IMPLEMENTATION OF THE RECIPROCAL SQUARE ROOT IN MPFR (EXTENDED ABSTRACT)

PAUL ZIMMERMANN

ABSTRACT. We describe the implementation of the reciprocal square root — also called inverse square root — as a native function in the MPFR library. The difficulty is to implement Newton's iteration for the reciprocal square root on top's of GNU MP's MPN layer, while guaranteeing a rigorous 1/2 ulp bound on the roundoff error.

The reciprocal square root is an important function in 3D graphics, for the normalization of 3D vectors, and as such has received much attention in the literature [5]. Indeed, given (x, y, z), compute $t = x^2 + y^2 + z^2$, $u = t^{-1/2}$, $x' = ux$, $y' = uy$, $z' = uz$, then (x', y', z') is a normalized vector with $x'^2 + y'^2 + z'^2 = 1$. A nice implementation attributed to Greg Walsh by Wikipedia is the following:

```
float invSqrt(float x)
 {
     float xhalf = 0.5f*x;
     int i = *(int *)&x;
     i = 0x5f3759df - (i >> 1);
     x = *(float *)&i;
     x = x * (1.5f - xhalf * x * x);
     return x;
 }
```

See also IBM's US patent 6654777 (Single precision inverse square root generator) issued on November 25, 2003.

The reciprocal square root is an interesting function *per se*, since it can be computed with Newton's iteration without any division, similarly to the reciprocal. Karp and Markstein have shown how the square root can be efficiently computed, by modifying the last Newton iteration of the reciprocal square root [4].

In the context of the MPFR library [2], our aim is to provide a native implementation of the reciprocal square root which guarantees correct rounding. By "native implementation", we mean it should rely on the GMP MPN layer only, which provides operations on arrays of words (addition, subtraction, multiplication) [3].

MPFR reciprocal square root (function `mpfr_rec_sqrt`) is based on Ziv's strategy and the `mpfr_mpn_rec_sqrt` function, which given a precision p, and an input $1 \leq A < 4$, returns an approximation x satisfying

$$x - \frac{1}{2} \cdot 2^{-p} \leq A^{-1/2} \leq x + 2^{-p}.$$

The `mpfr_mpn_rec_sqrt` function is based on Newton's iteration and the following lemma [1]:

Date: March 2008.

Centre de Recherche INRIA Nancy Grand Est, Projet CACAO - Bâtiment A, 615 rue du Jardin Botanique, F-54602 Villers-lès-Nancy Cedex, zimmerma@loria.fr.

Lemma 1. *Let $A, x > 0$, and $x' = x + \frac{x}{2}(1 - Ax^2)$. Then*

$$0 \leq A^{-1/2} - x' = \frac{3}{2}\frac{x^3}{\theta^4}(A^{-1/2} - x)^2,$$

for some $\theta \in (x, A^{-1/2})$.

Proof. Newton's iteration consists in approximating the function by its tangent. Let $f(t) = A - 1/t^2$, with ρ the root of f. The second-order expansion of f at $t = \rho$ with explicit remainder is:

$$f(\rho) = f(x) + (\rho - x)f'(x) + \frac{(\rho - x)^2}{2}f''(\theta),$$

for some $\theta \in (x, \rho)$. Since $f(\rho) = 0$, this simplifies to

$$\rho = x - \frac{f(x)}{f'(x)} - \frac{(\rho - x)^2}{2}\frac{f''(\theta)}{f'(x)}.$$

Substituting $f(t) = A - 1/t^2$, $f'(t) = 2/t^3$ and $f''(t) = -6/t^4$, it follows:

$$\rho = x + \frac{x}{2}(1 - Ax^2) + \frac{3}{2}\frac{x^3}{\theta^4}(\rho - x)^2,$$

which proves the Lemma. □

The implementation is as follows: (i) first compute an initial approximation x_0 of the reciprocal square root, using a few most significant bits of the input; (ii) then refine this approximation using Newton's iteration

$$x_{k+1} = x_k + \frac{x_k}{2}(1 - Ax_k^2), \tag{1}$$

until x_{k+1} is precise enough. Of course rounding errors occur in computing the right-hand side of Eq. (1); we have to take them into account, and to do that we have to explicit how this right-hand side is computed.

We first describe the recursive iteration, i.e., step (ii). We denote by A the number we want to compute the reciprocal square root of, and n is the precision we want (in bits) after the binary point. The number a is a truncation of A to h bits after the binary point, and x is an approximation of $a^{-1/2}$ to h bits.

Algorithm `ApproximateInverseSquareRoot`.
Input: $1 \leq a, A < 4$, $1/2 \leq x < 1$ with $x - \frac{1}{2}\cdot 2^{-h} \leq a^{-1/2} \leq x + 2^{-h}$, $a \leq A < a + 2^{-h}$
Output: X with $X - \frac{1}{2}\cdot 2^{-n} \leq A^{-1/2} \leq X + 2^{-n}$, where $n \leq 2h - 3$
$r \leftarrow x^2$ [exact]
$s \leftarrow Ar$ [exact]
$t \leftarrow 1 - s$ [rounded at weight 2^{-2h} towards $-\infty$]
$u \leftarrow xt$ [exact]
$X \leftarrow x + u/2$ [rounded at weight 2^{-n} to nearest]

Lemma 2. *If $h \geq 11$, the output X of Algorithm `ApproximateInverseSquareRoot` satisfies*

$$X - \frac{1}{2}\cdot 2^{-n} \leq A^{-1/2} \leq X + 2^{-n}. \tag{2}$$

Proof. Firstly, $a \leq A < a + 2^{-h}$ yields $a^{-1/2} - \frac{1}{2}\cdot 2^{-h} \leq A^{-1/2} \leq a^{-1/2}$, thus $x - 2^{-h} \leq A^{-1/2} \leq x + 2^{-h}$.

Lemma 1 implies that the value x' that would return Algorithm `ApproximateInverseSquareRoot` if there was no rounding error satisfies $0 \leq A^{-1/2} - x' = \frac{3}{2}\frac{x^3}{\theta^4}(A^{-1/2} - x)^2$. Since $\theta \in (x, A^{-1/2})$,

and $A^{-1/2} \le x + 2^{-h}$, we have $x \le \theta + 2^{-h}$, which yields $\frac{x^3}{\theta^3} \le (1 + \frac{2^{-h}}{\theta})^3 \le (1 + 2^{-10})^3 \le 1.003$ since $\theta \ge 1/2$ and $h \ge 11$. Thus $0 \le A^{-1/2} - x' \le 3.01 \cdot 2^{-2h}$.

Finally the errors while rounding $1 - s$ and $x + u/2$ in the algorithm yield $\frac{1}{2} \cdot 2^{-n} \le x' - X \le \frac{1}{2} \cdot 2^{-n} + \frac{1}{2} \cdot 2^{-2h}$, thus the final inequality is:

$$\frac{1}{2} \cdot 2^{-n} \le A^{-1/2} - X \le \frac{1}{2} \cdot 2^{-n} + 3.51 \cdot 2^{-2h}.$$

For $2h \ge n + 3$, we have $3.51 \cdot 2^{-2h} \le \frac{1}{2} \cdot 2^{-n}$, which concludes the proof. □

The initial approximation is obtained using a bipartite table for $h = 11$. More precisely, we split a 13-bit input $a = a_1 a_0 . a_{-1} \dots a_{-11}$ into three parts of 5, 4 and 4 bits respectively, say α, β, γ, and we deduce a 11-bit approximation $x = 0.x_{-1}x_{-2} \dots x_{-11}$ of the form $T_1[\alpha, \beta] + T_2[\alpha, \gamma]$, where both tables have 384 entries each. Those tables satisfy:

$$x + (\frac{1}{4} - \varepsilon)2^{-11} \le a^{-1/2} \le x + (\frac{1}{4} + \varepsilon)2^{-11},$$

with $\varepsilon \le 1.061$. This does not fulfill the initial condition of Alg. `ApproximateInverseSquareRoot`, since we have $x - 0.811 \cdot 2^{-h} \le a^{-1/2} \le x + 1.311 \cdot 2^{-h}$, which yields $X - \frac{1}{2} \cdot 2^{-n} \le A^{-1/2} \le X + 1.21 \cdot 2^{-n}$, thus the right bound is not a priori fulfilled. However the only problematic case is $n = 19$, which gives exactly $(n + 3)/2 = 11$, since for $12 \le n \le 18$, the error terms in 2^{-2h} are halved. An exhaustive search of all possible inputs for $h = 11$ and $n = 19$ gives

$$X - \frac{1}{2} \cdot 2^{-n} \le A^{-1/2} \le X + 0.998 \cdot 2^{-n},$$

the worst case being $A = 1990149, X = 269098$ (scaled by 2^{19}). Thus as soon as $n \ge 2$, Eq. (2) is fulfilled.

References

[1] Brent, R. P., and Zimmermann, P. *Modern Computer Arithmetic*. Version 0.1.1, 2006. In preparation. Current version available at `http://www.loria.fr/~zimmerma/mca/pub226.html`.

[2] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw. 33*, 2 (2007), article 13.

[3] *GNU MP: The GNU Multiple Precision Arithmetic Library*, 4.2.2 ed., 2007. `http://gmplib.org/`.

[4] Karp, A. H., and Markstein, P. High precision division and square root. HP Labs Report 93-93-42, Hewlett Packard, June 1993. Revised October 1994.

[5] Schulte, M. J., and Wires, K. E. High-speed inverse square roots. In *Proc. 14th Symp. Computer Arithmetic (ARITH14)* (1999), pp. 124–131.

[6] Zimmermann, P. Karatsuba square root. Research Report 3805, INRIA, 1999.