Fast Inverse Square Root

Bingyu Wang

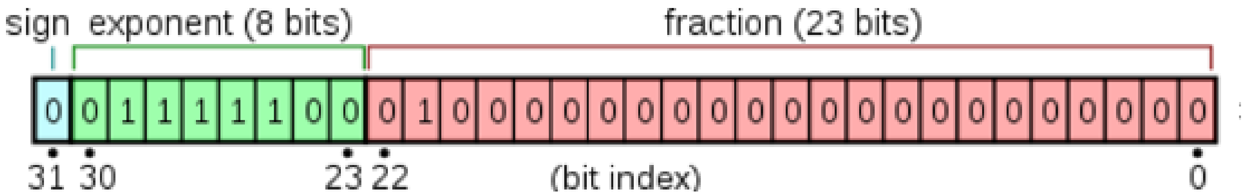
September 15, 2017

The goal is calcultae

$$y = \frac{1}{\sqrt{x}} \tag{1}$$

,where the denominator is an Euclidean norm of a vector. Sum of square is fast enough to calculate, but the main problem is to calculate the inverse square root(see equation (1)).

Single-precision floating-point format



which contains three part: $sign : bit(31); exponent : bit(23-30); fraction : bit(0-22)$. And its value can be written as[1]

$$\begin{aligned} x &= (-1)^{b_{31}} \times (1 + 0.b_{22}b_{21}\cdots b_0)_2 \times 2^{(b_{30}b_{29}\cdots b_{23})_2 - 127} & (2)\\ &= (-1)^{b_{31}} \times (1+f) \times 2^e & (3)\\ &= (1+f) \times 2^e & (4) \end{aligned}$$

since x is a norm, always positive and where:

$$\begin{aligned} f &= (0.b_{22}b_{21}\cdots b_0)_2 \\ &= \frac{(b_{22}b_{21}\cdots b_0)_2}{2^{23}} \\ &= \frac{F}{L} \end{aligned} \tag{5}$$

where F transform the fraction into integer, and L is a constant(2^{23}).

$$\begin{aligned} e &= (b_{30}b_{29}\cdots b_{23})_2 - 127 \\ &= E - B \end{aligned} \tag{6}$$

where E is the bits format for exponent 8-bits, and B is a constant(127).

Floating-point format to Integer-Format What if we transform the floating-point format(see in the figure) into integer bit using (5) and (6), which can be easily written as:

$$\begin{aligned} Integer(x) &= (b_{22}b_{21}\cdot b_0)_2 + (b_{30}b_{29}\cdot b_{23})_2 \times 2^{23} & (7)\\ &= F + EL & (8) \end{aligned}$$

[1] `https://en.wikipedia.org/wiki/Single-precision_floating-point_format`

First Step Approximation

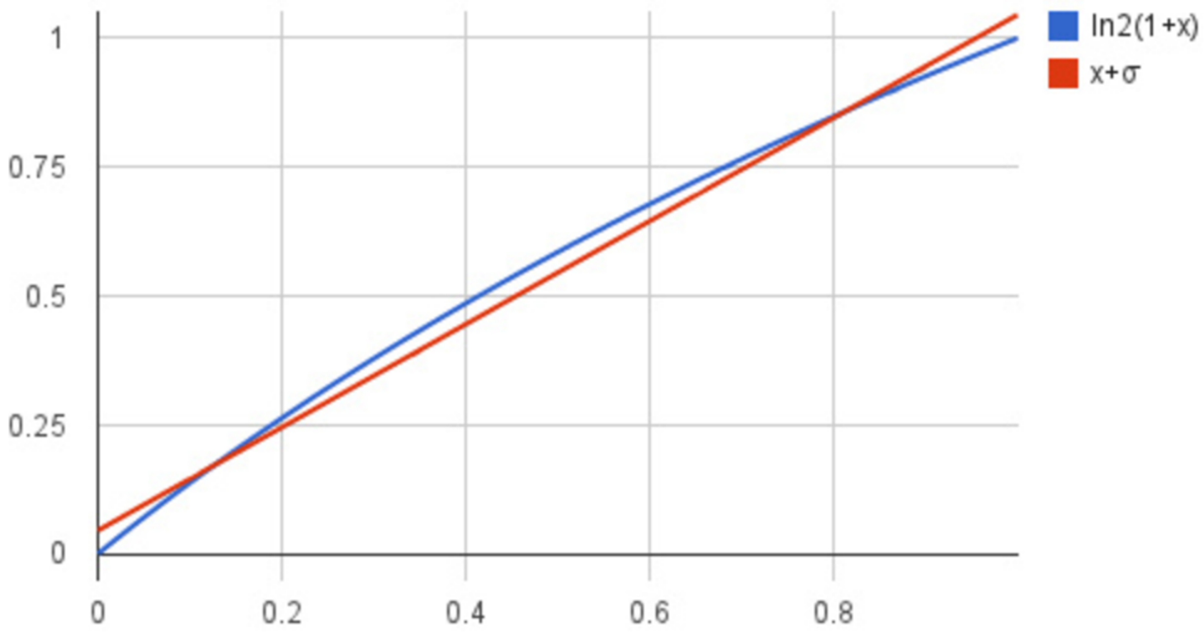
Take a log based on 2 for equation (1):

$$\log_2(y) = -\frac{1}{2}\log_2(x) \tag{9}$$

$$\Rightarrow \quad \log_2\left[(1+f_y)\times 2^{e_y}\right] = -\frac{1}{2}\log_2\left[(1+f_x)\times 2^{e_x}\right] \tag{10}$$

$$\Rightarrow \quad \log_2(1+f_y) + e_y = -\frac{1}{2}\left[\log_2(1+f_x) + e_x\right] \tag{11}$$

There is an approximation for $log_2(1+x)$ if $x \in [0,1)$, which is $x+\sigma$, where σ is pre-defined constant[2](see the following picture)



Therefore equation(11) can be further inferred:

$$f_y + \sigma + e_y \approx -\frac{1}{2}(f_x + \sigma + e_x) \tag{12}$$

$$\Rightarrow \frac{F_y}{L} + \sigma + E_y - B \approx -\frac{1}{2}\left(\frac{F_x}{L} + \sigma + E_x - B\right) \text{ using (5), (6)} \tag{13}$$

$$\Rightarrow \frac{3}{2}L(\sigma - B) + F_y + E_y L \approx -\frac{1}{2}(F_x + E_x L) \tag{14}$$

$$\Rightarrow \frac{3}{2}L(\sigma - B) + Integer(y) \approx -\frac{1}{2}Integer(x) \text{ using (8)} \tag{15}$$

$$\Rightarrow Integer(y) \approx -\frac{1}{2}Integer(x) + \text{magic-number} \tag{16}$$

where magic-number is $-\frac{3}{2}L(\sigma - B)$.

In the algorithm, step

$$i = *(long*)\&y;$$

is trying to transform floating into integer format and then (16) is corresponding to the algorithm step:

$$i = \texttt{0x5F3759DF} - (i >> 1); \tag{17}$$

where $i >> 1$ is divided by 2.

[2]`https://en.wikipedia.org/wiki/Fast_inverse_square_root`

Second Step Approximation So far, the first step approximation already did a pretty good job, but there is a way we could improve it even further, which is using Newton method. (1) can be written as function of y:

$$f(y) = \frac{1}{y^2} - x \tag{18}$$

easily to get the first derivation is:

$$f'(y) = -\frac{2}{y^3} \tag{19}$$

According to Newton method

$$\begin{aligned} y_{n+1} &= y_n - \frac{f(y_n)}{f'(y_n)} && (20) \\ &= y_n + \frac{1}{2}y_n - \frac{1}{2}xy_n^3 && (21) \\ &= \frac{3}{2}y_n - \frac{1}{2}xy_n^3 && (22) \end{aligned}$$

where y_n is the first step approximation in equation (16), and x is the original input x, which explains the last step in the algorithm:

$$y_{new} = y_{old} * (\frac{3}{2} - \frac{x}{2} * y_{old}^2); //\text{Newton iteration}$$