Kratsuba 곱셈의 적정 Flag 값 설정에 대한 연구

- 효율성과 최적화를 위한 경계값 분석 -

지용현 | 문예찬 | 김예찬 | 유근오

국민대학교 과학기술대학 정보보안암호수학과

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

소개

컴퓨터 과학 분야에서 알고리즘의 최적화는 매우 중요합니다. 본 연구는 큰 두 정수를 TextBook 곱셈과 같은 기존의 방법보다 더 효율적으로 곱하는 분할 정복 알고리즘인 32-bit 워드 단위 Karatsuba 곱셈에 초점을 맞추고 있습니다. 이 연구는 C 언어로 구현된 32-bit 워드 단위 Karatsuba 곱셈의 성능을 TextBook 곱셈 및 그보다 변형된 TextBook 곱셈과 비교 평가하는 것을 목적으로 합니다. 또한, 실험 분석을 통해 플래그(flag) 값의 최적 설정을 탐구하여 계산 속도를 향상시키고자 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,

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

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				

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

(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_0	y_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					

(3) **Kratsuba 곱셈**

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

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

Karatsuba 곱셈은 다음 상기 식을 통해 이용해 계산합니다.

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

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 $x_i, y_i \in \left[0, 2^{32}\right)$

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

Function $\mathsf{MUL}^{\mathsf{Krtsb}}(X, Y)$**:**

  /* 특정 길이 이하에서는 (2)에서의 변형된 TextBook 곱셈으로 처리한다. */

  if $FLAG \geq \min(n, m)$ **then**

    return $\mathsf{MUL}^{\mathsf{ImpTxtBk}}(X, Y)$; // Improved TextBook Multiplication

  end

  $l \leftarrow \max(n, m+1) \gg 1$;

  /* X와 Y를 분할한다. */

  $x_1, x_0 \leftarrow X \gg l * 32, X \bmod 2^{l*32}$; // $x_i \in \left[0, (2^{32})^l\right)$

  $y_1, y_0 \leftarrow Y \gg l * 32, Y \bmod 2^{l*32}$; // $y_i \in \left[0, (2^{32})^l\right)$

  $T_1, T_0 \leftarrow \mathsf{MUL}^{\mathsf{Krtsb}}(x_1, y_1), \mathsf{MUL}^{\mathsf{Krtsb}}(x_0, y_0)$; // $T_i \in \left[0, (2^{32})^{2l}\right)$

  $Z \leftarrow (T_1 \ll 2l * 32) + T_0$; // $Z = T_1 \parallel T_0 \in \left[0, (2^{32})^{4l}\right)$

  $S_1, S_0 \leftarrow \mathsf{SUB}(x_0, x_1), \mathsf{SUB}(y_1, y_0)$; // 큰 두 정수의 뺄셈 함수를 구현하여 사용

  $S \leftarrow (-1)^{\mathsf{sgn}(S_1) \oplus \mathsf{sgn}(S_2)} \mathsf{MUL}^{\mathsf{Krtsb}}(|S_1|, |S_0|)$;

  $S \leftarrow \mathsf{ADD}(S, T_1)$; // 큰 두 정수의 덧셈 함수를 구현하여 사용

  $S \leftarrow \mathsf{ADD}(S, T_0)$;

  $S \leftarrow S \ll l * 32$;

  return $\mathsf{ADD}(Z, S)$;

end

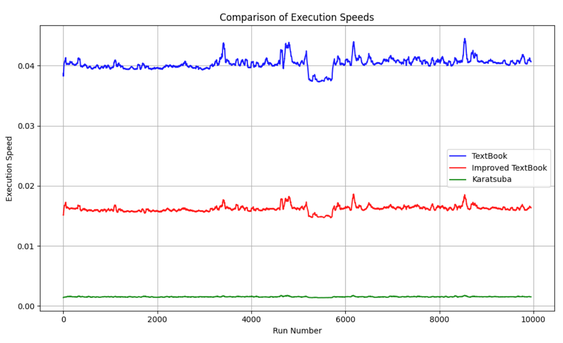
속도 측정

하드웨어 환경은 다음과 같습니다.

- **Processor:** AMD Ryzen 7 5800X3D, 3400 MHz, 8-Core
- **Memory:** 32 GB, DDR4-3200 (16GB) PC4-25600 ×2
- **Operating System:** Linux Mint 21.1 Vera x86_64
- **Compiler Version:** gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0

랜덤한 수는 `rand()` 함수을 이용해 생성하였으며, 시간측정은 `clock()` 함수를 이용하였습니다. FLAG 값은 8로 설정하였습니다.

3072-bit 크기의 두 수의 곱셈 속도



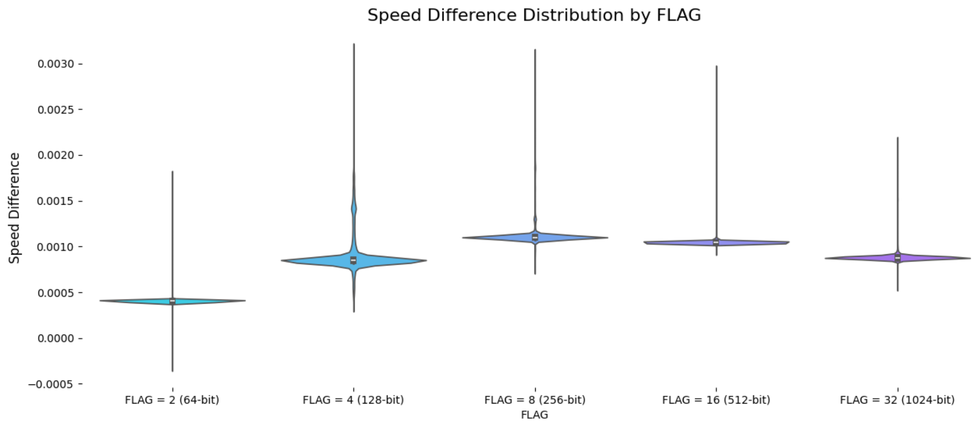
7680-bit 크기의 두 수의 곱셈 속도

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

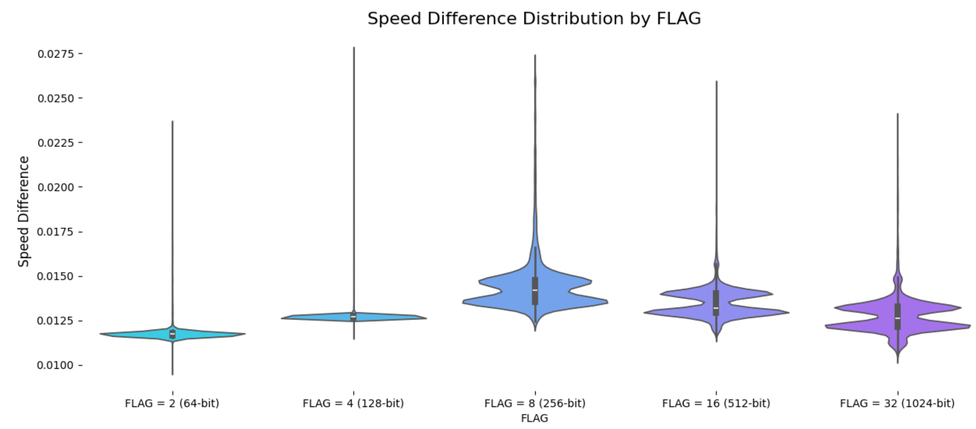
방식	3072-bit		7680-bit	
TextBook	0.003339s	$x_1 := 0.003339$s	0.040331s	$x_2 := 0.040331$s
Improved TextBook	0.001417s	$y_1 \approx 0.42x_1$	0.016176s	$y_2 \approx 0.40x_2$
Karatsuba	0.000296s	$0.2y_1 \approx 0.09x_1$	0.001516s	$0.09y_2 \approx 0.04x_2$

Flag 값에 따른 속도 차이

Flag값의 변화에 따라 속도가 얼마나 차이나는지에 대한 실험결과는 다음과 같습니다.



3072-bit 크기의 두 수의 곱셈



7680-bit 크기의 두 수의 곱셈

Improved TextBook 곱셈 속도와 Karatsuba 곱셈의 속도 차이 분포를 바이올린 플롯으로 시각화하여 분석한 결과, 3072-bit 와 7680-bit 모두 flag 값이 8로 설정되었을 때, 즉, 256-bit 이하에서는 Improved TextBook 곱셈으로 처리하였을 때 가장 큰 속도 차이를 보이는 것으로 확인하였습니다.

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.
또한, 7680-bit 곱셈에서는 flag 값이 8 이상부터는 성능의 변동성이 크고 분포가 넓어짐을 확인할 수 있었습니다. 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.