Acronym	**MetaLibm**		
Titre du projet	Générateurs de code pour les fonctions mathématiques et les filtres		
Proposal title	Code generators for mathematical functions and filters		
Theme(s)	1 □ 2 ☑		
Type of research	☑ Basic Research □ Industrial Research □ Expérimental Development		
International cooperation	International cooperation with : □ other countries		
Grant requested	622782€	**Project duration**	48 mois
Coordinator partner	Identité du coordinateur : De Dinechin Florent Identification de l'établissement : LIP, ENS Lyon		
Link with a project of the *Investment for the Future* programme	☑ Yes: Labex MILyon, Labex CalSimLab □ No		

SUMMARY OF CHANGES WRT INITIAL SUBMISSION

We have removed Kalray as a partner. However we still want to consider their processor technology, and hopefully provide them with a high-performance libm, in accordance with the initial scientific document. Therefore,

- There is no change to the work plan. Among tasks, deliverables and milestones, only D5.2b was specific to Kalray, and we wish to maintain it.
- We still mention Kalray as a key target in 2.3, p. 10.
- We will buy two Kalray *MPPA developer* workstations, one for INRIA and one for CERN. Justifications for this are given on pages 34 and 35.
- We have removed the funding of missions to selected Kalray customers.

We also mention p. 34 that we hope that this work will attract further research contracts with Kalray.

We have also slightly updated next page as a few participants have changed positions since submission. These changes are within the project and will have no financial nor scientific impact.

People involved in the project:

Organisation	Last name	First name	Current position	Involvement in the project (person.month)	Contribution to the project
INRIA / LIP	de Dinechin [1]	Florent	Professor	36 p.m	project leader, libm generation, FPGA arithmetic
INRIA / LIP	Brunie[2]	Nicolas	PhD student	12 p.m	hardware floating-point, libm for embedded processors
INRIA / LIP	Jeannerod	Claude-Pierre	CR1 Inria	12 p.m	floating-point on embedded processors
INRIA / LIP	Muller	Jean-Michel	DR1 CNRS	12 p.m	floating-point, elementary functions
INRIA / LIP	???	???	Ph.D. student to be recruited	36 p.m	fixed point implementations for processors and FPGAs
UPMC / LIP6	Lauter	Christoph	Maître de conférences	24 p.m	coordinator for LIP6, open-ended function generation
UPMC / LIP6	Hilaire	Thibault	Maître de conférences	24 p.m	advisor for ANR project Ph.D. student (at LIP6), filter generation
UPMC / LIP6	Kupriianova	Olga	Ph.D. student	18 p.m	open-ended function generation toolchain
UPMC / LIP6	Mezzarobba[3]	Marc	CR2 CNRS	12 p.m	computer algebra, implementation of special functions
UPMC / LIP6	???	???	Ph.D. student to be recruited	36 p.m	open-ended function and filter generation toolchain
LIRMM / DALI	Revy	Guillaume	Maître de conférences	24 p.m	coordinator for LIRMM, parallel polynomial evaluation
LIRMM / DALI	Defour	David	Maître de conférences	24 p.m	elementary functions on GPUs
LIRMM / DALI	???	???	Ph.D. student to be recruited	36 p.m	elementary functions on GPUs
CERN	Innocente	Vincenzo	Senior Staff Member	12 p.m	high-performance computing for high-energy physics
CERN	Piparo	Danilo	Junior Staff Member	6 p.m	optimization of ad-hoc functions for physics code, compilation

[1] Florent de Dinechin was an assistant professor with LIP in the initial proposal. He is now a full professor at INSA-Lyon, in the INRIA Socrate team. He remains fully committed to this project in this new position.

[2] Nicolas Brunie is in his last year of PhD under the supervision of F. de Dinechin and R. Ayrignac (Kalray), with a Kalray CIFRE grant. He was counted as Kalray staff in the initial proposal.

[3] Marc Mezzarobba was a LIP Post-Doc in the initial proposal. He is now a permanent researcher at LIP6.

CONTENTS

1. EXECUTIVE SUMMARY OF THE PROPOSAL

This project first addresses the implementation of *elementary functions* such as exponential or trigonometric. The standard mathematical library (`libm`) offers a small set of such functions in a small set of precisions. Performance of `libm` functions is of critical importance, in particular in scientific and financial computing. Therefore, they have always received special attention. For instance, processor manufacturers have teams of engineers who implement an optimized `libm` for each new processor, a time-consuming and error-prone task. A first objective of this project is therefore to **automate libm development**, up to the point where `libm`s generated in a fraction of the time match in performance hand-coded ones for a wide range of targets: general-purpose processors, embedded processors, graphical processing units, and even FPGA and digital circuits.

Applications often require functions not present in the `libm`. In such case, a composition of `libm` functions may be very inefficient, or simply not possible (function defined by a differential equation or by interpolation of a set of data points for instance). Even the `libm` functions are offered in a limited choice of implementations which will most of the time poorly match the actual needs of an application in terms of range, accuracy, performance and resource consumption. The second objective of this project is therefore to **open the set of functions and contexts** available to programmers. This qualitative progress will be enabled by automation: we aim at designing and distributing tools that implement mathematical functions on-demand for a given context. Such implementations can not be tested *a priori*, their accuracy needs to be proven by construction: this will be a pervasive concern in this project.

The process of implementing a mathematical function into code is composed of several steps, typically range reduction, approximation, evaluation. In the signal processing and control communities, a very similar process, based on similar steps, consists in implementing a digital filter out of its mathematical specification. In both cases, the final code or circuit consists of the same primitives (additions, multiplications, and pre-computed constant values). A third objective of this project is therefore to **unify the understanding of function and filter development**, sharing common mathematical, methodological, and implementation tools. In particular this will lead to on-demand generation of filters for the same breadth of targets and with the same guarantees on numerical quality.

To reach these three objectives, the method will consist of

- a systematic attempt to *formalize* the development process *as a whole*, up to the point where it can be described algorithmically;
- motivation by, and validation against, practical case studies from partners of the project and beyond;
- the development of open-source code (and proof) generators.

The proposed consortium (INRIA/AriC, UPMC/LIP6, LIRMM, CERN) gathers, around worldwide leaders in elementary functions, a number of expertises covering filters and control, a wide range of technological targets, and scientific and industrial applications.

2. CONTEXT, POSITION AND OBJECTIVES OF THE PROPOSAL

Computing systems must provide support for basic mathematical functions. This includes

- the basic operations: addition, subtraction, multiplication, division, square root, and various conversions;
- *elementary* functions [67] such as exponential and logarithm, sine, cosine, tangent and their inverses, hyperbolic functions and their inverses;

- and other "special" functions, such as the power function x^y, erf, Γ or the Airy function [1], that appear in various computing contexts.

For the pervasive floating-point formats defined in 1985 by the IEEE 754 standard, some programming languages and operating systems have defined at various levels the set of functions to be implemented. The 2008 revision of the IEEE 754 standard [41] has attempted to merge these into a common standardization[4] of the elementary function library (`libm`).

Mathematical support is provided by a combination of hardware and operating system, the latter offering in software what the hardware does not natively provide. The optimal repartition between hardware and software has evolved with time [71]. For instance, the first 80287 mathematical coprocessor, in 1985, included support for a range of elementary functions (albeit in microcode) in addition to the basic operations. It was later found that software could outperform such microcode, by relying for instance on large tables of precomputed values as memory got cheaper [72, 30]. Furthermore, progress in compiler technology allowed for a deeper integration of elementary functions software in the overall compilation process [60, 13, 61], another case for software-based functions. These days, even the relevance of hardware division and square root is disputed [13]. At the same time, table-based algorithms are being replaced with high-degree polynomials as multicore architecture are increasingly memory-constrained, and GPUs offer a few hardware-accelerated elementary functions.

Indeed, performance of an elementary function library is of paramount importance. For example, profiling the SPICE electronic simulator shows that it spends most of its time in the evaluation of elementary functions [51]. The same holds for large-scale simulation and analysis code run at CERN [42, 14, 3]. Therefore, the processor vendors (Intel, AMD, ARM, nVIDIA) employ large teams of engineers to develop optimized versions of the core elementary functions for each processor they bring to market. This represents considerable amount of work, in particular because each function comes in several variants corresponding to a range of constraints on performance (e.g. optimized for throughput or optimized for latency) or accuracy.

With more limited manpower, the open-source mathematical libraries (most notably in the GNU `glibc`) lag behind in performance, all the more as they have to support a much wider range of processors.

In parallel to this work on elementary functions by the computer science and mathematics communities, the implementation of *digital filters* was studied by two different scientific communities: signal processing and control. Control uses the same linear filters as signal processing, but in a closed-loop context, where the output of the filter controls e.g. a plant, the input coming from the plant and exogenous signals. This different context translates to a different specification: Filters are typically specified as a transfer function in signal processing, and as a state-space description in control.

The very first works on digital filter synthesis appeared in the late Fifties, when filters transitioned from analog to digital. Among the first questions studied was the effect of quantization on filters [6, 74, 46, 58]. In signal processing, roundoff error analysis considered computational errors as numerical noises that perturb the intern signal flow. In control, roundoff analysis and the impact of the quantization were studied on the state-space form, first without any concrete implementation considerations [40, 69, 73, 32]. For a given state-space realization, there exist an

[4] Strictly speaking, the `libm` is specified in the C language standard (currently ISO/IEC 9899:2011) which is compatible with IEEE 754, while other language standards, (C++, Fortran) also maintain `libm` compatibility with C. In addition, language standards also specify elementary functions on complex numbers, but for historical reasons not in the "mathematical library" section. Finally, many operating systems offer functions not present in the `libm`. In this project we understand the term "`libm`" in its widest sense and address all these functions.

infinity of mathematically equivalent realizations (by applying a basis transform). However, all these realizations are no longer equivalent in finite precision arithmetic, which raised the question of the best basis in terms of robustness [50, 75, 31]. These works were then extended to various filter structures (sparse realization [27, 55, 45, 77], δ-operators [64], ρ-operators [56, 28], etc.), leading to different tradeoffs between complexity and resilience. A lot of different measures of the numerical degradations (associated with the application's desired performances) have been built (transfer function sensitivity [73, 31, 44], pole sensitivity [57], for open and closed-loop [43, 39] and MIMO[5] systems [59, 38]).

2.1. OBJECTIVES, ORIGINALITY AND NOVELTY OF THE PROJECT

The main objective of this project is to reach a deeper understanding of the implementation process of functions and filters. Implementation here consists in transforming a mathematical object into a program or circuit (a computer science artifact) that approximates it in a satisfactory way. In addition to the usual *performance metrics* (speed, memory cost, efficiency with respect to the available hardware, power consumption), this process defines metrics of *accuracy* to capture the quality of the approximation performed: typically, the relative difference between the function and its implementation, or a signal-to-noise ratio for the filter.

All these factors are well managed by experts in a more or less intuitive way, and textbooks on elementary functions [60, 67] or filters [31, 44, 63, 76] provide implementation recipes for this purpose, sometimes general, sometimes function-specific. An expert is thus able to obtain an implementation of a function or filter matching a predefined set of requirements. Given enough time, the expert will even be able to assert in a convincing way that an implementation is close to optimal.

In the realm of elementary function development, this manual approach has several limitations that the present project intends to lift:

- The number of functions of interest is huge. There is an even larger number of implementation contexts: required precision, target hardware, performance/cost constraints. This defines an enormous set of implementations of interest, far too large to manually implement them all. In addition, hardware evolves quickly, leading to implementation obsolescence. The current solutions consist in living with a restricted set of functions/precisions (the `libm` approach), and living with implementations that are widely sub-optimal for most contexts. For instance, the code of some functions in open-source `libm`s is now twenty years old. A processor maker like Intel maintains a team of engineers who reimplement the same `libm` functions for each new processor produced by the company. They achieve 2x-5x speedups compared to the open-source implementation, but at a huge manpower-to-market cost.

- As long as each function must be implemented manually, the set of functions available to programmers will be limited. Programmers, however, do need application-specific functions not present in the `libm`. In such cases they have to tinker an implementation out of existing functions and numerical methods, and this implementation is likely to be much less accurate and much less efficient than that of an expert. We must insist again on the accuracy issue: this is not only a problem of performance, but also of the validity of the result.

The ambition of this project is therefore, through the automation of `libm` development, to solve the issues of implementation inefficiency and implementation obsolescence, and open up the set of functions and precisions, well beyond the `libm`.

[5] Multiple Inputs Multiple Outputs

Interestingly, the set of filters of interest is naturally open: automation is already provided by commercial tools like Matlab/Simulink, or tools supplied by FPGA vendors. However the implementations obtained through such tools are under-specified in terms of accuracy, and widely sub-optimal in terms of performance. An engineer who has to implement an application-specific filter has the choice between using such tools or spending weeks designing a high-performance implementation.

One decisive originality of this project is thus to attempt to capture filters and functions in the same framework. There are differences in the mathematical objects to be implemented (a filter is not a mathematical function $f(x)$: it processes signals and therefore depends on history). There are also differences in the metrics (relative error versus signal-to-noise ratio [62], transfer function error [37] or stability-related measures [78]). There are (fewer and fewer) differences in the target hardware (general-purpose processor versus DSP processor). However, there are also many similarities, and it is striking how two disjoint communities (computer science and signal processing) have developed parallel tools and formalisms which could be converged. For instance, the first step of the implementation process, approximation, uses two variants of the Remez algorithm (a numerical algorithm published in the 30s). The last step (evaluation) is based on the same primitives that have efficient hardware implementations: addition, multiplication, and pre-computed constant values.

Automating function implementation, and improving the automation of filter synthesis, will require us to formalize expert knowledge that covers and combines mathematical knowledge [1, 70, 65], algorithmic knowledge [60, 67, 31], computer arithmetic knowledge [68], and knowledge of the processor or target technology [60, 13, 63].

Several formalisms exist in some of these fields. Some more must be developed, for example to model the relevant capabilities of a given computing target. Here the most novel aspect of this project is to attempt to *integrate* all these formalisms, and to define and solve a global optimisation problem that covers the whole implementation process. This is a real challenge 1/ because of the span of fields to be covered, and 2/ because manual development involves global interdependencies between the various implementation steps that are non trivial, and sometimes cyclic. In many cases, experts still resort to trial and error (Matlab toolboxes for filter implementation are nothing more than a collection of tools to help the expert produce finite precision code with a trial and error approach).

Finally, with open-ended sets of functions to filters, we cannot rely on extensive testing of the generated code: this code must be correct by construction. Here, correctness essentially comes down to a complete and automated error analysis of the generated code, including the approximation issues but also the management of floating-point or fixed-point finite precision (entailing rounding errors) and finite range (entailing overflows and underflows). The developed formalisms will have to take this into account: we need to generate a proof of the numerical behavior [34, 35, 19] along with the code. This is another originality of the present project, in particular because we intend to bring to signal processing and control this focus on reliability, accuracy, and numerical correctness. In classical textbooks [52, 76, 63], there are very few pages on the impact of the implementation choices (such as wordlength or roundoff mode) on numerical quality.

2.2. STATE OF THE ART

Current state of the art in `libm` development currently essentially happens in two places.

In terms of performances, the best implementations are provided by the Intel team developing the Math Kernel Libraries (MKL). This team is co-located between Portland (Oregon) and Nizhniy-Novgorod (Russia). They have early access to future processors of the company, allowing them to optimize for these chips even before they are released on the market. They do publish some of their methods [13, 2], but we also have close collaborations with them, which have led to common publications [17, 21, 11]. In particular, among the participants to the present project, Christoph Lauter has worked for two years at Intel in this team in Portland after his thesis, Florent de Dinechin spent 9 month in the Intel Nizhniy-Novgorod Lab, and Nicolas Brunie should work as an intern in Portland for 6 months between March and September 2013. Similar efforts are made by most large processor makers, for instance AMD[6].

Performance of open-source implementations lag behind. For instance there was no high-performance vector implementation of the `libm` before the VDT effort at CERN [14].

In terms of assisting the development process, the state of the art is the body of work developed in the Arénaire team (renamed to AriC in 2011) in the last decade.

This work started with the CRLibm project[7]. The goal of CRLibm was to build on early work by Ziv at IBM [79] and to provide a library of *correctly rounded* elementary functions [17, 20], at a time where the accuracy of `libm` implementation was not specified in standards. The main outcome of this project was to show that correct rounding (that is, returning the best possible result considering the floating-point format) is achievable with a negligible performance overhead. This led to the recommendation in the 2008 revision of the IEEE 754 standard that the elementary functions should be correctly rounded.

However, another outcome of the CRLibm effort was to provide better tools to `libm` implementers. In particular, Arénaire developed the Sollya toolbox[8] to address implementation steps for which commercial computer algebra software, such as Maple or Mathematica, did not offer a satisfactory solution:

- The original Remez algorithms converges to the best polynomial approximation to a given function, but the coefficients of the obtained polynomial are real numbers. Textbooks suggested to round these to machine-representable coefficients, but the obtained polynomial is not, in the general case, the best approximation among polynomials with machine coefficients. Sollya includes a variant of the Remez algorithm which does provide the best polynomial with machine-representable coefficients [8].

- To implement this algorithm, but also to provide a guaranteed error bound on the approximation process, requires a safe supremum norm of the difference between a polynomial and a function. This is another unique feature of Sollya [10, 11].

Another distinctive aspect of the work around CRLibm was the focus on *formal proof*, then *formal proof generation* [19]. This was motivated by the sheer difficulty of ensuring the correct rounding property, and prompted the development of the Gappa tool[9] to assist the proof of tight error bounds on numerical code. Gappa can now be routinely applied to most `libm` developments.

Meanwhile, Arénaire/AriC also worked on the FLIP project, a floating-point library for an STMicroelectronics embedded processor without floating-point hardware. The most efficient implementations of division, square root, and some trigonometric functions were based on bivariate polynomial approximation [47, 48, 49]. A feature of this processor is that it has two parallel multipliers, each pipelined: this exposes both SIMD parallelism and pipeline parallelism, and the optimal evaluation of a polynomial in this case is a non-trivial optimization problem. This issue, which in the same context had been studied by the Intel team for the Itanium processor [13], typically leads on such VLIW targets to highly parallel evaluation schemes

[6] `http://developer.amd.com/tools/cpu-development/libm/`
[7] `http://lipforge.ens-lyon.fr/www/crlibm/`
[8] `http://sollya.gforge.inria.fr/`
[9] `http://gappa.gforge.inria.fr/`

that outperform Estrin's classical scheme [53, 67]. With respect to these previous works, the originality of the code generator for polynomial evaluation (CGPE) [66] developed for FLIP is to control the accuracy of parallel evaluation. It also addresses fixed-point implementations.

At the same time, the Arénaire project also pioneered the implementation of elementary functions on FPGAs [26]. It was shown that FPGA implementations of exponential, logarithm [25] and trigonometric functions [24] vastly outperformed their processor counterparts while consuming very little resources. These first developments were completely manual, and were then partly automated in the FloPoCo project [10]. FloPoCo currently offers 3 elementary functions for FPGAs (exponential [22], logarithm [15] and power [16]). In terms of automation, the development is strongly assisted, but far from being automated [23]. To match FPGA bit-level flexibility, all the operators are fully parameterized by precision and range. This alone requires some automation [23], but otherwise the development of a new function in FloPoCo is mostly manual. Internally, these FPGA implementations use fixed-point arithmetic: there is a lot of convergence with FLIP. Altera, one of the two main FPGA makers, now also develops elementary functions, but with much less flexibility [54].

Beyond AriC, Nelson Beebe has attempted to automate `libm` generation with the MathCW[11] library [4], but this project seems dormant.

To address a wider set of functions, the Inria *Algorithms* project-team developed the *Dynamic Dictionary of Mathematical Functions*[12] (DDMF) [5], an interactive website about elementary and special functions, where the chapter devoted to each function is generated from both a differential equation and a set of initial values. The DDMF demonstrated that most basic properties and approximations needed to implement a function could be determined automatically using Computer Algebra tools.

On the filter side, a few toolchains are able to produce code from transfer function (Mathworks Filter Design HDL Coder[13], Xilinx FIR Compiler[14], Altera Megawizard[15], etc.). However they provide implementations that are far from the best implementation we can manually achieve (they target a fixed structure, they use a fixed wordlength, they do not provide numerical guarantees, etc.). For the implementation of Linear Time Invariant (LTI) filters, these tools only consider the direct form II, i.e., the classical recurrence relation computing the output from the input. It has been proved to be far from optimal, especially for sensitive filters [31], although such studies did not cover the full implementation process, ignoring the evaluation step.

The present project will build upon expertise accumulated since 2006 at the IRCCyN Lab on linear time invariant (LTI) filter implementation [36]. A Matlab toolbox (Finite Wordlength Realization (FWR) toolbox [16]) has been developed to analyze the finite wordlength effects of digital filters and controllers, and to find optimal realizations (algorithms) for a given transfer function with respect to performance measures, accuracy measures, and implementation cost. Another tool for the evaluation of filters in fixed-point (FiPoGen, not yet publicly released) is currently developed at LIP6 within the ANR project DEFIS (Design of fixed-point embedded systems).

The convergence of approximation problems for filter and function was the subject of the FiltrOptim exploratory project (PEPS) between AriC (then Arénaire) and the CAIRN team

[10] `http://flopoco.gforge.inria.fr/`

[11] `http://ftp.math.utah.edu/pub/mathcw/`

[12] `http://ddmf.msr-inria.inria.fr/`

[13] `http://www.mathworks.fr/products/filterhdl/`

[14] `http://www.xilinx.com/products/intellectual-property/FIR_Compiler.htm`

[15] `http://www.altera.com/products/ip/altera/megawizd.html`

[16] `https://gforge.inria.fr/projects/fwrtoolbox/`

(Lannion, where Thibault Hilaire was as a postdoc). A PhD on this subject is expected to start in AriC in the coming years.

To summarize, the current situation is the following. For a function that is regular enough (e.g., C^{∞}) on a closed interval, we are able to automatically generate a provably correct and provably optimal polynomial approximation. This has been integrated in a first metalibm prototype [17] currently developed at LIP6. This prototype should be extended with range reduction capabilities in order to address a wider space of functions. Besides, it currently does not attempt to exploit parallel polynomial evaluation. FloPoCo also provides a universal polynomial approximation engine comparable in scope (and in limitations) to the metalibm prototype [18]. There is a lot of convergence.

CGPE, currently maintained at LIRMM, does address parallel evaluation but was developed for a specific integer processor from STMicroelectronics. It should now be extended to floating-point as well, since modern floating-point units (e.g., SSE2 and AVX on x86 processors) but also GPUs and FPGAs exhibit SIMD and pipeline parallelism that should be exploited.

FLIP has demonstrated that implementations of floating-point elementary functions, internally using only fixed-point arithmetic, could match in performance floating-point-based implementations, while running on less powerful processors. This naturally leads us to attempt to generalize this approach even to floating-point enabled processors.

The back-end code generation is very different when one targets hardware or FPGAs instead of software. To start with, one must generate structural hardware description language (VHDL or Verilog) instead of C. FloPoCo will provide state of the art back-end code generation for FPGAs. However, it should share a common front-end for all that is concerned with range reduction, polynomial approximation, and polynomial evaluation.

CRLibm has been mostly dormant in the past 5 years, while the research focus shifted to assisting the development process. Besides, some of its design choices are obsolete (e.g., it still uses large tables) and reviving it in its current form would require considerable development effort. The present proposal will instead regenerate it from scratch, among other variants of the `libm`.

2.3. Position of the project

We hope to demonstrate a dramatic reduction of time-to-market in `libm` development, from weeks to minutes for a function, from several person·years to hours for a full `libm`.

For instance, the Kalray company designed a radically new processor, but currently relies on an open-source `libm` that was mostly written for the x86 family. We hope to improve the performance of this `libm` by a factor 2 to 5, with a corresponding improvement in power consumption (a critical aspect in embedded computing).

The open-source community will be a longer-term beneficiary: we hope to bring open-source `libm` performance in sync with the best commercial approaches, and set new standards for accuracy, in particular by disseminating correctly rounded functions (see Section 4.3 p. 26).

The relevance of FPGA and GPU acceleration will also be boosted by the availability of high-performance standard-compliant libms, and on-demand function and filter generation.

This project will benefit from results from two other recent ANR-funded projects: The DEFIS project aims at obtaining fixed-point implementations out of floating-point ones. Our difference is that we focus on filters or functions, therefore we have the opportunity to start with a mathematical specification, which is both purer and richer than existing floating-point C code. However we will share expertise on fixed-point implementation issues. The TaMaDi project aims at determining the precision required for correct rounding. We will exploit their results to implement correctly rounded functions.

[17] `http://lipforge.ens-lyon.fr/www/metalibm/`

This project also fits within the objectives of the MILyon labex and CalsimLab labex.

3. SCIENTIFIC AND TECHNICAL PROGRAMME, PROJECT ORGANISATION

3.1. SCIENTIFIC PROGRAMME AND PROJECT STRUCTURE

Filter and `libm` code generation share a common algorithmic structure. First, the mathematical object in input, a function (resp. a filter's frequency response), is approximated to yield a polynomial or rational function (resp. a finite- or infinite-response filter). This reduces the initial object to an expression involving only basic operations and constant values. Then, this expression must be evaluated on the target hardware architecture in an optimal way.

This structure is reflected in the 5 tasks of this project.

Task 1 manages the project and organizes the scientific dissemination.

Task 2 is concerned with the first steps of the implementation process, the front-end. It will define mathematical representations of a function (for instance as an expression, as a differential equation) or a filter (transfer function, frequency response specification, etc.). Such mathematical representations need to be completed with accuracy and performance specifications or constraints, in order to form the input to an automatic approximation step. The method will be based on case studies and the construction of prototype software. Our concern of unifying functions and filters essentially addresses this front-end, because this is where disjoint communities have different languages and constraints.

Task 3 is concerned with the last steps of the implementation process, the back-end. It addresses the implementation of an expression (including parallelisation, scheduling, etc) irrespective of the origin of this expressions (filters or functions). Here the objective is to model the various computing systems we target (embedded processors, high-end processors, GPUs, FPGAs), with their capabilities and constraints (available operators, available parallelism, memory hierarchy constraints, etc.), in a way that enables the generation of efficient code for each of these targets. A promising approach is an object-oriented hierarchy of targets, each target providing performance analysis methods and code generation methods, with common methods factored through inheritance.

Task 4 will study the global optimization issues that link the front-end and the back-end. For instance, the capabilities of the target should orient the range reduction and approximation steps. For filters, the optimal realization problem is also a global one. The overall implementation accuracy involves factors from the front-end (approximation error) and the back-end (evaluation error due to rounding). These complex interdependencies are well understood by experts, who nevertheless typically need to perform iterations of trial and errors. This task will attempt to capture and formalize this expertise, again in implementable form.

Task 5 is a transversal set of applications and case studies. A first objective will be to produce, using the tools developed in the other tasks, several instances of the standard mathematical library for several contexts. A second objective will be to offer also non-standard instances (for example lower-accuracy, higher-performance implementations, or implementations with restricted validity range, etc., all well specified). The last objective will be to package the metalibm generator and filter generator in a tool that can be used by non-experts to implement non-standard functions and filters. All these case studies will feed and be fed by tasks 2, 3, and 4.

3.2. DESCRIPTION BY TASK

Task 1: Project management (head: AriC)

Task 1 : Project management				
Start: *M*1 Duration: 48m Effort: 9 p.m				
Partners	AriC	LIP6	LIRMM	CERN
Effort (p.m)	4	2	2	1

Subtask 1.1: Project management *M*1 → *M*48

Participants: **AriC**, LIP6, LIRMM, CERN

- Project management and planning (progress analysis, conclusion on the project)
- Consortium agreement and intellectual property management
- Cost management (budgets and investment)
- Identification and resolution of potential conflicts
- Management of legal aspects for the consortium agreement
- Risk management (identification, assessment and risk control) and potential project strategy update

Subtask 1.2: Scientific and technical coordination *M*1 → *M*48

Participants: **AriC**, LIP6, LIRMM, CERN

- Coordination of tasks
- Management of shared resources (web sites, wiki, code repositories, shared equipment)
- Organization of internal meetings
- Monitoring of task deliverables
- Management of reports, cost statements, publications
- Compilation of information necessary for sub-task 1.1 to analyze the progress, results, and performance

Subtask 1.3: Scientific dissemination *M*1 → *M*48

Participants: AriC, **LIP6**, LIRMM, CERN

- A centralized website will expose the project, the partners, the publications, and offer the tools to download.
- There is a potential of one or two "popular science" publications on our subject.
- We will organize a special session and/or a tutorial session in an international conference on this topic. A special day of training in a thematic school is also planned. We will also promote our scientific and technical progress by integrating researchers and industrial practitioners throughout project development (thesis, inviting potential scientific advisors to annual project meetings, etc.).
- During our plenary sessions, we will organize some internal training courses for all the partners of the project.
- We will publish the results of this project in scientific journals, proceedings of international conferences, and seminars.

Task Deliverables and milestones:

Id	Title	Type	Date
M1a	Kick-off meeting	M	M1
D1b	Website + open-source repositories	W	M3 + updates
D1c	Consortium agreement	M	M6
M1d	Yearly workshop	M	M12
D1e	Yearly progress report	R	M13
M1f	Yearly workshop	M	M24
D1g	Yearly progress report	R	M25
M1h	Yearly workshop	M	M36
D1i	Yearly progress report	R	M37
M1j	Final open workshop	M	M48
D1k	Final report	R	M48

Id: M=Milestone, D=Deliverable Type: M=Meeting, R=Report, S=Software, W=Website

Task 2: Specification and approximation (head: LIP6)

Task 2 : Specification and approximation				
Start: *M1* Duration: 48m Effort: 55 p.m				
Partners	LIP	LIP6	LIRMM	CERN
Effort (p.m)	15	32	6	2

Task Objectives:

The goal of this task is to develop a formalism for the front-end of the implementation process that is common to filters and functions. This includes:

- the *specification* of a filter or function, including the description of the mathematical object but also of the constraints/targets of the implementation;
- *approximation* of the mathematical object according to the specification, yielding implementable operators (mostly additions and multiplications) and machine-representable constant values.

The method used will be first a sharing of bibliographies from computer science, signal processing, and control, and secondly, the development, in parallel but in close collaboration, of prototype integrated toolchains for filters and functions.

Subtask 2.1: Bibliography *M1* → *M6*

Participants: LIP, **LIP6**, LIRMM

The literature contains several techniques and approaches to the implementation of mathematical functions found in a typical `libm`. However, most contributions are mainly focused on one `libm` function (or a small class of functions, such as exp(x), 2^x and 10^x). This is due to the tediousness and complexity of manual development of `libm` functions. The knowledge of `libm` development is hence spread over different scientific works, written and denoted in different styles. Most often only the final implementation proposal is presented but the steps that led to it are omitted.

The aim of this subtask is first to exhibit such expert knowledge buried in the literature (both filters and functions) and to represent it in a uniform way. Then the algorithmic steps yielding the different algorithms are to be re-discovered and explicited, with common steps factored out.

Subtask 2.2: Specifying filters and functions $M18 \rightarrow M48$

Participants: **AriC**, LIP6, LIRMM, CERN

Computations are currently specified as *programming language expressions* involving basic operators and a limited set of elementary functions. For instance, the mathematical function $\sin(\omega t + \phi)$ will be implemented as `sin(omega*t+phi)`. This is unsatisfactory for several reasons.

Firstly, programmers have no way to express on which range they need a function, nor with which accuracy. With considerable efforts, they may assess the validity and precision of their code out of the specification of their library `sin`, but this is not trivial (think of what happens if $t \gg 1/\omega$). Then, chances are that this expression will be either not accurate enough, or much too accurate and therefore wasteful.

Secondly, the composition of library *computer artifacts* is very different from the mathematical composition, again leading to a loss of accuracy and performance. In the previous example, chances are that ω is defined somewhere in the code as `omega=2*Pi*T`: we have two occurrences of the irrational constant π (one hidden in `phi`, the other one is in the implementation of `sin` which will reduce its argument to a period by subtracting a multiple of π). These two occurrences should cancel their effect as in the mathematical function. On the opposite, in the computer implementation, they accumulate two inaccuracies due to the impossibility of representing π exactly using finite precision. And they do so at a large cost with respect to an ideally optimal implementation of this mathematical function.

The purpose of this subtask is therefore to go beyond this library-based representation of functions. We want to be able to input to our tools

- a specification of the mathematical function either as an expression (but then considered as a mathematical expression), or as a differential equation (the work on the DDMF has shown the potential of this approach), or as a set of experimental data points to be abstracted into a black-box function;
- a specification of a filter at the highest-possible level, either as an input-output mathematical expression, a transfer function, or as a frequency response specification;
- a specification of the accuracy, range, and performance constraints of the implementation.

The forms in which this specification will be expressed will be defined in close collaboration with CERN physicists (in the framework of sub-task 5.3).

Subtask 2.3: Unifying formalisms and algorithms for filters and functions $M9 \rightarrow M21$

Participants: AriC, **LIP6**, LIRMM, CERN

The main objective of this subtask is to bring together function code generation and filter code generation. If we attempt to write a common tool from scratch, we feel the risk is high to waste much time specifying an over-general tool that will end up as an empty shell. Instead, we suggest a pragmatic approach that builds up on expertises and existing prototypes of the project members. We intend to co-develop these prototypes, with regular discussions and a mutual commitment to converge on common formalisms, methodologies, and code.

A prototype tool-chain for filter approximation and evaluation will combine the evaluation part and the approximation part on the filter side. We will first focus on discrete approximation optimized for particular evaluation schemes, together with choosing a scheme for which approximation has near-optimal results. These two optimization goals conflict, so this sub-task will feed task 4 in charge of finding a global or Pareto optimal. Later, this prototype will be enhanced with the computation of a transfer function from a frequency response specification.

In parallel, a prototype integrated tool-chain for `libm` generation will apply the knowledge exhibited in subtask 2.2. Its goal will be to be complete (in the sense that it allows for end-to-end generation of certain `libm` functions), but initially neither optimal (in terms of generated code) nor complete in its `libm` covering. Lifting these restrictions will be the goal of subtask 5.1, which however may start earlier as it doesn't depend on subtask 2.2). Here we really focus on using this prototype to unify the understanding of function and filter development.

Task Deliverables and milestones:

Id	Title	Type	Date
D2.1	Survey on existing bibliography	R	M6
M2.2a	Meeting for filters and functions specification	M	M9
D2.2b	Intermediate filters and functions specifications	R	M18
D2.2c	Complete filters and functions specifications	R	M30
D2.3a	Prototype tool-chain for filter approximation and evaluation	S	M10
D2.3b	Prototype integrated tool-chain for `libm` generation	S	M13
D2.3c	Common formalism for `libm` and filter code generation	R	M33

Id: M=Milestone, D=Deliverable Type: M=Meeting, R=Report, S=Software, W=Website

Risks and alternative solutions

We will have to carefully evaluate, step by step, the relevance of attempting to unify the two worlds of filters and functions. For instance, two domain-specific input formalisms, speaking the languages of their respective communities, may turn out to be a better solution that one unified language. Similarly, if the size of common code is larger than the sum of the sizes of two disjoint codes, its relevance must be discussed.

Task 3: High-performance back-ends under constraints (head: LIRMM)

Task 3 : High-performance back-ends under constraints				
Start: *M*3 Duration: 45m Effort: 90 p.m				
Partners	LIP	LIP6	<u>LIRMM</u>	CERN
Effort (p.m)	36	12	36	6

Task Objectives

This task starts with a given expression approximating a filter or a function (for instance a FIR or a polynomial). It is concerned with its actual implemention on a given target hardware. The goal here is to design tools that automate the synthesis of codes or circuits using at best the underlying hardware features (in particular instruction-level, SIMD, and pipeline parallelisms). In addition, such tools will also assess the numerical accuracy of such codes or circuits. For instance, there are many possible parenthesizations (hence potential parallelizations) of a polynomial or of a sum of products, but when implemented in fixed- or floating-point, all do not offer the same numerical quality.

The approach will first consist in proposing a common formalism to model various computing systems with their capabilities and constraints, expressive enough to describe any of the targeted systems. Then it will propose tools to evaluate various metrics (accuracy, performance, ...). These tools will be used within this task to write code optimized for a particular hardware, but also to enable global optimizations that rely on design space exploration in Task4.

The achievements of this task will be applied to the automated implementation of `libm` functions and filters using floating-point or fixed-point arithmetic, or even mixing both of them.

Subtask 3.1: Arithmetic resource modeling *M3 → M38*

Participants: LIP, LIP6, **LIRMM**
This subtask is intended to propose a uniform formalism for the arithmetic units of the hardware architectures considered in this proposal, together with tools based on this formalism to output code optimized for various criteria like accuracy, performance, or ILP exposure. It will have to be expressive enough to represent (1) functional units available on general purpose processors, DSP processors, GPU, or FPGA (like FPU/ALU or DSP blocks for example), and (2) the language constructs relevant to access them. These language constructs will be very different when targetting software (in C or assembly code) or hardware/FPGAs (in a hardware description language, most probably VHDL). A second goal is therefore to hide these target-specific aspects behind a common layer.

This subtask will essentially focus on polynomial evaluation. For this purpose, LIRMM will use its experience in code synthesis with the CGPE tool (a Code Generator for Polynomial Evaluation). Using the formalism above, CGPE will be extended to be parameterized by a hardware target. It will thus produce optimized code for all the architectures targeted by the project, while satisfying accuracy constraints.

Beyond polynomials, the CGPE approach will also be extended to a wider class of expressions, including rational functions, FIRs, and IIRs.

Subtask 3.2: Parallelism modeling and auto-vectorization *M9 → M48*

Participants: **CERN**, LIP, LIRMM
On modern hardware, the computational efficiency of applications depends on the ability to generate large streamlined vectorized portions of the code. Modern compilers are more and more able to auto-vectorize long, computationally intensive loops with target-specific optimizations. Functions can be considered in two ways: First by providing “external” specialized routines for each type of vector and making them known to the compiler itself, as done for instance by Intel and AMD. It is limited to the standard set of elementary functions, and no standard covering functions with vector arguments exists. Second by providing inlined routines with scalar arguments, that the compiler will be able to auto-vectorize. It can be used for any function, and in [14], for the Cephes library, it was proven to be extremely effective, both in terms of computational efficiency and portability when compiled with recent GCC versions. In addition, some architectures embed dedicated hardware to efficiently perform some operations such as vectorized fixed-point function evaluation. It has to be wisely chosen for efficient executions and factored for various function evaluations.

For this purpose, the formalism of Subtask 3.1 will have to be adapted to represent parallelism available both at compile-time and at run-time. CERN, with the other partners, will also work on (1) proposing a “standard” signature for elementary functions with vector arguments, (2) engineering the “scalar version” of these functions, (in link with Subtask 5.1), such that it can be efficiently auto-vectorized for a large set of hardware targets, (3) evaluating the efficiency of this vectorization on different implementations, in particular in terms of how branches are evaluated, and (4) integrating this in large scientific applications to verify their computational efficiency w.r.t. current solutions.

Subtask 3.3: Circuit implementations of filters and functions *M6 → M45*

Participants: **LIP**, LIRMM, CERN
This subtask addresses the generation of *hardware description* (circuits), for filters and functions.

For this purpose, the specificities of the hardware target have to be considered. *(1) Unbounded parallelism.* A processor exposes fixed parallelism, while a VLSI circuit may be designed with arbitrary parallelism. An FPGA is between both, with a fixed but massive parallelism of the basic resources (LUTs, small multipliers and memories). *(2) Arbitrary precision at the granularity of the bit.* A processor works on fixed-size words, typically 32 or 64 bits, and at best, some operations may be performed on 8 or 16 bits SIMD-wise. In a circuit, the precision of each operation can be fixed at the bit resolution. Again, FPGAs present an intermediate situation where the bit-level flexibility is sometimes constrained for efficiency, for instance with their fixed-size (but small) multipliers.

These two specificities make the design of optimal circuits different from optimal software. On the one side, it is easier with fewer constraints. On the other side, it is also more difficult with a larger implementation space and more continuous cost/performance tradeoffs. In this subtask, these issues will be studied in two directions: by *factoring* at most the efforts between *hardware* and *software*, and by *separating* cleanly the back-ends and (possibly) the optimizers, with a strong link with Subtask 5.2 since such implementations are more efficient in *fixed-point*.

Task Deliverables and milestones:

Id	Title	Type	Date
M3.1a	Preliminary results on the unified formalism of arithmetic resources	R	M12
D3.1b	Unified formalism of arithmetic resources	S	M21
D3.1c	Tool using M3.1a for polynomial evaluation case study	S	M38
M3.2a	Preliminary results on the parallelism modeling and auto-vectorization	R	M18
D3.2b	First example of auto-vectorized functions	S	M33
D3.2c	Tool to auto-vectorize function implementations	S	M48
M3.3a	Preliminary results on the generation of hardware description	R	M15
D3.3b	First example of hardware description generation for filters/functions	S	M24
D3.3c	Tool to generate hardware description for filters/functions	S	M45

Id: M=Milestone, D=Deliverable Type: M=Meeting, R=Report, S=Software, W=Website

Risks and alternative solutions

Here also, one perceived risk is that the complexity of generic models (capturing both programmable processors and FPGAs/circuits) would be much larger than the sum of the complexities of individual models. The cost/benefits of this unification will be evaluated carefully, and backup solution is always to fall back on individual models. Another backup plan is to focus on one single language, C, and rely on C-to-hardware or high-level synthesis tools.

Task 4: Global optimization in the code generation (head: LIP6)

Task 4 : Global optimization in the code generation

Start: *M*1 Duration: 48m Effort: 65 p.m

Partners	LIP	LIP6	LIRMM	CERN
Effort (p.m)	20	35	10	0

Task Objectives

Code generation for mathematical functions and signal processing filters splits – at first sight – into two linearly aligned phases: an *approximation phase* and an *evaluation phase*. The amount of optimization in the finally generated code of course stems from local optimization in each of these phases.

However, in order to yield a globally optimized code, these local optimizations need to be combined into a global one. Doing so, interdependencies between the phases need to be understood, modelled and resolved.

The objective of this task is to guide local optimization done by tasks 2 and 3 in a way that leads to globally optimized code. The following method will be used:

- First, one subtask is to find ways to compute and express *schemes of evaluation and range reduction* that leave just enough local optimization space to the appropriate backends while setting narrow constraints to early prune the local searches.
- Similarily, schemes will be required to limit search spaces when approximation is to go *beyond polynomial approximation*.
- In order to efficiently resolve the interdependencies between the approximation phase and the evaluation phase, a globally optimizing algorithm must *model these interdependencies* on a higher level. This modelling is challenging but, once done, will allow classical global optimization algorithms to be used to guide the global code generation process.

Subtask 4.1: Range reduction schemes for elementary functions *M7 → M27*

Participants: **LIP6**, LIRMM
Range reduction is an important step in the implementation of most libm functions. The reduction step uses algebraic properties of the function together with properties of floating-point formats. Strictly speaking, there exists no equivalent step for the implementation of filters, but it could be argued that, for instance, transforming a signal from time domain to frequency domain is comparable.

While considerable work has already been done to generate the optimal or a near-optimal sequence of machine operations to evaluate a polynomial, almost nothing has been done on generating optimized range reduction sequences.

This subtask is intended to cover this research from both the theoretical side and the practical side. On the theory side, questions include how to represent a range reduction scheme, encompassing several possible realizations each with its proper error bound, how to model and map to hardware and how to efficiently run through the possible research space. On the practical side, one of the goals of the subtask is to produce efficient generator codes for at least the three major range reduction schemes [67].

Subtask 4.2: Beyond polynomial approximation *M11 → M31*

Participants: AriC, **LIP6**, LIRMM
Classically, mathematical functions are implemented using polynomial approximation: the (range reduced) function is replaced by a polynomial.

While the theory and practice of polynomial approximation and their algorithms, such as the Remez algorithm, are well understood, these optimizing algorithms need to be queried with an appropriate scheme in input. A degree of the approximation polynomial needs to be determined in any case; for certain functions with particular properties like a zero in their definition domain or symmetries, a certain monomial basis needs to be chosen globally before locally optimizing the approximation polynomial.

Some mathematical functions may also benefit from rational approximation or approximation with polynomials of rational functions. In this case, the scheme the approximation algorithm is queried with is more complex. Additionally the scheme needs to be put in adequation with the evaluation phase that may or may not be able to efficiently evaluate certain types of rational functions.

Furthermore, some mathematical functions do not map to efficient range reductions while their inverse functions do allow for efficient range reduction. In these cases, iterative approximation of the functions using their inverses may be interesting. The unrolling of such iterative approximations boils down to evaluating rational approximations of high degrees that are very structured. Exhibiting schemes for such structured high-degree rational approximations is most challenging but is part of this subtask's objectives.

On the signal processing filter side, going beyond rational approximations means splitting a filter into several sub-filters when translating a signal response specification into transfer functions' coefficients. Time permitting, such multi-level filters may be studied as part of this subtask.

Subtask 4.3: Integrating local optimizers into a global one *M21 → M48*

Participants: Aric, **LIP6**, LIRMM

The algorithmic task of integration of the local optimizing routines for function and filter specification approximation, range reduction scheme implementation and for final evaluation code generation is one of the most challenging ones.

During this step, constraints of different natures will have be modeled and taken into account. They include, for instance:

- *Constraints due to the function's or filter's mathematical behavior*. Different range reduction and approximation schemes require the function or filter to be implemented to fulfil certain mathematical behavior: to be sufficiently smooth up to a certain order of differentiation, to present algebraic properties, etc. Global optimization may make use of certain locally optimizing approximation bricks if and only if the properties required by these algorithms have been shown to be satisfied by the respective function or filter.

- *Numerical accuracy constraints for evaluation code*. While subsequent steps, to be designed by LIRMM, will take them into account, global optimization will have to allocate accuracy targets in an optimized fashion.

- *Inherent hardware constraints*. These constraints may enable or disable certain ways of argument reduction, approximation schemes, etc. The use of floating-point or fixed-point arithmetic in the evaluation step will have to be taken into account early in the approximation phases.

The main method to integration will be to model the different constraints and corresponding behaviors with easy-to-compute cost (estimator) functions in a software interface. The final globally optimizing "director" algorithm, calling the different approximation and evaluation basic bricks with appropriate schemes in input, will then be inspired and based on classical optimization algorithms, minimizing the overall cost.

This subtask of integration of local optimizers is not merely reduced to produce theoretical algorithmic descriptions of globally optimized code generation. One of its objectives is to produce a generic tool-chain, based on the basic bricks developed for the prototype generators, for both `libms` and filters. It is intended to replace the prototypes in the end. The tool-chain will be available as a functional Open Source software deliverable to future research.

Task Deliverables and milestones:

Id	Title	Type	Date
M4.1a	Meeting to define range reduction scheme interface	M	M9
M4.1b	Range reduction scheme interface specification	R	M13
D4.1c	Range reduction detection prototype code	S	M27
M4.2a	Meeting to define approximation scheme interface	M	M13
M4.2b	Approximation scheme interface specification	R	M17
D4.2c	Prototype code for approximation beyond polynomials	S	M31
M4.3a	Basic bricks maturity evaluation meeting	M	M23
M4.3b	Algorithmic description of global code generator	R	M32
M4.3c	Meeting to specify interfaces with basic bricks	M	M34
D4.3d	Globally optimizing generic tool-chain	S	M48

Id: M=Milestone, D=Deliverable Type: M=Meeting, R=Report, S=Software, W=Website

Risks and alternative solutions

Approximations of a function by a rational functions or by iterative approximation by the inverse have been well studied in mathematical approximation theory, but there is very little literature about their practical implementation, or actual code. In classical `libm` implementations they are not commonly used for this reason. In the case when such novel approximation techniques would prove unfeasible in the early stages of this project, the more classical polynomial approximations schemes would be used as a fallback.

Task 5: Applications and case studies (head: AriC)

Task 5 : Applications and case studies				
Start: $M1$ Duration: 48m Effort: 93 p.m				
Partners	AriC	LIP6	LIRMM	CERN
Effort (p.m)	33	21	30	9

Subtask 5.1: Generating the core elementary functions of the standard `libm` $M1 \rightarrow M48$

Participants: **AriC**, LIP6, LIRMM

`Libm` functions are now perfectly specified by the IEEE 754-2008 standard [41]. For this reason, automating the development of such functions is both easier and more difficult than writing an open-ended function generator. One one hand, it is easier because the specification is there: this subtask does not depend on subtask 2.2, for instance. Besides, the gaps in the automation can be filled in an ad-hoc way for each function. For instance, we do not know yet how to provide the best possible range reduction for an arbitrary function (this is the goal of sub-task 4.1). However, when generating `libm` code for the exponential function for instance, we have a wide body of literature describing and comparing range reduction schemes for this specific function [29, 72, 30, 33, 60, 67], and we can automate among this known set of techniques, which is simpler. On the other hand, it is more difficult, because we must compete with code hand-written by experts.

Therefore, this subtask intends to design code generation tools that automate not only generic expertise, such as polynomial evaluation, but also function-specific expertise, for instance the various trigonometric range reductions that are relevant in various domains. The code will progress one function at a time, and this subtask will spread over the length of the project. For the easiest functions (the exponential and logarithm variants) we believe the generator may be written in a few months. For some parameterized special functions, we doubt we can achieve full automation in the 4 years of the project. The list of deliverables reflects this incremental development.

However, this subtask should generate each function in parallel for a wide range of `libm` variants, including

- a `libm` for Linux (including a correct rounding option, reviving CRLibm [79, 20]),
- a `libm` optimized for the Kalray MPPA embedded multicore,
- a `libm` using only integer instructions in link with subtask 5.2,
- well-documented sub-standard, but higher-performance implementations.

Subtask 5.2: Filters, controllers and functions in fixed point *M1 → M48*

Participants: **AriC**, LIP6, LIRMM

In principle, implementing a computation in fixed point is intrinsically more efficient than doing the same in floating-point: the management of the floating point involves all sorts of shifts which are not directly useful to the computation. Indeed, application-specific hardware is almost always designed in fixed-point, and the goal of the ANR project DEFIS is to obtain fixed-point implementations out of floating-point ones. When implementing filters or functions, we do not start with code, but with a mathematical specification, which is both purer and richer. Besides, this specification is relatively simple: this enables a deep understanding of their numerical behavior, enabling us to size each and every operator and data-path to the minimum.

When targeting hardware or FPGAs, the benefits are huge. The key to the efficiency of FloPoCo's implementation of elementary functions, with respect to their processor counterpart, is that there are simply fewer bits flipped or moved in the FloPoCo version. Similar benefits are obtained when targeting processors without floating-point hardware, as was demonstrated by the FLIP project [47].

However, beyond these obvious targets, this subtask will also investigate fixed-point implementations for FPU-enabled processors or GPUs. There are two reasons why this could improve performance. The first one is that fixed-point addition typically costs one cycle, versus five for floating-point addition. The second reason is that for filters and functions, we will be able to manage the exponent (the point) statically, which means that for a given word size (32 or 64 bits) we have more bits of precision: we recover the exponent bits of a floating-point format.

Finally this subtask will investigate using fixed- and floating-point at the same time, to fully utilize processor resources.

Subtask 5.3: A toolbox for high-energy physics code *M6 → M48*

Participants: **CERN**, AriC, LIP6, LIRMM

The particle physics experiments at the Large Hadron Collider (LHC[18]) at CERN record every year tens of petabytes of raw data representing the electronic signals coming from the detectors' readouts. The transformation of these signals into the building blocks of prestigious discoveries is delegated to *reconstruction* and *simulation* programs [12], accounting for millions of lines of code, often part of codebases untouched since decades. The computing power needed to run these programs can be only provided by a network of computing centers spread all over the world: the LHC Computing Grid [7].

For such complex applications, the impact of elementary functions evaluation on the overall runtime is sizeable (up to one fourth of the total [3, 42]), with an obvious effect on the

[18] `http://public.web.cern.ch/public/en/lhc/lhc-en.html`

power consumption of the hardware exploited for their execution. This situation clearly needs improvement.

In order to address this issue, this subtask will investigate a tool that could enable easily tuning of the accuracy/performance trade-off of numerical function implementations.

The main challenges facing this goal are:

- to reach common terminology and formalisms in this interdisciplinarity part of the project,
- to cover typical High Energy Physics (HEP) use cases, such as mathematical functions of multiple variables, or functions originated from empirical fits of experimental data,
- to achieve the proper balance between the level of automation and the flexibility of the tool.

Addressing these challenges will require assessing the level of usability necessary in the context of HEP software, and identifying a metric to quantify the quality of the physics results obtained, with respect to the variations in accuracy introduced in mathematical functions.

Task Deliverables and milestones:

Id	Title	Type	Date
D5.1a	Generator for exponential, logarithm, trigonometric functions	S	M12
D5.1b	Generator for inverse trigonometric, atan2, and power	S	M24
D5.1c	Generator for selected special functions	S	M32
D5.1d	Generator for all `libm` functions	S	M45
M5.2a	Experiments around exp and log in fixed point	S	M6
D5.2b	Report on the relevance of fixed-point for core functions on the MPPA	R	M18
M5.2c	Integration of FloPoCo as a hardware back-end for filters and functions	S	M36
M5.3a	Specification meeting at CERN	M	M6
D5.3b	First prototype toolbox for physicists	S	M18
M5.3c	Return of experience meeting at CERN	M	M30
D5.3d	Public release of the toolbox for physicists	S	M42

Id: M=Milestone, D=Deliverable Type: M=Meeting, R=Report, S=Software, W=Website

Risks and alternative solutions

The main risk of subtask 5.1 is the sheer number of functions/contexts to implement with respect to the manpower involved. If automation and a careful planning of training periods are not enough, some intermediate meetings will re-prioritize our implementations to answer the perceived demand.

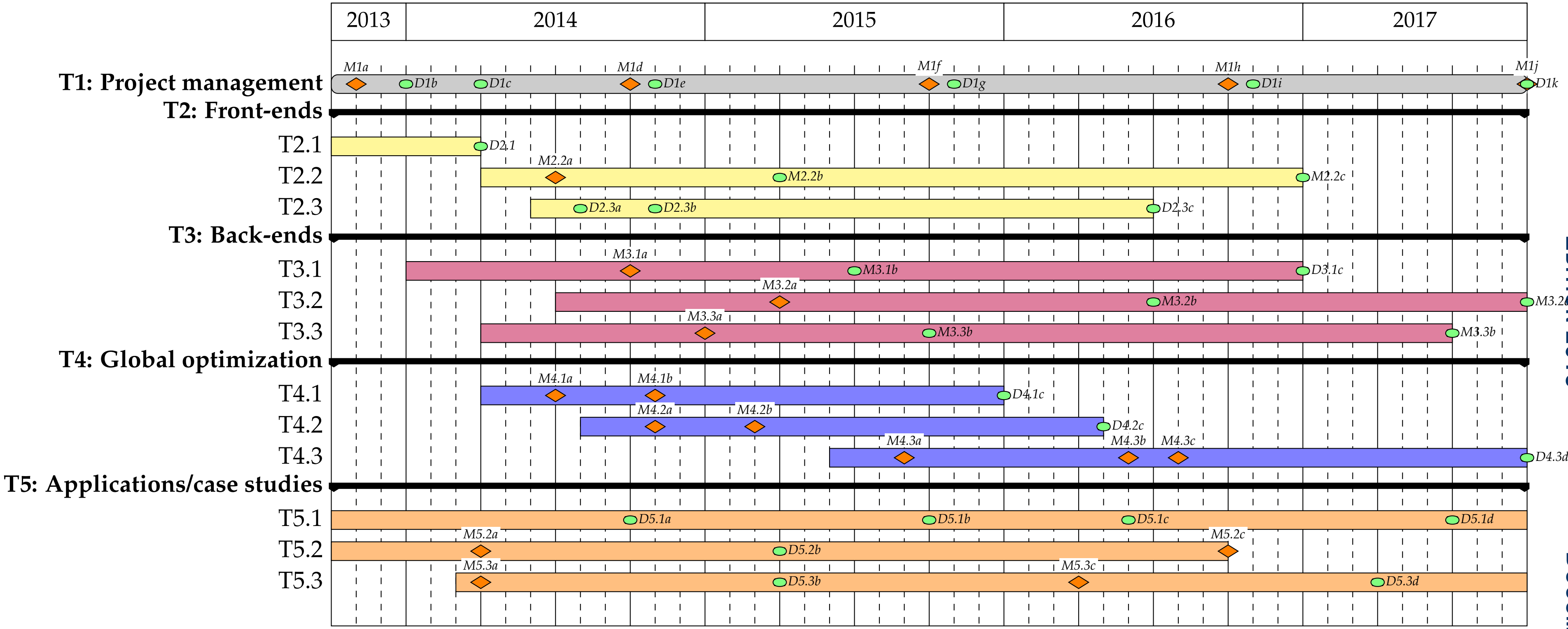
A risk in subtask 5.3 would be that the functions really needed by high-energy physics are too difficult to implement to be considered as case studies. We will then really need to implement a two-way dialog to explain the limits of the automatic approach, while we attempt to lift them.

3.3. CALENDRIER / TASKS SCHEDULE

Id	Title	Type	Date
M1a	Kick-off meeting	M	M1
D1b	Website + open-source repositories	W	M3 + updates
D1c	Consortium agreement	M	M6
D2.1	Survey on existing bibliography	R	M6
M5.3a	Specification meeting at CERN	M	M6
M5.2a	Experiments around exp and log in fixed point	S	M6
M4.1a	Meeting to define range reduction scheme interface	M	M9
M2.2a	Meeting for filters and functions specification	M	M9
D2.3a	Prototype tool-chain for filter approximation and evaluation	S	M10
M1d	Yearly workshop	M	M12
M3.1a	Preliminary results on the unified formalism of arithmetic resources	R	M12
D5.1a	Generator for exponential, logarithm, and trigs	S	M12
D1e	Yearly progress report	R	M13
M4.1b	Range reduction scheme interface specification	R	M13
M4.2a	Meeting to define approximation scheme interface	M	M13
D2.3b	Prototype integrated tool-chain for `libm` generation	S	M13
D2.2b	Intermediate filters and functions specifications	R	M13
M3.3a	Preliminary results on the generation of hardware description	R	M15
M4.2b	Approximation scheme interface specification	R	M17
M3.2a	Preliminary results on the parallelism modeling and auto-vectorization	R	M18
D5.2b	Report on the relevance of fixed-point functions on the MPPA	R	M18
D5.3b	First prototype toolbox for physicists	S	M18
D3.1b	Unified formalism of arithmetic resources	S	M21
M4.3a	Basic bricks maturity evaluation meeting	M	M23
M1f	Yearly workshop	M	M24
D3.3b	First example of hardware description generation for filters/functions	S	M24
D5.1b	Generator for inverse trigonometric, atan2, and power	S	M24
D1g	Yearly progress report	R	M25
D4.1c	Range reduction detection prototype code	S	M27
M5.3c	Return of experience meeting at CERN	M	M30
D4.2c	Prototype code for approximation beyond polynomials	S	M31
M4.3b	Algorithmic description of global code generator	R	M32
D5.1c	Generator for selected special functions	S	M32
D2.3c	Common formalism for `libm` and filter code generation	R	M33
D3.2b	First example of auto-vectorized functions	S	M33
M4.3c	Meeting to specify interfaces with basic bricks	M	M34
M1h	Yearly workshop	M	M36
M5.2c	Integration of FloPoCo as a hardware back-end	S	M36
D1i	Yearly progress report	R	M37
D2.2c	Complete filters and functions specifications	R	M39
D3.1c	Tool using M3.1a for polynomial evaluation case study	S	M38
D5.3d	Public release of the toolbox for physicists	S	M42
D3.3c	Tool to generate hardware description for filters/functions	S	M45
D5.1d	Generator for all elementary functions	S	M45
M1j	Final open workshop	M	M48
D1k	Final report	R	M48
D3.2c	Tool to auto-vectorize function implementations	S	M48
D4.3d	Globally optimizing generic tool-chain	S	M48

Id: M=Milestone, D=Deliverable Type: M=Meeting, R=Report, S=Software, W=Website

Gantt graph



Most software will be open-source (with the exception of closed-source components developed specifically for the industrial partner). Therefore, the incremental progress will be observable in the public repository of the project. The software deliverables listed above will correspond to beta or stable releases.

4. DISSEMINATION AND EXPLOITATION OF RESULTS, INTELLECTUAL PROPERTY

In terms of dissemination, this project has three main aspects:

1. the improvement and formalization of knowledge related to the implementation of elementary functions,
2. the development of a code generator framework that transforms the specification of a filter or mathematical function into an efficient implementation,
3. the improvement of computing systems at large, by the dissemination of more accurate and more efficient computing kernels.

4.1. PUBLICATIONS

Concerning the first point, this is basic research that will be published in the usual academic way. It should be noted that the project involves the author of the reference textbook on elementary functions [67], already in its second edition. This project will help updating this book to a third edition, and the time is right.

4.2. SOFTWARE DEVELOPMENTS

Concerning the second point, we envision a modular framework linking several components. These include existing components such as Sollya for polynomial approximation, CGPE for the generation of parallel evaluation code for these polynomials, FloPoCo for the FPGA back-end, all of which will be developed further in the framework of this project. Other modules will be developed from scratch, for instance modules related to range reduction, modules related to the input of a function defined by a differential equation, approximation and evaluation modules for filters, back-end modules for specific processors, and others. There will be a lot of glue code, and several experimental prototypes which will not necessarily lead to a final product, for instance those developed in subtask 2.3.

We envision this development in an open-source model, with one possible exception: a back-end module could be developed specifically for an industrial partner, targetting their technology (for instance Kalray with the MPPA processor), in the case it would include or expose technology that they do not wish to give away. This is consistent with what can be observed on the market at large: the main industry involved in the development of elementary functions libraries are processor makers such as Intel[19] or AMD[20], and they do so in order to sell their processors, not to sell the libm itself. Such libms are typically distributed for free but in binary form (closed source).

In addition to the classical argument that publicly-funded development should be open-source, open-source (with a free or permissive license) is preferred in this project for the following practical reasons:

- It does not hinder the publication of results, and actually helps it by making articles easily reproducible;
- The project depends on several open-source projects, and on no proprietary one;
- A wide dissemination of better libm functions will benefit computing at large (this is detailed below in 4.3). The best vector for that is an involvement in the free libms;

[19] http://software.intel.com/en-us/intel-mkl

[20] http://developer.amd.com/tools/cpu-development/libm/

- We have not performed any serious market analysis, but none of the partners sees a viable business model around the software developed in this project.

The actual license (GPL-like, LGPL-like or BSD-like) will be decided when setting up the consortium agreement. An issue is to properly manage will be the legal status (license) of the automatically generated `libm` and filter code.

We currently envision nothing that could be patentable, but this is not a dogmatic position.

4.3. Dissemination of more accurate and more efficient computing

One diffuse but important impact of the present project will be to improve the numerical quality of software at large. By disseminating on-demand generation of efficient and numerically well-specified functions, we will improve the performance of, and confidence in, applications that use them. This impact will span many fields, in particular scientific computing and control.

In particular, the industry has been reluctant so far to implement the transition to correctly rounded elementary functions, because it has a negative impact on performance (small as it may be). However, this transition would benefit to the public at large, making computations better specified, more deterministic, more reproducible and more trustable. Our objective is therefore to establish correct rounding as a de-facto standard in the open-source world, with the hope that it will compel the industry to align on this best practice. CRLibm was the first step of this plan: it was the reference implementation that enabled the recommendation of correct rounding in the IEEE 754-2008 standard. The next step is to disseminate widely this part of the standard, and this should be an outcome of the present proposal.

5. Consortium description

5.1. Partners description, relevance and complementarity

The proposed consortium (INRIA/AriC, UPMC/LIP6, LIRMM/DALI, CERN) gathers, around worldwide leaders in elementary functions, a number of expertises covering filters and control (in UPMC/LIP6), approximation (INRIA/AriC and UPMC/LIP6) and a wide range of technological targets (GPUs in LIRMM/DALI, embedded processors in INRIA/AriC and LIRMM/DALI, and FPGAs in INRIA/AriC). CERN brings scientific applications and case studies, and existing collaborations with companies such as Altera and Kalray will bring industrial ones.

As this project has ambitious software development goals, it should be noted that most members have established track records as software developers and managers of collaborative open-source projects. Among those projects mentioned in the state of the art p. 7, Jean-Michel Muller launched the CRLibm project (whose architect was Florent de Dinechin) then the TaMaDi effort. Christoph Lauter is one of the main developers of Sollya, and has been developing a metalibm prototype. Claude-Pierre Jeannerod is the main architect of FLIP and Guillaume Revy is the architect of the CGPE software. Thibault Hilaire is the main developper of the FWR toolbox. Florent de Dinechin pioneered the development of elementary functions for FPGAs and leads the FloPoCo project. Marc Mezzarobba contributed to the design of the DDMF and developed its numerical part. Danilo Piparo and Vincenzo Innocente developed the VDT library at CERN. Nicolas Brunie, in addition to his CIFRE PhD with LIP, is responsible at Kalray for the market-critical floating-point hardware/software stack.

5.1.1 Inria/AriC

AriC (Arithmetic and Computing, `http://www.ens-lyon.fr/LIP/AriC/`) is a joint CNRS-ENSL-Inria-UCBL team located at *Laboratoire d'Informatique du Parallélisme* (LIP) of École Normale Supérieure de Lyon.

The overall objective of AriC is, through *Computer Arithmetic*, to improve *computing at large*, in terms of *performance*, *efficiency*, and *reliability*.

Toward these goals, the AriC way is to

- work from the high-level specification of a computation to the lower-level details of its implementation,
- attempt to reconcile performance and numerical quality, both when building operators and when using existing operators,
- develop the mathematical and algorithmic foundations of computing.

To this purpose, the 12 permanent members of AriC, together with a similar number of PhD students and post-docs, gather a wide span of expertise, including floating-point computing, elementary functions, numerical validation and formal proof, FPGA arithmetic, algorithms for Euclidean lattices and their applications, cryptography, exact linear algebra, computer algebra, number theory.

AriC belongs to the MiLyon Labex, and is a follow-up of the Arénaire team which was ranked full A+ by AERES in 2010.

5.1.2 UPMC/LIP6

The PEQUAN (Performance and Quality of Numerical Algorithms) team is part of the department of Scientific Computing at Laboratoire d'Informatique de Paris 6 (LIP6) at Université Pierre et Marie Curie, Paris (`http://www-pequan.lip6.fr/`). The PEQUAN team counts amongst its members 3 professors, 8 assistant professors, 3 Post-Doc researchers and 10 PhD students.

One of the strengths of the PEQUAN team is the wide spectrum of aspects covered in numerical high-performance computing, computer arithmetic, cryptography and signal processing. The different researchers in PEQUAN put their different backgrounds at stake in order to:

- assess and control the accuracy of numerical codes through several a priori and posteriori approches [9, 11],
- trade off computing precision for speed in high-performance computing and, conversely, speed for reliability and numerical safety as required,
- apply approaches from computer arithmetic to symbolic calculations, computer algebra and to numerical applications in the fields of cryptography, signal processing and imaging.

PEQUAN is a member of the Labex CalSimLab coordinated by the Institute for Scientific Computation and Simulation (ICS) at UPMC Paris 6 and, through its research axis in Imaging, also a member of the Labex Smart promoted by the Institute for Intelligent Systems and Robotics (ISIR).

Since 2008, the PEQUAN team has contributed to several research projects related to scientific computing and computer arithmetic: ANR DEFIS, ANR TaMaDi, ANR Morpho, ANR GREMLINS, cooperations with TOTAL SA and AMD/CAPS Entreprise, Electricité de France, and Knowledge Inside SAS.

The PEQUAN team was ranked A by AERES in 2008.

5.1.3 LIRMM

The DALI research team (`http://webdali.univ-perp.fr/`) is a joint team in computer science of the Universite de Perpignan Via Domitia and the LIRMM laboratory (Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier - CNRS: UMR 5506). LIRMM was ranked A+ laboratory by AERES in 2009, and DALI was ranked A as a reseach team. The DALI team has 2 professors, 5 associate professors, 2 temporary professors, and 4 PhD students.

DALI's focus is on computer arithmetic and high-performance computing. Its main objective is to improve the performance and the reliability of both numerical programs and hardware architectures:

- For performance, it focuses on enhancing instruction-level and task parallelism exposed in numerical cores;
- For reliability, it focuses on controlling and validating computations in algorithms or codes, as well as improving the numerical quality of these computations.

For these purposes, DALI benefits from a strong expertise in computer arithmetic, floating-point arithmetic and numerical validation, high-performance and GPU computing, computer architecture and microarchitecture design, code transformation and synthesis, and static analysis.

Since 2008, DALI has contributed to several research projects related to the enhancement of performance and reliability of numerical programs and hardware architectures: ANR EvaFlo, ANR BioWic, ANR DEFIS, SARDANES (Fondation pour la Recherche en Aéronautique et Espace), Compil'HD (Concours "Chercheur d'avenir Région Languedoc-Roussillon"), MASSANE (CEA and Airbus).

5.1.4 CERN

The CERN SFT group (`http://sftweb.cern.ch`) develops and maintains common software for the physics experiments in close collaboration with the PH experimental groups, the CERN-IT Department and collaborating HEP Institutes. The scope includes common applications software infrastructure, frameworks, libraries, and tools; common applications such as simulation and analysis toolkits; and assisting the integration and adaptation of physics applications software in the running environment. Members of the group are making significant contributions to the work of the ROOT and Geant 4 collaborations, and also in projects organised as part of the Applications Area of the LHC Computing Grid Project. In addition several group members have direct responsibilities in the software projects of the LHC experiments, fulfilling key roles such as "software architect" and project leadership for the various data processing applications.

5.2. QUALIFICATION AND CONTRIBUTION OF EACH PARTNER

5.2.1 Inria/AriC

Florent de Dinechin, project coordinator, 43 years old. He is a senior assistant professor *(Maître de Conférences HDR)* at École Normale Supérieure de Lyon, where he leads the AriC team. His research interests include

- *implementation of software elementary functions*: he advised two PhDs on the subject, he was the main architect of the CRLibm software project, and he co-advised the initial development of the Sollya software.
- *hardware computer arithmetic*, and in particular *arithmetic for reconfigurable computing*: he advised three PhDs on the subject, and launched the FPLibrary, then FloPoCo software projects. FloPoCo has been used by tens of companies and universities worldwide, and was used to design the floating-point unit of the Kalray processor.

F. de Dinechin participated to the ANR projects EVA-Flo, TCHATER, and TaMaDi, and to industrial collaborations with Alcatel, Altera, Intel, and Kalray.

In the past 5 years he obtained one patent, authored or co-authored 5 chapters of the*Handbook of Floating-Point Arithmetic*, and published two book chapters, 6 articles in peer-reviewed international journals, and 19 articles in peer-reviewed international conferences (including one best paper award).

- J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhauser Boston, 2009.
- F. de Dinechin and B. Pasca: Reconfigurable arithmetic for HPC. In Vanderbauwhede and Benkrid, eds: *High-Performance Computing using FPGAs*, Springer, 2013.
- F. de Dinechin, P. Echeverría, M. López-Vallejo, and B. Pasca. Floating-point exponentiation units for reconfigurable computing. *ACM Transactions on Reconfigurable Technology and Systems*, to appear, 2013.
- F. de Dinechin and B. Pasca. Designing custom arithmetic data paths with FloPoCo. *IEEE Design & Test of Computers*, 28(4) :18–27, July 2011.
- F. de Dinechin, Ch. Lauter, and G. Melquiond. Certifying the floating-point implementation of an elementary function using Gappa. *IEEE Transactions on Computers*, 60(2) :242–253, February 2011.

Nicolas Brunie, 25 years old. He is completing a PhD at Kalray (CIFRE co-advised by F. de Dinechin). His research interests include hardware implementation of floating-point operators (he is in charge of the floating-point units of Kalray processors), and the implementation of low-level software primitives using them. He also investigates adding reconfigurable logic to the Kalray MPPA processor array.

- N. Brunie, F. de Dinechin, B. de Dinechin: A mixed-precision fused multiply and add. In *45th Asilomar Conference on Signals, Systems and Computers (ASILOMAR 2011)*, Asilomar, US, 2011.
- N. Brunie, S. Collange, G. Diamos: Simultaneous branch and warp interweaving for sustained GPU performance. In *39th International Symposium on Computer Architecture (ISCA 2012)*, Portland, US, 2012.
- N. Brunie, F. de Dinechin, M. Istoan and G. Sergent. L'arithmétique sur le tas. In *Conférence en parallélisme, architecture et système (COMPAS)*, 2013.
- N. Brunie, F. de Dinechin, B. de Dinechin. Conception d'une matrice reconfigurable pour coprocesseur fortement couplé. In *Conférence en parallélisme, architecture et système (COMPAS)*, 2013.

Claude-Pierre Jeannerod, 38 years old. He is a researcher *(Chargé de Recherche)* at Inria and a member of the AriC team. His research interests include *floating-point arithmetic* (design and software implementation of IEEE 754-2008) as well as *computer algebra* (fast algorithms for polynomial matrices and structured matrices).

He was/is a member of the ANR projects EVA-Flo and HPAC, and has been collaborating for 10 years with the Compilation Expertise Center of STMicroelectronics, Grenoble. In the context of this collaboration he has co-advised three PhD theses (Raina (2006), Revy (2009), Jourdan-Lu (2012)) on various aspects of *efficient algorithms for floating-point using integer arithmetic* and, with Guillaume Revy, is the main designer and developer of the FLIP library. His current research interests in computer arithmetic deal mainly with *how to best combine the integer and floating-point capabilities of modern architectures* in order to design efficient and accurate software for elementary functions and other small numerical kernels.

During the last five years he published 4 peer-reviewed journal papers (in TCS, IEEE TC, Math. of Comp.), and 9 peer-reviewed conference papers (at ARITH, ASAP, ISSAC,...), including one invited paper; he also authored or co-authored some chapters of the Handbook of Floating-Point Arithmetic.

- C.-P. Jeannerod, N. Louvet, J.-M. Muller. *Further analysis of Kahan's algorithm for the accurate computation of 2 x 2 determinants*. Math. of Comp. Accepted in 2012, to appear.
- C.-P. Jeannerod and J. Jourdan-Lu. *Simultaneous floating-point sine and cosine for VLIW integer processors*. Proceedings of ASAP 2012.
- C.-P. Jeannerod, H. Knochel, Ch. Monat, and G. Revy. *Computing floating-point square roots via bivariate polynomial evaluation*. IEEE TC, 2011.

- C.-P. Jeannerod, Ch. Mouilleron. *Computing specified generators of structured matrix inverses*. Proceedings of ISSAC 2010.
- J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhauser Boston, 2009.

Jean-Michel Muller is senior researcher *(Directeur de Recherches de 1ère classe)* at CNRS. His research interest span all aspects of computer arithmetic, in particular elementary functions, floating-point, and their formal proof. He founded (1998) then lead (1998-2004) the Arénaire team (CNRS/INRIA/ENS) which became the AriC team in 2011.

J.-M. Muller received the CNRS silver medal in 2013. He was head of the LIP laboratory from September 2001 to June 2006. He coordinated the ANR project TaMaDi and participated to the ANR project EVA-Flo.

In the past 5 years he coordinated the *Handbook of Floating-Point Arithmetic*, published one book, one book chapter, 11 peer-reviewed international journal papers, and 14 peer-reviewed international conference articles (including one best paper award).

- J.-M. Muller *Elementary Functions, Algorithms and Implementation*. Birkhauser Boston, 2nd edition, 2006.
- J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhauser Boston, 2009.
- F. de Dinechin, M. Ercegovac, J.-M. Muller and N. Revol, chapter Digital Arithmetic, in *Encyclopedia of Computer Science and Engineering*, B. W. Wah Editor, Wiley, January 2009.
- P. Kornerup, C. Lauter, V. Lefèvre, N. Louvet and J.-M. Muller, Computing Correctly Rounded Integer Powers in Floating-Point Arithmetic. *ACM Transactions on Mathematical Software*, Vol. 37 no 1, january 2010.
- C.-P. Jeannerod, N. Louvet, J.-M. Muller and A. Panhaleux, Midpoints and exact points of some algebraic functions in floating-point arithmetic, *IEEE Transactions on Computers*, Vol. 60 No 2, february 2011.

5.2.2 UPMC/LIP6

Christoph Lauter, coordinator for UPMC/LIP6, 32 years old. He is an associate professor *(Maître de Conférences)* at Université Pierre et Marie Curie, LIP6, PEQUAN team. His research interests include

- *software implementation of mathematical functions* for both IEEE 754-2008 based and arbitrary precision arithmetic. He is the main contributer of correctly rounded elementary functions in the CRLibm software project. He currently co-advises a PhD on the subject.
- *reliable floating-point environments* that, through rigorously verified and proven software, exploit best the underlying hardware for speed and, in particular, for enabling reliable and dependable results. He is the main software architect in the Sollya project. Sollya is used by about a dozen of international research and software development teams.

After his PhD at ENS-Lyon, Christoph Lauter has worked as a software engineer in the Numerics Team at Intel Corporation, Hillsboro, OR, USA for 2 years. His responsiblities included ensuring the compliance of Intel systems with the IEEE 754-2008 standard. He then joined UPMC/LIP6.

Christoph Lauter holds 6 publications in international journals (1 to appear) and 6 publications in international peer-reviewed conferences (1 to appear). He participated to the ANR projects EVA-Flo and TaMaDi.

- Ch. Lauter, and V. Lefèvre. An efficient rounding boundary test for pow(x,y) in double precision. *IEEE Transactions on Computers*, 58(2): 197-207, 2009.
- S. Chevillard, Ch. Lauter, J. Harrison, and Joldeş. Efficient and accurate computation of upper bounds of approximation errors. *Theoretical Computer Science*, 412(16): 1523-1543, 2011.
- F. de Dinechin, Ch. Lauter, and G. Melquiond. Certifying the floating-point implementation of an elementary function using Gappa. *IEEE Transactions on Computers*, 60(2): 242-253, 2011.

- F. de Dinechin, Ch. Lauter: Optimizing polynomials for floating-point implementation. In M. Daumas, and J. D. Bruguera, eds: *Proceedings of RNC8*, Elsevier, 2008.
- S. Chevillard, M. Joldeş, and Ch. Lauter: Sollya: an environment for the development of numerical codes, In K. Fukuda, J. van der Hoeven et al., eds: *Proceedings of ICMS'10*, Springer, 2010.

Olga Kupriianova, 23 years old. She is a PhD student in the PEQUAN team. Her research interests include numerical code generation for the implementation of mathematical functions, and IEEE 754-2008 compliant software environments. Olga Kupriianova is the main contributor to the Libieee754 library which aims at providing, on common Linux systems, a software environment that is fully compatible with the revised IEEE 754-2008 standard.

Olga Kupriianova graduated from Mechnikov university, Odessa, Ukraine, where she was awarded a "honor diploma" by her university for her Master work.

- O. Kupriianova, and Ch. Lauter: The libieee754 compliance library for the IEEE 754-2008 standard. In *15th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetics and Verified Numerics (SCAN 2012)*, Novosibirsk, Russia, 2012.

Thibault Hilaire, 35 years old. He is an associate professor *(Maître de Conférences)* at Université Pierre et Marie Curie, LIP6, PEQUAN team. His research interests include

- *equivalent transformations for linear control or signal processing algorithms* in order to find the most resilient realizations (ie algorithms) with respect to the finite wordlength effects (quantization of the parameters, roundoff erros, etc.);
- *the transfer function to code transformation* with fixed-point arithmetic, in order to provide efficient and accurate code for architectures without floating-point units. For a given transfer function, the goal is to find the optimal software/hardware implementation, according to numerical precision, performance and hardware/software cost criteria.

T. Hilaire obtained his PhD from Université de Nantes: "Analysis and synthesis of the Finite Word Length implementation of filters and controllers" in 2006. He was then a postdoctoral fellow at in Rennes at IRISA/INRIA (Cairn project), then at the Institute of Telecommunications of Vienna University of Technology, Austria before joining PEQUAN in 2009.

In the past 5 years, he published 7 peer-reviewed papers in international journals and 16 peer-reviewed papers in international conferences.

- B. Lopez, T. Hilaire, and L.-S. Didier. Sum-of-products evaluation schemes with fixed-point arithmetic, and their application to iir filter implementation. In Conference on Design and Architectures for Signal and Image Processing (DASIP), Oct. 2012.
- T. Hilaire and P. Chevrel. Sensitivity-based pole and input-output errors of linear filters as indicators of the implementation deterioration in fixed-point context. EURASIP Journal on Advances in Signal Processing, special issue on Quantization of VLSI Digital Signal Processing Systems, 2011.
- T. Hilaire. On the transfer function error of state-space filters in fixed-point context. IEEE Trans. on Circuits & Systems II, 56(12):936–940, December 2009.
- T. Hilaire, P. Chevrel, and J. Whidborne. Finite Wordlength Controller Realizations using the Specialized Implicit Form, Int. Journal of Control, 2(83):330-346, February 2010.
- T. Hilaire, D. Ménard, and O. Sentieys. Bit accurate roundoff noise analysis of fixed-point linear controllers. In Proc. IEEE Int. Symposium on Computer-Aided Control System Design (CACSD08), September 2008.

Marc Mezzarobba, 29 years old. He is a CNRS CR2 at LIP6. Before that, he was a PhD student in Computer Algebra (supervised by B. Salvy) in the Algorithms team of Inria Paris-Rocquencourt. His thesis, defended in October 2011, centers around the numerical evaluation of functions defined by linear differential equations. He then spent one year as a post-doctoral researcher in the AriC team. His current research interests include *symbolic-numeric algorithms*, especially dealing with ordinary differential equations and their solutions, and the *application of Computer Algebra to floating-point arithmetic*, in particular in the context of the numerical evaluation of special functions.

He authored or coauthored one article in a peer-reviewed international journal and 6 publications in peer-reviewed international conferences (including 2 to appear). His PhD thesis was awarded the École polytechnique 2012 PhD award in Computer Science.

- M. Mezzarobba and B. Salvy. Effective Bounds for P-Recursive Sequences. *Journal of Symbolic Computation* 45(10):1075–1096, 2010.
- M. Mezzarobba. NumGfun: a Package for Numerical and Analytic Computation with D-finite Functions. In S. M. Watt, editor, *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ACM, 2010, pages 139–146.
- S. Chevillard and M. Mezzarobba. Multiple precision evaluation of the Airy Ai function with reduced cancellation. In A. Nannarelli, P.-M. Seidel, P. T. P. Tang, editors, *Proceedings of the 21st IEEE Symposium on Computer Arithmetic*, IEEE, to appear.
- A. Benoit, F. Chyzak, A. Darrasse, M. Mezzarobba, S. Gerhold and B. Salvy. The Dynamic Dictionary of Mathematical Functions (DDMF). In K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software–ICMS 2010*, Lecture Notes in Computer Science, vol. 6327, Springer, pages 35–41.
- M. Mezzarobba. *Autour de l'évaluation numérique des fonctions D-finies*. Thèse de doctorat, École polytechnique, 2011, 222 pages.

5.2.3 LIRMM

Guillaume Revy, coordinator for LIRMM, 30 years old. He is associate professor *(Maître de Conférences)* at Université de Perpignan Via Domitia, and a member of the DALI UPVD-LIRMM project-team.

G. Revy obtained his Ph.D from ENS Lyon in 2009, was a post-doc within the ParLab group of University of California at Berkeley in 2010, then joined UPVD. He was/is a member of the ANR projects EVA-Flo and DEFIS. His research interests include computer arithmetic, implementation of floating-point operators for integer processors, and automated code synthesis particularly for polynomial evaluation. He is one of the main developer of the CGPE software tool and, with Claude-Pierre Jeannerod, is the main designer and developer of the FLIP library.

In the last 5 years, he published 1 peer-reviewed paper in an international journal, 8 peer-reviewed papers in international conferences (including one invited paper), and 1 peer-reviewed paper in a national conference.

- Ch. Mouilleron and G. Revy. Automatic Generation of Fast and Certified Code for Polynomial Evaluation. In 20th IEEE Symposium on Computer Arithmetic (ARITH'20), pages 233-242, 2011.
- C.-P. Jeannerod, H. Knochel, Ch. Monat, and G. Revy. Computing floating-point square roots via bivariate polynomial evaluation. In IEEE Transactions on Computers, vol. 60, no. 2, pages 214-227, 2011.
- Ch. Bertin, C.-P. Jeannerod, J. Jourdan-Lu, H. Knochel, Ch. Monat, Ch. Mouilleron, J.-M. Muller, and G. Revy. Techniques and tools for implementing IEEE 754 floating-point arithmetic on VLIW integer processors. In 4th International Workshop on Parallel and Symbolic Computing (PASCO'2010), 2010.
- F. de Dinechin, M. Joldes, B. Pasca, and G. Revy. Multiplicative square root algorithms for FPGAs. In 20th IEEE International Conference on Field Programmable Logic and applications (FPL'2010), pages 574-577, 2010.
- C.-P. Jeannerod, H. Knochel, Ch. Monat, G. Revy, and G. Villard. A new binary floating-point division algorithm and its software implementation on the ST231 processor. In 19th IEEE Symposium on Computer Arithmetic (ARITH'19), pages 95-103, 2009.

David Defour, 36 years old. He is associate professor *(Maître de Conférences)* at Université de Perpignan Via Domitia, and a member of the DALI UPVD-LIRMM project-team. His research interests include GPGPU computing, computer arithmetic and microarchitecture.

In the last 5 years, he published 1 peer-reviewed paper in an international journal, 7 peer-reviewed papers in international conferences, 1 peer-reviewed paper in a national journal, 1 book chapter, and 3 peer-reviewed papers in national conferences.

- S. Collange, M. Daumas, and D. Defour. État de l'intégration de la virgule flottante dans les processeurs graphiques. Revue des sciences et technologies de l'information, vol. 27/6, pp. 719-733, 2008.
- S. Collange, M. Daumas, and D. Defour. Line-by-line spectroscopic simulations on graphics processing units. Computer Physics Communications, vol. 178, pp. 135-143, 2008.
- S. Collange, D. Defour, Y. Zhang. Dynamic detection of uniform and affine vectors in GPGPU computations. Europar 3rd Workshop on Highly Parallel Processing on a Chip (HPPC), 2009.
- S. Collange, M. Daumas, D. Defour, and D. Parello. Barra, a modular functional GPU simulator for GPGPU. In MASCOTS, no. hal-00359342, 2010.
- S. Collange, M. Daumas, D. Defour. Interval arithmetic in CUDA. In GPU Computing Gems Jade Edition. Morgan-Kaufmann Publishers, 2011.

5.2.4 CERN

Vincenzo Innocente, coordinator for CERN, 52 years old. He is Senior Computational Physicist in the Physics department at CERN. He has been responsible of simulation and reconstruction software for three generations of collider experiments at CERN.

In the CMS experiment at LHC he has been Chief Software Architect till year 2006 and later leader of the "Performance Task Force".

Since 2008 he is involed in R&D projects in the CERN Physics Department to adapt HEP software to run efficiently on new multicore computing platforms.

- The challenge of adapting HEP physics software applications to run on many-core CPUs. Invited plenary talk to the 17th International Conference on Computing in High Energy and Nuclear Physics `http://indico.cern.ch/contributionDisplay.py?contribId=520&confId=35523`
- V. Innocente, L. Silvestris and D. Stickland, "CMS software architecture - Software framework, services and persistency in high level trigger, reconstruction and analysis," Comput. Phys. Commun. **140** (2001) 31.
- M. J. Kortelainen, P. Elmer, G. Eulisse, V. Innocente, C. D. Jones and L. Tuura, "The evolution of CMS software performance studies," J. Phys. Conf. Ser. **331** (2011) 042013.

Danilo Piparo 29 years old, is Junior Computational Physicist in the Physics department at CERN. A member since 2006 of the CMS "Performance Task Force", he has been responsible for the physics results validation of the experimental software for the CMS experiment between 2011 and 2012. Since 2013 he is part of the Frameworks and Multicore R&D project in the SFT group, involved in the effort of fitting the CERN software suite for the many-cores era.

D. Piparo obtained his PhD from Karlsruhe Institute of Technology in 2010.

- B. Hegner, P. Mato and D. Piparo, "Evolving LHC Data Processing Frameworks for Efficient Exploitation of New CPU Architectures", Proceedings IEEE-NSS 2012
- T. Hauth and V. Innocente and and D. Piparo, "Development and Evaluation of Vectorised and Multi-Core Event Reconstruction Algorithms within the CMS Software Framework", Journal of Physics: Conference Series, Vol 396 (2012)
- G. Benelli, B. Bozsogi, A. Pfeiffer, D. Piparo and V. Zemleris, "Measuring CMS Software Performance in the first years of LHC collisions", Proceedings 2011 IEEE NSS-MIC

5.2.5 Other involvements of project members

Partner	Involved people	Project name	Start/end dates
INRIA/AriC	F. de Dinechin, J.M. Muller	TaMaDi (ANR Blanc)	11/2010 - 11/2013
UPMC/Lip6	Thibault Hilaire	DEFIS (ANR INS)	11/2011 - 10/2014
INRIA/AriC	C.-P. Jeannerod	HPAC (ANR Blanc)	01/2012 - 01/2016
LIRMM	Guillaume Revy	DEFIS (ANR INS)	11/2011 - 10/2014

The relationship with DEFIS is detailed in 2.3 page 10. The TaMaDi ANR project was dedicated to one-time computations necessary to the efficient implementation of correctly rounded functions. The on-going ANR project HPAC (High-Performance Algebraic Computing) has a different focus, namely, parallel exact computing; a small part of this project studies how to use floating-point arithmetic in a certified way, but not how to implement libms.

6. Scientific justification of requested ressources

6.1. INRIA/LIP

Equipment: 10k€

We will buy a *Kalray MPPA developer* workstation for **10k€**. This is a high-end PC equipped with an accelerator board based on the Kalray K1 processor, and including the Kalray AccessCore software development kit (see `www.kalray.eu`). This workstation will enable us to develop a metalibm back-end targetted to this processor. The Kalray target is very different from the current mainstream platforms (x86 and ARM) in the core microarchitecture, but also at the macro-architecture level (memory organisation, wich is critical for table-based function evaluation, etc). We need this diversity to address the main challenge in task 4: understanding and formalizing the relationship between architecture parameters and elementary function performance. The access to actual hardware will also enable us to quantitatively assess *power consumption* and integrate it as a first-class performance metrics. All this work will also hopefully attract further research contracts with Kalray.

We will also need powerful FPGA hardware, but we have so far sucessfully attracted donation of such hardware (one Altera board worth 8000€ in 2011, one StoneRidge Technology board worth 5000 € in 2009, one FPGA-based XtremeData workstation worth 15000€ in 2008) and we expect this to continue.

Staff: 112 740€

This project will fund a PhD at Inria to work specifically on subtasks 3.3 and 5.2.

Operating costs: 67 800€

The main operating costs will cover missions related to the project:

- Two missions in France for project meetings per person per year (500€ each): **20k€**.
- Presentation of the project results at two international conferences (2.5k€ each) each year, starting in the second year of the project: **15k€**.

We will also buy 5 laptops or workstations over the span of the project: **10k€**.
Licenses for FPGA CAD software (ModelSim, ISE, Quartus) amount to 3k€/year: **12k€**.
Finally, we expect two 3-month training periods per year on the project.
Gratifications for these trainees (450€/month): **10.8k€**.

6.2. UPMC/LIP6

Equipment: 0€

LIP6 will not need any particular equipment over the span of the project, except small equipments detailed below.

Staff: 115 866€

A PhD at UPMC will work specifically on subtasks 2.1 and 2.3 as well as on task 4, with a specific focus on filter implementations.

Operating costs: 83 000€

At LIP6, 4 researchers (2 permanent members, 2 PhD students) will be implied in the project. LIP6 will require a workstation for each of these 4 participants, at a **total cost of 8k€**.

The LIP6 project members will participate to 2 coordination meetings per year with 2-3 persons per meeting, at 500€ each meeting per person. This yields a **total cost of 11k€**.

Two long-term technical meetings on a partner site are foreseen for 2 persons, for a **total cost of 4k€**. These meetings of about 1 week in length will permit these project members to discuss, specify, and detail software interfaces with the other project partners.

With a total implication of 102 person.months over 4 years, LIP6 will be involved in the project at the height of about 2.1 full-time researchers. Each researcher is expected to publish in at least 1 international conference per year. The total registration and travel costs for an international conference is estimated at 2.5k€. This yields a **total cost of 20k€** in international conferences for LIP6 in the project.

Moreover, the two permanent members (T. Hilaire and C. Lauter) are coordinators of two of the five tasks of the project. The tasks 2 and 4 will be particularly busy in 2014 and 2015, and the permanent members will have two PhD students to co-advise on the project in the same period. Consequently, LIP6 requires two years of partial teaching duty exemption (at 25%) per permanent. The **total cost is 40k€**.

6.3. LIRMM/DALI

Equipment: 0€

LIRMM will not need any particular equipment over the span of the project, except small equipments detailed below.

Staff: 95 000€

This project will fund a PhD at LIRMM, to work specifically work on subtasks 3.1 and 3.2.

Operating costs: 39 000€

For LIRMM, operating costs will cover in particular the following:

- Two project meetings in France per year and per person (500€ each): **total 12k€**.
- Participation to one national conference (1.25k€ each) and one international conference (2.5k€ each) per year, for a **total of 15k€**.

LIRMM will also buy 3 laptops or workstations over the span of the project, for a **total 6k€**.

Finally, LIRMM aims at recruiting two 6-month trainee students (MSc students, 500€/month each) for a **total of 6k€**.

6.4. CERN

Equipment: 35 000€

The interaction of large software applications with modern computing hardware is very complex. We will need to properly test, benchmark, and tune the produced mathematical libraries, in the context of actual CERN scientific code, and under realistic conditions. To do so, we want to aquire a variety of hardware close to what will be used in production in the next few years:

- We will buy two dual multi-core-cpu servers, respectively equipped with Nvidia GPUs and Intel Xeon-PHI co-processors: **25k€**.

  This equipment will be shared by all the participants of the project.

- We will buy a Kalray MPPA developer: **10k€**.

Staff: 0€

The CERN staff contributing to the project is entirely permanent to this institution.

Operating costs: 41 000€

The main operating costs will cover missions related to the project:

- Two missions in France for project meetings per person per year (500€ each): **total 8k€**.
- Presentation of the project results at two international conferences (2.5k€ each) each year, starting in the second year of the project: **total 15k€**.

We will also buy 3 laptops or workstations over the span of the project, for a **total of 6k€**. Gratifications for trainees should amount to a **total of 12k€**.

7. REFERENCES

[1] M. Abramowitz and I. A. Stegun. *Handbook of mathematical functions*. Applied Math. Series 55. National Bureau of Standards, Washington, D.C., 1964.

[2] C. S. Anderson, S. Story, and N. Astafiev. Accurate math functions on the Intel IA-32 architecture: A performance-driven design. In *7th Conference on Real Numbers and Computers*, pages 93–105, 2006.

[3] J. Apostolakis, A. Buckley, A. Dotti, and Z. Marshall. Final report of the ATLAS detector simulation performance assessment group. Cern-lcgapp-2010-01, CERN/PH/SFT, 2010.

[4] Nelson HF Beebe. A new math library. *International Journal of Quantum Chemistry*, 109(13):3008–3025, 2009.

[5] Alexandre Benoit, Frédéric Chyzak, Alexis Darrasse, Stefan Gerhold, Marc Mezzarobba, and Bruno Salvy. The Dynamic Dictionary of Mathematical Functions (DDMF). In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 35–41. Springer, 2010.

[6] J. E. Bertram. The effects of quantization in sampled-feedback systems. *Trans. of the American Institute of Electrical Engineers*, 77:177–182, 1958.

[7] Ian Bird et al. LHC computing grid: Technical design report. Technical Report LCG-TDR-001, CERN, 2005.

[8] Nicolas Brisebarre and Sylvain Chevillard. Efficient polynomial L^∞- approximations. In *18th Symposium on Computer Arithmetic*, pages 169–176. IEEE, 2007.

[9] Jean-Marie Chesneaux, Fabienne Jézéquel, and Jean-Luc Lamotte. *Stochastic arithmetic and verification of mathematical models*, pages 101–125. Springer, 2009. PEQUAN INT LIP6.

[10] S. Chevillard and Ch. 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.

[11] Sylvain Chevillard, Mioara Joldes, John Harrison, and Christoph Lauter. Efficient and accurate computation of upper bounds of approximation errors. *Theoretical Computer Science*, 412(16):1523–1543, April 2011.

[12] The CMS Collaboration. *CMS Physics: Technical Design Report Volume 1: Detector Performance and Software*. Technical Design Report CMS. CERN, Geneva, 2006.

[13] Marius Cornea, John Harrison, and Ping Tak Peter Tang. *Scientific Computing on Itanium®-based Systems*. Intel Press, 2002.

[14] Piparo Danilo. The VDT mathematical library. In *2nd CERN Openlab/INTEL Workshop on Numerical Computing*, 2012.

[15] Florent de Dinechin. A flexible floating-point logarithm for reconfigurable computers. Lip research report rr2010-22, ENS-Lyon, 2010.

[16] Florent de Dinechin, Pedro Echeverría, Marisa López-Vallejo, and Bogdan Pasca. Floating-point exponentiation units for reconfigurable computing. *ACM Transactions on Reconfigurable Technology and Systems*, 2013. To appear.

[17] Florent de Dinechin, Alexey Ershov, and Nicolas Gast. Towards the post-ultimate libm. In *17th Symposium on Computer Arithmetic*, pages 288–295. IEEE, 2005.

[18] Florent de Dinechin, Mioara Joldes, and Bogdan Pasca. Automatic generation of polynomial-based hardware architectures for function evaluation. In *Application-specific Systems, Architectures and Processors*. IEEE, 2010.

[19] Florent de Dinechin, Christoph Lauter, and Guillaume Melquiond. Certifying the floating-point implementation of an elementary function using Gappa. *IEEE Transactions on Computers*, 60(2):242–253, February 2011.

[20] Florent de Dinechin, Christoph Quirin Lauter, and Jean-Michel Muller. Fast and correctly rounded logarithms in double-precision. *Theoretical Informatics and Applications*, 41:85–102, 2007.

[21] Florent de Dinechin and Sergey Maidanov. Software techniques for perfect elementary functions in floating-point interval arithmetic. In *Real Numbers and Computers*, 2006.

[22] Florent de Dinechin and Bogdan Pasca. Floating-point exponential functions for DSP-enabled FPGAs. In *Field Programmable Technologies*, pages 110–117, December 2010.

[23] Florent de Dinechin and Bogdan Pasca. Designing custom arithmetic data paths with FloPoCo. *IEEE Design & Test of Computers*, 28(4):18–27, July 2011.

[24] Jérémie Detrey and Florent de Dinechin. Floating-point trigonometric functions for FPGAs. In *Field-Programmable Logic and Applications*, pages 29–34. IEEE, 2007.

[25] Jérémie Detrey and Florent de Dinechin. Parameterized floating-point logarithm and exponential functions for FPGAs. *Microprocessors and Microsystems, Special Issue on FPGA-based Reconfigurable Computing*, 31(8):537–545, 2007.

[26] Jérémie Detrey, Florent de Dinechin, and Xavier Pujol. Return of the hardware floating-point elementary function. In *18th Symposium on Computer Arithmetic*, pages 161–168. IEEE, 2007.

[27] P. Diniz and A. Antoniou. More economical state-space digital-filter structures which are free of constant-input limit cycles. *Acoustics, Speech and Signal Processing*, 34(4):807–815, August 1986.

[28] Y. Feng, P. Chevrel, and T. Hilaire. Generalized modal realisation as a practical and efficient tool for FWL implementation. *International Journal of Control*, 84(1):66–77, 2011.

[29] S. Gal. Computing elementary functions: A new approach for achieving high accuracy and good performance. In *Accurate Scientific Computations, LNCS 235*, pages 1–16. Springer Verlag, 1986.

[30] Schmuel Gal and Boris Bachelis. An accurate elementary mathematical library for the IEEE floating point standard. *ACM Transactions on Mathematical Software*, 17(1):26–45, 1991.

[31] M. Gevers and G. Li. *Parametrizations in Control, Estimation and Filtering Probems*. Springer-Verlag, 1993.

[32] H. Hanselmann. Implementation of digital controllers - a survey. *Automatica*, 23(1):7–32, January 1987.

[33] J. Harrison, T. Kubaska, S. Story, and P.T.P. Tang. The computation of transcendental functions on the IA-64 architecture. *Intel Technology Journal*, Q4, 1999.

[34] John Harrison. Floating point verification in HOL light: The exponential function. In *Algebraic Methodology and Software Technology*, pages 246–260, 1997.

[35] John Harrison. Formal verification of floating point trigonometric functions. In *Formal Methods in Computer-Aided Design: Third International Conference FMCAD 2000*, volume 1954 of *Lecture Notes in Computer Science*, pages 217–233. Springer-Verlag, 2000.

[36] T. Hilaire. *Analyse et synthèse de l'implémentation de lois de contrôle-commande en précision finie (Étude dans le cadre des applications automobiles sur calculateur embarquée)*. PhD thesis, Université de Nantes, June 2006.

[37] T. Hilaire and P. Chevrel. Sensitivity-based pole and input-output errors of linear filters as indicators of the implementation deterioration in fixed-point context. *EURASIP Journal on Advances in Signal Processing*, special issue on Quantization of VLSI Digital Signal Processing Systems, January 2011.

[38] T. Hilaire, P. Chevrel, and J-P. Clauzel. Low parametric sensitivity realization design for FWL implementation of MIMO controllers. In *Proc. of Control Applications of Optimisation CAO'O6*, April 2006.

[39] T. Hilaire, P. Chevrel, and J. Whidborne. Low parametric closed-loop sensitivity realizations using fixed-point and floating-point arithmetic. In *Proc. European Control Conference (ECC'07)*, July 2007.

[40] S.Y. Hwang. Dynamic range constraint in state-space digital filtering. In *IEEE Trans. on Acoustics, Speech and Signal Processing*, volume 23, pages 591–593, 1975.

[41] IEEE standard for floating-point arithmetic. IEEE 754-2008, also ISO/IEC/IEEE 60559:2011, August 2008.

[42] Vincenzo Innocente. Floating point in experimental HEP data processing. In *2nd CERN Openlab/INTEL Workshop on Numerical Computing*, 2012.

[43] R. Istepanian, G. Li, J. Wu, and J. Chu. Analysis of sensitivity measures of finite-precision digital controller structures with closed-loop stability bounds. In *IEEE Proc. Control Theory and Appl.*, volume 145, 1998.

[44] R. Istepanian and J.F. Whidborne, editors. *Digital Controller implementation and fragility*. Springer, 2001.

[45] R.H. Istepanian, J. Wu, and S. Chen. Sparse realizations of optimal finite-precision teleoperation controller structures. pages 687–691, June 2000.

[46] L. Jackson. Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form. *Audio and Electroacoustics, IEEE Transactions on*, 18(2):107–122, June 1970.

[47] Claude-Pierre Jeannerod and Jingyan Jourdan-Lu. Simultaneous floating-point sine and cosine for VLIW integer processors. In *23rd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2012)*, pages 69–76, Delft, Netherlands, February 2012.

[48] Claude-Pierre Jeannerod, Hervé Knochel, Christophe Monat, and Guillaume Revy. Computing floating-point square roots via bivariate polynomial evaluation. *IEEE Transactions on Computers*, 60(2):214–227, February 2011.

[49] Claude-Pierre Jeannerod, Hervé Knochel, Christophe Monat, Guillaume Revy, and Gilles Villard. A new binary floating-point division algorithm and its software implementation on the ST231 processor. In J.D. Bruguera, M. Cornea, D. DasSarma, and J. Harrison, editors, *Proceedings of the 19th IEEE Symposium on Computer Arithmetic (ARITH'19)*, pages 95–103, Portland, OR, USA, June 2009. IEEE Computer Society.

[50] T. Kailath. *Linear Systems*. Prentice-Hall, 1980.

[51] Nachiket Kapre and Andre DeHon. Accelerating spice model-evaluation using FPGAs. *Field-Programmable Custom Computing Machines*, pages 37–44, 2009.

[52] Ryan Kastner, Anup Hosangadi, and Farzan Fallah. *Arithmetic Optimization Techniques for Hardware and Software Design*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[53] Donald Knuth. *The Art of Computer Programming, vol.2: Seminumerical Algorithms*. Addison Wesley, 3rd edition, 1997.

[54] Martin Langhammer and Bogdan Pasca. Faithful single-precision floating-point tangent for FPGAs. In *Field Programmable Gate Arrays*, pages 39–42. ACM/SIGDA, 2013.

[55] G. Li. On the structure of digital controllers with finite word length consideration. *IEEE Trans. on Autom. Control*, 43(5):689–693, May 1998.

[56] G. Li and Z. Zhao. On the generalized DFIIt structure and its state-space realization in digital filter implementation. *IEEE Trans. on Circuits and Systems*, 51(4):769–778, April 2004.

[57] Gang Li. On pole and zero sensitivity of linear systems. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 44(7):583 –590, jul 1997.

[58] B. Liu. Effect of finite word length on the accuracy of digital filters - a review,. In *IEEE Trans. on Circuit Theory*, volume 18, 1971.

[59] W. Lutz and L. Hakimi. Design of multi-input multi-output systems with minimum sensitivity. *IEEE Trans. on Circuits and Systems*, 35(9):1114–1122, september 1988.

[60] Peter Markstein. *IA-64 and Elementary Functions: Speed and Precision*. Hewlett-Packard Professional Books. Prentice Hall, 2000.

[61] Peter Markstein. Accelerating sine and cosine evaluation with compiler assistance. In 2005. Proceedings. 8th Euromicro Conference on-Claude Bajard Jean and Michael Schulte, editors, *16th Symposium on Computer Arithmetic*, pages 137–140. IEEE Computer Society, 2003.

[62] Daniel Ménard and Olivier Sentieys. A methodology for evaluating the precision of fixed-point systems. In *ICASSP*, pages 3152–3155, 2002.

[63] U. Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Signals and Communication Technology. Springer, 2007.

[64] R. Middleton and G. Goodwin. *Digital Control and Estimation, a unified approach*. Prentice-Hall International Editions, 1990.

[65] S. Mitra and J. Kaise, editors. *Handbook for Digital Signal Processing*. Wiley-Interscience, 1993.

[66] Christophe Mouilleron and Guillaume Revy. Automatic Generation of Fast and Certified Code for Polynomial Evaluation. In *20th IEEE Symposium on Computer Arithmetic (ARITH)*, pages 233–242, Tübingen, Germany, August 2011.

[67] Jean-Michel Muller. *Elementary Functions, Algorithms and Implementation*. Birkhäuser, 2nd edition, 2006.

[68] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhauser Boston, 2009.

[69] C. Mullis and R. Roberts. Synthesis of minimum roundoff noise fixed point digital filters. In *IEEE Transactions on Circuits and Systems*, volume CAS-23, September 1976.

[70] F. W. Olver, D. W. Lozier, R. F. Boisvert, and C. W. Clark, editors. *NIST Handbook of Mathematical Functions*. Cambridge University Press, 2010.

[71] G. Paul and M. W. Wilson. Should the elementary functions be incorporated into computer instruction sets? *ACM Transactions on Mathematical Software*, 2(2):132–142, 1976.

[72] Ping Tak Peter Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 15(2):144–157, 1989.

[73] L. Thiele. Design of sensitivity and round-off noise optimal state-space discrete systems. *Int. J. Circuit Theory Appl.*, 12:39–46, 1984.

[74] B. Widrow. Statistical analysis of amplitude quantized sampled-data systems. *Trans AIEE*, 2(79):555–568, 1960.

[75] D. Williamson. Roundoff noise minimization and pole-zero sensibivity in fixed-point digital filters using residue feedback. In *IEEE Trans. on Acoustics, Speech and Signal Processing*, volume ASSP-43, October 1986.

[76] Roger Woods, John Mcallister, Richard Turner, Ying Yi, and Gaye Lightbody. *FPGA-based Implementation of Signal Processing Systems*. Wiley Publishing, 2008.

[77] J. Wu, S. Chen, G. Li, and J. Chu. Constructing sparse realizations of finite-precision digital controllers based on a closed-loop stability related measure. *IEE Proc. Control Theory and Applications*, 150(1):61–68, January 2003.

[78] J. Wu, S. Chen, J.F. Whidborne, and J. Chu. A unified closed-loop stability measure for finite-precision digital controller realizations implemented in different representation schemes. *IEEE Trans. Automatic Control*, 48(5):816–823, May 2003.

[79] A. Ziv. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Transactions on Mathematical Software*, 17(3):410–423, 1991.