Systemy Dedykowane w Układach Programowalnych

Temat: Szybka odwrotność pierwiastka kwadratowego

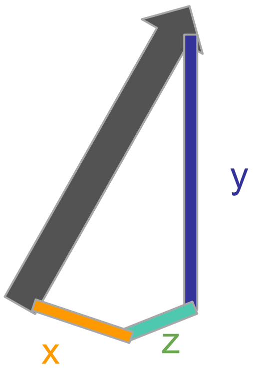
Moudjo Oscar DAGA

Piotr Słonka

CELE I ZAŁOŻENIA

Szybka odwrotność pierwiastka kwadratowego Jest to metoda obliczania x^-1/2, odnosząca się do przekształceń z 32-bitowej liczby zmiennoprzecinkowej w standardzie IEEE 754. Największą zaletą tego algorytmu jest uniknięcie kosztownych obliczeniowo operacji zmiennoprzecinkowych na korzyść operacji na liczbach całkowitych.

W celach normalizacji np. wektorów.

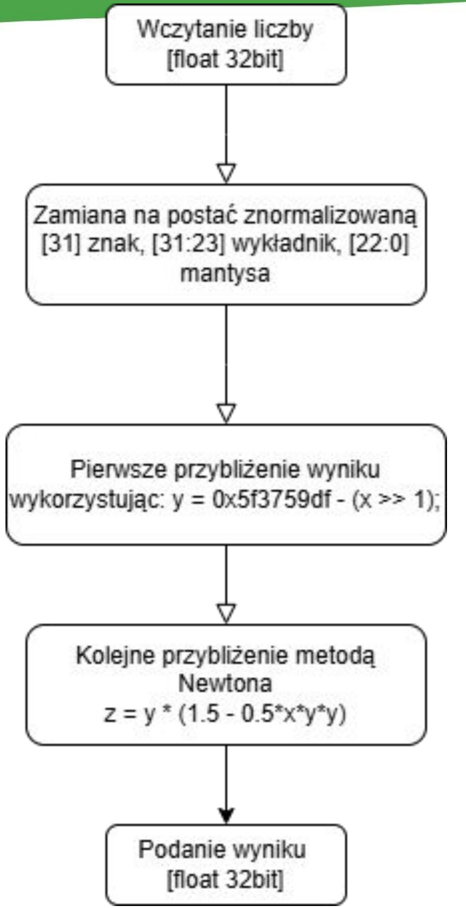


Długość wektora wyraża się wzorem sqrt(x^2+y^2+z^2). Takie równanie procesor policzy bardzo szybko. Jednakże jeżeli chcielibyśmy unormowaną formę wektora (żeby jego długość wynosiła 1), to współrzędne wektora również powinny być odpowiednio unormowane. W takim wypadku wartości każdego z nich wynosiłyby:

$$x_1 = x \cdot \frac{1}{\sqrt{x^2+y^2+z^2}} \qquad y_1 = y \cdot \frac{1}{\sqrt{x^2+y^2+z^2}} \qquad z_1 = z \cdot \frac{1}{\sqrt{x^2+y^2+z^2}}$$

A takie działanie nasz procesor wykona w znacznie dłuższym czasie. Stąd pomysł na ten algorytm - żeby przyspieszyć to działanie.

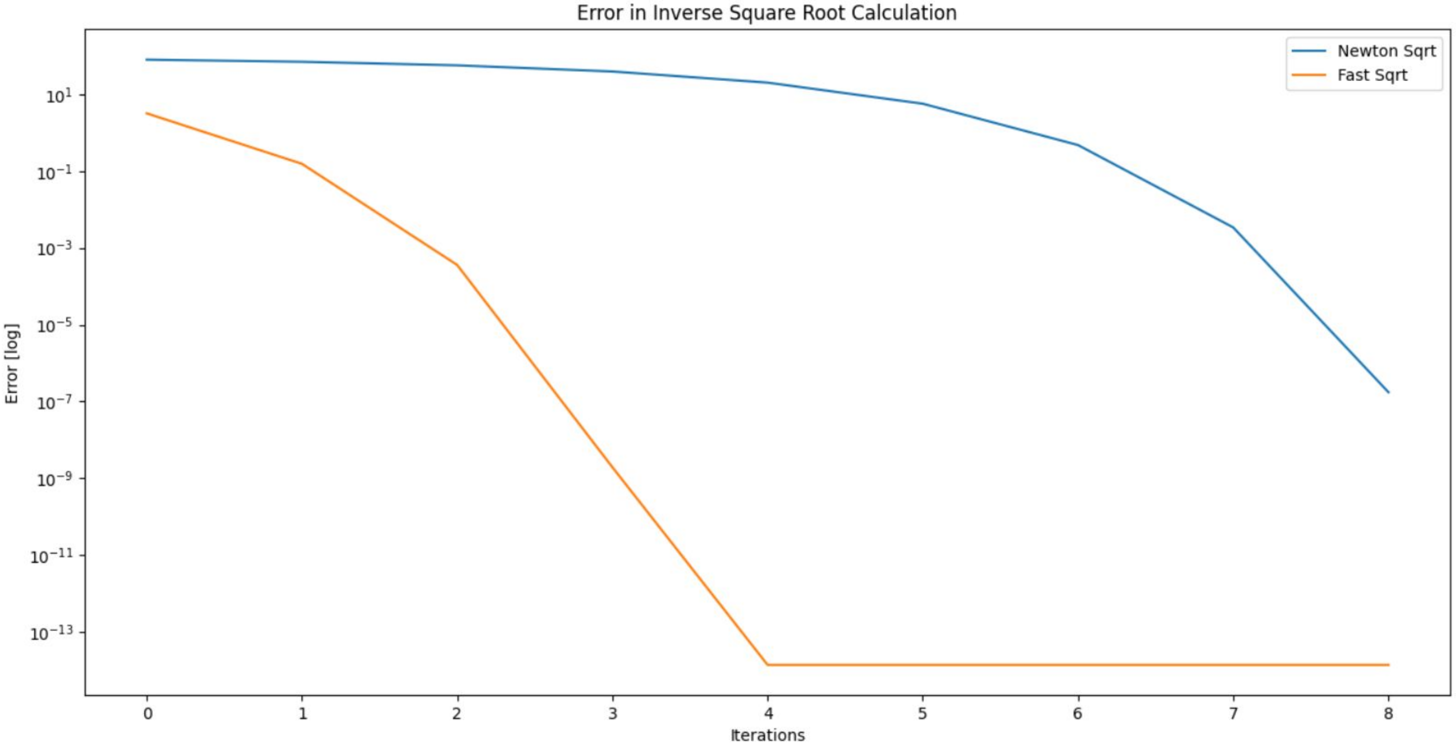
```c
float Q_rsqrt( float number )
{
	long i;
	float x2, y;
	const float threehalfs = 1.5F;

	x2 = number * 0.5F;
	y  = number;
	i  = * ( long * ) &y;                       // evil floating point bit level hacking
	i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
	y  = * ( float * ) &i;
	y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//	y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

	return y;
}
```

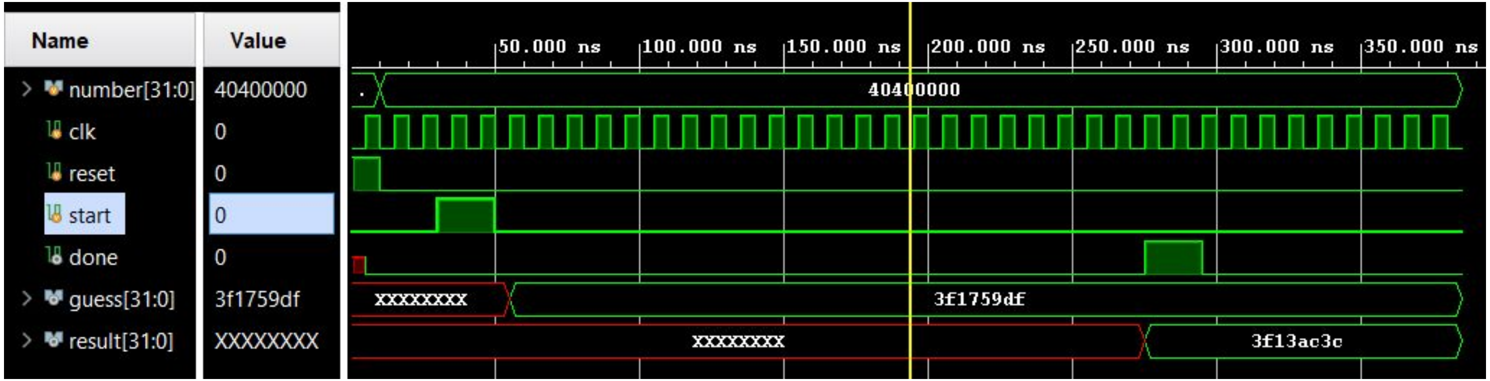

Wczytanie liczby [float 32bit]
Zamiana na postać znormalizowaną [31] znak, [31:23] wykładnik, [22:0] mantysa
Pierwsze przybliżenie wyniku wykorzystując: y = 0x5f3759df - (x >> 1);
Kolejne przybliżenie metodą Newtona z = y * (1.5 - 0.5*x*y*y)
Podanie wyniku [float 32bit]

Prototyp algorytmu w python

```python
def fast_isqrt(number, iterations):
    threehalfs = 1.5
    x2 = number * 0.5
    y = number

    packed_y = struct.pack('f', y)
    i = struct.unpack('i', packed_y)[0]  # treat float's bytes as int
    i = 0x5f3759df - (i >> 1)            # arithmetic with magic number
    packed_i = struct.pack('i', i)
    y = struct.unpack('f', packed_i)[0]  # treat int's bytes as float

    for _ in range(iterations):
        y = y * (threehalfs - (x2 * y * y))  # Newton's method

    return y
```

Porównanie z klasycznym algorytmem newtona 1/sqrt(x), x = 25





Po symulacji mamy 3f13ac3c
czy szybka odwrotność pierwiastka kwadratowego 40400000 jest 3f13ac3c ?

Sprawdzimy : 3f13ac3c (hexa) = 0011 1111 0001 0011 1010 1100 0011 1100 (bin)

z użycia formatu zmiennoprzecinkowego pojedynczej precyzji $\text{Value} = (-1)^{\text{Sign}} \times (1 + \text{Mantissa}) \times 2^{\text{Exponent}}$

mamy :

- Sign = `0` (positive)
- Mantissa = `1.14990234375`
- Exponent = `-1`

czyli mamy : 3F13AC3C (hexa) = 0,574951171875 (decimal)

A oczekiwany wynik to :

40 400 000 (hex) = 0100 0000 0100 0000 0000 0000 0000 0000 (bin)

z użycia formatu zmiennoprzecinkowego pojedynczej precyzji

- Bit sign (S): ‘ 0 ‘ (positif)
- Exposant (E): ‘ 10000000 ‘ (128 decimal)
- Mantisse (M): ‘10000000000000000000000 ’

real exposant : Exposant = 128 - 127 =1

Mantysa jest normalizowana z dorozumianą liczbą „1” na początku, więc przyjmuje postać: 1.10000000000000000000000 = 1,5 (decimal)

wartość reprezentowana przez 40400000 to: $1,5 \times 2^1 = 3,0$

$\frac{1}{\sqrt{x}}$ **z x = 3 $\Rightarrow$ 0,5773502692**

Szybki odwrotny pierwiastek kwadratowy z „40400000” (który odpowiada wartości 3) wynosi w przybliżeniu **0,5773502692 = 0,57**

Obliczony błąd: 100 * (0,57735 - 0,57495) / 0,57735 = 0,415%

Resources

https://pl.wikipedia.org/wiki/Szybka_odwrotno%C5%9B%C4%87_pierwiastka_kwadratowego

https://pl.wikipedia.org/wiki/Pierwiastek_kwadratowy

https://www.researchgate.net/publication/378163213_Fast_Inverse_Square_Root_using_FPGA

https://www.dline.info/ed/fulltext/v3n1/4.pdf

Link do kodu źródłowego:
https://drive.google.com/file/d/1WQxHiLcrK0n3PLvvzi07iSsR8z4G3QC8/view?usp=sharing

Dziękuję …