3-1-1999

On Average Bit Complexity of Interval Arithmetic

Chadi Hamzo

Vladik Kreinovich
University of Texas at El Paso, vladik@utep.edu

Comments:
UTEP-CS-99-20.
Published in: *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 1999, Vol. 68, pp. 153-156.

On Average Bit Complexity of Interval Arithmetic

Chadi Hamzo and Vladik Kreinovich

Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
emails {chamzo,vladik}@cs.utep.edu

Abstract

In many practical situations, we know only the intervals which contain the actual (unknown) values of physical quantities. If we know the intervals $\mathbf{x}$ for a quantity x and $\mathbf{y}$ for another quantity y, then, for every arithmetic operation $\odot$, the set of possible values of $x \odot y$ also forms an interval; the operations leading from $\mathbf{x}$ and $\mathbf{y}$ to this new interval are called *interval arithmetic* operations. For addition and subtraction, corresponding interval operations consist of two corresponding operations with real numbers, so there is no hope of making them faster.

The best known algorithms for interval multiplication consists of 3 real-number multiplications and several comparisons. We describe a new algorithm for which the average time is equivalent to using only 2 multiplications of real numbers.

What is interval arithmetic. Many computer algorithms for processing real numbers have been designed to process measurement results. Measurements are never 100% precise; therefore, when after measuring a physical quantity we get the measurement result $\widetilde{x}$, this does not mean that the actual value of this quantity is equal to $\widetilde{x}$: this actual value can take any value from the *interval* $\mathbf{x} = [\widetilde{x} - \Delta, \widetilde{x} + \Delta]$, where Δ is an upper bound on the measurement error (usually supplied by the manufacturer of the measuring instrument).

If for two quantities x and y, we only know the intervals $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$ of possible values, then, for every arithmetic operation $\odot$ $(+, -, \cdot, /)$, we can only only conclude that $x \odot y \in \mathbf{x} \odot \mathbf{y}$, where $\mathbf{x} \odot \mathbf{y}$ is defined as

$$\mathbf{x} \odot \mathbf{y} = \{x \odot y \,|\, x \in \mathbf{x}, y \in \mathbf{y}\}.$$

Thus defined arithmetic operations on the set of all intervals are called *interval arithmetic*.

The results of interval arithmetic operations can be easily described in terms of lower and upper endpoints of the operand intervals:

$$[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}];$$

$$[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}],$$

$$[\underline{x}, \overline{x}] \cdot [\underline{y}, \overline{y}] = [\min(\underline{x} \cdot \underline{y}, \underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}), \max(\underline{x} \cdot \underline{y}, \underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y})];$$

$$[\underline{x}, \overline{x}]/[\underline{y}, \overline{y}] = [\underline{x}, \overline{x}] \cdot \left[\frac{1}{\overline{y}}, \frac{1}{\underline{y}}\right] \text{ when } 0 \notin [\underline{y}, \overline{y}].$$

(For details, see, e.g., [6, 7, 9, 10, 11].)

What is computational complexity of interval arithmetic? Formulation of the problem. When we process intervals corresponding to data uncertainty, our goal is to perform the corresponding interval arithmetic operations. The faster we perform them, the better, so the natural question is: what is the algebraic complexity of these operations? In other words, how many elementary operations with real numbers (actually, rational numbers) do we need to perform to implement one interval arithmetic operation?

For addition, the answer is easy: addition of two intervals means two additions: of lower endpoints and of upper endpoints. Similarly, subtraction of two intervals means two subtractions of real numbers. Division of interval reduces to multiplication, so the only non-trivial problem is: what is the algebraic complexity of interval multiplication?

In the above formula, we use 4 multiplications and several comparisons (to find min and max). Since multiplication requires asymptotically more time than comparison (for n-digit numbers, comparison takes n bit operations while known multiplication algorithms requires n^2 or at least $O(n^{\log_2(3)})$; see, e.g., [5], Ch. 29

and 32), it is reasonable to take the number of multiplications as the measure of the computation time (ignoring the number of comparisons).

Known result: often, we only need two real-number multiplications. Before we describe the new algorithm, let us write down the known reformulation of the interval product formula. Namely, instead of a single formula which covers all possible signs of the four real numbers $\underline{x}$, $\overline{x}$, $\underline{y}$, and $\overline{y}$, we can describe separate formulas for different sign combinations:

- if $\underline{x} \geq 0$ and $\underline{y} \geq 0$, then $\mathbf{x} \cdot \mathbf{y} = [\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}]$;
- if $\underline{x} \geq 0$ and $\underline{y} \leq 0 \leq \overline{y}$, then $\mathbf{x} \cdot \mathbf{y} = [\overline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}]$;
- if $\underline{x} \geq 0$ and $\overline{y} \leq 0$, then $\mathbf{x} \cdot \mathbf{y} = [\overline{x} \cdot \underline{y}, \underline{x} \cdot \overline{y}]$;
- if $\underline{x} \leq 0 \leq \overline{x}$ and $\underline{y} \geq 0$, then $\mathbf{x} \cdot \mathbf{y} = [\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}]$;
- if $\underline{x} \leq 0 \leq \overline{x}$ and $\underline{y} \leq 0 \leq \overline{y}$, then $\mathbf{x} \cdot \mathbf{y} = [\min(\underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}), \max(\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y})]$;
- if $\underline{x} \leq 0 \leq \overline{x}$ and $\overline{y} \leq 0$, then $\mathbf{x} \cdot \mathbf{y} = [\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}]$;
- if $\overline{x} \leq 0$ and $\underline{y} \geq 0$, then $\mathbf{x} \cdot \mathbf{y} = [\underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}]$;
- if $\overline{x} \leq 0$ and $\underline{y} \leq 0 \leq \overline{y}$, then $\mathbf{x} \cdot \mathbf{y} = [\underline{x} \cdot \overline{y}, \underline{x} \cdot \underline{y}]$;
- if $\overline{x} \leq 0$ and $\overline{y} \leq 0$, then $\mathbf{x} \cdot \mathbf{y} = [\overline{x} \cdot \overline{y}, \underline{x} \cdot \overline{y}]$.

We see that in 8 out of 9 cases, we need only 2 multiplications, and the only case when we really need 4 multiplications is when $0 \in \mathbf{x}$ and $0 \in \mathbf{y}$. Let us show that in this case, 3 multiplications is enough.

In all cases, 3 real-number multiplications are sufficient: Heindl's fast algorithm [8]. In the case when $0 \in \mathbf{x}$ and $0 \in \mathbf{y}$, we have 4 non-negative numbers: $|\underline{x}| = -\underline{x}$, $|\underline{y}| = -\underline{y}$, $\overline{x}$, and $\overline{y}$. Let us show that for all four possible combinations of orders between $|\underline{x}|$ and $\overline{x}$, and between $|\underline{y}|$ and $\overline{y}$, only 3 multiplications will be needed. Indeed:

If $0 \leq |\underline{x}| \leq \overline{x}$ and $0 \leq |\underline{y}| \leq \overline{y}$, then, multiplying these two inequalities, we conclude that $|\underline{x}| \cdot |\underline{y}| \leq \overline{x} \cdot \overline{y}$. Hence, $\underline{x} \cdot \underline{y} \leq \overline{x} \cdot \overline{y}$, and $\max(\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}) = \overline{x} \cdot \overline{y}$. So, to compute the upper endpoint of the interval $\mathbf{x} \cdot \mathbf{y}$, we only need 1 multiplication. Together with 2 multiplications to compute the lower endpoint, we get a total of 3 multiplications.

If $0 \leq \overline{x} \leq |\underline{x}|$ and $0 \leq \overline{y} \leq |\underline{y}|$, then, multiplying these tow inequalities, we conclude that $\overline{x} \cdot \overline{y} \leq |\underline{x}| \cdot |\underline{y}|$. Hence, $\overline{x} \cdot \overline{y} \leq \underline{x} \cdot \underline{y}$, and $\max(\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}) = \underline{x} \cdot \underline{y}$. So, to compute the upper endpoint of the interval $\mathbf{x} \cdot \mathbf{y}$, we only need 1 multiplication. Together with 2 multiplications to compute the lower endpoint, we get a total of 3 multiplications.

If $0 \leq |\underline{x}| \leq \overline{x}$ and $0 \leq \overline{y} \leq |\underline{y}|$, then, multiplying these tow inequalities, we conclude that $|\underline{x}| \cdot \overline{y} \leq \overline{x} \cdot |\underline{y}|$. Hence, $\underline{x} \cdot \overline{y} \geq \overline{x} \cdot \underline{y}$, and $\min(\underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}) = \overline{x} \cdot \underline{y}$. So, to compute the lower endpoint of the interval $\mathbf{x} \cdot \mathbf{y}$, we only need 1 multiplication. Together with 2 multiplications to compute the upper endpoint, we get a total of 3 multiplications.

If $0 \leq \overline{x} \leq |\underline{x}|$ and $0 \leq |\underline{y}| \leq \overline{y}$, then, multiplying these tow inequalities, we conclude that $\overline{x} \cdot |\underline{y}| \leq |\underline{x}| \cdot \overline{y}$. Hence, $\overline{x} \cdot \underline{y} \geq \underline{x} \cdot \overline{y}$, and $\min(\underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}) = \underline{x} \cdot \overline{y}$. So, to compute the lower endpoint of the interval $\mathbf{x} \cdot \mathbf{y}$, we only need 1 multiplication. Together with 2 multiplications to compute the upper endpoint, we get a total of 3 multiplications.

In all four cases, 3 real-number multiplications is enough.

Formulation of the problem. A natural question is: can we compute interval multiplication still faster? If we cannot perform interval multiplication faster in *all* cases, maybe we can perform it faster *on average*? This is the question to which we answer in this paper. Namely, we describe a new algorithm for which the average time is equivalent to using only 2 multiplications of real numbers.

The idea behind our new algorithm. According to the formula for interval multiplication, each of the endpoints is equal to the product of two original endpoints; so, if we knew which original endpoints we have to multiply, then *two* real-number multiplications would be sufficient. The reason why we have to consider *three* real-number multiplications is that we do not know which original endpoints to multiply: e.g., when $0 \leq -\underline{x} = |\underline{x}| \leq \overline{x}$ and $0 \leq -\underline{y} = |\underline{y}| \leq \overline{y}$, to compute the lower endpoint of the interval $\mathbf{x} \cdot \mathbf{y}$, we must compute the smallest $\min(\underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y})$ of the two products $\underline{x} \cdot \overline{y}$ and $\overline{x} \cdot \underline{y}$. If we could determine which of these two products is smaller without actually computing both of them, we would have saved time.

We would therefore like to check the sign of the inequality $\underline{x} \cdot \overline{y} \ ? \ \overline{x} \cdot \underline{y}$, i.e., the sign of the difference $D = \underline{x} \cdot \overline{y} - \overline{x} \cdot \underline{y}$.

From the mathematical viewpoint, the difference D is the *determinant* of the matrix

$$\begin{pmatrix} \underline{x} & \overline{x} \\ \underline{y} & \overline{y} \end{pmatrix}.$$

In computational geometry, there are known fast algorithms which check the sign of the determinant faster than known algorithms for computing it (see, e.g., [1, 2, 3, 4, 12]). These algorithms are not directly applicable in our case, because they cut down on the *algebraic* complexity (number of algebraic operations – real-number multiplications and additions) in the limit when the size of the matrix increases, while we are interested in cutting down the *bit* complexity for the simplest case of a 2×2 matrix. However, these algorithms inspired us to produce a similar algorithm for our case as well.

New algorithm: discussion. We will describe an auxiliary algorithm which, given an arbitrary four n-digit real numbers a, b, c, and d, checks whether $a \cdot b \le c \cdot d$, and whose average running time is bounded by a constant which is independent on n.

If we use this auxiliary algorithm to check which of the two products is larger, then, on average, we will get a new algorithm for interval multiplication, for which the average time is asymptotically equivalent to using only 2 multiplications of real numbers.

Comparing the product of real numbers is equivalent to comparing their absolute values; therefore, without losing generality, we can assume that all four real numbers are non-negative.

For simplicity, we will describe our new algorithm for the case when all the bounds are within the interval $[0, 1]$ and are represented as fixed-point real numbers (the asymptotic average complexity does not depend on this behavior). In this case, each number x is represented as a sequence of n bits $x = 0.x_1x_2 \ldots x_n$. For each n, by *average* complexity, we mean an arithmetic average over all possible combinations of binary digits (there are 2^n possibilities for each of the numbers, so totally, we have 2^{4n} possibilities).

New algorithm: description. Our auxiliary algorithm for checking whether $a \cdot b \le c \cdot d$ consists of several stages. The first stage is performed unconditionally; if on k-th stage, we have not yet found out whether $a \cdot b \le c \cdot d$, we perform stage # $(k+1)$, etc, until we reach stage # n. If on n-th stage, we still have not yet found out the answer, we then compute both products $a \cdot b$ and $c \cdot b$ and compare them.

On k-th stage, we compute the products $L_k = a^{(k)} \cdot b^{(k)}$ and $R_k = c^{(k)} \cdot d^{(k)}$ (where $x^{(k)}$ denotes $0.x_1 \ldots x_k$). Then:

- if $L_k + \varepsilon_k \le R_k$ (where we denoted $\varepsilon_k = 2^{-(k-1)} + 2^{-2k}$), then we conclude that $a \cdot b \le c \cdot d$;
- if $R_k + \varepsilon_k \le R_k$, then we conclude that $c \cdot d \le a \cdot b$;
- in all remaining cases, we move to the next stage.

For each k, to compute L_k and R_k, we use the fact that $x^{(k)} = x^{(k-1)} + x_k \cdot 2^{-k}$ and therefore, $L_k = a^{(k)} \cdot b^{(k)} = a^{(k-1)} \cdot b^{(k-1)} + 2^{-k} \cdot (a_k \cdot b^{(k)} + b_k \cdot a^{(k)}) = L_{k-1} + 2^{-k} \cdot (a_k \cdot b^{(k)} + b_k \cdot a^{(k)})$. Thus, when we already know L_{k-1}, computing L_k means three additions of $2k$-digit numbers, i.e., $O(k)$ steps.

This algorithm is correct. Indeed, by definition of $x^{(k)}$, we can conclude that $x^{(k)} \le x \le x^{(k)} + 2^{-k}$. Therefore, for non-negative numbers a and b, we conclude that

$$a^{(k)} \cdot b^{(k)} \le a \cdot b \le (a^{(k)} + 2^{-k}) \cdot (b^{(k)} + 2^{-k}) = a^{(k)} \cdot b^{(k)} + 2^{-k} \cdot (a^{(k)} + b^{(k)}) + 2^{-2k}.$$

From $a^{(k)} \le 1$ and $b^{(k)} \le 1$, we can now conclude that $a \cdot b \le a^{(k)} \cdot b^{(k)} + \varepsilon_k = L_k + \varepsilon_k$. So, $L_k \le a \cdot b \le L_k + \varepsilon_k$; similarly, $R_k \le c \cdot d \le R_k + \varepsilon_k$. Thus, e.g., from $L_k + \varepsilon_k \le R_k$, we can indeed conclude that $a \cdot b \le c \cdot d$.

Computing asymptotic average bit complexity: main idea of the proof. The average bit complexity can be expressed as $A = \sum p_k \cdot A_k$, where p_k is the probability that we need k stages, and A_k is the average number of bit operations for the situation when we need k stages.

Each stage requires $O(k) \le C_1 \cdot k$ bit operations. Thus, if the algorithm stops on k-th stage, we will need a total of $\le C_1 \cdot (1 + 2 + \ldots + k) \le C_1 \cdot k^2$ bit operations. So, $A_k \le C_1 \cdot k^2$.

Let us now estimate the probability p_k that we will need k stages. We need k stages if we did not get an answer on the previous stage, i.e., if $|a^{(k-1)} \cdot b^{(k-1)} - c^{(k-1)} \cdot d^{(k-1)}| \le \varepsilon_{k-1}$. When the number $a = 0.a_1 \ldots a_n$ takes all 2^n possible values with equal probability, then, for each k, the value $a^{(k-1)} = 0.a_1 \ldots a_{k-1}$ also takes all 2^{k-1} possible values with equal probability. So, the probability that we will not get an answer on

$(k-1)$-st step is equal to the probability that for randomly chosen values $a^{(k-1)}$, $b^{(k-1)}$, $c^{(k-1)}$, and $d^{(k-1)}$, the above inequality holds.

When $k \to \infty$, the distribution of $a^{(k-1)}$ tends to a uniform distribution on the interval $[0,1]$. Thus, for large k, the probability of the above inequality is getting close to the probability that for uniformly distributed four numbers a, b, c, and d, we have $|a \cdot b - c \cdot d| \le \varepsilon_{k-1}$. This probability is equal to the 4-volume of the area; for $\varepsilon_k \to 0$, one can compute the corresponding integrals and show that this volume is asymptotically equal to $C_2 \cdot \varepsilon_k$, for some constant $C_2 > 0$, i.e., equivalently, to $\sim C_2 \cdot 2^{-k}$. Thus, the probability p_k that we need k stages is $\le C_2 \cdot 2^{-k}$.

From $p_k \le C_2 \cdot 2^{-k}$ and $A_k \le C_1 \cdot k^2$, we conclude that $A = \sum p_k \cdot A_k \le (C_1 \cdot C_2) \cdot \Sigma_n$, where $\Sigma_n = \sum_{k=1}^{n} 2^{-k} \cdot k^2$. Clearly, Σ_n is smaller than the corresponding infinite sum Σ_∞, and it is known that this infinite sum is finite. Thus, we have a bound on A (namely, $C_1 \cdot C_2 \cdot \Sigma_\infty$) which does not depend on n. The result is proven.

Warning. The fact that the new algorithm is theoretically faster does not necessarily mean that it is practically faster: on some computers, comparing two numbers takes the same time as multiplying them. However, we hope that this new algorithm will be helpful in designing computer hardware which supports interval arithmetic.

Acknowledgments. This research was partly supported by NASA under cooperative agreement NCC5-209, by NSF grants No. DUE-9750858 and CDA-9522207, by United Space Alliance, grant No. NAS 9-20000 (PWO C0C67713A6), by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518, and by the National Security Agency under Grant No. MDA904-98-1-0564.

One of the authors (V.K.) is thankful to Luc Jaulin, Sylvain Pion, and Jürgen Wolff von Gudenberg for valuable discussions.

References

[1] F. Avnaim, J. D. Boissonnat, O. Devillers, F. Preparata, and M. Yvinec, "Evaluating signs of determinants using single-precision arithmetic", *Algorithmica*, 1997, Vol. 17, pp. 111–132.

[2] H. Brönnimann, I. Emiris, V. Pan, and S. Pion, "Computing exact geometric predicates using modular arithmetic with single precision", *Proc. 13th Annual ACM Symposium on Computational Geometry*, 1997, pp. 174–182.

[3] H. Brönnimann and M. Yvinec, "Efficient exact evaluation of signs of determinants", *Proc. 13th Annual ACM Symposium on Computational Geometry*, 1997, pp. 166–173.

[4] K. L. Clarkson. "Safe and effective determinant evaluation", *Proc. 33rd IEEE Symposium on Foundations of Computer Science FOCS'92*, 1992, pp. 387–395.

[5] Th. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, MIT Press, Cambridge, MA, and Mc-Graw Hill Co., N.Y., 1994.

[6] R. Hammer, M. Hocks, U. Kulisch, D. Ratz, *Numerical toolbox for verified computing. I. Basic numerical problems*, Springer Verlag, Heidelberg, N.Y., 1993.

[7] E. R. Hansen, *Global optimization using interval analysis*, Marcel Dekker, N.Y., 1992.

[8] G. Heindl, *An improved algorithm for computing the product of two machine intervals*, Interner Bericht IAGMPI- 9304, Fachbereich Mathematik, Gesamthochschule Wuppertal, 1993.

[9] R. B. Kearfott, *Rigorous global search: continuous problems*, Kluwer, Dordrecht, 1996.

[10] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and feasibility of data processing and interval computations*, Kluwer, Dordrecht, 1997.

[11] R. Moore, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

[12] S. Pion, "Interval arithmetic: an efficient implementation and an application to computational geometry", *Proceedings of the Workshop on Applications of Interval Analysis to Systems and Control with special emphasis on recent advances in Modal Interval Analysis MISC'99*, Girona, Spain, February 24–26, 1999, pp. 99–109.