Acronyme	TaMaDi		
Titre du projet en français	Dilemme du Fabricant de Tables		
Titre du projet en anglais	Table Maker's Dilemma		
Comité d'Evaluation référence (CE)	SIMI 2 *(p. 3 de l'appel à projets)*		
Projet multidisciplinaire	☐ OUI ☒ NON Si oui, indiquez l'intitulé du second CE		
Coopération internationale (si applicable)	Non		
Aide totale demandée	423033 €	Durée du projet	36 mois

Contents

1 Contexte et positionnement du projet / Context and positioning of the proposal

Context. In floating-point (FP) arithmetic having fully-specified "atomic" operations is a key-requirement for portable, predictable and provable numerical software. Since 1985, the four arithmetic operations ($+$, $-$, $\times$, $\div$) and the square root are IEEE specified (it is required that they should be *correctly rounded*: roughly speaking, the system must always return the floating-point number nearest the exact mathematical result of the operation). This is not fully the case for the basic mathematical functions (sine, cosine, exponential, etc.). Indeed, the same function, on the same variable, with the same format, may return significantly different results depending on the environment. As a consequence, numerical programs using these functions suffer from various problems. Among them:

- it is almost impossible to estimate their accuracy (since little is known about what is returned by the implemented mathematical functions);
- their portability is difficult to guarantee (a program which works on one system may be too inaccurate—or may even crash—on different systems).

The lack of specification (partly solved, thanks to our earlier work, in the recent IEEE 754-2008 standard for floating-point arithmetic) is due to a problem called the *Table Maker's Dilemma* (TMD). To compute $f(x)$ in a given format, where x is a FP number, we must first compute an approximation to $f(x)$ with a given precision, which we round to the nearest FP number in the considered format. The problem is the following: finding what the accuracy of the approximation must be to ensure that the obtained result is always equal to the "exact" $f(x)$ rounded to the nearest FP number.

To solve that problem we have to locate, for each considered floating-point format and for each considered function f, the *hardest to round* (HR) points, that is, what is the floating-point numbers x such that $f(x)$ is closest to the exact middle of two consecutive floating-point numbers (we call such a middle a *breakpoint*). The naive method of finding these points (evaluating the function with large precision at each floating-point number) is far too impractical. This project aims at providing the following:

- for the most common formats and functions, tables of HR points, along with checkable certificates that show that the given values are indeed the correct ones;
- open source software which allows the computation of HR points for less common functions and formats.

State of the art and positioning The state-of-the art in the domain can be summarized as follows:

- IBM released a library, libultim, that claimed correct rounding in binary double precision[1] (53 bits). Since this was done without knowledge of the HR points, the necessary precision for the intermediate calculation was grossly overestimated, which resulted in very poor performance;
- the Arénaire team (one of the applicants) and the CACAO (formerly named SPACES) team of LORIA Lab. (Nancy) have designed algorithms, and obtained and published HR points for some functions in double precision. For instance, one of the two hardest-to-round cases for radix-2 exponentials in the full double-precision range [41] is

$$1.11100100010110010110010100100110101111111100101001101 \times 2^{-10}$$

whose radix-2 exponential is

$$\overbrace{1.00000000010100111111110000101110110000101011010010011}^{53\text{ bits}}$$
$$0\underbrace{1111111111111111\cdots1111111111111111}_{59\text{ ones}}0100\cdots.$$

[1] http://web.archive.org/web/20050207110205/http://oss.software.ibm.com/mathlib/

As one can see, it is very close to the middle of two consecutive floating-point numbers. The (heuristic) complexity of our methods is an exponential function of the number of bits of the FP format, so that they cannot be applied to formats significantly larger than double (or double extended) precision;

- in the recent years, using the obtained HR points, Arénaire developed a library, CRlibm [39, 30, 7], that implements the most common functions in double precision. Each function comes with a proof of its correctness. CRlibm offers 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 does not impose) correct rounding for some mathematical functions: the new 754-2008 standard for floating-point arithmetic was adopted in August 2008. The influence of Arénaire's research on the elementary functions in that standard clearly shows: *among the 22 bibliographic references listed in the standard [16], 7 are authored by Arénaire's members*.

Challenges As explained before, our methods cannot be used in big precisions. And yet, the IEEE 754-2008 standard specifies, for instance a binary128 ("quad precision") and a decimal128 (128-bit decimal) formats. Also, the processes that generate our HR cases are based on complex and very long calculations (years of cumulated CPU time) that inevitably cast some doubt on the correctness on their results. Hence, we need to fully reconsider the methods used to get HR points, and mainly focus on three aspects:

- **big precisions:** we must get new algorithms for dealing with precisions larger than double precision. Such precisions will become more and more important (even if double precision may be thought as more than enough for a final result, it may not be sufficient for the intermediate results of a long or critical calculations);
- **formal proof:** we must provide formal proofs of the critical parts of our methods. Another possibility is to have our programs generating certificates that show the validity of their results. We should then focus on proving the certificates;
- **aggressive computing:** the methods we have designed for generating HR points in double precision require weeks of computation on hundreds of PCs. Even if we design faster algorithms, we must massively parallelize our methods, and study various ways of doing that.

We must point out that we aim at getting *practical results*: tables of HR points, programs for getting them, formal proofs of their correctness, with the consequence of faster and better mathematical function programs, and an influence on future standards for FP arithmetic. We will of course appreciate to publish better complexity results, but this is not our major goal.

Getting these HR points will make it possible to design very efficient mathematical function libraries, leading the path to a full specification of the most common functions in FP arithmetic. Indeed, the only big obstacle for an almost complete automation of the design of these libraries is the need to know the hardest-to-round cases.

2 Description scientifique et technique / Scientific and technical description

2.1 Background, state of the art

2.1.1 Correct rounding

In general, the result of an operation (or function) on floating-point (FP) numbers is not exactly representable in the FP system being used, so it has to be *rounded*. In the earliest floating-point systems, the way results were rounded was not specified. One of the most interesting ideas brought out by the IEEE 754-1985 Standard for Floating-Point Arithmetic [1] was the concept of *rounding mode*: how a numerical value is rounded to a finite (or, possibly, infinite) FP number is specified by a rounding mode, that defines a rounding function $\circ$. For example, when computing $a + b$, where a and b are floating-point numbers, the returned result is $\circ(a + b)$. The 4 rounding modes that are defined in the IEEE 754-2008 standard are:

- round toward $-\infty$: RD(x) is the largest floating-point number less than or equal to x;
- round toward $+\infty$: RU(x) is the smallest floating-point number greater than or equal to x;
- round toward zero: RZ(x) is the closest floating-point number to x that is no greater in magnitude than x;
- round to nearest: RN(x) is the floating-point number that is the closest to x. A tie-breaking rule is chosen when x falls exactly halfway between two consecutive floating-point numbers.

Requiring correctly rounded arithmetic operations has a number of advantages. Among them, it greatly improves the portability of numerical software; it allows one to design algorithms that use this requirement (e.g. division algorithms such as those used in the Itanium [6]); this requirement can be used for designing formal proofs of software.

2.1.2 The table maker's dilemma

The IEEE 754-1985 standard for floating-point arithmetic required correctly rounded arithmetic operations. It seems natural to try to enforce the same requirement for the most common mathematical functions. This was not done in 754-1985 because of a difficulty known as the *Table Maker's Dilemma*. Assume we wish to implement a function f, and that we use a radix-2 floating-point format[2] of precision[3] p. Let us denote $\circ$ the rounding function, and call *rounding breakpoints* the values where the value of $\circ$ changes (for rounding toward $\pm\infty$ or 0, the breakpoints are the floating-point numbers; for rounding to nearest, they are the exact middle of two consecutive floating-point numbers).

The real number $f(x)$ cannot, in general, be represented with a finite number of digits. Hence, we must *approximate* it by some other function that is easier to evaluate (e.g., a polynomial or rational function), say $\hat{f}$. More precisely, what we will call $\hat{f}$ here is the *computed* approximation, not a real-valued theoretical approximation: all various rounding errors are included in the definition of $\hat{f}$, including the roundings of the coefficients of the polynomial or rational approximation and the roundings of the arithmetic operations. If x is a floating-point number, then $\hat{f}(x)$ is a number of possibly much higher precision than the "target precision" p. We assume that the very accurate but finitely precise mantissa[4] of $\hat{f}(x)$ approximates the infinitely precise mantissa of $f(x)$ within an error less than 2^{-m}, where m is significantly larger than p. The only information that we have on $f(x)$ is that it is located in some interval I, centered on $\hat{f}(x)$, of width 2^{-m+1}. We would like to always return $\circ(f(x))$, and yet the only thing we can easily return is $\circ(\hat{f}(x))$: Can we guarantee that if m is large enough, then these two values will

[2] We are also interested in radix-10 calculations, but for the sake of simplicity we limit this presentation to radix 2.

[3] For short, the precision of a floating-point system is the number of mantissa digits.

[4] That is, $\hat{f}(x)/2^{\lfloor \log_2 |\hat{f}(x)| \rfloor}$: if $x = x_0.x_1x_2x_3x_4\cdots \times 2^n$, its infinitely precise mantissa is $x_0.x_1x_2x_3x_4\cdots$

always be equal? The TMD occurs, for a given x, when the interval I contains a breakpoint, i.e., when $\circ(f(x))$ is so close to a breakpoint that, taking into account the error 2^{-m} of the approximation, it is impossible to decide whether $f(x)$ is above or below the breakpoint. In such a case, one does not know which result should be returned. So the problem to be solved is:

1. whenever possible, to find the *hardest-to-round cases*, i.e., the numbers x for which the infinitely precise mantissa of $f(x)$ is not a breakpoint and the distance between $f(x)$ and a breakpoint is minimum. The smallest m such that that minimum distance is larger than 2^{-m} will be called the *hardness-to-round* for function f;

2. or, when this is not possible, to fix a value and check whether this value is a lower bound on the distance between the infinitely precise mantissa of $f(x)$ and a breakpoint (except for the case when $f(x)$ itself is a breakpoint);

3. possibly, find a way to convince users that a huge computation producing a yes/no answer or a short list of numbers was actually correct.

Concerning part 1, we have designed an algorithm (called L-algorithm in the proposal), and obtained and published in 2001 [41, 26] hardest-to-round cases, either in the full floating-point range (exponentials and logarithms in bases e, 2, 10, functions x^n for small values of n) or in some limited range (trigonometric functions) in binary double-precision arithmetic.

2.1.3 Other teams working on similar topics

- members from the CACAO team of LORIA (Nancy) have designed and published in 2003 and 2005 another algorithm [35] (called SLZ in the proposal), of better asymptotic complexity, for finding hardest-to-round cases. It cannot really be viewed as a "concurrent" team since two of the authors of SLZ, Damien Stehlé and Vincent Lefèvre, as well as another member of CACAO (Hanrot) are now members of Arénaire;

- IBM and SUN have designed elementary function libraries that claim correct rounding. The IBM library is now obsolete, and SUN's library has significantly lower performance than Arénaire's;

- since the release of the IEEE 754-2008 standard, in August 2008, the major manufacturers are working on the design of elementary function libraries with correct rounding. Without knowing the hardest-to-round points for the considered points and formats, these libraries would be doomed to be either slow (if the necessary intermediate precision is overestimated) or incorrect (if it is underestimated). Hence, for the computer arithmetic community, computing hardest-to-round cases is a major goal. For instance the floating-point development team of Intel at Portland hired in 2008 a former PhD student of Arénaire, who was one of the two main developers of CRlibm.

This is becoming a "hot" topic and we wish to keep our leading position in addressing the new challenges of higher precisions and certification. The case of double precision arithmetic functions is solvable by anybody with enough computing power. For instance, Harrison (Intel) used one of our hardest-to-round-case search algorithms in 2009 for implementing the Bessel functions. And yet, we strongly believe that our consortium have almost unique complementary abilities for dealing with larger precisions: computer arithmetic, number theory, massive parallelism, formal proof.

2.2 Rationale highlighting the originality and novelty of the proposal

2.2.1 Major goals

As stated before, we mainly have two goals in mind:

1. actually obtain "hardest-to-round" cases, or for some functions/formats a proof that there are no cases with a hardness-to-round greater than some bound—this is what we call the "degraded" version of the problem. Due to their nature, these results must come with either easily checkable certificates that prove their validity, or with formal proofs of the critical parts of the programs that generated them;

2. give to the community reliable tools for performing the same work on other functions, domains, and formats.

We will need to work on three different aspects of the problem:

1. first, from the algorithmic/mathematical point of view, we need to investigate what can be expected concerning the "degraded' instance of (1). Also, we need to adapt the LLL algorithm (see Section 3.1.2) so that it becomes more efficient with our problem (for instance, we will follow the paths of Damien Stehlé concerning *floating-point* implementations of LLL—since we wish to check the solutions, we may accept getting sometimes wrong solutions, if this is unfrequent enough);

2. second, from the formal proof point of view: in the SLZ algorithm (or a possible extension) we must choose what kind of "certificate" would be suitable: we must for instance keep enough information to make sure that we have obtained a basis of the initial lattice (see Section 3.1.2), but it would be useless to send to end-users terabytes of certificates; we must also get more confidence in the results provided by the L-algorithm;

3. even with the best algorithms we know so far, computing hardest-to-round cases requires years of CPU time. We must use massive parallelism techniques if we wish to get results in a reasonable delay.

2.2.2 Novelty

Arénaire and the CACAO team of Loria, Nancy (three of its former members joined Arénaire) were the first groups who deeply tackled the problem of correctly rounding mathematical functions. They are still the only groups that have published hardest-to-round points. Trying to validate these results with formal proof is a new approach. Arénaire and CACAO's libraries libraries (CRlibm, the Arénaire double-precision library, and MPFR, the CACAO multiple-precision library) are proofs-of-concept that show that correct rounding is a clear improvement and can be done with very little overhead. We hope the results of this project will show that correct rounding can be achieved for all usual functions and formats. We expect to have a strong influence on the future standards for floating-point arithmetic (as we already had on the IEEE 754-2008 standard). The work we have done for library functions (e.g., $\log(x)$) could be extended to "compound" functions (e.g., $\log(1+\sqrt{1-e^{-x^2}})$). Obviously, we cannot find hardest-to-round cases for all possible compound functions, but we wish to provide the community with tools that allow a user to implement the functions he or she needs with correct rounding.

2.2.3 Expected results, with evaluation criteria

By the end of the project, we aim at producing:

- for double precision (and possibly, double-extended) and the corresponding decimal format (named decimal64 in the IEEE 754-2008 standard): we wish to get the hardest-to-round cases for all the functions of a standard C math library, and publish them on a freely accessible web page (the one of the project) and on an open archive;

- the certificates proving the validity of these cases will be freely available too (possibly, if they are too large to be posted on the web, they will be sent on-demand);

- for formats significantly larger than double precision (typically, the 128-bit binary and decimal formats), we wish to compute and publish on a freely accessible web page (the one of the project) and on an open archive, for the most common mathematical functions, a value ϵ such that if we approximate the function with relative accuracy ϵ, rounding the approximation will always be equivalent to rounding the exact value.

The parameter ϵ will be far from optimal unless we find the hardest-to-round cases (which is unlikely for these formats). As previously, for each function/format a certificate would be available;

- publication on a freely accessible web page (the one of the project) of open-source software, as portable as possible, for computing hardest-to-round cases or accuracy bounds for other formats and functions and generating the certificates;
- publication in journals and/or proceedings of international conferences of some of our results: algorithms, properties, etc.

A rather short-term consequence of these results will be that libraries of basic mathematical functions (sine, exp, arctan, etc.) will be rather easily adaptable to offer correct rounding for the most common functions and formats (currently, Arénaire's CRlibm library offers correct rounding in double precision only, and for 10 functions). *This consequence may come extremely quickly*: indeed, the only big obstacle for an almost complete automation of the design of these libraries is the need to know the hardest-to-round cases (or, at least, with degraded performances, a bound on the necessary accuracy). For instance a new function can be integrated in CRlibm in a few hours, if the hardest-to-round cases are known.

A medium-term consequence is that the next releases of the IEEE 754 standard for floating-point arithmetic will be able to require (and not only to recommend) correct rounding for some functions (typically those of a programming language such as C or FORTRAN). All basic building blocks of numerical computing will therefore be fully specified, which will significantly enhance portability and "provability" of numerical software, with important consequences in terms of cost and reliability.

3 Programme scientifique et technique, organisation du projet / Scientific and technical program, project management

3.1 Scientific program, specific aims of the proposal

The ultimate goal of the project is to conduct an in-depth study of the methods available for hardest-to-round cases. This means bringing them to a level of maturity which they are still far from reaching; developing tools suitable for dissemination and tools suitable for internal, high performance, computations; and finally managing validation of the methods and programs for those large computations, through formal proof techniques.

In this section, we describe in more detail the two algorithms under study, point out their strengths and weaknesses, and explain how the general goals presented above apply in each case.

3.1.1 Brief description of the L-algorithm

Define $N = 2^p$ and assume that function f maps $[1/2, 1]$ to $[1/2, 1]$ (elementary transformations and domain splitting always allow one to assume that). Our initial problem can be formulated as the following integer-variable problem [24]: solve the equation

$$\left| \left(N \cdot f\left(\frac{x}{N}\right) - \frac{1}{2} \right) \bmod 1 \right| \leq \frac{1}{M} \tag{1}$$

where x (the unknown) is an integer between 0 and N, M is a large integer, and $y \bmod 1$ is the distance between y and the nearest integer.

To solve (1) using a piecewise degree-1 approximation to function f, in each subdomain, the L-algorithm performs a modified (subtractive) Euclidean GCD iteration. In order to illustrate this, we display part of its main loop:

while $x < y$ **do**
  if $u + v \geq N$ **then return** (N, d) $[h = y, b \in x_r]$
  $y \leftarrow y - x$
  $u \leftarrow u + v$
end while
if $u + v \geq N$ **then return** (N, d) $[h = x, b \in x_r]$

The main strengths of the L-algorithm are related to its simplicity: it is easy to implement, and this ease also means that finely tuned optimization (e.g. assembly code) is feasible, if not easy to achieve.

On the other hand, the major weaknesses of the L-algorithm are:

- from a performance point-of-view, the necessity of having a huge number of subdomains (since in practice, a degree-1 approximation is accurate enough in very small domains only); due to this huge number of subdomains, the overhead related to one single subdomain must be kept as small as possible. Typically, the generation of the polynomial approximation must be very fast. Currently, higher-degree polynomial approximations are generated for a family of consecutive subdomains, then a linear approximation is quickly obtained for each subdomain from the higher degree domain. This somewhat reduces the ease of implementation;
- from a validation point-of-view, this tricky process that generates the coefficients of the linear approximations (it is "tricky" because it must be *very fast*, still because of the huge number of subdomains) is hard to deal with. What matters is to make sure that the approximation errors are correctly computed: if they are underestimated then we may miss hardest-to-round cases (which would make the whole process useless), and if they are overestimated too much then we have an unnecessarily large number of subdomains (hence, an unnecessarily long process for computing hardest-to-round cases). The problem also occurs with the algorithms that use approximations of larger degree (such as SLZ) but is less crucial: since there are much fewer subdomains, we can spend more time carefully computing the approximations.

3.1.2 Coppersmith's technique (used in the SLZ algorithm)

Description of Coppersmith's technique We still deal with Equation (1). As already mentioned, we replace f by a suitable polynomial approximation F; in order to achieve this, we have to restrict to a small sub-interval $x \in [x_0 - a, x_0 + a]$. Up to replacing $F(x)$ by $F(x + x_0)$, we can thus assume that $|x| \leq a$.

The problem can thus be rewritten as

$$N \cdot F\left(\frac{x}{N}\right) - z - \frac{1}{2} = 0 \bmod 1,$$

with $|x| \leq a$, $|z| \leq 1/M$. After clearing the denominators, we can thus assume we are looking for the solutions to the equation

$$P(u, v) = 0 \mod R, \tag{2}$$

where P is a polynomial, u and v are integers, $|u| < U$ and $|v| < V$.

The technique introduced by Coppersmith consists in deducing equations over the integers, which are much easier to solve. This is achieved by the following steps:

- Notice that if (u, v) satisfy (2), then for fixed α and all $i \leq \alpha, j, k$,

$$P_{i,j,k}(u, v) := R^{\alpha - i} P(u, v)^i u^j v^k = 0 \bmod R^\alpha.$$

  The same is thus true of any integer linear combination of such $P_{i,j,k}$.

- Now, if $Q(u,v) = \sum_{i \le d_1, j \le d_2} q_{i,j} u^i v^j$ is such a polynomial with

$$\sum_{i \le d_1, j \le d_2} |q_{i,j}| U^i V^j < R^\alpha, \tag{3}$$

  we have both $|u| \le U, |v| \le V \Rightarrow |Q(u,v)| < R^\alpha$ and $Q(u,v) = 0 \bmod R^\alpha$. In turn this means that we have $Q(u,v) = 0$, an equation over the integers.

- To any finite set of $P_{i,j,k}$, we can associate the lattice that they generate; using lattice basis reduction (the LLL algorithm) techniques in a suitable way, we can look for two polynomials in this lattice with small enough coefficients. A rather intricate analysis is needed to understand for which values of M, a and what sets of $P_{i,j,k}$ such polynomials do exist.

- We are finally left with a polynomial system $Q_1(u,v) = 0, Q_2(u,v) = 0$ over the integers, which can be solved either by elimination techniques or by a suitable version of Hensel lemma ("modular Newton method").

The complexity of the algorithm, for solving (1) with $N = 2^p$ is essentially

$$\mathrm{poly}(p + \log(M)) \cdot 2^{\frac{p^2}{p + \log(M)}},$$

In this complexity, the polynomial term accounts for the LLL+Hensel calls; the exponential term accounts for the number of intervals one has to use.

Here, $1/M$ is the bound that we want to achieve on hardest-to-round cases. If we are interested to find the actual hardest-to-round cases, a simple probabilist argument shows that we should choose $M \approx 2^p$. This bound shows the interest that we could have, from the computational point of view, to restrict to computing a lower bound rather than the exact solution. We discuss this in more detail in the next section.

3.1.3 Improvements to Coppersmith's technique

Getting certificates. In that method, from the computational point of view, the most time-consuming part is the call to the LLL algorithm. With small parameters (degree 3 polynomials, α small, lattices of dimension ≤ 10) this is already 95% of the computing time. Yet, an important remark is that if one is interested in *checking* the result only, we only need the *output* of this LLL algorithm (which is used as an oracle), and namely the two polynomials Q_1 and Q_2. We can then check that they verify (3), and recompute the roots of the corresponding system, which is cheap. We can thus expect to compute a *certificate*, which for each interval would mainly contain the pair (Q_1, Q_2) (plus some tuning parameters chosen for this interval). This part is of course to be explored in close relationship with the formal proof aspects (Section 3.1.5).

Identifying the right parameters. In the "usual" version of the problem (i.e., we wish to actually get hardest-to-round cases), for which we have good probabilistic reasons to choose $M \approx 2^p$, the complexity is essentially a polynomial function times $2^{p/2}$ which is asymptotically better than the heuristic $2^{2p/3}$ of the L-algorithm but remains too costly to be useful for big values of p (e.g., $p = 113$ in the case of quad precision).

And yet, in the $M \approx 2^{p^2}$ the complexity boils down to polynomial. For practical applications, such a value of M would be far too big, but this gives us some hope of being able to get useful results for values of M large (significantly larger than 2^p), yet not too large. This is what we will call the "degraded" version of the problem: we will not get hardest-to-round cases, but we will have an information of the form "if the approximation to function f is with error less than ϵ—where ϵ is smaller than actually necessary yet still reasonable for an efficient implementation to be built—then rounding the approximation is equivalent to rounding f". We wish to investigate that possibility, which for the moment is just a theoretical consideration.

Fine algorithmic details As far as the computational step is concerned, most of the effort of optimization should concern the LLL step. The lattices on which LLL is applied have a very strong and specific shape; how to speed

up LLL in that case should (and will) definitely be investigated. However the best hope at that point seems the fact that, in two consecutive subdomains, the polynomial approximations will be very close, and hence the lattice to reduce, which depends continuously on the polynomial approximation.

Hence, the first steps of LLL will be likely to be the same for a large set of intervals; this can be exploited by storing a transformation matrix corresponding to those first steps. We should find a way to use that property, to avoid re-doing the same calculations many times; this requires a fine understanding of the "continuity of LLL", which is a difficult theoretical problem but which we will mostly investigate from an experimental point of view.

3.1.4 Implementation

The two classes of algorithms considered here (the L-algorithm; and the SLZ algorithms and its generalizations) are highly computationally intensive. To be usable, implementations will have to overcome complexity and performance issues. The main contemplated strategy is parallelization since these algorithms are intrinsically massively parallel as all subdomains can be processed independently. This has already allowed us to compute hardest-to-round cases for some functions in double-precision, although the cumulated CPU-time for each function is several years.

Concerning the L-algorithm (Task 1), its great simplicity (or, merely, the great simplicity of the part constituted by Algorithm 3.1.1—in a given subdomain, find the points of a grid nearest a straight line through a variant of the Euclidean GCD algorithm) might make possible a very low-level implementation (FPGA, GPU).

Concerning the SLZ algorithm, two parallel versions will be targeted in practice (Task 7). The first one will have to be portable on supports largely available, focusing mainly on the TMD for double precision. This version will be released as open source software to the community. The second parallel version will have to be specialized for high end supercomputers in order to deal with the largest formats.

As the world of parallelism is quickly evolving, and considering the peculiarities our applications (at first glance, massive parallelism with almost no communications, but other models may emerge from a closer scrutiny), we wish to first evaluate several options. Solutions may differ (and/or combine) depending on the type of software:

- programs that will be widely distributed and hence need to be as portable as possible;
- high-throughput software for computing hardest-to-round cases or certificates as quickly as possible in a controlled, and possibly specific, environment.

After some experiments, we will focus on those which will prove to the be most suited from, but not limited to the following list:

- a local cluster of GPU processors;
- a local cluster of more conventional processors;
- GRID computing (e.g. EGEE);
- volunteer or cloud computing, possibly through middleware like BOINC;
- specific massively parallel architectures (e.g. IBM BlueGene/P), hybrid parallel computers (e.g., the new BULL CPU-GPU Titane computer), or other supercomputer architectures hosted by HPC national centers (e.g. IDRIS Institute or CINES) using CPU time obtained through the new GENCI agency (http://www.genci.fr/)..

Special attention will be paid to stabilize enough the tools that will be widely distributed and to build a production environment, for the high performance version, that will provide a workable platform, possibly opened as a service to other stakeholders.

3.1.5 Formal proof

Following the pioneering works of John Harrison and David Russinoff, formal methods have proved to be very successful in verifying floating-point arithmetic. Typically, they have been used to prove the correctness of software (or hardware) implementation of some specific function (square root, exponential, etc.). In this project, we are aiming at something completely new and much more ambitious: we are dealing with very complex and very long computations for which we want to apply formal methods in order to increase the confidence in their results. We do not claim that at this end of this project everything will be proved formally but we strongly believe that it is crucial that correctness issues and how they could be proved formally should be taken into account at the very earliest stages of development. This is what we plan to do in this project. We will follow a very pragmatic approach trying to adapt verification techniques to the specificities of each algorithm:

- The L-algorithm divides the work on very small intervals. So, even if we were capable of generating a very small certificate for each sub-interval, the overall certificate would be far too big to be manageable. For this reason, we are going to apply standard technique of program verification using tools like Frama-C. The organisation of the L-algorithm being rather complex, it is rather unlikely that in this project we could cover all the aspects formally. For this reason, we first plan to isolate some critical parts of the algorithm and then prove them formally. One of this part is clearly the polynomial approximations that are intensively performed by the algorithms. Polynomial approximations is a hot topic in the theorem proving community (see for example [8, 12, 25]). In this particular case, it is a two-step approximation (the function is approximated by a polynomial of rather large degree, common to several sub-intervals, then in each of these sub-intervals, that first approximation is itself quickly approximated by a polynomial of degree 1).

- As described in Section 3.1.3, the SLZ-algorithm works on larger intervals so a verification of the result based on certificates seems more appropriate. This has the clear benefit of avoiding to have to deal with LLL formally. Still, this approach has to be validated, we need to find "good" certificates that are of reasonable size and can be easily checked. The goal is then to get a fully verified checker for these certificates in a similar way to what was done for primality certificates in [44].

3.2 Project management

The project will have very different aspects: algorithms/mathematics, formal proof, and massive computing. It is essential to have frequent communication and exchange of information between the different groups (and to have overlap between them). We do not just want theorems, published algorithms, and speed-up graphs. In order to have a strong influence on the design of mathematical function libraries, we do actually want hardest-to-round cases for the most common functions and formats, proofs and/or certificates, reasonably portable, open-source, and programs available for function developers. This requires strong coordination between the various tasks.

First, we need to plan frequent meetings:

- two plenary meetings per year (TaMaDi workshops);
- for each task: meetings every other month (possibly some through videoconference);
- very frequent individual mutual visits (one or two-day stayings).

The TaMaDi workshops will be "open meetings" (people not belonging to the project may be invited to share their experience). One of the workshops of the second year (mid-project workshop) will be a mini-conference.

The coordinator will check every month with the various participants that everything is fine. He will send to the ANR a full report every year.

3.3 Detailed description of the work organized by tasks

3.3.1 Task 1: L-algorithm

Coordinator: Arénaire
Participants: Arénaire (16), Pequan (6)

The current implementation that allowed us to publish hardest-to-round points [41] consists of a set of Perl scripts which generate C code, whose core is the L-algorithm. This implementation is mainly targeted to 32-bit computers and all the proofs are pen-and-paper proofs. Maple with the intpakX interval arithmetic package is used to generate the initial coefficients of the polynomials.

More or less a full rewrite is necessary to get much more efficient implementations on various targets (64-bit conventional CPU, FPGAs, GPUs). This can be divided into several subtasks:

- There are several problems with Maple/intpakX. First intpakX is based on accuracy assumptions that have never been documented (and even less proved). Second, Maple is not a free software, so that it is not available everywhere. So, a first subtask would be to replace Maple/intpakX by free, more reliable tools, such as MPFR, MPFI or Boost, or Sollya (a tool from Arénaire).
- The L-algorithm can be implemented in many ways, and a full analysis of the current implementation (time taken by each part, by the branches, etc.) would be useful to improve it.
- Among the various possible improvements (some of them may depend on the target), we may mention:
  - It is possible to prove that some multiple-precision variables have leading zeros on the whole domain; some operations could be avoided if these zeros were detected by the code generator.
  - The alignment of the multiple-precision values in memory may have an influence on the timings.
  - It might be possible to share the first few iterations of the L-algorithm between consecutive intervals (as they generally behave in a similar way) without introducing too much overhead.
  - The length of the intervals on which the L-algorithm is run is fixed. It might be possible to generate code that computes an error bound at each iteration in order to dynamically increase the length of the intervals. As the L-algorithm has a sublinear complexity (an additional iteration roughly corresponds to multiplying the length of the interval by some factor), the gain may be lower than the overhead due to the error bound computation.
- Port the L-algorithm to FPGAs, based on the analysis and improvements done in the above subtasks, and even further improvements: better choice of the internal precision, which no longer needs to be a multiple of 32 or 64, on-chip generation of the degree-1 approximations for low processor-FPGA communications, and use of the parallelism to do some detections that could be more costly on a conventional CPU (shared first few iterations, better choice between doing a division or several subtractions, etc.). The target is a speedup of more than 50 using currently available FPGA co-processors.

3.3.2 Task 2: SLZ algorithm.

Coordinator: Arénaire
Participants: Arénaire (20), Pequan (3)

Our knowledge of the practical behaviour of the SLZ algorithm is still limited. Task 2 will tend to improve this knowledge and then to use it to improve the algorithm itself. Making SLZ really practical requires improving it by a not so large factor (say 10), which seems tractable after such a study.

First, we want to study *fine algorithmic improvements*: use similar LLL steps on neighbour intervals, try to use the structure of the lattices (known beforehand) in order to speed-up the LLL reduction; We also need to study precision and exponents required for inputs of the floating-point variant of LLL, and devise a quick precomputation

step in order to be able to switch to doubles afterwards (required for speedup but also if we want to use GPU for the LLL step);

Second, we need to *understand the SLZ variant* which only produces an upper bound for the size of the hardest-to-round cases, and define which parameters are most likely to produce meaningful, computable and provable results. This variant also needs to be implemented, and its fine algorithmic details again studied.

3.3.3 Task 3: Generating a certificate for the SLZ algorithm

Coordinator: Arénaire
Participants: Arénaire (16), Pequan (3)
SLZ computations, especially those based on the SLZ variant, have the drawback that they imply running a huge piece of complicated code during a long time, and getting mostly a yes/no answer. Some kind of outside validation is thus required. Proving the SLZ algorithm and code implementing it seems an overkill and outside of reach anyhow.

However, since the LLL part is the most time consuming part (more than 90% already with very small parameters; this should increase since we plan to use larger parameters and the LLL cost is asymptotically dominant), we can treat the LLL calls as an oracle, and log its results in a so-called "certificate", the verification of which no longer implies LLL calls.

According to the degree of confidence expected, this certificate can then be checked by a piece of code developed independently from the actual SLZ implementation, or by piece of code extracted from some Coq proof (see Task 5).

The parameters of SLZ need to be rethought with this in mind. Producing an "easy-to-check" certificate implies using the smallest possible number of intervals, which in turns suggests using (largely) sub-optimal parameters for the computation part. It also seems possible to produce several kinds of certificates, according to whether we want to put them on the web (lightweight highly desirable) or store them on a hard drive (easier).

3.3.4 Task 4: Developing formal mathematical background

Coordinator: Marelle
Participants: Arénaire(19), Marelle (16)
The main tool we are going to use for our verification is the Coq proof assistant. Before being able to get to the algorithms, we first need to do some preliminary work in order to get a sufficient amount of mathematical knowledge inside the prover. One important bit is the one about polynomial approximations that are present in both algorithms in different flavour. Another part is the manipulations on polynomials done in the SLZ algorithm that require some basic properties of bivariate polynomials and Hensel's Lemma.

3.3.5 Task 5: Dealing formally with the algorithms

Coordinator: Marelle
Participants: Arénaire(23), Marelle (20)
In this task, we directly attack the algorithms. As explained in Section 3.1.5, we adopt a different strategy for the two algorithms:

L-algorithm Critical parts of the code implementing L-algorithm will be formally proved in order to increase the trust level in the results of the corresponding computations.

SLZ-algorithm Within the logic of Coq, there is a small programming language that provides a way to compute safely. This language is a natural candidate for writing the checker for the certificates generated by the SLZ-algorithm. We will then get for free a verified checker. Unfortunately, this language is purely functional so has an execution time that is an order of magnitude slower than usual programming languages. A key aspect will then

be to evaluate the actual computing power needed to check these certificates. If what is provided by Coq reveals to be insufficient, a mixed approach based on oracles, or a parameterized checker that will be later instantiated by efficient data structures after extraction is taken into consideration.

3.3.6 Task 6: sequential reference implementation of SLZ

Coordinator: Arénaire
Participants: Arénaire (15), Pequan (18)
A complete sequential reference implementation of SLZ is to be written. As of today, only a moderately optimized version of the basic SLZ algorithm is available to us, and is not functional for all parameters.

Such an implementation is required

- to test Stehlé's modified version and estimate what kind of lower bounds can be hoped for in higher precisions;
- as a testbed for trying new ideas, especially fine algorithmic details or producing certificates, and as a reference implementation in order to check the validity of more aggressively optimized versions (see Task 7).

The implementation will rest as much as possible on existing libraries such as FLINT, GMP or MPIR and libfpLLL.

3.3.7 Task 7: Parallel portable and specialized versions of SLZ

Coordinators: Pequan (Pierre Fortin, Stef Graillat), Arénaire (Serge Torres).
Participants: Arénaire(15), Pequan (64.8)
Once the sequential reference implementation of SLZ has been achieved, and once its optimal parameters have been determined, we will turn to its massive parallelization.

As stated in Section 3.1.4, the use of the SLZ algorithm for solving the table maker's dilemma is embarrassingly parallel: the main source of parallelism is thus the huge number of subdomains that can be processed independently. Since the LLL reduction represents the most time consuming part of the SLZ algorithm, we will focus here on the LLL algorithm. Nowadays, parallel architectures are evolving very fast. That is why we will stick here to two general types of parallel architectures: conventional multicore processors and hybrid parallel architectures. Depending on the features of the sequential SLZ reference implementation (optimal parameters and heuristics, memory requirements) and on the underlying hardware, different problems will have to be faced.

On conventional multicore processors, a possible strategy consists in running an instance of SLZ (on one subdomain) per core. This approach seems a priori to be more promising than trying to parallelize the SLZ algorithm on several cores. Indeed, some variants of LLL, using the Schnorr-Euchner LLL algorithm, have already been parallelized on multicore processors [2]. The literature indicates that interesting parallel efficiencies require lattice dimensions greater than 200, which is greater than the dimension of the lattice we will deal with. Nevertheless, the presence of vector instructions (like SSE on Intel processors) makes it possible to study SIMD implementations of LLL (with a small vectorization degree, like 2 or 4). This work on conventional multicore processors could then be extended to massively parallel supercomputers such as the IBM BlueGene/P, where a two-way double floating point unit is also available.

On hybrid parallel architectures like GPUs, the vectorization issue will be much more critical. Indeed, GPUs rely on the SIMD execution model heavily. The larger vectorization degree (current NVIDIA GPUs run threads simultaneously by warps of 32) may imply strong algorithmic modifications of our reference LLL reduction. The GPU will then have to be filled with multiple concurrent LLL executions since, as for multicore, the targeted lattice dimensions are much too low to offer enough computation power to the whole GPU. On other hybrid parallel architectures, like the expected successors[5] to the Cell Broadband Engine or to the Intel Larrabee prototype, fine-

[5] See for example the Intel Single-chip Cloud Computer:
`http://techresearch.intel.com/articles/Tera-Scale/1826.htm`.

grained parallelism may be required: the "one LLL instance per core" strategy might need to be reconsidered here.

While current GPU (as well as the OpenCL standard) now offer IEEE-754 double precision, and while forthcoming GPU like the NVIDIA Fermi will offer important double precision computation power, another issue with such new hybrid architectures is the general lack of a multiprecision library. The sequential reference implementation of SLZ will enable us to determine whether or not multiprecision is required in our LLL implementation in the TMD context. If multiprecision is required, then several options may be studied:

- introduce a preprocessing step that makes possible an LLL reduction in double precision format;
- use the Rump compensated algorithms[6] to restrict ourselves to double precision operations;
- perform the multiprecision operations on the CPU while the GPU is kept busy with other LLL reductions;
- implement the required multiprecision operations on the GPU (in CUDA or OpenCL).

Besides, as mentioned in Section 3.1.3, LLL reductions between two consecutive subdomains contain redundant computations. An appropriate distribution of the subdomains among the multiple processors (bloc distribution for example) will thus be necessary to exploit such a property.

In the end, as mentioned in Section 3.1.4, two parallel versions of SLZ are targeted in practice, leading to two subtasks of Task 7. The first version will be generic (i.e. portable on supports largely available), released as open source software to the community, and focusing mainly on the TMD for double precision. Thanks to parallelization standards (MPI, OpenMP, OpenCL, etc.), this first version will run on conventional multicore processors and possibly on GPUs from NVIDIA and AMD, since these are more and more widely available.

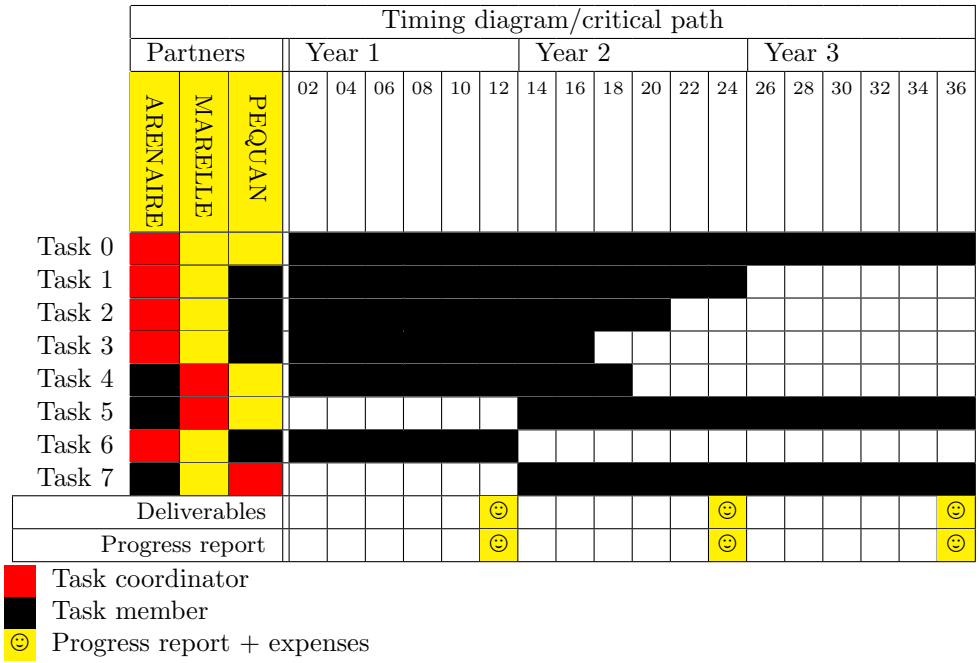
The second parallel version will be specialized to tackle specific challenges (large precisions). This second version will run on national high end supercomputers as provided by the GENCI agency (`http://www.genci.fr/`). These supercomputers include classical parallel computers (clusters of SMP nodes), specific parallel computers (IBM BlueGene/P), as well as hybrid parallel computers (like the new BULL CPU-GPU Titane computer). This specialized version may also include specific algorithms, heuristics and implementations to target particular challenges. Along with this second version, another direction to tackle challenges may be the use of distributed computing which offer huge computational power for such embarrassingly parallel computations, as shown by the SETI@home project. The standalone sequential reference implementation of SLZ may here be plugged into software platforms: for example BOINC (Berkeley Open Infrastructure for Network Computing) for volunteer distributed computing, or EGEE (Enabling Grids for E-sciencE) for grid computing.

This task 7 requires a lot of work: the PhD student requested in Section 6.2.2 will therefore mainly work on this task, as well as partially on Task 6.

[6] Compensated algorithms compute or approximate rounding errors to correct the computed result. This generally increases the accuracy of the result. (S.M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation part I: Faithful rounding. SIAM J. Sci. Comput., 31(1):189-224, 2008).

3.4 Planning of the tasks, deliverables and milestones

3.4.1 General planning



3.4.2 Deliverables

Assuming T is the starting date of the project:

- Task 0 (coordination): scientific and financial reports at times T+12, T+24, T+36 by project coordinator. At time T+36, tables of hardest-to-round for the usual functions and formats are posted on the web site of the project.
- Task 1: at time T+12 a progress report (included in the T+12 report of the coordinator), at time T+24 programs put on the web page of the project as well as a final report (included in the T+24 report of the coordinator).
- Task 2: at time T+12 a progress report (included in the T+12 report of the coordinator), at time T+24 a final report (included in the T+24 report of the coordinator).
- Task 3: at time T+12 a progress report (included in the T+12 report of the coordinator), at time T+24 a final report (included in the T+24 report of the coordinator).
- Task 4: at time T+12 a progress report (included in the T+12 report of the coordinator), at time T+24 a final report (included in the T+24 report of the coordinator) and Coq proofs posted on the web page of the project.
- Task 5: at time T+24 a progress report (included in the T+24 report of the coordinator), at time T+36 a final report (included in the T+36 report of the coordinator) and Coq proofs posted on the web page of the project.
- Task 6: at time T+12 a final report (included in the T+12 report of the coordinator).
- Task 7: at time T+24 a progress report (included in the T+24 report of the coordinator), at time T+36 a final report (included in the T+36 report of the coordinator) and programs for finding hardest-to-round cases posted on the web site of the project.

4 Stratégie de valorisation des résultats et mode de protection et d'exploitation des résultats / Data management, data sharing, intellectual property and results exploitation

We wish our results (hardest-to-round cases, certificates, programs and proofs) to be in the public domain. More precisely:

- the hardest-to-round cases (and, when such cases are out of reach, the bounds on the accuracy required to ensure correct rounding) and the formal proofs will be put on the web site of the project and on an open archive (to guarantee long-term reachability);
- the programs for finding hardest-to-round cases will be open-source, and put on the web site of the project;
- depending on their sizes, the certificates will either be put on the web site of the project, or sent on demand.

We will prepare one or two "popular science" publications explaining the interest of that research, and organize courses explaining how to build function programs from the obtained hardest-to-round cases.

Of course we will also present our results in a more conventional way (scientific journals, proceedings of international conferences, and seminars).

5 Organisation du partenariat / Consortium organization and description

5.1 Relevance and complementarity of the partners within the consortium

The project requires a rather wide domain of expertise: good knowledge of the elementary function algorithms and expertise in lattice-reduction algorithms, massively-parallel computing, formal proof, and floating-point arithmetic. This led us to the choice of teams presented here: Arénaire brings expertise in floating-point arithmetic[7], elementary function algorithms[8] and lattice reduction. Pequan brings expertise in floating-point arithmetic[9], GPU computing and massively parallel computing, and Marelle brings expertise in formal proof, and especially unique expertise in formal proof for floating-point arithmetic algorithms.

5.1.1 Partner 1: The Arénaire project-team

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

Our general goal is to improve arithmetic operators. Among various hardware and software constraints, we focus on reliability, accuracy, speed and energy efficiency of the arithmetic computations. The project is working in four main directions:

Hardware arithmetic operators: number systems, basic operations, algebraic (division, square root, power) and elementary functions (sine, cosine, exponential, etc.), dedicated arithmetic operators for cryptography, digital signal processing, and image processing;

Software arithmetic operators: fixed and floating-point arithmetic, correct rounding, fused-mac, range reduction, multiprecision, algebraic, elementary and special functions, standardization, function approximation, and analysis of algorithms;

[7]For instance, the *Handbook of Floating-Point Arithmetic*, `http://www.springer.com/birkhauser/mathematics/book/978-0-8176-4704-9`, was written by Arénaire members.

[8]For instance, the book *Elementary functions, algorithms and implementation*, `http://www.springer.com/birkhauser/computer+science/book/978-0-8176-4372-0`, was written by an Arénaire member.

[9]The CADNA software, `http://www-pequan.lip6.fr/cadna/`, was developed by Pequan members.

Models and properties of computation, validation and proof: correct rounding, interval arithmetic, use of formal proof, specification of the arithmetic, program and algorithm analysis error bounds, Taylor models, and bounding of results;

Use and applications of arithmetic operators: approximate / interval / exact computations, specific arithmetic library development and test, numerical analysis for intervals, automatic precision choice computer algebra, binary versus algebraic complexity of algorithms, exact linear algebra, and polynomial matrices.

As we are writing this application, there are 11 permanent members in Arénaire (1 PR, 2 DR CNRS, 2 MCF, 3 CR INRIA, 1 CR CNRS, 1 IR ENS, 1 "délégation" INRIA), 10 PhD students, 2 postdocs, 2 engineers. They are not all part of this project (only those listed in Table 1).

5.1.2 Partner 2: The PEQUAN-LIP6 team

The PEQUAN team (PErformance and QUality of Algorithms for Numerical applications) belongs to the department of Scientific Computing of the LIP6 UMR 7606, laboratory at University Pierre and Marie Curie (Paris VI).

The researches of PEQUAN concern different aspects of numerical computation. The main actions can be described in three topics:

the control of accuracy and debugging for numerical applications,

the implementation on different devices (homogeneous or heterogeneous parallel architectures), and

floating point or finite precision arithmetics for symbolic algorithms.

For the first point, PEQUAN had developed a very well-known library named CADNA and a toolbox SOFA, using stochastic approaches. For the second point, in parallel computation, they have studied numerical applications on GRID, and now they focus their work on multicore architectures like GPU and Cell. They also deal with compute-intensive applications through collaborations with some industrials. The last point is oriented towards numeric-symbolic algorithms, the aim being to use the performance of floating-point units in symbolic algorithms.

PEQUAN is composed of 13 permanent researchers (3 professors, 2 emeriti, 8 associate professors MCF) and 5 PhD students. They are not all part of this project (only those listed in Table 2).

5.1.3 Partner 3: The Marelle project-team

The Marelle project-team is situated at INRIA Sophia-Antipolis. It is composed of 5 permanent researchers, 8 PhD students and 2 engineers. They are not all part of this project (only those listed in Table 3).

Its main interest is in building theorem proving tools in order to produce highly dependable software. In particular, one activity is the development of formal mathematical libraries that can be used by mathematicians but also for verifying software dealing with numerical computation. Currently, the mathematical libraries that are concerned are polynomials, algebra, group theory, floating point numbers, real numbers, big integers, probabilities and geometrical objects. In the long run, the goal is to be able to manipulate formally any function that may be of use in embedded software for automatics or robotics (in what is called hybrid systems, systems that contain both software and physical components) and in cryptographical systems. The Marelle team will mainly contribute to this project by its expertise in formal methods and theorem proving.

5.2 Qualification of the project coordinator

Jean-Michel Muller is "Directeur de Recherches de 1re classe" (senior researcher) at CNRS, posted at Laboratoire LIP, Équipe-Projet Arénaire.

- Recent evidence of international recognition: Jean-Michel Muller is a member of the Steering Committee of the IEEE Symposia on Computer Arithmetic, ARITH (the major conference in Computer Arithmetic), and of the Real Numbers and Computers (RNC) series of conferences. He was co-program chair of the 2007 edition of ARITH. He was associate editor of IEEE Transactions on Computers from 1996 to 2000, co-guest editor of a "special section on computer arithmetic" of the same journal (Feb. 2009), co-guest editor of two issues of Theoretical Computer Science (May 2002 and Jan. 2003), and he was one of the four co-authors of the chapter "Digital arithmetic" of the Encyclopedia of Computer Science and Engineering (Wiley, January 2009).

- Scientific production **since January 2005:** 2nd Edition of *Elementary functions, algorithms and implementation* (Birkhaüser Boston, 2006, only author) coordination (and writing of approximately one fourth) of the *Handbook of Floating-Point Arithmetic* (Birkhaüser Boston, Dec. 2009), 14 papers in international journals (IEEE Transactions on Computers, Theoretical Computer Science, ACM Transactions on Mathematical Software, Electronic Letters, Theoretical Informatics and Applications, Journal of VLSI Signal Processing, and International Journal of Electronics), 18 papers in proceedings of international conferences.

- Management/coordination of projects: founder (1998) then head (1998-2004) of the Arénaire CNRS/INRIA/ENS Lyon team; head of the LIP laboratory (91 people in Sept. 2004) from September 2001 to June 2006; "chargé de mission", ST2I department of CNRS, from April 2006 to September 2009.

5.3 Contribution and qualification of each project participant

Arénaire	Nom	Prénom	Emploi actuel	Discipline	Personne · mois	Rôle / responsabilité dans le projet
Coordinateur / responsable	MULLER	Jean-Michel	DR1 CNRS	Informatique	18	project coordinator (10 months); Tasks 2 (4 months) and 6 (4 months)
Autres membres	BRISEBARRE	Nicolas	CR1 CNRS	Informatique	8	Tasks 2 (3 months), 3 (2 months), 6 (3 months)
	DE DINECHIN	Florent	MCF ENS Lyon	Informatique	4	Task 1
	HANROT	Guillaume	Prof. ENS Lyon	Informatique	12	Tasks 2 (6 months), 3 (3 months), 6 (3 months)
	LEFÈVRE	Vincent	CR1 INRIA	Informatique	18	Tasks 1 (12 months) and 5 (6 months)
	MARTIN-DOREL	Érik	Contrat doctoral ENS Lyon (sans financement ANR demandé)	Informatique	30	Tasks 3 (8 months), 4 (11 months) and 5 (11 months)
	MAYERO	Micaela	MCF Paris 13 en délégation INRIA (sans financement ANR demandé)	Informatique	10.8	Tasks 4 (5 months) and 5 (6 months)
	NOVOCIN	Andrew	Postdoc INRIA	Informatique	9	Tasks 2 (2 months), 4 (3 months), 7 (4 months)
	STEHLE	Damien	CR2 CNRS	Informatique	4	Task 2 (2 months), 6 (2 months)
	TORRES	Serge	Ingénieur de recherches ENS Lyon	Informatique	8	Task 7
	XX	YY	Post Doc	Informatique	12	Tasks 2 (3 months), 3 (3 months), 6 (3 months), 7 (3 months)

Table 1: Members of the Arénaire (Partner 1) part of the project. Here, the PostDoc named "XX YY" would be funded by the ANR project.

PEQUAN	Nom	Prénom	Emploi actuel	Discipline	Personne · mois	Rôle / responsabilité dans le projet
Coordinateur / responsable	BAJARD	Jean-Claude	Professeur UPMC	Informatique	18	Tasks 1 (6 months), 6 (6 months), 7 (6 months)
Autres membres	FORTIN	Pierre	MCF UPMC	Informatique	14.4	Tasks 6 (3 months) and 7 (11.4 months)
	GRAILLAT	Stef	MCF UPMC	Informatique	14.4	Tasks 2 (3 months), 3 (3 months) and 7 (8.4 months)
	HILAIRE	Thibault	MCF UPMC	Informatique	12	Tasks 6 (6 months) and 7 (6 months)
	XX	YY	Doctorant UPMC	Informatique	36	Tasks 6 (12 months) and 7 (24 months)

Table 2: Members of the Pequan (Partner 2) part of the project. Here, the PhD student named "XX YY" would be funded by the ANR project.

Marelle	Nom	Prénom	Emploi actuel	Discipline	Personne · mois	Rôle / responsabilité dans le projet
Coordinateur / responsable	THÉRY	Laurent	CR1 INRIA	Informatique	12	Tasks 4 (8 months), 5 (4 months)
Autres membres	RIDEAU	Laurence	CR1 INRIA	Informatique	12	Tasks 4 (8 months), 5 (4 months)
	XX	YY	Post Doc	Informatique	12	Task 5

Table 3: Members of the Marelle (Partner 3) part of the project. Here, the PostDoc named "XX YY" would be funded by the ANR project.

6 Justification scientifique des moyens demandés / Scientific justification of requested budget

6.1 Partner 1: Arénaire

6.1.1 Equipment

A high-end x86 server (8 sockets $\times$ 6 cores = 48 cores) will be one of the computing platforms needed for the development, debugging, and small to medium scaling testing of the LLL algorithm implementations. This "classical" architecture and the rather important number of computing units will allow us to explore the most traditional solutions (similar to those we already use for the L algorithm) as well as new approaches (at least in the context of this project).

More precisely, it will be possible:

- to test implementations based on well known models (multiple independent processes, shared memory (e.g. openMP) or message passing (e.g. MPI)) over a large number of cores;
- to use vanilla and yet powerful system tools (Linux, hypervisor as Xen) to emulate cluster and Grid environments through virtualization ("cluster or Grid in-a-box");
- to assess the impact of, at least, a limited degree of vectorization through the SIMD instructions of the SEE extensions;
- to leverage on the huge high-performance software base already at hand for some critical aspects (multi-precision, all sorts of specific algorithms).

This "one-box" solution is also attractive from other perspectives:

- limited need of administration manpower that matches the scarcity of available resources;
- shrunk platform footprint: computer room space, power, cooling, network connectivity, etc.

At this point, we must stress that this equipment is not devised to become the production environment for the high-throughput implementation of SLZ. Instead we plan to rely on existing (and future) HPC resource available to the computer science community (HPC centers, clusters, Grid, etc.).

Summary description of the equipment: 8 sockets $\times$ 6 cores - 64 GB memory x86-64 server (e.g. HP DL 785 G6 or Sun Fire X4600 M2).

Cost: 35000 €.

6.1.2 Staff

- 12 months of postdoc: the required person will be necessary for partly handling the work on the SLZ algorithm (he or she will focus on Tasks 2, 3, 6 and 7). A good understanding of algorithms as well as skills in programing will be required;
- 12 months (2×6) of training period of students (for helping to write the various programs, mainly the non-critical parts);

6.1.3 Subcontracting

6.1.4 Missions

- 12 travels (and short, one-or-two-night stayings) per year in France, for participating to the plenary meetings and for mutual visits between members of the project;

- three participations (travel + staying + registration fees) per year to international conferences during years 2 and 3, two only during year 1.

Which gives a total of 36 travels and short stayings in France, and 8 participations (travel + staying + registration fees) to international conferences.

6.1.5 Internal expenses

Three laptops, the first year (including one for the recruited PostDoc). Estimate of the cost: $3 \times 2500 = 7500$€.

Arénaire owns two state-of-the art FPGA co-processor boards that can be used for Task 1: one RDX-11 with Virtex-5 (donated by StoneRidge Technologies, inc), and one Altera Stratix-IV GX development kit (funded by the ANR TCHATER project). Funding is only needed for the maintenance of the associated development tools. What is needed is licenses for FPGA development tools (Altera Quartus, Xilinx ISE and Mentor's Modelsim): 1000 euros/year.

6.1.6 Other expenses

6.2 Partner 2: PEQUAN

6.2.1 Equipment

1 rack server with multiple GPUs (4 for example) and multicore processors (at least, 1 core per GPU). Estimated cost (currently for 4 NVIDIA Tesla C1060 GPUs and 1 Intel Xeon quad-core processor): around 11,000 €. This server will be firstly used as a software development platform, where the first, small but real, production cases could then also be run.

Bigger production runs will afterwards be run on national supercomputers managed by the GENCI agency (`http://www.genci.fr/`) where CPU time will have to be obtain through the DARI (*Demande d'Attribution de Ressources Informatiques*) process (`https://www.edari.fr/`).

6.2.2 Staff

- 36 months of PhD support. This PhD student will participate to Tasks 6 and 7.

  Working on Task 6 is required to achieve a good overview of the sequential SLZ algorithm. This is necessary to be able to parallelize the algorithm later. The PhD student will then work on Task 7 mainly focusing on the specific LLL reduction required for the TMD, on parallel LLL algorithmics as well as on multiprecision issues on hybrid parallel architectures.

  In particular, different strategies will have to be studied for parallelization on conventional multicore processors and on various hybrid parallel architectures (like GPUs). Depending on the parallel version considered, the portable or the specialized one (see Sections 3.1.4 and 3.3.7), different algorithmic and implementation choices might have to be considered. The portable version will indeed require generic parallel algorithms and data layouts enabling efficient use of common CPUs or GPUs, or even both at a time. On the other side, specific heuristics, algorithms and data layouts might be necessary for the specialized version to tackle particular challenges.

  The lack of multiprecision in current hybrid architectures can be annoying because the SLZ algorithm may need extra precision. Different strategies could be studied like introducing a preprocessing step to make the SLZ algorithm work only in double precision, or performing multiprecision operations on the CPU. The PhD student will have to consider all these possibilities and choose the one that is more convenient for our purpose.

  Aside from an important work on algorithmic issues, the student will have to make an important effort on programming and tuning those programs for numerous different architectures.

6.2.3 Subcontracting

6.2.4 Missions

- 12 travels (and short, one-or-two-night stayings) per year in France, for participating to the plenary meetings and for mutual visits between members of the project;
- three participations (travel + staying + registration fees) per year to international conferences during years 2 and 3, two only during year 1.

Which gives a total of 36 travels and short stayings in France, and 8 participations (travel + staying + registration fees) to international conferences.

6.2.5 Internal expenses

Three laptops, the first year (including one for the recruited PhD student). Estimate of the cost: $3 \times 2500 = 7500$€.

6.2.6 Other expenses

6.3 Partner 3: Marelle

6.3.1 Equipment

6.3.2 Staff

- 12 months of postdoc.

The post-doc will contribute to the effort of deriving a verified checker for the SLZ certificates (Task 5).

6.3.3 Subcontracting

6.3.4 Missions

- 10 travels (and short, one-or-two-night stayings) per year in France, for participating to the plenary meetings and for mutual visits between members of the project;
- three participations (travel + staying + registration fees) per year to international conferences during years 2 and 3, two only during year 1.

Which gives a total of 30 travels and short stayings in France, and 8 participations (travel + staying + registration fees) to international conferences.

6.3.5 Internal expenses

Two laptops, the first year (including one for the recruited PostDoc). Estimate of the cost: $2 \times 2500 = 5000$€.

6.3.6 Other expenses

7 Annexes / Appendix

7.1 Références bibliographiques / References

7.1.1 References used in the document

References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers. *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Standard 754–1985, 1985.

[2] W. Backes and S. Wetzel. Parallel Lattice Basis Reduction Using a Multi-threaded Schnorr-Euchner LLL Algorithm. *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, volume 5704 of *LNCS*, pages 960–973, Delft, 2009.

[3] S. Boldo, M. Daumas, and L. Théry. Formal proofs and computations in finite precision arithmetic. In T. Hardin and R. Rioboo, editors, *Proceedings of the 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, 2003.

[4] D. Coppersmith. Finding a small root of a univariate modular equation. In U. M. Maurer, editor, *Proceedings of EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer-Verlag, Berlin, 1996.

[5] D. Coppersmith. Finding small solutions to small degree polynomials. In J. H. Silverman, editor, *Proceedings of Cryptography and Lattices (CaLC)*, volume 2146 of *Lecture Notes in Computer Science*, pages 20–31. Springer-Verlag, Berlin, 2001.

[6] M. Cornea, J. Harrison, and P. T. P. Tang. *Scientific Computing on Itanium®-based Systems*. Intel Press, Hillsboro, OR, 2002.

[7] F. de Dinechin and N. Gast. Towards the post-ultimate libm. Research Report 2004-47, LIP, École normale supérieure de Lyon, 2004. Available at `http://www.ens-lyon.fr/LIP/Pub/Rapports/RR/RR2004/RR2004-47.pdf`.

[8] Marc Daumas, David Lester, and César Muñoz. Verified real number calculations: A library for interval arithmetic. *IEEE Trans. Computers*, 58(2):226–237, 2009.

[9] N. D. Elkies. Rational points near curves and small nonzero $|x^3 - y^2|$ via lattice reduction. In Wieb Bosma, editor, *Proceedings of the 4th Algorithmic Number Theory Symposium (ANTS IV)*, volume 1838 of *Lecture Notes in Computer Science*, pages 33–63. Springer-Verlag, Berlin, 2000.

[10] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[11] G. Hanrot, V. Lefèvre, D. Stehlé, and P. Zimmermann. Worst cases of a periodic function for large arguments. In *ARITH '07: Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 133–140, Washington, DC, 2007. IEEE Computer Society.

[12] J. Harrison. Verifying the accuracy of polynomial approximations in HOL. In *TPHOLs'97*, volume 1275 of *LNCS*, pages 137–152, 1997.

[13] J. Harrison. A machine-checked theory of floating-point arithmetic. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics: 12th International Conference, TPHOLs'99*, volume 1690 of *Lecture Notes in Computer Science*, pages 113–130. Springer-Verlag, Berlin, September 1999.

[14] J. Harrison. Formal verification of floating-point trigonometric functions. In W. A. Hunt and S. D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer-Aided Design, FMCAD 2000*, number 1954 in Lecture Notes in Computer Science, pages 217–233. Springer-Verlag, Berlin, 2000.

[15] J. Harrison. Fast and accurate bessel function computation. In *Proceedings of the 19th IEEE Symposium on Computer Arithmetic (ARITH-19)*. IEEE Computer Society, June 2009.

[16] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, August 2008. available at `http://ieeexplore.ieee.org/servlet/opac?punumber=4610933`.

[17] W. Kahan. Why do we need a floating-point standard? Technical report, Computer Science, UC Berkeley, 1981. Available at `http://www.cs.berkeley.edu/~wkahan/ieee754status/why-ieee.pdf`.

[18] W. Kahan. A logarithm too clever by half. Available at `http://http.cs.berkeley.edu/~wkahan/LOG10HAF.TXT`, 2004.

[19] C. Q. Lauter. *Arrondi Correct de Fonctions Mathématiques*. PhD thesis, École Normale Supérieure de Lyon, Lyon, France, October 2008. In French, available at `http://www.ens-lyon.fr/LIP/web/`.

[20] V. Lefèvre, D. Stehlé, and P. Zimmermann. Worst cases for the exponential function in the ieee 754r decimal64 format. In *Reliable Implementation of Real Number Algorithms: Theory and Practice*, pages 114–126. Springer-Verlag, 2008.

[21] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[22] M. J. Schulte and E. E. Swartzlander. Exact rounding of certain elementary functions. In E. E. Swartzlander, M. J. Irwin, and G. Jullien, editors, *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pages 138–145. IEEE Computer Society Press, Los Alamitos, CA, June 1993.

[23] D. Stehlé. *Algorithmique de la Réduction de Réseaux et Application à la Recherche de Pires Cas pour l'Arrondi de Fonctions Mathématiques*. PhD thesis, Université Henri Poincaré – Nancy 1, France, December 2005.

[24] D. Stehlé. On the randomness of bits generated by sufficiently smooth functions. In F. Hess, S. Pauli, and M. E. Pohst, editors, *Proceedings of the 7th Algorithmic Number Theory Symposium, ANTS VII*, volume 4078 of *Lecture Notes in Computer Science*, pages 257–274. Springer-Verlag, Berlin, 2006.

[25] Roland Zumkeller. Formal global optimisation with taylor models. In *IJCAR*, volume 4130 of *LNCS*, pages 408–422, 2006.

7.1.2 Publications by members of the Arénaire team related to the project

Notice. Some of these publications are written by applicants to this project that were not members of the Arénaire team at the date of publication. For instance, Vincent Lefèvre and Guillaume Hanrot were members of the CACAO team of lab. LORIA when they participated to papers [34, 35, 11, 40, 44]

Books

[26] J.-M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhaüser Boston, MA, 2nd edition, 2006.

[27] J.-M. Muller, N. Briseba rre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhaüser Boston, December 2009.

Articles in international journals

[28] V. Lefèvre, J.-M. Muller, and A. Tisserand. Toward correctly rounded transcendentals. *IEEE Transactions on Computers*, 47(11):1235–1243, November 1998.

[29] N. Brisebarre and J.-M. Muller. Correct rounding of algebraic functions. *Theoretical Informatics and Applications*, 41:71–83, Jan–March 2007.

[30] F. de Dinechin, C. Q. Lauter, and J.-M. Muller. Fast and correctly rounded logarithms in double-precision. *Theoretical Informatics and Applications*, 41:85–102, 2007.

[31] N. Brisebarre and J.-M. Muller. Correctly rounded multiplication by arbitrary precision constants. *IEEE Transactions on Computers*, 57(2):165–174, February 2008.

[32] N. Brisebarre, J.-M. Muller, and A. Tisserand. Computing machine-efficient polynomial approximations. *ACM Transactions on Mathematical Software*, 32(2):236–256, June 2006.

[33] N. Brisebarre, J.-M. Muller, A. Tisserand, and S. Torres. Hardware operators for function evaluation using sparse-coefficient polynomials. *Electronic Letters*, 42(25):1441–1442, December 2006.

[34] D. Defour, G. Hanrot, V. Lefèvre, J.-M. Muller, N. Revol, and P. Zimmermann. Proposal for a standardization of mathematical function implementation in floating-point arithmetic. *Numerical Algorithms*, 37(1–4):367–375, 2004.

[35] D. Stehlé, V. Lefèvre, and P. Zimmermann. Searching worst cases of a one-variable function. *IEEE Transactions on Computers*, 54(3):340–346, March 2005.

[36] C. Q. Lauter and V. Lefèvre. An efficient rounding boundary test for pow(x, y) in double precision. *IEEE Transactions on Computers*, 58(2):197–207, February 2009.

[37] P. Kornerup, C. Q. Lauter, N. Louvet, V. Lefèvre, and J.-M. Muller. Computing correctly rounded integer powers in floating-point arithmetic. *ACM Transactions on Mathematical Software*, January 2010.

Articles in proceedings of international conferences and book chapters

[38] S. Chevillard and C. Q. Lauter. A certified infinite norm for the implementation of elementary functions. In *Seventh International Conference on Quality Software (QSIC 2007)*, pages 153–160. IEEE, 2007.

[39] C. Daramy, D. Defour, F. de Dinechin, and J.-M. Muller. CR-LIBM, a correctly rounded elementary function library. In *SPIE 48th Annual Meeting International Symposium on Optical Science and Technology*, August 2003.

[11] G. Hanrot, V. Lefèvre, D. Stehlé, and P. Zimmermann. Worst cases of a periodic function for large arguments. In *ARITH-18: Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 133–140, Washington, DC, 2007. IEEE Computer Society.

[40] V. Lefèvre. New results on the distance between a segment and $\mathbb{Z}^2$. Application to the exact rounding. In *Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH-17)*, pages 68–75. IEEE Computer Society Press, Los Alamitos, CA, June 2005.

[41] V. Lefèvre and J.-M. Muller. Worst cases for correct rounding of the elementary functions in double precision. In N. Burgess and L. Ciminiera, editors, *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH-16)*, Vail, CO, June 2001.

[42] V. Lefèvre, J.-M. Muller, and A. Tisserand. Towards correctly rounded transcendentals. In *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[20] V. Lefèvre, D. Stehlé, and P. Zimmermann. Worst cases for the exponential function in the ieee 754r decimal64 format. In *Reliable Implementation of Real Number Algorithms: Theory and Practice*, pages 114–126. Springer-Verlag, 2008.

[43] J.-M. Muller and A. Tisserand. Towards exact rounding of the elementary functions. In Alefeld, Frommer, and Lang, editors, *Scientific Computing and Validated Numerics (Proceedings of SCAN'95)*. Akademie Verlag, 1996.

[44] Laurent Théry and Guillaume Hanrot. Primality proving with elliptic curves. In *TPHOLs*, volume 4732 of *LNCS*, pages 319–333, 2007.

Ph.D. dissertations and technical reports

[19] C. Q. Lauter. *Arrondi Correct de Fonctions Mathématiques*. PhD thesis, École Normale Supérieure de Lyon, Lyon, France, October 2008. In French, available at `http://www.ens-lyon.fr/LIP/web/`.

[45] V. Lefèvre. An algorithm that computes a lower bound on the distance between a segment and $\mathbb{Z}^2$. Research report RR97-18, LIP, École Normale Supérieure de Lyon, 1997. Available at `http://www.ens-lyon.fr/LIP/research_reports.us.html`.

[46] V. Lefèvre. *Moyens Arithmétiques Pour un Calcul Fiable*. PhD thesis, École Normale Supérieure de Lyon, Lyon, France, 2000.

7.1.3 Publications of the LIP6 team related to the project

Articles in international journals

[47] Diep Nguyen Hong, Stef Graillat and Jean-Luc Lamotte. Extended precision with a rounding mode toward zero environment. application on the cell processor. *Int. J. Reliability and Safety*, 3(1/2/3):153–173, 2009. special issue on "Reliable Engineering Computing".

[48] Stef Graillat. Accurate floating point product and exponentiation. *IEEE Transactions on Computers*, 58(7):994–1000, 2009.

[49] Stef Graillat. Accurate simple zeros of polynomials in floating point arithmetic. *Comput. Math. Appl.*, 56(4):1114–1120, 2008.

Articles in proceedings of international conferences and book chapters

[50] Jean-Marie Chesneaux, Stef Graillat and Fabienne Jézéquel. *Encyclopedia of Computer Science and Engineering*, volume 4, chapter Rounding Errors, pages 2480–2494. Wiley, 2009.

[51] O. Coulaud, P. Fortin, and J. Roman. Hybrid MPI-thread parallelization of the Fast Multipole Method. In *6th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 391–398, 2007.

[52] P. Fortin and J.-L. Lamotte. Fast Multipole Method on the Cell Broadband Engine: the Near Field Part. In *International Conference on Parallel Computing (ParCo2009)*, 2009.

[53] J.C. Bajard, D. Michelucci, P. Langlois G. Morin and N. Revol. In *Advanced Signal Processing Algorithms, Architectures, and Implementations XVIII, SPIE International Symposium*, San Diego, USA, 2008.

[54] J.C. Bajard, M.D. Ercegovac, L. Imbert and F. Rico. Floating Point Geometry: toward guaranteed geometric computations with approximate arithmetics In *4th Conference on Real Numbers and Computers,* Schloss Dagstuhl, Germany, 2000.

[55] JC Bajard and L. Imbert Evaluation of Complex elementary functions: new version of BKM In *Advanced Signal Processing Algorithms, Architectures and Implementations, SPIE International Symposium,* Denver, Colorado, USA, 1999.

7.1.4 Publications of the Marelle team related to the project

Articles in proceedings of international conferences and book chapters

[3] S. Boldo, M. Daumas, and L. Théry. Formal proofs and computations in finite precision arithmetic. In T. Hardin and R. Rioboo, editors, *Proceedings of the 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, 2003.

[56] M. Daumas, L. Rideau, and L. Théry. A Generic Library for Floating-Point Numbers and Its Application to exact Computing. In *TPHOLs*, volume 2152 of *LNCS*, 2001.

[57] Yves Bertot, Georges Gonthier, Sidi Ould Biha, Ioana Pasca. Canonical Big Operators. In *TPHOLs*, volume 5170 of *LNCS*, 2008.

[58] François Garillot, Georges Gonthier, Assia Mahboubi, and Laurence Rideau. Packaging Mathematical Structures. In *TPHOLs*, volume 5674 of *LNCS*, 2009.

[44] Laurent Théry and Guillaume Hanrot. Primality proving with elliptic curves. In *TPHOLs*, volume 4732 of *LNCS*, pages 319–333, 2007.

7.2 CV, resume

7.2.1 Jean Claude Bajard

- born March 1, 1957;
- Master Degree, ENS Lyon, 1990;
- PhD degree, ENS Lyon, 1993;
- "Habilitation à diriger des recherches", Université de Provence Marseille, 1998;
- **1979-1990:** Mathematic Teacher in Highschool. **1990-1993:** Assistant, ENS Lyon LIP. **1993-1999:** Associate professor (Maître de conférences) Université de Provence, Marseille. **1999-2009:** Professor, Université de Montpellier 2, LIRMM UMR CNRS 5506. **2009-2027:** Professor, Université Pierre et Marie Curie, Paris, LIP6 UMR CNRS 7606.

Publications: 3 books, 7 articles in international journals, 2 in national journals, 33 articles in proceedings of international conferences.
Various responsibilities: head of Theoretical Computer Science dept. Laboratoire LIRMM (2000–2003), head of the PEQUAN team at LIP6 (2009-...).
Supervisor of 19 PhD students since 1995 (currently two, they will defend in 2010).

7.2.2 Nicolas Brisebarre

- born November 26, 1971;
- "Agrégation" in mathematics, 1995;
- PhD degree, Université Bordeaux 1, 1998;
- Assistant professor at Université Jean Monnet (Saint-Étienne) from 1999 to 2007 then researcher at CNRS at Lab. LIP, ENS Lyon since 2007.

Publications: 10 articles in international journals; 10 articles in proceedings of international conferences.

7.2.3 Florent de Dinechin

- born February 11, 1970;
- Received ENS Lyon 1990;
- PhD degree, Université Rennes, 1997;
- PostDoc Imperial College (london), 1998;
- Assistant professor at ENS Lyon since 1998.

Publications: 15 articles in international journals; 36 articles in proceedings of international conferences.

7.2.4 Pierre Fortin

- born June 11, 1979;
- engineer degree (ENSEIRB, Bordeaux), 2002;
- PhD degree, Université Bordeaux 1, 2006;
- Assistant professor at Université Pierre et Marie Curie, in Lab. LIP6, Paris, since 2007.

Publications: 2 articles in international journals; 3 articles in proceedings of international conferences.
Various responsibilities: Since 2009, in charge of a new fifth year engineering school "minor" in high performance computing on new parallel architectures (see `http://www-pequan.lip6.fr/~fortin/CINAP`).

7.2.5 Stef Graillat

- born April 8, 1978;
- engineer degree (ENSIMAG, Grenoble) and M.Sc. in applied mathematics (UJF, Grenoble), 2001;
- student at École normale supérieure de Cachan, 2001-2003;
- "Agrégation" in mathematics, 2002;
- PhD degree in computer science, Université de Perpignan, 2005;
- Associate professor at Université Pierre et Marie Curie (Paris 6), in LIP6 laboratory, Paris, since 2006.

Publications: 12 articles in international journals; 11 articles in proceedings of international conferences; 1 chapter for encyclopedia.

7.2.6 Guillaume Hanrot

- born January 17, 1973;
- "Agrégation" in mathematics, 1995;
- PhD degree, Université Bordeaux 1, 1997;
- Habilitation à diriger des recherches, Université Henri-Poincaré Nancy 1, 2005.
- Researcher at INRIA Lorraine (CR from 1998 to 2007 then DR from 2007 to 2009) then Professor at ENS Lyon since 2009.

Publications: 15 articles in international journals; 6 articles in proceedings of international conferences; 3 book chapters.
Various responsibilities: Vice-chair, INRIA Evaluation Committee (2008-); *délégué scientifique*, INRIA Nancy-Grand Est (2008-2009); *vice-président du comité des projets*, INRIA Lorraine (2004-2008); head of the CACAO team (2006-2009).

7.2.7 Thibault Hilaire

- born October 27, 1977;
- engineer degree (École Centrale de Nantes) and M.Sc. in Control and Applied Computer Science (University of Nantes), 2002;
- PhD degree in Control and Applied Computer Science (University of Nantes), 2006;
- Associate professor at Université Pierre et Marie Curie (Paris 6), in LIP6 laboratory, Paris, since 2009.

Publications: 4 articles in international journals; 11 articles in proceedings of international conferences.

7.2.8 Vincent Lefèvre

- born May 23, 1973;
- "Agrégation" in mathematics, 1997;
- PhD degree in computer science, École Normale Supérieure de Lyon, 2000;
- researcher at INRIA since 2000.

Publications: 1 book, 8 articles in international journals, 1 article in national journals, 12 articles in proceedings of international conferences; software: MPFR library.

7.2.9 Érik Martin-Dorel

- born March 20, 1986;
- M.Sc. in mathematics and computer science, Université Montpellier 2, 2009;
- PhD student at École Normale Supérieure de Lyon, since 2009.

Publications: 2 articles in proceedings of international conferences.

7.2.10 Micaela Mayero

- born September 27, 1971;
- PhD in computer science, Paris 6 University, 2001;
- PostDoc, Chalmers University - Göteborg, Sweden, 2001–2002;
- ATER, Paris 7 university, 2002–2003;
- Researcher at CEA, 2003-2004;
- Assistant professor at IUT de Villetaneuse, in Lab. LIPN, since 2004.

Publications: 2 articles in international journals, 5 articles in proceedings of international conferences.

7.2.11 Jean-Michel Muller

- born May 8, 1961;
- engineer degree (ENSIMAG, Grenoble), 1983;
- PhD degree, Institut National Polytechnique de Grenoble, 1985;
- "Habilitation à diriger des recherches", Institut National Polytechnique de Grenoble, 1989;
- Researcher at CNRS (CR since 1986, DR2 since 1999, DR1 since 2006), first posted in Lab. TIM3, Grenoble, then (since 1989) in Lab. LIP, Lyon.

Publications: 5 books, including *Elementary functions, Algorithms and Implementation* (only author, 2nd ed., Birkhaüser Boston, 2006) and *Handbook of Floating-Point Arithmetic* (coordinator, Birkhaüser Boston, 2009); 38 articles in international journals; 83 articles in proceedings on international conferences; guest editor of 8 special issues of journals.
Various responsibilities: *chargé de mission*, ST2I department of CNRS (2006–2009), head of Laboratoire LIP (2001–2006), head of the Arénaire team (1998-2004).
Supervisor of 15 PhD students since 1989.

7.2.12 Andrew Novocin

- born June 12, 1983;
- PhD degree (FSU, Tallahassee), 2008;
- Post-Doctorate with INRIA and ENS de Lyon.

Publications: 3 articles in proceedings on international conferences.

7.2.13 Laurence Rideau

- born December 21, 1957;
- École Normale Supérieure Fontenay-aux-Roses, 1977-1981.
- PhD thesis (Paris Sud, Orsay), 1983;
- Researcher at INRIA Sophia-Antipolis since 1984.

Publications: 1 article in international journals; 17 articles in proceedings of international conferences.

7.2.14 Laurent Théry

- born May 17, 1966;
- PhD thesis (Denis Diderot, Paris), 1994;
- researcher at INRIA Sophia-Antipolis since 1994.

Publications: 4 articles in international journals; 18 articles in proceedings of international conferences.

7.2.15 Serge Torres

- born December 11, 1953;
- engineer degree (CNAM, Paris), 2000;
- research engineer with the ENS de Lyon since 2001.

Publications: 2 articles in international journals; 1 book chapter.

7.3 Involvement of project participants to other grants, contracts, etc.

Arénaire	Nom de la personne participant au projet	personne × mois	Intitulé de l'appel à projets Source de financement Montant attribué	Titre du projet	Nom du coordinateur	Date début & date fin
	G. Hanrot	12	Projet ANR "Chic" €	Courbes hyperelliptiques : isogénies et comptage, 380000 €	D. Lubicz	Sept. 2009 - Sept. 2012
	M. Mayero	15	Projet ANR "FOST" 116480 €	Formal prOofs about Scientific compuTations	S. Boldo	Jan. 2009 - Jan. 2012
	M. Mayero	8	Projet ANR "COMPLICE" 299766 €	Implicit Computational Complexity, Concurrency and Extraction	P. Baillot	Jan. 2009 - Jan. 2013
	J.-M. Muller	10	Projet "CIBLE" de la région Rhône-Alpes 103000 € (soit une bourse de thèse plus 12000 €)	Synthèse d'arithmétiques logicielles pour processeurs d'applications multimédia	J.-M. Muller	Sep. 2008 - Sep. 2011

Marelle	Nom de la personne participant au projet	personne × mois	Intitulé de l'appel à projets Source de financement Montant attribué	Titre du projet	Nom du coordinateur	Date début & date fin
	Laurent Théry	12	ANR Blanc	Galapagos	Yves Bertot	Nov. 2007 - Nov. 2010
	Laurent Théry	12	ANR Domaines Émergents	Decert	Thomas Jensens	Jan. 2009 - Jan. 2012

Pequan	Nom de la personne participant au projet	Intitulé de l'appel à projets Source de financement Montant attribué	Titre du projet	Nom du coordinateur	Date début & date fin