논문 2018-13-06

K차 뉴톤-랍손 부동소수점수 N차 제곱근

(Kth order Newton-Raphson's Floating Point Number Nth Root)

조 경 연*
(Gyeong-Yeon Cho)

Abstract : In this paper, a tentative Kth order Newton-Raphson's floating point number Nth root algorithm for K order convergence rate in one iteration is proposed by applying Taylor series to the Newton-Raphson root algorithm. Using the proposed algorithm, $F^{-1/N}$ and $F^{-(N-1)/N}$ can be computed from iterative multiplications without division. It also predicts the error of the algorithm iteration and iterates only until the predicted error becomes smaller than the specified value. Since the proposed algorithm only performs the multiplications until the error gets smaller than a given value, it can be used to improve the performance of a floating point number Nth root unit.

Key Words : Floating point number Nth root, Kth order Newton-Raphson, Square root, Cubic root

Ⅰ. 서 론

부동소수점수 F의 N차 제곱근 $F^{1/N}(\sqrt[N]{F})$의 계산은 오래된 문제이다. 제곱근 ($\sqrt{F}$)은 과학 및 공학 기술 분야에서 많이 사용된다. 최근에는 로봇제어, 사물인터넷, 음성 처리 및 그래픽 분야 등 멀티미디어 분야에도 폭넓게 사용되면서 CPU의 기본 기능으로 채택되고 있다 [1, 2]. 뉴톤 알고리즘은 반복 연산으로 2차 수렴 (quadratic convergence) 하는 근을 구하는 방식으로 이를 적용한 제곱근 계산 방식이 많이 연구되고 있다 [3]. 또한 N차 제곱근을 구하는데도 뉴톤 알고리즘이 주로 사용된다 [4, 5].

부동소수점수 F의 N차 제곱근 $F^{1/N}$의 근 X의 i번째 근사값이 X_i일 때 X_{i+1}과 오차 ϵ_{i+1}은 2차 뉴톤 알고리즘에서 식 1과 식 2가 된다 [5].

$$X_{i+1} = X_i + \frac{1}{N}\left(\frac{F}{X_i^{N-1}} - X_i\right) \tag{1}$$

$$\epsilon_{i+1} \fallingdotseq \frac{N-1}{2\sqrt[N]{F}} \frac{X_i^2}{X_{i+1}} \left(\frac{X_i - \sqrt[N]{F}}{X_i}\right)^2 \tag{2}$$

IEEE-754 [6]로 정의되는 부동소수점수는 $F = 1.f_2 \times 2^p$으로 표현된다. 가수 부분과 지수 부분을 분리해서 N차 제곱근을 구하면 식 3이 된다.

$$\sqrt[N]{F} = \sqrt[N]{1.f} \times 2^{\frac{p}{N}} = \sqrt[N]{1.f \times 2^{p \bmod N}} \times 2^{\left(\frac{p}{N}\right)} \tag{3}$$

식 3에서 $(\frac{p}{N})$은 나눗셈 결과의 정수부분만을 취한다. $1.f \times 2^{p \bmod N}$이 식 1과 식 2의 F이다. 근의 수렴속도와 오차는 F에 비례하므로 본 논문에서는 식 4로 F차 제곱근을 구한다.

$$\sqrt[N]{F} = \sqrt[N]{1.f} \times 2^{\frac{p}{N}} = \sqrt[N]{1.f}\sqrt[N]{2^{p \bmod N}} \times 2^{\left(\frac{p}{N}\right)} \tag{4}$$

뉴톤 알고리즘을 3차, 4차 등 고차 (high order) 수렴으로 확장한 연구들이 있다 [7, 8]. 이들 종래 방식은 CPU에서 나눗셈과 곱셈의 연산속도가 동일하다는 가정을 근거로 하고 있다 [5]. 그러나 실제는 부동소수점 나눗셈은 곱셈에 비하여 대단히 느리다 [9]. 그러므로 나눗셈을 사용하지 않는 알고리즘에 대한 연구가 요구된다.

곱셈만을 사용하여 나눗셈과 제곱근을 구하는

*Corresponding Author (gycho@pknu.ac.kr)
Received: July 5 2017, Revised: Nov. 16 2017, Accepted: Nov. 29 2017.
G. Cho: Pukyong National University
* 이 논문은 부경대학교 자율창의학술연구비 (2017년)에 의하여 연구되었음
 2018 Feb. 13(1) 45-51
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2018.13.1.45

방식은 뉴톤-랍손 방식, 골드스미트 방식 등이 있다. 이들 방식은 초기 근사값을 설정하고, 반복 연산해서 오차를 줄인다. 초기 근사 값이 가지는 최대 오차를 계산하고, 오차를 부동소수점에서 표현 가능한 최소값보다 작게 될 때까지 반복 연산을 수행하였다. 최대 오차만을 고려했기 때문에, 실제 구하고자 하는 결과 값에 도달했음에도 불구하고 가외의 연산을 수행하여 연산 속도를 저하시키는 단점이 있었다. [10, 11]은 가변 시간 알고리즘을 제안하여 이러한 문제를 개선했다. [12]는 테일러급수 정리로부터 K차 나눗셈 알고리즘을 제안하였다. K차 수렴 나눗셈 알고리즘은 한 번 반복에 K번 곱셈을 수행한다. 또한 알고리즘 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 수행한다.

본 논문에서는 뉴톤-랍손 제곱근 알고리즘에 테일러급수를 적용하여 N차 제곱근을 구하는 가칭 K차 뉴톤-랍손 N차 제곱근 알고리즘을 제안한다.

종래 뉴톤-랍손 알고리즘은 $\frac{1}{\sqrt[N]{F}}(F^{-1/N})$을 구하고, 이의 역수를 취하여 $\sqrt[N]{F}$을 구한다. 이는 역수를 구하는데 많은 시간이 소요되는 단점이 있다. 본 논문에서는 $F^{-1/N}$과 $F^{-(N-1)/N}$을 구하는 두 가지 알고리즘을 제안한다. 또한 다음 반복 연산에서의 오차를 예측하여 가변시간 연산이 가능하도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 가칭 K차 뉴톤-랍손 N차 역제곱근 알고리즘을 제안한다. 3장에서는 가칭 K차 뉴톤-랍손 $F^{-(N-1)/N}$ 알고리즘을 제안한다. 4장에서 결론을 맺는다.

Ⅱ. K차 뉴톤-랍손 N차 역제곱근 알고리즘

1. K차 뉴톤-랍손 N차 역제곱근 알고리즘

부동소수점 수 F의 N차 역제곱근 $X_n = \frac{1}{F^{1/N}}$은 초기값 X_0를 정의하고, 반복식으로 $X_i (i = 1,...,n)$을 구한다. IEEE-754로 규정되는 부동소수점 수 F는 $1.f_2 * 2^{p+bias}$이다. 가수부 $1.f_2$는 단정도실수에서 24 비트, 배정도실수에서는 53 비트이다. 역제곱근의 지수부 연산은 $\sqrt[N]{2^{p \bmod N}} \times 2^{(\frac{p}{N})}$를 계산하는 것이다. $\sqrt[N]{2^{p \bmod N}}$은 상수이므로 $\frac{1}{\sqrt[N]{1.f_2}}$을 계산한 후에 $\frac{1}{\sqrt[N]{2^{p \bmod N}}}$을 곱한다. $2^{(\frac{p}{N})}$은 가수부 처리와 별도의 하드웨어에 의해서 병렬적으로 처리하므로 본 논문에서는 생략한다.

부동소수점수 F의 가수부 $1.f_2$는 식 5와 같이 두 부분으로 나눌 수 있다.

$$1.f_2 = 1.g_2 + h_2 1.d = 1.g + h \tag{5}$$

식 (5)에서 g와 h의 길이를 각각 n_g및 n_h 비트로 정의한다. h는 '$0 \le h < 2^{-n_g}$'이고, h의 최대값은 '$h_{\max} = 2^{-n_g} - 2^{-n_g - n_h}$'이다. 반복식의 수렴 속도를 빠르게 하기 위해서 $X_0 = \frac{1}{1.g^{1/N}}$를 근사계산하여 테이블에 미리 작성해놓는다 [13]. 초기 근사값 X_0는 ROM에 저장하거나 또는 별도의 회로를 사용해서 산출하기도 한다. X_0는 $\frac{1}{1.g^{1/N}}$의 근사계산이므로 '$X_0 = \frac{1}{F^{1/N}} - \epsilon_0$이다. ϵ_0는 근사에 따른 초기 오차이다.

i번째 반복식에서 X_i는 $\frac{1}{F^{1/N}}$의 근사값으로 오차를 ϵ_i라고 하면 식 6이 된다.

$$X_i = \frac{1}{F^{1/N}} - \epsilon_i = \frac{1 - \epsilon_i F^{1/N}}{F^{1/N}} \tag{6}$$

$a_i = \frac{1 - F X_i^N}{N}$로 정의하고, 식 6에 대입하면 식 7이 성립한다. $\frac{1}{N}$은 상수이다.

$$F X_i^N = 1 - N a_i = F\left(\frac{1 - e_i F^{1/N}}{F^{1/N}}\right)^N = (1 - e_i F^{1/N})^N \tag{7}$$

식 7로부터 식 8이 성립한다.

$$\frac{1}{1 - \epsilon_i F^{1/N}} = \frac{1}{(1 - N a_i)^{1/N}} \tag{8}$$

식 8의 오른쪽 항을 테일러급수로 전개하면 식 9가 된다.

$$\frac{1}{(1-Na_i)^{1/N}} = \sum_{j=0}^{\infty} c_j a_i^j = \sum_{j=0}^{K-1} c_j a_i^j + \sum_{j=k}^{\infty} c_j a_i^j$$
$$= R_i^{(K)} + \epsilon_{Ki}$$
$$c_0 = 1, \quad c_1 = 1 \tag{9}$$
$$c_m = \frac{\prod_{j=1}^{m-1}(jN+1)}{m!}, \quad m \geq 2$$

식 9의 테일러급수항에서 초기 K항을 취한 것이 $R_i^{(K)}$이고, 나머지 항들이 ϵ_{Ki}이다. R_i^K는 $\frac{1}{1-\epsilon_i F^{1/N}}$의 K차 근사값이므로 이를 식 6에 대입하면 $\frac{1}{F^{1/N}}$의 i+1번째 근사값 X_{i+1}과 오차 ϵ_{i+1}가 식 10과 같이 구해진다.

$$X_{i+1} = X_i R_i^{(K)} = X_i \sum_{j=0}^{K-1} c_j a_i^j =$$
$$\frac{1}{F^{1/N}} - X_i \sum_{j=K}^{\infty} c_j a_i^j = \frac{1}{F^{1/N}} - \epsilon_{i+1}$$
$$\epsilon_{i+1} = \frac{1-\epsilon_i F^{1/N}}{F^{1/N}} \sum_{j=K}^{\infty} c_j a_i^j \fallingdotseq \tag{10}$$
$$\frac{c_K a_i^K}{F^{1/N}} < \frac{c_K (\epsilon_i F^{1/N})^K}{F^{1/N}} = c_K \epsilon_i^K F^{(K-1)/N}$$

식 10을 K=2,3,4에 대하여 간략하게 정리하면 식 11과 같다.

$$X_{i+1}^{(2)} = X_i(1+c_1 a_i) \fallingdotseq \frac{1}{F^{1/N}} - c_2 e_i^2 F^{1/N}$$
$$X_{i+1}^{(3)} = X_i(1+c_1 a_i + c_2 a_i^2) \fallingdotseq \frac{1}{F^{1/N}} - c_3 e_i^3 F^{2/N} \tag{11}$$
$$X_{i+1}^{(4)} = X_i(1+c_1 a_i + c_2 a_i^2 + c_3 a_i^3) \fallingdotseq \frac{1}{F^{1/N}} - c_4 e_i^4 F^{3/N}$$

$|a_i| = \left|\frac{1-FX_i^N}{N}\right| < 2^{-t}$이면 i+1번째 오차 ϵ_{i+1}는 식 12와 같이 된다.

$$e_{i+1}^{(k)} \fallingdotseq \frac{c_K a_i^K}{F^{1/N}} < c_K 2^{-Kt} \tag{12}$$

$|a_i| < 2^{-t}$인 조건은 소수점 이하 연속되는 '0' 또는 '1'의 개수가 t보다 많다는 것으로 간단한 하드웨어로 판단이 용이하다. i+1번째 오차 ϵ_{i+1}가 일정한 값 이하가 되면 반복연산을 종료할 수 있으므로 불필요한 연산을 줄일 수 있다.

2. K차 뉴톤-랍손 제곱근 알고리즘

$\sqrt{F} = \frac{F}{\sqrt{F}}$이므로 F의 제곱근은 F의 역제곱근 $\frac{1}{\sqrt{F}}$에 F를 곱하면 된다. K차 역제곱근의 i+1번째 근사값 X_{i+1}을 K=2,3,4에 대하여 간략하게 정리하면 식 12와 같다.

$$a_i = \frac{1-FX_i^2}{2}$$
$$X_{i+1}^{(2)} = X_i(1+a_i) \fallingdotseq \frac{1}{\sqrt{F}} - \frac{3\epsilon_i^2\sqrt{F}}{2}$$
$$X_{i+1}^{(3)} = X_i\left(1+a_i+\frac{3a_i^2}{2}\right)) \fallingdotseq \frac{1}{\sqrt{F}} - \frac{5\epsilon_i^3 F}{2} \tag{12}$$
$$X_{i+1}^{(4)} = X_i\left(1+a_i+\frac{3a_i^2}{2}+\frac{5a_i^3}{2}\right)) \fallingdotseq \frac{1}{\sqrt{F}} - \frac{35\epsilon_i^4 F\sqrt{F}}{8}$$

식 12에서 $X_{i+1}^{(2)} = X_i\left(\frac{1-FX_i^2}{2}\right)$로 뉴톤-랍손 역제곱근 알고리즘 [11]이다.

식 12로부터 $|a_i| < 2^{-t}$이면 i+1번째 오차 ϵ_{i+1}은 식 13이 된다.

$$e_{i+1}^{(2)} \fallingdotseq \frac{3a_i^2}{2\sqrt{F}} < \frac{3}{2\sqrt{F}} 2^{-2t} < 2^{-2t+1}$$
$$e_{i+1}^{(2)} \fallingdotseq \frac{3a_i^2}{2\sqrt{F}} < \frac{3}{2\sqrt{F}} 2^{-2t} < 2^{-2t+1} \tag{13}$$
$$e_{i+1}^{(4)} \fallingdotseq \frac{35a_i^4}{8\sqrt{F}} < \frac{35}{8\sqrt{F}} 2^{-4t} < 2^{-4t+3}$$

3. 제곱근 오차 보정

식 12로 구한 X_n은 근사값으로 ' $X_n = \frac{1}{\sqrt{F}} - \epsilon_n$, $1 < F < 2$'이다. F의 유효자릿수를 w라고 하면, $\epsilon_n < 2^{-w-3}$이 될 때까지 반복연산을 수행한다. $0.75 * 2^{-w} \leq s < 2^{-w}$를 F에 더하여 X_n에 곱하면 식 14가 된다.

$$D = (F+s)X_n = \sqrt{F} - e_n F + sX_n = \sqrt{F} + e_d$$
$$= 1.f_1 + 0.f_2 * 2^{-w} + e_d \tag{14}$$

식 14에서 '$0 \leq (0.f_2 * 2^{-w} + e_d) < 2^{-w+1}$'이므로 식 15가 성립한다.

$$D = 1.m_1 + 0.m_2 * 2^{-w}$$
$$(1.m_1)^2 = 1.t_1 + 0.t_2 * 2^{-w} \quad (15)$$

식 15는 다음의 4가지 경우로 구분된다.

1. $1.t_1 = F$이고 $0.t_2 = 0$이면 $\sqrt{F} = 1.m_1$이고 스티키(sticky) 비트는 0이다.
2. $1.t_1 = F$이고 $0.t_2 \neq 0$이면 $\sqrt{F} = 1.m_1 - 2^{-w}$이고 스티키 비트는 1이다.
3. $1.t_1 < F$이면 $\sqrt{F} = 1.m1$이고 스티키 비트는 1이다.
4. $1.t_1 > F$이면 $\sqrt{F} = 1.m_1 - 2^{-w}$이고 스티키 비트는 1이다.

Ⅲ. K차 뉴톤-랍손 $F^{-(N-1)/N}$ 알고리즘

1. K차 뉴톤-랍손 $F^{-(N-1)/N}$ 알고리즘

초기 근사값으로 $X_0 = \frac{1}{1.g^{(N-1)/N}}$를 설정한다. i번째 반복식에서 X_i는 $\frac{1}{F^{(N-1)/N}}$의 근사값으로 오차를 ϵ_i라고 하면 식 16이 된다.

$$X_i = \frac{1}{F^{(N-1)/N} - \epsilon_i} = \frac{1}{F^{(N-1)/N}(1 - \epsilon_i F^{(N-1)/N})} \quad (16)$$

$a_i = \frac{1 - F^{N-1} X_i^N}{N}$로 정의하고, 식 16에 대입하면 식 17이 성립한다.

$$F^{N-1} X_i^N = 1 - N a_i = F^{N-1}\left(\frac{1 - e_i F^{(N-1)/N}}{F^{(N-1)/N}}\right)^N = (1 - e_i F^{(N-1)/N})^N \quad (17)$$

식 17로부터 식 18이 성립한다.

$$\frac{1}{1 - \epsilon_i F^{(N-1)/N}} = \frac{1}{(1 - N a_i)^{1/N}} \quad (18)$$

식 18의 오른쪽 항을 테일러급수로 전개하면 식 19가 된다.

$$\frac{1}{(1 - N a_i)^{1/N}} = \sum_{j=0}^{\infty} c_j a_i^j = \sum_{j=0}^{K-1} c_j a_i^j + \sum_{j=k}^{\infty} c_j a_i^j = R_i^{(K)} + \epsilon_{Ki}$$
$$c_0 = 1, \quad c_1 = 1$$
$$c_m = \frac{\prod_{j=1}^{m-1}(jN+1)}{m!}, \quad m \geq 2 \quad (19)$$

식 19의 테일러급수항에서 초기 K항을 취한 것이 $R_i^{(K)}$이고, 나머지 항들이 ϵ_{Ki}이다. R_i^K는 $\frac{1}{1 - \epsilon_i F^{(N-1)/N}}$의 K차 근사값이므로 이를 식 16에 대입하면 $\frac{1}{F^{(N-1)/N}}$의 i+1번째 근사값 X_{i+1}과 오차 ϵ_{i+1}가 식 20과 같이 구해진다.

$$X_{i+1} = X_i R_i^{(K)} = X_i \sum_{j=0}^{K-1} c_j a_i^j = \frac{1}{F^{(N-1)/N}} - X_i \sum_{j=K}^{\infty} c_j a_i^j = \frac{1}{F^{(N-1)/N}} - e_{i+1}$$
$$e_{i+1} = \frac{1 - e_i F^{(N-1)/N}}{F^{(N-1)/N}} \sum_{j=k}^{\infty} c_j a_i^j \fallingdotseq \frac{c_K a_i^K}{F^{(N-1)/N}} < \frac{c_K (\epsilon_i F^{(N-1)/N})^K}{F^{(N-1)/N}} = c_K \epsilon_i^K F^{(K-1)(N-1)/N} \quad (20)$$

식 20을 K=2,3,4에 대하여 간략하게 정리하면 식 21과 같다.

$$X_{i+1}^{(2)} = X_i(1 + c_1 a_i) \fallingdotseq \frac{1}{F^{(N-1)/N}} - c_2 e_i^2 F^{(N-1)/N}$$
$$X_{i+1}^{(3)} = X_i(1 + c_1 a_i + c_2 a_i^2) \fallingdotseq \frac{1}{F^{(N-1)/N}} - c_3 e_i^3 F^{2(N-1)/N} \quad (21)$$
$$X_{i+1}^{(4)} = X_i(1 + c_1 a_i + c_2 a_i^2 + c_3 a_i^3) \fallingdotseq \frac{1}{F^{(N-1)/N}} - c_4 e_i^4 F^{3(N-1)/N}$$

$|a_i| = \left|\frac{1 - F^{N-1} X_i^N}{N}\right| < 2^{-t}$이면 i+1번째 오차 ϵ_{i+1}는 식 22와 같이 된다.

$$e_{i+1}^{(K)} \fallingdotseq \frac{c_K a_i^K}{F^{(N-1)/N}} < c_K 2^{-Kt} \quad (22)$$

2. K차 뉴톤-랍손 $\sqrt[3]{F}$ 알고리즘

$\sqrt[3]{F}=\frac{F}{F^{2/3}}$이므로 F의 3차 제곱근은 $F^{-2/3}$에 F를 곱하면 된다. $F^{-2/3}$의 i+1번째 근사값 X_{i+1}을 K=2,3,4에 대하여 간략하게 정리하면 식 23과 같다.

$$
\begin{aligned}
a_i &= \frac{1-F^2X_i^3}{3} \\
X_{i+1}^{(2)} &= X_i(1+a_i) \fallingdotseq \frac{1}{F^{2/3}} - 2e_i^2F^{2/3} \\
X_{i+1}^{(3)} &= X_i(1+a_i+2a_i^2) \fallingdotseq \frac{1}{F^{2/3}} - \frac{14e_i^3F^{4/3}}{3} \\
X_{i+1}^{(4)} &= X_i(1+a_i+2a_i^2+\frac{14a_i^3}{3}) \fallingdotseq \frac{1}{F^{2/3}} - \frac{35e_i^4F^2}{3}
\end{aligned}
\tag{23}
$$

식 22로부터 $|a_i|<2^{-t}$이면 i+1번째 오차 ϵ_{i+1}은 식 24가 된다.

$$
\begin{aligned}
e_{i+1}^{(2)} &\fallingdotseq \frac{2a_i^2}{F^{2/3}} < \frac{2}{F^{2/3}}2^{-2t} < 2^{-2t+1} \\
e_{i+1}^{(3)} &\fallingdotseq \frac{14a_i^3}{3F^{2/3}} < \frac{14}{3F^{2/3}}2^{-3t} < 2^{-3t+2} \\
e_{i+1}^{(4)} &\fallingdotseq \frac{7a_i^4}{3F^{2/3}} < \frac{35}{3F^{2/3}}2^{-4t} < 2^{-4t+4}
\end{aligned}
\tag{24}
$$

Ⅳ. 결 론

부동소수점수 F의 N차 제곱근은 공학 분야에서 폭넓게 사용되고 있다. 제곱근 (square root) 및 3차 제곱근 (cubic root)의 사용빈도는 높지 않으나 계산 시간이 오래 걸리므로 전용 회로를 기본 기능으로 채택한 CPU가 늘고 있다.

본 논문에서는 뉴톤-랍손 역제곱근 알고리즘을 테일러 급수를 적용하여 확장하여 가칭 K차 뉴톤-랍손 N차 제곱근 알고리즘을 제안하였다. 제안한 알고리즘은 $F^{-1/N}$과 $F^{-(N-1)/N}$을 반복 곱셈으로 계산한다.

i번째 반복식에서 X_i는 $\frac{1}{F^{1/N}}$의 근사값으로 오차를 ϵ_i라고 하면 '$X_i=\frac{1}{F^{1/N}}-\epsilon_i$'이 된다. $a_i=\frac{1-FX_i^N}{N}$를 정의하고, $\frac{1}{1-\epsilon_iF^{1/N}}$를 테일러급수로 전개하면 '$\sum_{j=0}^{K-1}c_ja_i^j+\sum_{j=k}^{\infty}c_ja_i^j=R_i^{(K)}+\epsilon_{Ki}$'이 된다. $c_0=1,\ \ c_1=1,\ c_m=\frac{\prod_{j=1}^{m-1}(jN+1)}{m!},\ \ m\geq 2$이다. 테일러급수항에서 초기 K항을 취한 것이 $R_i^{(K)}$이고, 나머지 항들이 ϵ_{Ki}이다. R_i^K는 $\frac{1}{1-\epsilon_iF^{1/N}}$의 K차 근사값이므로 $X_{i+1}=X_iR_i^{(K)},\ \epsilon_{i+1}=\frac{1-\epsilon_iF^{1/N}}{F^{1/N}}\sum_{j=K}^{\infty}c_ja_i^j$가 된다.

또한 i번째 반복식에서 X_i는 $\frac{1}{F^{(N-1)/N}}$의 근사값으로 오차를 ϵ_i라고 하면 '$X_i=\frac{1}{F^{(N-1)/N}}-\epsilon_i$'이 된다. $a_i=\frac{1-F^{N-1}X_i^N}{N}$를 정의하고, $\frac{1}{1-\epsilon_iF^{(N-1)/N}}$를 테일러급수로 전개하면 '$\sum_{j=0}^{K-1}c_ja_i^j+\sum_{j=k}^{\infty}c_ja_i^j=R_i^{(K)}+\epsilon_{Ki}$'이 된다. 테일러급수항에서 초기 K항을 취한 것이 $R_i^{(K)}$이고, 나머지 항들이 ϵ_{Ki}이다.

R_i^K는 $\frac{1}{1-\epsilon_iF^{(N-1)/N}}$의 K차 근사값이므로 $X_{i+1}=X_iR_i^{(K)},\ \ \epsilon_{i+1}=\frac{1-\epsilon_iF^{(N-1)/N}}{F^{(N-1)/N}}\sum_{j=K}^{\infty}c_ja_i^j$가 된다.

본 논문에서 제안한 알고리즘은 곱셈만을 사용하므로 하드웨어 구현이 용이하다. 또한 a_i로부터 ϵ_{i+1}을 예측할 수 있으므로 유효자릿수까지만 연산할 수 있으므로 불필요한 연산을 줄일 수 있다. 종래 뉴톤-랍손 역수 제곱근 알고리즘에서 단정도실수 역수 제곱근은 '192x7' 테이블을 사용하면 9회의 곱셈을 수행하였으나 [11], 본 논문에서 제안한 알고리즘에서는 평균 5.7회의 곱셈으로 계산할 수 있었다. 배정도실수 역수 제곱근은 종래 알고리즘에서는 '192X7' 테이블을 사용하면 12회의 곱셈을 수행하였으나 [11] 본 논문에서 제안한 알고리즘에서는 평균 8.9회의 곱셈으로 계산할 수 있었다.

References

[1] V. Lappalainen, T.D. Hamalainen, P. Liufa,

"Overview of Research Efforts on Media ISA Extension and Their Usage in Video Coding," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, pp. 660-670, 2002.

[2] R.B. Lee, "Multimedia Extensions for General Purpose Processor," Proceedings of IEEE Workshop on Signal Processing Systems, pp. 9-23, 1997.

[3] C. Shuang-yan, W. Dong-hui, Z. Tie-jun H. Chao-huan, "Design and Implementation of a 64/32-bit Floating-point Division, Reciprocal, Square Root, and Inverse Square Root Unit," Proceedings of IEEE International conference on Solid-State and Integrated Circuit Technology, pp. 1976-1979, 2006.

[4] F. Dubeau, "nth Root extraction: Double iteration process and Newtons's method," Journal of Computational and Applied Mathematics, Vol. 91, pp. 191-198, 1998.

[5] S.G. Chen, P.Y. Hsieh, "Fast Computation of the Nth Root," Computers & Mathematics Applications, Vol. 17, No. 10, pp. 1423-1427, 1989.

[6] IEEE, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard, Std. 754-1985.

[7] J.M. Gutierrez, M.A. Hernandez, M.A. Salanova, "Calculus of nth Roots and Third Order Iterative Methods," Nonlinear Analysis: Theory, Methods & Applications, Vol. 47, No. 4, pp. 2875-2880, 2001.

[8] F. Dubeau, "Newton's Method and High-order Algorithm for the n-th Root Computation," Journal of Computational and Applied Mathematics, Vol. 224, No. 1, pp. 66-76, 2009.

[9] S.F. Oberman, M.J. Flynn, "Design Issues in Division and Other Floating Point Operations," IEEE Transactions on Computer, Vol. C-46, pp. 154-161, 1997.

[10] S. Kim, H. Song, G. Cho, "A Variable Latency Goldschmidt's Floating Point Number Square Root Computation," Korea Institute of Maritime Information and Communication Sciences, Vol. 9, No. 1, pp. 188-198, 2004 (in Korean).

[11] S. Kim, G. Cho, "A Variable Latency Newton-Rapson's Floating Point Number Reciprocal Square Root Computation," Korea Information Processing Society, Vol. 12-A, pp. 413-420, 2005 (in Korean).

[12] G. Cho, "A Variable Latency K'th Order Newton-Raphson's Floating Point Number Divider," Institute of Embedded Engineering of Korea, Vol. 9, No. 4, pp. 285-292, 2014 (in Korean).

[13] D. DasSarma, D. Matula, "Measuring and Accuracy of ROM Reciprocal Tables," IEEE Transactions on Computer, Vol. 43, No. 8, pp. 932-930, 1994.

Gyeong-Yeon Cho (조 경 연)

He received the B.S. degree in Electronics Engineering from Inha University in 1981, the M.S. degree in Electronics Engineering from Inha University in 1983, and Ph.D. degree in Electronics Engineering from Inha University in 1990. During 1983-1991, he worked in Trigem computer. During 1991-current, he is a professor in the Department of IT Convergence and Application Engineering in Pukyong National University. His research interests include computer architecture, ASIC design, and Cryptographic chip design.

Email : gycho@pknu.ac.kr