GF(2^n) 상에서 병렬 멱승 연산의 라운드 수 향상 기법

김윤정^O
서울여자대학교 정보통신공학부
yjkim@swu.ac.kr

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

Yoonjeong Kim^O
Division of Information & Communication Engineering
Seoul Women's University

요 약

본 논문에서는 정규 기저 표현(normal bases representation)을 갖는 GF(2^n) 상에서의 병렬 멱승 연산에 있어서, 프로세서 수가 고정된 경우에 라운드 수를 개선하는 방안에 대하여 기술한다.

1. 서 론

갈로아 필드 GF(2^n) 상에서의 멱승 연산은 암호 관련 응용에서 폭넓게 이용되고 있으며, 지수 n의 값이 작으면 이산로그가 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 정규기저표현인 것[1,2,3,4,5,6]과 다항식 표현인 것들[7,8]로 나눌 수 있다. 이 중 정규기저 표현을 갖는 병렬 멱승 연산에 대한 연구는 고정된 라운드에 대하여 프로세서 수를 줄이는 것과 프로세서×라운드 의 바운드를 분석한 것 등이 있다[1,2,4,5,6,7,8,9].

본 논문에서는 정규 기저 표현을 갖는 GF(2^n) 상에서의 멱승 연산에 있어서, 프로세서 수가 고정된 경우에 라운드 수를 줄일 수 있는 방안에 대하여 기술한다. 기존에 제안된 방법은 Stinson에 의하여 제안된 것으로, $\lceil \log_2 k \rceil + \lfloor s/2^k \rfloor + k - 1$이다. 본 논문에서 제안하는 방안은 $k + \lfloor (s-2)/2^k \rfloor$으로 값을 줄인 것이다.

2 장에서는, 정규기저 표현을 갖는 GF(2^n) 상에서 병렬 멱승에 대한 일반적인 방안을 기술한다. 그리고, 3장에서 프로세서 수가 고정된 경우의 라운드 수에 대한 기존 연구 내용을 기술하며, 4장에서 본 논문에서 새로이 제안하는 라운드 수를 개선하는 방안에 대하여 소개한다.

2. 정규기저표현을 갖는 GF(2^n) 상에서의 병렬멱승

GF(2^n)은 GF(2) 상에서의 n차원 벡터 공간으로, 이 공간에서의 $\{\beta, \beta^2, \beta^4, ..., \beta^{2^{n-1}}\}$ 형태의 기저는 정규기저라 불린다. GF(2^n)은 $n \geq 1$인 모든 n에 대하여 정규 기저를 갖는다고 알려져 있다. 어느 한 필드 요소 a∈ GF(2^n)가 정규기저에 대한 계수로 표시될 때 이것을 " 정규기저표현" 이라 하는데, 정규기저표현으로 나타내어진 필드요소 a를 제곱하는 연산은 계수에 대한 환형 쉬프트만으로 이루어진다는 특성을 갖는다 [10].

예를 들어, x^3+x^2+1을 근저 다항식으로 갖는 GF(2^3)의 경우 α를 이 다항식의 근이라 할 때, ($\alpha, \alpha^2, \alpha^4$)은 정규기저이다. 또한, $\alpha^4=1+\alpha+\alpha^2$으로 표시될 수 있으므로, ($\alpha$, α^2, $1+\alpha+\alpha^2$)도 정규기저이다. 이 때, 값이 a=α인 필드 요소 a의 정규기저표현은 (1,0,0)이며, a^2은 (0,1,0), a^4은 (0,0,1), a^8은 (1,0,0)의 정규기저표현으로 나타내진다.

위와 같이, GF(2^n)의 정규기저표현에서 제곱은 환형쉬프트만으로 구성될 수 있으므로, 보통 제곱에 필요한 시간은 무시한다 [1,2,3,4,5,6]. 본 논문에서도 제곱에 대한 시간은 무시하고 계산을 진행하기로 한다.

GF(2^n) 상에서의 멱승 (exponentiation) 연산이란 주어진 r에 대하여 $a^e \in GF(2^n)$을 계산하는 것으로 이 때,

본 연구는 2002년 서울여자대학교 교내학술연구비 지원 사업의 지원을 받았음.

$\sum_{i=0}^{n-1} e_i 2^i$. e_i=0 또는 1, $0 \le i \le n-1$ 이다. 기본적인 멱승 연산 기법은 ' 이진 방법' 으로 이 기법에서는 $0 \le i \le n-1$인 $a^{e_i 2^i}$ 들을 모두 곱한다. 즉, $\prod_{i=0}^{n-1} a^{e_i 2^i}$ 를 계산한다. 정규기저표현에서는 a^{2^i} 의 비용을 무시할 수 있으므로, 이진방법에서의 곱셈의 개수는 값이 1인 $e_i (0 \le i \le n-1)$ 의 개수가 된다. 결과적으로 이진 방법은 평균적으로 $n/2-1$개의 곱셈을 필요로 한다.

이진 방법보다 더 작은 수의 곱셈을 필요로 하는 ' 윈도우 방법 (window method)' 는 지수에 특정 패턴이 반복적으로 나타나는 특성을 이용한다. 윈도우 방법에서는 지수의 비트들을 k 개씩 나누어, 지수 e 를 $e = \sum_{i=0}^{s-1} w_i 2^{ki}$ 형태로 표시한다. 여기서 $s = \lceil n/k \rceil$이며 $0 \le w_i \le 2^k - 1$, $0 \le i \le s-1$이다. 이 때, k 는 윈도우 길이라 불린다. 이런 표현하에서, 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.

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.**

위 알고리즘의 예는 다음과 같다. n=10, k=2라 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 $(a^2)^{2^8}$, $(a^1)^{2^6}$, $(a^3)^{2^4}$, $(a^1)^{2^2}$, $(a^3)^{2^0}$ 들 간의 곱이 수행된다. a^{2^i} 가 비용이 없는 제곱 연산에 의하여 수행되므로, 단계 2는 단지 4 개의 곱셈만을 필요로 한다.

n 개 요소의 곱셈을 병렬로 진행하는 가장 기본적인 방법은, 그림 1과 같이, $n/2$ 개의 프로세서로 각 2 요소를 곱하고, 이들 결과의 각 2 요소를 다시 $n/4$ 개의 프로세서로 곱하는 것이다. 이런 식으로 최종 결과가 나올 때까지 진행한다. 즉, n 개 요소의 곱을 진행하는데, $n/2$ 개의 프로세서로 $\log_2 n$ 라운드가 필요하다. (정확히 표현하면, n 이 짝수인 경우는 $n/2$ 개의 프로세서로 $\lceil \log_2 n \rceil$ 라운드가 필요하며, n이 홀수인 경우는 $\lfloor n/2 \rfloor$ 개의 프로세서로 $\lceil \log_2 n \rceil$ 라운드가 필요하다)

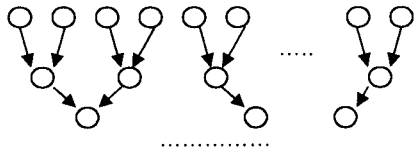

그림 1. n 개 요소의 곱의 기본 병렬 연산: $n/2$ 개의 프로세서로 $\log_2 n$ 라운드가 필요함

그림 1의 방법을 이용하여, 알고리즘 window를 병렬로 진행하는 방법은 다음과 같다[4]. 우선, 단계 2에서 필요한 총 곱셈의 개수는 $s = \lceil n/k \rceil$이며, 이를 $\lfloor s/2 \rfloor$ 개의 프로세서로 그림 1의 방법으로 진행하면 $\lceil \log_2 S \rceil$라운드가 필요하다. 다음으로, 단계 1은 알고리즘 d-c-small-powers 에 의하여 수행될 수 있으며, $P(k) = (2^{k/2} - 1)^2$ (k 가 짝수인 경우) 또는 $P(k) = (2^{(k+1)/2} - 1)(2^{(k-1)/2} - 1)$ (k 가 홀수인 경우) 개의 프로세서로 $\lceil \log_2 k \rceil$ 라운드 동안 진행된다.

알고리즘 d-c-small-powers (a,k)

부단계 1: $2^j - 1$ **개의** $a^w (1 \le w \le 2^j - 1)$**를 구한다. 이 때,** $j = \lceil k/2 \rceil$**이다.**

부단계 2: 부단계 1에서 계산한 $a^u (1 \le u \le 2^{k-j} - 1)$ **에 대하여,** $(a^u)^{2^j}$**를 계산한다.**

부단계 3: 부단계 1에서 계산한 a^w **각 항과 부단계 2에서 계산한** $(a^u)^{2^j}$ **각 항을 곱한다.**

d-c-small-powers (a,5) 가 호출된 경우를 예로 들면 다음과 같다. $j = 3$ 이 되고, 부단계 1에서 $a^1, a^2, a^3, a^4, a^5, a^6, a^7$ 을 계산한다. 부단계 2에서는 a^8, a^{16}, a^{24} 를 계산한다. 부단계 3에서는 부단계 1과 부단계 2에서 계산된 요소들을 각각 곱하여 21 개의 값을 얻는다.

위에서 기술한 바와 같이, Stinson은 알고리즘 window를 병렬로 진행하는 경우, 프로세서 수가 $\max\{P(k), \lfloor s/2 \rfloor\}$ 일 때 최대 $\lceil \log_2 k \rceil + \lceil \log_2 s \rceil$ 라운드가 필요함을 밝히고 있다 [4].

3. 프로세서 수가 2^k 로 제한된 경우의 라운드 수 (기존연구내용)

한편, 프로세서 수가 2^k (이 때, $2^k \le \lfloor s/2 \rfloor$)로 제한된 경우의 최대 라운드 수에 대한 연구로는, 최대 $\lceil \log_2 k \rceil + \lfloor s/2^k \rfloor + k - 1$ 라운드라는 결과를 Stinson이 발표한 바 있다 [4]. 이것은, window의 단계 1은 d-c-small-powers 를 통하여 진행하고, 단계 2는 다음과 같이 진행하여 얻어진다. p_0를 2의 제곱 값들 중 p를 넘지 않는 최대값이라 할 때, 단계 2의 마지막 $\log_2 2p_0 = \log_2 p_0 + 1$ 라운드는 그림 1의 기본 병렬 연산을 통하여 $2p_0$ 개의 요소를 곱한다. 이 $\log_2 p_0 + 1$ 라운드에 수행되는 곱셈의 개수는 $2p_0 - 1$이다. 따라서, 단계 2의 앞부분 라운드에서는 $s - 1 - (2p_0 - 1) = s - 2p_0$ 개의 곱셈을 필요로 한다. $s - 2p_0$ 개의 곱셈은 p 개의 프로세서로 $\lceil (s - 2p_0)/p \rceil$ 라운드에 수행될 수 있다.

4. 프로세서 수가 2^k 로 제한된 경우의 라운드 수 (제안방법)

표 1. 제안 방법의 라운드 수 비교

k	s	2^k	Stinson방법 $\lceil \log_2 k \rceil + \lfloor s/2^k \rfloor$ $k-1$	제안방법 $k+\left\lceil \frac{s-2}{2^k} \right\rceil$
2	297	4	77(=1+75+2-1)	76(=2+74)
3	198	8	29(=2=25+3-1)	28(=3+25)
4	149	16	15(=2+10+4-1)	14(=4+10)
5	119	32	11(=3+4+5-1)	9(=5+4)

새로이 제안하는 방법의 기본 개념은, 알고리즘 window처럼 단계 1과 단계 2를 양분하지 않고 단계 1 수행 중 휴지 상태에 놓인 프로세스들로 하여금 단계 2의 곱셈을 일부 미리 계산하자는 것이다. 이 방안은 RSA와 같이 지수가 미리 고정되어 설정되는 경우에 유용하게 사용될 수 있다.

세부 내용은 알고리즘 parallel_exp에 기술되어 있다. 여기서, 단계0와 단계 j: $(1 \le j \le k-1)$는 window 알고리즘의 단계 1에 대응하고, 단계 j: $(k \le j \le k+\left\lceil \frac{s-2}{2^k} \right\rceil -1)$는 단계 2에 대응한다.

알고리즘 parallel_exp (a,x,k)

단계 0: for (i=0;i>k;i++) a^{2^i} 구함

단계 j: $(1 \le j \le k-1)$

for (i=j; i<k; i++)

for($h \in$ 단계(j-1)에서 구한 윈도우의 지수들 && h<i)

a^{2^i+h} 구함

min(2^k -pp(j), a^x 값이 계산된 윈도우 중 아직 곱해지지 않은 것) 개의 프로세스로 하여금 윈도우 간 곱셈을 수행하도록 함. [pp(j): j 단계에서 a^x 값이 계산되는 윈도우의 개수]

단계 j: $(k \le j \le k+\left\lceil \frac{s-2}{2^k} \right\rceil -1)$

나머지 윈도우들을 그림 1의 방법으로 구함

제안하는 방법의 전체 라운드 수는 $k+\left\lceil \frac{s-2}{2^k} \right\rceil$이다. 이것은 처음 $k-1$ 라운드 동안 $(k-1)2^k$ 개의 프로세서를 이용할 수 있는데, 2^k-1개의 프로세서로 윈도우 값을 계산하고 $(k-2)2^k+1$ 개의 프로세서로 윈도우 간의 곱셈을 수행하도록 하고, 이후의 라운드에서 $(s-1)-((k-2)2^k+1)$개의 곱셈을 수행하도록 하는데, 이 때 필요한 라운드 수는 $\left\lceil \frac{(s-1)-((k-2)2^k+1)-2^{k+1}}{2^k} + k+1 \right\rceil$이기 때문이다.

표 1에 $n=593$인 경우의 Stinson이 제안한 기존 라운드 수와 본 논문에서 제안하는 라운드 수가 비교되어 나타나 있다.

5. 결론

본 논문에서는 $GF(2^n)$상의 병렬 멱승 연산에서 프로세서 수가 2^k로 제한된 경우의 라운드 수 향상을 위한 새로운 방안을 제안하였다.

참고문헌

[1] 김윤정, 박근수, 조유근, GF(2^n) 상에서 병렬 멱승 연산의 프로세서 바운드 향상 기법, 정보과학회 춘계학술발표논문집, 제 27권 제 1호, 2000년 4월

[2] M.Lee, Y.Kim, K.Park, Y.Cho, Efficient Parallel Exponentiation in GF(q^n) using Normal Basis Representations, Journal of Algorithms, Academic Press, Inc., Accepted for Publication.

[3] G.B. Agnew, R.C. Mullin, 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.

[4] 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.

[5] Joachim von zur Gathen, " Processor-efficient exponentiation in finite fields," Computational Complexity, vol. 1, pp. 360-394, 1991.

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

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

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

[9] A. Haalbutogullari, C.K.Koc, " Parallel multiplication in GF(2^K) using polynomial residue arithmetic," Designs, Codes and Cryptography, vol. 20, no. 2, pp. 155-173, June 2000.

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