$O(n)$-Depth Circuit Algorithm for Modular Exponentiation

Takafumi HAMANO†, Naofumi TAKAGI‡, Shuzo YAJIMA†, and Franco P. PREPARATA*

† Dept. of Information Science, Kyoto University, Kyoto 606-01, Japan
‡ Dept. of Information Engineering, Nagoya University, Nagoya 464-01, Japan
* Dept. of Computer Science, Brown University, Providence, RI 02912, USA

Abstract

An $O(n)$-depth polynomial-size combinational circuit algorithm is proposed for n-bit modular exponentiation, i.e., for the computation of "$x^y \bmod m$" for arbitrary integers x, y and m represented as n-bit binary integers, within bounds $2^{n-1} \leq m < 2^n$ and $0 \leq x, y < m$. The algorithm is a generalization of the square-and-multiply method. An obvious implementation of the square-and-multiply method yields a circuit of depth $O(n \log n)$ and size $O(n^3)$. In the proposed algorithm, the terms $x^{2^i} \bmod m$'s for all i's $\in \{0, \cdots, n-1\}$ are computed in $\left\lceil \frac{n-1}{\lceil \alpha \log n \rceil} \right\rceil$ parallel rounds, each of which computes $\lceil \alpha \log n \rceil$ consecutive terms, where $\frac{1}{\log n} \leq \alpha$. The circuit implementing a round has depth $O((1+\alpha)\log n)$ and size $O(n^{2(1+\alpha)})$ yielding a circuit for modular exponentiation of depth $O(\frac{1+\alpha}{\alpha} n)$ and size $O(\frac{n^{3+2\alpha}}{\alpha \log n})$.

1 Introduction

Modular exponentiation plays important roles in several public key cryptosystems. In the RSA cryptosystem[1], for example, encryption and decryption are performed by means of modular exponentiation. It is important to investigate the smallest circuit depth achievable for this operation.

We define n-bit modular exponentiation as the computation of "$x^y \bmod m$" where we assume that x, y and m are n-bit binary integers satisfying the bounds $2^{n-1} \leq m < 2^n$, $0 \leq x < m$ and $0 \leq y < m$. In this paper, we consider its implementation by means of a combinational circuit with the restriction of bounded-fan-in. It is obvious that there exists an $O(n)$ depth circuit for any n-variable Boolean function when exponential size is allowed. Our attention, however, is directed to polynomial-size circuits.

Most of the common algorithms for modular exponentiation are based on the "square-and-multiply" method, such as the binary method[2]. In the square-and-multiply method, $x^{2^i} \bmod m$'s are computed for i's $\in \{0, 1, \cdots, n-1\}$ by performing modular multiplication $O(n)$ times sequentially. We can achieve an $O(n^2)$ depth $O(n^3)$ size modular exponentiation circuit by simply cascading modular multipliers based on Brickell's[3] or Montgomery's[4] algorithm. We can reduce the depth to $O(n \log n)$ by first computing an approximation of the reciprocal of a modulus for residue calculation. Therefore, the question concerns the existence of polynomial-size circuits for this problem, whose depth is $o(n \log n)$.

In this paper, we propose an $O(n)$-depth polynomial-size circuit algorithm for n-bit modular exponentiation. The algorithm is also based on multiplication of powers of x of the form $\{x^{2^i} \bmod m : i = 0, 1, \cdots, n-1\}$. Rather than one at a time (as in the square-and-multiply method), these powers are generated in $\Theta(\frac{n}{\alpha \log n})$ rounds, for a chosen positive $\alpha \geq \frac{1}{\log n}$. The basic building block of the overall circuit is a module of depth $O((1+\alpha)\log n)$ and size $O(n^{2(1+\alpha)})$ which computes $\lceil \alpha \log n \rceil$ consecutive terms of the above set. Cascading $\Theta(\frac{n}{\alpha \log n})$ such modules, and suitably combining their outputs, yields an overall circuit for modular exponentiation of depth $O(\frac{1+\alpha}{\alpha} n)$ and size $O(\frac{n^{3+2\alpha}}{\alpha \log n})$.

This paper is organized as follows. In Section 2, we describe an $O(\log n)$-depth module implementing an algorithm for n-bit powering which is the main component of the proposed algorithm. In Section 3, we describe the overall circuit implementing an algorithm for n-bit modular exponentiation. In Section 4, we show a numerical example of modular exponentiation based on the proposed algorithm. Section 5 concludes the paper.

2 A Logarithmic-Depth Circuit for Powering

In the algorithm to be described, we use logarithmic-depth circuits for powering [5][6][7]. We define n-bit powering as the computation of x^l in the binary representation for an arbitrary n-bit integer x and a given, fixed, positive integer l. Beame, Cook, and Hoover showed that there exists a logarithmic-depth circuit for computing x^l using the residue number system if $l \leq n^{O(1)}$[5]. Okabe *et al.*[6], and Mehlhorn and Preparata[7] modified the algorithm by Beame *et al.* to reduce the circuit size using table look-ups. We explicitly use the circuits by Okabe *et al.*

Let $\{m_1, m_2, \cdots, m_h\}$, with $m_j < m_{j+1}$ ($j =$

1063-6889/95 $4.00 © 1995 IEEE

$1, \cdots, h-1$), be the set of the first h consecutive primes, referred to as moduli. A residue number system based on $\{m_1, m_2, \cdots, m_h\}$ represents an integer $x < M = \prod_{j=1}^{h} m_j$ as a sequence of residues $< x_1, \cdots, x_h >$, where $x_j = x \bmod m_j$. (In general, a residue number system only requires that the moduli be pairwise relatively prime.) As is well-known(see, e.g., [8]), by the Chinese Remainder Theorem, we have $x = \sum_{j=1}^{h} Q_j x_j \bmod M$, where $Q_j = \left(\frac{M}{m_j}\right)^{m_j-1} \bmod M$.

The algorithm for n-bit powering using the residue number system and table look-ups is as follows.

Algorithm [l-POWER]
(l is a given, fixed, positive integer. Let h be the smallest integer such that $2^{nl} \le \prod_{j=1}^{h} m_j$.)

Input: x ($0 \le x < 2^n$, an n-bit binary integer)
Output: x^l (an nl-bit binary integer)

Step 1. For $j = 1, 2, \cdots, h$,
compute $x_j = x \bmod m_j$ (in parallel).
Step 2. For $j = 1, 2, \cdots, h$,
compute $x_j^l \bmod m_j$
(in parallel by table look-up).
Step 3. Compute x^l from $\{x_j^l \bmod m_j : j = 1, \cdots, h\}$ using the Chinese Remainder Theorem.
□

n-bit powering, *i.e.*, the computation of x^l, can be accomplished on the basis of the algorithm [l-POWER] by an $O(\log(nl))$-depth $O(n^2 l^2)$-size P-uniform circuit[6]. P-uniformity holds when $l \le n^{O(1)}$. P-uniformity means that a circuit for an n-bit input can be generated in time polynomial in n by a deterministic Turing machine. Note that P-uniformity is weaker than log-space uniformity which is in common use.

3 Circuit for Modular Exponentiation

In this section, we describe a circuit implementing an algorithm for n-bit modular exponentiation $x^y \bmod m$. We first give a brief overview and then describe the details.

We first compute $s(i)$'s such that $s(i) \equiv x^{2^i} \pmod m$ and $0 \le s(i) < 2m$ for all i's$\in \{0, \cdots, n-1\}$, and then obtain $x^y \bmod m$ multiplying with modular reduction the terms $s(i)$'s for the i's such that the ith bit of y is 1.

Let $\alpha \ge \frac{1}{\log n}$ be a constant and let k denote $\lceil \alpha \log n \rceil$. Since α is a design parameter, it must appear in the evaluation of performance. The computation proceeds in $\lceil \frac{n-1}{k} \rceil$ rounds. In each round, for an n-bit input z, we compute $\{t(i) : i = 1, \cdots, k\}$ such that $t(i) \equiv z^{2^i} \pmod m$ and $0 \le t(i) < 2m$. Successively, z assumes the values $\{s(pk)$(in case $s(pk) < 2^n$) or $s(pk) - m$(otherwise): $p = 0, 1, \cdots, \lceil \frac{n-1}{k} \rceil\}$. In other words, one of the outputs of a round is supplied, after possible subtracting of m, as the input to the next round. In the pth round, $t(i) = s((p-1)k + i)$ hold for i's$\in \{1, \cdots, k\}$. The computation of $\{t(i) : i = 1, \cdots, k\}$ is carried out by modular powering. Specifically, we first carry out powering by means of the algorithm described in Section 2, and then perform modular reduction by m. The modular reduction is achieved multiplying z^{2^i} by a suitable approximation of the reciprocal of m. This approximate reciprocal is computed only once.

Finally, letting $y = \sum_{i=0}^{n-1} y_i 2^i$ ($y_i \in \{0,1\}$), we have $x^y \bmod m = \prod_{i=0}^{n-1} s(i)^{y_i} \bmod m$.

Specifically, the whole algorithm for n-bit modular exponentiation is as follows.

Algorithm [MODEXP]
Let $\alpha \ge \frac{1}{\log n}$ be a constant independent of n and let $k = \lceil \alpha \log n \rceil$.

Input: x, y and m
$$\left(\begin{array}{l} n\text{-bit binary integers} \\ 2^{n-1} \le m < 2^n \\ 0 \le x, y < m \\ y = \sum_{i=0}^{n-1} y_i 2^i \ (y_i \in \{0,1\}) \end{array}\right)$$
Output: $w = x^y \bmod m$ (an n-bit binary integer)

Step 1. $\widetilde{m^{-1}}$:= approximation of $\frac{1}{m}$ with $(2^k n)$-bit precision;
Step 2. $z := x$;
for $p := 1$ **to** $\lceil \frac{n-1}{k} \rceil$ **do**
begin
for each $i \in \{1, \cdots, k\}$ **parallel do**
compute $s((p-1)k + i) \equiv z^{2^i} \pmod m$;
{ *modular powering* }
$\{0 \le s((p-1)k + i) < 2m\}$
{$\widetilde{m^{-1}}$ is used in this operation.}
if $(s((p-1)k + k) \ge 2^n)$ **then**
$z := s((p-1)k + k) - m$;
else $z := s((p-1)k + k)$;
end
Step 3. $w := \prod_{i=0}^{n-1} s(i)^{y_i} \bmod m$;
□

Figure 1 illustrates the structure of a modular exponentiation circuit based on the algorithm [MODEXP]. We now analyze the performance of this algorithm.

In Step 1, we compute $\widetilde{m^{-1}}$ by retaining only the $(2^k n)$ most significant bits of $\frac{1}{m}$. Consequently, the following inequalities hold:

$$0 \le \frac{1}{m} - \widetilde{m^{-1}} < 2^{-2^k n} \tag{1}$$

The reciprocal of m can be computed using the Newton-Raphson method, which requires two multiplications and one subtraction at each iteration. Starting with a 1-bit initial approximation, since at each iteration the number of exact bits doubles, we have a total of $O((1+\alpha)\log n)$ iterations. Therefore, we can perform Step 1 with an $O((1+\alpha)^2 \log^2 n)$-depth $O(n^{2(1+\alpha)})$-size circuit (since $2^k n \simeq n^{1+\alpha}$).

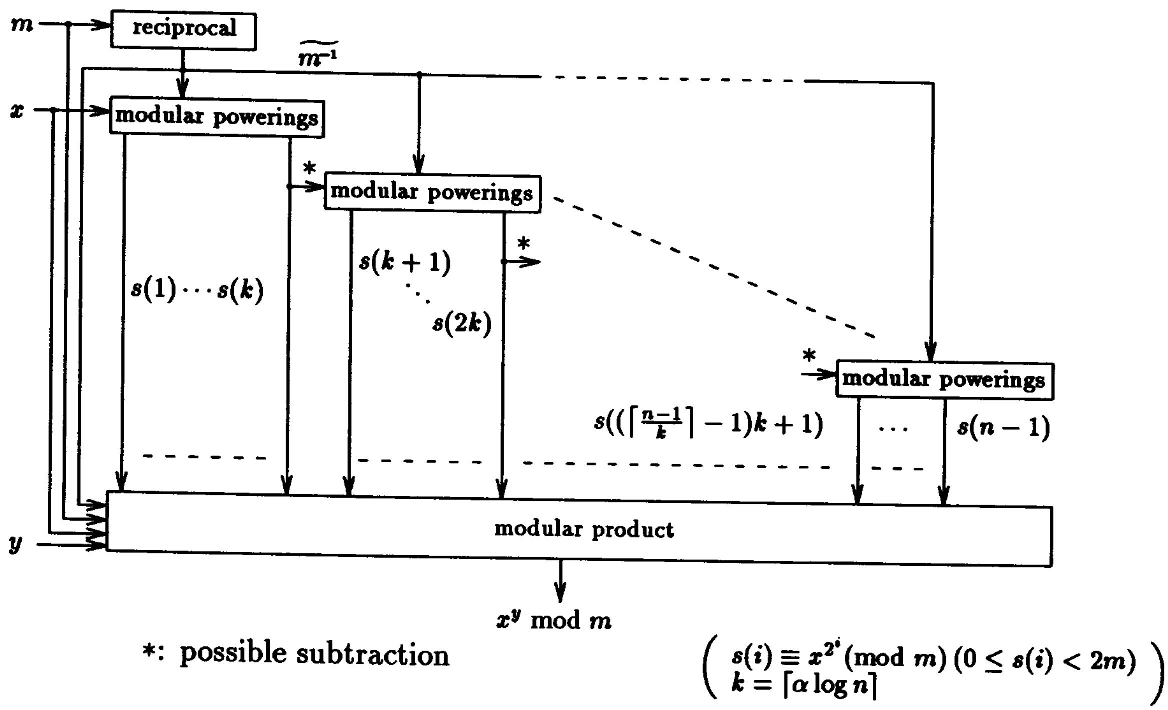

Figure 1: Configuration of a modular exponentiation circuit based on the algorithm [MODEXP]

In Step 2, we calculate the $s(i)$'s for $i = 1, 2, \cdots, n-1$ by means of modular powerings using $\widetilde{m^{-1}}$. Specifically, to compute z^l, we use an nl-bit approximation $\widetilde{m^{-1}}_{(l)}$ of $\frac{1}{m}$ by retaining the nl most significant bits of $\widetilde{m^{-1}}$. The following inequalities hold:

$$0 \leq \frac{1}{m} - \widetilde{m^{-1}}_{(l)} < 2^{-nl} \tag{2}$$

The algorithm for modular powering, which computes Z satisfying $Z \equiv z^l \pmod m$ and $0 \leq Z < 2m$ (referred to here as Algorithm [l-MODPOWER]), is a minor modification of Algorithm [l-POWER], to which the following step is added:

Step 4. Compute $Z = z^l - \lfloor z^l \widetilde{m^{-1}}_{(l)} \rfloor m$.

With regard to the Step 4 of [l-MODPOWER], we observe the following. From inequalities (2), we have:

$$\begin{aligned} \left\lfloor \frac{z^l}{m} \right\rfloor &= \lfloor z^l \widetilde{m^{-1}}_{(l)} + z^l (\frac{1}{m} - \widetilde{m^{-1}}_{(l)}) \rfloor \\ &= \lfloor z^l \widetilde{m^{-1}}_{(l)} \rfloor \quad \text{or} \quad \lfloor z^l \widetilde{m^{-1}}_{(l)} \rfloor + 1 \end{aligned}$$

Consequently, from inequalities $0 \leq z^l - \left\lfloor \frac{z^l}{m} \right\rfloor m < m$, we have:

$$0 \leq z^l - \lfloor z^l \widetilde{m^{-1}}_{(l)} \rfloor m < 2m$$

This modular reduction can be performed by a multiplication with flooring, a multiplication and a subtraction. Since z^l and $\widetilde{m^{-1}}_{(l)}$ are nl-bit numbers and m is a n-bit number, a circuit for Step 4 has $O(\log(nl))$ depth and $O(n^2 l^2)$ size. Therefore, we can compute [l-MODPOWER] with an $O(\log(nl))$-depth $O(n^2 l^2)$-size circuit.

In the inner loop of Step 2 of the main algorithm [MODEXP], we compute $\{t(i) : i = 1, \cdots, k\}$ in parallel. Variable $t(i)$ $(1 \leq i \leq k)$ is computed using the algorithm [l-MODPOWER] for $l = 2^i$. Hence, we can compute the $t(i)$'s, for $i = 1, 2, \cdots, k$, with an $O((1 + \alpha) \log n)$-depth and $\sum_{i=0}^{k} O(n^2 (2^i)^2) = O(n^{2(1+\alpha)})$-size circuit. Since the outer loop is sequential, we can perform Step 2 with an $O(\frac{1+\alpha}{\alpha} n)$-depth $O\left(\frac{n^{3+2\alpha}}{\alpha \log n}\right)$-size circuit.

In Step 3, we compute the modular product $\prod_{i=0}^{n-1} s(i)^{y_i} \bmod m$ from inputs $s(i)$'s by connecting $(n+1)$-bit modular multiplication circuits according to a binary tree, where $(n+1)$-bit modular multiplication means the computation of $XY \bmod m$ for $(n+1)$-bit binary integers X and Y within bounds $0 \leq X, Y < 2m$. We compute $XY \bmod m$ in two steps. The first is multiplication and the second modular reduction. Consequently, we can perform the n-bit modular multiplication $XY \bmod m$ and the leading

Inputs: $x = 231$ $\quad$ $n = 8,\ \alpha = 1$

$y = 245$ $\quad$ $k = \lceil \alpha \log n \rceil = 3$

$m = 249$ $\quad$ moduli$= \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53\}$

Outputs: $x^y \bmod m = 189$

Step 1.

$\widetilde{m^{-1}} = 0.01073260A47F7C66$ (hexadecimal)

Step 2.

Iteration 1: $z = s(0) = x = 231$

Binary→RNS: $< 1, 0, 1, 0, 0, 10, 10, 3, 1, 28, 14, 9, 26, 16, 43, 19 >$

	z^{2^1}	z^{2^2}	z^{2^3}
Table look-ups:	$<1, 0, 1, 0, 0, 9, 15>$	$<1, 0, 1, 0, 0, 3, 4, 5, 1, 1>$	$<1, 0, 1, 0, 0, 9, 16, 6, 1, 1, 18, 33, 18, 16, 18, 36>$
RNS→Binary:	53361	2847396321	8107665808844335041
Modular reduction: $(z^{2^i} - \lfloor z^{2^i} \widetilde{m^{-1}}_{(2^i)} \rfloor m)$	$s(1) = 75$	$s(2) = 147$	$s(3) = 195$

Iteration 2: $z = s(3) = 195$

Binary→RNS: $< 1, 0, 0, 6, 8, 0, 8, 5, 11, 21, 9, 10, 31, 23, 7, 36 >$

	z^{2^1}	z^{2^2}	z^{2^3}
Table look-ups:	$<1, 0, 0, 1, 9, 0, 13>$	$<1, 0, 0, 1, 4, 0, 16, 17, 13, 7>$	$<1, 0, 0, 1, 5, 0, 1, 4, 8, 20, 28, 26, 16, 9, 16, 49>$
RNS→Binary:	38025	1445900625	2090628617375390625
Modular reduction: $(z^{2^i} - \lfloor z^{2^i} \widetilde{m^{-1}}_{(2^i)} \rfloor m)$	$s(4) = 177$	$s(5) = 204$	$s(6) = 33$

Iteration 3: $z = s(6) = 33$

Binary→RNS: $< 1, 0, 3, 5, 0, 7, 16, 14, 10, 4, 2, 33, 33, 33, 33, 33 >$

	z^{2^1}	z^{2^2}	z^{2^3}
Table look-ups:	$<1, 0, 4, 4, 0, 10, 1>$	$<1, 0, 1, 2, 0, 9, 1, 17, 18, 24>$	$<1, 0, 1, 4, 0, 3, 1, 4, 2, 25, 8, 9, 16, 17, 7, 49>$
RNS→Binary:	1089	1185921	1406408618241
Modular reduction: $(z^{2^i} - \lfloor z^{2^i} \widetilde{m^{-1}}_{(2^i)} \rfloor m)$	$s(7) = 93$	183	123

Step 3.

$$\prod_{i=0}^{7} s(i)^{y_i} \bmod z = ((231 \times 1 \bmod 249) \times (147 \times 1 \bmod 249) \bmod 249) \times ((177 \times 204 \bmod 249) \times (33 \times 93 \bmod 249) \bmod 249) \bmod 249 = 189$$

(Numbers are described in decimal unless otherwise stated.)

Figure 2: Example of modular exponentiation

$(2n+2)$-bits of $\widetilde{m^{-1}}$ as in Step 4 of [l-MODPOWER] with an $O(\log n)$-depth $O(n^2)$-size circuit. Therefore, Step 3 can be performed by an $O(\log^2 n)$-depth $O(n^3)$-size circuit.

It is clear that the circuit based on the algorithm [MODEXP] is P-uniform, because the circuit for powering is P-uniform. We summarize the proposed algorithm into the following theorem.

Theorem 1 *n-bit modular exponentiation $x^y \bmod m$ can be performed with an $O(\frac{1+\alpha}{\alpha}n)$-depth $O\left(\frac{n^{3+2\alpha}}{\alpha \log n}\right)$-size P-uniform circuit, where $\alpha \geq \frac{1}{\log n}$ is a positive constant.*

4 Example

We give a numerical example of 8-bit modular exponentiation based on the proposed algorithm [MODEXP]. Figure 2 shows the computation flow of modular exponentiation $231^{245} \bmod 249$ for $n = 8$. We set $\alpha = 1$. Therefore, the set of moduli is $\{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53\}$. In Step 1, we compute 64-bit number $\widetilde{m^{-1}}$. Step 2 proceeds in $\lceil \frac{n-1}{k} \rceil = 3$ rounds. In each round, we compute $k = 3$ numbers in parallel. In Step 3, since $y = 11110101$ in binary, we compute $(s(0) \cdot s(2) \cdot s(4) \cdot s(5) \cdot s(6) \cdot s(7)) \bmod m$.

5 Conclusion

We have proposed an $O\left(\frac{1+\alpha}{\alpha}n\right)$-depth $O\left(\frac{n^{3+2\alpha}}{\alpha \log n}\right)$-size P-uniform circuit algorithm for n-bit modular exponentiation with $\alpha \geq \frac{1}{\log n}$. Note that for $\alpha = \frac{1}{\log n}$ we obtain the classical square-and-multiply algorithm.

J. von zur Gathen[9] showed that modular exponentiation can be computed in an $O(\log n)$-depth polynomial-size circuit if the modulus m has only small prime factors $p \leq n$, i.e. m is "n-smooth". In his algorithm, this property of the modulus m is essential. Therefore, the technique of [9] cannot be applied to modular exponentiation with an arbitrary modulus as assumed in this paper.

Our result indicates that n-bit modular exponentiation belongs to a complexity class in which operations are executed by an $O(n)$-depth polynomial-size circuit. It is an open problem whether there exists a $\log^{O(1)} n$-depth circuit for such operations.

Acknowledgments

The authors would like to thank the referees for their useful comments which improved this paper. The authors would like to thank the members of Professor Yajima's Laboratory at Kyoto University for their valuable comments on this work.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signature and public-key cryptosystems," *Commun. ACM*, Vol. 21, No. 2, pp. 120–126, Feb. 1978.

[2] D. E. Knuth, *The Art of Computer Programming volume 2*, Addison-Wesley, Reading, Massachusetts, 1981.

[3] E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography," in *Advances in Cryptology, Proc. CRYPTO 82*, D. Chaum *et al.*, Eds. New York: Plenum, 1983, pp.51–60.

[4] P. L. Montgomery, "Modular multiplication without trial division," *Math. Computation*, Vol. 44, No. 170, pp.519–521, Apr. 1985.

[5] P. W. Beame, S. A. Cook, and H. J. Hoover, "Log depth circuits for division and related problems," *Proc. 25th IEEE FOCS*, pp. 1–6, Oct. 1984.

[6] Y. Okabe, N. Takagi, and S. Yajima, "Log depth circuits for elementary functions using residue number system," *IEICE Trans.*, Vol. J73-D-I, No. 9, pp.723–728, Sep. 1990 (in Japanese). English translated version is available in *Electronics and Communications in Japan*, Part 3, Vol. 74, No. 8, pp.31–38, 1991.

[7] K. Mehlhorn and F. P. Preparata, "Area-time optimal division for $t = \Omega((\log n)^{1+\epsilon})$," *Information and Computation*, Vol. 72., pp.270–282, 1987.

[8] N. S. Szavo and R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*, McGraw-Hill, New York, 1967.

[9] J. von zur Gathen, "Computing powers in parallel," *SIAM J. Comput.*, Vol. 16, No. 5, pp. 930–945, Oct. 1987.