

Karatsuba 곱셈의 적정 FLAG 값 설정에 대한 연구

- 효율성과 최적화를 위한 경제값 분석 -

지용현 | 문예찬 | 김예찬 | 유근오

국민대학교 과학기술대학 정보보안암호수학과

hacker3740@gmail.com | dlsnditi7845@gmail.com | 1018dhkdwk@gmail.com | geuno0109@gmail.com



소개

컴퓨터 과학 분야에서 알고리즘 최적화는 핵심적인 역할을 차지합니다. 본 연구는 기존의 TextBook 곱셈 방법보다 효율적으로 큰 두 정수를 곱할 수 있는 분할 정복 알고리즘인 32-bit 워드 단위 Karatsuba 곱셈에 집중하고 있습니다. 이 연구의 목적은 C 언어로 구현된 32-bit 워드 단위 Karatsuba 곱셈의 성능을 TextBook 곱셈 및 그 변형된 방식들과 비교 평가하는 것입니다. 또한, 실험 분석을 통해 플래그 (FLAG) 값의 최적 설정을 찾아내어 계산 속도의 향상을 도모하고자 합니다.

TextBook vs Improved TextBook vs Karatsuba

본 실험의 모든 워드 길이는 32-bit로 설정합니다.

(1) TextBook 곱셈

X와 Y를 각각 n-워드, m-워드 정수라고 가정합니다. 즉,

$$\begin{cases} X = \sum_{i=0}^{n-1} x_i(2^{32})^i =: x_{n-1} \parallel \cdots \parallel x_0 \\ Y = \sum_{j=0}^{m-1} y_j(2^{32})^j =: y_{m-1} \parallel \cdots \parallel y_0 \end{cases}, \quad x_i, y_i \in [0, 2^{32}).$$

TextBook 곱셈은 다음과 같이 진행됩니다.

	x_2	x_1	x_0
\times	y_2	y_1	y_0
			x_0y_0
		x_1y_0	
	x_2y_0		
		x_0y_1	
	x_1y_1		
	x_2y_1		
		x_0y_2	
	x_1y_2		
	x_2y_2		

$$\begin{aligned} Z = XY &= \left(\sum_{i=0}^{n-1} x_i(2^{32})^i \right) \left(\sum_{j=0}^{m-1} y_j(2^{32})^j \right) \\ &= \sum_{j=0}^{m-1} \left(\sum_{i=0}^{n-1} x_i y_j (2^{32})^{i+j} \right) \in [0, (2^{32})^{n+m}). \end{aligned}$$

(2) Improved TextBook 곱셈

(1)에서의 TextBook 곱셈을 다음과 같이 변형시켜 덧셈의 횟수를 줄일 수 있습니다.

	x_3	x_2	x_1	x_0
\times	y_3	y_2	y_1	y_0
			x_2y_0	x_0y_0
		x_3y_0	x_1y_0	
		x_2y_1	x_0y_1	
		x_3y_1	x_1y_1	
		x_2y_2	x_0y_2	
	x_3y_2	x_1y_2		
	x_2y_3	x_0y_3		
	x_3y_3	x_1y_3		

$n \times m$ 크기의 곱셈에서의 덧셈 횟수를

$$(nm - 1) \times \text{add} \rightarrow \left(\left\lfloor \frac{nm}{2} \right\rfloor - 1 \right) \times \text{add}$$

로 줄일 수 있습니다.

(3) Kratsuba 곱셈 ($O(n^{\log_2 3})$)

두 큰 정수 $X \in [(2^{32})^{n-1}, (2^{32})^n)$ 와 $Y \in [(2^{32})^{m-1}, (2^{32})^m)$ 에 대해 다음을 생각합니다.

$$X = x_1(2^{32})^l + x_0, \quad Y = y_1(2^{32})^l + y_0, \quad x_i, y_i \in [0, (2^{32})^l), \quad l = \left\lfloor \frac{\max(n, m) + 1}{2} \right\rfloor.$$

Karatsuba 곱셈은 다음 상기 식을 통해 이용해 계산합니다.

$$\begin{aligned} XY &= (x_1(2^{32})^l + x_0)(y_1(2^{32})^l + y_0) \\ &= ((x_1y_1)(2^{32})^{2l} + (x_0y_0)) + (x_0y_1 + x_1y_0)(2^{32})^l \\ &= ((x_1y_1)(2^{32})^{2l} + (x_0y_0)) + ((x_0 - x_1)(y_1 - y_0) + x_0y_0 + x_1y_1)(2^{32})^l. \end{aligned}$$

구현한 Karatsuba 곱셈의 의사코드는 다음과 같습니다.

Algorithm 1: Karatsuba 곱셈

Input: FLAG, $X = \sum_{i=0}^{n-1} x_i(2^{32})^i, Y = \sum_{j=0}^{m-1} y_j(2^{32})^j$, where $x_i, y_i \in [0, 2^{32})$

Output: $Z = XY \in [0, (2^{32})^{n+m})$

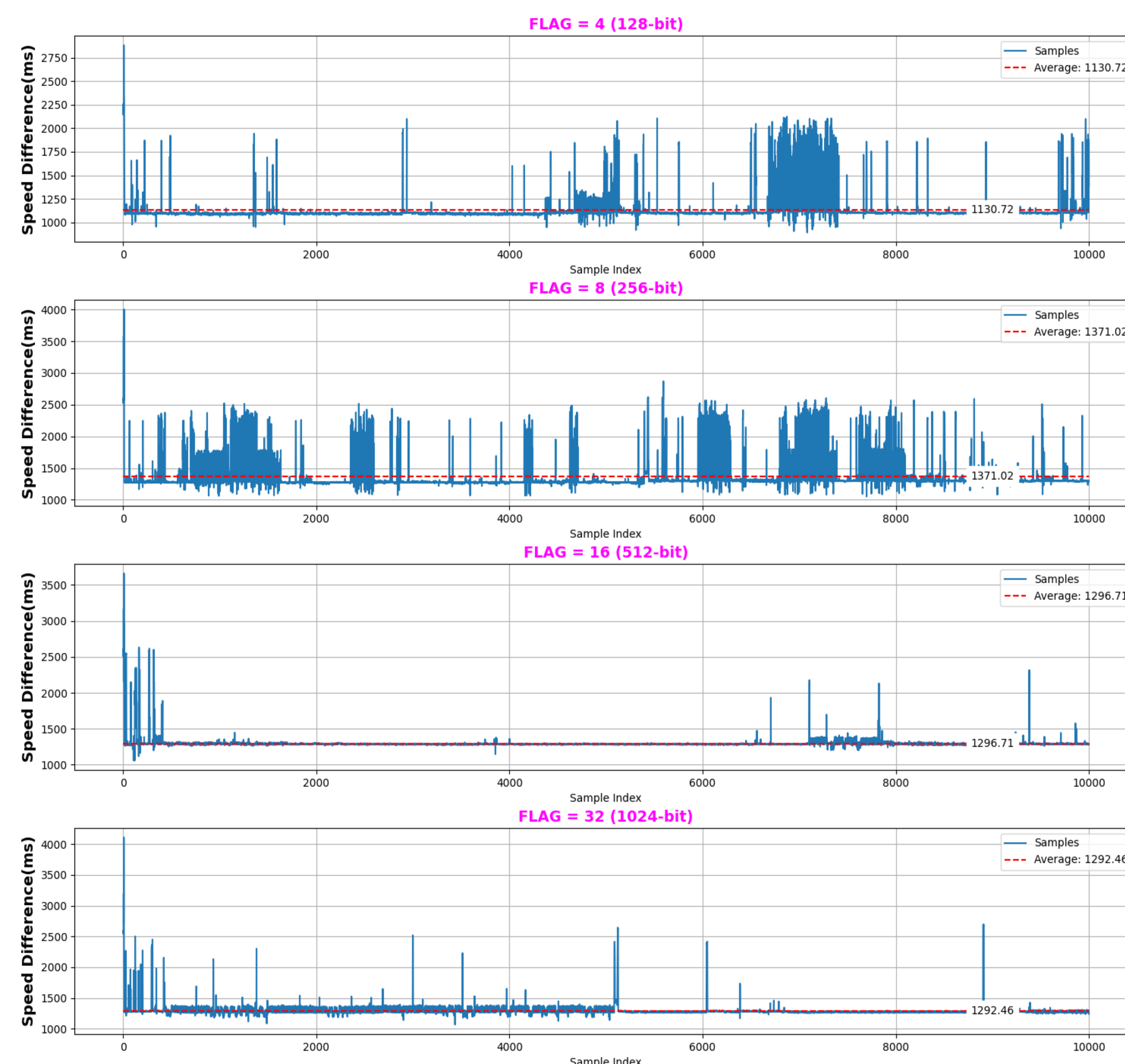
```
1 Function MULKrtsb(X, Y):
  /* 특정 FLAG 길이에서는 (2)에서의 변형된 TextBook 곱셈으로 처리한다. */
2 if FLAG ≥ min(n, m) then
3   return MULImpTxtBk(X, Y); // Improved TextBook Multiplication
4 end
5 l ← max(n, m + 1) >> 1;
  /* X와 Y를 분할한다. */
6 x1, x0 ← X >> l * 32, X mod 2l*32; // xi ∈ [0, (232)l)
7 y1, y0 ← Y >> l * 32, Y mod 2l*32; // yi ∈ [0, (232)l)
8 T1, T0 ← MULKrtsb(x1, y1), MULKrtsb(x0, y0); // Ti ∈ [0, (232)2l)
9 Z ← (T1 << 2l * 32) + T0; // Z = T1 || T0 ∈ [0, (232)4l)
10 S1, S0 ← SUB(x0, x1), SUB(y1, y0); // 큰 두 정수의 뺄셈 함수를 구현하여 사용
11 S ← (-1)sgn(S1) ⊕ sgn(S2) MULKrtsb(|S1|, |S0|);
12 S ← ADD(S, T1); // 큰 두 정수의 덧셈 함수를 구현하여 사용
13 S ← ADD(S, T0);
14 S ← S << l * 32;
15 return ADD(Z, S);
16 end
```

성능 측정 환경

- **Processor:** AMD Ryzen 7 5800X3D, 3400 MHz, 8-Core
- **Memory:** 32 GB, DDR4-3200 (16GB) PC4-25600 x2
- **Operating System:** Linux Mint 21.1 Vera x86 64
- **Compiler Version:** gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.

FLAG 값에 따른 속도 차이

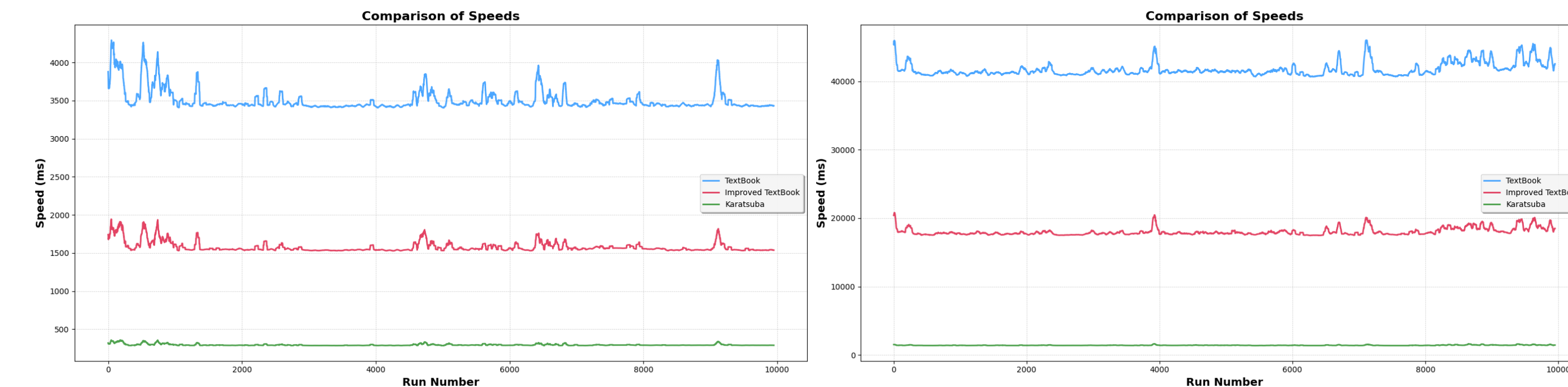
FLAG값의 변화에 따라 속도가 얼마나 차이 나는지에 대한 실험결과는 다음과 같습니다.



Improved TextBook 곱셈 속도와 Karatsuba 곱셈의 속도 차이 분포를 위 그림과 같이 시각화하여 분석한 결과, 3072-bit와 7680-bit 모두 FLAG 값이 8로 설정되었을 때, 즉, 256-bit 이하에서는 Improved TextBook 곱셈으로 처리하였을 때 가장 큰 속도 차이를 보이는 것으로 확인하였습니다.

속도 측정

랜덤한 수는 rand() 함수를 이용해 생성하였으며, 시간측정은 clock() 함수를 이용하였습니다. FLAG 값은 위 결과에 따라 8로 설정하였습니다.



3072-bit 곱셈

7680-bit 곱셈

큰 두 정수의 곱셈을 처리하는 데 10000개의 샘플에 대한 평균적인 속도는 다음 표와 같습니다.

방식	3072-bit (ms)	상대 속도(s)	7680-bit (ms)	상대 속도(s)
TextBook	3339	2.356	40331	2.493
Improved TextBook	1417	1	16176	1
Karatsuba	296	0.209	1516	0.094

결론

본 연구는 서로 다른 플래그 값에 대해 Karatsuba 곱셈 알고리즘의 성능을 평가하였습니다. 다양한 데이터셋을 분석한 결과, 3072-bit 및 7680-bit 곱셈을 32-bit 워드 단위로 처리할 때 FLAG 값을 8로 설정하는 것이 계산 속도와 자원 활용의 최적 균형을 제공하는 것으로 나타났습니다. 이러한 발견은 Karatsuba 곱셈 알고리즘의 최적화 및 응용에 있어서 중요한 시사점을 제공합니다. 특정 응용 프로그램이나 하드웨어 환경에서 최적의 성능을 달성하기 위해서는 알고리즘의 FLAG 설정을 세심하게 조정할 필요가 있습니다. 이 연구는 향후 효율적인 곱셈 알고리즘 선택과 고성능 컴퓨팅 설계에 유용한 참고 자료가 될 것입니다.

참고 문헌

김동찬. (2023). 고급응용프로그래밍 [Big Integer Arithmetic]. 정보보안암호수학과, 국민대학교.