TaMaDi
Table Maker's Dilemma
Septembre 2010

Jean-Michel Muller, CNRS

http://perso.ens-lyon.fr/jean-michel.muller/TaMaDi.html

Agenda

- présentation des équipes;
- présentation du projet TaMaDi (aspects administratifs et scientifiques);
- présentation de LLL
- point sur les embauches à prévoir: doctorant Péquan immédiatement, PostDoc Arénaire dans les mois qui viennent, PostDoc Marelle plus tard, stages
- discussion sur l'opportunité d'adjoindre d'autres personnes au projet
- discussion sur les achats de matériel à prévoir
- organisation de la coordination et de la communication du projet: page web, management du projet au quotidien
- planning prévisionnel de la première année: réunions plénières, visites inter-équipes

Presentation of the TaMaDi project

- correct rounding
- the Table Maker's Dilemma
- challenges
- L-algorithm, SLZ algorithm
- tasks

Correct rounding

In the IEEE 754 standard, the user defines an *active rounding mode* (or *rounding direction attribute*) among:

- round to the nearest (default) in case of a tie, value whose integral significand is even;
- round towards $+\infty$.
- round towards $-\infty$.
- round towards zero.

A correctly-rounded operation whose entries are FP numbers must return what we would get by infinitely precise operation followed by rounding.

Correct rounding

IEEE-754 (1985): Correct rounding for $+$, $-$, $\times$, $\div$, $\sqrt{}$ and some conversions. Advantages:

- if the result of an operation is exactly representable, we get it;
- if we just use the 4 arith. operations and $\sqrt{}$, deterministic arithmetic: one can elaborate algorithms and proofs that use the specifications;
- accuracy and portability are improved;
- playing with rounding towards $+\infty$ and $-\infty$ $\rightarrow$ certain lower and upper bounds.

FP arithmetic becomes a structure in itself, that can be studied.
IEEE-754 (2008): suggests correct rounding for some elementary functions.

The Table Maker's Dilemma

Consider the binary64/double precision FP number (base 2, $p = 53$)

$$x = \frac{8520761231538509}{2^{62}}$$

We have

$$2^{53+x} = 9018742077413030.9999999999999999805240837303\cdots$$

The Table Maker's Dilemma

Consider the binary64/double precision FP number (base 2, $p = 53$)

$$x = \frac{8520761231538509}{2^{62}}$$

We have

$$2^{53+x} = 9018742077413030.9999999999999999805240837303\cdots$$

So what ?

The Table Maker's Dilemma

Consider the binary64/double precision FP number (base 2, $p = 53$)

$$x = \frac{8520761231538509}{2^{62}}$$

We have

$$2^{53+x} = 9018742077413030.9999999999999999805240837303\cdots$$

So what ?

Hardest-to-round (HR) case for function 2^x and double precision FP numbers.

Correct rounding of the elementary functions

- this presentation: base 2, precision p;
- quite similar in base 10 (yet probably less important);
- FP number x and integer m (with $m > p$) $\rightarrow$ one can compute an approximation y to $f(x)$ whose error on the significand is $\leq 2^{-m}$;
- can be done with a possible wider format, or using algorithms such as TwoSum, TwoMultFMA, Dekker product, etc.
- getting a correct rounding of $f(x)$ from y: not possible if $f(x)$ is too close to a breakpoint: a point where the rounding function changes.

Correct rounding of the elementary functions

- RN mode,

$$\overbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}1000000\cdots000000}^{m \text{ bits}}xxx\cdots$$

or

$$\overbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}0111111\cdots111111}^{m \text{ bits}}xxx\cdots;$$

- other modes,

$$\overbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}0000000\cdots000000}^{m \text{ bits}}xxx\cdots$$

or

$$\overbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}1111111\cdots111111}^{m \text{ bits}}xxx\cdots.$$

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,
- Lindemann's theorem ($z \neq 0$ algebraic $\Rightarrow e^z$ transcendental) $\longrightarrow$ except for straightforward cases (e^0, ln(1), sin(0), ...), if x is a FP number, there exists an m, say m_x, s.t. rounding the m_x-bit approximation $\Leftrightarrow$ rounding $f(x)$;

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,
- Lindemann's theorem ($z \neq 0$ algebraic $\Rightarrow e^z$ transcendental) $\longrightarrow$ except for straightforward cases (e^0, ln(1), sin(0), ...), if x is a FP number, there exists an m, say m_x, s.t. rounding the m_x-bit approximation $\Leftrightarrow$ rounding $f(x)$;
- finite number of FP numbers $\longrightarrow \exists m_{\max} = \max_x(m_x)$ s.t. $\forall x$, rounding the $m_{\max}$-bit approximation to $f(x)$ is equivalent to rounding $f(x)$;

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,
- Lindemann's theorem ($z \neq 0$ algebraic $\Rightarrow e^z$ transcendental) $\longrightarrow$ except for straightforward cases (e^0, ln(1), sin(0), ...), if x is a FP number, there exists an m, say m_x, s.t. rounding the m_x-bit approximation $\Leftrightarrow$ rounding $f(x)$;
- finite number of FP numbers $\longrightarrow \exists m_{\max} = \max_x(m_x)$ s.t. $\forall x$, rounding the $m_{\max}$-bit approximation to $f(x)$ is equivalent to rounding $f(x)$;
- this reasoning does not give any hint on the order of magnitude of $m_{\max}$. Could be huge (yet some results, e.g., Nesterenko & Waldschmidt, give bounds).

Some insight, but no proof...

- the infinitely precise significand y of $f(x)$ has the form:

$$y = y_0.y_1y_2\cdots y_{p-1}\underbrace{01111111\cdots11}_{k\ bits}xxxxx\cdots$$

or with $k \geq 1$.

$$y = y_0.y_1y_2\cdots y_{p-1}\overbrace{10000000\cdots00}^{k\ bits}xxxxx\cdots$$

- Assuming that after the p^{th} position the "1" and "0" are equally likely, the "probability" of having $k \geq k_0$ is 2^{1-k_0};
- if we consider N input FP numbers, around $N \times 2^{1-k_0}$ values for which $k \geq k_0$;

$\longrightarrow$ no longer happens as soon as k_0 is significantly larger than $\log_2(N)$ (for one given value of the exponent, as soon as $k_0 \gg p$).

$\longrightarrow$ more precisely,

$$E(m_{max}) = \log_2(N) + \mu + o(1).$$

Problems to be solved

1. whenever possible, find the HR cases: x for which the infinitely precise significand of $f(x)$ is not a breakpoint and the (significand, relative) distance between $f(x)$ and a breakpoint is minimum. Smallest m s.t. that minimum distance $\geq 2^{-m}$: hardness-to-round for f;
2. or, when this is not possible, fix a value and check whether it is a lower bound on the distance between the infinitely precise significand of $f(x)$ and a breakpoint (except when $f(x)$ is a breakpoint);
3. convince users that a huge computation producing a yes/no answer or a short list of numbers was actually correct.

So far...

- L-algorithm: f approximated by linear functions in tiny subdomains $\longrightarrow$ given a finite grid and a straight line, *is there a point of the grid within ϵ from the line?* Drawback: many many subdomains.
- SLZ-algorithm: polynomial approximations of higher degree. Much fewer subdomains, yet much more complex calculations.
- HR cases obtained in binary64 for the most common functions (whole domain for logs and exps, limited domains for trigs);
- still very costly for binary64 (still unreachable for trigs of big arguments and power), getting results in higher precisions is quite a challenge;
- hum... *it might be possible that there remain a few bugs in the programs...* embarrassing: in CRLibm, functions come with proofs that the result is correctly rounded... if our HR cases are the good ones.

Challenges

- Our current methods cannot be used in big precisions. IEEE 754-2008 specifies binary128 and decimal128 formats;
- processes that generate HR cases: complex & long calculations (years of cumulated CPU time) $\longrightarrow$ casts doubt on the correctness on their results.

Need to reconsider our methods, and focus on 3 aspects:

1. big precisions: new algorithms for dealing with precisions $>$ binary64;
2. formal proof: provide proofs of the critical parts of our methods, or have our programs generating certificates that show the validity of their results (& focus on proving the certificates);
3. aggressive computing: even if we have faster algorithms, we must massively parallelize our methods.

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

- binary64/double precision (and possibly, double-extended) and decimal64: HR cases for all the functions of a standard C math library (will x^y and some trigs of large arguments be reachable?);
- the certificates proving the validity of these cases will be freely available too;
- larger formats: for the most common functions, a value ϵ s.t. if we approximate the function with relative accuracy ϵ, rounding approximation $\Leftrightarrow$ rounding exact value + certificates;
- open-source software, as portable as possible, for computing HR cases or accuracy bounds for other formats and functions and generating the certificates;
- publication in journals and/or proceedings of international conferences of algorithms, properties, etc.

Formulation (assuming round-to-nearest)

- Define $N = 2^p$ and assume that f maps $[1/2, 1]$ to $[1/2, 1]$ (elementary transformations & domain splitting);
- our problem can be formulated as: solve the equation

$$\left| \left(N \cdot f\left(\frac{3}{4} + \frac{x}{N} \right) - \frac{1}{2} \right) \text{ cmod } 1 \right| \leq \frac{1}{M} \qquad (1)$$

where

- x (the unknown) is an integer, $|x| \leq N/4$, M is a large integer, and
- y cmod 1 is the distance between y and the nearest integer.

L-algorithm

- Equation (1) solved using a piecewise degree-1 approximation to f
- in each subdomain, the L-algorithm performs a modified (subtractive) Euclidean GCD iteration. Extract from main loop:

while $x < y$ **do**
  if $u + v \geq N'$ **then**
    return (N', d)
  end if
  $y \leftarrow y - x; \quad u \leftarrow u + v$
end while
if $u + v \geq N'$ **then**
  return (N', d)
end if

Pros and cons

- simplicity:
  - easy to implement, finely tuned optimization (e.g. assembly code) is feasible, if not easy to achieve;
  - parallelization, hardware implementation of inner loops, probably easier
- huge number of subdomains $\longrightarrow$ the overhead related to one single subdomain must be kept as small as possible
  - tricky process for generating coefficients of the linear approximations;
  - hackings everywhere
- Comes with paper & pencil proof: difficult to make sure there is no hole in the proof.

Current version has been optimized for years $\longrightarrow$ very efficient, yet not so evolvable and provable.

Coppersmith's technique (used in the SLZ algorithm)

$$\left| \left(N \cdot f\left(\frac{3}{4} + \frac{x}{N}\right) - \frac{1}{2} \right) \text{ cmod } 1 \right| \leq \frac{1}{M}$$

- replace f by polynomial approximation F and $1/M$ by $1/M + \epsilon$;
- we restrict to a small sub-interval $x \in [x_0 - a, x_0 + a]$. Up to replacing $F(x)$ by $F(x + x_0)$, we assume that $|x| \leq a$, so that the problem becomes;

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

with $|x| \leq a$, $|z| \leq 1/M + \epsilon$;

Coppersmith's technique (used in the SLZ algorithm)

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

with $|x| \leq a$, $|z| \leq 1/M + \epsilon$.

- After clearing the denominators, we can thus assume we are looking for the solutions to

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

  where P is a polynomial, u and $v \in \mathbb{Z}$, $|u| < U$ and $|v| < V$.
- Technique: deduce equations over the integers, easier to solve.

Coppersmith's technique (used in the SLZ algorithm)

$$P(u, v) = 0 \mod R,$$

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

- for fixed α and all $i \leq \alpha, j, k$,

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

  same holds for any integer linear combination of such $P_{i,j,k}$.
- if $Q(u, v) = \sum_{i \leq d_1, j \leq d_2} q_{i,j} u^i v^j$ is such a polynomial with

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

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

Coppersmith's technique (used in the SLZ algorithm)

- To any finite set of $P_{i,j,k}$, we associate the lattice that they generate;
- using lattice basis reduction (the LLL algorithm), we look for two polynomials in this lattice with small enough coefficients (intricate analysis needed to understand for which values of M and a and which sets of $P_{i,j,k}$ such polynomials do exist).
- We are finally left with a polynomial system $Q_1(u, v) = 0, Q_2(u, v) = 0$ over the integers $\longrightarrow$ solved either by elimination techniques or by a suitable version of Hensel lemma.

LLL

- from a validation point of view: black box. Can be (very) quick and (quite) dirty. Once we get the two polynomials Q_1 and Q_2, verifying consists in:
  - checking that they satisfy (3),
  - checking that they have small enough coefficients,
  - recomputing the roots of the corresponding system.
- from a computational point of view: by far the most critical part.

Complexity

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

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

- "usual version" of the problem: $M \approx 2^p$ (comes from the "probabilistic" reasoning given above). The complexity is essentially a polynomial function times $2^{p/2}$. Too costly to be useful for big values (e.g., $p = 113$);
- "degraded" version: use the fact that if $M \approx 2^{p^2}$ the complexity boils down to polynomial $\Rightarrow$ information of the form *"if the approximation to f is with error $< \epsilon$ (with ϵ smaller than necessary yet reasonable for efficient implementation) then rounding the approximation $\Leftrightarrow$ rounding f"*

Task 1 — L-Algorithm

Participants: Arénaire, Pequan.
Current implementation: Perl scripts that generate C code, whose core is the L-algorithm. Mainly targeted to 32-bit computers and all the proofs are pen-and-paper proofs. Maple/intpakX used to generate the initial coefficients of the polynomials.
More or less a full rewrite is necessary. Subtasks:

- Replace Maple/intpakX by free, more reliable tools, such as MPFR, MPFI or Boost, or Sollya;
- The L-algorithm can be implemented in many ways, and a full analysis of the current implementation (time taken by each part, by the branches, etc.) would be useful to improve it.
- Port the L-algorithm to FPGAs (and GPUs?).

Task 2 — SLZ

Participants: Arénaire, Pequan.
Making SLZ really practical requires improving it by a not so large factor (say 10), which seems tractable.

- fine algorithmic improvements: use similar LLL steps on neighbour intervals, try to use the structure of the lattices (known beforehand) in order to speed-up the LLL reduction;
- understand the "degraded" version, and define which parameters are most likely to produce meaningful, computable and provable results.

Task 3 — Generating a certificate for the SLZ algorithm

Participants: Arénaire, Pequan.

- SLZ computations, especially degraded variant: huge piece of complicated code running a long time, and getting mostly a yes/no answer.
- since the LLL part is the most time consuming part, we can treat it as an oracle, and log its results in a so-called "certificate", the verification of which no longer implies LLL calls.
- The parameters of SLZ need to be rethought with this in mind.

Task 4 — Developing formal mathematical background

Participants: Arénaire, Marelle

Need preliminary work in order to get a sufficient amount of mathematical knowledge inside the prover.

- One important bit is the one about polynomial approximations that are present in both algorithms;
- Another part is the manipulations on polynomials done in, SLZ that require some basic properties of bivariate polynomials and Hensel's Lemma.

Task 5: Dealing formally with the algorithms

Participants: Arénaire, Marelle

- L-algorithm Critical parts will be formally proven.
- SLZ-algorithm Key aspect: evaluate the actual computing power needed to check the certificates.
  If what is provided by Coq is insufficient, a mixed approach based on oracles, or a parameterized checker that will be later instantiated by efficient data structures after extraction is taken into consideration.

Task 6: sequential reference implementation of SLZ

Participants: Arénaire, Pequan.
Today, `http://perso.ens-lyon.fr/damien.stehle/downloads/bacsel-1.0.tar.gz`: moderately optimized, not functional for all parameters. New reference implementation required:

- to test degraded version and estimate what kind of lower bounds can be hoped for in higher precisions;
- as a testbed for trying new ideas, especially fine algorithmic details or producing certificates;
- as a reference implementation in order to check the validity of more aggressively optimized versions

Task 7: Parallel portable and specialized versions of SLZ

Participants: Arénaire, Pequan

- First version: generic (i.e. portable on supports largely available, conventional multicore processors and possibly GPUs from NVIDIA and AMD), released as open source software to the community, and focusing mainly on the TMD for double precision;
- second version: specialized to tackle specific challenges (large precisions). Will run on national high end supercomputers.
- Along with this second version, another direction to tackle challenges may be a "HRCase@home" project.

Deliverables

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

Deliverables

- Task 4: at time T+12 a progress report (included in the T+12 report of the coordinator), at time T+24 a final report (included in the T+24 report of the coordinator) and Coq proofs posted on the web page of the project.
- Task 5: at time T+24 a progress report (included in the T+24 report of the coordinator), at time T+36 a final report (included in the T+36 report of the coordinator) and Coq proofs posted on the web page of the project.
- Task 6: at time T+12 a final report (included in the T+12 report of the coordinator).
- Task 7: at time T+24 a progress report (included in the T+24 report of the coordinator), at time T+36 a final report (included in the T+36 report of the coordinator) and programs for finding hardest-to-round cases posted on the web site of the project.