An Overview of Numalis Software Suite for Reliable Numerical Computation

Arnault Ioualalen
Numalis, Montpellier, France
Email: ioualalen@numalis.com

Matthieu Martel
LAMPS Laboratory & Numalis
Université de Perpignan, France
Email: matthieu.martel@univ-perp.fr

Nicolas Normand
Numalis, Montpellier, France
Email: nnormand@numalis.com

***Abstract*—Numerical algorithms are used in many areas but they rely on approximate computations due to the finite precision computer arithmetic. As critical systems perform more and more calculations, needs for verification and validation techniques and for assisted development increase, the computer arithmetics being particularly not intuitive. It is then necessary to provide tools to the programmers, to help them to validate and increase the numerical quality of their codes and, broadly, to develop more fastly more reliable numerical codes. In this article, we give a description of the main problems concerning numerical accuracy encountered in industry at the software engineering level and we give an overview of the solutions proposed by the software suite developed by the Numalis Company. This suite contains tools for verification and validation by static and dynamic analysis as well as assisted development tools. The latter tools optimize programs in order to make them compute more accurate results and they also infer the least formats, in terms of bit size, in order to fulfill accuracy requirements.**

I. Introduction

Numerical algorithms are used in many areas ranging from scientific computing to digital processing in embedded systems. All these computations necessarily have a limited accuracy and the needs for verification and validation techniques increase as quickly as critical tasks relying on complex computations are delegated to computers, for example in cars, aircrafts or space vehicles. In addition to verification and validation concerns, assisted methods of conception are strongly desired since it is extremely difficult to understand the reasons why the implementation of a formula is numerically inaccurate and how to improve it. This is because the computer arithmetics, mainly the fixed-point [15] and floating-point arithmetics [2], are particularly not intuitive. It is then necessary to provide tools to the programmers, to help them to validate and increase the numerical quality of their codes and, broadly, to develop more fastly more reliable numerical codes.

Numalis is a company specialized in numerical accuracy. Numalis software suite aims at bringing solutions to the problems mentionned earlier, for V&V and assisted development of numerical algorithms. Numalis activity is mainly focused on critical embedded systems in defense, aeronautic and space, automotive but also in other economic activities such as finance or geophysics.

In this article, we give a return of experience concerning the needs, at software engineering level, of our consumers for the development of numerical code and we introduce Numalis software suite. This suite contains tools for static analysis by abstract interpretation and dynamic analysis based on statistical estimation of the test datasets. It also contains tools for optimizing the accuracy of programs and for mixed-precision format tuning. This suite bring partial answers to the challenges raised by our industrial partners.

This article is organized as follows. Section II gives a brief description of the fixed-point and floating-point arithmetics. Section III introduces the main challenges in the domain at software engineering level and Section IV gives an overview of Numalis Software Suite. Section V concludes.

II. Computer Arithmetics

Numerical computations rely on non-intuitive computer arithmetics, mainly the floating-point and the fixed-point arithmetic, briefly described hereafter.

An important step towards the design of reliable numerical software was the definition, in the 1980's, of the IEEE754 Standard for floating-point arithmetic [2]. A floating-point number x in base β is defined by $x = s \cdot (d_0.d_1 \dots d_{p-1}) \cdot \beta^e = s \cdot m \cdot \beta^{e-p+1}$ where $s \in \{-1, 1\}$ is the sign, $m = d_0 d_1 \dots d_{p-1}$ is the significand, $0 \leq d_i < \beta$, $0 \leq i \leq p-1$, p is the *precision* and e is the exponent. The IEEE754 Standard specifies a few values for β, p, e_{min} and e_{max} and special values such as NaN (Not a Number) or $\pm\infty$ for overflows. Finally, the IEEE754 Standard defines rounding modes towards $-\infty$, towards $+\infty$, towards zero and to the nearest for elementary operations between floating-point numbers.

The floating-point arithmetic differs strongly from the real number arithmetic. Values have a finite number of digits and the algebraic laws such as associativity or distributivity do not hold. Consequently, the evaluations by a computer of mathematically equivalent formulas (for example $x \times (1+x)$ and $x + x^2$) possibly lead to very different results.

There exists no standard for the fixed-point arithmetic comparable to the IEEE754 Standard. A fixed-point format [15] $\langle w, i\rangle$ depends on the total number of bits w used to encode the value and on the location of the fixed-point relative to the most significant bit. In general, the numbers are encoded using two's complement and the sequence of bits $b_{w-1} \dots b_0$ represents the value $-b_{w-1} \cdot 2^{i-1} + \sum_{j=2}^{j=w} b_{w-j} \cdot 2^{i-j}$ and the distance between two consecutive numbers is 2^{i-w}. The format of the result of an elementary operation depends on the formats of its operands.

Implementing efficiently an expression in the fixed-point arithmetic requires to find an evaluation scheme which minimizes the total size w of the formats of the intermediary results. For example, we may give two implementations of the polynomial $x^2 - x + 9$, with x in the format $\langle 5, 3\rangle$. The first scheme corresponds to the direct implementation and requires 68 bits to store the intermediary results while the second scheme implements the equivalent formula $(x - 3) \times (x - 3)$ and necessitates 40 bits only [18].

To help programmers to understand more the accuracy of their numerical algorithms, Numalis provides an online accuracy toolkit in the form of a small analyzer. This analyzer allows the user to write programs and discover their numerical accuracy[1].

III. Challenges in Reliable Computation

The production of reliable numerical code introduces strong needs of assisted development tools as well as of verification and validation tools. These needs as even more important when numerical computations are performed by critical embedded systems as it is the case in many industries such as aeronautic, space, automotive, medical instrumentation, robotics, etc. The returns of experience from Numalis industrial partners make us identify the following needs.

Numerical Accuracy Determination: Determining the accuracy of the results of a program is a central question for verification and validation. Needs in this domain become even more important when the manufactured systems must be certified, for example in aeronautics. During the last fifteen years, static analyses of the numerical accuracy of floating-point computations have been introduced. This work has also been motivated by a few disasters due to numerical bugs [1]. While these methods compute an over-approximation of the worst error arising during the executions of a program, they operate on final codes, during the verification phase and not at implementation time. Static analyses [10], [14], [26] have been proposed and implemented in academic tools such as Fluctuat [14], based on abstract interpretation or FP-Taylor [26], which performs a static analysis using Taylor series expansions. Dynamic techniques have also been proposed but they only estimate the accuracy without formal guaranties [3], [11]

Automatic Accuracy Optimization: Automatically transforming a program at compile-time in function of given ranges for the inputs and in order to make its numerical computations more accurate is an important aspect of assisted development in our domain. Indeed, understanding the reasons why the implementation of a formula is numerically inaccurate and how to improve it is usually difficult because computer arithmetic is particularly not intuitive. So, it is necessary to provide tools to the programmers, to help them to increase the numerical quality of their codes. Academic tools have been developed. Salsa [8] takes inter-procedural imperative programs and optimize then using a source-to-source transformation. Herbie [24] optimizes the arithmetic expressions of Scala codes. While Salsa uses a static analysis to select the best program, Herbie uses dynamic analysis (a set or random runs). These tools are compared in [9].

Floating-Point Format Determination: This problem, also related to assisted development, consists of determining the minimal precision on the inputs and on the intermediate results of a program performing fixed-point or floating-point computations in order to ensure a desired accuracy on the outputs. This allows compilers to select the most appropriate formats and to save memory, reduce CPU usage and use less bandwidth for communications. Research work has been carried out to determine the best floating-point formats. Darulova and Kuncak use a static analysis to compute the propagation of errors [10]. In this approach, all the values have the same format. Martel proposed another static analysis which makes it possible to determine different formats for different variables (mixed precision). Chiang *et al.* [6] use Symbolic Taylor Expansions to allocate a precision to the terms of an arithmetic expression (only). Other approaches rely on dynamic analysis. Precimonious tries to decrease the precision of variables and checks whether accuracy requirements are still fulfilled [22], [25]. Lam *et al* instrument binary codes in order to modify their precision without modifying the source codes [17].

Fixed-Point Format Determination Determining the best fixed-point formats is mandatory in order to implement a numerical algorithm in hardware (e.g. on a FPGA) or software. This may be done in two steps. First, a range analysis determines the integer wordlength of each variable and then the number of bits to allocate to the fractional part is decided [21]. Various strategies have been proposed to solve this problem, based on simulation [23] or analytic models [12], [20].

Other challenges have been identifed but less work have been done currently to address them. The question of optimizing simultaneously the accuracy and execution time is recurrent. It has been shown on academic examples that optimizing the accuracy of numerical iterative algorithms

[1]Numalis online accuracy analysis toolkit: http://www.numalis.com/demonstrateur.php

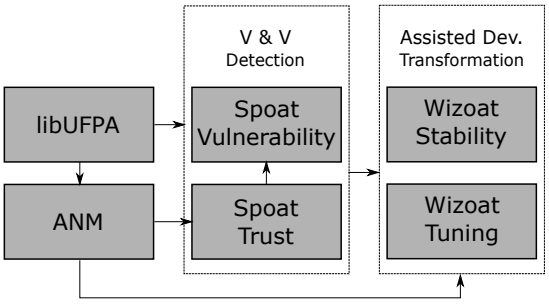

Figure 1. Interactions between the tools of Numalis Software Suite.

accelerate their convergence speed [7]. Time is then saved thanks to a better accuracy. Finally, much work concerning formal proofs of algorithms in the floating-point arithmetic have been done in academic contexts [4], [5].

IV. Numalis Software Suite

Numalis Software Suite is intended to answer to the main challenges enumerated in Section III. For verification and validation, our tools compute over-approximations of the ranges of the variables as well as their accuracy, defined as the distance in the worst case between the result returned by the machine and the result that we would obtain if all the computation were done in the exact arithmetic. Our tools mostly rely on static analysis by abstract interpretation. However, to avoid some false alarms inherent to the over-approximations done in static analysis, dynamic analysis methods are also used. They rely on a statiscical method which allows us to estimate with high probability the number of test-cases needed to estimate safely the ranges of the variables and on a multi-arithmetic library which, thanks to a homogeneous API, makes it possible to execute easily a code in various, IEEE754 compliant or multiple precision arithmetics. The static and dynamic analysis tools may cooperate in our suite.

The tools for assisted development use the range over-approximations computed by the V&V tools. They perform program transformation in order to generate more accurate codes and mixed-precision floating-point format determination. Currently, the support of the fixed-point arithmetic is still under development. The program transformation uses Abstract Program Expression Graphs [8], [16] to generate in polynomial space and time an exponential number of mathematically equivalent expressions among which the most accurate expression for the floating-point arithmetic is searched. The ranges provided by the V&V tools are used to determine the worst errors on the original expressions and on the candidate optimized expressions. The format determination also uses the ranges computed by static analysis to determine the least formats, in mixed precision (i.e. each variable may have its own format) required in order to ensure a certain accuracy on the outputs, determined by the user [19]. This tools also relies on a SMT solver. We give hereafter a brief description of the tools integrated in Numalis Software Suite.

- `libUFPA` is a library which contains many arithmetics used by the other tools, for V&V and for program transformation. This library has a uniform API to use floating-point arithmetic (any IEEE754 format), multiple precision arithmetic (with the MPFR library), interval arithmetic and affine arithmetic [13]. `libUFPA` supports all the usual elementary functions (e.g. for trigonometry.)
- `ANM` is Numalis static analyzer based on abstract interpretation. `ANM` uses some of the arithmetic of `libUFPA`. As `libUFPA`, `ANM` is an internal tool which provides basic information to the backend tools enumerated hereafter. `ANM` uses relational abstract domains such as Affine Forms [13].
- `Spoat-Vulnerability` is Numalis tool to detect accuracy errors in programs. `Spoat-Vulnerability` uses the results of the static analysis provided by `ANM`. However, when this information is not precise enough, `Spoat-Vulnerability` performs a local dynamic analysis in order to obtain more realistic range estimations (yet unsafe). Several runs are performed and the results are merged into intervals.
- `Spoat-Trust` is a tool for statistical analysis to estimate with high probability how many datasets must be taken to find safe ranges on the outputs of a program in function of the ranges of the inputs. Then it generates the random tests based on the statistical information and run them using `Spoat-Vulnerability`.
- `Wizoat-Stability` optimizes programs. First, it optimizes arithmetic expressions. It takes as inputs an expression and ranges for the free variables (given by the `Spoat` tools) and generates a new expression, more accurate as long as the values of the free variables belong to the given ranges. Then the computations done at several lines of code of the original program are merged or combined differently than in the original code in order to augment the optimization opportunities (otherwise, for example, one cannot optimize a three address code.) `Wizoat-Stability` calls `Spoat-Vulnerability` to evaluate the accuracy of the new arithmetic expressions.
- `Wizoat-Tuning` determines the least floating-point formats needed in order to ensure a user-defined accuracy on the outputs of a code [19]. This tool also relies on `Spoat-Vulnerability` and calls a SMT solver (Z3) for the format inference.

The tools described above support several programming languages (C, C++, Ada, etc.). We illustrate in Figure 1 how

these tools interact altogether. Basically, the `Spoat` tools are for error detection (V&V) while the `Wizoat` tools perform program transformation (used in assisted development).

V. Conclusion

In this article we have given an overview of the problems encountered at software engineering level concerning the development of numerical codes, specially for critical systems, as well as a description of academic solutions and of Numalis solutions. Numalis software suite has been used in several industries, mainly in defense and space industries.

Our Software Suite is still under development. Improving the precision of the static and dynamic analysis is an endless problem. We also plan to add the support of other programming languages (Lustre, Fortran, etc.) Program optimizations may still be improved in many ways. The support of fixed-point arithmetic and of parallel programs are important objectives. In particular, concerning parallel programs, transformations ensuring the reproductibility of the results would be of great interest. For all these development, Numalis team pay much attention to the latest research results in the domain and to research collaborations.

References

[1] "Patriot missile defense: Software problem led to system failure at dhahran, saudi arabia," General Accounting office, Tech. Rep. GAO/IMTEC-92-26, 1992.

[2] *IEEE Standard for Binary Floating-point Arithmetic*, ANSI/IEEE, 2008.

[3] E. T. Barr, T. Vo, V. Le, and Z. Su, "Automatic detection of floating-point exceptions," in *Principles of Programming Languages, POPL '13*. ACM, 2013.

[4] S. Boldo, J. Jourdan, X. Leroy, and G. Melquiond, "Verified compilation of floating-point computations," *J. Autom. Reasoning*, vol. 54, no. 2, 2015.

[5] M. Brain, C. Tinelli, P. Rümmer, and T. Wahl, "An automatable formal semantics for IEEE-754 floating-point arithmetic," in *Symposium on Computer Arithmetic*. IEEE, 2015.

[6] W. Chiang, M. Baranowski, I. Briggs, A. Solovyev, G. Gopalakrishnan, and Z. Rakamaric, "Rigorous floating-point mixed-precision tuning," in *POPL*. ACM, 2017.

[7] N. Damouche, M. Martel, and A. Chapoutot, "Impact of accuracy optimization on the convergence of numerical iterative methods," in *LOPSTR*, ser. LNCS, vol. 9527. Springer, 2015.

[8] ——, "Intra-procedural optimization of the numerical accuracy of programs," in *FMICS*, ser. LNCS, vol. 9128. Springer, 2015.

[9] N. Damouche, M. Martel, P. Panchekha, C. Qiu, A. Sanchez-Stern, and Z. Tatlock, "Toward a standard benchmark format and suite for floating-point analysis," in *NSV*, ser. LNCS, vol. 10152, 2016.

[10] E. Darulova and V. Kuncak, "Sound compilation of reals," in *POPL*. ACM, 2014.

[11] C. Denis, P. de Oliveira Castro, and E. Petit, "Verificarlo: Checking floating point accuracy through monte carlo arithmetic," in *ARITH*. IEEE Computer Society, 2016.

[12] C. F. Fang, R. A. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," in *ICCAD'03*. IEEE / ACM, 2003.

[13] K. Ghorbal, E. Goubault, and S. Putot, "The zonotope abstract domain taylor1+," in *CAV*, ser. LNCS, vol. 5643, 2009.

[14] E. Goubault and S. Putot, "Static analysis of finite precision computations," in *VMCAI*, ser. LNCS, vol. 6538, 2011.

[15] M. Graphics, *Algorithmic C Datatypes*, software version 2.6 ed., 2011, http://www.mentor.com/esl/catapult/algorithmic.

[16] A. Ioualalen and M. Martel, "A new abstract domain for the representation of mathematically equivalent expressions," in *SAS*, ser. LNCS, vol. 7460. Springer, 2012.

[17] M. O. Lam, J. K. Hollingsworth, B. R. de Supinski, and M. P. LeGendre, "Automatically adapting programs for mixed-precision floating-point computation," in *Supercomputing, ICS'13*. ACM, 2013.

[18] M. Martel, "Accurate evaluation of arithmetic expressions," *Electr. Notes Theor. Comput. Sci.*, vol. 287, 2012.

[19] ——, "Floating-point format inference in mixed-precision," in *NASA Formal Methods*, vol. 10227, 2017.

[20] M. Martel, A. Najahi, and G. Revy, "Code size and accuracy-aware synthesis of fixed-point programs for matrix multiplication," in *PECCS*. SciTePress, 2014.

[21] D. Ménard, D. Chillet, and O. Sentieys, "Floating-to-fixed-point conversion for digital signal processors," *EURASIP J. Adv. Sig. Proc.*, vol. 2006, 2006.

[22] C. Nguyen, C. Rubio-Gonzalez, B. Mehne, K. Sen, J. Demmel, W. Kahan, C. Iancu, W. Lavrijsen, D. H. Bailey, and D. Hough, "Floating-point precision tuning using blame analysis," in *ICSE*. ACM, 2016.

[23] Z. Nikolic, H. T. Nguyen, and G. Frantz, "Design and implementation of numerical linear algebra algorithms on fixed point dsps," *EURASIP J. Adv. Sig. Proc.*, 2007.

[24] P. Panchekha, A. Sanchez-Stern, J. R. Wilcox, and Z. Tatlock, "Automatically improving accuracy for floating point expressions," in *PLDI*. ACM, 2015.

[25] C. Rubio-Gonzalez, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: tuning assistant for floating-point precision," in *High Performance Computing, Networking, Storage and Analysis*. ACM, 2013.

[26] A. Solovyev, C. Jacobsen, Z. Rakamaric, and G. Gopalakrishnan, "Rigorous estimation of floating-point round-off errors with symbolic taylor expansions," in *FM*, ser. LNCS, vol. 9109. Springer, 2015.