The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 23, No. 6, pp.55-60, Dec. 31, 2023. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2023.23.6.55
JIIBC 2023-6-9

이진수 곱셈 문제의 덧셈 최소화 자리이동-덧셈 알고리즘

Algorithm for Addition Minimization Shift-and-Add of Binary Multiplication Problem

이상운*

Sang-Un Lee*

요 약 컴퓨터로 두 이진수 m과 r의 곱셈 $m\times r=p$을 수행함에 있어 시간이 과다 소요되는 곱셈을 전혀 수행하지 않고 단지 덧셈과 우측 자리이동만을 수행하는 방법으로 자리이동-덧셈법이 있다. SA는 승수 r의 자리 수 r_i가 0이면 $m\times 0=0$으로 결과 값 p를 우측 자리이동만 하면 되며, r_i가 1이면 $m\times 1=m$으로 결과 값 $p=p+m$을 수행하고, p를 우측자리이동하면 되는 매우 단순한 방법이다. SA에서 SR 횟수는 더 이상 단축시킬 수 없으며, 단지 개선부분은 덧셈 횟수 단축 여부이다. 본 논문에서는 인간이 수행하는 방식인 10진수를 기준으로 보다 작은 수를 r로 설정하는 경우에 비해 컴퓨터가 처리할 이진수로 변환시켰을 때 1의 개수가 보다 작은 수를 r로 설정하는 방법이 덧셈 횟수를 크게 줄일 수 있다는 점에 착안하여 덧셈 최소화 SA 방법을 제안하였다. 제안된 알고리즘을 [-127,128] 범위에서 일부 숫자를 대상으로 부호가 (-,-), (-,+), (+,-), (+,+)인 4가지 경우에 대해 덧셈 횟수를 비교하였다. 실험 결과 얻은 결론은 m과 r을 결정할 때 10진수가 아닌 2진수로 판단해야 함을 보였다.

Abstract When performing the multiplication $m\times r=p$ of two binary numbers m and r on a computer, there is a shift-and-add(SA) method in which no time-consuming multiplication is performed, but only addition and shift-right(SR). SA is a very simple method in which when the value of the multiplier r_i is 0, the result p is only SR with $m\times 0=0$, and when r_i is 1, the result $p=p+m$ is performed with $m\times 1=m$, and p is SR. In SA, the number of SRs can no longer be shortened, and the improvement part is whether the number of additions is shortened. This paper proposes an SA method to minimize addition based on the fact that setting a smaller number to r when converted to a binary number to be processed by a computer can significantly reduce the number of additions compared to the case of setting a smaller number to r based on the decimals that humans perform. The number of additions to the proposed algorithm was compared for four cases with signs (-,-), (-,+), (+,-), and (+,+) for some numbers in the range [-127,128]. The conclusion obtained from the experiment showed that when determining m and r, it should be determined as a binary number rather than a decimal number.

Key Words : Multiplication, Multiplicand, Multiplier, Shift-and-add, Number of 1's

*정회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
접수일자 2023년 3월 3일, 수정완료 2023년 11월 19일
게재확정일자 2023년 12월 8일

Received: 3 March, 2023 / Revised: 19 November, 2023 / Accepted: 8 December, 2023
*Corresponding Author: sulee@gwnu.ac.kr
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

Ⅰ. 서 론

컴퓨터의 곱셈은 덧셈에 비해 많은 시간이 소요된다. 또한 덧셈은 비트 연산에 비해 많은 시간이 소요된다. 이러한 곱셈의 시간 단축을 위해 곱셈을 전혀 수행하지 않고 단지 덧셈과 비트연산만을 수행하는 방법이 자리이동-덧셈(shift-and-add, SA) 법이다.[1] 왜냐하면 일반적으로 인간이 취급하는 수는 10진수이지만 컴퓨터는 0과 1의 2진수만을 취급하므로 피승수(multiplicand, m)에 승수(multiplier, r)를 곱한 결과(product, p)를 얻는다고 할 경우 $m\times r$은 r의 자리 수 r_i의 값(0 or 1)에 대해 $r_i=1:m\times 1=m, r_i=0:m\times 0=0$이 성립하므로, r의 i번째 자리수를 LSB(least significant bit)로 하여 p에 m을 더해 주면 되기 때문이다. 이와 같이 m의 LSB 위치를 결정하는 방법은 $p=m\cup r$로 설정하고 p를 우측 자리 이동(shift right, SR)하면 된다.[2] 이 방법은 초기에는 m과 r을 분리된 레지스터에 저장하는 형태[1,2]에서 r을 p의 p_{RH}에 저장하여 r 레지스터 없이 p의 LSB를 확인하는 형태[2]로 발전하였다.

SA는 엄밀히 말하면 인간이 사용하는 10진수의 곱셈법을 컴퓨터의 2진수 곱셈법으로 진수변환 한 형태로 최적화 시킨 방법에 불과하다고 할 수 있다. 그럼에도 불구하고 2진수 특성 상 덧셈에 비해 과다한 시간이 소요되는 곱셈을 전혀 하지 않아도 되는 수행시간 단축 효과를 얻을 수 있었다.

SA에서 $m\times r_i$ 계산 결과의 자리 맞춤은 연산자들 중 가장 빠른 비트연산의 일종인 SR로 수행되기 때문에 더 이상 줄일 수 없다. 단지 덧셈 횟수를 줄일 수 있는 방법이 존재한다면 이 부분이 성능개선의 대상이 된다.

인간은 10진수 곱셈을 할 때 편의상 또는 곱셈의 결과를 얻은 수를 최소화 시켜 이들을 합산(덧셈 최소화)한 결과로 최종 결과를 얻고자 하기 때문에 숫자 크기를 비교하여 보다 큰 수를 m으로, 보다 작은 수를 r로 설정한다. 이러한 인간의 본질적 특성을 반영하여 SA의 곱셈 계산을 할 경우 최소의 덧셈 횟수를 얻지 못하는 경우가 발생할 수 있다. 참고로, 10진수 곱셈을 순수한 덧셈으로 계산하는 방법에 대해 Lee[3]는 자리 수 이동 덧셈(shift-and-add) 방법을 제안하기도 하였다.

본 논문에서는 덧셈 횟수 최소화(add minimization, AM)를 달성할 수 있는 SA법을 제안한다. 2장에서는 컴퓨터 곱셈과 관련된 연구 결과와 문제점을 고찰한다. 3장에서는 덧셈을 최소화시키도록 SA의 m과 r을 결정하는 법을 제안한다. 4장에서는 [-127,128] 범위 중 일부 데이터를 추출하여 (m,r)의 부호가 (-,-), (-,+), (+,-), (+,+)인 경우에 대해 기존의 SA법과 본 논문에서 제안된 AMSA의 덧셈 횟수를 비교하여 AMSA의 우수성을 검증해 본다.

Ⅱ. 관련연구와 문제점

컴퓨터 연산시간 빠르기 순서는 비트 수를 n이라 할 경우, 표 1과 같이 비트연산〉덧셈〉뺄셈〉곱셈〉나눗셈으로 알려져 있다.

곱셈연산을 가장 많이 수행하는 분야가 행렬곱셈(matrix multiplication, MM)이며, 이 문제는 컴퓨터 공학 분야에서 가장 빠른 행렬 곱셈법이 알려져 있지 않는 난제(미 해결 문제)로 남아있다.[4,5]

표 1. 연산자 수행시간 복잡도
Table 1. Time complexity of Operators

연산자	시간 복잡도	병렬처리
비트 연산 (Shift, AND, OR, NOT, XOR)	$O(n)$	$O(1)$
덧셈	$O(n^2)$	$O(n)$
곱셈	-	$O(2n^n)$

컴퓨터로 덧셈을 수행하는 시간에 비해 곱셈은 수십 배의 시간이 소요된다. 이와 같은 이유로 인해, 행렬곱셈 문제에 대해 전형적인 계산 방법에 비해 곱셈 횟수를 단축시키는 방법은 표 2와 같은 연구 결과를 얻었다.[6-8]

표 2. 행렬 곱셈 단축 알고리즘
Table 2. Matrix multiplication reduction algorithm

$A\cdot B=C$			수행횟수(증감)			
			전형적 방법		Reference[1-3]	
A	B	C	곱셈	덧셈	곱셈	덧셈
(2x2)	(2x2)	(2x2)	8	4	7(-1)	19(+15)
(3x3)	(3x3)	(3x3)	27	18	23(-4)	98(+80)
					23(-4)	107(+89)

$$\begin{pmatrix}a_{11}\,a_{12}\\ a_{21}\,a_{22}\end{pmatrix}\times\begin{pmatrix}b_{11}\,b_{12}\\ b_{21}\,b_{22}\end{pmatrix}=\begin{pmatrix}c_{11}\,c_{12}\\ c_{21}\,c_{22}\end{pmatrix}=\begin{pmatrix}a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22}\\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22}\end{pmatrix}$$

$$\begin{pmatrix}a_{11}\,a_{12}\,a_{13}\\ a_{21}\,a_{22}\,a_{23}\\ a_{31}\,a_{32}\,a_{33}\end{pmatrix}\times\begin{pmatrix}b_{11}\,b_{12}\,b_{13}\\ b_{21}\,b_{22}\,b_{23}\\ b_{31}\,b_{32}\,b_{33}\end{pmatrix}=$$

$$\begin{pmatrix}a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} & a_{11}b_{13}+a_{12}b_{23}+a_{13}b_{33}\\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} & a_{21}b_{13}+a_{22}b_{23}+a_{23}b_{33}\\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32} & a_{31}b_{13}+a_{32}b_{23}+a_{33}b_{33}\end{pmatrix}$$

이와 같이 (2x2)와 (3x3)의 단순한 행렬에 대해서도 곱셈 횟수를 1회 감축시키기 위해 덧셈/뺄셈 횟수가 평균 약 20회 증가하는 결과를 얻었으며, $(m\times n)\cdot(n\times m)=(m,m)$의 행렬곱셈에서 $m\neq n$인 경우에 대한 연구결과가 없을 뿐 아니라 4 이상의 행렬에 대한 연구결과도 없는 실정이다.

행렬 곱 계산을 가장 많이 적용하고 있는 분야는 최근에 각광을 받고 있는 인공지능의 딥러닝 신경망(deep learning neural network)이다. 만약 100단(층), 각 층의 뉴런 수 100개가 가중치들로 연결된 경우 각 단(layer)에는 (100x100)x (100x100)=(100x100)의 $\boldsymbol{Y}=\sum_{i=1}^{n}\boldsymbol{X}\cdot\boldsymbol{W}$ 행렬 곱을 계산해야 한다. 여기서 하나의 y_{ij}를 얻는데 곱셈은 100회, 덧셈은 99회로 (100x100)의 $\boldsymbol{Y}$를 얻는 데는 덧셈은 990,000, 곱셈은 1,000,000을 수행해야 한다. 이를 100단으로 계산하면 덧셈은 99,000,000, 곱셈은 100,000,000회이다.

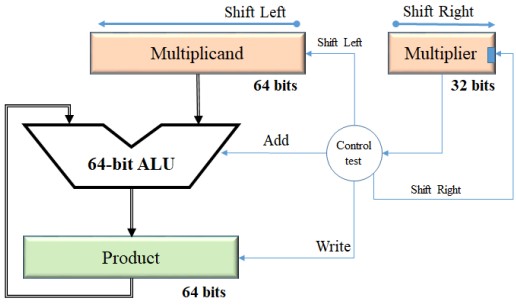
또한 이는 데이터 진행의 순방향과 오차 감소의 역방향으로 2회를, 학습 시행횟수를 1,000회라 가정한다면 1,000x2 =2,000회를 곱해야 하므로 결론적으로 덧셈은 198,000,000, 000회, 곱셈은 200,000,000,000(2천억)회를 수행해야 한다.

이와 같이 엄청난 횟수의 곱셈 수행시간을 단축시키기 위해 곱셈과 나눗셈을 덧셈만으로 처리하는 방법이 자리이동-덧셈(SA)법이다.[1,2,9-12] 여기서 l 비트의 피승수(m)와 n 비트의 승수(r)을 곱하여 결과(p)를 얻는 경우를 가정한다. 이 때 $l\geq r$이며, p는 $2l$ 비트이다.

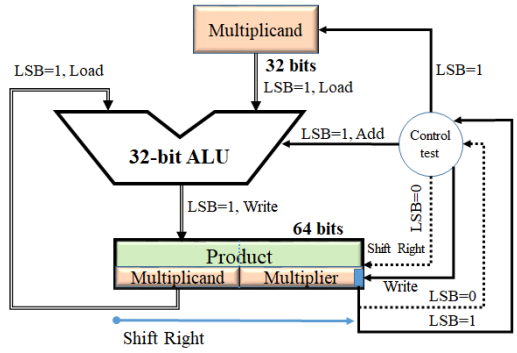
SA는 $p=p_{LH}(m)\cup p_{RH}(r)$로 설정하고, p를 우측 자리이동(SR)시키면서 LSB=0이면 자리이동만을, LSB=1이면 $p=p_{LH}+m$과 자리이동을 시키는 단순한 방법이다. 따라서 자리이동 횟수는 l회, 덧셈 횟수는 r이 갖고 있는 '1'의 개수(number of 1's, k)가 된다.[2]

Booth[1]의 SA에 기반하여 컴퓨터 회로 설계 분야에서는 이와 같은 SA가 그림 1과 같이 구현되어 적용되고 있다.[2]

결국 난제로 알려진 행렬 곱 문제에 대해 SA는 곱셈횟수를 0로 하는 대신에 덧셈만을 수행하여 수행시간을 현격하게 단축시킨 결과를 얻었다. 이는 겉으로 보기에는 표 3과 같이 두 수 a,b를 곱하여 c를 얻는 $a\times b=c$의 초등학교법(grade school method)에 비해 SA는 $O(n^2)$ 수행 복잡도의 $l\times n$회 곱셈을 전혀 수행하지 않기 위해 n회의 우측 자리이동 방법을 적용하였으며, n회의 덧셈 횟수를 $k(k\leq n)$회로 줄여 연산시간을 획기적으로 단축시켰다.



(a) 64-bits ALU multiplier



(b) 32-bits ALU multiplier

그림 1. 자리이동-덧셈 곱셈기 회로
Fig. 1. Shift-and-Add multiplier circuit

표 3. 곱셈의 곱셈과 덧셈 횟수
Table 3. Number of multiply and addition for multiplication

연산자 수행 복잡도	초등학교 법	SA
곱셈 : $O(n^2)$	$l\times n$	0
덧셈 : $O(n)$	n	$k(k\leq n)$
자리이동 비트연산 : $O(1)$	0	n

그러나 실제로 상세하게 내용을 살펴보면 그림 2와 같이 초등학교법은 인간이 사용하는 10진수의 곱셈이며, SA는 컴퓨터가 다루는 2진수의 특성을 반영하여 $m_{(2)}\times 1_{(2)}=m_{(2)}+m_{(2)}$로 r의 비트 단위 곱셈을 덧셈으로 변경하였으며, 이 결과를 r의 "1" 위치로 우측 정렬시키는 과정은 우측 자리이동(SR)로 해결하였다는 점이다. 결론적으로 SA는 곱셈을 하는데 있어 인간의 10진수 곱셈법을 컴퓨터의 2진수 곱셈법으로 단순 전환시켰을 뿐 특이한 기술혁신이 이루어진 것이 아니라고 할 수 있다.

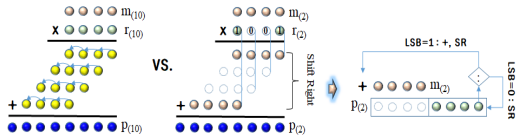


그림 2. 10진수와 2진수 곱셈법
Fig. 2. Multiplication of Decimal and Binary

$m \times r$의 의미는 피승수 m을 승수 r배하라는 의미로, SA에서 자리이동 횟수를 결정하는 r의 자리 수 n은 더 이상 단축시킬 수 없다. 그러나 덧셈 횟수는 보다 줄일 수 있다. 이에 대한 해결법을 3장에서 논한다.

Ⅲ. 피승수와 승수 결정 알고리즘

10진수의 곱셈 계산을 할 경우 l 비트길이 피승수 m과 n 비트 길이 승수 r을 결정할 때 일반적으로는 $l \geq r$을 선호한다. 왜냐하면 큰 수에서 작은 수를 곱하는 $m \times r = p$ 형태가 r의 자리 수만큼 각 비트 값에 m을 곱한 결과가 n개 생성되며 n개의 생성된 결과를 모두 더하면 최종적으로 p를 얻기 때문이다. 덧셈 수행 횟수를 보다 단축시킬 수 있기 때문이다. 그러나 컴퓨터는 2진수를 사용하기 때문에 $l \geq r_{(10)} = l \geq r_{(2)}$가 반드시 성립하지 않는 경우가 많다는 점이다. 따라서 본 장에서는 두 수 a와 b가 양의 정수 또는 음의 정수와 무관하게 10진수를 2진수로 변환시키고, 2진수의 '1'의 자리 수(number of 1's, k)를 비교하여 $\max\{k_a, k_b\}$를 m으로, $\min\{k_a, k_b\}$를 r로 하는 SA를 제안한다. 제안된 방법을 덧셈 최소화 자리이동-덧셈(AMSA) 알고리즘이라 하며 그림 3과 같이 수행된다.

입력 : a, b
a, b에 대한 2진수의 '1' 빈도수 a_k, b_k 계산.
if $a_k \geq b_k$ then $m \leftarrow a, r \leftarrow b$
else $m \leftarrow b, r \leftarrow a$
m의 자리 수 : l, r의 자리 수 : n.
p 자리 수 = $\max\{l, n\} \times 2$, $p_{LH} = 0$, $p_{RH} \leftarrow r$.
for $i = 1$ to n
  if p의 LSB=0 then p shift right(SR)
  else if p의 LSB=1 then
    $p_{LH} \leftarrow p_{LH} + m$, p shift right(SR)
  end if
end for

그림 3. 덧셈 최소화 자리이동-덧셈 알고리즘
Fig. 3. Addition minimization shift-and-add algorithm

Booth[1]에서 인용된 다음의 예제를 대상으로 SA와 AMSA의 덧셈 수행 횟수를 비교하여 본 결과는 표 4에 제시되어 있다.

표 4. 덧셈 횟수 비교
Table 4. Comparison with the number of addition

a	b	Booth's SA				AMSA			
		m	r	+/-	SR	m	r	+	SR
6(0100)	5(0101)	6	5	4	3	5	6	1	4
6(0100)	-5(1011)	6	-5	3	3	-5	6	1	4
-6(1010)	5(0101)	-6	5	4	3	-6	5	2	4
-5(1011)	-6(1010)	-6	-5	3	3	-5	-6	2	4

Booth[1]는 부호에 상관없이 10진수를 기준으로 큰 수를 m, 작은 수를 r로 취급한 반면에 AMSA는 2진수의 '1'의 개수가 많은 수를 m으로, 적은 수를 r로 설정하였다. 두 알고리즘을 비교한 결과 AMSA가 Booth[1] 법에 비해 덧셈 수행횟수를 보다 줄일 수 있음을 알 수 있다. 또한 Booth[1]는 덧셈/뺄셈을 수행하는 방법인데 비해 AMSA는 덧셈만을 수행한다. 우측 자리이동(SR)은 Booth[1] 방법이 '1' 적으나 이는 실제로 구현된 SA에서는 AMSA와 같이 r의 자리 수인 '4'와 동일하다.

Ⅳ. 알고리즘 적용 및 결과 분석

본 장에서는 8 bit를 적용한 숫자 표현 범위 [-127,128]에서 양수와 음수 각에서 '1'의 개수가 가장 적은 경우와 가장 큰 경우의 값들만을 대상으로 SA와 AMSA의 덧셈 수행 횟수를 비교하여 본다. SA는 부호에 상관없이 10진수 기준 큰 수를 m, 작은 수를 r로 설정하며, AMSA는 2진수의 '1'의 개수가 많은 수를 m으로, 적은 수를 r로 설정한다고 가정한다. 실험은 (-,-), (-,+), (+,-)와 (+,+)인 경우에 대해 수행하였으며, 결과는 표 5에 제시되어 있다.

실험 결과 단순히 부호를 고려하지 않고 10진수를 기준으로 보다 큰 수를 피승수(m)로, 작은 수를 승수(r)로 하는 SA에 비해 2진수에 대한 '1'의 개수가 보다 큰 수를 m으로, 작은 수를 r로 하는 AMSA가 덧셈 횟수를 최소화 시킬 수 있는 효과를 거두었다.

여기서는 극단적인 경우만을 고려하였지만 일반적으로 10진수 기준으로 보다 큰 수가 이진수 변환 시 '1'의 개수도 많은 경향을 나타내므로 이 경우에 대해서는 SA와 AMSA가 동일한 결과를 얻는다.

표 5. 실험 데이터에 대한 덧셈 횟수 비교
Table 5. Comparison with the number of addition for experimental data

a	b	SA			AMSA		
		m	r	+	m	r	+
-127(10000001)	-65(10111111)	-127	-65	7	-65	-127	2
-126(10000010)	-33(11011111)	-126	-33	7	-33	-126	2
-124(10000100)	-17(11101111)	-124	-17	7	-17	-124	2
-120(10001000)	-9(11110111)	-120	-9	7	-9	-120	2
-112(10010000)	-5(11111011)	-112	-5	7	-5	-112	2
-96(10100000)	-3(11111101)	-96	-3	7	-3	-96	2
-64(11000000)	-2(11111110)	-64	-2	7	-2	-64	2
-127(10000001)	126(01111110)	-127	126	6	126	-127	2
-126(10000010)	125(01111101)	-126	125	6	125	-126	2
-124(10000100)	123(01111011)	-124	123	6	123	-124	2
-120(10001000)	119(01110111)	-120	119	6	119	-120	2
-112(10010000)	111(01101111)	-112	111	6	111	-112	2
-96(10100000)	95(01011111)	-96	95	6	95	-96	2
-64(11000000)	63(00111111)	-64	63	6	63	-64	2
64(01000000)	-33(11011111)	64	-33	7	-33	64	1
32(00100000)	-17(11101111)	32	-17	7	-17	32	1
16(00010000)	-9(11110111)	16	-9	7	-9	16	1
8(00001000)	-5(11111011)	8	-5	7	-5	8	1
4(00000100)	-3(11111101)	4	-3	7	-3	4	1
2(00000010)	-1(11111111)	2	-1	8	-1	2	1
4(00000100)	3(00000011)	4	3	2	3	4	1
8(00001000)	7(00000111)	8	7	3	7	8	1
12(00001100)	11(00001011)	12	11	3	11	12	2
12(00001100)	7(00000111)	12	7	3	7	12	2
16(00010000)	15(00001111)	16	15	4	15	16	1
32(00100000)	31(00011111)	32	31	5	31	32	1
64(01000000)	63(00111111)	64	63	6	63	64	1
96(01100000)	95(01011111)	96	95	6	95	96	2

결론적으로 10진수 기준으로 보다 작은 수를 r로 설정하기 보다는 2진수의 '1'의 개수 기준으로 보다 작은 수를 r로 설정하는 AMSA 알고리즘이 덧셈 횟수를 최소화 시킬 수 있음을 보였다.

V. 결 론

본 논문에서는 컴퓨터로 두 수 m과 r의 곱셈 $m \times r = p$을 수행함에 있어 곱셈을 전혀 수행하지 않고 단지 덧셈과 우측 자리이동만을 수행하는 SA를 고찰하였다.

SA는 2진수에 대해 승수 r의 자리 수 r_i가 '0'이면 $m \times 0 = 0$으로 결과 값 p를 SR하면 되며, r_i가 '1'이면 $m \times 1 = m$으로 결과 값 $p = p + m$을 수행하고, p를 SR하면 되는 매우 단순한 방법이다.

따라서 SA에서 SR 횟수는 더 이상 단축시킬 수 없다. 다만 덧셈 횟수를 어떻게 하면 단축시킬 수 있는가에 대한 해결책만이 개선 가능한 부분이다. 이에 대해 본 논문에서는 인간이 수행하는 방식인 10진수 기준 보다 큰 수를 m으로, 보다 작은 수를 r로 설정하는 경우에 비해 컴퓨터가 처리할 이진수로 변환시켰을 때 '1'의 개수가 보다 큰 수를 m으로, 보다 작은 수를 r로 설정하는 방법이 덧셈 횟수를 크게 줄일 수 있다는 점에 착안하여 덧셈 최소화 SA 방법을 제안하였다.

제안된 알고리즘을 [-127,128] 범위에서 일부 숫자를 대상으로 부호가 (-,-), (-,+), (+,-), (+,+)인 4가지 경우에 대해 기존의 SA법과 본 논문에서 제안된 AMSA의 덧셈 횟수를 비교하여 AMSA의 우수성을 검증하였다. 검증결과 AMSA가 보다 우수함을 보여 m과 r을 결정할 때 10진수가 아닌 2진수로 판단해야 함을 보였다.

References

[1] A. D. Booth, "A Signed Binary Multiplication Technique," The Quarterly Journal of Mechanics and Applied Mathematics, Vol. 4, No. 2, pp. 236-240, Jan. 1951, https://doi.org/10.1093/qjmam/4.2.236

[2] B. Z. Francisc, "Structure of Computer Systems : 3.2.1 Shift-and-Add Multiplication," pp. 62-65, Technical University of Cluj-Napoca, Computer Science Department, U. T. PRES. 2002.

[3] S. U. Lee, "Shift-and-Add Multiplication Algorithm for Decimal System," The Journal of The Institute of Internet, Broadcasting and Communication, Vol. 14, No. 2, pp. 121-126, Apr. 2014, https://doi.org/10.7236/JIIBC.2014. 14.2.121

[4] Wikipedia, "List of unsolved problems in computer science," https://en.wikipedia.org/wiki/List_of_unsolved_problems_in_computer_science, Retrieved Mar. 2022.

[5] Wikipedia, "Computational complexity of matrix multiplication," https://en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication, Retrieved Mar. 2022.

[6] V. Strassen, "Gaussian Elimination is not Optimal," Numerical Mathematics, Vol. 13, No. 4, pp. 354-356, 1969, https://doi.org/10.1007/BF02165411

[7] J. B. Laderman, "A Noncommutative Algorithm for Multiplying 3×3 Matrices Using 23 Multiplications," Bulletin of the American Mathematical Society, Vol. 82, No. 1, pp. 126-128, Jan. 1976.

[8] N. T. Courtois, G. V. Bard, and D. Hulme, "A New General-Purpose Method to Multiply 3x3 Matrices Using Only 23 Multiplications," arXiv:1108.2830, pp. 1-10, Aug. 2011.

[9] M. Azarmehr, "Multipliers, Algorithms, and Hardware Designs," Research Centre for Integrated Microsystems, University of Windsor, pp. 1-42, Aug. 2009.

[10] K. Johansson, "Low Power and Low Complexity Shift-and-Add Based Computations," Department of Electrical Engineering, Linköping University, 2008.

[11] S. V. Padmajarani and M. Muralidhar, "FPGA Implementation of Multiplier Using Shift and Add Technique," International Journal of Advances in Electronics and Computer Science, Vol. 2, No. 9, pp. 1-5, Sep. 2019.

[12] Z. M. Patel, "Enhancing Speed and Reducing Power of Shift and Add Multiplier," International Journal of Electrical, Electronics and Data Communication, Vol. 4, No. 6, pp. 13-17, Jul. 2016.

저 자 소 개

이 상 운(정회원)

- 1987년 : 한국항공대학교 항공전자공학과 (학사)
- 1997년 : 경상대학교 컴퓨터과학과 (석사)
- 2001년 : 경상대학교 컴퓨터과학과 (박사)
- 2003년 : 강원도립대학 컴퓨터응용과 전임강사
- 2004년 ~ 2007.2 : 국립 원주대학 여성교양과 조교수
- 2007.3 ~ 2015.3 : 강릉원주대학교 멀티미디어공학과 부교수
- 2015.4 ~ 현재 : 강릉원주대학교 멀티미디어공학과 정교수
- 관심분야 : 소프트웨어 프로젝트 관리, 개발 방법론, 분석과 설계 방법론, 시험 및 품질보증, 소프트웨어 신뢰성, 인공지능과 빅데이터분석, 최적화 알고리즘
- e-mail : sulee@gwnu.ac.kr