암호 프로세서용 고속 64 X 64 곱셈기

서정욱, 이상흥

한국전자통신연구원 회로소자기술연구소 암호프로세서팀
대전광역시 유성구 가정동 161, 우편번호: 305-350
TEL 042-860-5856, FAX 042-860-6108

A Fast 64X64-bit Multiplier for Crypto-Processor

Chung-Wook Suh and Sang-Heung Lee

Micro-Electronics Technology Laboratory, ETRI
161 Kajong-dong, Yusong-Gu, Taejon, 305-350
Tel 042-860-5856, Fax 042-860-6108

요 약

피승수를 승수로 곱하는 곱셈연산은 승수에 대한 많은 부분곱을 더하기 때문에 본질적으로 느린 연산이다. 특히, 큰 수를 사용하는 암호 프로세서에서는 매우 빠른 곱셈기가 요구된다. 현재까지 느린 연산의 개선책으로 radix 4, radix 8, 또는 radix 16 의 변형 부스 알고리즘을 사용하여 부분곱의 수를 줄이려는 연구와 더불어 Wallace tree 나 병렬 카운터를 사용하여 부분곱의 합을 빠르게 연산하는 방법이 연구되어 왔다. 본 논문에서는 암호 프로세서용 64X64 비트 곱셈기를 구현하는데 있어서, (1) 고속의 곱셈을 위하여 고속의 (7,3) 병렬 카운터를 제안하였으며, (2) radix 4 의 변형 부스 알고리즘을 이용하여 부분합을 만들고 부분합의 덧셈은 제안한 (7,3) 카운터를 사용하였다. 64X64 비트 곱셈기를 구현함에 있어서 본 논문에서 제안된 (7,3) 카운터를 이용하는 것이 속도면에서 Wallace scheme 또는 Dadda scheme 을 적용하여 구현하는 것 보다 31% 정도, Mehta 의 (7,3) 카운터를 적용하여 구현하는 것 보다 21% 정도 개선되었다.

1. 서론

곱셈 연산은 많은 부분곱들을 더하기 때문에 본질적으로 연산이 느리다. 예를들어, 기본적인 16 X 16 비트 곱셈에서는 16 개의 부분곱들이 더해진다. 많은 사람들이 부분곱의 수를 줄이려는 노력과 함께, 부분곱의 합을 빠르게 연산하는 방법을 연구해 왔다.

먼저, 부분곱의 수를 줄이기 위하여 변형 부스(Modified Booth) 인코딩(encoding)이 사용될 수 있다. 일반적으로 많이 쓰이는 Radix 4 변형 부스 인코딩 알고리즘은 16 X 16 비트 곱셈의 경우에 부분곱의 수를 16 개에서 8 개로 줄일 수 있으나 이것 역시 여전히 큰 수이며 전체 지연시간(delay time)을 실질적으로 좌우한다. 특히, 매우 큰 수를 사용하는 암호시스템에서는 지연시간을 줄이기 위하여 radix-4, radix-8, radix-16 등이 사용될 수 있으나 radix 8 이상은 인코딩시 상당한 하드웨어를 수반하여야 한다[1,2,3].

다음은, 부분곱의 합을 빠르게 하려는 시도에서, Wallace 는 리플 캐리가 없는 전가산기(full adder)들의 배열로서, 세 입력을 취하여 두개의 등가 출력을 만들어 내는 가산기(pseudo-adder)를 사용하였다. Wallace 는 부분곱의 합을 구하는데 여러 레벨에서 가산기(pseudo-adder)들을 사용하였다[4].

Dadda 는 부분곱 행렬을 줄이기 위하여 (n,m) 병렬 카운터들을 도입하였다[5,6]. 카운터는 active 인 1 의 개수를 세는 디바이스를 말한다. 이 카운터는 병렬 곱셈기, 컴퓨터 연산기, 다입력(multiple-input) 가산기 등을 실현하는데 아주 용이하다[7,8,9].

(n,m) 병렬 카운터는 n 개의 입력과 m 개의 출력을 갖는 조합 논리 회로이다. 여기서, m 은 입력에 들어오는 1 의 개수를 2 진수로 코드화한 값이다. 따라서, 반가산기는 (2,2) 카운터가 되며, 전가산기는 (3,2) 카운터가 된다. Dadda 는 적절한 병렬 카운터들을 사용하여 부분곱의 높이를 줄이려고 하였으며, 사용 가능한 병렬 카운터에 따라 각 레벨의 부분곱 행렬의 적절한 높이가 존재한다는 것을 이론적으로 정립하였다. 각 레벨에서, Dadda 는 부분곱 행렬의 높이를 줄이기 위하여 (3,2) 카운터와 (2,2) 카운터만을 사용하거나 (3,2) 이외의 카운터(예를들어, (7,3) 카운터, (15,4) 카운터 등)를 사용하였다.

Dadda 의 (n,m) 병렬 카운터에서, n 은 같은 weight 를 갖는 입력의 수이다. n 은 부

분곱 행렬의 같은 열(column)으로부터 나와야 한다. 그런데, Stenzel 은 Dadda 의 개념을 확장하여 다중 열(multiple columns)로 구성된 입력 비트를 갖는 병렬 카운터를 제안하였다[10].

Swartzlander 는 2 개의 k 입력 카운터와 $(1+\lfloor \log_2 k \rfloor)$ 단(stage) 캐리 리플 가산기를 사용하여 카운터를 설계하는 방법((2k+1)개 입력을 갖는 카운터)을 사용하였다[11]. 예를들어 (7,3)은 3 입력의 (3,2) 카운터 2 개와 2 단의 리플 캐리 가산기로 구현된다.

Mehta 는 기존의 Wallace, Dadda 의 scheme 들을 사용하는 것 보다 빠른 (7,3) 카운터를 고안하여 16 X 16 비트 곱셈기를 설계하여 연산시간을 단축하였다[12].

본 논문에서는 암호프로세서의 빠른 곱셈을 위하여, 매우 빠른 (7,3) 카운터를 제안한다. 그리고, 제안된 (7,3) 카운터를 사용하여 radix 4 의 64X64 비트 곱셈기를 구현한다.

2. (7,3) 카운터의 설계

그림 1 은 (3,2) 카운터 만을 사용하여 구현된 Swartzlander 의 (7,3) 카운터이다. 첫번째 레벨에서, 두 개의 (3,2) 카운터는 7 개의 입력 중 6 개를 취하여 두 개의 합과 두 개의 캐리 출력을 발생시킨다. 첫번째 레벨의 합 출력들은 두 번째 레벨에서 합 출력을 발생시키기 위하여 7 번째 입력과 결합된다. 이 (3,2) 카운터의 캐리 출력은 다시 다른 (3,2) 카운터를 사용하여 첫번째 레벨에서의 두 캐리 출력과 결합되어 weight 2 의 캐리 C_1 과 weight 4 의 캐리 C_2 를 만든다. 이 회로는 6 개의 exclusive-OR 게이트 지연시간을 가진다[11].

Mehta 가 제안한 (7,3) 카운터는 다음과 같다. 먼저, 7 개의 입력 (x0~x6)을 두 군 (group) (x0, x1, x2, x3), (x4, x5, x6)으로 나눈다. 먼저, 첫번째 군에서 입력 중 1 의 개수가 2 개, 3 개, 또는 4 개일 때 캐리신호가 있고, 두번째 군에서 입력 중 1 의 개수가 2 개 또는 3 개일 때 처리하는 캐리신호가 있다. 첫번째 군에서 입력 중 1 의 개수가 4 개일 때와 두 군 사이의 1 의 개수가 각각 1 과 1, 1 과 3, 3 과 1 일 때 처리하는 캐리신호로 구성된다. Weight 2 의 C_1 과 weight 4 의 C_2, 2 개의 캐리신호를 얻기 위하여 1 개의 전가산기를 사용하여 처리한다[12].

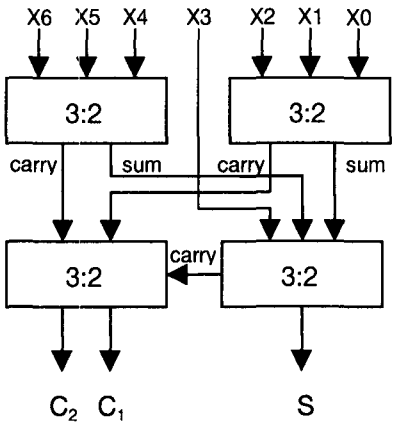

그림 1. (3,2) 카운터로 구현된 (7,3) 카운터

앞의 두 (7,3) 카운터는 현재 까지 잘 알려진 (7,3) 카운터들이다. 본 논문에서는 이들 카운터들 보다 더 빠른 (7,3) 카운터를 제안한다. 본 논문의 (7,3) 카운터의 설계 방법은 다음과 같다.

먼저, (7,3) 카운터의 진리표로부터, C_1과 C_2의 최소항(minterm)들을 얻는다.

$C_1=\Sigma(3,5,6,7,9,10,11,12,13,14,17,18,19,20,21,22,24,25,26,28,33,34,35,36,37,38,40,41,42,44,48,49,50,52,56,63,65,66,67,68,69,70,72,73,74,76,80,81,82,84,88,95,96,97,98,100,104,111,112,119,123,125,126,127)$ (1)

$C_2=\Sigma(15,23,27,29,30,31,39,43,45,46,47,51,53,54,55,57,58,59,60,61,62,63,71,75,77,78,79,83,85,86,87,89,90,91,92,93,94,95,99,101,102,103,105,106,107,108,109,110,111,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127)$ (2)

위에서 얻어진 최소항들은 Quine-McCluskey 방법에 의해 간소화되며, 간소화된 식으로부터 논리 회로를 얻는다. 그림 2는 이와 같은 과정을 거쳐서 얻은 (7,3) 카운터이다. 그림으로부터 C_1에 대한 회로가 C_2에 대한 회로 보다 더 복잡함을 알 수 있는데, 그 이유는 weight 2의 캐리(C_1)가 발생하게 되는 1의 입력 개수 분포가 분산되는 반면 weight 4의 캐리(C_2)가 발생하게 되는 1의 입력 개수 분포는 집중되는 현상이 발생되기 때문이다.

다음은 본 논문의 카운터와 기존의 카운터를 지연시간 관점에서 비교하였다. 먼저, 단순 게이트(인버터, 2-입력 NAND, 3-입력 NAND, 4-입력 NAND, 2-입력 NOR, 3-입력 NOR, 4-입력 NOR)가 1Δ 지연시간을 갖는다고 가정하면, exclusive-OR 게이트는 2Δ의 지연시간을 갖고, 복합 게이트인 AOI(AND-OR-INVERTER), OAI(OR-AND-INVERTER)는 1.5Δ의 지연시간을 갖는다. 따라서, 각 카운터의 총 게이트 수와 총 게이트 지연시간을 계산해 보면, (1) (3,2) 카운터 만을 사용한 Swartzlander (7,3) 카운터는 12Δ의 지연시간을 갖는다. (2) Mehta 의 (7,3) 카운터는 8Δ의 지연시간을 갖는다. (3) 본 논문의 카운터는 6Δ의 지연시간을 갖는다. 본 논문의 카운터가 (3,2) 만을 사용한 카운터 보다 지연시간면에서 50%가 개선되었으며, Mehta 의 카운터 보다는 25%가 개선되었다. 이를 요약하여 표 1 에 나타내었다.

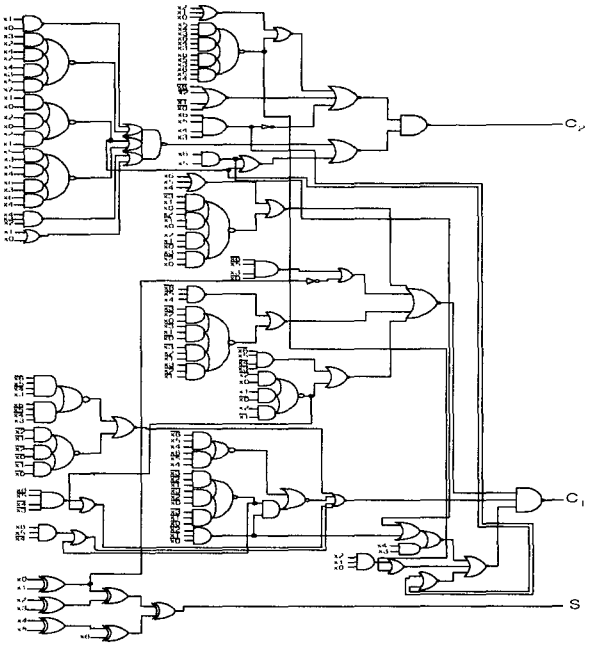

그림 2. 제안된 (7,3) 카운터

표 1. (7,3) 카운터의 complexity 비교

	Method using (3,2) counters	Mehta method	proposed method
delay	12Δ	8Δ	6Δ

(7,3) 카운터 이외에도 (6,3), (5,3), (4,3) 카운터가 같은 방법으로 설계되었으며, 지연 시간은 각각 6Δ, 6Δ, 4Δ의 지연시간을 갖는다.

3. 64 X 64 비트 곱셈기의 구현

곱셈의 연산속도를 개선하기 위하여 많은 사람들이 부분곱의 수를 줄이려는 노력과 함께, 부분곱의 합을 빠르게 연산하는 방법을 연구해 왔다

이 절에서는 연산 속도의 개선을 위하여 radix 4 의 체계를 사용한 변형 부스 알고리즘을 사용하여 부분곱의 수를 줄이고 아울러 제안된 (7,3) 카운터를 이용하여 부분곱의 합을 빠르게 함으로써 고속 64X64 비트 곱셈기를 구현하였다.

3.1 Radix 4 를 사용한 변형 부스 인코딩과 부호 확장 방법

곱셈시 곱셈기 operand 내에 zero 비트가 많을수록 더해야 되는 operand 수가 줄어드는 성질을 string property 라고 하는데, 부스 알고리즘의 기본 원리는 이와 같은 string property 를 이용한 것이다.

부스 알고리즘은 기본적으로 string properity 를 검사(check)하기 위하여 하위 비트에서부터 2 비트 단위로 1 비트씩 중첩시키면서 부스 인코딩을 해 주는 것이다. Radix 4 변형 부스 알고리즘은 3 비트 단위로 1 비트를 중첩시키면서 string property 를 검사하는 방법이다.

표 2. Radix 4 부스 인코딩

$Y_{i-1}\ Y_i\ Y_{I+1}$	$Y_{i-1}+Y_i-2Y_{i+1}$	partial product
0 0 0	0	0
0 0 1	1	X
0 1 0	1	X
0 1 1	2	2X
1 0 0	-2	-2X
1 0 1	-1	-X
1 0 0	-1	-X
1 0 1	0	0

본 논문에서는 64 X 64 비트의 곱셈에서 부분곱의 수를 줄여 속도 향상을 꾀하기 위하여 radix 4 의 변형 부스 알고리즘을 적용하였다. 표 2 는 radix 4 의 부스 인코딩 표이다. 예를들어 P = X x Y 의 곱셈에서 Y 의 값에따라 부분곱은 표에서 처럼 계산되며, 주의할 점은 Y 의 최하위 비트 아래에 0 을 두고 시작해야 한다. 표 2 에 의해 64 X 64 비트의 곱셈의 경우 radix 4 변형 부스 알고리즘을 적용하면 총 32 개의 부분곱 행(row)이 생성된다.

다음은 부호(sign) 확장 방법에 대하여 설명하겠다. 가장 간단한 부호 확장방법은 다음과 같다. 2 의 보수로 표현된 64 X 64 비트의 곱셈의 경우 계산 결과는 총 127 비트로 표시되고 부분곱은 64 비트로 표시된다. 따라서, 부분곱은 부분곱의 부호에 따라 127 비트까지 부호를 확장해주어야 한다. 이때 양수인 경우 0 을 63 비트를 연장시키고 1 을 63 비트 연장시켜 준다. 이 방법은 부분곱의 부호를 계산 결과에 필요한 만큼 확장해 주어야 하므로 매우 큰 하드웨어를 부담하여야 한다. 이와 같은 이유로 본 논문에서는 부호를 포함한 64 비트에 대하여 다음의 방법으로 부호를 확장하여 사용하였다. S_i 를 i 번째 부분곱 행의 부호 비트라 하자.

칫번째 부분곱 행의 부호 확장은 다음과 같다.

$$S_0(-2^{126} + \sum_{i=64}^{125} 2^i) = S_0(-2^{64})$$

$$S_1(-2^{126} + \sum_{i=66}^{125} 2^i) = S_1(-2^{66})$$

두번째 부분곱 행의 부호 확장은 다음과 같다.

세번째 부분곱 행의 부호 확장은 다음과 같다.

$$S_2(-2^{126} + \sum_{i=68}^{125} 2^i) = S_2(-2^{68})$$

중간과정은 동일하므로 생략한다. 31 번째 부분곱 행의 부호 확장은 다음과 같다.

$$S_{30}(-2^{126} + \sum_{i=124}^{125} 2^i) = S_{30}(-2^{124})$$

마지막으로 32 번째 부분곱 행의 부호 확장은 다음과 같다.

$$S_{31}(-2^{126})$$

따라서, 전체 부분곱(partial product: pp) 행의 부호 확장을 2 의 보수 형태로 표현하면 다음과 같다.

$$\sum_{i=0}^{31}(pp)^i = -\overline{S}_{31}2^{126} + 1\cdot 2^{125} + \overline{S}_{30}2^{124} + \bullet\bullet\bullet + \overline{S}_2 2^{68} + 1\cdot 2^{67} + \overline{S}_1 2^{66} + 1\cdot 2^{65} + (\overline{S}_0 + 1)\cdot 2^{64}$$

상기의 2 의 보수 형태로 표현된 부호 확장식의 의미는 다음과 같다. 첫번째 부분곱 행의 부호 확장은 부호 S_0의 보수를 취한 후 1 을 더한 후 1 을 1 비트 확장한다. 두번째에서 31 번째 부분곱 행들의 부호 확장은 부호의 보수를 취한 후 1 을 1 비트 확장한다. 마지막 32 번째 부분곱 행의 확장은 부호의 보수를 취한다.

3.2 Radix 4 체계와 (7,3) 카운터를 사용한 64 X 64 비트 곱셈기

본 논문에서는 고속용 64 X 64 비트 곱셈기를 설계하기 위하여 앞 절에서 언급한 radix 4 체계를 이용하여 부분합을 구하고 이를 (7,3) 카운터를 사용하여 구현하였다.

그림 3 은 64 X 64 비트 곱셈기로써 64 비트의 입력으로부터 radix 4 을 적용하여 총 32 개의 부분곱을 발생시키고 (7,3) 카운터를 이용하여 부분곱을 더하는 연산을 수행한다.

생성된 32 개 행의 부분곱들의 합을 구하기 위하여 첫번째 레벨에서는 (7,3) 카운터, (6,3) 카운터, (5,3) 카운터, (4,3) 카운터, 전가산기 및 반가산기가 사용되었다. 두번째 레벨, 세번째 레벨, 네번째 레벨에서도 같은 방법으로 필요한 카운터를 사용하였다.

이와 같이 구현된 곱셈기의 지연시간을 계산하면 다음과 같다. 표 3 으로부터 radix 4 를 이용하여 64X64 의 곱셈기를 구현하는 경우, 본 논문의 카운터를 이용하

여 부분곱을 더하는 연산을 수행하는 것이 Wallace scheme 또는 Dadda scheme 를 적용하여 구현하는 것 보다 31% 정도, Mehta 카운터를 적용하여 구현하는 것 보다는 21% 정도 개선되었음을 알 수 있다.

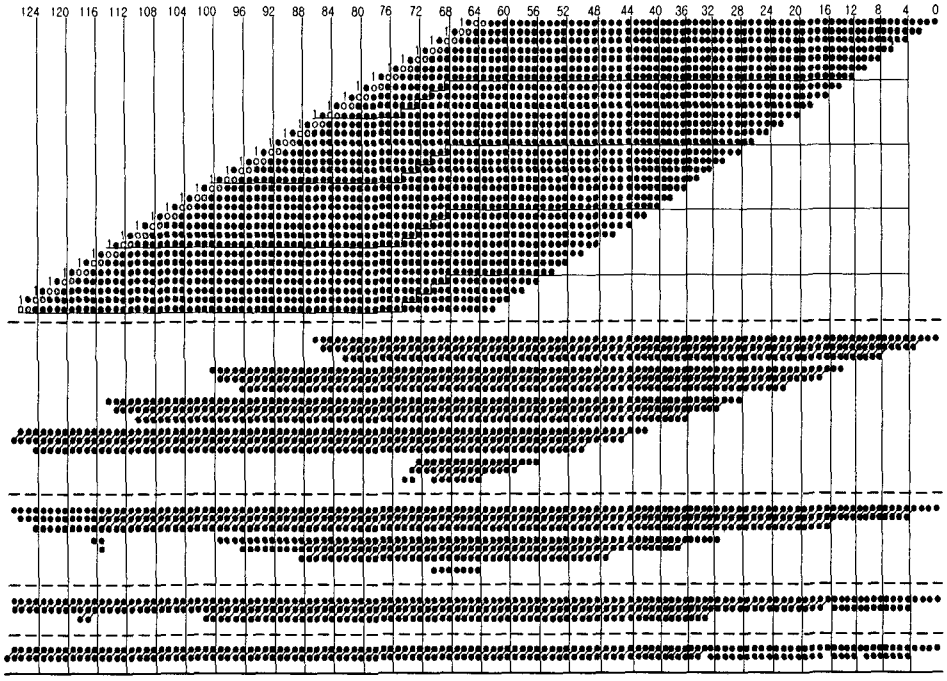

그림 3. 64 X 64 비트 곱셈기

표 2. 64 X 64 비트 곱셈기의 지연시간 비교

	Wallace or Dadda	Mehta method	proposed method
delay	32Δ	28Δ	22Δ

4. 결론

본 논문에서는 암호 프로세서용 64X64 비트 곱셈기를 구현하였다. (1) 고속의 곱셈을 위하여 기존의 카운터 대신에 고속의 (7,3) 병렬 카운터를 제안하였으며, (2)

radix 4 의 변형 부스 알고리즘을 이용하여 32 개의 부분합을 만들고 부분합의 덧셈은 제안한 (7,3) 카운터를 사용하여 구현하였다. 64X64 비트 곱셈기를 구현함에 있어서 본 논문에서 제안된 (7,3) 카운터를 이용하는 것이 속도면에서 Wallace scheme 또는 Dadda scheme 을 적용하여 구현하는 것 보다 31% 정도, Mehta 의 (7,3) 카운터를 적용하여 구현하는 것 보다 21% 정도 개선되었다.

참고문헌

[1] H. Sam and A. Gupta, "A Generalized Multibit Recoding of Two's Compliment Binary Numbers and Its Proof withApplication in Multiplier Implementations," IEEE Trans. on Computers, vol. C-39, pp. 1006-1015, Aug. 1990.

[2] P. E. Madrid, B. Millar, and E. E. Swartzlander, Jr., "Modified Booth Algorithm for High Radix Fixed-Point Multiplication," IEEE Trans. on Very Large Scale Integration (VLSI) systems, vol. 1, June 1993.

[3] C. N. Lyu and D. W. Matula, "Redundant Binary Booth Recoding," Proc. of the 12th Sym. on Comupter Arithmetic, 1995.

[4] C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Trans. on Electronic Computers, vol. EC-13, pp. 14-17, Feb. 1964.

[5] L. Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, vol. 34, pp. 349-356, May 1965.

[6] L. Dadda, "On Parallel Digital Multiplier," Alta Frequenza, vol. 45, pp. 574-580, 1976.

[7] A. Habibi and P. A. Wintz, "Fast Multipliers," IEEE Trans. on Computer, vol. C-19, pp. 153-157, Feb. 1970.

[8] R. H. S. Riordan and R. R. A. Morton, "The Use of Analog Techniques in Binary Arithmetic Units," IEEE Trans. on Electronic Computers, vol. EC-14, pp. 29-35, Feb. 1965.

[9] A. Svoboda, "Adder with Distributed Control," IEEE Trans. on vol. C-19, pp. 749-751, Aug. 1970.

[10] W. J. Stenzel, W. J. Kubitz, and G. H. Garcia, "A Compact High-Speed Parallel Multiplicaion Scheme," IEEE Trans. on Computers, vol. C-26, 948-975, Oct. 1977.

[11] E. E. Swartzlander, JR., "Parallel Counters," IEEE Trans. on Computers, vol. C-22, pp. 1021-1024, 1973.

[12] M. Mehta, V. Parmar, and E. Swartzlander, Jr., "High-Speed Multiplier Design Using Multi-Input Counter and Compressor Circuits," Proc. of the 10th Sym. on Computer.