$GF(2^n)$ 상에서 병렬 멱승 연산의 프로세서 바운드 향상 기법

김윤정○ 박근수 조유근
서울대학교 컴퓨터공학부
yjkim@ssrnet.snu.ac.kr, kpark@theory.snu.ac.kr, cho@ssrnet.snu.ac.kr

The Improved Processor Bound for Parallel Exponentiation in $GF(2^n)$

Yoonjeong Kim Kunsoo Park Yookun Cho
School of Computer Science and Engineering, Seoul National University

요 약

본 논문에서는 정규 기저 표현(normal bases representation)을 갖는 $GF(2^n)$ 상에서의 병렬 멱승 연산에 있어서 2 가지의 개선 사항을 기술한다. 첫째는, k를 윈도우 길이로 할 때 라운드가 $\lceil \log k \rceil + \lceil \log \lceil n/k \rceil \rceil$로 고정된 경우에 현재까지 알려진 방법보다 더 작은 수의 프로세서를 갖는 방안이다. 둘째는 점근적인(asymptotic) 분석을 통하여 $GF(2^n)$ 상에서의 병렬 멱승 연산이 $O(n/\log^2 n)$ 개의 프로세서로 $O(\log n)$ 라운드에 수행될 수 있음을 보인다. 이것은 프로세서×라운드의 바운드를 $O(n/\log n)$으로 하는 것으로 이전까지 알려졌던 $O(n)$을 개선한 것이다.

1. 서론

갈로아 필드 $GF(2^n)$ 상에서의 멱승 연산은 암호 관련 응용에서 폭넓게 이용되고 있으며, 지수 n의 값이 작으면 이산 로그가 쉽게 구해지므로, 보통 안전한 시스템을 유지하기 위해 n 값을 크게 설정한다. 그런데, n의 값이 커짐에 따라 멱승 연산을 수행하는 시간도 따라서 증가하게 되고, 결과적으로 속도가 빠른 멱승 알고리즘의 개발이 대단히 중요한 문제로 대두되고 있다.

속도를 증진시키기 위한 일환으로 병렬 알고리즘이 제안되었는데, 이들은 필드 요소가 정규 기저 표현인 것 [1, 2, 3, 4, 5]과 다항식 표현인 것들[6, 7]로 나눌 수 있다. 이 중 정규 기저 표현을 갖는 병렬 멱승 연산에 대한 연구는 고정된 라운드에 대하여 프로세서 수를 줄이는 것과, 프로세서 × 라운드의 바운드를 분석한 것 등이 있다.

본 논문에서는 정규 기저 표현을 갖는 $GF(2^n)$ 상에서의 멱승 연산에 있어서 우선, 라운드가 $\log n$으로 고정된 경우에 현재까지 알려진 것보다 더 작은 수의 프로세서를 갖는 방안을 제시한다. 또한 점근적인 분석을 통하여 프로세서 × 라운드의 복잡도가 $O(n/\log n)$이 될 수 있음을 보이는데 이것은 기존의 분석인 $O(n)$을 개선한 것이다.

2 장에서는 $GF(2^n)$상의 병렬 멱승 연산을 정의하며, 3 장에서는 라운드가 $\log n$으로 고정된 경우에 프로세서 수를 개선한 사항을, 4장에서는 프로세서 × 라운드의 바운드 향상 내용을 다룬다.

2. 정규기저표현을 갖는 $GF(2^n)$상에서의 병렬 멱승

$GF(2^n)$은 $GF(2)$ 상에서의 n 차원 벡터 공간으로, 이 공간에서의 $\{\beta, \beta^2, \beta^4, \cdots, \beta^{2^{n-1}}\}$ 형태의 기저는 정규 기저라 불린다. $GF(2^n)$은 $n \geq 1$인 모든 n에 대하여 정규 기저를 갖고 있다고 알려져 있다 [2, 3]. 필드 요소가 정규 기저에 대한 계수로 표시될 때 이것을 '정규 기저 표현'이라 하는데, 정규 기저 표현의 경우에, 필드 요소 a를 제곱하는 연산은 a에 대한 계수들의 환형 쉬프트만으로 이루어진다는 특성을 갖는다[1, 2, 3, 4, 8]. 예를 들어, $x^3 + x^2 + 1$을 근저 다항식으로 갖는 $GF(2^3)$의 경우 α를 다항식의 근이라 할 때, $(\alpha, \alpha^2, 1+\alpha+\alpha^2)$은 정규 기저이다. 왜냐하면, $1+\alpha+\alpha^2$이 α^4으로 표현될 수 있기 때문이다. 이 경우, 값이 α인 필드 요소 a의 정규 기저 표현은 (1,0,0)이며, a^2은 (0,1,0), a^4은 (0,0,1), a^8은 (1,0,0)으로 나타내진다. 이처럼, $GF(2^n)$의 정규 기저 표현에서는 제곱이 환형 쉬프트 만으로 구성될 수 있으므로, 제곱에 필요한 시간을 무시한다고 가정할 수 있다. 이러한 가정은 여러 관련 논문들[1, 2, 3, 4]에서 사용되는 것으로, 본 논문에서도 제곱의 시간은 무시하기로 한다.

$GF(2^n)$ 상에서의 멱승 연산이란 주어진 $a \in GF(2^n)$에 대하여 $a^e \in GF(2^n)$을 계산하는 것으로 이 때, $e = \sum_{i=0}^{n-1} e_i 2^i$, $e_i = 0$ 또는 $1, 0 \leq i \leq n-1$ 이다. 기본적인 멱승 연산 기법은 '이진 방법'으로 이 기법에서는 $0 \leq i \leq n-1$ 인 $a^{e_i 2^i}$들을 모두 곱한다. 즉, $\Pi_{i=0}^{n-1} a^{e_i 2^i}$를 계산한다. 정규 기저 표현에서는 a^{2^i}의 비용을 무시할 수 있으므로, 이진 방법에서의 곱셈의 개수는 값이 1인 e_i $(0 \leq i \leq n-1)$의 개수가 된다. 결과적으로 이진 방법은 평균적으로 $n/2-1$ 개의 곱셈을 필요로 한다.

이진 방법보다 더 작은 수의 곱셈을 필요로 하는 '윈도우 방법'은 지수에 특정 패턴이 반복적으로 나타나는 특성을 이용한다. 윈도우 방법에서는 지수의 비트들을 k 개씩 나누어, 지수 e를 $e = \sum_{i=0}^{s-1} w_i 2^{ki}$ 형태로 표시한다. 여기서, $s = \lceil n/k \rceil$이며 $0 \leq w_i \leq 2^k - 1$, $0 \leq i \leq s-1$ 이다. 이 때, k는 윈도우 길이라 불린다. 이런 표현 하에서, 멱승 연산은 2 가지 단계로 나뉘어 처리된다. 첫째는 윈도우 값 계산 단계이며, 둘째는 윈도우 들간의 곱셈을 수행하는 것이다. 아래에 주어진 알고리즘 *window*()가 윈도우 방법을 설명해 준다. 알고리즘에 이어서 이의 예가 하나 주어진다.

알고리즘 **window(a,e,k)**.
단계 1: $2^k - 1$ 개의 $a^w (1 \le w \le 2^k - 1)$를 구한다. .
단계 2: $(a^w)^{2^{ki}} (0 \le i \le s-1)$ 항들을 서로 곱한다.

예 1. $n = 10$, $k = 2$라 할 때, $e = 631 = 1001110111(binary)$라 하자. 그러면, $s = 5$가 된다. 단계 1에서는, 윈도우 값인 a^1, a^2, a^3을 계산하며 단계 2에서는, 윈도우 $(a^2)^{2^8}$, $(a^1)^{2^6}$, $(a^2)^{2^4}$, $(a^2)^{2^2}$, $(a^2)^{2^0}$ 들 간의 곱이 수행된다. a^{2^i}가 비용이 없는 제곱 연산에 의하여 수행되므로, 단계 2 는 단지 4 개의 곱셈만을 필요로 한다.

3. 라운드가 $\lceil \log k \rceil + \lceil \log s \rceil$로 고정된 경우의 개선된 프로세서 바운드

멱승 연산에 있어서 가장 효율적인 방법으로 알려진 윈도우 방법을 이용하여 병렬 멱승 연산을 수행하는 경우 최소 $\lceil \log k \rceil + \lceil \log s \rceil$ 라운드를 필요로 한다. 여기서, $s = \lceil n/k \rceil$ 이다. $\lceil \log k \rceil$는 단계 1에서 윈도우 값을 계산하는데 필요한 최소 라운드이고 $\lceil \log s \rceil$는 단계 2에서 s 개의 윈도우들을 곱하는데 필요한 최소 라운드이다.

라운드 수가 이 값으로 고정된 경우에 프로세서 수는 단계 1의 라운드들 중 최대 프로세서 갯수인 p_1과 단계 2의 라운드들 중 최대 프로세서 갯수인 p_2 중 큰 값인 $max(p_1, p_2)$로 정해진다.

p_1에 대하여 알려진 값 중 가장 좋은 것은 Gathen이 제안한 것이며[4], p_2는 Stinson과 Gathen이 제안한 것이다 [2, 4]. 본 절에서는, Gathen이 제안한 p_1은 오류가 있으며 p_2는 라운드 수를 $\lceil \log k \rceil + \lceil \log s \rceil$로 고정하면서도 Gathen과 Stinson이 제안한 값보다 더 개선될 수 있음을 밝힌다.

Gathen이 제안한 단계 1 알고리즘은 $\lceil \log k \rceil$ 라운드동안 진행되는데, 각 라운드 i에서는 다음을 계산한다. 우선, 이진 표현이 $d = \Sigma d_i 2^i$인 d에 대하여 이진 *해밍 무게* $w(d)$를 d의 이진 표기 중 값이 1인 비트들의 갯수라 하자. 각 단계 i에서는 $2 < d < 2^k$, $d \neq 0 \bmod 2$, and $w(d) \le 2^i$ 조건을 만족하는 x^d 중 아직 계산되지 않은 값들을 구한다. 라운드 i에서 새로 계산되는 d는 $d = d_1 + 2^j d_2$ 형태이고, 이 때 $j \ge 1$이고, d_1과 d_2는 이전 라운드에서 이미 구해진 값이다. 이렇게 하면, 각 라운드 $1, \ldots, \lceil \log k \rceil$는 한 라운드안에 수행될 수 있다.

예를 들어 $k = 5$이면, $\lceil \log k \rceil = 3$ 라운드가 필요하다. 즉, 라운드 1에서는 d의 해밍 무게가 2인 ($d = 3, 5, 9, 17$ 총 4 항) x^d가 계산되고 (이 라운드에서 해밍 무게가 1인 것은 계산하지 않는데, 이것은 해밍 무게가 1인 d는 $d = 0 \bmod 2$이기 때문이다), 라운드 2에서는 해밍 무게가 3인 것($d = 7, 11, 13, 19, 21, 25$)과 4 인것 ($d = 15, 23, 27, 29$)이 (총 10 항), 라운드 3에서는 5 인것 ($d = 31$, 총 1 항)이 계산된다.

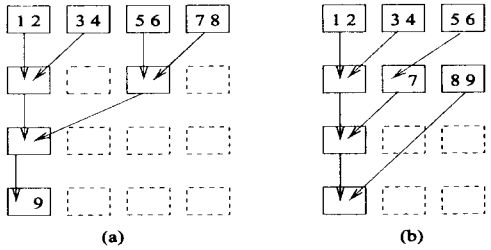

그림 1: 윈도우의 갯수가 9인 경우의 프로세서 수와 라운드 수: (a) 4 개의 프로세서로 4 라운드 동안 진행된다 (b) 3 개의 프로세서로 4 라운드 동안 진행된다

표 1: 라운드 수가 $\lceil \log k \rceil + \lceil \log s \rceil$로 고정된 경우의 프로세서 수: $max(p_1, p_2)$

	p_1	p_2
Stinson (theorem 3.1 in [2])	$(2^j - 1)(2^{k-j} - 1)$, where $j = \lceil k/2 \rceil$	$\lfloor s/2 \rfloor$
Gathen (theorem 3.1 in [4])	마지막 2 라운드의 평균	$\lfloor s/2 \rfloor$
본 논문	max(마지막 2 라운드의 평균, 이전 라운드의 프로세서수)	$\lfloor s/2 \rfloor$와 같은 라운드 수를 갖는 프로세서 수 중 최소값

Gathen은 이 알고리즘에 기반하여 마지막 이전 라운드에서 가장 많은 프로세서를 필요로 하고 마지막 라운드에서 하나의 프로세서를 필요로 하는 경우에 마지막 2 라운드를 균등화시켜 단계 1에서 필요한 최대 프로세서의 갯수를 줄이는 방안을 제안하였다. 그는 또, 마지막 2 라운드의 프로세서 수를 평균한 값이 그 이전 라운드들에서 필요로 하는 각 프로세서의 갯수보다 크다는 것을 보이고 따라서 단계 1에서 필요한 최대 프로세서의 갯수가 마지막 2 라운드의 평균임을 보였다 [4].

그러나, 그의 주장에 오류가 있음을 밝힌다. 반례를 들면 다음과 같다. 윈도우 길이가 9일 때, 라운드 1에서 필요한 프로세서의 수는 8이고, 라운드 2는 84, 라운드 3은 162, 라운드 4는 1이다. 마지막 2 라운드를 균등화하면, $\lfloor (162+1)/2 \rfloor = 82$개의 프로세서가 필요하다. 따라서, 필요한 전체 프로세서 수는 8, 84, 82 중의 최대값인 84로, Gathen의 주장처럼 82가 아니다. 결과적으로 [4]의 Table 3의 마지막 줄에 있는 값인 82는 84로 수정되어야 한다.

한편, 단계 2의 최대 프로세서 갯수인 p_2에 대하여 Gathen과 Stinson은 라운드가 $\lceil \log s \rceil$로 고정된 경우에 필요한 프로세서의 수를 $\lfloor s/2 \rfloor$로 밝히고 있다. 그런데, 라운드는 같으면서도 프로세서 수를 줄일 수 있는 방안이 있다. 예를 들어, 윈도우 갯수가 9인 경우에 윈도우들간의 곱은 4 라운드동안 진행된다. 이 때, 필요한 라운드 수는 Gathen과 Stinson의 theorem 대로 라면, 4가 되어야 한다. 그런데, 그림 1에 보이듯이, 3 개의 프로세서만으로도 같은 라운드 동안 진행될 수 있다.

또 다른 예로 윈도우 갯수가 228인 경우 ($n = 2048$이고 $k = 9$인 경우) Stinson과 Gathen의 방법을 따르면, $\lfloor 228/2 \rfloor = 114$ 프로세서로 8 라운드 동안 진행어야 한다. 그런데, 우리

표 2: 라운드 수가 $\lceil \log k \rceil + \lceil \log s \rceil$로 고정된 경우의 프로세서 수: 예

		p_1	p_2	$\max(p_1, p_2)$
예1 n=1013,k=9	Stinson	465	56	465
	Gathen	82*	56	82*
	본논문	84	56	84
예2 n=2048,k=9	Stinson	465	114	465
	Gathen	82*	114	114
	본논문	84	100	100

* 82는 잘못된 값이다

표 3: $GF(2^n)$에서의 병렬 멱승을 위한 프로세서 × 라운드의 바운드

		프로세서	라운드	프로세서 × 라운드
Stinson의 분석		$\frac{n}{\log n - \log\log n}$	$\log n$	n
Gathen의 분석		$\frac{n}{2\log n}$	$\log n$	$\frac{n}{2}$
본 논문의 분석	Stinson의 방법	$\frac{n}{\log^2 n}$	$2\log n$	$\frac{2n}{\log n}$
	Gathen의 방법	$\frac{1}{2}\cdot\frac{n}{\log^2 n}$	$3\log n$	$\frac{3}{2}\cdot\frac{n}{\log n}$
	단계1에도 프로세서 수 줄이는 기법 적용 수정	$\frac{1}{4}\cdot\frac{n}{\log^2 n}$	$5\log n$	$\frac{5}{4}\cdot\frac{n}{\log n}$
		$\frac{1}{8}\cdot\frac{n}{\log^2 n}$	$9\log n$	$\frac{9}{8}\cdot\frac{n}{\log n}$

의 방안대로 하면, 100 개의 프로세서만으로도 같은 라운드에 진행될 수 있다. 결과적으로 [4]의 표 4의 아래에서 2 번 째 줄에 있는 값인 114는 100으로 개선될 수 있다.

결론적으로, p_1은 마지막 2 라운드의 평균값[4]이 아닌 마지막 2 라운드의 평균값과 그 이전 라운드들의 프로세서 수 중 최대 값으로 수정되어야 하고, p_2는 $\lfloor s/2 \rfloor$가 아닌 [2, 4], "s 개의 요소를 곱할 때 $\lfloor s/2 \rfloor$ 개의 프로세서로 필요한 라운드 수와 같은 라운드를 갖는 가장 작은 프로세서 수"로 개선될 수 있다. 즉, [2]의 theorem 3.1과 [4]의 theorem 3.1을 각각 개선할 수 있다. 표 1은 이들 결과를 보여 주며 표 2는 이의 예를 제시한다.

4. 프로세서 × 라운드의 바운드 분석

본 절에서는 $GF(2^n)$ 상에서의 병렬 멱승 연산에 있어서의 프로세서 × 라운드의 점근적 분석 내용을 기술한다(표 3). 이에 대한 기존의 분석은 Stinson과 Gathen의 것으로, Stinson의 결과는 $\frac{n}{\log n - \log\log n}$ 프로세서와 $\log n$ 라운드로 프로세서 × 라운드는 n이 된다 (이것은 [2]의 theorem 3.1. 다음 문단에 기술되어 있다). Gathen의 결과는 $\frac{n}{2\log n}$ 프로세서와 $\log n$ 라운드로 프로세서 × 라운드의 바운드는 $\frac{n}{2}$이다 (이것은 [4]의 Section 2 마지막 2 문장에 나타나 있다). 이들은 라운드를 $\lceil \log k \rceil + \lceil \log s \rceil = \lceil \log k \rceil + \lceil \log \lceil n/k \rceil \rceil \doteq \log n$으로 고정한 경우의 분석이었다.

한편, s 개의 요소를 곱하는 단계 2의 방법에 대하여, 라운드 수는 늘리면서도 프로세서 수를 줄이는 방안이 제안되었다 [2, 4]. 이것은 v 개의 프로세서가 주어진 경우, 처음 각 $d = \lfloor s/v \rfloor - 1$ 단계에서는 v 쌍의 요소를 곱하고 이 후의 단계에서는 남은 $s - dv < 2v$ 요소들을 이진 트리 형태로 곱하는 것이다. $s - dv$ 요소들을 이진 트리 형태로 곱하는데 필요한 라운드 수는 $\lceil \log(s - dv) \rceil$이므로, 단계 2의 전체 라운드는 $d + \lceil \log(s - dv) \rceil$이 된다.

본 논문에서는 Stinson과 Gathen의 방법에, 단계 2에 있어서 프로세서 수를 단계 1에서 필요한 프로세서 수로 설정하고 라운드 수를 늘리는 경우에 대한 점근적 분석을 수행하였다. 이 때 $k = \log \frac{n}{\log^2 n}$로 설정한다. 분석 결과, Stinson의 방법은 $\frac{n}{\log^2 n}$ 개의 프로세서로 $2\log n$ 라운드에 Gathen의 방법은 $2^{k-1} = \frac{1}{2}\cdot\frac{n}{\log^2 n}$ 프로세서로 $\log n$ 라운드에 수행됨을 알 수 있었다. 또한 단계 1에도 프로세서 수를 줄이면서 라운드 수를 늘리는 기법을 적용하면, 프로세서 수가 $2^{k-2} = \frac{1}{4}\cdot\frac{n}{\log^2 n}$인 경우 라운드 수는 $5\log n$, 프로세서 수가 $2^{k-3} = \frac{1}{8}\cdot\frac{n}{\log^2 n}$인 경우 라운드 수는 $9\log n$임을 알 수 있었다. 우리의 분석은 프로세서 × 라운드를 $O(n/\log n)$으로 한다.

5. 결론

본 논문에서는 $GF(2^n)$ 상에서 필드 요소가 정규 기저 표현으로 표현될 경우에 병렬 멱승 연산에 대한 분석 내용을 제시하였다. 제시한 내용은 첫째, 라운드 수가 대략 $\log n$으로 고정된 경우에 프로세서 수에 대한 개선 사항과, 둘째, 프로세서 × 라운드의 바운드가 $O(n/\log n)$이 될 수 있음이다.

참고문헌

[1] G.B. Agnew, R.C. Mullin and S.A. Vanstone, "Fast exponentiation in $GF(2^n)$," *Advances in Cryptology - EUROCRYPT'88*, Lecture Notes in Computer Science, vol. 330, pp. 251–255, 1988.

[2] D.R. Stinson, "Some observations on parallel algorithms for fast exponentiation in $GF(2^n)$," *SIAM Journal on Computing*, vol. 19, no. 4, pp. 711–717, August, 1990.

[3] Joachim von zur Gathen, "Efficient and optimal exponentiation in finite fields," *Computational Complexity*, vol. 1, pp. 360–394, 1991.

[4] Joachim von zur Gathen, "Processor-efficient exponentiation in finite fileds," *Information Processing Letters*, vol. 41, pp. 81–86, 1992.

[5] Daniel M. Gordon, "A survey of fast exponentiation methods," *Journal of Algorithms*, vol. 27, pp. 129–146, 1998.

[6] C.K. Koc, T. Acar, "Montgomery multiplication in $GF(2^k)$," *Designs, Codes and Cryptography*, vol. 14, no. 1, pp. 57–69, April, 1998.

[7] A. Halbutogullari and C.K. Koc, "Parallel multiplication in $GF(2^k)$ using polynomial residue arithmetic," *Designs, Codes and Cryptography*, to be appeared, pdf file can be obtained from www.security.ece.orst.edu/publications.html.

[8] Rudolf Lidl, *Introduction to finite fields and their applications*, Cambridge University Press, London, 1994.