개선한 Goldschmidt 부동소수점 역수 알고리즘

한경헌*O, 최명용*, 김성기*, 조경연*
*부경대학교 대학원 컴퓨터공학과

The improved Goldschmidt floating point reciprocal algorithm

Kyoung-houn Han*O , Myoung-yong Choi* , Soung-ki Kim* , Gyoung-youn Cho*
* Computer Science Dept. Graduate school, Pukyong National University
*O E-mail:dreamplus@hanmir.com

요 약

Goldschmidt 알고리즘에 의한 부동소수점 1.f2의 역수는 q=NK1K2....Kn (Ki=1+Aj, j=2i)이다. 본 논문에서는 N과 A 값을 1.f2의 값에 따라서 선정하고 Aj의 값이 유효자리수의 반이하 값을 가지면 연산을 종료하는 개선된 Goldschmidt 부동소수점 역수 알고리즘을 제안한다.

1.f2가 1.01012보다 작으면 N=2-1.f2, A=1.f2-1로 하며, 1.01012보다 크거나 같으면 N=2-0.1f2, A=1-0.1f2로 한다. 한편 Goldschmidt 알고리즘은 곱셈을 반복해서 수행하므로 계산 오류가 누적이 된다. 이러한 누적 오류를 감안하면 배정도실수 역수에서는 2-57, 단정도실수 역수에서는 2-28의 유효자리수까지 연산해야 한다. 따라서 Aj가 배정도실수 역수에서는 2-29, 단정도실수 역수에서는 2-14 보다 작아지면 연산을 종료한다.

본 논문에서 제안한 개선한 Goldschmidt 역수 알고리즘은 N=2-0.1f2, A=1-0.1f2로 계산하는 종래 알고리즘과 비교하여 곱셈 연산 회수가 배정도실수 역수는 22%, 단정도실수 역수는 29% 감소하였다. 본 논문의 연구 결과는 테이블을 사용하는 Goldschmidt 역수 알고리즘에 적용해서 연산 시간을 줄일 수 있다.

Ⅰ.서 론

부동수소점 나눗셈 연산은 출현 빈도는 낮지만 연산 시간이 길게 걸리므로 성능에 많은 영향을 미친다. Obermann은 통상적인 계산 프로그램에서 나눗셈 연산 처리에 소요되는 시간이 덧, 뺄셈이나 곱셈 연산 처리에 소요되는 시간과 비슷하다는 연구 결과를 발표하였다[1]. 따라서 시스템 성능을 높이기 위해서는 고성능 나눗셈 알고리즘에 대한 연구가 필수적이다.

나눗셈 알고리즘은 SRT 알고리즘으로 대표되는 숫자 순환 방식(digit recurrence method)과 곱셈을 반복 사용하여 제수의 역수를 구해서 피제수와 곱하는 반복 방식(iterative method)이 있다[2][3]. 숫자 순환 방식은 몫과 나머지를 구할 수 있는 장점을 가지나 처리 속도가 늦은 단점을 가진다. 반복 방식은 근사계산이라는 한계 가 있으나 처리 속도가 빠른 장점을 가진다. 본 논문에서는 반복 방식의 하나인 Goldschmidt 역수 알고리즘[5]을 개선하여 더욱 빠른 처리 속도를 구현할 수 있는 개선한 Goldschmidt 역수 알고리즘을 제안한다.

Goldschmidt 역수 알고리즘에서 부동소수점 1.f2의 역수는 q=NK1K2....Kn (Ki=1+Aj, j=2i)로 주어진다. 이 식에서 A의 값이 유효자리수 최소값보다 작아지면 Ki는 '1'이 되므로 그 이상 연산은 수행할 필요가 없다. 본 논문에서 제안하는 개선한 Goldschmidt 역수 알고리즘은 Aj의 값을 판단하여 불필요한 연산을 수행하지 않도록 한다. 또한 Aj가 유효자리수 최소값에 수렴하는 반복 횟수를 줄이기 위해서 1.f가 1.01012보다 작으면 N=2-1.f2, A=1.f2-1로 하며, 1.01012보다 크거나 같으면 N=2-0.1f2, A=1-0.1f2로 한다.

본 논문에서 제안한 개선한 Goldschmidt 역수 알고리즘은 종래 Goldschmidt 역수 알고리즘과 비교하여 곱셈 횟수를 단정도실수 역수에서는 29%, 배정도실수 역수에서는 22% 줄일 수 있었다.

Ⅱ. Goldschmidt 역수 알고리즘

IEEE-754에서 규정한 부동소수점수는 F=1.f2 X

2n의 포맷을 가진다. 따라서 F의 역수를 구하는 문제는 1.f2의 역수를 구하는 문제로 귀결된다. 단정도실수에서 1.f2의 역수는 식-1같이 변환할 수 있다.

$$1/1.f_2 = N/(1-A^2) \quad \text{---------- (1)}$$

식-1에서 A는 1/2보다 작은 수이다. 식-1의 분모, 분자에 (1+A2)를 곱하면 식-2가 된다.

$$1/1.f_2 = N(1+A^2)/(1-A^4) \quad \text{---------- (2)}$$

이와 같은 과정을 반복하면 1.f2의 역수는 식-3이 된다.

$$1/1.f_2 = N(1+A^2)(1+A^4)(1+A^8)(1+A^{16})/(1-A^{32}) \quad \text{---------- (3)}$$

식-3에서 분모는 (1-A32) ≥ (1-2-32) ≒ 1 이 된다. 따라서 분모는 계산할 필요가 없으며 분자만을 연산하면 1.f2의 역수를 계산할 수 있다.

배정도실수연산은 식-3의 분모, 분자에 (1+A32)를 곱해주면 분모가 약 1이 되므로 1.f2의 역수를 계산할 수 있다. 배정도실수의 Goldschmidt 역수 알고리즘은 식-4가 된다.

$$1/1.f_2 = N(1+A^2)(1+A^4)(1+A^8)(1+A^{16})(1+A^{32}) \quad \text{---------- (4)}$$

식-3과 식-4에서 수렴을 빠르게 하기 위해서 본 논문에서는 1.f2가 1.01012보다 작으면 식-1을 식-5 같이 구한다.

$$1/1.f_2 = 1/(1+0.f_2) = (1-0.f_2)/(1-0.f_2^2) \quad \text{---------- (5)}$$

식-5로부터 'N = 1-0.f2 = 2-1.f2', 'A = 0.f2 = 1.f2-1'가 된다. 한편 1.f2가 1.01012보다 크거나 같으면 식-1을 식-6같이 구한다.

$$1/1.f_2 = 1/(2\times 0.1f_2) = 1/(2\times(1-A)) = 1/(1-A)\times 2^{-1} = (1+A)/(1-A^2)\times 2^{-1} \text{------- (6)}$$

식-6에서 2-1은 부동소수점수 누승부 연산에서 처리하면 되므로 실수부 연산시에는 고려하지 않아도 된다. 따라서 식-6으로부터 'A = 1 - 0.1f2', 'N = 1 + A = 2 - 0.1f2'가 된다.

III.오류 계산

Goldschmidt 역수 알고리즘은 식-3 및 식-4같이 곱셈을 반복하므로 계산 오류가 누적된다. IEEE-754에서 정의한 유효자릿수의 연산을 수행하기 위해서 최대 누적 오류를 산출한다. 본 논문에서는 최대 누적 오류를 구하기 위해서 중간 곱셈 결과를 절삭한다.

역수 알고리즘 식-3과 식-4에서 N은 오류없이 정확하게 연산할 수 있다. A2 연산은 A와 A를 곱하고 소수점 아래 m 자릿수 이하를 절삭하면 식-7 같이 된다.

$$A^2 = [A^2] - e \quad \text{---------- (7)}$$

식-7에서 [A2]은 오류가 없는 정확한 값을 나타내며 e는 m 자릿수 이하를 절삭하므로 발생한 오류이다. 따라서 'e 〈 2-(m-1)'이다.

A4 연산은 식-7의 A2을 자승하여서 구하면 식-8 같이 된다.

$$A^4 = ([A^2]-e)^2 - e \fallingdotseq [A^4] - 2e[A^2] - e \fallingdotseq [A^4] - 1.5e \quad \text{---------- (8)}$$

식-8에서 [A2]의 최대값은 2-2이다. A8, A16 및 A32은 식-9부터 식-11이 된다.

$$A^8 = ([A^4]-1.5e)^2 - e \fallingdotseq [A^8] - 3e[A^4] - e \fallingdotseq [A^8] - 1.2e \quad \text{---------- (9)}$$

$$A^{16} = ([A^8]-1.2e)^2 - e = [A^{16}] - 2.4[A^8]e - e \fallingdotseq [A^{16}] - e \quad \text{---------- (10)}$$

$$A^{32} = ([A^{16}]-e)^2 - e = [A^{32}] - 2[A^{16}]e - e \fallingdotseq [A^{32}] - e \quad \text{---------- (11)}$$

식-7부터 식-11을 식-3 및 식-4에 대입하여 누적이 되는 최대 오류를 계산한다. 식-3 및 식-4에서 N(1+A2) 곱셈 연산은 식-12같이 된다.

$$N(1+A^2) = N(1+[A^2]-e)-e = N(1+[A^2]) - e(N+1) = X1 - 2.5e \quad \text{---------- (12)}$$

식-5 및 식-6에서 N의 최대값은 '1.f2 = 1.0'인 경우로 1.5가 된다. 식-12에서 X1 = N(1+[A2])으로 오류없이 계산된 정확한 값이다.

식-12의 연산 과정을 반복하여 단정도실수 및 배정도실수 역수의 누적 최대 오류를 계산하면 식-13부터 식-16이 된다. $Xi(i=1,2,3,4,5)$는 오류없이 계산된 정확한 값을 나타낸다.

$$
\begin{aligned}
&N(1+A^2)(1+A^4)\\
&=(X1-2.5e)(1+[A^4]-1.5e)-e\\
&\fallingdotseq X1(1+[A^4])-2.5(1+[A^4])e-1.5X1e-e\\
&\fallingdotseq X2-3e-e-2.53e-0.2e\\
&\fallingdotseq X2-7e \qquad \text{----------} \quad (13)
\end{aligned}
$$

$$
\begin{aligned}
&N(1+A^2)(1+A^4)(1+A^8)\\
&=(X2-7e)(1+[A^8]-1.2e)-e\\
&\fallingdotseq X2(1+[A^8])-1.2X2e-(1+[A^8])7e-e\\
&\fallingdotseq X3-2.4e-7e-e \fallingdotseq X3-10e \quad \text{----} \quad (14)
\end{aligned}
$$

$$
\begin{aligned}
&N(1+A^2)(1+A^4)(1+A^8)(1+A^{16})\\
&=(X3-10e)(1+[A^{16}]-e)-e\\
&\fallingdotseq X3(1+[A^{16}])-X3e-10e(1+[A^{16}])-e\\
&\fallingdotseq X4-2e-10e=X4-13e \quad \text{--------} \quad (15)
\end{aligned}
$$

$$
\begin{aligned}
&N(1+A^2)(1+A^4)(1+A^8)(1+A^{16})(1+A^{32})\\
&=(X4-13e)(1+A^{32}-e)-e\\
&\fallingdotseq X4(1+[A^{32}])-X4e-13e(1+[A^{32}])-e\\
&\fallingdotseq X5-14e-X4e \fallingdotseq X5-14e-2e\\
&=X5-16e \qquad \text{----------} \quad (16)
\end{aligned}
$$

단정도실수 역수는 식-15까지 연산하므로 누적 최대 오류는 $13e$이다. 한편 단정도실수의 유효자릿수는 소숫점이하 23 비트까지이며 반올림 자리수를 포함하여 24 비트까지의 정확한 연산 결과가 요구된다. 식-15로부터 최대 누적 오류 13e 〈2-24이 되어야 하므로, e 〈2-28이 요구된다.

배정도실수 역수는 식-16까지 연산하므로 누적 최대 오류는 $16e$이다. 배정도실수의 유효자릿수는 소숫점이하 52 비트까지이며 반올림 자리수를 포함하여 53 비트까지의 정확한 연산 결과가 요구된다. 식-16으로부터 최대 누적 오류 16e 〈2-53이 되어야 하므로, e 〈2-57이 요구된다.

IV. 개선한 알고리즘

단정도실수 역수에서는 e<2-28이 요구되기 때문에 Aj 〈2-14이면 식-3의 연산을 종료하고, 배정도실수 역수에서는 e<2-57이 요구되기 때문에 Aj 〈2-29이면 식-4의 연산을 종료한다.

본 논문에서 제안하는 개선한 Goldschmidt 역수 알고리즘의 실수 부분은 표-1과 같다.

표-1. 개선한 Goldschmidt 역수 알고리즘의 실수 부분

```
Input : significant = 1.f2 ;
Output : Q = 1/1.f2 ;

1st stage :
    If(1.f >= 1.0101₂ ) Q=2-0.1f2, A=1-0.1f2;
    else Q=2-1.f2, A=1.f2-1 ;

2nd stage :
    A = A * A ;

3rd stage :
    Q = Q * (1 + A) ;
    If in single precision (A < 2^-14 )
        exit ;
    If in double precision (A < 2^-29 )
        exit ;
    Goto 2nd stage ;
```

표-1의 개선한 역수 알고리즘을 적용하여 단 정도실수 역수 및 배정도실수 역수를 계산하여 1.f2의 범위에 따른 곱셈 횟수를 표-2 및 표-3에 보인다.

표-2. 단정도실수 역수의 곱셈 횟수

$1.f_2$의 범위	곱셈횟수	유효곱셈횟수
$1.0 \le 1.f_2 < 1.0078$	2	0.0156
$1.0078 \le 1.f_2 < 1.0875$	4	0.3188
$1.0875 \le 1.f_2 < 1.2958$	6	1.2498
$1.2958 \le 1.f_2 < 1.4083$	8	0.9000
$1.4083 \le 1.f_2 < 1.8250$	6	2.5002
$1.8250 \le 1.f_2 < 1.9843$	4	0.6372
$1.9843 \le 1.f_2 < 2.0$	2	0.0314

표-3. 배정도실수 역수의 곱셈 횟수

$1.f_2$의 범위	곱셈횟수	유효곱셈횟수
$1.0 \le 1.f_2 < 1.00004$	2	0.00008
$1.00004 \le 1.f_2 < 1.00653$	4	0.02596
$1.00653 \le 1.f_2 < 1.08084$	6	0.44586
$1.08084 \le 1.f_2 < 1.28433$	8	1.62792
$1.28433 \le 1.f_2 < 1.43132$	10	1.46990
$1.43132 \le 1.f_2 < 1.83830$	8	3.25584
$1.83830 \le 1.f_2 < 1.98692$	6	0.89172
$1.98692 \le 1.f_2 < 1.99998$	4	0.05224
$1.99998 \le 1.f_2 < 2.0$	2	0.00004

표-2 및 표-3에서 유효곱셈횟수는 1.f2의 범위에 곱셈횟수를 곱한 것으로 모든 항목의 유효곱셈횟수를 더하면 평균 곱셈 횟수가 된다.

표-2 및 표-3으로부터 개선한 알고리즘에 의한 단정도실수 역수의 평균 곱셈 횟수는 5.65이고, 배정도실수 역수의 평균 곱셈 횟수는 7.77이다. 따라서 종래의 식-3과 식-4에 의한 Goldschmidt 역수 알고리즘과 비교하여 단정도실수 역수는 29%, 배정도실수연산은 22% 평균 곱셈 횟수를 감소시킨다.

V.결 론

본 논문에서는 부동소숫점 역수 연산에 널리 사용되는 Goldschmidt 알고리즘을 개선하였다.

Goldschmidt 역수 알고리즘에서 부동소수점 1.f2의 역수는 q=NK1K2....Kn (Ki=1+Aj, j=2i)로 주어진다. 본 논문에서 제안하는 개선한 Goldschmidt 역수 알고리즘은 Aj의 값을 판단하여 불필요한 연산을 수행하지 않도록 하였다. Aj가 유효자리수 최소값에 수렴하는 반복 횟수를 줄이기 위해서 1.f2가 1.01012보다 작으면 N=2-1.f2, A=1.f2-1로 하며, 1.01012보다 크거나 같으면 N=2-0.1f2, A=1-0.1f2로 하였다.

Goldschmidt 역수 알고리즘은 곱셈을 반복하므로 곱셈 연산 오차가 누적된다. 본 논문에서는 누적 최대 오차를 계산하였고, 그 결과 단정도실수 역수에서는 소숫점 아래 28 비트까지, 배정도실수 역수는 57 비트까지 연산해야 함을 밝혔다. 따라서 본 논문에서는 단정도실수 역수에서는 Aj 〈2-14이면 연산을 종료하였고, 배정도실수 역수에서는 Aj 〈2-29이면 연산을 종료하였다.

본 논문에서 제안한 개선한 Goldschmidt 부동소수점 역수 알고리즘은 기존의 Goldschmidt 알고리즘과 비교하여 곱셈 횟수를 단정도실수 역수 연산에서는 29%, 배정도실수 역수 연산에서는 22% 줄일 수 있었다.

참고문헌

[1] Stuart Franklin Oberman, "DESIGN ISSUES IN HIGH PERFORMANCE FLOATING POINT ARITHMETIC UNITS," Technical Report: CSL-TR-96-711, pp. 43-87, December 1996.

[2] Israel Koren, Computer Arithmetic Algorithms 2nd EDITION, Prentice-Hall, pp. 181-221 , 2001.

[3] 정재원 , "내장형 프로세서를 위한 IEEE- 754 고성능 부동소수점 나눗셈기의 설계" , 전자공학회논문지 , pp. 1-49 , 2002.

[4] Guy Even , Peter-Michael Seidel and Warren E. Ferguson , "A Parametric Error Analysis of Goldschmidt's Division algorithm," 12th IEEE Symposium on Computer Arithmetic, pp. 1-7 . 2003.

[5] Milos Ercegovac , Laurent Imbert , David Matula, Jean-Michel Muller and Guoheng Wei, "Improving Goldschmidt Division, Square Root and Square Root Reciprocal," INSTITUT NATIONAL DE RECHERCE ENINFORMATIQUE ET AUTOMATIQUE, pp. 5-20 , September 1999.