높은 자릿수 나눗셈 연산기에서의 영역변환상수를 위한 검색테이블 설계 및 구현

이병석, 송문식, 이정아
조선대학교 전자계산학과
{novrain, song, jeong}@rain.chosuna.ac.kr

Design and Implementation of Look-up Table for Pre-scaling in Very-High Radix Divider

Byeong-Seok Lee, Moon-Sik Song, Jeong-A Lee
Dept. Computer Science, Chosun University

요 약

나눗셈 알고리즘은 다른 덧셈이나 곱셈 알고리즘에 비해 복잡하고, 수행 빈도수가 적다는 이유로 그 동안 고속 나눗셈의 하드웨어 연구는 활발하지 않았다. 그러나 멀티미디어의 발전 및 고성능의 그래픽 랜더링을 위한 보다 빠른 부동소수점연산기(FPU)가 필요하게 되었으며, 이에 따라서 고속의 나눗셈 연산기의 필요성이 증가하게 되었다. 특히, 전체의 수행 시간 향상을 위해서라도 고속 나눗셈 연산기의 중용성은 더욱 부각되고 있다. 그러나 고속 나눗셈 연산기는 연산 속도와 크기라는 서로 상반되는 요소를 가지고 있다. 즉, 연산 속도가 빠르면 크기는 늘어나고, 크기를 줄이면 연산 속도는 늦어지게 된다.

본 논문은 높은 자릿수(Very-High Radix) 나눗셈 알고리즘에서 영역변환상수를 구하는 방법으로 연산이 아닌 검색테이블(Look-up Table)을 이용한다. 그리고 검색테이블의 크기를 줄이는 방법으로 영역변환상수의 범위 분석 및 캐리 저장형을 이용한 검색테이블 분할 방법을 이용하였다. 전체적으로는 영역변환상수를 구하는 연산주기가 필요 없게 되므로 나눗셈 연산기의 영역 크기의 변화가 적으면서 연산 속도는 빨라졌음을 알 수 있다.

1. 서론

나눗셈 알고리즘은 덧셈이나 곱셈 알고리즘보다 복잡하여 소프트웨어로 처리하는 경우 연산 수행 시간이 길어지는 문제점이 있었으나 지금까지는 그 빈도수가 다른 연산작업에 비해 적다는 이유로 고속 나눗셈 하드웨어의 연구가 활발하지는 않았다. 그러나 멀티미디어의 발달로 나눗셈이 차지하는 비중이 점점 더 커지게 되었고, 나눗셈은 연산 수행 시간이 길기 때문에 빈도 수는 적어도 전체 수행 시간을 고려할 때 고속 나눗셈 연산기의 중요성이 더욱 더 부각되고 있다.

나눗셈 알고리즘에서 높은 자릿수(Very-High Radix) 나눗셈 알고리즘은 한 연산주기에 n개의 몫 비트를 생성하기 때문에 연산 속도가 빠르다는 장점이 있으나 몫을 선택하는 범위가 크기 때문에 몫을 선택하는 함수가 복잡하고, 몫을 선택하는 시간이 많이 소요되기 때문에 제수와 피제수의 영역을 이동하여서 몫 선택을 간단하게 하는 방법을 이용한다.[1][2][3][4] 이 때, 영역변환을 하기 위해서는 영역변환상수가 필요하다. 일반적으로 이 영역변환상수는 감마테이블에 저장된 감마 값과 제수의 일부 비트 값을 연산하여 구한다.

본 논문은 영역변환상수를 연산이 아닌 영역변환상수 검색테이블을 이용하여서 구하였다. 검색테이블의 크기는 입력 비트의 길이에 따라 크기가 크게 달라지므로 최소한의 입력 비트를 이용하여 검색테이블을 구성해야 한다. 이러한 방법으로 영역변환상수의 범위를 분석하여 검색테이블의 범위를 축소하였으며, 영역변환상수 검색테이블을 분할하여 검색테이블의 크기를 줄이는 방법을 이용하였다. 또한 캐리 저장형(Carry-Save Form)의 특성을 이용하여 추가적인 연산 없이 영역변환상수를 구하도록 하였다.

2. 높은 자릿수 나눗셈 알고리즘에서의 영역변환상수

높은 자릿수 나눗셈 알고리즘의 특징은 하나의 주기(Cycle)에 여러 개의 q 비트를 찾아낸다. 이 알고리즘은 $q = x/d$일 때 다음 식(1)를 만족해야 한다.

$$1 > d > x \geq 1/2 \tag{1}$$

그리고, 다음 식(2)을 반복함으로서 몫을 생성한다.

$$w_{i+1} = rw_i - q_{i+1}z \tag{2}$$

식(2)에서 몫 (q_{i+1})의 선택을 간단하게 하기 위해서 식(3)처럼 영역변환상수 (M)를 이용하여 제수 (d)와 피제수 (x)의 영역을 이동한다.

$$z = Md, \quad w_0 = Mx \tag{3}$$

영역변환상수 (M)는 식(4)를 이용하여 구하며, 감마 (γ_1, γ_2) 값은 식(5, 6)을 이용하여 구한다. 일반적으로 감마 값은 검색테이블로 구성한다.

$$M = -\hat{\gamma}_1 d_h + \hat{\gamma}_2 \tag{4}$$

$$\gamma_1 = \frac{1}{d_r^2 + d_r 2^{-r} + 2^{-(b+4+2)}} \tag{5}$$

$$\gamma_2 = \frac{2d_r + 2^{-r}}{d_r^2 + d_r 2^{-r} + 2^{-(b+4+2)}} \tag{6}$$

여기서, $h=b+5$, $r=\left(\left\lfloor\frac{b}{2}\right\rfloor+2\right)$

높은 자릿수 나눗셈 연산기에서 영역변환상수는 따로 연산기를 이용하거나 누적연산기를 공유하여 연산하는 방법이 있다.[4][5]

w_i 값과 q_{i+1} 값은 항상 고정된 연산시간을 위해서 캐리 저장형으로 표현하여 덧셈한다. 그리고 리코딩(Recoding)을 위해서 Radix-4 형식으로 변환한다. 최종 몫은 반올림 및 On-The-Fly 변환으로 얻는다.[4]

3. 검색테이블을 이용한 영역변환상수 설계

3.1 영역변환상수의 범위 분석

검색테이블의 크기는 입력 비트의 길이에 따라 크기가 크게 좌우된다. 따라서 검색테이블을 설계할 경우 입력 비트의 길이를 최소로 해야 한다. 영역변환상수를 구하기 위해서는 제수의 $b+5$비트의 길이를 이용하여 두 자리 정수와 $b+4$의 실수로 출력한다.[2][4][5] 검색테이블로 구성할 경우 크기는 식(7)과 같다.

$$\text{Table Size} = 2^{b+5}\times(2+b+4)\ \text{(bit)} \tag{7}$$

식(7)처럼 검색테이블을 구성할 경우, 검색테이블의 크기가 기존의 연산기보다 더 커지는 형상이 발생하기 때문에 전혀 쓸모 없게 된다. 따라서 검색테이블을 구성할 영역변환상수의 범위를 분석하여 검색테이블의 크기를 줄이는 방법을 이용한다.

높은 자릿수 나눗셈 연산기에서 영역변환된 제수 (z)의 범위는 식(8)의 조건을 만족해야 한다. 식(8)를 비트로 분석하면 식(9)와 같다.

$$1-\frac{r-2}{4r(r-1)} < z < 1+\frac{r-2}{4r(r-1)} \tag{8}$$

$$\sum_{i=1}^{b+2}2^{-i} < z < 1+2^{-(b+2)} \tag{9}$$

다음 영역변환된 제수의 최소값 ($z_{\min}$)과 식(3)을 이용하여 식(10)과 같은 영역변환된 제수를 얻는다. 그리고 식(10)에서 구한 값은 식(8)을 만족해야 한다.

$$\tilde{z_i} = \left(\frac{z_{\min}}{d_{i-1}-\sum_{i=f+1}^{n}\delta_i 2^{-i}}\right)\left(d_i-\sum_{i=f+1}^{n}\delta_i 2^{-i}\right) \tag{10}$$

.여기서 f는 입력 비트의 길이, $\delta_i \in \{0,1\}$

입력 비트의 길이 (f)가 적은 경우 영역변환된 제수의 범위가 식(8)을 벗어나게 된다. 따라서 영역변환된 제수의 범위가 식(8)를 만족하는 최소의 입력 비트의 길이인 f와 출력 비트의 길이인 g를 찾아야 한다. 이 f와 g는 프로그램을 이용하여 구하였으며, 최종적인 영역변환상수는 식(12)와 같다.

$$f = b+2,\quad g = b+3 \tag{11}$$

$$\hat{M} = \frac{z_{\min}}{d_{i-1}-\sum_{i=f+1}^{n}\delta_i 2^{-i}} - \sum_{i=g+1}^{n}\delta_i 2^{-i} \tag{12}$$

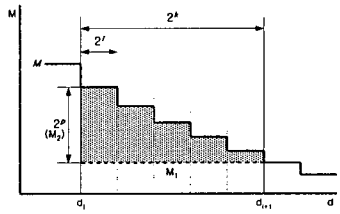
식(12)에 의하여 얻어진 영역변환상수를 검색테이블로 구성하였을 경우 크기는 식(13)과 같다. 여기서 제수는 식(1)에 의하여 항상 0.5 (2^{-1})이기 때문에 1비트를 생략하고, 출력에서 정수는 항상 1 (2^0)이기 때문에 2비트를 생략한다.

$$\text{Table Size} = 2^{b+1}\times(b+2)\ \text{(bit)} \tag{13}$$

3.2 캐리 저장형을 이용한 검색테이블의 분할

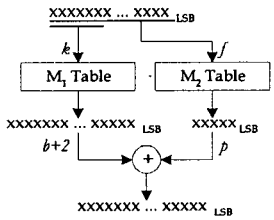
검색테이블의 크기를 줄이는 방법은 입력 비트의 길이를 줄이는 방법과 출력 비트의 길이를 줄이는 방법이 있다. 특히, 입력 비트의 크기를 줄이는 방법이 검색테이블의 크기를 줄이는데 매우 효과적이다. 즉, 최소한의 입력 비트의 길이를 이용하여 영역변환상수를 구하는 방법이 매우 좋은 방법이다.

[그림 1]은 입력 비트의 길이에 대한 영역변환상수의 관계를 나타낸다. [그림 1]에서처럼 입력 비트의 길이가 f일 때 영역변환상수는 M이지만, 입력 비트의 길이를 k로 줄일 경우 영역변환상수는 M_1이 된다. 즉, 입력비트의 길이를 줄이게 되면 검색테이블의 크기 역시 줄일 수 있으나, 실제 영역변환상수와는 많은 오차가 발생하게 된다. 여기서 오차를 수정할 수 있는 값으로 구성된 검색테이블을 이용하면 문제를 해결할 수 있다. 하나의 검색테이블에는 실제의 입력 비트 (f)의 길이보다 작은 입력 비트의 길이 (f)를 이용하여 영역변환상수를 구성하고, 다른 하나의 검색테이블은 실제의 입력비트 (f)를 이용하여 오차 값을 구한다. 그리고 두 검색테이블의 결과를 더하면 필요한 영역변환상수를 얻을 수 있다.



[그림 1] 입력비트의 길이와 영역변환상수와의 관계

[그림 2]는 입력 비트에 대한 M_1검색테이블과 M_2검색테이블의 입력 및 출력 비트를 나타낸다.



[그림 2] 입력 비트에 대한 분할된 검색테이블의 입력과 출력 비트의 관계

특히, 캐리 저장형을 이용하기 때문에 M_1값은 Sum에 M_2값은 Carry에 데이터를 전송하면 별도의 가산기가 필요 없다는 장점이 있다.

분할된 영역변환상수 M_1과 M_2는 식(13)으로 구한다.

$$M_1 = \frac{z_{\min}}{d_{i+1}},\quad M_2 = \hat{M}-M_1 \tag{13}$$

그리고 M_1 검색테이블을 위한 입력비트 k와 M_2 값의 길이 비트는 프로그램일 이용하여 구하며, 각 자릿수에 대한 k와 p의 크기는 [표 1]과 같다.

[표1] 각 자릿수에 대한 k와 p값

b	4	5	6	7	8	9	10	11	12	13	14	15
k	2	3	3,4	4	5	6	7	8	9	10	10,11	11
p	5	5	6,5	6	6	6	6	6	6	6	7,6	6

검색테이블의 크기는 다음과 같다.

$$M_1\ \text{Table Size} = 2^k\times(b+2)\ \text{(bit)} \tag{14}$$

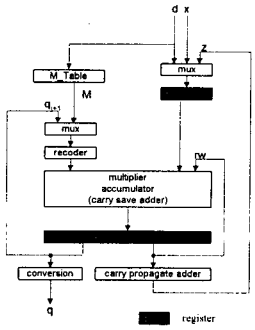
$$M_2\ \text{Table Size} = s^f\times p\ \text{(bit)}$$

여기서 $p = \lceil \log_2(\hat{M}-M_1)2^g \rceil$

Table Size = M_1 Table Size + M_2 Table Size (bit)

4. 구현 및 평가

영역변환상수 검색테이블을 이용한 높은 자릿수 나눗셈 연산기의 구조는 [그림 3]과 같다.



[그림 3] 검색테이블을 이용한 높은 자릿수 나눗셈 연산기

구현 및 평가의 기준으로 입력 비트의 길이는 IEEE754 배정도에서 가수비트인 52비트로 한다. 그러나 가수비트에는 정수 1이 생략되었고, 식(1)을 만족해야 하므로 실제 입력으로 제수는 53비트, 피제수는 54비트를 입력한다.

먼저 각 자릿수에 대한 감마테이블, 영역변환상수 검색테이블, 영역변환상수 범위 분석 후의 검색테이블, 캐리 저장형을 이용한 검색테이블의 크기를 비교하면 [표2]와 같다.

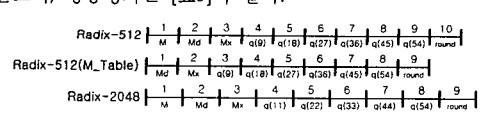
[표2] 각 검색테이블의 비교

Table / b	감마	영역변환상수	영역변환상수 (영역 분석 후)	영역변환상수 (M_1 + M_2)
	크기(비트)	비율	비율	비율
4	152	13.5	1.3	0.7
5	168	27.4	2.7	1.3
6	368	27.8	2.8	1.2
7	400	56.3	5.8	2.3
8	864	56.9	5.9	2.1
9	928	114.8	12.1	4.1
11	2,112	232.7	25.2	7.4
14	9,984	472.6	52.5	13.1
18	48,128	1917.3	217.9	44.9
⋮	⋮	⋮	⋮	⋮

(비율: 각 자릿수에 대한 감마 검색테이블이 기준)

[표 2]를 보면 영역변환상수의 범위를 분석한 다음 검색테이블을 분할하는 방법이 아무런 효과를 주지 않는 검색테이블보다 약 40배정도 작으며, 감마 검색테이블보다 약 1~25배 정도의 차이가 있지만, 영역변환상수 검색테이블을 이용할 경우 연산순환주기가 감소하기 때문에 연산순환주기가 같은 감마 검색테이블과 비교하면 약 0.5~2.7배의 차이밖에 없다. 더구나 영역변환상수를 구하는 연산기가 필요 없기 때문에 나눗셈 연산기의 영역 크기 변화는 매우 적다.

다음 기본자릿수를 512로 하여, 기본자릿수가 같은 나눗셈 연산기와 연산순환주기가 같은 나눗셈 연산기를 구현 및 평가하였다. 구현 환경을 보면 각 검색테이블은 UltraSPARC1, Solaris 2.5.1, WORKSHOP™ SPARCompiler C++4.1을 이용하여 검색테이블 VHDL 코드 생성기를 만들었으며, Synopsys 98.02와 Compass v8r4.10(VTI cmn12 (1.2㎛) 라이브러리)를 이용하여 합성, 동작 확인 및 속도와 크기를 구하였다. 각 연산기의 연산수환주기는 [그림 4]와 같으며, 성능평가는 [표3]과 같다.



[그림 4] 각 나눗셈 연산기의 연산순환주기

[표 3] 각 나눗셈 연산기의 성능 평가

	radix-512	radix-512 (M_Table)	radix-2048
크기(Area)	15,479(NAND)	17,335(NAND)	19,348(NAND)
주기(Cycles)	10	9	9
Critical Path	53.58ns	54.61ns	60.00ns
속도(Speed)	535.8ns	491.49ns	540.09
크기 비율	1	1.12	1.25
속도 비율	1	0.92	1.01

(기준: radix-512 나눗셈 연산기)

[표 3]에서 보는바와 같이 영역변환상수 검색테이블을 이용한 연산기는 기본자릿수가 같은 나눗셈 연산기와 비교하여 크기는 약 1.12배, 속도는 연산순환주기가 감소하기 때문에 약 0.92배 차이가 있고, 연산순환주기가 같은 나눗셈 연산기와 비교하면 크기는 약 0.9배, 속도는 0.91배의 차이가 있다. 또한, 나눗셈 연산기 성능 비교시 연산순환주기를 기준으로 비교하기 때문에 전체적인 성능을 보면 크기는 0.9배 적어지면서, 속도는 0.9배 빨라짐은 확인 할 수 있다. 여기서 radix-2048의 경우 누적 연산기의 수행 시간이 늘어나는 현상이 발생하므로 수행시간이 가장 긴 구간(Critical Path)의 시간이 증가하였다.

5. 결론 및 향후 계획

지금까지 높은 자릿수 나눗셈 연산기에서 영역변환상수를 연산이 아닌 검색테이블을 이용하는 방법을 논하였다. 본 연구는 높은 자릿수 나눗셈 알고리즘에서 연산순환주기를 감소하는 방법으로 영역변환상수를 검색테이블로 구성하였으며, 검색테이블의 크기를 줄이는 방법으로 영역변환상수의 범위를 분석하였고, 캐리 저장형을 이용하여 검색테이블을 분할하는 방법을 이용하였다. 그리고 기본 자릿수가 같은 나눗셈 연산기보다 크기는 조금 커졌지만, 연산 속도는 더 빠르고, 연산순환주기가 같은 나눗셈 연산기보다 크기는 더 작아지면서, 속도는 더 빠름을 확인하였다. 특히, 나눗셈 연산기는 연산순환주기를 기준으로 평가하기 때문에 전체적으로 성능이 향상되었다.

본 연구의 결과는 고속 부동소수점 연산기, 암호화 칩, 멀티미디어 시스템의 연산기 및 고성능의 그래픽 랜더링에 이용할 수 있으며, 나눗셈 연산이 시스템의 병목 현상이 되는 경우 해결책으로 제시될 수 있다. 그리고 높은 자릿수 나눗셈 알고리즘에서 성능 향상을 위한 기본 자릿수 선택에 방향을 제시할 수 있으며, 다른 연산 알고리즘에서 검색테이블을 이용한 방법을 제시할 수 있다.

향후, 검색테이블의 최적화, 최소한의 입력 비트를 이용한 검색테이블 설계, 그리고 다른 알고리즘을 이용한 나눗셈 연산기와의 크기 및 속도에 관한 비교 연구가 이루어져야 할 것이다.

참고문헌

[1] M. D. Erecegovac and T. Lang, "A division algorithm with divisor scaling", in Proc. IEEE 7th Symp. Comput. Arithmetic, pp. 51-56, 1985.

[2] M.D. Ergegovac and T. Lang, "Division and Square Root: Digit-Recurrence Algorithm and Implementations", Kluwer Academic Publisher, 1st edition, 1994.

[3] Stuart F. Oberman and Michael J. Flynn, "Division Algorithms and Implementations", IEEE Trans. Compute., Vol. 46, No. 8, 1997.

[4] M.D. Ergegovac, T. Lang and P.Montuschi, "Very-High Radix Division with Prescaling and Selection by Rounding", IEEE Trans. Comput., vol. 43, pp.909-918, 1994.

[5] Alberto Nannarelli, "Implementation of a Radix-512 Divider", Master thesis, Univ. of Calif., 1995.