논문 2014-09-35

가변 시간 K차 뉴톤-랍손 부동소수점 나눗셈

(A Variable Latency K'th Order Newton-Raphson's Floating Point Number Divider)

조 경 연*
(Gyeong-Yeon Cho)

Abstract : The commonly used Newton-Raphson's floating-point number divider algorithm performs two multiplications in one iteration. In this paper, a tentative K'th Newton-Raphson's floating-point number divider algorithm which performs K times multiplications in one iteration is proposed. Since the number of multiplications performed by the proposed algorithm is dependent on the input values, the average number of multiplications per an operation in single precision and double precision divider is derived from many reciprocal tables with varying sizes. In addition, an error correction algorithm, which consists of one multiplication and a decision, to get exact result in divider is proposed. 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 divider unit. Also, it can be used to construct optimized approximate reciprocal tables.

Keywords : Error correction, Floating point division, K'th order Newton-Raphson, Variable latency

Ⅰ. 서론

부동소수점 계산은 과학 및 공학 기술 분야에서 많이 사용된다. 최근에는 음성 처리 및 3차원 그래픽 분야 등 멀티미디어 분야에도 폭넓게 사용되면서 CPU의 기본 기능으로 채택되고 있다[1]. 멀티미디어 자료 처리를 위해서 종래는 정수 연산을 많이 사용했지만 최근에는 부동소수점 연산이 요구되고 있다[2]. 나눗셈은 덧셈, 뺄셈 및 곱셈보다 출현 빈도가 낮지만, Oberman과 Flynn의 연구[3]는 나눗셈의 수행 시간이 덧셈이나 곱셈과 비슷하게 소요됨을 보이고 있다. 특히 2D와 3D 그래픽 응용에서 나눗셈의 출현 빈도가 높으므로 나눗셈기의 수행 속도를 높이는 연구가 요구되고 있다.

*Corresponding Author(gycho@pknu.ac.kr)
Received: 20 Feb. 2014, Revised: 29 Apr. 2014, 3 June 2014, Accepted: 18 June 2014.
G.Y. Cho: Pukyong National University
※ 본 논문은 부경대학교 자율창의학술연구비(2014년)에 의하여 연구되었음(C-D 2014-0329)
 2014 Aug.: 9(5) 285-292
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2014.9.5.285

부동소수점 나눗셈은 뺄셈을 반복하는 SRT[4, 5] 알고리즘과 곱셈을 이용한 알고리즘으로 뉴톤-랍손(Newton-Raphson) 역수 알고리즘 및 골드스미트(Goldschmidt) 나눗셈 알고리즘이 있다. 곱셈을 이용하는 방식은 SRT와 비교하여 속도가 빠르고 추가적인 하드웨어가 크지 않다는 장점이 있으나 근사 값만을 얻는다.

Shuang[6]은 작은 곱셈기를 직렬로 연결하여 뉴톤-랍손 방식으로 부동소수점의 나눗셈, 역수, 제곱근, 역제곱근을 구하는 방식을 제안하였다. 곱셈을 이용한 알고리즘은 초기 근사 값을 이용하여 수렴 속도를 향상시키고 있다. 그러므로 초기 근사 값을 결정하는 알고리즘과 수렴 속도를 높이기 위한 방식에 대하여 연구가 진행되었다[7-9].

Agrawak[10]은 뉴톤-랍손 알고리즘에서 초기 값을 테일러급수 정리로 계산하는 방식을 제안하였다. 이를 통하여 초기값을 저장하는 ROM의 크기를 줄일 수 있었다.

종래 연구는 초기 근사 값이 가지는 최대 오차를 계산하고, 오차를 부동소수점에서 표현 가능한

최소값보다 작게 될 때까지 반복 연산을 수행하였다. 이러한 종래 연구에서는 최대 오차만을 고려했기 때문에, 실제 구하고자 하는 결과 값에 도달했음에도 불구하고 가외의 연산을 수행하여 연산 속도를 저하시키는 단점이 있었다. 김성기[11]는 가변 시간 알고리즘을 제안하여 이러한 문제를 개선했으나 한 회 반복에 항상 두 번의 곱셈을 사용했으므로 가외의 연산이 남아있으며 또한 근사 값만을 얻을 수 있었다.

본 논문에서는 테일러급수 정리로부터 가칭 K차 뉴톤-랍손 부동소수점 역수 알고리즘을 유도한다. K차 뉴톤-랍손 부동소수점 역수 알고리즘은 한 번 반복에 K번 곱셈을 수행한다. 또한 알고리즘 반복 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 수행하는 알고리즘을 제안한다.

한편 뉴톤-랍손 알고리즘을 사용한 나눗셈은 근사계산으로 정확한 결과를 얻기 위해서 보정이 필요하다. Anderson[12]은 요구되는 정밀도보다 10 비트를 더 계산하는 것을 제안했으나 항상 정확한 값을 보장하지는 못한다. Markstein[13]은 요구되는 정밀도의 2배 길이 계산을 제안했으나 이 또한 항상 정확한 값을 보장하지는 못한다. Schwarz[14]가 제안한 방식은 스티키 비트를 정확히 산출하지 못한다.

본 논문에서는 부동소수점 나눗셈 $\frac{N}{1.d}$에서 제수 $1.d$의 역수로 '$1.d*0.y \le 1+2^{-w}$'이 되는 $0.y$를 K차 뉴턴-랍손 역수 알고리즘으로 계산하고, 이를 피제수 N에 곱하여 '$N*0.g = Mm$'을 계산하고 그 결과를 보정하여 정확한 나눗셈 값을 산출한다. 이를 K차 뉴톤-랍손 나눗셈 알고리즘이라 가칭한다.

본 논문에서 제안한 알고리즘은 C 모델링하여 정확한 값이 계산되는 것을 검증하였고, Verilog HDL로 설계하고 로직 시뮬레이션하여 동작을 검증하였다.

본 논문의 구성은 다음과 같다. 2장에서는 테일러급수 정리로부터 가칭 K차 뉴톤-랍손 부동소수점 역수 알고리즘을 유도하고, 나눗셈 결과 보정 알고리즘을 제안하고, 반복 연산에서의 오차를 예측하는 방법을 제안하고, 연산 자릿수 및 반복을 종료할 오차 한계를 계산한다. 3장에서는 제안한 알고리즘을 구현하는 하드웨어 알고리즘을 제시한다. 4장에서는 근사 테이블을 구성하고, 나눗셈 계산에 소요되는 평균 곱셈 횟수를 계산한다. 그리고 그 결과를 종래 뉴톤-랍손 알고리즘과 비교 분석한다. 5장에서 결론을 맺는다.

Ⅱ. K차 뉴톤-랍손 나눗셈 알고리즘

1. K차 뉴톤-랍손 역수 알고리즘

부동소수점 수 D의 역수 X_n은 초기값 X_0를 정의하고, 반복식으로 $X_i (i=1,...,n)$을 구한다. IEEE-754[15]로 규정되는 부동소수점 수 D는 $1.d_2 * 2^{n+base}$이다. 가수부 $1.d_2$는 단정도실수에서 24 비트, 배정도실수에서는 53 비트이다. 역수의 지수부 연산은 '-n+ base'를 계산하는 것으로 가수부 처리와 별도의 하드웨어에 의해서 병렬적으로 처리하므로 본 논문에서는 생략한다.

부동소수점 수 D의 가수부 1.d는 식 (1)과 같이 두 부분으로 나눌 수 있다.

$$1.d = 1.g + h \tag{1}$$

식 (1)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 $\frac{1}{1.g}$를 근사계산하여 테이블 $T(g)$를 미리 작성해놓는다. 근사 테이블은 ROM에 저장하거나 또는 별도의 회로를 사용해서 산출하기도 한다. $T(g)$는 $\frac{1}{1.g}$의 근사계산이므로 '$T(g) = \frac{1}{1.g} + e_t$'이다. e_t는 근사에 따른 오차이다. $T(g)$를 X의 초기 근사 값 X_0로 정의한다.

i번째 반복식에서 X_i는 $\frac{1}{D}$의 근사값으로 오차를 e_i라고 하면 식 (2)가 된다.

$$X_i = \frac{1}{D} - e_i = \frac{1 - e_i D}{D} \tag{2}$$

$a_i = 1 - DX_i$를 정의하면 $\frac{1}{D}$는 식 (3)으로 구해진다.

$$\frac{1}{D} = \frac{X_i}{1-e_i D} = \frac{X_i}{1-a_i} = X_i \sum_{j=0}^{\infty} a_i^j = X_i (R_i^{(k)} + e_{ri}) \tag{3}$$

여기서 $R_i^{(k)} = \sum_{j=0}^{k-1} a_i^j$, $e_{ri} = \sum_{j=k}^{\infty} a_i^j$이다.

$R_i^{(k)} \gg e_{ri}$이므로 X_{i+1}은 식 (4)와 같이 정의한다.

$$X_{i+1}^{(k)} = X_i R_i^{(k)} = \frac{1}{D} - e_{i+1}$$
$$e_{i+1} = \frac{1-e_i D}{D} \sum_{j=k}^{\infty} a_i^j \fallingdotseq \frac{a_i^k}{D} = e_i^k D^{k-1} \quad (4)$$

식 (4)에서 $k=2$이면 $X_{i+1}^{(2)} = X_i(1+a_i) = X_i(2-DX_i)$로 뉴톤-랍손 역수 알고리즘이 된다. 이에 본 논문에서는 식 (4)를 K차 뉴톤-랍손 역수 알고리즘으로 가칭한다.

2. 나눗셈 및 오차 보정

$Ceiling(\frac{N}{1.d}) = Q, \quad d \neq 0$을 연산하기 위하여 $\frac{1}{1.d} \fallingdotseq 0.y, \quad 0.y * 1.d = 1 + e, \quad e < 2^{-w}$가 되는 $0.y$를 구한다. w는 워드 길이로 '$2^{w-1} \leq N < 2^w$'이다. $0.y$은 식 (4)에서 $e_{i+1} < 2^{-w-1}$이 되는 $X_{i+1} = X_n$을 계산하고, $0.y = X_n + 2^{-w-1}$으로 구한다. N에 $0.y$을 곱하여 정리하면 식 (5)가 된다.

$$N * 0.y = \frac{N}{1.d} + \frac{eN}{1.d} = Q.q + \frac{eN}{1.d} = M.m \quad (5)$$

식 (5)에서 '$0 \leq 0.q + \frac{eN}{1.d} < 2$'이므로 '$Q \leq N * 0.q < Q+2$'이 되고, 따라서 '$Q \leq M \leq Q+1$'이 성립한다. 이제 '$M * 1.d = T.t$'를 계산하면, 다음의 3가지 경우가 생긴다.

1) '$T = N$'이고 '$t = 0$'인 경우 -- '$Q = M, q = 0$'이다. 스티키 비트는 '0'이 된다.
2) '$T = N-1$' 또는 '$T = N-2$'인 경우 -- '$Q = M, q \neq 0$'이다. 스티키 비트는 '1'이 된다. '$Q * 1.d = (Q.q - 0.q) * 1.d = N - 0.q * 1.d$'인 경우이다.
3) '$T = N+1$'이거나 '$T = N, t \neq 0$'인 경우 -- '$Q = M-1, q \neq 0$'이다. 스티키 비트는 '1'이 된다.

$(Q+1) * 1.d = (Q.q + 0.q') * 1.d = N + 0.q' * 1.d$인 경우이다. '$Q = M-1, q = 0$'인 경우는 발생하지 않는다.

3. 오차 분석

'$a_i = 1 - DX_i$'에서 뺄셈은 하드웨어 구현 시에 캐리 전달 지연이 발생한다. 이러한 문제점을 해결하기 위하여 본 논문에서는 식 (6)으로 a_i를 구한다.

$$a_i = 1 - 2^{-p} - DX_i \quad (6)$$

식 (6)에서 DX_i 곱셈은 소수점 이하 p 비트 미만을 절삭한다. 식 (4)의 K차 뉴톤-랍손 알고리즘을 반복 연산하면 오차가 누적된다. 그러므로 구하고자 하는 부동소수점의 정밀도보다 긴 자리수의 연산이 요구된다. a_i는 식 (7)이 된다.

$$a_i = 1 - 2^{-p} - (D(\frac{1}{D} - e_i) - u2^{-p})$$
$$= De_i - (1-u)2^{-p} \geq De_i - 2^{-p} \quad (7)$$

식(7)에서 $u2^{-p} (0 \leq u < 1)$는 곱셈 결과를 절삭하면서 발생하는 오차이며, '$u = 0$'에서 오차가 최대가 된다. 식 (7)과 식 (4)로부터 $X_{i+1}^{(2)}$는 식 (8)이 된다.

$$X_{i+1}^{(2)} = (\frac{1}{D} - e_i)(1 + De_i - 2^{-p}) - u2^{-p}$$
$$\leq \frac{1}{D} - De_i^2 - 2 * 2^{-p} \quad (8)$$

또한 $X_{i+1}^{(3)}$은 식(9)가 된다.

$$X_{i+1}^{(3)} = X_i(1 + a_i(1 + a_i))$$
$$\leq \frac{1}{D} - D^2 e_i^3 - 3 * 2^{-p} \quad (9)$$

식 (8)과 식 (9)로부터 반복 연산중에 절삭으로 발생하는 최대오차는 $3 * 2^{-p}$보다 작다. 그러므로 식 (4)의 e_{i+1}이 $3 * 2^{-p}$보다 작으면 알고리즘의 반복을 종료한다.

4. 오차 예측

식 (7)에서 '$a_i = 2^{-x}, \quad 2^{-x} \gg 2^{-p}$'이라 하면 식 (10)이 성립한다.

$$De_i^2 = \frac{2^{-2x} + 2^{-x-p+1} + 2^{-2p}}{D}$$
$$< \frac{2^{-2x+1} + 2^{-p}}{D} \leq 2^{-2x+1} + 2^{-p}, \quad D \geq 1 \quad (10)$$

$X_{i+1}^{(2)}$에서 $De_1^2 < 2 * 2^{-p}$이면 반복식을 종료하는 조건을 식 (10)에 대입하면, 식 (11)이 성립한다.

$$2^{-x} < 2^{-\frac{p+1}{2}} < 2^{\frac{-p}{2}} \quad (11)$$

표 1. 유효자릿수. 단위는 비트
Table 1. Significant digit. Unit is bit.

	Single precision	Double precision
w	24	53
p	29	58
$x^{(2)}$	15	29
$x^{(3)}$	10	20

'$3x \geq p$'이면 식 (10)의 양변에 '$De_i = 2^{-x} + 2^{-p}$'를 곱해서 정리하면 식 (12)가 된다.

$$D^2 e_i^3 < (2^{-2x+1} + 2^{-p})(2^{-x} + 2^{-p}) < 2^{-3x+2} + 2^{-2p} < 2^{-3x+2} + 2^{-p} \quad (12)$$

$X_{i+1}^{(3)}$에서 $D^2 e_1^3 < 3 * 2^{-p}$이면 반복식을 종료하는 조건을 식 (12)에 대입하면, 식 (13)이 성립한다.

$$2^{-x} < 2^{-\frac{p+1}{3}} < 2^{\frac{-p}{3}} \quad (13)$$

5. 연산 유효자릿수

식 (8)과 (9)에서 De_i^2와 $D^2 e_i^3$는 반복 알고리즘에 따른 오차이고, $2*2^{-p}$와 $3*2^{-p}$는 연산 과정의 절삭으로 발생하는 오차이다. 반복 알고리즘 오차가 절삭 오차보다 작으면 더 이상 반복하는 것은 의미가 없다. IEEE-754 단정도실수와 배정도실수에서 가수부의 유효자릿수는 각각 24 비트와 53비트이다. 유효자릿수에 라운드 한 비트를 더하면 25 비트와 54비트이다. $3*2^{-p}$가 2^{-25} 및 2^{-54}보다 작아야 한다. 또한 연산 중간에 음수가 발생하므로 사인 비트와 나눗셈 오차 보장을 위해 각각 한 비트가 추가로 필요하다. 따라서 단정도실수와 배정도실수에서 p는 각각 29과 58이다. 연산 유효자릿수를 표 1에 보인다. 표 1에서 $x^{(2)}$와 $x^{(3)}$은 식 (11)과 식 (12)에서 오차 예측을 위한 x이다.

Ⅲ. 가변 시간 K차 뉴톤-랍손 나눗셈 계산기

하드웨어 구현을 위한 가변 시간 K차 뉴톤-랍손 나눗셈 알고리즘을 표 2에 보인다. 표 2에서 $\frac{N}{D} = Q,\ 1 < D = 1.g + h < 2,\ 2^{w-1} \leq N < 2^w$ 이다.

표 2. 가변 시간 K차 뉴톤-랍손 나눗셈 알고리즘
Table 2. Variable latency K'th order Newton-Raphson's division algorithm

(State-1)
  Reciprocal table T(1.g) => X;
(State-2)
  $1 - 2^{-p} - DX => A$;
  No. of Leading bits
    after period of A $=> B$;
  If $B \geq x^{(2)}$ OR $B < x^{(3)}$,
    then goto state-4 ;
(state-3)
  $A(1+A) => A$;
(state-4)
  $X(1+A) => X$;
  If $B < x^{(3)}$, then goto state-2 ;
    else $X + 2^{-w-1} => X$;
(state-5)
  $X(N \ll 2) => X$;
(state-6)
  $XD => Tt$;
  $T \wedge 3 => r$;
(state-7)
  If r==0 AND t==0,
    then {$(X \gg 2) => Q$, 0=>ST } ;
  If r==1 OR (r==0 AND t!=0),
    then {$(X \gg 2) - 1 => Q$, 1=>ST} ;
  If r>1,
    then {$(X \gg 2) => Q$, 1=>ST } ;

표 2에서 상태-1부터 상태-4에서 D의 근사 역수 X를 구한다. 상태-0에서 $1.g$의 근사 역수 X_0를 테이블로부터 읽어서 레지스터 X에 저장한다. 상태-1에서 식 (6)의 a_i를 계산하여 레지스터 A에 저장한다. 또한 A의 소수점 이하부터 연속해서 나타나는 '0' 또는 '1' 비트의 수를 세서 레지스터 B에 저장한다. 하드웨어 설계시에 B는 $x^{(2)}$보다 큰 경우와 $x^{(3)}$보다 작은 경우만이 참조되므로 $x^{(2)}$ 비트 입력 AND 게이트와 OR 게이트, $x^{(3)}$ 비트 입력 AND 게이트와 OR 게이트로 구현한다. B가 $x^{(2)}$보다 크면 $X_{i+1}^{(2)}$이 구하려는 근사 역수이므로 상태-4로 전이한다. 또한 B가 $x^{(3)}$보다 작으면 $X_{i+1}^{(2)}$을 구하고 반

복 연산을 해야 하므로 상태-4로 전이한다. 상태-3은 B가 $x^{(2)}$보다 작으면 $x^{(3)}$보다 큰 경우로 $X_{i+1}^{(3)} = X_i(1+a_i(1+a_i))$가 구하려는 근사 역수이므로 $a_i(1+a_i)$를 연산하여 레지스터 A에 저장하고, 상태-4로 전이한다. 상태-4에서 $X_{i+1}^{(2)}$ 또는 $X_{i+1}^{(3)}$을 계산하여 레지스터 X에 저장한다. 레지스터 B가 $x^{(3)}$보다 작으면 반복 연산이 필요하므로 상태-2로 전이하고, 크면 2^{-w-1}을 곱셈 결과에 더해서 근사 역수 $0.y$를 구해서 레지스터 X에 저장하고, 반복 연산을 종료한다. 2^{-w-1}을 더하는 것은 곱셈 과정 중의 부분곱 행에 한 비트를 추가하는 것이므로 추가적인 지연이 발생하지 않으며, 추가되는 회로 또한 극히 미미하다.

상태-5에서는 N을 왼쪽으로 2비트 이동시키고 $0.y$를 곱해서 식 (5)의 M을 계산하여 레지스터 X에 저장한다. 2비트 왼쪽으로 이동시킨 이유는 2.2절의 조건 $T=N-1$, $T=N-2$, $T=N$, $T=N-1$을 T의 비트 0와 비트 1로 판별할 수 있기 때문이다. 상태-6에서 '$M*1.d=T.t$'를 계산하여 레지스터 T와 t에 저장한다. 또한 레지스터 T의 하위 2비트를 레지스터 r에 저장한다. 상태-7에서 2.2절의 조건을 판별하여 나눗셈 결과 Q와 스티키 비트 ST를 구한다.

제시한 알고리즘은 C 언어로 모델링하였다. 단정도실수에서 상태-0부터 상태-4까지의 근사 역수 $0.y$를 전수 계산하여 SRT로 계산한 결과와 비교하여 일치하는 것을 확인하였다. 단정도실수와 배정도실수 각각에서 SHA 해쉬 함수[16]를 사용하여 N과 D 10^7 개를 생성하고, 제시한 알고리즘으로 나눗셈을 수행하고, 그 결과를 SRT로 계산한 결과와 비교하여 일치하는 것을 확인하였다.

IBM-PC의 window-7에서 Icarus Verilog ver. 0.9를 사용하여 HDL로 코딩하고 로직 시뮬레이션하여 제시한 알고리즘을 검증하였다.

Ⅳ. 연구 결과 및 분석

DasSarma[17]의 연구 결과 최적의 근사 역수는 식 (14)로 주어진다.

$$T(g) = \frac{1}{1.g} \fallingdotseq RN\left(\frac{1}{1.g + 2^{-2n_g-1}}\right) \tag{14}$$

여기서 RN은 round to nearest이다.

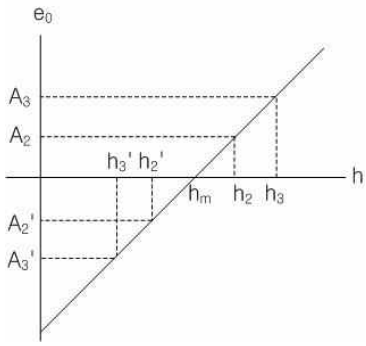

그림 1. h와 e_0의 그라프

Fig. 1 h versus e_0 graph

$T(g)$의 소수점 이하 길이를 t 비트라고 하면 '$T(g) = (b_0, b_1, ..., b_t)_2$, $0.5 < T(g) \le 1.0$. '$b_0 b_1 = 10$'인 경우는 '$g = 0$'일 때이다. 이외의 경우는 항상 '$b_0 b_1 = 01$'이다. 그러므로 근사 역수 테이블에 '$b_2, ... b_t$'만을 저장하면 된다. 따라서 근사 역수 테이블의 크기는 '$2^{n_g} * (t-1)$' 비트가 되어서, 테이블의 길이는 2^{n_g}이며, 폭은 '$t-1$' 비트이다.

T(g)에서 초기 오차 e_0는 식 (15)가 된다.

$$e_0 = \frac{1}{1.g + h} - T(g) \tag{15}$$

식 (14)로부터 e_0는 $h_m = 100...0$에서 가장 작으며, $h_z = 000...0$과 $h_{\max} = 111...1$에서 가장 커서 그림 1과 같이 된다.

식 (7)과 식 (11)로부터 식 (16)이 성립하면 2회의 곱셈으로 근사 역수를 계산할 수 있다.

$$e_0 < A_2 = \frac{2^{-p} + 2^{-\frac{p}{2}}}{1.g + h_{\max}} \tag{16}$$

식 (16)에서 A_2가 최소가 되는 값을 선택했다. 초기 오차 e_0는 양수와 음수의 두 가지 값을 가지며, 그림 1에 각각 A_2와 A_2'로 나타나고 있으며, A_2와 A_2'에서의 h 값이 각각 h_2와 h_2'이다. $h_2' < h < h_2$에서 2회의 곱셈으로 근사 역수를 계산할 수 있다.

식 (7)과 식 (13)로부터 식 (17)이 성립하면 3회의 곱셈으로 근사 역수를 계산할 수 있다.

표 3. IEEE 단정도실수 나눗셈 계산에 필요한 곱셈 횟수

Table 3. Number of multiplication for IEEE single precision division

Table size	Average No. of Multiply	NR No. of Multiply
16 X 3	6.23	7
32 X 5	5.19	7
32 X 6	4.97	7
64 X 5	5.00	7
64 X 6	4.69	7
128 X 6	4.67	5
256 X 7	4.63	5

표 4. IEEE 배정도실수 나눗셈 계산에 필요한 곱셈 횟수

Table 4. Number of multiplication for IEEE double precision division

Table size	Average No. of Multiply	NR No. of Multiply
64 X 5	7.42	9
64 X 6	6.88	9
128 X 6	6.68	9
128 X 7	6.67	7
256 X 7	6.67	7
512 X 8	6.63	7

$$e_0 < A_3 = \frac{2^{-p} + 2^{-\frac{p}{3}}}{1.g + h_{\max}} \tag{17}$$

그림 1에 A_3와 A_3'에서의 h 값이 각각 h_3와 h_3'이다. $h_3' < h < h_2'$와 $h_2 < h < h_3$에서 3회의 곱셈으로 근사 역수를 계산할 수 있다.

식 (8)로부터 '$e_1 = De_0^2 + 2^{-p+1}$'이다. $e_0 < A_2$가 되는 $e_1 = A_4$를 뉴톤-랍손 알고리즘으로 구할 수 있다. 이로부터 A_4와 A_4'에서의 h 값 h_4와 h_4'을 구할 수 있다. $h_4' < h < h_3'$와 $h_3 < h < h_4$에서 4회의 곱셈으로 근사 역수를 계산할 수 있다. 이와 같은 계산을 계속하여 수행하면 초기 오차 e_0에 따라서 근사 역수를 계산하기 위한 곱셈의 횟수를 산출할 수 있다.

본 논문에서 제안한 알고리즘에 의한 IEEE 단정도실수 및 배정도실수의 테이블 크기에 따른 나눗셈 계산에 필요한 곱셈 횟수를 표 3과 표 4에 보인다.

종래 뉴톤-랍손 알고리즘에서는 최대 오차를 고려해서 반복 횟수를 정했다. 표 3과 표 4에서 'NR No. of Multiply'는 종래 뉴톤-랍손 알고리즘에서의 곱셈 횟수이다. 즉, 종래 알고리즘에 의한 단정도실수 나눗셈은 '64x5' 테이블을 사용하면 7회의 곱셈을 수행하였고, '128x6' 테이블을 사용하면 5회의 곱셈을 수행하였음을 표 3으로부터 알 수 있다. 그러나 본 논문에서 제안한 알고리즘에서는 '64X5' 테이블에서 평균 5.00회의 곱셈, '128x6' 테이블과 '64x6' 테이블에서 평균 4.67회와 4.69회의 곱셈으로 나눗셈을 할 수 있다. 즉, 본 논문에서 제안한 알고리즘에서 평균 5회의 곱셈으로 나눗셈을 하려면 '64x5' 테이블을 사용하면 되므로, 종래 알고리즘에서 사용하던 '128x6' 테이블과 비교하여 테이블 크기를 반 이하로 줄일 수 있다.

이러한 결과는 배정도실수 연산에서도 동일하게 나타난다. 배정도실수 나눗셈은 표 4로부터 종래 알고리즘에서는 '128X6' 테이블을 사용하면 9회의 곱셈, '256X7' 테이블을 사용하면 7회의 곱셈으로 나눗셈을 계산하였다. 그러나 본 논문에서 제안한 알고리즘에서는 '128X6' 테이블을 사용하면 평균 6.68회의 곱셈, '64X6' 테이블을 사용하면 6.88회의 곱셈으로 나눗셈을 계산한다. 평균 7회 곱셈으로 나눗셈을 계산하려면 종래 알고리즘에서는 '256X7' 테이블을 사용했지만, 본 논문에서 제안하는 알고리즘에서는 '64X6' 테이블을 사용해서 평균 7회 곱셈으로 나눗셈을 수행할 수 있다. '64X6'테이블의 크기는 '256X7' 테이블의 사분의 일이다. 따라서 본 논문에서 제안한 알고리즘은 테이블 크기를 사분의 일로 줄이면서 동일한 성능을 보인다.

표 3과 표 4는 근사 나눗셈에 소요되는 곱셈 회수이며 정확한 결과를 구하기 위한 오차 보정에 한 번의 곱셈과 판정이 추가된다.

Ⅴ. 결론

부동소수점 나눗셈은 뺄셈을 반복하는 SRT 알고리즘과 곱셈을 반복하는 뉴톤-랍손(Newton-Raphson) 역수 알고리즘 및 골드스미트(Goldschmidt) 나눗셈 알고리즘이 있다. 뉴톤-랍손 역수 알고리즘은 제수의 역수를 피제수에 곱해서 나눗셈을 계산한다. 역수 계산은 제수의 역수의 근사 값을 초기 값으로 해서 반복 연산으로 오차를 줄여나간다. 반복 연산을 수행할 때마다 상대 오차

는 자승으로 줄어들며, 한 회의 반복 연산에 2회의 곱셈이 필요하다.

본 논문에서는 테일러급수로부터 가칭 K차 뉴톤-랍손 부동소수점 역수 알고리즘을 유도하였다. K차 알고리즘에서는 한 회 반복에 K번의 곱셈을 수행한다. 반복 연산 과정의 오차를 예측하고, 예측한 오차가 정해진 값보다 작아지는 시점까지만 반복 연산을 수행한다. 나눗셈 $\frac{N}{1.d}$에서 $1.d$의 역수로 '$1.d * 0.y \leq 1+2^{-w}$'이 되는 $0.y$를 제안한 역수 알고리즘으로 계산하고, '$N * 0.g = Mm$'을 계산하고 그 결과를 보정하여 정확한 나눗셈 값을 산출한다. 이를 K차 뉴톤-랍손 나눗셈 알고리즘이라 가칭한다.

제안한 알고리즘을 구현하는 회로와 상태 기계를 구성하였다, 또한 근사 테이블을 구성하고, 나눗셈 계산에 소요되는 평균 곱셈 횟수를 계산해서 그 결과를 종래의 뉴톤-랍손 알고리즘과 비교 분석하였다.

종래의 뉴톤-랍손 알고리즘에 의한 배정도실수 나눗셈은 '128X6' 근사 역수 테이블을 사용하면 9회의 곱셈, '256X7' 테이블을 사용하면 7회의 곱셈을 수행하였다. 그러나 본 논문에서 제안한 가변 시간 K차 뉴톤-랍손 역수 알고리즘에서는 '64X6' 테이블을 사용해서 평균 6.88회의 곱셈으로 나눗셈을 할 수 있었다, '64X6' 테이블의 크기는 '256X7' 테이블의 사분의 일에 불과하다. 따라서 본 논문에서 제안한 알고리즘은 근사 테이블 크기를 크게 줄일 수 있다

본 논문에서 제안한 K차 가변 시간 뉴톤-랍손 나눗셈 알고리즘은 평균 곱셈 횟수가 중요한 디지털 신호처리, 컴퓨터 그래픽스, 멀티미디어, 과학 기술 연산 등에서 폭 넓게 사용될 수 있다. 또한 계산기 성능에 따른 최적의 근사 역수 테이블을 구성할 수 있으므로 하드웨어 사양에 제한적인 SOC(System On Chip)에 유용하게 적용될 수 있다.

References

[1] V. Lappalainen, T.D. Hamalainen, P. Liuha, "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, No. 8, 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] S.F. Oberman, M.J. Flynn, "Design Issues in Division and Other Floating Point Operations," IEEE Transactions on Computer, Vol. C-46, No. 2, pp. 154-161, 1997.

[4] S.F. McQuillan, J.V. McCanny, R. Hamill, "New Algorithms and VLSI Architectures for SRT Division and Square Root," Proceedings of IEEE Symposium on Computer Arithmetic, pp. 80-86, 1993.

[5] D. Harris, S. Oberman, M. Horowitz, "SRT Division Architectures and Implementations," Proceedings of IEEE Symposium on Computer Arithmetic, 1997.

[6] C. Shuang, 모든저자기재요망, "Design and Implementation of a 64/32-bit Floating-point Division, Reciprocal, Square root, and Inverse Square root Unit," Proceedings of International Conference on Solid-State and Integrated Circuits Technology, pp. 1976-1979, 2006.

[7] M.D. Ercegovac, D.H. Wang, T.J. Zhang, C.H. Hou, "Improving Goldschmidt Division, Square Root, and Square Root Reciprocal," IEEE Transactions on Computer, Vol. 49, No. 7, pp. 759-763, 2000.

[8] D.L. Fowler, J.E. Smith, "An Accurate, High Speed Implementation of Division by Reciprocal Approximation," Proceedings of IEEE Symposium on Computer Arithmetic, pp. 60-67, 1989.

[9] S. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microprocessors, " Proceedings of IEEE Symposium on Computer Arithmetic, pp. 106-115, 1999.

[10] G. Agrawal, A. Khandelwal, E.E. Swartzlander, "An improved reciprocal approximation algorithm for a Newton Raphson divider," Proceedings of Advanced Signal Processing Algorithms, Architectures and Implementations XVII, 66970M, 2007.

[11] S.G. Kim, G.Y. Cho, "A Variable 2Latency Newton-Rapson's Floating Point Number

Reciprocal Computation," Korea Information Processing Society, Vol. 12-A, No. 2, pp. 95-102, 2005. (in Korea)
[12] S.F. Anderson, J. Earle, R. Coldschmidt, D. Powers, "The IBM System/360 model 91 Floating Point Execution Unit," IBM Journal of Research and Development, Vol. 11, No. 1, pp. 34-53, 1967.
[13] P. Markstein, "Computation of elementary functions on the IBM RISC system/6000 processor," IBM Journal of Research and Development, Vol. 34, No. 1, pp. 111-119, 1990.
[14] E.M. Schwarz, "Rounding for Quadratically Converging Algorithm for Division and Square Root," Proceedings of IEEE Asilomar Conference on Signals and Computers, pp. 600-603, 1996.
[15] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard, Std. 754-1985.
[16] http://www.itl.nist.gov/fipspubs, Oct. 2008.
[17] D. DasSarma, D. Matula, "Measuring and Accuracy of ROM Reciprocal Tables," IEEE Transactions on Computer, Vol. 43, No. 8, pp. 932-930, 1994.

저 자 소 개

조 경 연

1990년 인하대학교 전자공학과 박사.
1983년~1991년, 삼보컴퓨터 기술연구소 책임연구원.
1998년~현재, 에이디칩스(주) 비상임 기술고문.
현재, 부경대학교 공과대학 IT융합응용공학과 교수.
관심분야 : 컴퓨터구조, 반도체회로 설계, 암호 알고리즘
Email : gycho@pknu.ac.kr