INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Arénaire

Computer Arithmetic

Grenoble - Rhône-Alpes

Theme : Algorithms, Certification, and Cryptography

Activity Report

2009

Table of contents

Arénaire is a joint project of CNRS, École Normale Supérieure de Lyon (U. Lyon), INRIA, and Université Claude Bernard de Lyon (U. Lyon). As part of the Laboratoire de l'Informatique du Parallélisme (LIP, UMR 5668), it is located at Lyon in the buildings of the ÉNS.

1. Team

Research Scientist

Nicolas Brisebarre [CR CNRS]
Claude-Pierre Jeannerod [CR INRIA]
Vincent Lefèvre [CR INRIA]
Micaela Mayero [CR INRIA (on partial secondment), since September 2009]
Jean-Michel Muller [DR CNRS, HdR]
Nathalie Revol [CR INRIA]
Damien Stehlé [CR CNRS, until July 2008, back in July 2010, currently seconded to Macquarie University and the University of Sydney]
Gilles Villard [Team leader, DR CNRS, HdR]

Faculty Member

Florent de Dinechin [Team leader, Associate Professor ÉNS Lyon, *Maître de Conférences*, HdR]
Guillaume Hanrot [Professor ÉNS Lyon, *Professeur*, since October 2009, HdR]
Nicolas Louvet [Associate Professor UCBL, *Maître de Conférences*]

Technical Staff

Christman Fagbohoun [Technical Staff, since April 2009]
Honoré Takeugming [Engineer on the ANR TCHATER project, since November 2008]
Philippe Théveny [Technical Staff, since October 2009]
Serge Torres [Technical Staff, 40% on the project]

PhD Student

Sylvain Chevillard [*Allocataire-moniteur*, ÉNS grant, thesis defended in July 2009, now postdoc within the Cacao project-team, LORIA, Nancy]
Mioara Joldes [*Allocataire-moniteur*, Région Rhône-Alpes grant, 2nd year]
Jingyan Jourdan Lu [CIFRE grant, 1st year]
Érik Martin-Dorel [MESR grant, 1st year]
Ivan Morel [*Allocataire-moniteur*, ÉNS grant, cotutelle with the University of Sydney, 3rd year]
Christophe Mouilleron [*Allocataire-moniteur*, ÉNS grant, 2nd year]
Hong Diep Nguyen [INRIA grant, 3rd year]
Adrien Panhaleux [*Allocataire-moniteur*, ÉNS grant, 2nd year]
Bogdan Pasca [*Allocataire-moniteur*, MESR grant, 2nd year]
David Pfannholzer [MEFI grant, 1st year]
Xavier Pujol [*Allocataire-moniteur*, ÉNS grant, 1st year]
Guillaume Revy [*Allocataire-moniteur*, MESR grant, thesis defended in December 2009]

Post-Doctoral Fellow

Andrew Novocin [INRIA grant, since September 2009]
Álvaro Vázquez Álvarez [INRIA grant, since October 2009]

Administrative Assistant

Marie Buillas [TR INRIA, 50% on the project]

2. Overall Objectives

2.1. Introduction

The Arénaire project aims at elaborating and consolidating knowledge in the field of Computer Arithmetic, which studies how a machine deals with numbers. Reliability, accuracy, and performance are the major goals that drive our research. We study basic arithmetic operators such as adders, dividers, etc. We work on new operators for the evaluation of elementary and special functions ($\log$, $\cos$, erf, etc.), and also consider the composition of previous operators. In addition to these studies on the arithmetic operators themselves, our research focuses on specific application domains (cryptography, signal processing, linear algebra, lattice basis reduction, etc.) for a better understanding of the impact of the arithmetic choices on solving methods in scientific computing.

We contribute to the improvement of the available arithmetic on computers, processors, dedicated or embedded chips, etc., both at the hardware level and at the software level. Improving computing does not necessarily mean getting more accurate results or getting them faster: we also take into account other constraints such as power consumption, code size, or the reliability of numerical software. All branches of the project focus on algorithmic research and on the development and the diffusion of corresponding libraries, either in hardware or in software. Some distinctive features of our libraries are numerical quality, reliability, and performance.

The study of the number systems and, more generally, of data representations is a first topic of uttermost importance in the project. Typical examples are: the redundant number systems used inside multipliers and dividers; alternatives to floating-point representation for special purpose systems; finite field representations with a strong impact on cryptographic hardware circuits; the performance of an interval arithmetic that heavily depends on the underlying real arithmetic.

Another general objective of the project is to improve the validation of computed data, we mean to provide more guarantees on the quality of the results. For a few years we have been handling those validation aspects in the following three complementary ways: through better qualitative properties and specifications (correct rounding, error bound representation, and portability in floating-point arithmetic); by proposing a development methodology focused on the proven quality of the code; by studying and allowing the cooperation of various kinds of arithmetics such as constant precision, intervals, arbitrary precision and exact numbers.

These goals may be organized in four directions: *hardware arithmetic*, *software arithmetic for algebraic and elementary functions*, *validation and automation*, and *arithmetics and algorithms for scientific computing*. These directions are not independent and have strong interactions. For example, elementary functions are also studied for hardware targets, and scientific computing aspects concern most of the components of Arénaire.

- *Hardware Arithmetic.* From the mobile phone to the supercomputer, every computing system relies on a small set of computing primitives implemented in hardware. Our goal is to study the design of such arithmetic primitives, from basic operations such as the addition and the multiplication to more complex ones such as the division, the square root, cryptographic primitives, and even elementary functions. Arithmetic operators are relatively small hardware blocks at the scale of an integrated circuit, and are best described in a structural manner: a large operator is assembled from smaller ones, down to the granularity of the bit. This study requires knowledge of the hardware targets (ASICs, FPGAs), their metrics (area, delay, power), their constraints, and their specific language and tools. The input and output number systems are typically given (integer, fixed-point of floating-point), but internally, non-standard internal number systems may be successfully used.
- *Algebraic and Elementary Functions.* Computer designers still have to implement the basic arithmetic functions for a medium-size precision. Addition and multiplication have been much studied but their performance may remain critical (silicon area or speed). Division and square root are less critical, however there is still room for improvement (e.g. for division, when one of the inputs is constant). Research on new algorithms and architectures for elementary functions is also very active. Arénaire has a strong reputation in these domains and will keep contributing to their expansion.

Thanks to past and recent efforts, the semantics of floating-point arithmetic has much improved. The adoption of the IEEE-754 standard for floating-point arithmetic has represented a key point for improving numerical reliability. Standardization is also related to properties of floating-point arithmetic (invariants that operators or sequences of operators may satisfy). Our goal is to establish and handle new properties in our developments (correct rounding, error bounds, etc.) and then to have those results integrated into the future computer arithmetic standards.

- *Validation and Automation.* Validation corresponds to some guarantee on the quality of the evaluation. Several directions are considered, for instance the full error (approximation plus rounding errors) between the exact mathematical value and the computed floating-point result, or some guarantee on the range of a function. Validation also comprises a proof of this guarantee that can be checked by a proof checker. Automation is crucial since most development steps require specific expertise in floating-point computing that can neither be required from code developers nor be mobilised manually for every problem.
- *Arithmetics and Algorithms.* When conventional floating-point arithmetic does not suffice, we use other kinds of arithmetics. Especially in the matter of error bounds, we work on interval arithmetic libraries, including arbitrary precision intervals. Here a main domain of application is global optimization. Original algorithms dedicated to this type of arithmetic must be designed in order to get accurate solutions, or sometimes simply to avoid divergence (e.g., infinite intervals). We also investigate exact arithmetics for computing in algebraic domains such as finite fields, unlimited precision integers, and polynomials. A main objective is a better understanding of the influence of the output specification (approximate within a fixed interval, correctly rounded, exact, etc.) on the complexity estimates for the problems considered. Those problems mainly come from two application domains: exact linear algebra and lattice basis reduction.

Our work in Arénaire since its creation in 1998, and especially since 2002, provides us a strong expertise in computer arithmetic. This knowledge, together with the technology progress both in software and hardware, draws the evolution of our objectives towards the *synthesis of validated algorithms*.

2.2. Highlights of the Year

Our *Handbook of Floating-Point Arithmetic* (Birkhäuser Boston, November 2009, 600 pages, ISBN 78-0-8176-4704-9) gathers years of common work on IEEE floating-point arithmetic.

3. Scientific Foundations

3.1. Introduction

As stated above, four major directions in Arénaire are *hardware arithmetic*, *algebraic and elementary functions*, *validation and automation*, and *arithmetics and algorithms*. For each of those interrelated topics, we describe below the tools and methodologies on which it relies.

3.2. Hardware Arithmetic

A given computing application may be implemented using different technologies, with a large range of trade-offs between the various aspects of performance, unit cost, and non-recurring costs (including development effort).

- A software implementation, targeting off-the-shelf microprocessors, is easy to develop and reproduce, but will not always provide the best performance.
- For cost or performance reasons, some applications will be implemented as application specific integrated circuits (ASICs). An ASIC provides the best possible performance and may have a very low unit cost, at the expense of a very high development cost.
- An intermediate approach is the use of reconfigurable circuits, or field-programmable gate arrays (FPGAs).

In each case, the computation is broken down into elementary operations, executed by elementary hardware elements, or *arithmetic operators*. In the software approach, the operators used are those provided by the microprocessor. In the ASIC or FPGA approaches, these operators have to be built by the designer, or taken from libraries. Our goals include studying operators for inclusion in microprocessors and developing hardware libraries for ASICs or FPGAs.

Operators under study. Research is active on algorithms for the following operations:

- Basic operations (addition, subtraction, multiplication), and their variations (multiplication and accumulation, multiplication or division by constants, etc.);
- Algebraic functions (division, inverse, and square root, and in general, powering to an integer, and polynomials);
- Elementary functions (sine, cosine, exponential, etc.);
- Combinations of the previous operations (norm, for instance).

A hardware implementation may lead to better performance than a software implementation for two main reasons: parallelism and specialization. The second factor, from the arithmetic point of view, means that specific data types and specific operators, which would require costly emulation on a processor, may be used. For example, some cryptography applications are based on modular arithmetic and bit permutations, for which efficient specific operators can be designed. Other examples include standard representations with non-standard sizes, and specific operations such as multiplication by constants.

Hardware-oriented algorithms. Many algorithms are available for the implementation of elementary operators (see for instance [54]). For example, there are two classes of division algorithms: digit-recurrence and function iteration. The choice of an algorithm for the implementation of an operation depends on, and sometimes imposes, the choice of a number representation. Besides, there are usually technological constraints such as the area and power budget, and the available low-level libraries.

The choice of the number systems used for the intermediate results is crucial. For example, a redundant system, in which a number may have several encodings, will allow for more design freedom and more parallelism, hence faster designs. However, the hardware cost can be higher. As another example, the power consumption of a circuit depends, among other parameters, on its activity, which in turn depends on the distribution of the values of the inputs, hence again on the number system.

Alternatives exist at many levels in this algorithm exploration. For instance, an intermediate result may be either computed, or recovered from a precomputed table.

Parameter exploration. Once an algorithm is chosen, optimizing its implementation for area, delay, accuracy, or energy consumption is the next challenge. The best solution depends on the requirements of the application and on the target technology. Parameters which may vary include the radix of the number representations, the granularity of the iterations (between many simple iterations, or fewer coarser ones), the internal accuracies used, the size of the tables (see [55] for an illustration), etc.

The parameter space quickly becomes huge, and the expertise of the designer has to be automated. Indeed, we do not design operators, but *operator generators*, programs that take a specification and some constraints as input, and output a synthesizable description of an operator.

3.3. Algebraic and Elementary Functions

Elementary Functions and Correct Rounding. Many libraries for elementary functions are currently available. We refer to [54] for a general insight into the domain. The functions in question are typically those defined by the C99 and LIA-2 standards, and are offered by vendors of processors, compilers or operating systems.

Though the 1985 version of the IEEE-754 standard does not deal with these functions, there is some attempt to reproduce some of their mathematical properties, in particular symmetries. For instance, monotonicity can be obtained for some functions in some intervals as a direct consequence of accurate internal computations or numerical properties of the chosen algorithm to evaluate the function; otherwise it may be *very* difficult to guarantee, and the general solution is to provide it through correct rounding. Preserving the range (e.g., $\mathrm{atan}(x) \in [-\pi/2, \pi/2]$) may also be a goal though it may conflict with correct rounding (when supported).

Concerning the correct rounding of the result, it was not required by the IEEE-754-1985 standard: during the elaboration of this standard, it was considered that correctly rounded elementary functions were impossible to obtain at a reasonable cost, because of the so-called *Table Maker's Dilemma*: an elementary function is evaluated to some internal accuracy (usually higher than the target precision), and then rounded to the target precision. What is the minimum accuracy necessary to ensure that rounding this evaluation is equivalent to rounding the exact result, for all possible inputs? This question could not be answered in a simple manner, meaning that correctly rounding elementary functions may require arbitrary precision, which is very slow and resource-consuming.

Indeed, correctly rounded libraries already exist, such as GNU MPFR (http://www.mpfr.org/), the Accurate Portable Library released by IBM in 2002, or the `libmcr` library, released by Sun Microsystems in late 2004. However they have worst-case execution time and memory consumption up to 10,000 worse than usual libraries, which is the main obstacle to their generalized use.

We have focused in the previous years on computing bounds on the intermediate precision required for correctly rounding some elementary functions in IEEE-754 double precision. This allows us to design algorithms using a tight precision. That makes it possible to offer the correct rounding with an acceptable overhead: we have experimental code where the cost of correct rounding is negligible in average, and less than a factor 10 in the worst case. These performances led the IEEE-754 revision committee to recommend (yet not request) correct rounding for some mathematical functions. It also enables to prove the correct-rounding property, and to show bounds on the worst-case performance of our functions. Such worst-case bounds may be needed in safety critical applications as well as a strict proof of the correct rounding property. Concurrent libraries by IBM and Sun can neither offer a complete proof for correct rounding nor bound the timing because of the lack of worst-case accuracy information. Our work actually shows a posteriori that their overestimates for the needed accuracy before rounding are however sufficient. IBM and Sun for themselves could not provide this information. See also §3.4 concerning the proofs for our library.

Approximation and Evaluation. The design of a library with correct rounding also requires the study of algorithms in large (but not arbitrary) precision, as well as the study of more general methods for the three stages of the evaluation of elementary functions: argument reduction, approximation, and reconstruction of the result.

When evaluating an elementary function for instance, the first step consists in reducing this evaluation to the one of a possibly different function on a small real interval. Then, this last function is replaced by an approximant, which can be a polynomial or a rational fraction. Being able to perform those processes in a very cheap way while keeping the best possible accuracy is a key issue [2]. The kind of approximants we can work with is very specific: the coefficients must fulfill some constraints imposed by the targeted application, such as some limits on their size in bits. The usual methods (such as Remez algorithm) do not apply in that situation and we have to design new processes to obtain good approximants with the required form. Regarding the approximation step, there are currently two main challenges for us. The first one is the computation of excellent approximations that will be stored in hardware or in software and that should be called thousands or millions of times. The second one is the target of automation of computation of good approximants when the function is only known at compile time. A third question concerns the evaluation of such good approximants. To find a best compromise between speed and accuracy, we combine various approaches ranging from numerical analysis (tools like backward and forward error analysis, conditioning, stabilization of algorithms) to computer arithmetic (properties like error-free subtraction, exactly-computable error bounds, etc.). The structure of the approximants must further be taken into account, as well as the degree of parallelism offered by the processor targeted for the implementation.

Adequation Algorithm/Architecture. Some special-purpose processors, like DSP cores, may not have floating-point units, mainly for cost reasons. For such integer or fixed-point processors, it is thus desirable to have software support for floating-point functions, starting with the basic operations. To facilitate the development or porting of numerical applications on such processors, the emulation in software of floating-point arithmetic should be compliant with the IEEE-754 standard; it should also be very fast. To achieve this twofold goal, a solution is to exploit as much as possible the characteristics of the target processor (instruction set, parallelism, etc.) when designing algorithms for floating-point operations.

So far, we have successfully applied this "algorithm/architecture adequation" approach to some VLIW processor cores from STMicroelectronics, in particular the ST231; the ST231 cores have integer units only, but for their applications (namely, multimedia applications), being able to perform basic floating-point arithmetic very efficiently was necessary. When various architectures are targeted, this approach should further be (at least partly) automated. The problem now is not only to write some fast and accurate code for one given architecture, but to have this optimized code generated automatically according to various constraints (hardware resources, speed and accuracy requirements).

3.4. Validation and Automation

Validating a code, or generating a validated code, means being able to prove that the specifications are met. To increase the level of reliability, the proof should be checkable by a formal proof checker.

Specifications of qualitative aspects of floating-point codes. A first issue is to get a better formalism and specifications for floating-point computations, especially concerning the following qualitative aspects:

- *specification:* typically, this will mean a proven error bound between the value computed by the program and a mathematical value specified by the user in some high-level format;
- *tight error bound computation*;
- *floating-point issues:* regarding the use of floating-point arithmetic, a frequent concern is the portability of code, and thus the reproducibility of computations; problems can be due to successive roundings (with different intermediate precisions) or the occurrence of underflows or overflows;
- *precision:* the choice of the method (compensated algorithm versus double-double versus quadruple precision for instance) that will yield the required accuracy at given or limited cost must be studied;
- *input domains and output ranges:* the determination of input domain or output range also constitutes a specification/guarantee of a computation;
- *other arithmetics, dedicated techniques and algorithms for increased precision:* for studying the quality of the results, most of conception phases will require *multiple-precision* or *exact* solutions to various algebraic problems.

Certification of numerical codes using formal proof. Certifying a numerical code is error-prone. The use of a proof assistant will ensure the code correctly follows its specification. This certification work, however, is usually a long and tedious work, even for experts. Moreover, it is not adapted to an incremental development, as a small change to the algorithm may invalidate the whole formal proof. A promising approach is the use of automatic tools to generate the formal proofs of numerical codes with little help from the user.

Instead of writing code in some programming language and trying to prove it, we can design our own language, well-suited to proofs (e.g., close to a mathematical point of view, and allowing metadata related to the underlying arithmetics such as error bounds, ranges, and so on), and write tools to generate code. Targets can be a programming language without extensions, a programming language with some given library (e.g., MPFR if one needs a well-specified multiple-precision arithmetic), or a language internal to some compiler: the proof may be useful to give the compiler some knowledge, thus helping it to do particular optimizations. Of course, the same proof can hold for several targets.

We worked in particular also on the way of giving a formal proof for our correctly rounded elementary function library. We have always been concerned by a precise proof of our implementations that covers also details of the numerical techniques used. Such proof concern is mostly absent in IBM's and Sun's libraries. In fact, many misroundings were found in their implementations. They seem to be mainly due to coding mistakes that could have been avoided with a formal proof in mind. In CRlibm we have replaced more and more hand-written paper proofs by Gappa (http://lipforge.ens-lyon.fr/www/gappa/) verified proof scripts that are partially generated automatically by other scripts. Human error is better prevented.

Integrated and interoperable automatic tools. Various automatic components have been independently introduced above, see §3.2 and §3.3. One of our main objectives is to provide an entire automatic approach taking in input an expression to evaluate (with possible annotations), and returning an executable validated code. The complete automation with optimal or at least good resulting performance seems to be far beyond the current knowledge. However, we see our objective as a major step for prototyping future compilers. We thus aim at developing a piece of software that automates the steps described in the previous pages. The result should be an easy-to-use integrated environment.

3.5. Arithmetics and Algorithms

When computing a solution to a numerical problem, an obvious question is that of the *quality* of the produced numbers. One may also require a certain level of quality, such as: approximate with a given error bound, correctly rounded, or –if possible– exact. The question thus becomes twofold: how to produce such a well-specified output and at what cost? To answer it, we focus on *polynomial and integer matrix operations*, *Euclidean lattices* and *global optimization*, and study the following directions:

- We investigate new ways of producing well-specified results by resorting to various arithmetics (intervals, Taylor models, multi-precision floating-point, exact). A first approach is to *combine* some of them: for example, guaranteed enclosures can be obtained by mixing Taylor model arithmetic with floating-point arithmetic [9]. Another approach is to *adapt* the precision or even *change* the arithmetic during the course of a computation. Typical examples are iterative refinement techniques or exact results obtained via floating-point basic operations. This often requires arithmetics with very-well *specified properties* (like the IEEE-754 standard for floating-point arithmetic).
- We also study the impact of certification on algorithmic complexity. A first approach there is to augment existing algorithms with validated error bounds (and not only error estimates). This leads us to study the (im)possibility of *computing such bounds* on the fly at a negligible cost. A second approach is to study the *algorithmic changes* needed to achieve a higher level of quality without, if possible, sacrificing for speed. In exact linear algebra, for example, the fast algorithms recently obtained in the bit complexity model are far from those obtained decades ago in the algebraic complexity model.

Numerical Algorithms using Arbitrary Precision Interval Arithmetic. When validated results are needed, interval arithmetic can be used. New problems can be solved with this arithmetic, which provides sets instead of numbers. In particular, we target the global optimization of continuous functions. A solution to obviate the frequent overestimation of results is to increase the precision of computations.

Our work is twofold. On the one hand, efficient software for arbitrary precision interval arithmetic is developed, along with a library of algorithms based on this arithmetic. On the other hand, new algorithms that really benefit from this arithmetic are designed, tested, and compared.

To reduce the overestimation of results, variants of interval arithmetic have been developed, such as Taylor models arithmetic or affine arithmetic. These arithmetics can also benefit from arbitrary precision computations.

Algorithms for Exact Linear Algebra and Lattice Basis Reduction. The techniques for exactly solving linear algebra problems have been evolving rapidly in the last few years, substantially reducing the complexity of several algorithms (see for instance [6] for an essentially optimal result, or [53]). Our main focus is on matrices whose entries are integers or univariate polynomials over a field. For such matrices, our main interest is how to relate the size of the data (integer bit lengths or polynomial degrees) to the cost of solving the problem exactly. A first goal is to design asymptotically faster algorithms, to reduce problems to matrix multiplication in a systematic way, and to relate bit complexity to algebraic complexity. Another direction is to make these algorithms fast in practice as well, especially since applications yield very large matrices that are either sparse or structured. Within the LinBox international project, we work on a software library that corresponds to our algorithmic research on matrices. LinBox is a generic library that allows to plug external components in a plug-and-play fashion. The library is devoted to sparse or structured exact linear algebra and its applications.

We recently started a direction around lattice basis reduction. Euclidean lattices provide powerful tools in various algorithmic domains. In particular, we investigate applications in computer arithmetic, cryptology, algorithmic number theory and communications theory. We work on improving the complexity estimates of lattice basis reduction algorithms and providing better implementations of them, and on obtaining more reduced bases. The above recent progress in linear algebra may provide new insights.

Certified Computing. Most of the algorithmic complexity questions that we investigate concern algebraic or bit-complexity models for exact computations. Much less seems to be known in approximate computing, especially for the complexity of computing (certified) error bounds, and for establishing bridges between exact, interval, and constant precision complexity estimates. We are developing this direction both for a theoretical impact, and for the design and implementation of algorithm synthesis tools for arithmetic operators, and mathematical expression evaluation.

4. Application Domains

4.1. Application Domains

Our expertise covers application domains for which the quality, such as the efficiency or safety, of the arithmetic operators is an issue. On the one hand, it can be applied to hardware oriented developments, for example to the design of arithmetic primitives which are specifically optimized for the target application and support. On the other hand, it can also be applied to software programs, when numerical reliability issues arise: these issues can consist in improving the numerical stability of an algorithm, computing guaranteed results (either exact results or certified enclosures) or certifying numerical programs.

- The application domains of hardware arithmetic operators are **digital signal processing**, **image processing**, **embedded applications**, **reconfigurable computing** and **cryptography**.
- Developments of **correctly rounded elementary functions** is critical to the **reproducibility** of floating-point computations. Exponentials and logarithms, for instance, are routinely used in accounting systems for interest calculation, where roundoff errors have a financial meaning. Our current focus is on bounding the worst-case time for such computations, which is required to allow their use in **safety critical** applications, and in proving the correct rounding property for a complete implementation.

- Certifying a numerical application usually requires bounds on rounding errors and ranges of variables. Some of the tools we develop compute or verify such bounds. For increased confidence in the numerical applications, they may also generate formal proofs of the arithmetic properties. These proofs can then be machine-checked by proof assistants like **Coq**.
- Arbitrary precision interval arithmetic can be used in two ways to **validate a numerical result**. To **quickly check the accuracy** of a result, one can replace the floating-point arithmetic of the numerical software that computed this result by high-precision interval arithmetic and measure the width of the interval result: a tight result corresponds to good accuracy. When **getting a guaranteed enclosure** of the solution is an issue, then more sophisticated procedures, such as those we develop, must be employed: this is the case of global optimization problems.
- The design of faster algorithms for matrix polynomials provides faster solutions to various problems in **control theory**, especially those involving multivariable linear systems.
- Lattice reduction algorithms have direct applications in public-key cryptography. They also naturally arise in computer algebra. A new and promising field of applications is communications theory.

5. Software

5.1. Introduction

Arénaire proposes various software and hardware realizations that are accessible from the web page http://www.ens-lyon.fr/LIP/Arenaire/Ware/. We describe below only those which progressed in 2009.

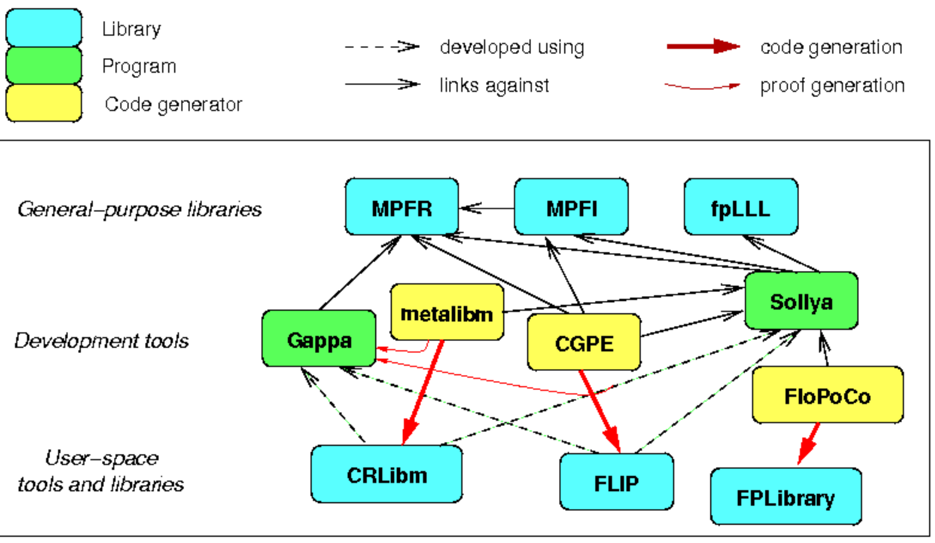

Figure 1. Relationships between some Arénaire developments.

5.2. FLIP: a floating-point library for integer processors

Participants: Christman Fagbohoun, Claude-Pierre Jeannerod, Jingyan Jourdan Lu, Guillaume Revy.

FLIP is a C library for the efficient software support of *binary32* IEEE 754-2008 floating-point arithmetic on processors without floating-point hardware units, such as VLIW or DSP processors for embedded applications. The current target architecture is the VLIW ST200 family from STMicroelectronics (especially the ST231 cores).

FLIP 1.0 has been released in 2009. Most notably, it implements highly-parallel algorithms for correctly-rounded division and square root. Also, more functions have been added to the FLIP project this year, such as square, reciprocal, reciprocal square root, cube root, reciprocal cube root, reciprocal fourth root, fused multiply-add, fused square-add, as well as lower level functions for comparisons, multiplication by 2, min, max, etc. A systematic validation framework has further been set up.

Status: Stable, version 1.0 / **Target:** VLIW processors (ST200 family from STMicroelectronics) / **License:** LGPL / **OS:** Linux / **Programming Language:** C / **URL:** http://flip.gforge.inria.fr/

5.3. Sollya: a toolbox of numerical algorithms for the development of safe numerical codes

Participants: Sylvain Chevillard, Mioara Joldes.

Sollya aims at providing a safe, user-friendly, all-in-one environment for manipulating numerical functions. Its distinguishing feature is that the focus is on safety: numerical results are certified, or a warning is produced. Functionalities include plotting, infinite norm, polynomial approximation (including an original minimax approximation among polynomials with floating-point coefficients), zero finding, etc., and an interpreter for the Sollya scripting language.

During 2009, the development of Sollya has been reduced to small improvements and bug corrections. Sollya is used by several teams in France (we are aware of its use by people in the Arénaire team, at INRIA Saclay, INRIA Rocquencourt and Perpignan University). It is used not only by members of state funded research teams but also by engineers at private companies, in particular in the US and Russia.

Status: Stable, version 1.1 / **Target:** ia32, ia64, PPC, ia32-64, other / **License:** CeCILL-C / **OS:** Unix / **Programming Language:** C / **URL:** http://sollya.gforge.inria.fr/

5.4. Metalibm, a code generator for elementary functions

Participants: Sylvain Chevillard, Florent de Dinechin.

The Metalibm project provides a tool for the automatic implementation of mathematical (libm) functions. A function f is automatically transformed into Gappa-certified C code implementing an approximation polynomial in a given domain with given accuracy. Metalibm is based on the Sollya tool and has been used to produce large parts of CRlibm.

Status: alpha / **Target:** any / **License:** LGPL / **OS:** Unix / **Programming Language:** C / **URL:** http://lipforge.ens-lyon.fr/www/metalibm/

5.5. CGPE: Code Generation for Polynomial Evaluation

Participants: Guillaume Revy, Christophe Mouilleron, Claude-Pierre Jeannerod.

The CGPE project, which has been created this year, aims at generating C codes for fast and certified polynomial evaluation, given various accuracy and architectural constraints. Version 0.1.0 has been released by Guillaume Revy. This first version implements generation and certification algorithms for bivariate polynomials and fixed-point arithmetic, and is parameterized by the operator latencies and the kind of parallelism available on the targeted processor. The numerical accuracy of the generated C codes is certified using multiple-precision interval arithmetic as available in tools like Gappa and MPFI. CGPE 0.1.0 has been used to generate significant parts of the codes of the FLIP library.

Status: beta / **Target:** any / **License:** CeCiLL / **OS:** Unix / **Programming Language:** C++ / **URL:** http://cgpe.gforge.inria.fr/

5.6. CRlibm: a Library of Elementary Functions with Correct Rounding

Participants: Florent de Dinechin, Jean-Michel Muller, Guillaume Revy.

The CRlibm project aims at developing a mathematical library (`libm`) which provides implementations of the double precision C99 standard elementary functions,

- correctly rounded in the four IEEE-754 rounding modes,
- with a comprehensive proof of both the algorithms used and their implementation,
- sufficiently efficient in average time, worst-case time, and memory consumption to replace existing `libm`s transparently.

In 2008, the main objective of the CRlibm project was reached when the revised floating-point standard IEEE-754-2008 was published with a recommendation for correctly rounded functions.

Version 1.0beta2 was released with a much improved power function x^y. However the development focus has now turned to automated libm development with the Metalibm project.

Status: Beta release / **Target:** ia32, ia64, Sparc, PPC / **License:** LGPL / **OS:** Unix / **Programming Language:** C / **URL:** http://www.ens-lyon.fr/LIP/Arenaire

5.7. FloPoCo: a Floating-Point Core generator for FPGAs

Participants: Florent de Dinechin, Bogdan Pasca.

The purpose of the FloPoCo project is to explore the many ways in which the flexibility of the FPGA target can be exploited in the arithmetic realm. FloPoCo is a generator of operators written in C++ and outputting synthesizable VHDL automatically pipelined to an arbitrary frequency.

In 2009, FloPoCo has undergone a complete rewrite that makes automatic pipeline much simpler. Division and square root operators were added. Sylvain Collange, from U. Perpignan, contributed basic operators for Logarithm Number System arithmetic, so FloPoCo is now a strict superset of FPLibrary, which is therefore no longer supported. Non-standard operators developed in 2009 include squarers, sum of squares, Karatsuba multipliers, integer adder trees, table-based constant multipliers and floating-point logarithm. Versions 0.9.3, 0.11, 1.15.0, 1.15.1 and 1.15.2 were released in 2009.

Among the known users of FloPoCo are U.T. Cluj-Napoca, Imperial College, U. Madrid, U. P. Milano, T.U. Muenchen, U. Paderborn, U. Pennsylvania, U. Pernambuco, U. Perpignan, U. Tokyo, and several companies.

Status: stable / **Target:** any / **License:** GPL/LGPL / **OS:** Unix, Linux, Windows / **Programming Language:** C++ / **URL:** http://www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/

5.8. GNU MPFR: a multiple-precision floating-point library with correct rounding

Participants: Vincent Lefèvre, Philippe Théveny.

GNU MPFR is an efficient multiple-precision floating-point library with well-defined semantics (copying the good ideas from the IEEE-754 standard), in particular correct rounding. GNU MPFR provides about 80 mathematical functions, in addition to utility functions (assignments, conversions...). Special data (*Not a Number*, infinities, signed zeros) are handled like in the IEEE-754 standard.

Since the end of 2006, it has become a joint project between the Arénaire and CACAO project-teams. MPFR became a GNU package on 26 January 2009, with the release of GNU MPFR 2.4.0. GNU MPFR 2.4.1 and 2.4.2 were released on 25 February 2009 and 30 November 2009 respectively. On 3 March 2009, the trunk switched from LGPL version 2.1 or later to LGPL version 3 or later.

The main changes done in 2009 on the Arénaire side are bug fixes, better portability, more tests and the distribution of examples with MPFR.

Moreover an ODL (*Opération de Développement Logiciel*) called MPtools had been supported by the INRIA between September 2007 and August 2009: An engineer, Philippe Théveny, was hired to work on MPFR in particular. He was in the CACAO project-team, but in addition to the collaboration by e-mail, Philippe came in Lyon for a few days in January 2008 and in May 2009, and an MPtools meeting took place in Paris in September 2008. Philippe has been in Arénaire since October 2009.

Many software systems use MPFR. The most common one is GCC, which, as of its version 4.3.0 released in March 2008, now even *requires* MPFR (the use of MPFR was previously optional).

Status: stable / **Target:** any / **License:** LGPL / **OS:** Unix, Windows (Cygwin or MinGW) / **Programming Language:** C / **URL:** http://www.mpfr.org/

5.9. MPFI: Multiple Precision Floating-Point Interval Arithmetic

Participants: Nathalie Revol, Sylvain Chevillard, Hong Diep Nguyen, Philippe Théveny.

MPFI is a library in C for interval arithmetic using arbitrary precision (arithmetic and algebraic operations, elementary functions, operations on sets). It is based on MPFR (see §5.8). MPFI is maintained on par with MPFR: it offers the interval counterpart of all mathematical functions provided by MPFR. This year, exhaustive tests have been added. The compliance has been enforced with the proposal in [52] for the implementation of interval arithmetic using floating-point arithmetic. A new release is planned for the first semester of 2010.

Status: Beta release / **Target:** any / **License:** LGPL / **OS:** Unix, Linux, Windows (Cygwin) / **Programming Language:** C / **URL:** http://mpfi.gforge.inria.fr/

5.10. fpLLL: lattice reduction using floating-point arithmetic

Participants: Ivan Morel, Xavier Pujol, Gilles Villard.

FPLLL is a library which allows to perform several fundamental tasks on Euclidean lattices, most notably compute a LLL-reduced basis and compute a shortest non-zero vector in a lattice. The latter functionality has been added in July 2008 by X. Pujol at the end of his Master's degree internship with D. Stehlé. This led to the release of version 3.0 of fpLLL. The fpLLL library relies on floating-point arithmetic for all the computations involving the Gram-Schmidt orthogonalisation of the lattice bases under scope.

FPLLL is becoming the reference for lattice reduction. It is, or adaptations of it are, included in SAGE, PARI GP and MAGMA.

Status: stable / **License:** LGPL / **Programming Language:** C++ / **URL:** http://perso.ens-lyon.fr/damien.stehle

5.11. Exhaustive Tests for the Correct Rounding of Mathematical Functions

Participant: Vincent Lefèvre.

The programs to search for the worst cases for the correct rounding of mathematical functions ($\exp$, $\log$, $\sin$, $\cos$, etc.) in a fixed precision (mainly double precision) using Lefèvre's algorithm have been further improved, in particular: The computations can optionally stop at the right time in case of high load average, in order to get results faster. Heuristics based on consistency of the results have been added to detect bugs in the compiler/libraries/etc.

5.12. SIPE: Small Integer Plus Exponent

Participant: Vincent Lefèvre.

SIPE (Small Integer Plus Exponent) is a C header file providing a fast floating-point arithmetic with correct rounding to the nearest in very small precision. Implemented operations are the addition, subtraction, multiplication, FMA, and minimum/maximum/comparison functions (of the signed numbers or in magnitude). SIPE has been written for exhaustive tests of simple algorithms in small precision in order to prove results or find conjectures (which could then be proved).

Status: alpha / **License:** LGPL / **Programming Language:** C / **URL:** http://www.vinc17.net/software/sipe.h

6. New Results

6.1. Hardware Arithmetic Operators

Participants: Nicolas Brisebarre, Mioara Joldes, Florent de Dinechin, Jean-Michel Muller, Bogdan Pasca, Guillaume Revy.

6.1.1. Complex Division, Table-Based Complex Reciprocal Approximation

A complex division algorithm introduced in 2003 by Ercegovac and Muller requires a prescaling step by a constant factor. Techniques for obtaining this prescaling factor had been mentioned by the authors, which serves to justify the feasibility of the algorithm, but were inadequate for obtaining efficient implementations. Pouya Dormiani, Milos Ercegovac (Univ. of California at Los Angeles) and Jean-Michel Muller formulated Table based solutions [24] for obtaining the prescaling factor, a low precision reciprocal approximation for a complex value, using techniques adopted from univariate function approximations. The main contribution of this work is the extension of generalized multipartite table methods to a function of two variables. The multipartite tables derived were up to 67% more memory efficient than their single table counterparts. They designed a radix-4 complex division unit that uses their technique [23].

6.1.2. Hardware complex polynomial evaluation

Milos Ercegovac (Univ. of California at Los Angeles) and Jean-Michel Muller proposed [14] an efficient hardware-oriented method for evaluating complex polynomials. The method is based on solving iteratively a system of linear equations. The solutions are obtained digit-by-digit on simple and highly regular hardware. The operations performed are defined over the reals. They described a complex-to-real transform, a complex polynomial evaluation algorithm, the convergence conditions, and a corresponding design and implementation. The main features of the method are: the latency of about m cycles for an m-bit precision; the cycle time independent of the precision; a design consisting of identical modules; and digit-serial connections between the modules. The number of modules, each roughly corresponding to serial-parallel multiplier without a carry-propagate adder, is $2(n+1)$ for evaluating an n-th degree complex polynomial. The design allows straightforward tradeoffs between latency and cost: a factor k decrease in cost leads to a factor k increase in latency. The proposed method is attractive for programmable platforms because of its regular and repetitive structure of simple hardware operators.

6.1.3. Large multipliers and squarers with fewer DSP blocks

Large integer multipliers and squarers are pervasively used to build floating-point operators. Bogdan Pasca and Florent de Dinechin proposed three methods to build them using fewer of the small multipliers available in the DSP blocks of current FPGAs [33]. A careful implementation of the classical Karatsuba-Ofman approach has a low overhead and may reduce embedded multiplier count from 4 to 3, from 9 to 6, or from 16 to 10. Building dedicated squarers entails the same savings without any overhead. For the recent Xilinx multipliers whose embedded multipliers are rectangular, these two approaches are inefficient, but a third approach, based on non-standard tiling, is proposed. The multipliers built this way are smaller and faster than those offered by vendor tools. In all the cases, the proposed architectures also try to make best use of the accumulation hardware present in the DSP blocks.

6.1.4. Multiplier-based square roots on FPGAs

Florent de Dinechin, Bogdan Pasca, Mioara Joldes, and Guillaume Revy also studied how these embedded multipliers can be used to implement the square root operation [34]. Compared to state-of-the-art digit recurrence approaches, an original polynomial approach, shown to be more efficient than classical quadratic iterations in this context, leads to a very short latency and low-power architecture for single precision. For double-precision, it appears that the amount of glue logic in these multiplier-based approaches is comparable to the cost of a complete digit-recurrence approach, so only the advantage of shorter latency remains.

6.1.5. Design of an arithmetic-oriented operator generator framework

Florent de Dinechin, Bogdan Pasca, and Cristian Klein, then an internship student, showed how the arithmetic context can be exploited to build generators of highly efficient arithmetic operators [32]. The salient features of the FloPoCo open-source architecture generator framework are: an easy learning curve from VHDL, the ability to embed arbitrary synthesizable VHDL code, portability to mainstream FPGA targets from Xilinx and Altera, automatic management of complex pipelines with support for frequency-directed pipeline, and support for automatic test-bench generation.

6.2. Efficient Floating-Point Arithmetic and Applications

Participants: Nicolas Brisebarre, Claude-Pierre Jeannerod, Vincent Lefèvre, Nicolas Louvet, Jean-Michel Muller, Adrien Panhaleux, Guillaume Revy, Gilles Villard.

6.2.1. Computation of Integer Powers in Floating-point Arithmetic

Peter Kornerup (Odense University, Denmark), Christoph Lauter (Intel, USA), Vincent Lefèvre, Nicolas Louvet and Jean-Michel Muller have introduced several algorithms for accurately evaluating powers to a positive integer in floating-point arithmetic, assuming a fused multiply-add (fma) instruction is available. For bounded, yet very large values of the exponent, they aim at obtaining correctly-rounded results in round-to-nearest mode, that is, their algorithms return the floating-point number that is nearest the exact value [17].

6.2.2. Correctly Rounded Sums

Peter Kornerup, Vincent Lefèvre, Nicolas Louvet, and Jean-Michel Muller have presented a study of some basic blocks needed in the design of floating-point summation algorithms. In particular, they have shown that among the set of the algorithms with no comparisons performing only floating-point additions/subtractions, the 2Sum algorithm introduced by Knuth is minimal, both in terms of number of operations and depth of the dependency graph. Under reasonable conditions, they have also proven that no algorithms performing only round-to-nearest additions/subtractions exist to compute the round-to-nearest sum of at least three floating-point numbers. Starting from an algorithm due to Boldo and Melquiond, they have presented new results about the computation of the correctly-rounded sum of three floating-point numbers [29].

6.2.3. Midpoints and Exact Points of Algebraic Functions

When implementing a function f in floating-point arithmetic with correct rounding, it is important to know if there are input floating-point values x such that $f(x)$ is either the middle of two consecutive floating-point numbers or a floating-point number: In the first case $f(x)$ is said to be a *midpoint*, and in the second case $f(x)$ is said to be an *exact point*. In [42], Claude-Pierre Jeannerod, Nicolas Louvet, Jean-Michel Muller and Adrien Panhaleux have studied the midpoints and the exact points of some usual algebraic functions: division, inversion, square root, reciprocal square root, 2D Euclidean norm and its reciprocal, and 2D normalization. The results and the techniques presented in this paper can be used to deal with both the binary and the decimal formats defined in the IEEE 754-2008 standard.

6.2.4. Binary Floating-point Operators for VLIW Integer Processors

In a joint work with H. Knochel and C. Monat (STMicroelectronics Compilation Expertise Center, Grenoble), C.-P. Jeannerod, G. Revy, and G. Villard [26] have extended the bivariate polynomial evaluation-based square rooting method of [46] to division. Compared to square root, the main difficulty was to automatically validate the numerical accuracy of the fast bivariate polynomial evaluation code that used to approximate the quotient. This required first to introduce a new set of approximation and evaluation error conditions that are sufficient to ensure correct rounding. Then, some efficient heuristics have been proposed to generate such evaluation codes and to validate their accuracy according to the new error conditions. Finally, a complete C implementation has been written (which is also part of FLIP 1.0 (§5.2)). With the ST200 VLIW compiler the speed-up factor is almost 1.8 (compared to FLIP 0.3).

C.-P. Jeannerod and G. Revy have worked on the design and implementation of a correctly-rounded reciprocal square root operator. They proposed in [27] a high-ILP algorithm as well as an efficient rounding algorithm (for rounding to nearest even). Their implementation, which fully supports subnormals, allowed to speed up the reciprocal square root of FLIP (§5.2) by a factor of 2. This compound operator is also about twice faster than a division followed by a square root, and entails only one rounding error instead of two.

6.2.5. Exact and Approximated Error of the FMA

The fused multiply-add (FMA) instruction, specified by the IEEE 754-2008 Standard for Floating-Point Arithmetic, eases some calculations, and is already available on some current processors. Sylvie Boldo (Proval Team, Saclay) and Jean-Michel Muller have first extended an earlier work on the computation of the exact error of an FMA (by giving more general conditions and providing a formal proof). Then, they have presented a new algorithm that computes an approximation to the error of an FMA, and provide error bounds and a formal proof for that algorithm [39].

6.3. Correct Rounding of Elementary Functions

Participants: Sylvain Chevillard, Florent de Dinechin, Mioara Joldes, Vincent Lefèvre, Jean-Michel Muller, Andrew Novocin.

6.3.1. Computing a certified bound of the supremum norm of an error

When implementing functions in libms, the function f to be implemented is often replaced by an approximation polynomial p on a closed bounded interval $[a, b]$. In order to guarantee the correctness of the implementation, one has to compute a tight upper bound of the supremum norm of the approximation error, being it absolute $p - f$ or relative $p/f - 1$, on the interval.

S. Chevillard, M. Joldes and C. Lauter proposed an algorithm for computing efficiently such a bound [22]. The algorithm uses automatic differentiation and interval arithmetic for overcoming the drawbacks of previous approaches. However, due mainly to the combination of several techniques that are not currently available in formal proof checkers (like automatic differentiation), this algorithm did not provide any formal proof. Also, there is no "a priori" control of the accuracy obtained for the supremum norm.

In consequence, in [41], S. Chevillard, J. Harrison, M. Joldes and Ch. Lauter proposed a novel algorithm for efficient and accurate computation of upper bounds of approximation errors. Key elements of this algorithm are the use of intermediate approximation polynomials with bounded truncation remainder and a non-negativity test based on a Sum-of-squares expression of polynomials. This algorithm is fully automated and the accuracy obtained for the result is controlled “a priori” by the user. It can handle not only the cases when f is a simple function (such as $\exp$, $\arcsin$, $\tan$, etc.) but also more complicated cases when f is obtained as a composition of functions such as $\exp(1 + \cos{(x)}^2)$ or $\sin(x)/\log(1 + x)$ for instance. It can deal uniformly with both absolute and relative errors. The algorithm focuses also on formally proving the numerical result and paves the way for formally certified supremum norms. It was implemented as a prototype using the scripting language of Sollya and is going to be integrated as a part of Sollya.

Both articles include experimental results on real-life examples.

6.3.2. Computation of function erf with correct rounding in arbitrary precision

Implementations of functions in arbitrary precision are useful in any context when the precision of the computations has to be modified on-the-fly, or when a high accuracy is required. In particular, when designing a libm (such as CRlibm), many crucial steps involve arbitrary precision computations.

S. Chevillard proposed an implementation of the special functions erf and erfc in arbitrary precision [13]. In fact, three algorithms are proposed: a heuristic can be used to choose the best algorithm depending on the desired accuracy. The implementation is proved in the very details (with explicit error bounds) and can serve as a reference for the implementation of other functions. Finally, this implementation has been compared to the reference library MPFR and proved to be more efficient, in particular when the precision becomes high.

6.4. Interval Arithmetic

Participants: Vincent Lefèvre, Hong Diep Nguyen, Nathalie Revol.

6.4.1. Efficient Implementation of Algorithms for Interval Linear Algebra

H.-D. Nguyen and N. Revol [49] proposed an algorithm for interval linear system solving, based on iterative refinement techniques and on a relaxation of the interval linear system. This algorithm is competitive with the best competitors in terms of efficiency while obtaining more accurate solutions. Ongoing work is the development of an algorithm for interval matrix product: this algorithm is a tradeoff between speed – it is almost as fast as the implementation in IntLab – and accuracy – the overestimation is smaller.

6.4.2. Standardization of Interval Arithmetic

We contributed to the creation, and now chair, an IEEE working group on the standardization of interval arithmetic. The main achievements of this working group, for the year 2009, are a working framework (notations, structure in levels), the first basic definitions (interval, arithmetic operations, elementary functions) and finally implementation issues (exception handling). We envision future discussion to address other operations on intervals (endpoints, midpoint...), on sets (union, intersection...), comparison and we will do our best to impel work on these topics.

6.5. Linear Algebra and Lattice Basis Reduction

Participants: Guillaume Hanrot, Claude-Pierre Jeannerod, Nicolas Louvet, Ivan Morel, Andrew Novocin, Xavier Pujol, Gilles Villard.

6.5.1. Elimination-free Algorithms for Cauchy- and Vandermonde-like matrices

In [47] Claude-Pierre Jeannerod, Christophe Mouilleron, and Gilles Villard have studied asymptotically fast algorithms for generating matrix inverses of families of structured matrices like those of Cauchy and Vandermonde type. To control the growth of intermediate generators, such algorithms typically rely on a compression step based on fast Gaussian elimination. In practice, this step makes the analysis more complicated and requires some care at the implementation level. The main contribution of this study is to show that this step is in fact unnecessary for Cauchy- and Vandermonde-like matrices. For such matrices, this allowed to propose new asymptotically fast inversion algorithms that are simpler to analyze and implement.

6.5.2. LLL-Reduction and Numerical Analysis

In 1982, Arjen Lenstra, Hendrik Lenstra Jr. and Laszló Lovász introduced an efficiently computable notion of reduction of basis of a Euclidean lattice that is now commonly referred to as LLL-reduction. The precise definition involves the R-factor of the QR factorisation of the basis matrix. A natural mean of speeding up the LLL reduction algorithm is to use a (floating-point) approximation to the R-factor. Xiao-Wen Chang (McGill University), Damien Stehlé and Gilles Villard [44] have investigated the accuracy of the factor R of the QR factorisation of an LLL-reduced basis. This has already proved very useful to devise LLL-type algorithms relying on floating-point approximations, as this is a key ingredient for the results of the following two subsections.

In parallel, a survey was written by Ivan Morel, Damien Stehlé and Gilles Villard, on the state-of-the art results about the use of numerical analysis within the LLL algorithm [21].

6.5.3. Floating-Point LLL-Reduction

Ivan Morel, Damien Stehlé and Gilles Villard [30] introduced a new LLL-type algorithm, H-LLL, that relies on Householder transformations to approximate the underlying Gram-Schmidt orthogonalizations. The latter computations are performed with floating-point arithmetic. They proved that a precision essentially equal to the dimension suffices to ensure that the output basis is reduced. H-LLL resembles the L^2 algorithm of Nguyen and Stehlé that relies on a floating-point Cholesky algorithm. However, replacing Cholesky's algorithm by Householder's is not benign, as their numerical behaviors differ significantly. Broadly speaking, the new correctness proof is more involved, whereas the new complexity analysis is more direct. Thanks to the new orthogonalization strategy, H-LLL is the first LLL-type algorithm that admits a natural vectorial description, which leads to a complexity upper bound that is proportional to the progress performed on the basis (for fixed dimensions).

6.5.4. Improving the LLL-Reducedness of an LLL-Reduced Basis

The LLL algorithm allows one to reduce a basis to an LLL-reduced basis in polynomial time. The quality of the obtained reduction is directly related to a parameter δ (the 3/4 factor in the original algorithm): the higher δ, the better the reduction. It was suggested by LaMacchia that one could gradually reduce the basis by first using a small value of δ and then increasing the value of δ to reach the maximum quality. Ivan Morel, Damien Stehlé and Gilles Villard designed an algorithm for the second phase, which is further reducing a basis that is already reduced. They take advantage of the knowledge of the lattice obtained from the first reduction to perform the second one. The results corresponding to this work are currently being written down.

6.5.5. Tweaking LLL for Particular Inputs

Mark van Hoeij and Andrew Novocin [51] introduced a new lattice algorithm specifically designed for some classical applications of lattice reduction. The applications are for lattice bases with a generalized knapsack-type structure, where the target vectors are boundably short. For such applications, the complexity of the algorithm improves traditional lattice reduction by replacing some dependence on the bit-length of the input vectors by some dependence on the bound for the output vectors. If the bit-length of the target vectors is unrelated to the bit-length of the input, then the algorithm is only linear in the bit-length of the input entries, which is an improvement over the quadratic complexity of floating-point LLL algorithms. To illustrate the usefulness of this algorithm, it is showed that a direct application to factoring univariate polynomials over the integers leads to the first complexity bound improvement since 1984. A second application is algebraic number reconstruction, where a new complexity bound is obtained as well.

6.5.6. Solving the Shortest Lattice Vector Problem

Xavier Pujol and Damien Stehlé [50] improved a Monte Carlo algorithm recently proposed by Daniele Micciancio and Panagiotis Voulgaris (to be published at SODA 2010), which finds a shortest non-zero vector of a given lattice in time $2^{3.199n}$ where n is the lattice dimension. Pujol and Stehlé modified the algorithm so that they could use the birthday paradox in the last stage of the algorithm. They achieve a time complexity bound of $2^{2.465n}$.

6.6. Code and Proof Synthesis

Participants: Florent de Dinechin, Claude-Pierre Jeannerod, Jingyan Jourdan Lu, Vincent Lefèvre, Nicolas Louvet, Christophe Mouilleron, David Pfannholzer, Nathalie Revol, Guillaume Revy, Philippe Théveny, Gilles Villard.

6.6.1. LEMA: a representation language for mathematical expressions

A study has begun to establish and implement a language that enables the programmer to represent not only the numerical code, but also the intended semantics and the programmer's knowledge. This language is based on MathML, which enables to represent mathematical content, and it adds to MathML properties such as range and error bounds, taking the underlying arithmetic into account. This language should also facilitate the transformation of the original code into another code which can be more efficient and/or more accurate for instance. This work takes place in the framework of the ANR project EVA-Flo.

7. Contracts and Grants with Industry

7.1. Grant from Minalogic/EMSOC

Participants: Christman Fagbohoun, Claude-Pierre Jeannerod, Jingyan Jourdan Lu, Jean-Michel Muller, Andrew Novocin, Guillaume Revy, Philippe Théveny, Gilles Villard.

This project is headed by C.-P. Jeannerod and J.-M. Muller. From October 2006 to September 2009, we have been involved in Sceptre, a project of the EMSOC cluster of the Minalogic Competitivity Centre. This project, led by STMicroelectronics, aims at providing new techniques for implementing software on system-on-chips. Within Arénaire, we are focusing on the generation of optimized code for accurate evaluation of mathematical functions; our partner at STMicroelectronics is the Compiler Expertise Center (Grenoble).

7.2. Grant from Région Rhône-Alpes

Participants: Nicolas Brisebarre, Sylvain Chevillard, Claude-Pierre Jeannerod, Mioara Joldes, Jingyan Jourdan Lu, Jean-Michel Muller, Nathalie Revol, Guillaume Revy, Gilles Villard.

Since October 2008, we have obtained a 3-year grant from Région Rhône-Alpes. That grant funds a PhD student, Mioara Joldes. The project consists in automating as much as possible the generation of code for approximating functions. Instead of calling functions from libraries, we wish to elaborate approximations at compile-time, in order to be able to directly approximate compound functions, or to take into account some information (typically, input range information) that might be available at that time. In this project, we collaborate with the STMicroelectronics' Compilation Expertise Center in Grenoble (C. Bertin, H. Knochel, and C. Monat). STMicroelectronics is funding another PhD grant on these themes (see the next item).

7.3. STMicroelectronics CIFRE Ph.D. Grant

Participants: Claude-Pierre Jeannerod, Jingyan Jourdan Lu, Jean-Michel Muller.

Jingyan Jourdan Lu is supported by a CIFRE Ph.D. Grant (Mar. 2009-Feb. 2012) from STMicroelectronics (Compilation Expertise Center, Grenoble) on the theme of arithmetic code generation and specialization for embedded processors. Advisors: C.-P. Jeannerod and J.-M. Muller (Arénaire), C. Monat (STMicroelectronics).

7.4. Mediacom Project

Participants: Florent de Dinechin, Claude-Pierre Jeannerod, Jingyan Jourdan Lu, Jean-Michel Muller, David Pfannholzer, Nathalie Revol.

We have been involved in Mediacom since September 1, 2009. Mediacom is a 40-month joint project with the Compiler Expertise Center (STMicroelectronics Grenoble) and INRIA project-teams Alchemy, Alf, and Compsys, and a Nano 2012 partner project. For Arénaire, it funds in particular the 3-year MEFI PhD grant of David Pfannholzer. Our long-term goal with this project is the design and implementation of a dynamic code generation tool, for numerical kernels typical of intensive mediaprocessing, and that could be integrated into production compilers.

7.5. Stone Ridge Technology donation

Participants: Florent de Dinechin, Bogdan Pasca.

Stone Ridge Technology has donated a RDX-11 FPGA-based accelerator board to support the FloPoCo project.

8. Other Grants and Activities

8.1. National Initiatives

8.1.1. ANR EVA-Flo Project

The EVA-Flo project (Évaluation et Validation Automatiques de calculs Flottants, 2006-2010) is headed by N. Revol (Arénaire). The other teams participating in this project are Dali (Eliaus, U. Perpignan), Measi (LIST, CEA Saclay) and Tropics (INRIA Sophia-Antipolis).

This project focuses on the way a mathematical formula is evaluated in floating-point arithmetic. The approach is threefold: study of algorithms for approximating and evaluating mathematical formulae, validation of such algorithms, and automation of the process.

The EVA-Flo project appears on *Cahiers de l'ANR no 3* and will be the subject of a talk and a poster during the ANR STIC colloque "Quelle recherche pour les STIC de demain?" in January 2010.

We had a meeting in Lyon in September this year. The work done so far encompasses the definition of the LEMA language, cf §6.6. Philippe Théveny has been hired to implement LEMA and interfaces between LEMA and software related to EVA-Flo topics.

8.1.2. ANR LaRedA Project

Participant: Ivan Morel.

The LaRedA project (Lattice Reduction Algorithms, 2008-2010) is funded by the ANR and headed by Brigitte Vallée (CNRS/GREYC) and Valérie Berthé (CNRS/LIRMM). The aim of the project is to finely analyze lattice reduction algorithms such as LLL, by using experiments, probabilistic tools and dynamic analysis. Among the major goals are the average-case analysis of LLL and its output distribution. In Lyon, we concentrate on the experimental side of the project (by using fpLLL and MAGMA) and the applications of lattice reduction algorithms.

8.1.3. ANR TCHATER Project

Participants: Florent de Dinechin, Honoré Takeugming, Gilles Villard.

The TCHATER project (Terminal Cohérent Hétérodyne Adaptatif TEmps Réel, 2008-2010) is a collaboration between Alcatel-Lucent France, E2V Semiconductors, GET-ENST and the INRIA Arénaire and ASPI project/teams. Its purpose is to demonstrate a coherent terminal operating at 40Gb/s using real-time digital signal processing and efficient polarization division multiplexing. In Lyon, we study the FPGA implementation of specific algorithms for polarization demultiplexing and forward error correction with soft decoding.

8.1.4. PHC Sakura - INRIA Ayame junior program (France-Japan)

Participants: Nicolas Brisebarre, Guillaume Hanrot.

Software and Hardware Components for Pairing-Based Cryptography, 2 years (2008-2009), headed by G. Hanrot (LORIA) and E. Okamoto (LCIS, Univ. of Tsukuba).

Keywords: cryptography, (hyper)elliptic curves, pairing, finite field arithmetic, hardware accelerator, FPGA.

Participant: N. Brisebarre.

The project "Software and Hardware Components for Pairing-Based Cryptography", (2008-2009), was headed by G. Hanrot (CACAO/LORIA then Arénaire/LIP) and E. Okamoto (LCIS, Univ. of Tsukuba).

The participants belong to the Arénaire (LIP, ENS Lyon) and CACAO (LORIA, Nancy) teams for the French part and to the LCIS (Univ. of Tsukuba, Japan) and the Future Univ. of Hakodate (Japan). The goal of this project was the enhancement of software and hardware implementations of pairings defined over algebraic curves. We worked on the development of a more efficient arithmetic over Jacobian of (hyper)elliptic curves, implementing genus 2 curve based pairings and designing algorithms and implementations resistant to side channel attacks.

8.1.5. CNRS Grant PEPS ELRESAS

Participants: Nicolas Brisebarre, Sylvain Chevillard, Guillaume Hanrot, Mioara Joldes, Andrew Novocin, Xavier Pujol, Gilles Villard.

The project ELRESAS (Réduction de réseaux : solutions exactes et approchées. Applications à la cryptologie et à l'arithmétique des ordinateurs) is within the 2009 program Programmes Exploratoires Pluridisciplinaires (PEPS) of the ST2I department of the CNRS. The participants belong to the Arénaire (LIP, ENS Lyon), CACAO (LORIA, Nancy), CASYS (LJK, Grenoble), MOAIS (LIG, Grenoble) and "Théorie des nombres" (Inst. C. Jordan, Univ. Lyon 1) teams. The general goal is the improvement of the algorithmic of euclidean lattices and some of its applications.

8.1.6. *CNRS Grant Échange de chercheurs*

Participant: Gilles Villard.

In 2008, G. Villard obtained a grant to visit the Magma team at the University of Sydney, Australia. The visit will take place within the end of 2009. *[VL: À mettre à jour.]*

8.1.7. *Rhône-Alpes Region Grant Exploradoc*

Participant: Ivan Morel.

I. Morel obtained this funding (Jul. 2008 - Jul. 2010) to cover the costs of his cotutelle PhD thesis between ENS Lyon (local advisor: G. Villard) and the University of Sydney (local advisor: J. Cannon). His research focuses on the development, analysis and implementation of fast LLL-type lattice reduction algorithms.

8.2. International Initiatives

8.2.1. *Contributions to Standardization Bodies*

V. Lefèvre participates in the Austin Common Standards Revision Group (for the revision of POSIX IEEE Std 1003.1).

N. Revol chairs the IEEE P1788 working group on the standardization of interval arithmetic. There were sessions dedicated to this work at the following conferences: ARITH-19 (19th IEEE Symposium on Computer Arithmetic, USA, June 2009), PPAM (8th International Conference on Parallel Processing and Applied Mathematics, Poland, September 2009) and during the 09471 Dagstuhl Seminar on Computer-assisted proofs - tools, methods and applications (Germany, November 2009).

9. Dissemination

9.1. Conferences, Edition

F. de Dinechin was in the program committee of FPL'09 (Field Programmable Logic and Applications) and FPT'09 (Field-Programmable Technologies). He is a member of the Steering Committee of the *Symposium en Architectures de Machines*.

J.-M. Muller was in the program committee of ARITH-19 (19th IEEE Symposium on Computer Arithmetic, June 2009), and in the program committee of ASAP'2009 (20th IEEE International Conference on Application-specific Systems, Architectures and Processors). With P. Montuschi (Politecnico di Torino), E. Schwarz (IBM), P. Kornerup (Odense Univ.), he was guest editor of a special section on Computer Arithmetic, published in the February issue of IEEE Transactions on Computers. He is a member of the steering committee of the RNC series of conferences. He is a member of the *board of foundation editors* of JUCS (*Journal for Universal Computer Science*).

G. Villard is in the editorial board of the Journal of Symbolic Computation.

9.2. Doctoral School Teaching

N. Brisebarre gave a 16h Master course "Cryptographic Algorithms" (November 2009) at the ENSIAS of Rabat (Morocco).

F. de Dinechin gave a 32h ÉNSL Master course “Computer Arithmetic”.

G. Hanrot, V. Lefèvre and J.-M. Muller gave a 24h ÉNSL Master course “Multiple-Precision Arithmetic” (fall 2009).

G. Hanrot gave a 24h ÉNSL Master course “Lattice basis reduction and applications” (fall 2009).

C.-P. Jeannerod, N. Louvet, and N. Revol gave a 24h ÉNSL Master course “Certified Linear Algebra” (fall 2009).

V. Lefèvre gives a 24h Master course "Computer Arithmetic" at Université Claude Bernard - Lyon 1 (2009/2010).

N. Louvet gave a 37h Master course "Numerical algorithms in floating-point arithmetic" at U. Claude Bernard - Lyon 1 (spring 2009).

J.-M. Muller gave a 10h ÉNSL Master course "Multiple-Precision Arithmetic" (fall 2009).

9.3. Other Teaching and Service

F. de Dinechin gave a 32h License course "Architecture, networks and systems". He is responsible of the international exchanges of the computer science department at ENS Lyon.

N. Louvet gave 97h of License courses during the academic year 2008/2009 at U. Claude Bernard - Lyon 1.

9.4. Leadership within Scientific Community

F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, N. Louvet, and N. Revol organized the *3es Rencontres "Arithmétique de l'Informatique Mathématique" (RAIM'09)* at ENS-Lyon (3 days, over 70 participants).

C.-P. Jeannerod co-organized (jointly with F. Rastello) a one-day workshop at ENS Lyon on the theme of *Compiler Optimization for Embedded Systems* (June 15, 2009 - about 25 participants).

J.-M. Muller has been "chargé de mission" at the ST2I department of CNRS from April 2006 to September 2009.

N. Revol chairs the IEEE P1788 working group on the standardization of interval arithmetic since November 2008.

G. Villard has been vice chair of the LIP laboratory until December 2008, and is chair starting January 2009.

G. Villard and J.M. Muller are members of the board of the CNRS-GDR Informatique Mathématique (headed by B. Vallée). The working group "Calcul formel, arithmétique, protection de l'information, géométrie" of this GDR is headed by G. Villard.

9.5. Committees

N. Brisebarre was a member of the board of examiners for the PhD defense of S. Chevillard (July 2009).

N. Brisebarre has been examiner for the ÉNS admissions (spring - summer 2009).

F. de Dinechin has been a reviewer for the habilitation thesis of L.-S. Didier (U. Pierre et Marie Curie, 2009).

G. Hanrot is the vice-president of INRIA Evaluation Committee.

G. Hanrot is a member of INRIA "Comité d'animation du domaine thématique" for theme "Algorithmics, Programming, Software and Architecture".

C.-P. Jeannerod and G. Villard were members of the board of examiners for the PhD defense of G. Revy (December 2009).

J.-M. Muller participated to the evaluation committees of the following laboratories: GSCOP (Grenoble, January 2008), LIP6 (Paris, January 2008), LIPN (Villetaneuse, January 2008), LIX (Palaiseau, February 2008), LORIA (Nancy, February 2008), LSV (Cachan, December 2008), LIFL (Lille, December 08).

J.-M. Muller is a member of the Scientific Committee of Grenoble INP (former INPG) since 2008.

J.-M.Muller was reviewer of the PhD dissertation of A.Vazquez (Univ. Santiago de Compostela, 2009). He was a member of the board of examiners for the PhD defense of S. Chevillard.

N. Revol belonged to the hiring committees for junior scientists at INRIA Grenoble - Rhône-Alpes in 2009. She belongs to the INRIA Rhône-Alpes committee for recruiting post-doctoral researchers since 2007.

G. Villard was in the habilitation committee of L. Imbert (U. Montpellier), and member of the board of examiners for the PhD defense of O. Bouissou (École Polytechnique).

G. Villard was a member of the evaluation committee of the ANR program Défis.

F. de Dinechin and J.M. Muller manage the PhD and habilitation theses for the LIP.

9.6. Seminars, Conference and Workshop Committees, Invited Conference Talks

National meetings:

F. de Dinechin gave a lecture at the spring school ARCHI'09 (Pleumeur-Bodou, spring 2009).

V. Lefèvre gave a presentation of MPFR at the CNC'2 summer school (*Certified Numerical Computation*) at LORIA (June 2009).

I. Morel gave a talk at the *École Jeunes Chercheurs en Informatique* (Clermont-Ferrand, April 2009).

J.-M. Muller gave a 3-hour invited lecture at the conference "Numeration: Mathematics and Computer Science" (Marseille/Luminy, march 2009). He gave another talk at an "open meeting" organized by STMicroelectronics (Grenoble), in January 2009.

N. Revol gave a talk at a MEA meeting, in December 2009. MEA (Méthodes Ensemblistes pour l'Automatique) is a working group of the GDR MACS on Control Theory.

International seminars and meetings:

N. Revol gave a talk at the Dagstuhl Seminar 09471 on Computer-assisted proofs - tools, methods and applications, Germany, November 2009.

10. Bibliography

Major publications by the team in recent years

[1] A. Bostan, C.-P. Jeannerod, É. Schost. *Solving structured linear systems with large displacement rank*, in "Theoretical Computer Science", vol. 407, n^{o} 1:3, November 2008, p. 155–181.

[2] N. Brisebarre, J.-M. Muller, A. Tisserand. *Computing machine-efficient polynomial approximations*, in "ACM Transactions on Mathematical Software", vol. 32, n^{o} 2, June 2006, p. 236–256.

[3] J. Detrey, F. de Dinechin. *Parameterized floating-point logarithm and exponential functions for FPGAs*, in "Microprocessors and Microsystems, Special Issue on FPGA-based Reconfigurable Computing", vol. 31, n^{o} 8, December 2007, p. 537–545, http://dx.doi.org/10.1016/j.micpro.2006.02.008.

[4] G. Hanrot, V. Lefèvre, D. Stehlé, P. Zimmermann. *Worst Cases of a Periodic Function for Large Arguments*, in "Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH-18)", IEEE computer society, 2007, p. 133–140, http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=4272859.

[5] G. Hanrot, D. Stehlé. *Improved Analysis of Kannan's Shortest Lattice Vector Algorithm (Extended Abstract)*, in "Proceedings of Crypto 2007", LNCS, vol. 4622, Springer, 2007, p. 170–186.

[6] C.-P. Jeannerod, G. Villard. *Essentially optimal computation of the inverse of generic polynomial matrices*, in "Journal of Complexity", vol. 21, n^{o} 1, 2005, p. 72–86.

[7] P. Kornerup, C. Q. Lauter, V. Lefèvre, N. Louvet, J.-M. Muller. *Computing Correctly Rounded Integer Powers in Floating-Point Arithmetic*, in "ACM Transactions on Mathematical Software", vol. 37, n^{o} 1, 2009, To appear.

[8] J.-M. MULLER, N. BRISEBARRE, F. DE DINECHIN, C.-P. JEANNEROD, V. LEFÈVRE, G. MELQUIOND, N. REVOL, D. STEHLÉ, S. TORRES. *Handbook of Floating-Point Arithmetic*, Birkhäuser Boston, December 2009, http://prunel.ccsd.cnrs.fr/ensl-00379167/en/, ISBN: 978-0-8176-4704-9.

[9] N. REVOL, K. MAKINO, M. BERZ. *Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY*, in "Journal of Logic and Algebraic Programming", vol. 64, 2005, p. 135–154.

[10] F. DE DINECHIN, C. Q. LAUTER, J.-M. MULLER. *Fast and correctly rounded logarithms in double-precision*, in "Theoretical Informatics and Applications", vol. 41, 2007, p. 85-102.

Year Publications

Doctoral Dissertations and Habilitation Theses

[11] S. CHEVILLARD. *Évaluation efficace de fonctions numériques - Outils et exemples*, École Normale Supérieure de Lyon, France, July 2009, Ph. D. Thesis.

[12] G. REVY. *Implementation of binary floating-point arithmetic on integer processors: polynomial evaluation-based algorithms and certified code generation*, Université de Lyon - École normale supérieure de Lyon, France, December 2009, Ph. D. Thesis.

Articles in International Peer-Reviewed Journal

[13] S. CHEVILLARD. *The functions erf and erfc computed with arbitrary precision*, in "Information and Computation", 2010, http://prunel.ccsd.cnrs.fr/ensl-00356709, accepted with the status major revision.

[14] M. ERCEGOVAC, J.-M. MULLER. *An Efficient Method for Evaluating Complex Polynomials*, in "Journal of Signal Processing Systems", vol. 58, n^{o} 1, 2010, p. 17–27.

[15] S. GRAILLAT, J.-L. LAMOTTE, H. D. NGUYEN. *Extended precision with a rounding mode toward zero environment. Application on the CELL processor*, in "International Journal of Reliability and Safety", vol. 3, n^{o} 1/2/3, 2009, p. 153–173, Special issue on "Reliable Engineering Computing".

[16] S. GRAILLAT, PH. LANGLOIS, N. LOUVET. *Algorithms for Accurate, Validated and Fast Polynomial Evaluation*, in "Japan Journal of Industrial and Applied Mathematics", 2010, To appear.

[17] P. KORNERUP, C. Q. LAUTER, V. LEFÈVRE, N. LOUVET, J.-M. MULLER. *Computing Correctly Rounded Integer Powers in Floating-Point Arithmetic*, in "ACM Transactions on Mathematical Software", vol. 37, n^{o} 1, 2010, To appear DK .

[18] C. Q. LAUTER, V. LEFÈVRE. *An efficient rounding boundary test for pow(x, y) in double precision*, in "IEEE Transactions on Computers", vol. 58, n^{o} 2, February 2009, p. 197–207, http://doi.ieeecomputersociety.org/10.1109/TC.2008.202.

[19] P. Q. NGUYEN, D. STEHLÉ. *An LLL Algorithm with Quadratic Complexity*, in "SIAM Journal on Computing", vol. 39, n^{o} 3, 2009, p. 874–903.

[20] P. Q. NGUYEN, D. STEHLÉ. *Low-dimensional lattice basis reduction revisited*, in "ACM Transactions on Algorithms", 2009, http://hal.inria.fr/inria-00328629/en/.

Articles in National Peer-Reviewed Journal

[21] I. MOREL, D. STEHLÉ, G. VILLARD. *Analyse numérique et réduction des réseaux*, in "Technique et Science Informatiques", 2010, To appear.

International Peer-Reviewed Conference/Proceedings

[22] S. CHEVILLARD, M. JOLDES, C. Q. LAUTER. *Certified and fast computation of supremum norms of approximation errors*, in "19th IEEE Symposium on Computer Arithmetic (ARITH-19), Portland, Oregon, U.S.A.", 2009, p. 169–176, http://prunel.ccsd.cnrs.fr/ensl-00334545/.

[23] P. DORMIANI, M. ERCEGOVAC, J.-M. MULLER. *Design and Implementation of a Radix-4 Complex Division Unit with Prescaling*, in "Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2009), Boston, U.S.A.", IEEE Computer Society, 2009, http://prunel.ccsd.cnrs.fr/ensl-00379147/en/, 9 pages.

[24] P. DORMIANI, M. ERCEGOVAC, J.-M. MULLER. *Low Precision Table Based Complex Reciprocal Approximation*, in "Proceedings of the 43rd Asilomar Conference on signals, systems and computers, Pacific Grove, California, USA", November 2009, 5 pages.

[25] S. GRAILLAT, J.-L. LAMOTTE, H. D. NGUYEN. *Error-Free Transformation in rounding mode toward zero*, in "Dagstuhl Seminar on Numerical Validation in Current Hardware Architectures", Lecture Notes in Computer Science, vol. 5492, 2009, p. 217–229.

[26] C.-P. JEANNEROD, H. KNOCHEL, C. MONAT, G. REVY, G. VILLARD. *A new binary floating-point division algorithm and its software implementation on the ST231 processor*, in "Proceedings of the 19th IEEE Symposium on Computer Arithmetic (ARITH-19), Portland, OR", June 2009, p. 95–103.

[27] C.-P. JEANNEROD, G. REVY. *Optimizing correctly-rounded reciprocal square roots for embedded VLIW cores*, in "Proceedings of the 43rd Asilomar Conference on signals, systems and computers, Pacific Grove, California, USA", November 2009, 5 pages.

[28] R. B. KEARFOTT, J. D. PRYCE, N. REVOL. *Discussions on an Interval Arithmetic Standard at Dagstuhl Seminar 08021*, in "Dagstuhl Seminar on Numerical Validation in Current Hardware Architectures", Lecture Notes in Computer Science, vol. 5492, 2009, p. 1–6 US GB .

[29] P. KORNERUP, V. LEFÈVRE, N. LOUVET, J.-M. MULLER. *On the Computation of Correctly-Rounded Sums*, in "19th IEEE Symposium on Computer Arithmetic (ARITH-19), Portland, Oregon, U.S.A.", 2009, p. 155-160, http://hal.inria.fr/inria-00367584/en/DK.

[30] I. MOREL, D. STEHLÉ, G. VILLARD. *H-LLL: Using Householder inside LLL*, in "Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC 2009), Seould, Korea", ACM Press, August 2009, p. 271–278.

[31] D. STEHLÉ, R. STEINFELD, K. TANAKA, K. XAGAWA. *Efficient Public-Key Encryption Based on Ideal Lattices (extended abstract)*, in "Proceedings of Asiacrypt 2009", Lecture Notes in Computer Science, vol. 5912, Springer-Verlag, 2009, p. 617–635, http://dx.doi.org/10.1007/978-3-642-10366-7_36.

[32] F. DE DINECHIN, C. KLEIN, B. PASCA. *Generating high-performance custom floating-point pipelines*, in "Proceedings of the 19th International Conference on Field Programmable Logic and Applications", IEEE, August 2009, http://prunel.ccsd.cnrs.fr/ensl-00379154/en/, 6 pages.

[33] F. DE DINECHIN, B. PASCA. *Large multipliers with less DSP blocks*, in "Proceedings of the 19th International Conference on Field Programmable Logic and Applications", IEEE, August 2009, http://prunel.ccsd.cnrs.fr/ensl-00356421/en/, 9 pages.

National Peer-Reviewed Conference/Proceedings

[34] F. DE DINECHIN, M. JOLDES, B. PASCA, G. REVY. *Racines carrées multiplicatives sur FPGA*, in "13 e SYMPosium en Architectures nouvelles de machines (SYMPA), Toulouse", September 2009, 10 pages.

Workshops without Proceedings

[35] H. D. NGUYEN, N. REVOL. *Relaxed method to certify the solution of a linear system*, in "SWIM'09 (Small Workshop on Interval Methods), Lausanne, Switzerland", June 2009.

Scientific Books (or Scientific Book chapters)

[36] J.-M. MULLER, N. BRISEBARRE, F. DE DINECHIN, C.-P. JEANNEROD, V. LEFÈVRE, G. MELQUIOND, N. REVOL, D. STEHLÉ, S. TORRES. *Handbook of Floating-Point Arithmetic*, Birkhäuser Boston, 2009, http://prunel.ccsd.cnrs.fr/ensl-00379167/en/, ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

[37] D. STEHLÉ. *Floating-point LLL: theoretical and practical aspects*, Information Security and Cryptography, Springer, 2009.

Books or Proceedings Editing

[38] P. KORNERUP, P. MONTUSCHI, J.-M. MULLER, E. SCHWARZ (editors). *Special Section on Computer Arithmetic*, IEEE Transactions on Computers, vol. 58, n^{o} 2, February 2009, http://prunel.ccsd.cnrs.fr/ensl-00383561/en/.

Research Reports

[39] S. BOLDO, J.-M. MULLER. *Exact and Approximated error of the FMA*, INRIA, 2009, http://hal.archives-ouvertes.fr/inria-00429617/en/, Soumis à IEEE-TC, Research report.

[40] S. CHEVILLARD. *The functions erf and erfc computed with arbitrary precision*, n^{o} RR2009-04, Laboratoire de l'Informatique du Parallélisme (LIP), 46, allée d'Italie, 69 364 Lyon Cedex 07, January 2009, http://prunel.ccsd.cnrs.fr/ensl-00356709, Research report.

[41] S. CHEVILLARD, J. HARRISON, M. JOLDES, C. Q. LAUTER. *Efficient and accurate computation of upper bounds of approximation error s*, n^{o} RR2009-, Laboratoire de l'Informatique du Parallélisme (LIP), 46, allée d'Italie, 69 364 Lyon Cedex 07, December 2009, Research report.

[42] C.-P. JEANNEROD, N. LOUVET, J.-M. MULLER, A. PANHALEUX. *Midpoints and exact points of some algebraic functions in floating-point arithmetic*, n^{o} ensl-00409366, LIP, École Normale Supérieure de Lyon, 2009, http://prunel.ccsd.cnrs.fr/ensl-00409366, Research report.

Scientific Popularization

[43] V. LEFÈVRE, J.-M. MULLER. *Erreurs en arithmétique des ordinateurs*, in "Images des mathématiques", June 2009, http://images.math.cnrs.fr/Erreurs-en-arithmetique-des.html.

Other Publications

[44] X.-W. CHANG, D. STEHLÉ, G. VILLARD. *Perturbation Analysis of the QR factor R in the Context of LLL Lattice Basis Reduction*, 2009, Submitted.

[45] K. R. GHAZI, V. LEFÈVRE, P. THÉVENY, P. ZIMMERMANN. *Why and how to use arbitrary precision*, 2009, Submitted to Computing in Science and Engineering.

[46] C.-P. JEANNEROD, H. KNOCHEL, C. MONAT, G. REVY. *Computing floating-point square roots via bivariate polynomial evaluation*, 2009, Submitted.

[47] C.-P. JEANNEROD, C. MOUILLERON, G. VILLARD. *Extending Cardinal's algorithm to a broader class of structured matrices*, July 2009, Poster at ISSAC 2009.

[48] J.-M. MULLER. *Exact computations with an arithmetic known to be approximate*, 2009, Invited lectures (3 hours) at the conference Numeration: Mathematics and Computer Science, CIRM, Marseille, Mar. 2009.

[49] H. D. NGUYEN, N. REVOL. *Solving and Certifying a Linear System*, 2009, Submitted to Reliable Computing.

[50] X. PUJOL, D. STEHLÉ. *Solving the Shortest Lattice Vector Problem in Time* $2^{2.465n}$, 2009, http://eprint.iacr.org/2009/605, IACR eprint.

[51] M. VAN HOEIJ, A. NOVOCIN. *Gradual sub-lattice reduction and a new complexity for factoring polynomials*, 2010, To appear in the proceedings of LATIN'10.

References in notes

[52] T. J. HICKEY, Q. JU, M. H. VAN EMDEN. *Interval arithmetic: From principles to implementation*, in "Journal of the ACM", vol. 48, n^{o} 5, 2001, p. 1038-1068, http://doi.acm.org/10.1145/502102.502106.

[53] E. KALTOFEN, G. VILLARD. *On the complexity of computing determinants*, in "Computational Complexity", vol. 13, 2004, p. 91–130 US .

[54] J.-M. MULLER. *Elementary Functions, Algorithms and Implementation*, Birkhäuser Boston, 2nd Edition, 2006.

[55] F. DE DINECHIN, A. TISSERAND. *Multipartite table methods*, in "IEEE Transactions on Computers", vol. 54, n^{o} 3, 2005, p. 319-330.