Efficient Initial Approximation and Fast Converging Methods for Division and Square Root

Masayuki Ito†*, Naofumi Takagi‡ and Shuzo Yajima†

† Department of Information Science
Kyoto University
Kyoto, 606-01 Japan

‡ Department of Information Engineering
Nagoya University
Nagoya, 464-01 Japan

Abstract

Efficient initial approximations and fast converging algorithms are important to achieve the desired precision faster at lower hardware cost in multiplicative division and square root. In this paper, a new initial approximation method for division, an accelerated higher order converging division algorithm, and a new square root algorithm are proposed. They are all suitable for implementation on an arithmetic unit where one multiply-accumulate operation can be executed in one cycle. In the case of division, the combination of our initial approximation method and our converging algorithm enables a single iteration of the converging algorithm to produce double-precision quotients. Our new square root algorithm can form double-precision square roots faster using smaller look-up tables than the Newton-Raphson method.

1 Introduction

Division and square root are essential operations in many scientific and engineering applications. With the increasing availability of high-speed multiplication units, multiplicative algorithms have become advantageous to the fast calculation of division and square root. In general, such methods begin with an initial approximation to the desired value, which is then improved by an iterative algorithm through some multiplications and additions. Therefore, efficient initial approximation methods and fast converging algorithms hold the key of achieving the desired precision faster at lower hardware cost.

We deal with division and square root operations on the mantissa parts of floating point numbers. We assume an arithmetic unit on which one multiply-accumulate operation $(A \times B) + C$ can be executed in one cycle. (The width of the unit should be slightly more than b bits for b-bit division and square root.) This operation takes no more time than a multiplication by incorporating addition into the adder tree of the multiplier. Many functions can be calculated efficiently on such a multiply-add unit.

We propose three new methods, an initial approximation method for division, a higher order converging division algorithm accelerated through table look-up, and a new converging algorithm for square root. They are all suitable for implementation on a multiply-add unit.

For the generation of initial approximations, look-up tables are commonly used. The simplest way is directly reading an initial approximation to the desired value through table look-up using some most significant bits of an operand as an index. Besides this direct approximation method, a linear approximation method can be employed. In this case, the table entry is the two coefficients of the linear function. Evaluation of the initial approximation from the linear function requires one multiply-accumulate operation, and the result is roughly twice as many bits of accuracy as that achieved by the direct approximation method. In this paper, we propose an efficient approximation method to form an initial approximation to the reciprocal of a divisor. It requires only one multiply-accumulate operation and its accuracy is higher than that achieved by the original linear approximation.

Multiplicative division algorithms, represented by the Newton-Raphson method and Goldschmidt's algorithm, use iterations to refine an approximation to the desired result. The typical rate of convergence is quadratic, which means the number of correct bits doubles with each iteration. An extended algorithm of the Newton-Raphson method has been presented, where an arbitrary higher order convergence can be achieved [1]. In this paper, we develop a scheme to speed up the convergence of the algorithm, and propose an accelerated higher order converging algorithm. The combination of this converging algorithm and our initial approximation method achieves very high-performance. A single iteration of the converging algorithm is enough for double-precision division.

Many multiplicative square root algorithms have been presented [2]. Among them, the multiplicative Newton-Raphson method has been widely used. It involves three multiplications per iteration. In this paper, we propose a new algorithm for square root. It calculates $\sqrt{X}$ in a different manner from previous converging algorithms. It does not directly refine an initial approximation to the final result, but refines an initial approximation to the difference between the ex-

*Presently, with Hitachi, Ltd., Tokyo, Japan.

1063-6889/95 $4.00 © 1995 IEEE

act $\sqrt{X}$ and an approximation to $\sqrt{X}$. The iterative formula of our algorithm consists of only one multiply-accumulate operation, which can be executed in one cycle on a multiply-add unit. The same speed-up technique on convergence as we propose for division can also be adopted.

This paper is organized as follows. In Section 2 and 3, we discuss division and square root operations respectively. In each section we review the conventional initial approximation methods and converging algorithms first, and propose our methods. We also compare these methods from the viewpoint of the required numbers of cycles on a multiply-add unit and necessary table sizes. A conclusion is shown in Section 4.

2 Division

We discuss multiplicative division on a multiply-add unit. We concentrate on forming the reciprocal of a given divisor $Y = [1.y_1y_2\cdots y_my_{m+1}\cdots y_b]$, where $1 \leq Y < 2$. We first review the conventional initial approximation methods and a higher order converging algorithm [1] in Section 2.1. Then we propose an efficient initial approximation method in Section 2.2. We also propose an accelerated higher order converging algorithm by developing a speed-up technique on convergence in Section 2.3. In Section 2.4, we compare our methods to the methods in Section 2.1.

2.1 Conventional multiplicative methods

Initial approximation

Multiplicative division methods begin with an initial approximation R_0 to the reciprocal of the divisor Y. After R_0 is formed, an iterative formula of a converging algorithm is used to achieve the desired precision.

The simplest way of forming R_0 is reading an approximation to $1/Y$ directly out of a look-up table [3][4]. Suppose the table width be t bits. As the table is addressed with the m most significant bits of Y, an initial approximation R_0 is in the range $0.5 < R_0 \leq 1$ and has the form $[0.1r_1r_2r_3\cdots r_t]$. To give the best value for the subinterval $[p, p+2^{-m})$, where $p = [1.y_1y_2\cdots y_m]$, the look-up table should contain the value $\frac{1}{2}(\frac{1}{p} + \frac{1}{p+2^{-m}})$. The worst error occurs on the first subinterval $p = 1$. The total error ϵ_d considering the truncations error due to storing only t bits in the table is

$$|\epsilon_d| = |R_0 - \frac{1}{Y}| < 2^{-m-1} + 2^{-t-2} \qquad (1)$$

and the table is of size $2^m \times t$ bits.

We can select any combination of m and t. When we set $t = m$, ϵ_d satisfies $|\epsilon_d| < \frac{3}{2}\cdot 2^{-m-1}$. When $t = m+3$, $|\epsilon_d| < \frac{17}{16}\cdot 2^{-m-1}$. m and t should be selected so that the required table size becomes smallest as the approximation satisfies the necessary accuracy. This is common to the approximation methods in this paper.

Besides this direct approximation, a linear approximation method can be employed. In this case, a linear function $-C_1 \cdot Y + C_0$ is adopted to approximate $1/Y$ and the calculation of it requires one cycle on a multiply-add unit. The two coefficients C_0 and C_1 are read out of a look-up table addressed with the m most significant bits of Y and have the length of t bits for each.

Let $E(Y)$ be the error function:

$$E(Y) = -C_1 \cdot Y + C_0 - \frac{1}{Y}. \qquad (2)$$

Differentiating (2) yields $E'(Y) = \frac{1}{Y^2} - C_1$. Hence,

$$E(Y)_{max} = E(\frac{1}{\sqrt{C_1}}) = C_0 - 2\sqrt{C_1}. \qquad (3)$$

To minimize $|E(Y)|_{max}$, the errors of both endpoints of each subinterval, i.e., $E(p)$ and $E(p+2^{-m})$, should have the same value and it should be equal to $-E(Y)_{max}$. From these conditions we get

$$\begin{cases} C_1 = \frac{1}{p\cdot(p+2^{-m})}, \\ C_0 = \frac{p+2^{-m-1}+\sqrt{p\cdot(p+2^{-m})}}{p\cdot(p+2^{-m})}, \end{cases} \quad \text{and} \qquad (4)$$

$$|E(Y)|_{max} < \frac{1}{p^3}\cdot 2^{-2m-3}. \qquad (5)$$

The total error of the linear approximation ϵ_l considering truncation errors due to storing only t bits of the two coefficients in the table is

$$|\epsilon_l| < 2^{-2m-3} + 2^{-t} \qquad (6)$$

and the table is of size $2^m \times t \times 2$ bits.

For example, setting $t = 2m+3$ yields $|\epsilon_l| < 2^{-2m-2}$ and the table is of size $2^m \times (4m+6)$ bits.

A polynomial approximation using a polynomial of degree $n \geq 2$ may be used [5][6]. It needs n multiply-accumulate operations and the look-up table must keep $(n+1)$ coefficients of the polynomial. It is not efficient as an initial approximation method, because the error decreases faster by adopting higher order converging algorithms.

Multiplicative division algorithm

The most commonly used multiplicative division algorithm is the Newton-Raphson method shown below using R_0 as an initial approximation to $1/Y$.

$$R_i = R_{i-1} \times (2 - R_{i-1} \times Y), \qquad (7)$$

where R_i converges to $1/Y$ quadratically.

The Newton-Raphson method (7) has been extended in the following way [1]:

step1: $D_i = 1 - R_{i-1} \times Y$
step2: $R_i = (1 + D_i + D_i^2 + D_i^3 + \cdots + D_i^{k_i-1}) \times R_{i-1}$

Note that this algorithm has a property of k_i-th order convergence.

We rewrite this higher order converging algorithm for implementation on a multiply-add unit as follows

and we call it Algorithm HOC. (n is the number of iterations.)

[Algorithm HOC]
for $i = 1$ to n do
  step1: $D_{i,0} = 1 - R_{i-1} \times Y$
  step2.1: for $j = 1$ to $k_i - 2$ do
    $D_{i,j} = D_{i,0} \times D_{i,j-1} + D_{i,0}$
  step2.2: $R_i = R_{i-1} \times D_{i,k_i-2} + R_{i-1}$

To analyze the error of Algorithm HOC, let $\epsilon_{i,j}$ be the error of R_i when step2.1 is iterated j times;

$$\epsilon_{i,j} = \frac{1}{Y} - (R_{i-1} \times D_{i,j} + R_{i-1}). \quad (8)$$

Then,

$$\begin{aligned} \epsilon_{i,0} &= Y \cdot \epsilon_{i-1,k_{i-1}-2}^2 \ (i \geq 1) \text{ and} \quad (9) \\ \epsilon_{i,j} &= D_{i,0} \cdot \epsilon_{i,j-1} \\ &= \epsilon_{i-1,k_{i-1}-2} \cdot Y \cdot \epsilon_{i,j-1} \ (i,j \geq 1). \quad (10) \end{aligned}$$

(9) and (10) show that Algorithm HOC has a property of k_i-th order convergence when step2.1 is iterated $(k_i - 2)$ times. The number of required cycles for the i-th iteration is also k_i. If we set $k_i = 2$ for all i's, the algorithm is the original Newton-Raphson algorithm (7). In the case $k_i = 4$, it is equivalent to the case where Newton-Raphson iterative formula (7) is iterated twice. In order to achieve the best accuracy for some given number of cycles M, k_i must be 3 for all i's if M is divisible by 3, otherwise the only exception should be $k_i = 2$ or 4 for only one i. Therefore, we choose such an optimized combination of k_i's in Algorithm HOC.

2.2 New initial approximation method

In this section, we propose a new initial approximation method. We adopt a modified linear function

$$A_1 \times (2p + 2^{-m} - Y) + A_0$$

instead of $-C_1 \cdot Y + C_0$ for the approximation to $1/Y$. Recall that $p = [1.y_1y_2 \cdots y_m]$. We find the following relation between C_1 and C_0 in the linear approximation (4):

$$C_0 \simeq C_1 \times (2p + 2^{-m}).$$

Then $A_1 \cdot (2p + 2^{-m} - Y)$ can form almost the same value as $-C_1 \cdot Y + C_0$ by setting $A_1 = C_1$. We can decide A_0 to improve the approximation.

We get the value $2p + 2^{-m} - Y$ as follows. Let $q = Y - p$. Then q has the form

$$q = [0.00 \cdots 0 \ \ y_{m+1}y_{m+2}y_{m+3} \cdots].$$

We get the relation below:

$$\begin{aligned} 2p + 2^{-m} - Y &= p + 2^{-m} - q \\ &= [1.y_1y_2y_3 \cdots y_m\tilde{y}_{m+1}\tilde{y}_{m+2}\tilde{y}_{m+3} \cdots], \quad (11) \end{aligned}$$

where $\tilde{1} = 0$ and $\tilde{0} = 1$. Thus $2p + 2^{-m} - Y$ can be obtained only by inverting less significant bits than y_m in Y.

Now we show the appropriate value of A_0. A_0 should be an approximation to $1/Y - A_1 \times (2p + 2^{-m} - Y)$, where $A_1 (= C_1) = \frac{1}{p \cdot (p + 2^{-m})}$. We use the following transformation:

$$\begin{aligned} A_0 &= \frac{1}{Y} - A_1 \times (2p + 2^{-m} - Y) \\ &= \frac{1}{p+q} - \frac{(p + 2^{-m}) + (p - Y)}{p \cdot (p + 2^{-m})} \\ &= \frac{1}{p} \cdot (1 + \frac{q}{p})^{-1} - \frac{(p + 2^{-m}) - q}{p \cdot (p + 2^{-m})} \\ &= (\frac{1}{p} - \frac{q}{p^2} + \frac{q^2}{p^3} - \cdots) - \{\frac{1}{p} - \frac{q}{p \cdot (p + 2^{-m})}\} \\ &= \frac{q \cdot (q - 2^{-m})}{p^3} + O(2^{-3m}). \end{aligned}$$

Since A_0 depends on both p and q, the table for A_0 should be addressed with the m_p most significant bits of p and the m_q most significant bits of q. Suppose the table width for A_0 and A_1 be t_0 and t_1, respectively. Note that $|A_0| < 2^{-2m-2}$. Then, the total error of the modified linear approximation ϵ_m is shown below:

$$|\epsilon_m| < (3 \cdot 2^{-m_p-1} + 2 \cdot 2^{-m_q} + 2^{-t_0-1}) \cdot 2^{-2m-2} + 2^{-t_1-1}. \quad (12)$$

In order to make the table size for A_0 nearly equal to that for A_1, we set

$$m_p = \lfloor \frac{m}{2} \rfloor \text{ and } m_q = \lceil \frac{m}{2} \rceil.$$

Adopting $t_0 = \lceil \frac{m}{2} \rceil + 1$ and $t_1 = \lfloor \frac{5m}{2} \rfloor + 4$ results in

$$|\epsilon_m| < 2^{-2.5m} \quad (13)$$

and the total table size is $2^m \times (3m + 5)$ bits. (Here, we fix the table widths t_0 and t_1 as to m, because they do not diminish the accuracy of the final result.) This result shows that the proposed method is more effective than the linear approximation when $m > 4$. For instance, when $m = 10$, the proposed method gives about three bits better approximation using smaller look-up tables than the linear approximation.

The proposed modified linear approximation method can also be applied in producing an approximation for multiplicative square root [7].

2.3 Speed-up technique on convergence

By the analysis of Section 2.1, Algorithm HOC gives the best performance when its convergent rate is cubic. In this section we propose an accelerated HOC algorithm by developing a scheme to speed up the convergence of Algorithm HOC. Since the iterative formula of step2.1 in Algorithm HOC is $D_{i,j} = D_{i,0} + D_{i,0} \times D_{i,j-1}$. We can calculate $D_{i,2} (= D_{i,0} + D_{i,0}^2 + D_{i,0}^3)$ in one cycle on a multiply-add unit in the following way:

Table 1 : Number of correct bits using m bits for initial approximation with M cycles

Method	M=1	M=2	M=3	M=4	M=5	M=6	Table size
DA+HOC	m	2m+1	3m+2	4m+3	6m+5	9m+8	$2^m \times t$
LA+HOC	2m+2	—	4m+5	6m+8	8m+11	12m+17	$2^m \times 2t$
ML+HOC	2.5m	—	5m	7.5m	10m	15m	$2^m \times (3m+5)$
ML+AHOC	—	—	—	7.5m+$l-2$	10m+l−2	15m+2l−4	$2^m \times (3m+5) + 2^l \times l$

$$D_{i,2} = D_{i,0} \times (1 + D_{i,0}) + \hat{D}^3_{i,0}, \qquad (14)$$

where $\hat{D}^3_{i,0}$ is an approximation to $D^3_{i,0}$ read through table look-up using some upper digits of $D_{i,0}$ as an index. $(1 + D_{i,0})$ can be obtained easily by adding 1 to $D_{i,0}$ using only small additional hardware. Now we show the accelerated HOC algorithm, Algorithm AHOC.

[Algorithm AHOC]
for $i = 1$ to n do
step1: $D_{i,0} = 1 - R_{i-1} \times Y$
step2.1: $D_{i,1} = D_{i,0} \times (1 + D_{i,0}) + \hat{D}^3_{i,0}$
step2.2: for $j = 2$ to $k_i - 2$ do
$D_{i,j} = D_{i,0} \times D_{i,j-1} + D_{i,0}$
step2.3: $R_i = R_{i-1} \times D_{i,k_{i-2}} + R_{i-1}$

Here, $\hat{D}^3_{i,0}$ is an approximation to $D^3_{i,0}$. Suppose the table for $\hat{D}^3_{i,0}$ is of size $2^l \times l$ bits$(3 \le l \le m)$. When $D_{i,0}$ is guaranteed to be $0 < D_{i,0} < 2^{-s}$, the s most significant bits of $D_{i,0}$ are all 0's. We use l bits from the $(s+1)$-th to the $(s+l)$-th position of $D_{i,0}$ as an index for the look-up table. In the case $D_{i,0} < 0$, these bits should be inverted before addressing the table. Let $\eta_d < 2^{-s-l}$ be the error due to using only l bits of $D_{i,0}$ for addressing the table. Then,

$$\begin{aligned} |\hat{D}^3_{i,0} - D^3_{i,0}| &= (D_{i,0} + \eta_d)^3 - D^3_{i,0} \\ &< 3D^2_{i,0} \cdot 2^{-s-l} + O(2^{-3s-2l}). \end{aligned} \qquad (15)$$

Since the truncation error of $\hat{D}^3_{i,0}$ is less than $2^{-3s-l-1}$, $\epsilon_{i,1}$ by Algorithm AHOC becomes

$$\begin{aligned} |\epsilon_{i,1}| &= |\frac{1}{Y} - R_i \times (1 + D_{i,0} + D^2_{i,0} + \hat{D}^3_{i,0})| \\ &< 2^{-3s-l+2}. \end{aligned} \qquad (16)$$

The worst error of (16) is 2^{-l+2} times smaller than $|\epsilon_{i,1}|_{max}$ by Algorithm HOC. It means AHOC produces $(l-2)$ more correct bits using a table of size $l \cdot 2^l$ bits than HOC in forming R_i. In the case of $l \simeq s$, AHOC achieves (k_i+1)-th order convergence with k_i cycles for $k_i \ge 3$. Recall that HOC gives k_i-th order convergence with k_i cycles.

2.4 Comparisons

In this section, we compare the proposed modified linear approximation (ML in short) to the direct approximation (DA in short) and the linear approximation (LA in short), and the proposed Algorithm AHOC to Algorithm HOC on a multiply-add unit. We compare four schemes, HOC with DA (DA+HOC), HOC with LA (LA+HOC), HOC with ML (ML+HOC) and AHOC with ML (ML+AHOC). We show how much precision can be obtained with equal numbers of cycles M. Table 1 shows the result. The precision can be evaluated by $\lfloor -\log_2 |R_n - 1/Y| \rfloor$, which shows the number of correct bits of R_n. Here, table width t for each method is assumed to be large enough not to diminish the accuracy of the final result. From Table 1, more accuracy can be produced with less cycles by the proposed method (ML+AHOC) than the conventional method (DA+HOC, LA+HOC) with the look-up tables of nearly same size. It shows that the proposed method can reduce the number of cycles and achieves fast division on a multiply-add unit.

When we calculate double-precision (53-bit) quotients for the IEEE standard, R_n must satisfy $|R_n - 1/Y| < 2^{-54}$ because the final multiplication by the dividend X, where $1 \le X < 2$, is needed. (We do not consider extra bits for rounding.) Table 2 shows how large look-up tables are required with M cycles for each method. From Table 2, in order to achieve the precision of 54 bits with $M = 3$, m must be 18, 13, and 11 for DA+HOC, LA+HOC, and ML+HOC, respectively. Corresponding required table sizes are 4352K, 448K, and 76K bits, respectively. By the proposed modified linear approximation, we can reduce the required table size into about 1/6 of that by the linear approximation, which is only 1/9 size of that by the direct approximation. In the case of $M = 4$, ML+AHOC requires 3.3K bits table, while ML+AHOC requires 7.3K bits. By Algorithm AHOC, we can save more than half of the table size. These results show that the necessary iterations of the converging formula is only one to create double-precision quotients when adopting an efficient initial approximation method.

Table 2 : The required table size for double-precision reciprocals with M cycles (bits)

Method	M=3	M=4	M=5	M=6
DA+HOC	$2^{18} \times 17$ =4352K	$2^{13} \times 14$ =112K	$2^9 \times 8$ =4.0K	$2^6 \times 5$ =320
LA+HOC	$2^{13} \times 28 \times 2$ =448K	$2^8 \times 19 \times 2$ =9.5K	$2^6 \times 13 \times 2$ =1.6K	$2^4 \times 10 \times 2$ =320
ML+HOC	$2^{11} \times (31+7)$ =76K	$2^8 \times (24+5)$ =7.3K	$2^6 \times (19+4)$ =1.4K	$2^4 \times (14+3)$ =272
ML+AHOC	— —	$2^7 \times (21+5) + 2^4 \times 4$ =3.3K	— —	— —

3 Square root

We discuss multiplicative square root. Among many converging algorithms for square root [2], the multiplicative Newton-Raphson method for calculating $1/\sqrt{X}$ is widely used. It requires three multiplications per iteration. We first review the multiplicative Newton-Raphson method in Section 3.1. In Section 3.2, we propose a new square root algorithm. The iterative formula of our algorithm consists of one multiply-accumulate operation and can be executed in one cycle on a multiply-add unit. It has a property of linear convergence. In Section 3.3, we compare our algorithm to the multiplicative Newton-Raphson method.

3.1 Newton-Raphson method

The multiplicative Newton-Raphson method for calculating $1/\sqrt{X}$ begins with an initial approximation G_0 to the reciprocal of the square root of given X, where $X = [1.x_1x_2\cdots x_mx_{m+1}\cdots x_b]$. After iterating the converging formula n times, $\sqrt{X}$ can be obtained by the additional multiplication $G_n \times X$.

In the case of the direct approximation method for $1/\sqrt{X}$, we use an m-bits-in, t-bits-out look-up table for producing G_0. For square root, however, m bits for the index consist of the $(m-1)$ most significant bits of the mantissa X, i.e., $[x_1x_2\cdots x_{m-1}]$,and the last bit of the exponent part of the original floating point number. The error $\delta_d = G_0 - 1/\sqrt{X}$ is bounded as

$$|\delta_d| = |G_0 - \frac{1}{\sqrt{X}}| < 2^{-m-1} + 2^{-t-2} \qquad (17)$$

and the table is of size $2^m \times t$ bits.

In the case of the linear approximation method, the two coefficients are read out of a look-up table addressed with the $(m-1)$ most significant bits of X together with the last bit of the exponent and have the length of t bits for each. The error of the linear approximation δ_l is bounded as

$$|\delta_l| < \frac{3}{4} \cdot 2^{-2m-2} + 2^{-t} \qquad (18)$$

and the table is of size $2^m \times t \times 2$. Setting $t = 2m+4$, for example, yields $|\delta_l| < 2^{-2m-2}$ and the table is of size $2^m \times (4m+8)$ bits.

The multiplicative Newton-Raphson iterative formula for calculating $1/\sqrt{X}$ is

$$G_i = \frac{G_{i-1}}{2} \times (3 - X \times G_{i-1}^2), \qquad (19)$$

where G_i converges to $1/\sqrt{X}$ quadratically. Each iteration (19) can be executed in three cycles as follows:

1. $Q_{i,1} = X \times G_{i-1}$
2. $Q_{i,2} = 3 - Q_{i,1} \times G_{i-1}$
3. $G_i = \frac{1}{2} \cdot (Q_{i,2} \times G_{i-1})$

After the final (n-th) iteration, one additional multiplication $G_n \times X$ is needed, which can be calculated in step3 of the final iteration by $\frac{1}{2} \cdot (Q_{n,2} \times Q_{n,1})$ instead of $\frac{1}{2} \cdot (Q_{n,2} \times G_{n-1})$. Since n iterations of the formula (19) involve $3n$ multiplications, $3n$ and $(3n+1)$ cycles are required in adopting the direct approximation and the linear approximation, respectively.

3.2 New square root algorithm

In this section we propose a new converging algorithm for calculating $\sqrt{X}$, which can be derived from the (divisional) Newton-Raphson method calculating $\sqrt{X}$ by the following iterative formula:

$$P_i = \frac{1}{2}(P_{i-1} + \frac{X}{P_{i-1}}). \qquad (20)$$

Here, P_0 is an initial approximation to $\sqrt{X}$. This method requires one division in each iteration. We divide the first iteration of (20), $P_1 = \frac{1}{2} \cdot (P_0 + \frac{X}{P_0})$, into two formulas using $K = 2P_0$.

$$S = \frac{1}{4} - \frac{1}{K^2} \times X \quad \text{and} \qquad (21)$$

$$P_1 = \frac{K}{2} - K \times S. \qquad (22)$$

If we can calculate $S_\infty = \frac{1}{2} - \frac{\sqrt{X}}{K}$ from $S_0 = \frac{1}{4} - \frac{X}{K^2}$, the exact $\sqrt{X}$ is obtained by $\frac{K}{2} - K \times S_\infty$ and iterations

of the converging formula (20) are no longer in need. Then, S_∞ and S_0 hold the following relation:

$$S_\infty - \frac{1}{2} = -\sqrt{|\frac{1}{4} - S_0|}. \quad (23)$$

Note that $S_\infty - \frac{1}{2} = -\frac{\sqrt{X}}{K}$ and $\frac{X}{K^2} = \frac{1}{4} - S_0$. (23) shows one of the two solutions of the following quadratic equation of S_∞:

$$S_\infty^2 - S_\infty + S_0 = 0. \quad (24)$$

From (24) we can derive an asymptotic equation of S:

$$S_i = S_{i-1}^2 + S_0, \quad \text{where } \lim_{i\to\infty} S_i = \frac{1}{2} - \frac{\sqrt{X}}{K}. \quad (25)$$

The new converging algorithm can be summarized as follows using $L = 1/K^2$. It has a property of linear (order) convergence and we call it Algorithm LOC.

[Algorithm LOC]
step1: $S_0 = \frac{1}{4} - L \times X$
step2: for $i = 1$ to n do
$S_i = S_{i-1}^2 + S_0$
step3: $P = \frac{K}{2} - K \times S_n$

Here, P is the final result obtained by the algorithm. The same algorithm can also be derived from a continued fraction representation of $\sqrt{X}$. (See the appendix.)

In Algorithm LOC, K is an approximation to $2\sqrt{X}$, because $K = 2P_0$, where P_0 is an approximation to $\sqrt{X}$. It is read through table look-up addressed with some upper bits of X. $L(= 1/K^2)$ must have the same precision as P. It can also be read out of a look-up table addressed with the same bits of X as for the table for K.

Unlike initial approximations for the Newton-Raphson method, we keep both K and $K' = \sqrt{2}K$ and use K' instead of K in step3 of Algorithm LOC, when the exponent part of the original floating point number is odd. Note that L is irrelevant to the exponent part. Therefore, in Algorithm LOC, we use the m most significant bits of X for m-bits-in tables.

The iterative formula of Algorithm LOC (step2) is a simple multiply-accumulate operation and can be executed in one cycle on a multiply-add unit. The operations of step1 and step3 also be done in one cycle for each. Hence, when step2 is iterated n times, the number of required cycles is $(n + 2)$ in total.

Let ξ_i be the error of $K/2 - K \cdot S_i$ from $\sqrt{X}$ when step2 is iterated i times;

$$\xi_i = \sqrt{X} - (\frac{K}{2} - K \times S_i). \quad (26)$$

Then,

$$\xi_0 = \frac{K}{4}(1 - \frac{2}{K}\sqrt{X})^2 \quad \text{and} \quad (27)$$

$$\xi_i = \xi_{i-1} \cdot (1 - \frac{2}{K}\sqrt{X}) + \frac{\xi_{i-1}^2}{K}. \quad (i \geq 1) \quad (28)$$

(28) shows that LOC has a property of linear convergence. Its coefficient is $1 - 2\sqrt{X}/K$. As K is an approximation to $2\sqrt{X}$, a high-precision approximation fastens the speed of convergence. It decreases the initial error ξ_0 at the same time. Thus, the convergent speed depends on the precision of K. When the look-up tables are addressed with the m most significant bits of X, the table for K should contain the value $K = \sqrt{w} + \sqrt{w + 2^{-m}}$ for the subinterval $[w, w+2^{-m})$, where $w = [1.x_1x_2\cdots x_m]$. Unless a truncation error of K is considered, the coefficient $(1 - 2\sqrt{X}/K)$ is less than 2^{-m-2}, which shows that the number of correct bits is increased by $(m + 2)$ bits with each iteration.

Now, let the table width for K be t. Then, $(1 - 2\sqrt{X}/K)$ satisfies the following inequality:

$$|1 - \frac{2}{K}\sqrt{X}| < \frac{1}{w} \cdot 2^{-m-2} + \frac{1}{2\sqrt{w}} \cdot 2^{-t-1}. \quad (29)$$

From (27),(28) and (29), the total error $\xi_n = P - \sqrt{X}$ is

$$|\xi_n| < \frac{1}{2}(2^{-m-2} + 2^{-t-2})^{n+2}. \quad (30)$$

We show that the convergent speed of the algorithm can be accelerated through another table look-up by the same scheme as we have proposed for division. The iterative formula of step2 in Algorithm LOC is $S_i = S_{i-1}^2 + S_0$. We can approximate $S_2(= S_0 + S_0^2 + 2S_0^3 + S_0^4)$ in one cycle on a multiply-add unit in the following way:

$$S_2 = S_0 \times (1 + S_0) + 2\hat{S}_0^3, \quad (31)$$

where $\hat{S}_0^3$ is an approximation to S_0^3 read through table look-up using some upper digits of S_0 as an index. $(1 + S_0)$ can be obtained easily by adding 1 to S_0 using only small additional hardware. Now we show the accelerated LOC algorithm, Algorithm ALOC.

[Algorithm ALOC]
step1: $S_0 = \frac{1}{4} - L \times X$
step2.1: $S_1 = S_0 \times (1 + S_0) + 2\hat{S}_0^3$
step2.2: for $i = 2$ to n do
$S_i = S_{i-1}^2 + S_0$
step3: $P = \frac{K}{2} - K \times S_n$

Suppose the table for $\hat{S}_0^3$ is of size $2^l \times l$ bits $(4 \leq l \leq m)$. Note that this table is identical to the table for speed up for Algorithm AHOC. Then, ξ_1 by Algorithm ALOC becomes

$$|\xi_1| = |\frac{K}{2} - K \cdot (S_0 + S_0^2 + 2\hat{S}_0^3) - \sqrt{X}| < 2^{-3s-l+3}. \quad (32)$$

This value is 2^{-l+3} times smaller than $|\xi_1|_{max}$ in Algorithm LOC. It means that ALOC produces $(l-3)$ more correct bits than LOC using $l \cdot 2^l$ bits table.

Table 3: Number of correct bits using m bits for initial approximation with M cycles

Method	M=1	M=2	M=3	M=4	M=5	M=6	M=7	Table size
DA+NR	—	—	2m+1	—	—	4m+2	—	$2^m \times t$
LA+NR	—	—	—	4m+4	—	—	8m+7	$2^m \times 2t$
LOC	—	2m+4	3m+6	4m+8	5m+10	6m+12	7m+14	$2^m \times (t+2b)$
ALOC	—	—	3m+l+3	4m+l+5	5m+l+7	6m+l+9	7m+l+11	$2^m \times (t+2b) + 2^l \times l$

Table 4: The required table size for double-precision square roots with M cycles (bits)

Method	M=3	M=4	M=5	M=6
DA+NR	$2^{26} \times 27$ =1728M	— —	— —	$2^{13} \times 15$ =120K
LA+NR	— —	$2^{13} \times 28 \times 2$ =448K	— —	— —
LOC	$2^{16} \times (17+106)$ =7.9M	$2^{12} \times (12+106)$ =472K	$2^9 \times (10+106)$ =58K	$2^7 \times (9+106)$ =14.4K
ALOC	$2^{13} \times (15+106) + 2^{11} \times 11$ =990K	$2^{10} \times (13+106) + 2^8 \times 8$ =121K	$2^8 \times (11+106) + 2^6 \times 6$ =29.6K	— —

3.3 Comparisons

In this section, we compare our converging algorithms (LOC and ALOC) to the multiplicative Newton-Raphson method (NR in short) with the direct approximation (DA+NR) and with the linear approximation (LA+NR) with equal numbers of cycles M on a multiply-add unit.

Table 3 shows the result. The precision or the number of correct bits of the result can be evaluated by $\lfloor -\log_2 |G_n \cdot X - \sqrt{X}| \rfloor$ for the NR method and $\lfloor -\log_2 |S_n - \sqrt{X}| \rfloor$ for our method. Here also, the table width t for each method is assumed to be large enough not to diminish the accuracy of the final result. The result indicates that the proposed algorithm can form square roots faster than the Newton-Raphson method with look-up tables of nearly same size.

When we calculate double-precision (53-bit) square roots for the IEEE standard, $|G_n \cdot X - \sqrt{X}| < 2^{-53}$ or $|S_n - \sqrt{X}| < 2^{-53}$ must be satisfied. Table 4 shows how large look-up tables are required with M cycles for each method. From Table 4, if we set $M = 6$, m must be 13 for DA+NR, while $m = 7$ for LOC. Required table sizes are 120K bits for DA+NR and 14.4K bits for LOC. This shows that LOC can form double-precision square roots with about 1/8 the table size required for the NR method. In this case, ALOC is no better than LOC, because the same $m(= 7)$ is required.

Table 4 also shows that the necessary iterations of the multiplicative Newton-Raphson method is at least 2, otherwise the required table size becomes impractically large. This means the NR method needs at least 6 cycles in forming double precision square roots. By Algorithm ALOC, however, $M = 5$ can be achieved using 29.6K bits table and $M = 4$ using 121K bits table.

The proposed linear converging algorithm can create double-precision square roots faster than the Newton-Raphson method. In contrast to the complicated quadratic converging iterative formula involving three multiplications, our linear converging iterative formula is only one multiply-accumulate operation. It contributes to high-speed square root operations.

4 Conclusion

We have proposed multiplicative division and square root methods suitable for implementation on a multiply-add unit. An efficient initial approximation method and fast converging algorithms are important to achieve the desired precision with small numbers of cycles and with small look-up tables.

In the case of division, we have proposed an efficient initial approximation method and an accelerated higher order converging algorithm. The combination of them enables a single iteration of the converging algorithm to produce double-precision quotients.

For the calculation of square root, we have proposed a new converging algorithm. The iterative formula of our algorithm consists of one multiply-accumulate operation and can be executed in one cycle on a multiply-add unit. When we calculate double-precision square roots with 6 cycles, our method requires only 1/8 the size of look-up tables compared to the multiplicative Newton-Raphson method. Furthermore, even with 4 cycles our method requires nearly same size of look-up tables as the Newton-Raphson method with 6 cycles.

Acknowledgments

The authors would like to thank the members of Professor Yajima's Laboratory at Kyoto University for their valuable comments on this work.

References

[1] D. Ferrari, A division method using a parallel multiplier, IEEE Trans. on Computers EC-16 (Apr.1967),224-226

[2] C. V. Ramamoorthy, James R. Goodman and K.H.Kim, Some properties of iterative square-rooting methods using high-speed multiplication, IEEE Trans. on Computers C-21 (Aug. 1972), 837-847

[3] D. L. Fowler and J. E. Smith, An accurate high speed implementation of division by reciprocal approximation, Proc. of 9th Symp. on Computer Arithmetic (Sep.1989), 60-67

[4] D. DasSarma and D. W. Matula, Measuring the accuracy of ROM reciprocal tables, IEEE Trans. on Computers, Vol.43, No.8(Aug.1994),932-940

[5] N. Anderson, Minimum relative error approximations for 1/t, Numerische Mathematic 54 (1988), 117-124

[6] M. Schlute and E. Swartzlander, Exact rounding of certain elementary functions, Proc. of 11th Symp. on Computer Arithmetic (Jun. 1993), 138-145

[7] M. Ito, N. Takagi, and S. Yajima, Efficient Initial Approximation Methods for Division and Square Using a Multiply-Add Unit, IPSJ SIG Notes 95-HPC-55-10 (Mar. 1995), 73-80

Appendix

Here we derive Algorithm LOC from the following continued fraction representation of $\sqrt{X}$:

$$\begin{aligned}
\sqrt{X} &= U_n + (\sqrt{X} - U_n) \\
&= U_n + \frac{X - U_n^2}{\sqrt{X} + U_n} \\
&= U_n + \frac{X - U_n^2}{(U_n + U_{n-1}) + (\sqrt{X} - U_{n-1})} \\
&= U_n + \frac{X - U_n^2}{(U_n + U_{n-1}) + \frac{X - U_{n-1}^2}{\sqrt{X} + U_{n-1}}} \\
&= U_n + \frac{X - U_n^2}{(U_n + U_{n-1}) + \frac{X - U_{n-1}^2}{(U_{n-1} + U_{n-2}) + (\sqrt{X} - U_{n-2})}} \\
&= \cdots
\end{aligned} \tag{33}$$

where U_i's $(0 \leq i \leq n)$ are arbitrary real numbers. Substituting $(\sqrt{X} - U_i = V_i)$ for all i's into (33) yields

$$V_i = \frac{X - U_i^2}{U_i + U_{i-1} + V_{i-1}}. \tag{34}$$

In order to set all the denominators in (34) equal to a certain constant K irrelevant to i, U's and V's should have the following relation:

$$\begin{aligned}
U_i &= K - U_{i-1} - V_{i-1} \quad \text{and} \\
V_i &= \frac{X - U_i^2}{K}, \\
&\text{where } \lim_{i \to \infty} (U_i + V_i) = \sqrt{X}.
\end{aligned} \tag{35}$$

An asymptotic equation of U can be obtained by eliminating V in (35).

$$\begin{aligned}
U_i &= K - U_{i-1} - \frac{X - U_{i-1}^2}{K}, \\
&\text{where } \lim_{i \to \infty} (U_i + \frac{X - U_i^2}{K}) = \sqrt{X}.
\end{aligned} \tag{36}$$

Substituting $S_i = \frac{U_i}{K} - \frac{1}{2}$ into (36) yields

$$S_i = S_{i-1}^2 + (\frac{1}{4} - \frac{X}{K^2}), \text{ where } \lim_{i \to \infty} (\frac{K}{2} - K \cdot S_i) = \sqrt{X}. \tag{37}$$

Setting $S_0 = 1/4 - X/K^2$ completes the derivation of the proposed algorithm.