $x = \mathrm{RN}(x_s)$ the rounded sum.

$$\begin{array}{rcccl} x_1 - \frac{1}{2}\delta_1 & \leq & t_1 & \leq & x_1 + \frac{1}{2}\delta_1 \\ x_2 - \frac{1}{2}\delta_2 & \leq & t_2 & \leq & x_2 + \frac{1}{2}\delta_2 \\ x_s - \frac{1}{2}(\delta_1 + e_2) & \leq & t & \leq & x_s + \frac{1}{2}(\delta_1 + e_2) \\ x - \frac{1}{2}\mathrm{ulp}(x) & \leq & x_s & \leq & x + \frac{1}{2}\mathrm{ulp}(x) \end{array}$$

However

$$\begin{array}{ll} & \delta_1 + \delta_2 + \mathrm{ulp}(x) \\ \leq & n_1\mathrm{ulp}(x_1) + n_2\mathrm{ulp}(x_2) + \mathrm{ulp}(x) \\ \leq & (n_1 + n_2 + 1)\times \\ & \max(\mathrm{ulp}(x_1), \mathrm{ulp}(x_2), \mathrm{ulp}(x_1 + x_2)) \end{array}$$

Thus

$$\delta = (n_1 + n_2 + 1)\max(\mathrm{ulp}(x_1), \mathrm{ulp}(x_2), \mathrm{ulp}(x_1 + x_2))$$

and

$$t \in \left[x_s - \frac{1}{2}\delta \ ; \ x_s + \frac{1}{2}\delta\right]$$

□

Proof of the Theorem 1 The proof mechanism is iterative. After m steps, X_m^+ has accumulated k values and X_m^- the $(m-k)$ other values. We suppose that the error on this two registers is bounded respectively by

$$\begin{array}{rcl} \delta_m^+ & = & (k-1) \times \mathrm{ulp}(X_m^+) \\ \delta_m^- & = & (m-k-1) \times \mathrm{ulp}(X_m^-) \end{array}$$

We can suppose that $x_{m+1} \geq 0$, then X_{m+1}^+ is known with an error bounded by

$$\delta_{m+1}^+ \geq k \times \max(\mathrm{ulp}(X_m^+), \mathrm{ulp}(x_m), \mathrm{ulp}(X_{m+1}^+))$$

But $X_{m+1}^+ \geq \max(x_m, X_m^+)$ thus the uncertainty on X_{m+1}^+ is bounded by

$$\delta_{m+1}^+ = k \times \mathrm{ulp}(X_{m+1}^+)$$

The iteration implies for $m = n$, that X_n^+ and X_n^- are known with a maximum error of

$$\begin{array}{rcl} \delta_n^+ & = & (k-1) \times \mathrm{ulp}(X_n^+) \\ \delta_n^- & = & (n-k-1) \times \mathrm{ulp}(X_n^-) \end{array}$$

The sum is known with an error at most

$$(n-1) \times \max(\mathrm{ulp}(X_n^+), \mathrm{ulp}(X_n^-), \mathrm{ulp}(X_n^+ + X_n^-))$$

but X_n^+ and X_n^- are of opposite sign, thus

$$|X_n^+ + X_n^-| \leq \max(|X_n^+|, |X_n^-|)$$

□

A.3 Sticky Accumulation

Proof of the Theorem 2 The proof mechanism is iterative and very close to the one presented for the sign segregated accumulation. After m steps, we suppose that the error on X_m is bounded by

$$\delta_m = (m-1) \times \mathrm{ulp}(X_m)$$

Then X_{m+1} is known with an error bounded by

$$\delta_{m+1} \geq m \times \max(\mathrm{ulp}(X_m), \mathrm{ulp}(x_m), \mathrm{ulp}(X_{m+1}))$$

But

$$\mathrm{ulp}(X_{m+1}) \geq \max(\mathrm{ulp}(X_m), \mathrm{ulp}(x_m))$$

Thus the uncertainty on X_{m+1} is bounded by

$$\delta_{m+1} = m \times \mathrm{ulp}(X_{m+1})$$

The iteration implies for $m = n$, that X_n is known with a maximum error of $(n-1) \times \mathrm{ulp}(X_n)$.

□