Proofs by Exhaustive Tests in Small Precision

Vincent LEFÈVRE

Arénaire, INRIA Grenoble – Rhône-Alpes / LIP, ENS-Lyon

GdT Arénaire, 2009-11-19

Outline

- Optimality of Algorithm 2Sum
- SIPE (Small Integer Plus Exponent)
- Optimal DblMult Error Bound

Optimality of Algorithm 2Sum

Based on the article *On the computation of correctly-rounded sums*, by Peter Kornerup, Vincent Lefèvre, Nicolas Louvet and Jean-Michel Muller, 19th IEEE Symposium on Computer Arithmetic (Arith-19), 2009.

Available on `http://hal.inria.fr/inria-00367584`.

Algorithm 2Sum*

$$
\begin{aligned}
s &= RN(a+b) \\
b' &= RN(s-a) \\
a' &= RN(s-b') \\
\delta_b &= RN(b-b') \\
\delta_a &= RN(a-a') \\
t &= RN(\delta_a+\delta_b)
\end{aligned}
$$

* due to Knuth and Møller.

- Floating-point system in radix 2.
- Correct rounding in rounding to nearest.
- Two finite floating-point numbers a and b.

→ Assuming no overflows, this algorithm computes two floating-point numbers s and t such that:

$$s = RN(a+b) \quad \text{and} \quad s+t = a+b.$$

Question: Is this algorithm optimal in any precision $p \geq 2$?

Optimality: In What Sense? Under What Conditions?

Optimality in time, size, depth?

Simple model:

- allowed operations: additions and subtractions;
- optionally minNum, maxNum, minNumMag, maxNumMag (IEEE 754-2008):
  - minNum and maxNum: minimum and maximum of 2 numbers;
  - minNumMag (resp. maxNumMag): the number with the smaller (resp. larger) magnitude, the minimum (resp. maximum) in case of equality;
- all operations take the same time;
- the first operation is $RN(a+b)$, without significant loss of generality.

Other common (standard) operations are probably useless or equivalent (e.g. $2x$).

$\rightarrow$ Minimality in term of:

- number of operations (sequential time);
- depth (parallel time).

The Mag2Sum Algorithm

If we have minNumMag and maxNumMag, we can derive a smaller algorithm from Fast2Sum:

Algorithm Mag2Sum

$$\begin{aligned} s &= RN(a+b) \\ a' &= \mathrm{maxNumMag}(a,b) \\ b' &= \mathrm{minNumMag}(a,b) \\ z &= RN(s-a') \\ t &= RN(b'-z) \end{aligned}$$

- 5 operations instead of 6;
- depth 3 instead of 5.

The Search for Minimal Algorithms

For the number of operations:

- enumerate all the possible algorithms (DAGs labelled by the operators) with at most n operations;
- equivalent DAGs (through obvious transformations in the order of operations and sign/add/sub changes) can be ordered, thus only one DAG can be kept;
- test each algorithm on 3 or 4 well-chosen pairs of inputs in precision p by comparing the result with the correct one;
- reject the algorithm if a result does not match.

For the depth:

- build a single DAG containing all the possible nodes at depth at most d;
- test the result of each node on 3 well-chosen pairs of inputs in precision p by comparing it with the correct one;
- take the maximum depth for which there is no match on the 3 pairs.

The Problem of the Precision

The number of possible precisions is infinite (or arbitrarily large).

The idea: choose the pairs of inputs in some form so that one can prove that a counter-example in one precision yields a counter-example in all (large enough) precisions.

Let us take $\varepsilon = \mathrm{ulp}(1) = 2^{1-p}$ and choose numbers of the form $u + v\varepsilon$, where u and v are small integers. Example of 2Sum on one of the pairs:

algorithm	precision 12	precision 17	expr.
a	1000.00000001	1000.0000000000001	$8 + 8\varepsilon$
b	1.00000000011	1.0000000000000011	$1 + 3\varepsilon$
$s = RN(a+b)$	1001.00000001	1001.0000000000001	$9 + 8\varepsilon$
$b' = RN(s-a)$	1.00000000000	1.0000000000000000	$1 + 0\varepsilon$
$a' = RN(s-b')$	1000.00000001	1000.0000000000001	$8 + 8\varepsilon$
$\delta_b = RN(b-b')$	0.00000000011	0.0000000000000011	$0 + 3\varepsilon$
$\delta_a = RN(a-a')$	0	0	$0 + 0\varepsilon$
$t = RN(\delta_a+\delta_b)$	0.00000000011	0.0000000000000011	$0 + 3\varepsilon$

The Pairs of Input Numbers

Chosen after testing various pairs.

If $\uparrow x$ denotes nextUp(x), the least floating-point number that compares greater than x:

$$\begin{aligned} a_1 &= \uparrow 8 & \qquad b_1 &= \uparrow\uparrow\uparrow 1 \\ a_2 &= \uparrow\uparrow\uparrow\uparrow\uparrow 1 & b_2 &= \uparrow 8 \\ a_3 &= 3 & b_3 &= \uparrow 3 \\ a_4 &= -a_1 & b_4 &= -b_1 \end{aligned}$$

In precision $p \geq 4$, this gives:

$$\begin{aligned} a_1 &= 8 + 8\,\varepsilon & \qquad b_1 &= 1 + 3\,\varepsilon \\ a_2 &= 1 + 5\,\varepsilon & b_2 &= 8 + 8\,\varepsilon \\ a_3 &= 3 & b_3 &= 3 + 2\,\varepsilon \\ a_4 &= -a_1 & b_4 &= -b_1 \end{aligned}$$

Precisions 2 to 12 (or 11) are tested. Results in precisions $p \geq 13$ can be deduced from the results in precision 12 (or 11).

The Proof of the Equivalence in Any Precision $p \geq 12$

Let us consider a computation DAG of maximum depth n. Here, $n = 6$. Assume that $p \geq n + 6$ (here, $p \geq 12$). We define: $\varepsilon_p = \mathrm{ulp}(1) = 2^{1-p}$. Main properties to be proved:

- the value of any node of the DAG has the form $u + v\varepsilon_p$, where u and v are "small" integers ($|v\varepsilon_p| < 1/2$) that do not depend on the precision p;
- since the integers u and v are small enough (see next slides), two values $u_1 + v_1\varepsilon_p$ and $u_2 + v_2\varepsilon_p$ are equal if and only if $u_1 = u_2$ and $v_1 = v_2$.

Note: we know that the depth of a minimal algorithm is bounded by 5, so that we could take $n = 5$ (but spurious algorithms might be obtained if the test is done in precision 11).

Ordering: $(u_1, v_1) < (u_2, v_2)$ if and only if $u_1 < u_2 \vee (u_1 = u_2 \wedge v_1 < v_2)$.

Because of the above properties, this will be equivalent to: $u_1 + v_1\varepsilon_p < u_2 + v_2\varepsilon_p$.

The Proof of the Equivalence in Any Precision $p \geq 12$ [2]

For each node with inputs $u_i + v_i\varepsilon_p$ and $u_j + v_j\varepsilon_p$, we define the pair $(u_k, \tilde{v}_k)$ as follows:

- add: $u_k = u_i + u_j$ and $\tilde{v}_k = v_i + v_j$
- sub: $u_k = u_i - u_j$ and $\tilde{v}_k = v_i - v_j$
- minNum: $(u_k, \tilde{v}_k) = \min((u_i, v_i), (u_j, v_j))$
- maxNum: $(u_k, \tilde{v}_k) = \max((u_i, v_i), (u_j, v_j))$
- minNumMag: $(u_k, \tilde{v}_k)$ is (u_i, v_i) if

  $$\begin{array}{l} |u_i| < |u_j| \quad \vee \quad (u_i = u_j = 0 \wedge |v_i| < |v_j|) \quad \vee \\ (|u_i| = |u_j| \wedge v_i \times \operatorname{sign}(u_i) < v_j \times \operatorname{sign}(u_j)) \quad \vee \\ (|u_i| = |u_j| \wedge |v_i| = |v_j| \wedge (u_i, v_i) < (u_j, v_j)), \end{array}$$

  else (u_j, v_j)
- maxNumMag: similar to minNumMag but changing the inequalities

and we define v_k by: $RN(u_k + \tilde{v}_k\varepsilon_p) = u_k + v_k\varepsilon_p$.

The Proof of the Equivalence in Any Precision $p \geq 12$ [3]

Properties to be proved by induction on the depth d of a node:

- The pair $(u_k, \tilde{v}_k)$ represents the exact value $u_k + \tilde{v}_k \varepsilon_p$ (i.e., the value of the operation before rounding).
- Unicity of the representation: $|v_k| \, \varepsilon_p < 1/2$.
- $|u_k| \leq 2^{d+3}$ and $|v_k| \leq 2^{d+3}$.
- The values u_k and v_k are integers that do not depend on p.

The consequence of the first property will be that the pair (u_k, v_k) represents the rounded value.

Any initial value (depth 0) has the form $u + v\varepsilon_p$, where u and v are integers that do not depend on p, such that $|u| \leq 8 = 2^3$ and $|v| \leq 8 = 2^3$.

Also, $|v| \, \varepsilon_p \leq 2^{3+1-p} \leq 2^{-n-2} < 1/2$.

The Proof of the Equivalence in Any Precision $p \geq 12$ [4]

Assume that the properties are satisfied for both inputs (u_i, v_i) and (u_j, v_j) of the node.

- Proof of the first property, i.e. $(u_k, \tilde{v}_k)$ represents the exact value $u_k + \tilde{v}_k \varepsilon_p$:
  - addition: $(u_i + v_i \varepsilon_p) + (u_j + v_j \varepsilon_p) = (u_i + u_j) + (v_i + v_j)\varepsilon_p = u_k + \tilde{v}_k \varepsilon_p$;
  - subtraction: $(u_i + v_i \varepsilon_p) - (u_j + v_j \varepsilon_p) = (u_i - u_j) + (v_i - v_j)\varepsilon_p = u_k + \tilde{v}_k \varepsilon_p$;
  - minNum and maxNum: $(u_1, v_1) < (u_2, v_2) \Leftrightarrow u_1 + v_1 \varepsilon_p < u_2 + v_2 \varepsilon_p$;
  - minNumMag and maxNumMag...

- From the definition of u_k for each operation, u_k is an integer and one has:

$$|u_k| \leq 2 \max(|u_i|, |u_j|).$$

Since the depth of each input is $\leq d - 1$, it follows that

$$|u_k| \leq 2 \cdot 2^{d-1+3} = 2^{d+3}.$$

Moreover the definition of u_k does not depend on p.

This proves the properties on u_k.

The Proof of the Equivalence in Any Precision $p \geq 12$ [5]

- For the same reasons, $|\tilde{v}_k| \leq 2^{d+3}$ and $\tilde{v}_k$ is an integer that does not depend on p.

  We need to prove that these properties are still satisfied after rounding, i.e., for v_k.

  If $u_k = 0$ (which does not depend on p), then $v_k = \tilde{v}_k$ (since $|\tilde{v}_k| \leq 2^p$), which proves the properties.

  Now let us assume that $u_k \neq 0$.

  Let E be the exponent of $u_k + \tilde{v}_k \varepsilon_p$, i.e.

  $$2^E \leq |u_k + \tilde{v}_k \varepsilon_p| < 2^{E+1}.$$

  Since $|\tilde{v}_k \varepsilon_p| \leq 2^{d+3+1-p} \leq 2^{n+4-p} \leq 1/2$ and u_k is an integer, E depends only on u_k and the sign of $\tilde{v}_k$ (it is the exponent of u_k, minus 1 if $|u_k|$ is a power of 2 and $u_k \tilde{v}_k < 0$); thus E does not depend on p.

The Proof of the Equivalence in Any Precision $p \geq 12$ [6]

The significand of $u_k + \tilde{v}_k \varepsilon_p$ as a number in $\left[2^{p-1}, 2^p\right[$ is

$$(u_k + \tilde{v}_k \varepsilon_p) 2^{p-1-E} = u_k 2^{p-1-E} + \tilde{v}_k 2^{-E}.$$

The rounding of $u_k + \tilde{v}_k \varepsilon_p$ to $u_k + v_k \varepsilon_p$ can be defined as the rounding of its significand to an integer (which will be equal to $u_k 2^{p-1-E} + v_k 2^{-E}$).

Since $2^E \leq |u_k| \leq 2^{d+3}$, one has $2^{p-1-E} \geq 2^{p-1-d-3} \geq 2$, so that $\boldsymbol{u_k 2^{p-1-E}}$ **is an even integer**, thus will not have any influence on the relative rounding error, defined as $\delta = (\tilde{v}_k - v_k) 2^{-E}$.

If $\{x\}$ denotes the nonnegative fractional part of x, then $\delta = \left\{\tilde{v}_k 2^{-E}\right\} - \Delta$, where $\Delta = 0$ if the rounding is done downward and $\Delta = 1$ if the rounding is done upward; by definition of the rounding-to-nearest with the even rounding rule, $\Delta = 1$ if and only if one of the following two conditions holds:

- $\left\{\tilde{v}_k 2^{-E}\right\} > 1/2$;
- $\left\{\tilde{v}_k 2^{-E}\right\} = 1/2$ and $\left\lfloor \tilde{v}_k 2^{-E} \right\rfloor$ is an odd integer.

The Proof of the Equivalence in Any Precision $p \geq 12$ [7]

As $\tilde{v}_k$ and E do not depend on p, the value of δ does not depend on p, and the value of v_k does not depend on p either.

Moreover $v_k 2^{-E} = \left\lfloor \tilde{v}_k 2^{-E} \right\rfloor$ or $\left\lceil \tilde{v}_k 2^{-E} \right\rceil$, so that v_k is an integer.

And since $|\tilde{v}_k| \leq 2^{d+3}$ and $E \leq d+3$, it follows that $\left|\tilde{v}_k 2^{-E}\right| \leq 2^{d+3-E}$, which is an integer. Hence $\left|v_k 2^{-E}\right| \leq 2^{d+3-E}$, and

$$|v_k| \leq 2^{d+3}, \quad \text{then} \quad |v_k|\,\varepsilon_p \leq 2^{d+3+1-p} \leq 2^{-2} < 1/2.$$

Assume that an algorithm $\mathcal{A}$ (or computation tree) is excluded for some precision $p \geq 12$ because on some input pair (a_p, b_p), $\mathcal{A}$ does not yield the expected result $t_p = a_p + b_p - RN(a_p + b_p) = u + v\varepsilon_p$.

Let $t'_p = u' + v'\varepsilon_p$ be the obtained result by running $\mathcal{A}$.

By hypothesis, $t'_p \neq t_p$, so that $(u', v') \neq (u, v)$. And since in a precision $q \geq 12$, any real can have at most one (u, v) representation satisfying $|v|\,\varepsilon_q < 1/2$, $(u', v') \neq (u, v)$ implies $t'_q = u' + v'\varepsilon_q \neq u + v\varepsilon_q = t_q$.

Thus $\mathcal{A}$ must be excluded for precision q.

Minimality of the size in precision $p \geq 2$

- The minimal add/sub algorithm giving the correct result is 2Sum (that is, with 6 operations); all the other equivalent algorithms reduce to 2Sum by using trivial transformations.

  Test on 33,467,556 DAGs (first 3 pairs only).

- The only minimal add/sub/minNum/maxNum algorithm giving the correct result is 2Sum, i.e. minNum and maxNum are useless here.

  Test on 308,124,270 DAGs.

- The only minimal add/sub/minNum/maxNum/minNumMag/maxNumMag algorithm giving the correct result is Mag2Sum (5 operations).

  Test on 9,274,728 DAGs.

Minimality of the depth in precision $p \geq 2$

For add/sub algorithms, the minimality for precision $p \geq 4$ is proved by computing the 89,903,977 values of depth less or equal to 4 for each of the first 3 pairs, in precisions 4 to 12.

The proof of the minimality for precisions 2 and 3 needs a 4th pair to test:

- Precision 2: $a_4 = 1$ and $b_4 = 6$.
- Precision 3: $a_4 = 10$ and $b_4 = 1$.

$\rightarrow$ In precision $p \geq 2$, the depth is at least 5 (depth of 2Sum).

If minNum, maxNum, minNumMag and maxNumMag are allowed, let us consider a domain for which all the operations are exact (e.g., a and b are small integers), so that the expression without the rounding must be mathematically equivalent to 0; a and b must also both appear in the expression. Impossible at depth 2.
$\rightarrow$ Thus the depth is at least 3 (depth of Mag2Sum).

SIPE (Small Integer Plus Exponent)

Correct rounding provided by:

- most processors, fast but only in 24, 53 and 64 bits;
- MPFR, but slow in small precision because of overhead due to generic precision.

→ Specific library for the small precisions: SIPE (Small Integer Plus Exponent).

- Idea based on DPE (Double Plus Exponent) by Paul Zimmermann and Patrick Pélissier: a header file (`.h`) providing the arithmetic, where a finite FP number is represented by a pair of integers (i, e), with the value $i \cdot 2^e$.
- Focus on efficiency:
  - exceptions are ignored and unsupported inputs are not detected;
  - restriction: the precision must be small enough to have a simple and fast implementation, without taking integer overflow cases into account. The maximal precision is deduced from the implementation (and the platform).
- Currently only the rounding attribute roundTiesToEven (rounding to nearest with the even rounding rule) is implemented.

SIPE: Provided Functions

Header file `sipe.h` providing:

- a macro `SIPE_ROUND(X,PREC)`, to round and normalize any pair (i, e);
- initialization: via `SIPE_ROUND` or `sipe_set_si`;
- `sipe_neg`, `sipe_add`, `sipe_sub`, `sipe_add_si`, `sipe_sub_si`;
- `sipe_nextabove` and `sipe_nextbelow`;
- `sipe_mul`, `sipe_mul_si`;
- `sipe_fma` and `sipe_fms` (optional, see below);
- `sipe_eq`, `sipe_ne`, `sipe_le`, `sipe_lt`, `sipe_ge`, `sipe_gt`;
- `sipe_min`, `sipe_max`, `sipe_minmag`, `sipe_maxmag`, `sipe_cmpmag`;
- `sipe_outbin`, `sipe_to_int`, `sipe_to_mpz`.

Bound on the precision:

FMA/FMS	32-bit integers	64-bit integers
No	15	31
Yes	10	20

SIPE: Implementation of Some Simple Operations

```
typedef struct { sipe_int_t i; sipe_exp_t e; } sipe_t;

static inline sipe_t sipe_neg (sipe_t x, int prec)
{ return (sipe_t) { - x.i, x.e }; }

static inline sipe_t sipe_set_si (sipe_int_t x, int prec)
{ sipe_t r = { x, 0 };
  SIPE_ROUND (r, prec);
  return r; }

static inline sipe_t sipe_mul (sipe_t x, sipe_t y, int prec)
{ sipe_t r;
  r.i = x.i * y.i;
  r.e = x.e + y.e;
  SIPE_ROUND (r, prec);
  return r; }
```

SIPE: Implementation of the Addition and Subtraction

```
#define SIPE_DEFADDSUB(OP,ADD,OPS)                                  \
  static inline sipe_t sipe_##OP (sipe_t x, sipe_t y, int prec) \
  { sipe_exp_t delta = x.e - y.e;                               \
    sipe_t r;                                                   \
    if (SIPE_UNLIKELY (x.i == 0))                               \
      return (ADD) ? y : (sipe_t) { - y.i, y.e };               \
    if (SIPE_UNLIKELY (y.i == 0) || delta > prec + 1)           \
      return x;                                                 \
    if (delta < - (prec + 1))                                   \
      return (ADD) ? y : (sipe_t) { - y.i, y.e };               \
    r = delta < 0 ?                                             \
      ((sipe_t) { (x.i) OPS (y.i << - delta), x.e }) :          \
      ((sipe_t) { (x.i << delta) OPS (y.i),   y.e });           \
    SIPE_ROUND (r, prec);                                       \
    return r; }

SIPE_DEFADDSUB(add,1,+)
SIPE_DEFADDSUB(sub,0,-)
```

Timings of the Search For Minimal Algorithms

Timings of the search for minimal algorithms with:

- the IEEE-754 double-precision arithmetic (binary64);
- MPFR 2.4.2-dev with 12-bit precision;
- SIPE with 12-bit precision.

Platform: Linux/x86_64 (3 GHz Pentium D).

Code compiled with GCC 4.3.4, using `-O3 -march=native -std=c99`.

	Timings			Ratios	
Allowed operations	`double`	MPFR/12	SIPE/12	S/D	M/S
add/sub	0.73	12.21	3.78	5.2	3.2
add/sub/min/max	8.79	91.95	27.33	3.1	3.4
all 6 operations	0.35	2.55	0.75	2.1	3.4

Optimal DblMult Error Bound

Algorithm DblMult defined in *Computing correctly rounded integer powers in floating-point arithmetic*, by Peter Kornerup, Christoph Lauter, Vincent Lefèvre, Nicolas Louvet and Jean-Michel Muller, to appear in *Transactions on Mathematical Software*.

Research report on `http://prunel.ccsd.cnrs.fr/ensl-00278430`.

Algorithm DblMult(a_h, a_ℓ, b_h, b_ℓ)

$$
\begin{array}{ccl}
[t_{1h}, t_{1\ell}] & = & \mathrm{Fast2Mult}(a_h, b_h) \\
t_2 & = & RN(a_h b_\ell) \\
t_3 & = & RN(a_\ell b_h + t_2) \\
t_4 & = & RN(t_{1\ell} + t_3) \\
[c_h, c_\ell] & = & \mathrm{Fast2Sum}(t_{1h}, t_4)
\end{array}
$$

Optimal DblMult Error Bound [2]

Theorem (from the article)

Let $\varepsilon = 2^{-p}$, where $p \geq 3$ is the precision of the radix-2 floating-point system used. If $|a_\ell| \leq 2^{-p}|a_h|$ and $|b_\ell| \leq 2^{-p}|b_h|$, then the returned value $[c_h, c_\ell]$ of DblMult satisfies:

$$c_h + c_\ell = (a_h + a_\ell)(b_h + b_\ell)(1 + \alpha) \quad \textit{with} \quad |\alpha| \leq \eta,$$

where $\eta = 7\varepsilon^2 + 18\varepsilon^3 + 16\varepsilon^4 + 6\varepsilon^5 + \varepsilon^6$.

Quasi-exhaustive tests in small precisions (3 to 8), i.e. with a bounded exponent, using SIPE. → Conjectured form of the worst case.

Tests using this conjectured form in precisions 3 to 14, using SIPE.
→ Conjectured error bound $|\alpha| < 6\varepsilon^2$, asymptotically reached.

Proof: work in progress.

DblMult: Quasi-Exhaustive Tests

```
$ ./dblmult 3 -
[...]
New worst case: (1.11e5,-1.10e2) * (1.11e5,-1.11e2)
|eta| = 146 / 2450
     <= 1.11101000000101101001000111011111e-5 <= 5.9591836748e-2
  bound 1.00110000001100010000000000000000e-3 (RNDZ) expected

$ ./dblmult 4 -
[...]
New worst case: (1.011e7,-1.010e3) * (1.101e7,-1.101e3)
|eta| = 626 / 32370
     <= 1.00111100110110010011001101011110e-6 <= 1.9338894039e-2
  bound 1.00000110000001100001000000000000e-5 (RNDZ) expected

$ ./dblmult 5 -
[...]
New worst case: (1.1011e9,-1.0111e4) * (1.0101e9,-1.0101e4)
|eta| = 2723 / 547491
     <= 1.01000101111100110001111110100010e-8 <= 4.9735977410e-3
  bound 1.11100101000000110000010000000000e-8 (RNDZ) expected
```

DblMult: Quasi-Exhaustive Tests – Bounds and Timings

Precision	Bound (in binary)	Timing
3	$1.1110100000101101001000111011111 \times 2^{-5}$	0
4	$1.0011110011011001001100110101110 \times 2^{-6}$	0
5	$1.0100010111110011000111111010010 \times 2^{-8}$	0
6	$1.0110100111001110111010010011111 \times 2^{-10}$	0
7	$1.0110100010001011000100111110011 \times 2^{-12}$	0.06
8	$1.0111011011000000010111010100011 \times 2^{-14}$	0.49
9	$1.0111101010001011100011011110010 \times 2^{-16}$	3.86
10	$1.0111101110111000100000101001001 \times 2^{-18}$	31.4
11	$1.0111110011010000100100001111001 \times 2^{-20}$	254
12	$1.0111110111101111000110000001010 \times 2^{-22}$	2055
13	$1.0111111011001000110010100001111 \times 2^{-24}$	16518
14	$1.0111111100111111111010110101101 \times 2^{-26}$	131522

Note: timings in seconds on a 2.2 GHz AMD Opteron.