Tradeoffs between Accuracy and Efficiency for Optimized and Parallel Interval Matrix Multiplication

Hong Diep Nguyen[1], Nathalie Revol[2], and Philippe Théveny[3]

[1] Computer Science Division - University of California at Berkeley
Berkeley, USA
hdnguyen@eecs.berkeley.edu
[2] INRIA - Université de Lyon
LIP (UMR 5668 CNRS - ENS de Lyon - INRIA - UCB Lyon 1), Lyon, France
Nathalie.Revol@ens-lyon.fr,
WWW home page: http://perso.ens-lyon.fr/nathalie.revol
[3] ENS de Lyon - Université de Lyon
LIP (UMR 5668 CNRS - ENS de Lyon - INRIA - UCB Lyon 1), Lyon, France
Philippe.Theveny@ens-lyon.fr,
WWW home page: http://perso.ens-lyon.fr/philippe.theveny

Abstract. Interval arithmetic is mathematically defined as set arithmetic. For implementation issues, it is necessary to detail the representation of intervals and to detail formulas for the arithmetic operations. Two main representations of intervals are considered here: inf-sup and mid-rad. Formulas for the arithmetic operations, using these representations, are studied along with formulas that trade off accuracy for efficiency. This tradeoff is particularly blatant on the example of interval matrix multiplication, implemented using floating-point arithmetic: according to the chosen formulas, the efficiency as well as the accuracy can vary greatly in practice, and not necessarily as predicted by the theory. Indeed, theoretical predictions are often based on exact operations, as opposed to floating-point operations, and on operations count, as opposed to measured execution time. These observations and the recommendations that ensue are further obfuscated by considerations on memory usage, multithreaded computations... when these algorithms are implemented on parallel architectures such as multicores.

Keywords: interval arithmetic, verified computation, inf-sup representation, mid-rad representation, floating-point arithmetic, accuracy, overestimation, efficiency, BLAS, parallel implementation

1 Introduction

Interval arithmetic is a mean to implement operations on sets and not only on single real values or points. It is also considered as a tool of choice for verified computations. Indeed, the fundamental property of interval arithmetic is the

inclusion property: every computed result encloses the exact result. However, it suffers from two main drawbacks: lack of accuracy and lack of efficiency. We show here that a panel of formulas exists for the implementation of interval arithmetic. This panel offers a choice ranging from maximal accuracy to acceptable runtime.

More precisely, the problems one has to face are a lack of:

- *accuracy:* the computed interval result is an overestimation of the exact result, and this overestimation may be too crude;
- *efficiency:* each interval operation entails several operations on single values (e.g., 1 interval addition requires to perform 2 real additions) and this seems to be the necessary price to pay. Another source of slowdown lies in the fact that implementing interval arithmetic using floating-point arithmetic requires to change frequently the rounding modes, in order to compute an "outward" enclosure of the result, i.e. to preserve the inclusion of the exact result into the computed one. However, on most processors and programming languages, changing the rounding modes incurs flushing the pipeline and leads to severe slowdowns, up to 2 orders of magnitude, when compared to the (unverified) corresponding floating-point computation.

To remedy the problem of frequent changes of the rounding modes, to enable the use of optimized routines in linear algebra, several variants of the arithmetic operations have been proposed [7, 10, 8, 9, 11]. **The main goal of this paper is to show that there is a wide choice of formulas, each satisfying different requirements in terms of accuracy and efficiency, at least as far as the problem of multiplying two interval matrices is concerned.** These variants are listed in Section 2, they are based on two representations for intervals, the inf-sup and the mid-rad representations which are recalled. The precise meaning of "the best representation in floating-point arithmetic" will be defined. The use of these variants for interval matrix multiplication is presented in Section 3; the questions are: how far it is possible to reduce the number of rounding modes' changes, and whether it is possible to use floating-point BLAS routines. A comparison of the theoretical complexity (the number of floating-point operations), the observed efficiency and the attained accuracy (the overestimation between the best result in floating-point arithmetic and the computed result) is performed experimentally. Finally, some remarks and (preliminary) results on the parallelization of these algorithms for interval matrix multiplication are presented in Section 4: optimizing the memory usage enters the scene.

2 Interval arithmetic: inf-sup and mid-rad representations

2.1 Inf-sup and mid-rad representations

The mathematical definition of binary arithmetic operations, in interval arithmetic, considers intervals as sets:

$$\text{for any operation } \diamond, \text{ for any sets } A \text{ and } B,\ A \diamond B = \text{Hull}\{a \diamond b,\ a \in A,\ b \in B\}.$$

The "Hull" means that the result is the smallest enclosing interval. Formulas have been devised that translate this abstract definition into implementable expressions. These formulas rely on the chosen representation for intervals. We will restrict ourselves to the inf-sup and mid-rad representations.

Another limit in this paper is the use of bounded intervals only. We refer the interested reader to a discussion on the mailing list of the IEEE 1788 working group for the standardization of interval arithmetic [5, January-February 2012] about the mid-rad representation of unbounded intervals.

In what follows, the notations below will be used:

- $\mathbf{a}$ in boldface corresponds to an interval quantity (whether it is an interval, a vector with interval components or a matrix with interval components will be made clear by the context);
- $[\underline{a}, \bar{a}]$ with square brackets corresponds to an inf-sup representation with $\underline{a}$ as the infimum and $\bar{a}$ as the supremum;
- $< a_m, a_r >$ with angular brackets corresponds to a mid-rad representation, with a_m as the midpoint and a_r as the radius.

2.2 Arithmetic operations: formulas for the inf-sup and mid-rad representations

Formulas for arithmetic operations $+$, $-$ and $\times$ for both representations can be found in [7, pp. 7-8 and pp.22-23].
Each interval addition or subtraction requires 2 real additions/subtractions. Each interval multiplication requires 4 real multiplications with the inf-sup representation and 6 real multiplications with the mid-rad representation. However, these formulas yield the same result when exact real arithmetic is employed.

Other formulas have been proposed [8–11] for the multiplication: they reduce the number of real operations, at the cost of an increase of the radius of the result. The ratio between the radius of the resulting interval and the radius of the exact interval is called *overestimation factor*. For the formulas mentioned above, these overestimation factors have been bounded, the bounds being $4 - 2\sqrt{2} \simeq 1.18$ for the inf-sup formulas in [8] and the mid-rad formulas in [9] and 1.5 for the mid-rad formulas in [10]. In [8] and [10] only 3 real products have to be performed, in [9] 5 real products are required.

2.3 Implementation using floating-point arithmetic

The overestimation ratios given above are established under the assumption that exact arithmetic is used. Let us focus now on what happens to the result of an interval operation when floating-point arithmetic is used for the implementation.

Floating-point arithmetic has been standardized as the IEEE-754 standard [3] and IEEE-754 2008 revision of the standard [4]. In particular, the standard specifies the result of an arithmetic operation between (one or) two floating-point operands: the operands are considered as exact values and the result is the exact result of the operation between these exact values, rounded to the closest floating-point number according to the prescribed rounding mode. Four rounding modes are defined by the IEEE-754 standard, three of them will be of use in this work: RN (*rounding-to-nearest*), RD (*rounding downwards*) and RU (*rounding upwards*).

A natural question when one implements interval arithmetic using floating-point arithmetic is "what is the best possible floating-point interval that represents a given interval $\mathbf{a}$?". Here, "best possible" means the smallest with regards to enclosure. The second result below is new.

Proposition 1 (Best floating-point interval for the inf-sup representation). *For the inf-sup representation,* $[\mathrm{RD}(\underline{a}), \mathrm{RU}(\bar{a})]$ *is the best floating-point interval that represents* $[\underline{a}, \bar{a}]$ *in the sense that for each floating-point interval* $[\underline{b}, \bar{b}]$ *such that* $[\underline{a}, \bar{a}] \subset [\underline{b}, \bar{b}]$ *we have* $[\mathrm{RD}(\underline{a}), \mathrm{RU}(\bar{a})] \subset [\underline{b}, \bar{b}]$.

Proposition 2 (Best floating-point interval for the mid-rad representation). *For the mid-rad representation, the best floating-point interval that represents the interval* $< a_m, a_r >$ *is the interval* $< c_m, c_r >$ *with*

$$\begin{aligned} c_m &= \mathrm{RN}(a_m), \\ c_r &= \mathrm{RU}(a_r + |c_m - a_m|). \end{aligned}$$

Every floating-point interval in mid-rad representation that contains $< a_m, a_r >$ *either contains* $< c_m, c_r >$ *or its midpoint is not* a_m *rounded to nearest.*

Note: In the case where a_m is the middle of two consecutive floating-point numbers, there are two best intervals in terms of smallest radius, but the tie rule determines the midpoint of one of them only.

These two theorems put in evidence the fact that, contrary to what happens in $\mathbb{R}$ where any representation corresponds to the same interval, in floating-point arithmetic the choice of the representation modifies the result. Indeed, when a mathematical $\mathbf{a}$ is rounded in floating-point arithmetic, the resulting interval depends on the chosen representation. In general, it holds that:

$$[\mathrm{RD}(\underline{a}), \mathrm{RU}(\bar{a})] \neq < \mathrm{RN}(a_m), \mathrm{RU}(a_r + |\mathrm{RN}(a_m) - a_m|) >$$

and it is generally not straightforward to determine a priori which interval is best. One can exhibit examples where one representation is better than the other for both cases.

In [11], formulas for the multiplication of two floating-point intervals in mid-rad representation are given. These formulas require 3 multiplication of floating-point numbers. They explicitely incorporate bounds on the roundoff errors, see notably [2] for an accessible and comprehensive analysis of roundoff errors in numerical computations.

3 Interval Matrix Multiplication

Notations: matrices denoted with capital letters. Matrices with interval coefficients denoted with boldface capital letters.

3.1 Why is it interesting to use the variants presented above?

Let us cite again [11] to explain why it is especially interesting to try different variants of the formulas for the arithmetic operations, when it comes to multiply two interval matrices. "*The classical approach* (i.e. the three nested loops using the classical formulas for the addition and multiplication) *is very slow on todays computers because i) it requires* $2n^3$ *times switching the rounding mode, thus ii) jeopardizing compiler optimization and iii) putting the burden of well-known acceleration techniques such as blocked code, parallel implementations etc. on the user.*" In other words, gain in efficiency is expected when trying other formulas, by factoring the changes of rounding modes, in order to insert changes of rounding modes around large blocks of computations which can be optimized using classical approaches. These large blocks of computations typically correspond to the routines implemented in BLAS, and thus the use of optimized BLAS, such as GotoBLAS or Atlas, directly benefits to the performances.

In what follows, we consider only algorithms for the product of two interval matrices. They are good candidates to put into practice the panel of existing formulas for arithmetic operations, as they are simple algorithms. Indeed, they offer a wide range of opportunities for optimization and parallelization, and still have a polynomial complexity.

Let us also insist on the fact that only the product of matrices based on the classical approach is considered, as fast matrix products are not suited to interval arithmetic, because they suffer from overestimation due to variable dependency [1]

3.2 Algorithms based on the inf-sup representation

The two algorithms considered here are on the one hand the so-called *classical approach*, where the three nested loops are explicitely written and a change of rounding mode is performed for each operation, at the very heart of these three loops. On the other hand, we also consider the algorithm presented in [8]. This algorithm adapts the formula for interval multiplication, with 3 real multiplications, to the case of interval matrix multiplication. The two notable features of this algorithm is that only 3 changes of the rounding modes are required, and that 9 products of floating-point matrices are used.

3.3 Algorithms for the mid-rad representation

The algorithms considered here are:

- a counterpart of the classical approach, where each operation between two intervals is performed using Neumaier's formulas [7, p. 22-23], which incur no overestimation in exact arithmetic;
- the algorithm presented in [9], which uses 7 products of floating-point matrices and has a theoretical overestimation ratio (i.e. without considering roundoff errors) of 1.18;
- the algorithm presented in [10], which uses 4 products of floating-point matrices and has a theoretical overestimation ratio (i.e. without considering roundoff errors) of 1.5;
- the algorithm presented in [11], which uses 3 products of floating-point matrices and explicitely incorporates a bound on roundoff errors, based on the computation of an upper bound for the condition number.

The last three references first present a formula for interval multiplication, with 3 to 5 real multiplications required for one interval multiplication. Then it is shown how to adapt this formula to the case of interval matrix multiplication.

3.4 Experiments: efficiency and accuracy.

Experiments have been performed using MatLab, matrices are generated using the `randn` function of MatLab, which takes as argument n the dimension of the matrices (all dimensions are equal). This function returns a matrix of size $n \times n$ whose components are uniformly distributed random numbers between 0 and 1.

Experimentally, one observes that the execution time of the more elaborate algorithms is roughly n times the execution time of a floating-point matrix multiplication, if the corresponding algorithm performs n punctual matrix multiplication. This is true when the dimension is large enough (typically above 500): the decomposition of the matrices thus becomes negligible (it costs $\mathcal{O}(n^2)$ operations).

The accuracy of each algorithm is related for one part to its theoretical overestimation ratio, but also for another part on the number of matrix operations it performs: a higher number of matrix operations entails a loss of accuracy. For the algorithm in [11], as stated by its author, the overestimation ratio is also related to the condition number and this can be experimentally observed.

4 Optimization and Parallelization of the Product of Interval Matrices

To improve further the execution time, to take benefit from the existing architectures present even in our laptops, we turn now to the parallelization of interval matrix multiplication.

The algorithms presented in Section 3 have some parts which have a high potential for parallelization:

- conversion to and fro a given representation to the other one: each coefficient can be converted independently of the other ones;
- with the mid-rad representation, the computation of the midpoint of the result can be usually performed using any optimized, parallel, routine; then the radius can be computed, again using any optimized, parallel, routine;
- the same phenomenon appears for the variant based on the inf-sup representation.

This potential should easily be exploited on parallel architectures.

However, the hope that algorithms for interval matrix multiplication, that ultimately resort to floating-point matrix multiplication, would benefit for free of optimizations made for this last computation, is not entirely grounded when it comes to parallelization. Indeed, it does not suffice to use parallel BLAS to gain performances. What follows is based on a preliminary work and we do not have yet any firm conclusion nor any recommendation on this topic.

Let us first make clear that our target architectures in this work are multicores, supporting multithreaded computations.

4.1 Rounding modes and multithreading

A first issue is quite technical: during multithreaded computations, it may be difficult to access the device that controls the current rounding mode, to modify it for the duration of the whole computation of a thread and to restore it, or at least to release it, for other threads. Mutual exclusion is a possibility, but it seems contradictory with the concurrent execution of different computations. Maybe a solution as was adopted in fi_lib [6] (and even as soon as in 1998) where no reference was made to the rounding mode and where one ulp was systematically added or subtracted to the endpoints or radius, is a promising one.

4.2 Memory usage

Even if some blocks of computations are independent and thus parallelizable, they operate on common data and this raises the issue of memory usage (are the right data available in the cache).

Another issue is that the algorithms mentioned above use temporary matrices. Cache misses, reduction of data transfer is already an issue for developers of floating-point matrix multiplication. This problem is amplified for developers of interval matrix multiplication. Possible hints are to reduce the size of blocks to accomodate extra matrix variables, or maybe to re-compute – rather than store – temporary results to reduce memory usage.

5 Conclusion

Several formulas are at hand when it comes to implement interval arithmetic and in particular interval matrix multiplication. Choosing the formulas or algorithm that is most suited to a particular application should be guided by the requirements in terms of

- accuracy: are input intervals thin or large? to what extent is overestimation acceptable?
- efficiency: what is the acceptable slowdown?

It is a nice result that interval computations are no more doomed to prohibitive slowdowns, thanks mainly to the continued advances by S. M. Rump during the last decade. However, the user still has to trade off one criterion for the other. The goal of this paper was in part to show that several possibilities exist, and in the other part to illustrate the differences between these possibilities, both on sequential and on parallel architectures.

Further work include the parallelization of the product of interval matrices on a wider range of architectures, notably including GPU.

References

1. Martine Ceberio and Vladik Kreinovich, *Fast Multiplication of Interval Matrices (Interval Version of Strassen's Algorithm)*, Reliable Computing **10** (2004), no. 3, 241–243.
2. Nicholas J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM Press, 2002.
3. IEEE Task P754, *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*, IEEE, New York, NY, USA, August 1985.
4. IEEE Tasl P754, *IEEE 754-2008, Standard for Floating-Point Arithmetic*, IEEE, New York, NY, USA, August 2008.
5. IEEE 1788 working group for the standardization of interval arithmetic: http://grouper.ieee.org/groups/1788/, mailing list: http://grouper.ieee.org/groups/1788/email/.
6. M. Lerch, G. Tischler, J. Wolff von Gudenberg, W. Hofschuster and W. Krämer, *filib++, a Fast Interval Library Supporting Containment Computations*, ACM TOMS **32** (2006), no. 2, pp. 299-324.
7. Arnold Neumaier, *Interval Methods for Systems of Equations*, Cambridge University Press, 1990.
8. Hong Diep Nguyen, *Efficient implementation of interval matrix multiplication*, Proceedings of PARA 2010: State of the Art in Scientific and Parallel Computing, to appear in LNCS.
9. Hong Diep Nguyen, *Efficient algorithms for verified scientific computing: numerical linear algebra using interval arithmetic*, Ph.D thesis, ENS de Lyon, January 2011.
10. Siegfried M. Rump, *Fast And Parallel Interval Arithmetic*, BIT Numerical Mathematics **39** (1999), no. 3, 539–560.
11. Siegfried M. Rump, *Fast Interval Matrix Multiplication*, to appear in Numerical Algorithms.