Article

A Modification of the Fast Inverse Square Root Algorithm

Cezary J. Walczyk [1], Leonid V. Moroz [2] and Jan L. Cieśliński [1,*]

[1] Uniwersytet w Białymstoku, Wydział Fizyki, ul. Ciołkowskiego 1L, 15-245 Białystok, Poland; c.walczyk@uwb.edu.pl, j.cieslinski@uwb.edu.pl

[2] Lviv Polytechnic National University, Department of Security Information and Technology, st. Kn. Romana 1/3, 79000 Lviv, Ukraine; moroz_lv@polynet.lviv.ua

Abstract: We present an improved algorithm for fast calculation of the inverse square root for single-precision floating-point numbers. The algorithm is much more accurate than the famous fast inverse square root algorithm and has a similar computational cost. The presented modification concern Newton-Raphson corrections and can be applied when the distribution of these corrections is not symmetric (for instance, in our case they are always negative).

Keywords: floating-point arithmetic; inverse square root; magic constant; Newton-Raphson method

1. Introduction

Floating-point arithmetic has became widely used in many applications such as 3D graphics, scientific computing and signal processing [1–5], implemented both in hardware and software [6–10]. Many algorithms can be used to approximate elementary functions [1,2,10–18]. The inverse square root function is of particular importance because it is widely used in 3D computer graphics, especially in lightning reflections [19–21], and has many other applications, see [22–36]. All of these algorithms require an initial seed to start the approximation. The more accurate is the initial seed, the fewer iterations are needed. Usually, the initial seed is obtained from a look-up table (LUT) which is memory consuming.

In this paper we consider an algorithm for computing the inverse square root using the so called magic constant instead of a LUT [37–40]. The following code realizes the fast inverse square root algorithm in the case of single-precision IEEE Standard 754 floating-point numbers (type **float**).

```
1.  float InvSqrt(float x){
2.        float halfnumber = 0.5f*x;
3.        int i = *(int*) &x;
4.        i = R - (i>>1);
5.        y = *(float*) &i;
6.        y = y*(1.5f - halfnumber*y*y);
7.        y = y*(1.5f - halfnumber*y*y);
8.        return y ;
9.  }
```

The code *InvSqrt* consists of two main parts. Lines *4* and *5* produce in a very cheap way a quite good zeroth approximation of the inverse square root of a given positive floating-point number x. Lines *6* and *7* apply the Newton-Raphson corrections twice (often a version with just one iteration is used, as well). Originally R was proposed as $0x5F3759DF$, see [37,38].

InvSqrt is characterized by a high speed, more that 3 times higher than in computing the inverse square root using library functions. This property is discussed in detail in [41]. The errors of the fast inverse square root algorithm depend on the choice of the "magic constant" R. In several theoretical papers [38,41–44] (see also Eberly's monograph [19]) attempts were made to determine analytically the optimal value of the magic constant (i.e., to minimize errors). In general, this optimal value can depend on the number of iterations, which is a general phenomenon [45]. The derivation and comprehensive mathematical description of all steps of the fast inverse square root algorithm is given in our recent paper [46]. We found the optimum value of the magic constant by minimizing the final maximum relative error.

In the present paper we develop our analytical approach to construct an improved algorithm (*InvSqrt1*) for fast computing of the inverse square root, see section 4. In both codes, *InvSqrt* and *InvSqrt1*, magic constants serve as a low-cost way of generating a reasonably accurate first approximation of the inverse square root. These magic constants turn out to be the same. The main novelty of the new algorithm is in the second part of the code which is changed significantly. In fact, we propose a modification of the Newton-Raphson formulae which has a similar computational cost but improve the accuracy even by several times.

2. Analytical approach to algorithm *InvSqrt*

In this paper we confine ourselves to positive floating-point numbers

$$x = (1 + m_x)2^{e_x} \tag{2.1}$$

where $m_x \in [0, 1)$ and e_x is an integer (note that this formula does not hold for subnormal numbers). In the case of the IEEE-754 standard, a floating-point number is encoded by 32 bits. The first bit corresponds to a sign (in our case this bit is simply equal to zero), the next 8 bits correspond to an exponent e_x and the last 23 bits encodes a mantissa m_x. The integer encoded by these 32 bits, denoted by I_x, is given by

$$I_x = N_m(B + e_x + m_x) \tag{2.2}$$

where $N_m = 2^{23}$ and $B = 127$ (thus $B + e_x = 1, 2, \ldots, 254$). The lines *3* and *5* of the *InvSqrt* code interprete a number as an integer (2.2) or float (2.1), respectively. The lines *4*, *6* and *7* of the code can be written as

$$I_{y_0} = R - \lfloor I_x/2 \rfloor, \qquad y_1 = \tfrac{1}{2}y_0(3 - y_0^2 x), \qquad y_2 = \tfrac{1}{2}y_1(3 - y_1^2 x). \tag{2.3}$$

The first equation produces, in a surprisingly simple way, a good zeroth approximation y_0 of the inverse square root $y = 1/\sqrt{x}$. Of course, this needs a very special form of R. In particular, in the single precision case we have $e_R = 63$, see [46]. The next equations can be easily recognized as the Newton-Raphson corrections. We point out that the code *InvSqrt* is invariant with respect to the scaling

$$x \to \tilde{x} = 2^{-2n}x, \qquad y_k \to \tilde{y}_k = 2^n y_k \qquad (k = 0, 1, 2), \tag{2.4}$$

like the equality $y = 1/\sqrt{x}$ itself. Therefore, without loss of the generality, we can confine our analysis to the interval

$$\tilde{A} := [1, 4). \tag{2.5}$$

The tilde will denote quantities defined on this interval. In [46] we have shown that the function $\tilde{y}_0$ defined by the first equation of (2.3) can be approximated with a very good accuracy by the piece-wise linear function $\tilde{y}_{00}$ given by

Peer-reviewed version available at Computation 2019, 7, 41; doi:10.3390/computation7030041

$$\tilde{y}_{00}(\tilde{x},t)=\begin{cases}-\frac{1}{4}\tilde{x}+\frac{3}{4}+\frac{1}{8}t & \text{for } \tilde{x}\in[1,2)\\ -\frac{1}{8}\tilde{x}+\frac{1}{2}+\frac{1}{8}t & \text{for } \tilde{x}\in[2,t)\\ -\frac{1}{16}\tilde{x}+\frac{1}{2}+\frac{1}{16}t & \text{for } \tilde{x}\in[t,4)\end{cases} \tag{2.6}$$

where

$$t=2+4m_R+2N_m^{-1}, \tag{2.7}$$

and $m_R:=N_m^{-1}R-\lfloor N_m^{-1}R\rfloor$ (m_R is the mantissa of the floating-point number corresponding to R). Note that the parameter t, defined by (2.7), is uniquely determined by R.

The only difference between y_0 produced by the code *InvSqrt* and y_{00} given by (2.6) is the definition of t, because t related to the code depends (although in a negligible way) on x. Namely,

$$|\tilde{y}_{00}-\tilde{y}_0|\leqslant\frac{1}{4}N_m^{-1}=2^{-25}\approx 2.98\cdot 10^{-8}. \tag{2.8}$$

Taking into account the invariance (2.4), we obtain

$$\left|\frac{y_{00}-y_0}{y_0}\right|\leqslant 2^{-24}\approx 5.96\cdot 10^{-8}. \tag{2.9}$$

These estimates do not depend on t (in other words, they do not depend on R). The relative error of the zeroth approximation (2.6) is given by

$$\tilde{\delta}_0(\tilde{x},t)=\sqrt{\tilde{x}}\,\tilde{y}_{00}(\tilde{x},t)-1 \tag{2.10}$$

This is a continuous function with local maxima at

$$\tilde{x}_0^I=(6+t)/6,\qquad \tilde{x}_0^{II}=(4+t)/3,\qquad \tilde{x}_0^{III}=(8+t)/3, \tag{2.11}$$

given respectively by

$$\begin{aligned}\tilde{\delta}_0(\tilde{x}_0^I,t)&=-1+\frac{1}{2}\left(1+\frac{t}{6}\right)^{3/2},\\ \tilde{\delta}_0(\tilde{x}_0^{II},t)&=-1+2\left(\frac{1}{3}\left(1+\frac{t}{4}\right)\right)^{3/2},\\ \tilde{\delta}_0(\tilde{x}_0^{III},t)&=-1+\left(\frac{2}{3}\left(1+\frac{t}{8}\right)\right)^{3/2}.\end{aligned} \tag{2.12}$$

In order to study global extrema of $\tilde{\delta}_0(\tilde{x},t)$ we need also boundary values:

$$\tilde{\delta}_0(1,t)=\tilde{\delta}_0(4,t)=\frac{1}{8}(t-4),\qquad \tilde{\delta}_0(2,t)=\frac{\sqrt{2}}{4}\left(1+\frac{t}{2}\right)-1,\qquad \tilde{\delta}_0(t,t)=\frac{\sqrt{t}}{2}-1, \tag{2.13}$$

which are, in fact, local minima. Taking into account

$$\tilde{\delta}_0(1,t)-\tilde{\delta}_0(t,t)=\frac{1}{8}\left(\sqrt{t}-2\right)^2\geqslant 0,\qquad \tilde{\delta}_0(2,t)-\tilde{\delta}_0(t,t)=\frac{\sqrt{2}}{8}\left(\sqrt{t}-\sqrt{2}\right)^2\geqslant 0, \tag{2.14}$$

we conclude that

$$\min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_0(\tilde{x},t) = \tilde{\delta}_0(t,t) < 0. \tag{2.15}$$

Because $\tilde{\delta}_0(\tilde{x}_0^{III},t) < 0$ for $t \in (2,4)$, the global maximum is one of the remaining local maxima:

$$\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_0(\tilde{x},t) = \max\{\tilde{\delta}_0(\tilde{x}_0^{I},t), \tilde{\delta}_0(\tilde{x}_0^{II},t)\}. \tag{2.16}$$

Therefore,

$$\max_{x\in\tilde{A}} |\tilde{\delta}_0(\tilde{x},t)| = \max\{|\tilde{\delta}_0(t,t)|, \tilde{\delta}_0(\tilde{x}_0^{I},t), \tilde{\delta}_0(\tilde{x}_0^{II},t)\}. \tag{2.17}$$

In order to minimize this value with respect to t, i.e., to find t_0^r such that

$$\max_{x\in\tilde{A}} |\tilde{\delta}_0(\tilde{x},t_0^r)| < \max_{x\in\tilde{A}} |\tilde{\delta}_0(\tilde{x},t)| \qquad \text{for } t \neq t_0^r, \tag{2.18}$$

we observe that $|\tilde{\delta}_0(t,t)|$ is a decreasing function of t, while both maxima ($\tilde{\delta}_0(\tilde{x}_0^{I},t)$ and $\tilde{\delta}_0(\tilde{x}_0^{II},t)$) are increasing functions. Therefore, it is sufficient to find $t = t_0^I$ and $t = t_0^{II}$ such that

$$|\tilde{\delta}_0(t_0^I,t_0^I)| = \tilde{\delta}_0(\tilde{x}_0^I,t_0^I)\,, \qquad |\tilde{\delta}_0(t_0^{II},t_0^{II})| = \tilde{\delta}_0(\tilde{x}_0^{II},t_0^{II}), \tag{2.19}$$

and to choose the greater of these two values. In [46] we have shown that

$$|\tilde{\delta}_0(t_0^I,t_0^I)| < |\tilde{\delta}_0(t_0^{II},t_0^{II})|. \tag{2.20}$$

Therefore $t_0^r = t_0^{II}$ and

$$\tilde{\delta}_{0\,\max} := \min_{t\in(2,4)} \left(\max_{x\in\tilde{A}} |\tilde{\delta}_0(\tilde{x},t)| \right) = |\tilde{\delta}_0(t_0^r,t_0^r)|. \tag{2.21}$$

The following numerical values result from these calculations [46]:

$$t_0^r \approx 3.7309796, \qquad R_0 = 0x5F37642F, \qquad \tilde{\delta}_{0\,\max} \approx 0.03421281. \tag{2.22}$$

Newton-Raphson corrections for the zeroth approximation ($\tilde{y}_{00}$) will be denoted by $\tilde{y}_{0k}$ ($k = 1,2,\ldots$). In particular, we have:

$$\begin{aligned} \tilde{y}_{01}(\tilde{x},t) &= \tfrac{1}{2}\tilde{y}_{00}(\tilde{x},t)(3 - \tilde{y}_{00}^2(\tilde{x},t)\,\tilde{x}), \\ \tilde{y}_{02}(\tilde{x},t) &= \tfrac{1}{2}\tilde{y}_{01}(\tilde{x},t)(3 - \tilde{y}_{01}^2(\tilde{x},t)\,\tilde{x}). \end{aligned} \tag{2.23}$$

and the corresponding relative error functions will be denoted by $\tilde{\delta}_k(\tilde{x},t)$:

$$\tilde{\delta}_k(\tilde{x},t) := \frac{\tilde{y}_{0k}(\tilde{x},t) - \tilde{y}}{\tilde{y}} = \sqrt{\tilde{x}}\tilde{y}_{0k}(\tilde{x},t) - 1, \qquad (k = 0,1,2,\ldots), \tag{2.24}$$

where we included also the case $k = 0$, see (2.10). The obtained approximations of the inverse square root depend on the parameter t directly related to the magic constant R. The value of this parameter can be estimated by analysing the relative error of $\tilde{y}_{0k}(\tilde{x},t)$ with respect to $1/\sqrt{\tilde{x}}$. As the best estimation we consider $t = t_k^{(r)}$ minimizing the relative error $\tilde{\delta}_k(\tilde{x},t)$:

$$\forall_{t\neq t_k^{(r)}}\left(\tilde{\delta}_{k\max}\equiv\max_{\tilde{x}\in\tilde{A}}|\tilde{\delta}_k(\tilde{x},t_k^{(r)})|<\max_{\tilde{x}\in\tilde{A}}|\tilde{\delta}_k(\tilde{x},t)|\right). \tag{2.25}$$

We point out that in general the optimum value of the magic constant can depend on the number of Newton-Raphson corrections. Calculations carried out in [46] gave the following results:

$$\begin{aligned} &t_1^r=t_2^r=3.7298003, \qquad R_1^r=R_2^r=0x5F375A86,\\ &\tilde{\delta}_{1\max}\approx 1.75118\cdot 10^{-3}, \quad \tilde{\delta}_{2\max}\approx 4.60\cdot 10^{-6}. \end{aligned} \tag{2.26}$$

We omit details of the computations except one important point. Using (2.24) for expressing $\tilde{y}_{0k}$ by $\tilde{\delta}_k$ and $\sqrt{\tilde{x}}$ we can rewrite (2.23) as

$$\tilde{\delta}_k(\tilde{x},t)=-\frac{1}{2}\tilde{\delta}_{k-1}^2(\tilde{x},t)(3+\tilde{\delta}_{k-1}(\tilde{x},t)), \quad (k=1,2,\ldots). \tag{2.27}$$

The quadratic dependence on $\tilde{\delta}_{k-1}$ means that every Newton-Raphson correction improves the accuracy by several orders of magnitude (until the machine precision is reached), compare (2.26).

The formula (2.27) suggests another way of improving the accuracy because the functions $\tilde{\delta}_k$ are always non-positive for any $k\geqslant 1$. Roughly saying, we are going to shift the graph of $\tilde{\delta}_k$ upwards by an appropriate modification of the Newton-Raphson formula. In the next section we describe the general idea of this modification.

3. Modified Newton-Raphson formulas

The formula (2.27) shows that errors introduced by Newton-Raphson corrections are nonpositive, i.e., they take values in intervals $[-\tilde{\delta}_{k\max}, 0]$, where $k=1,2,\ldots$. Therefore, it is natural to introduce a correction term into the Newton-Raphson formulas (2.23). We expect that the corrections will be roughly half of the maximal relative error. Instead of the maximal error we introduce two parameters, d_1 and d_2. Thus we get modified Newton-Raphson formulas:

$$\begin{aligned} \tilde{y}_{11}(\tilde{x},t,d_1)&=2^{-1}\tilde{y}_{00}(\tilde{x},t)(3-\tilde{y}_{00}^2(\tilde{x},t)\,\tilde{x})+\frac{d_1}{2\sqrt{\tilde{x}}},\\ \tilde{y}_{12}(\tilde{x},t,d_1,d_2)&=2^{-1}\tilde{y}_{11}(\tilde{x},t,d_1)(3-\tilde{y}_{11}^2(\tilde{x},t,d_1)\,\tilde{x})+\frac{d_2}{2\sqrt{\tilde{x}}}, \end{aligned} \tag{3.1}$$

where zeroth approximation is assumed in the form (2.6). In the following section the term $1/\sqrt{\tilde{x}}$ will be replaced by some approximations of $\tilde{y}$, tranforming (3.1) into a computer code. In order to estimate a possible gain in accuracy, in this section we temporarily assume that $\tilde{y}$ is the exact value of the inverse square root. The corresponding error functions,

$$\tilde{\delta}_k''(\tilde{x},t,d_1,\ldots,d_k)=\sqrt{\tilde{x}}\,\tilde{y}_{1k}(\tilde{x},t,d_1,\ldots,d_k)-1, \qquad k\in\{0,1,2,\ldots\}, \tag{3.2}$$

(where $\tilde{y}_{10}(\tilde{x},t):=\tilde{y}_{00}(\tilde{x},t)$), satisfy

$$\tilde{\delta}_k''=-\frac{1}{2}\tilde{\delta}_{k-1}''^2(3+\tilde{\delta}_{k-1}'')+\frac{d_k}{2}, \tag{3.3}$$

where: $\tilde{\delta}_0''(\tilde{x},t)=\tilde{\delta}_0(\tilde{x},t)$. Note that

$$\tilde{\delta}_1''(\tilde{x},t,d_1)=\tilde{\delta}_1(\tilde{x},t)+\frac{1}{2}d_1. \tag{3.4}$$

In order to simplify notation we usually will supress the explicit dependence on d_j. We will write, for instance, $\tilde{\delta}_2''(\tilde{x},t)$ instead of $\tilde{\delta}_2''(\tilde{x},t,d_1,d_2)$.

The corrections of the form (3.1) will decrease relative errors in comparison with the results of earlier papers [38,46]. We have 3 free parameters (d_1, d_2 and t) to be determined by minimizing the maximal error (in principle the new parameterization can give a new estimation of the parameter t). By analogy to (2.25), we are going to find $t = t^{(0)}$ minimizing the error of the first correction (2.25):

$$\forall_{t\neq t^{(0)}} \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1''(\tilde{x},t^{(0)})| < \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1''(\tilde{x},t)|, \tag{3.5}$$

where, as usual, $\tilde{A} = [1,4]$.

The first of Eqs. (3.3) implies that for any t the maximal value of $\tilde{\delta}_1''(\tilde{x},t)$ equals $\frac{1}{2}d_1$ and is attained at zeros of $\tilde{\delta}_0''(\tilde{x},t)$. Using results of section 2, including (2.15), (2.16), (2.20) and (2.21), we conclude that the minimum value of $\tilde{\delta}_1''(\tilde{x},t)$ is attained either for $\tilde{x} = t$ or for $\tilde{x} = x_0^{II}$ (where there is the second maximum of $\tilde{\delta}_0''(\tilde{x},t)$), i.e.,

$$\min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t) = \min\left\{\tilde{\delta}_1''(t,t), \tilde{\delta}_1''(x_0^{II},t)\right\} \tag{3.6}$$

Minimization of $|\tilde{\delta}_1''(\tilde{x},t)|$ can be done with respect to t and with respect to d_1 (these operations obviously commute), which corresponds to

$$\underbrace{\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t^{(0)})}_{\tilde{\delta}_{1\max}''} = -\min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t^{(0)}). \tag{3.7}$$

Taking into account

$$\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t^{(0)}) = \frac{d_1}{2}, \quad \min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_1''(\tilde{x},t^{(0)}) = \tilde{\delta}_1''(t^{(0)},t^{(0)}) = -\tilde{\delta}_{1\max} + \frac{d_1}{2}, \tag{3.8}$$

we get from (3.7):

$$\tilde{\delta}_{1\max}'' = \frac{1}{2}d_1 = \frac{1}{2}\tilde{\delta}_{1\max} \simeq 8.7559\cdot 10^{-4}, \tag{3.9}$$

where

$$\tilde{\delta}_{1\max} := \min_{t\in(2,4)} \left(\max_{x\in\tilde{A}} |\tilde{\delta}_1(\tilde{x},t)|\right). \tag{3.10}$$

and the numerical value of $\tilde{\delta}_{1\max}$ is given by (2.26). These conditions are satisfied for

$$t^{(0)} = t_1^{(r)} \simeq 3.7298003. \tag{3.11}$$

In order to minimize the relative error of the second correction we use equation analogous to (3.7):

$$\underbrace{\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_2''(\tilde{x},t^{(0)})}_{\tilde{\delta}_{2\max}''} = -\min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_2''(\tilde{x},t^{(0)}), \tag{3.12}$$

where from (3.3) we have

$$\max_{\tilde{x}\in\tilde{A}} \tilde{\delta}_2''(\tilde{x},t^{(0)}) = \frac{d_2}{2}, \quad \min_{\tilde{x}\in\tilde{A}} \tilde{\delta}_2''(\tilde{x},t^{(0)}) = -\frac{1}{2}\tilde{\delta}_{1\max}''^2\left(3+\tilde{\delta}_{1\max}''\right)+\frac{d_2}{2}. \tag{3.13}$$

Hence

$$\tilde{\delta}_{2\max}'' = \frac{1}{4}\tilde{\delta}_{1\max}''^2\left(3+\tilde{\delta}_{1\max}''\right). \tag{3.14}$$

Expressing this result in terms of formerly computed $\tilde{\delta}_{1\max}$ and $\tilde{\delta}_{2\max}$, we obtain

$$\tilde{\delta}_{2\max}'' = \frac{1}{8}\tilde{\delta}_{2\max} + \frac{3}{32}\tilde{\delta}_{1\max}^3 \simeq 5.75164\cdot 10^{-7} \simeq \frac{\tilde{\delta}_{2\max}}{7.99}, \tag{3.15}$$

where

$$\tilde{\delta}_{2\max} = \frac{1}{2}\tilde{\delta}_{1\max}^2(3-\tilde{\delta}_{1\max}).$$

Therefore, the above modification of Newton-Raphson formulas decreases the relative error 2 times after one iteration and almost 8 times after two iterations as compared to the standard *InvSqrt* algorithm.

In order to implement this idea in the form of a computer code, we have to replace the unknown $1/\sqrt{\tilde{x}}$ (i.e., $\tilde{y}$) on the right-hand sides of (3.1) by some numerical approximations.

4. New algorithm of higher accuracy

Approximating $1/\sqrt{\tilde{x}}$ in formulas (3.1) by values at left hand sides, we transform (3.1) into

$$\begin{aligned}\tilde{y}_{21} &= \frac{1}{2}\tilde{y}_{20}(3-\tilde{y}_{20}^2\,\tilde{x}) + \frac{1}{2}d_1\tilde{y}_{21},\\ \tilde{y}_{22} &= \frac{1}{2}\tilde{y}_{21}(3-\tilde{y}_{21}^2\,\tilde{x}) + \frac{1}{2}d_2\tilde{y}_{22},\end{aligned} \tag{4.1}$$

where $\tilde{y}_{2k}$ $(k=1,2,\ldots)$ depend on $\tilde{x}, t$ and d_j (for $1\leqslant j\leqslant k$). We assume $\tilde{y}_{20}\equiv\tilde{y}_{00}$, i.e., the zeroth approximation is still given by (2.6). We can see that $\tilde{y}_{21}$ and $\tilde{y}_{22}$ can be explicitly expressed by $\tilde{y}_{20}$ and $\tilde{y}_{21}$, respectively.

Parameters d_1 and d_2 have to be determined by minimization of the maximum error. We define error functions in the usual way:

$$\Delta_k^{(1)} = \frac{\tilde{y}_{2k}-\tilde{y}}{\tilde{y}} = \sqrt{\tilde{x}}\,\tilde{y}_{2k} - 1\,. \tag{4.2}$$

Substituting (4.2) into (4.1) we get:

$$\Delta_1^{(1)}(\tilde{x},t,d_1) = \frac{d_1}{2-d_1} - \frac{1}{2-d_1}\tilde{\delta}_0^2(\tilde{x},t)(3+\tilde{\delta}_0(\tilde{x},t)) = \frac{d_1+2\tilde{\delta}_1(\tilde{x},t)}{2-d_1}, \tag{4.3}$$

$$\Delta_2^{(1)}(\tilde{x},t,d,d_2) = \frac{d_2}{2-d_2} - \frac{1}{2-d_2}\left(\Delta_1^{(1)}(\tilde{x},t,d_1)\right)^2\left(3+\Delta_1^{(1)}(\tilde{x},t,d_1)\right). \tag{4.4}$$

The equation (4.3) expresses $\Delta_1^{(1)}(\tilde{x},t,d_1)$ as a linear function of the nonpositive function $\tilde{\delta}_1(\tilde{x},t)$ with coefficients depending on the parameter d_1. The optimum parameters t and d_1 will be estimated by the procedure described in section 3. First, we minimize the amplitude of the relative error function, i.e., we find $t^{(1)}$ such that

$$\max_{\tilde{x}\in\tilde{A}}\Delta_1^{(1)}(\tilde{x},t^{(1)}) - \min_{\tilde{x}\in\tilde{A}}\Delta_1^{(1)}(\tilde{x},t^{(1)}) \leqslant \max_{\tilde{x}\in\tilde{A}}\Delta_1^{(1)}(\tilde{x},t) - \min_{\tilde{x}\in\tilde{A}}\Delta_1^{(1)}(\tilde{x},t) \tag{4.5}$$

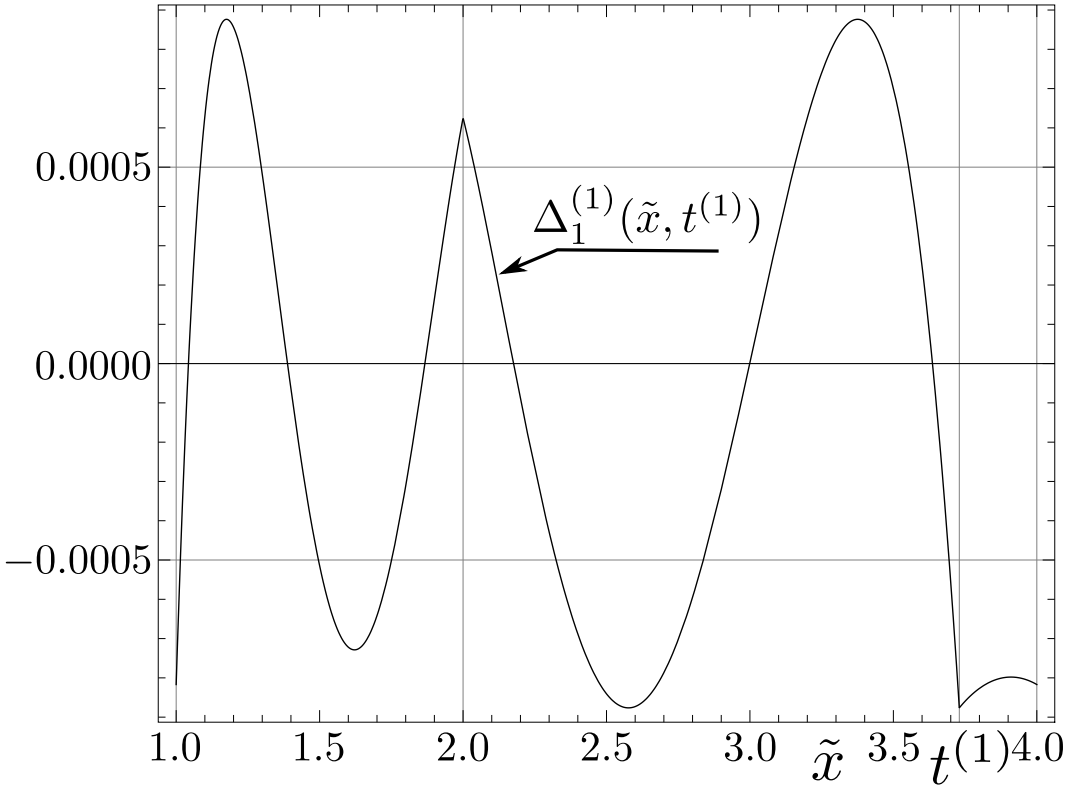

Figure 1. Graph of the function $\Delta_1^{(1)}(\tilde{x}, t^{(1)})$.

for all $t \neq t^{(1)}$. Second, we determine $d_1^{(1)}$ such that

$$\max_{\tilde{x}\in\tilde{A}} \Delta_1^{(1)}(\tilde{x}, t^{(1)}, d_1^{(1)}) = -\min_{\tilde{x}\in\tilde{A}} \Delta_1^{(1)}(\tilde{x}, t^{(1)}, d_1^{(1)})\,. \tag{4.6}$$

Thus we have

$$\max_{\tilde{x}\in\tilde{A}} |\Delta_1^{(1)}(\tilde{x}, t^{(1)}, d_1^{(1)})| \leqslant \max_{\tilde{x}\in\tilde{A}} |\Delta_1^{(1)}(\tilde{x}, t, d_1)| \tag{4.7}$$

for all real d_1 and $t \in (2, 4)$. $\Delta_1^{(1)}(\tilde{x}, t)$ is an increasing function of $\tilde{\delta}_1(\tilde{x}, t)$, hence

$$-\frac{d_1^{(1)} - 2\max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1(\tilde{x}, t_1^{(1)})|}{2 - d_1^{(1)}} = \frac{d_1^{(1)}}{2 - d_1^{(1)}}, \tag{4.8}$$

which is satisfied for

$$d_1^{(1)} = \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1(\tilde{x}, t_1^{(1)})| = \tilde{\delta}_{1\max}. \tag{4.9}$$

Thus we can find the maximum error of the first correction $\Delta_1^{(1)}(\tilde{x}, t_1^{(1)})$ (presented at Fig. 1):

$$\max_{\tilde{x}\in\tilde{A}} |\Delta_1^{(1)}(\tilde{x}, t^{(1)})| = \frac{\max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1(\tilde{x}, t^{(1)})|}{2 - \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1(\tilde{x}, t^{(1)})|}, \tag{4.10}$$

which assumes the minimum value for $t^{(1)} = t_1^{(r)}$:

Peer-reviewed version available at *Computation* **2019**, *7*, 41; doi:10.3390/computation7030041

$$\Delta_{1\,\max}^{(1)} = \frac{\max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1(\tilde{x}, t_1^{(r)})|}{2 - \max_{\tilde{x}\in\tilde{A}} |\tilde{\delta}_1(\tilde{x}, t_1^{(r)})|} = \frac{\tilde{\delta}_{1\,\max}}{2 - \tilde{\delta}_{1\,\max}} \simeq 8.7636 \cdot 10^{-4} \simeq \frac{\tilde{\delta}_{1\,\max}}{2.00}. \tag{4.11}$$

This result practically coincides with $\tilde{\delta}''_{1\,\max}$ given by (3.9).

Analogously we can determine the value of $d_2^{(1)}$ (assuming that $t = t^{(1)}$ is fixed):

$$-\frac{d_2^{(1)} - \max_{\tilde{x}\in\tilde{A}} |\Delta_1^{(1)2}(\tilde{x}, t^{(1)})(3 + \Delta_1^{(1)}(\tilde{x}, t^{(1)}))|}{2 - d_2^{(1)}} = \frac{d_2^{(1)}}{2 - d_2^{(1)}}. \tag{4.12}$$

Now, the deepest minimum comes from the global maximum

$$\max_{\tilde{x}\in\tilde{A}} |\Delta_1^{(1)2}(\tilde{x}, t^{(1)})(3 + \Delta_1^{(1)}(\tilde{x}, t^{(1)}))| = \frac{2\tilde{\delta}_{1\,\max}^2(3 - \tilde{\delta}_{1\,\max})}{(2 - \tilde{\delta}_{1\,\max})^3}. \tag{4.13}$$

Therefore we get

$$d_2^{(1)} = \frac{\tilde{\delta}_{1\,\max}^2(3 - \tilde{\delta}_{1\,\max})}{(2 - \tilde{\delta}_{1\,\max})^3} \simeq 1.15234 \cdot 10^{-6}, \tag{4.14}$$

and the maximum error of the second correction is given by

$$\Delta_{2\,\max}^{(1)} = \frac{d_2^{(1)}}{2 - d_2^{(1)}} \simeq 5.76173 \cdot 10^{-7} \simeq \frac{\tilde{\delta}_{2\,\max}}{7.98}, \tag{4.15}$$

which is very close to the value of $\tilde{\delta}''_{2\,\max}$ given by (3.15).

Thus we have obtained the algorithm *InvSqrt1* which looks like *InvSqrt* with modified values of numerical coefficients.

```
1.  float InvSqrt1(float x){
2.      float simhalfnumber = 0.500438180f*x;
3.      int i = *(int*) &x;
4.      i = 0x5F375A86 - (i>>1);
5.      y = *(float*) &i;
6.      y = y*(1.50131454f - simhalfnumber*y*y);
7.      y = y*(1.50000086f - 0.999124984f*simhalfnumber*y*y);
8.      return y ;
9.  }
```

Comparing *InvSqrt*1 with *InvSqrt* we easily see that the number of algebraic operations in *InvSqrt*1 is greater by 1 (an additional multiplication in line 7, corresponding to the second iteration of the modified Newton-Raphson procedure). We point out that magic constants for *InvSqrt* and *InvSqrt1* are the same.

5. Numerical experiments

The new algorithms were tested on the processor Intel Core i5-3470 using the compiler TDM-GCC 4.9.2 32-bit (then, in the case of *InvSqrt*, the values of errors are practically the same as those obtained by Lomont [38]). The same results were obtained also on Intel i7-5700. In this section we analyze rounding errors for the code *InvSqrt1*.

Applying algorithm *InvSqrt*1 we obtain relative errors characterized by "oscillations" with a center slightly shifted with respect to the analytical solution, see Fig. 2. Calculations were carried out for all

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 August 2019 doi:10.20944/preprints201908.0045.v1

Peer-reviewed version available at Computation 2019, 7, 41; doi:10.3390/computation7030041

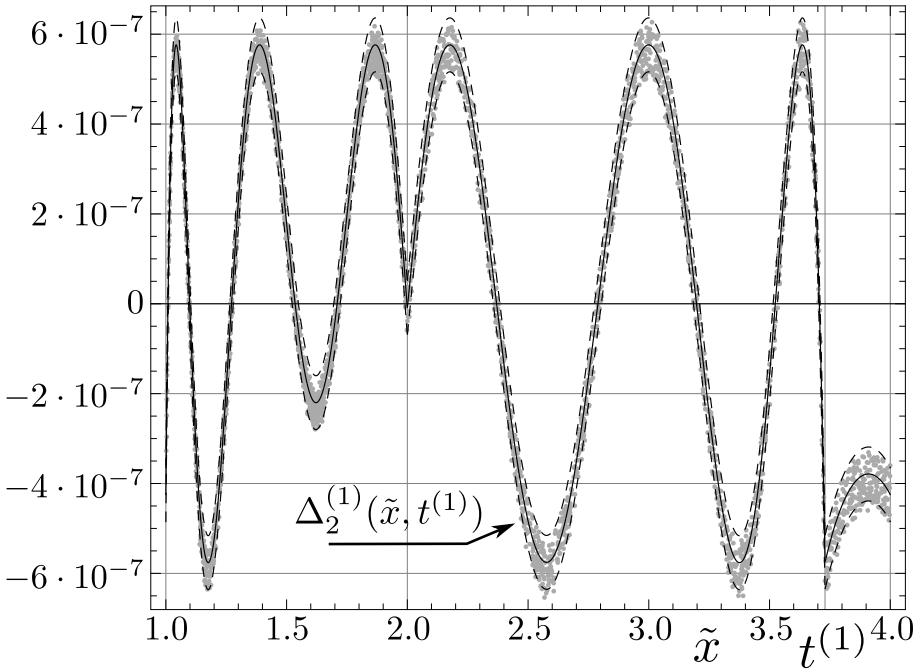

Figure 2. Solid lines represent function $\Delta_2^{(1)}(\tilde{x}, t^{(1)})$. Its vertical shifts by $\pm 6 \cdot 10^{-8}$ are denoted by dashed lines. Finally, dots represent relative errors for 4000 random values $x \in (2^{-126}, 2^{128})$ produced by algorithms *InvSqrt*1.

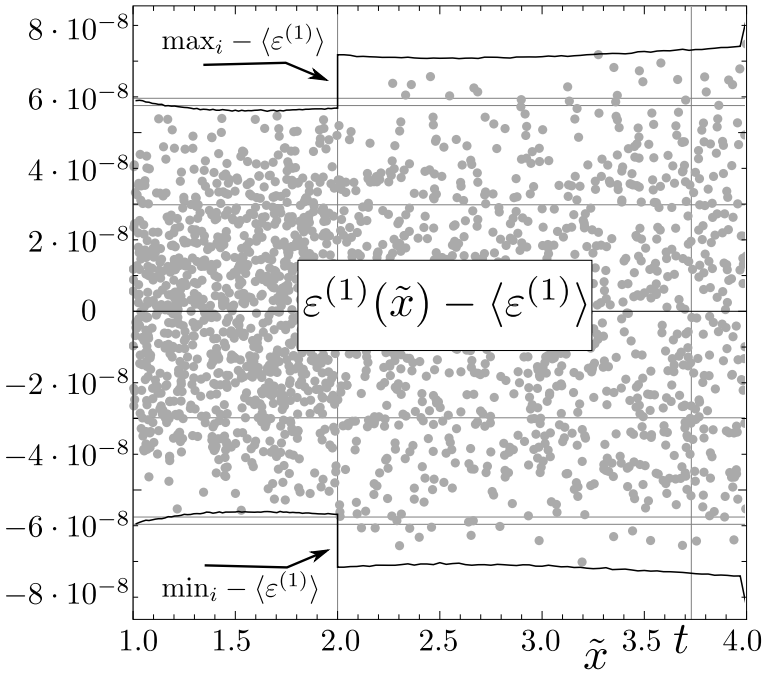

Figure 3. Relative error $\varepsilon^{(1)}$ arising during the **float** approximation of corrections $\tilde{y}_{22}(\tilde{x}, t)$. Dots represent errors determined for 2000 random values $\tilde{x} \in [1, 4)$. Solid lines represent maximum ($\max_i$) and minimum ($\min_i$) values of relative errors (intervals $[1, 2)$ and $[2, 4)$ were divided into 64 equal intervals, and then extremum values were determined in all these intervals).

Peer-reviewed version available at Computation 2019, 7, 41; doi:10.3390/computation7030041

numbers x of the type **float** such that $e_x \in [-126, 128)$. The range of errors is the same for all these intervals (except $e_x = -126$):

$$\Delta^{(1)}_{2;N}(x) = \mathrm{sqrt}(x) * \mathrm{InvSqrt1}(x) - 1. \in (\Delta^{(1)}_{2,N\,\min}, \Delta^{(1)}_{2,N\,\max}), \tag{5.1}$$

where

$$\Delta^{(1)}_{2,N\,\min} = -6.62 \cdot 10^{-7}, \qquad \Delta^{(1)}_{2,N\,\max} = 6.35 \cdot 10^{-7}.$$

For $e_x = -126$ the interval of errors is slightly wider:

$$[-6.72 \cdot 10^{-7}, 6.49 \cdot 10^{-7}].$$

This can be explained by the fact that the analysis presented in this paper is not applicable to subnormals numbers, see (2.1). The observed blur can be noticed already for the approximation error of the correction $\tilde{y}_{22}(\tilde{x})$:

$$\varepsilon^{(1)}(\tilde{x}) = \frac{\mathrm{InvSqrt1(x)} - \tilde{y}_{22}(\tilde{x}, t^{(1)})}{\tilde{y}_{22}(\tilde{x}, t^{(1)})}. \tag{5.2}$$

The values of this error are distributed symmetrically around the mean value $\langle \varepsilon^{(1)} \rangle$:

$$\langle \varepsilon^{(1)} \rangle = 2^{-1} N_m^{-1} \sum_{x \in [1,4)} \varepsilon^{(1)}(\tilde{x}) = -1.398 \cdot 10^{-8} \tag{5.3}$$

enclosing the range:

$$\varepsilon^{(1)}(\tilde{x}) \in [-9.676 \cdot 10^{-8}, 6.805 \cdot 10^{-8}], \tag{5.4}$$

see Fig. 3. The blur parameters of the function $\varepsilon^{(1)}(\tilde{x}, t)$ show that the main source of the difference between analytical and numerical results is the use of precision **float** and, in particular, rounding of constant parameters of the function InvSqrt1. It is worthwhile to point out that in this case the amplitude of the error oscillations is about 40% greater than the amplitude of oscillations of $(\tilde{y}_{00} - \tilde{y}_0)/\tilde{y}_0$ (i.e., in the case of *InvSqrt*), see the right part of Fig. 2 in [46].

6. Conclusions

In this paper we have presented a modification of the famous code *InvSqrt* for fast computation of the inverse square root. The new code has the same magic constant but the second part (which consists of Newton-Raphson iterations) is modified. In the case of one Newton-Raphson iteration the new code *InvSqrt1* has the same computational cost as *InvSqrt* and is 2 times more accurate. In the case of two iterations the computational cost of the new code is sligtly higher but its accuracy is higher by 8 times.

The main idea of our work consists in modifying coefficients in the Newton-Raphson method and demanding that the maximal error is as small as possible. Such modifications can be constructed if the distribution of errors for Newton-Raphson corrections is not symmetric (like in the case of the inverse square root, when they are non-positive functions).

Author Contributions: Conceptualization, Leonid V. Moroz; Formal analysis, Cezary J. Walczyk; Investigation, Cezary J. Walczyk, Leonid V. Moroz and Jan L. Cieśliński; Methodology, Cezary J. Walczyk and Leonid V. Moroz; Visualization, Cezary J. Walczyk; Writing–original draft, Jan L. Cieśliński; Writing–review & editing, Jan L. Cieśliński

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. M.D. Ercegovac, T. Lang: *Digital Arithmetic*, Morgan Kaufmann 2003.
2. B. Parhami: *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd edition, Oxford Univ. Press, New York, 2010
3. K. Diefendorff, P. K. Dubey, R. Hochsprung, H. Scales: AltiVec extension to PowerPC accelerates media processing, *IEEE Micro* 20 (2) (2000) 85-95.
4. D. Harris: A Powering Unit for an OpenGL Lighting Engine, *Proc. 35th Asilomar Conf. Singals, Systems, and Computers* (2001), pp. 1641–1645.
5. M. Sadeghian, J. Stine: Optimized Low-Power Elementary Function Approximation for Chybyshev series Approximation, *46th Asilomar Conf. on Signal Systems and Computers*, 2012.
6. D. M. Russinoff: A Mechanically Checked Proof of Correctness of the AMD K5 Floating Point Square Root Microcode, *Formal Methods in System Design* 14 (1) (1999) 75–125.
7. M.Cornea, C. Anderson, C. Tsen: Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic, *Software and Data Technologies* (*Communications in Computer and Information Science*, vol. 10), Springer 2008, pp. 97–109.
8. J.-M. Muller, N. Brisebarre, F. Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D.Stehlé, S. Torres: Hardware Implementation of Floating-Point Arithmetic, *Handbook of Floating-Point Arithmetic* (2009), pp. 269–320.
9. J.-M. Muller, N. Brisebarre, F. Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D.Stehlé, S. Torres: Software Implementation of Floating-Point Arithmetic, *Handbook of Floating-Point Arithmetic* (2009), pp. 321–372.
10. T. Viitanen, P. Jääskeläinen, O. Esko, J. Takala: Simplified floating-point division and square root, Proc. IEEE Int. Conf. Acoustics Speech and Signal Process., pp. 2707–2711, May 26–31 2013.
11. M.D. Ercegovac, T. Lang: *Division and Square Root: Digit Recurrence Algorithms and Implementations*, Boston: Kluwer Academic Publishers, 1994.
12. W. Liu, A. Nannarelli: Power Efficient Division and Square root Unit, *IEEE Trans. Comp.* 61 (8) (2012) 1059–1070.
13. L.X. Deng, J.S. An: A low latency High-throughput Elementary Function Generator based on Enhanced double rotation CORDIC, *IEEE Symposium on Computer Applications and Communications (SCAC)*, 2014.
14. M. X. Nguyen, A. Dinh-Duc: Hardware-Based Algorithm for Sine and Cosine Computations using Fixed Point Processor, *11th International Conf. on Electrical Engineering/Electronics Computer, Telecommuncations and Information Technology*, IEEE 2014.
15. M. Cornea, Intel® AVX-512 Instructions and Their Use in the Implementation of Math Functions, Intel Corporation 2015.
16. H. Jiang, S. Graillat, R. Barrio, C. Yang: Accurate, validated and fast evaluation of elementary symmetric functions and its application, *Appl. Math. Computation* 273 (2016) 1160–1178.
17. A. Fog: Software optimization resources, Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs, http://www.agner.org/optimize/
18. L. Moroz, W. Samotyy: Efficient floating-point division for digital signal processing application, *IEEE Signal Processing Magazine* 36 (1) (2019) 159–163.
19. D.H. Eberly: *GPGPU Programming for Games and Science*, CRC Press 2015.
20. N. Ide, M. Hirano, Y. Endo, S. Yoshioka, H. Murakami, A. Kunimatsu, T. Sato, T. Kamei, T. Okada, M. Suzuoki: 2.44-GFLOPS 300-MHz Floating-Point Vector-Processing Unit for High-Performance 3D Graphics Computing, *IEEE J. Solid-State Circuits* 35 (7) (2000) 1025-1033.
21. S. Oberman, G. Favor, F. Weber: AMD 3DNow! technology: architecture and implementations, *IEEE Micro* 19 (2) (1999) 37-48.
22. T.J. Kwon, J. Draper: Floating-point Division and Square root Implementation using a Taylor-Series Expansion Algorithm with Reduced Look-Up Table, 51st Midwest Symposium on Circuits and Systems, 2008.
23. T.O. Hands, I. Griffiths, D.A. Marshall, G. Douglas: The fast inverse square root in scientific computing, *Journal of Physics Special Topics* 10 (1) (2011) A2-1.

24. J. Blinn: Floating-point tricks, *IEEE Comput. Graphics Appl.* 17 (4) (1997) 80-84.
25. J. Janhunen: Programmable MIMO detectors, PhD thesis, University of Oulu, Tampere 2011.
26. J.L.V.M. Stanislaus, T. Mohsenin: High Performance Compressive Sensing Reconstruction Hardware with QRD Process, IEEE International Symposium on Circuits and Systems (ISCAS'12), May 2012.
27. Q. Avril, V. Gouranton, B. Arnaldi: Fast Collision Culling in Large-Scale Environments Using GPU Mapping Function, ACM Eurographics Parallel Graphics and Visualization, Cagliari, Italy (2012).
28. R. Schattschneider: Accurate high-resolution 3D surface reconstruction and localisation using a wide-angle flat port underwater stereo camera, PhD thesis, University of Canterbury, Christchurch, New Zealand, 2014.
29. S. Zafar, R. Adapa: Hardware architecture design and mapping of "Fast Inverse Square Root's algorithm", International Conference on Advances in Electrical Engineering (ICAEE), 2014, pp. 1-4.
30. T. Hänninen, J. Janhunen, M. Juntti: Novel detector implementations for 3G LTE downlink and uplink, *Analog. Integr. Circ. Sig. Process.* 78 (2014) 645–655.
31. Z.Q. Li, Y. Chen, X.Y. Zeng: OFDM Synchronization implementation based on Chisel platform for 5G research, *2015 IEEE 11th International Conference on ASIC (ASICON)*.
32. C.J. Hsu, J.L. Chen, L.G. Chen: An Efficient Hardware Implementation of HON4D Feature Extraction for Real-time Action Recognition, 2015 IEEE International Symposium on Consumer Electronics (ISCE).
33. C.H. Hsieh, Y.F. Chiu, Y.H. Shen, T.S. Chu, Y.H. Huang: A UWB Radar Signal Processing Platform for Real-Time Human Respiratory Feature Extraction Based on Four-Segment Linear Waveform Model, *IEEE Trans. Biomed. Circ. Syst.* 10 (1) (2016) 219–230.
34. J.D. Lv, F. Wang, Z.H. Ma: Peach Fruit Recognition Method under Natural Environment, Eighth International Conference on Digital Image Processing (ICDIP 2016), Proc. of SPIE Vol. 10033, edited by C.M.Falco, X.D.Jiang, 1003317 (29 August 2016).
35. D. Sangeetha, P. Deepa: Efficient Scale Invariant Human Detection using Histogram of Oriented Gradients for IoT Services, 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems, p. 61–66, IEEE 2016.
36. J. Lin, Z.G. Xu, A. Nukada, N. Maruyama, S. Matsuoka: Optimizations of Two Compute-bound Scientific Kernels on the SW26010 Many-core Processor, 46th International Conference on Parallel Processing, p. 432–441, IEEE 2017.
37. id software, quake3-1.32b/code/game/q_math.c , Quake III Arena, 1999.
38. C. Lomont, Fast inverse square root, Purdue University, Tech. Rep., 2003. Available online: http://www.lomont.org/Math/Papers/2003/InvSqrt.pdf.
39. H.S. Warren: *Hacker's delight*, second edition, Pearson Education 2013.
40. P. Martin: Eight Rooty Pieces, *Overload Journal* 135 (2016) 8–12.
41. M. Robertson: A Brief History of InvSqrt, Bachelor Thesis, Univ. of New Brunswick 2012.
42. B. Self: Efficiently Computing the Inverse Square Root Using Integer Operations. May 31, 2012.
43. C. McEniry: The Mathematics Behind the Fast Inverse Square Root Function Code, Tech. rep. 2007.
44. D. Eberly: An approximation for the Inverse Square Root Function, 2015, Available online: http://www.geometrictools.com/Documentation/ApproxInvSqrt.pdf.
45. P. Kornerup, J.-M. Muller: Choosing starting values for certain Newton-Raphson iterations, *Theor. Comp. Sci.* 351 (2006) 101–110.
46. L. Moroz, C.J. Walczyk, A. Hrynchyshyn, V. Holimath, J.L. Cieśliński: Fast calculation of inverse square root with the use of magic constant – analytical approach, *Appl. Math. Computation* 316 (2018) 245–255.