Available online at www.sciencedirect.com
SCIENCE DIRECT®
Applied Mathematics and Computation 176 (2006) 280–292
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

How to compute modular exponentiation with large operators based on the right-to-left binary algorithm

Da-Zhi Sun [a,*], Zhen-Fu Cao [a,*], Yu Sun [b]

[a] *Department of Computer Science, Shanghai Jiao Tong University, 1954 HuaShan Road, P.O. Box 282, Shanghai 200030, PR China*
[b] *Department of Management, Beijing Normal University, Beijing 100875, PR China*

Abstract

When the lengths of the operators are at least 1024 binary or 300 decimal digits, modular exponentiation can be time-consuming and is often the dominant part of the computation in many computer algebra systems. The prime approach on this computational problem is known as the square-and-multiply method, which includes two versions, i.e. the left-to-right binary algorithm and the right-to-left binary algorithm. For the past years, too many attentions have been paid to propose the fast modular exponentiation methods based on the left-to-right binary algorithm. However, extremely few attentions have been paid on developing the fast modular exponentiation methods based on the right-to-left binary algorithm. In this paper, we propose a t-fold exponent method based on the right-to-left binary algorithm. From the performance view, our t-fold exponent method is similar to the m-ary method based on the left-to-right binary algorithm. From the structure view, our t-fold exponent method offers a framework for the fast modular exponentiation methods based on the right-to-left binary algorithm. More important, it is the first but steady step to develop the fast modular exponentiation methods based on the right-to-left binary algorithm.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Computer algebra system; Modular exponentiation; Right-to-left binary algorithm; t-fold exponent method; Performance; Framework

1. Introduction

Modular exponentiation is a fundamental operation in computational number theory. The computation of modular exponentiation is widely required in many interesting and useful topics, which include all kinds of primality tests based on Fermat's little theorem, finding a primitive root modulo an integer, and deciding the order of an integer modulo an integer, etc. When the lengths of the operators are at least 1024 binary or 300 decimal digits, modular exponentiation can be time-consuming and is often the dominant part of the computation in many computer algebra systems. So, a research direction is to design the efficient modular exponentiation method, which often determines whether those given systems are practical.

[*] Corresponding authors.
E-mail addresses: sundazhi1977@126.com, sundzhmail@sohu.com, sundazhi@sjtu.edu.cn (D.-Z. Sun), zfcao@cs.sjtu.edu.cn (Z.-F. Cao).

0096-3003/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2005.09.062

Without loss generality, modular exponentiation can be written as x^E omitting the modulus, where the operators x and E are positive integers. We similarly imply moduli on the representations of modular multiplication and square, i.e. $A_1 \cdot A_2$ and $A_1 \cdot A_1$, where the operators A_1 and A_2 are positive integers.

For wide applications, we only consider the basic technique for modular exponentiation. That is, arbitrary choices of the base x and the exponent E are allowed.

1.1. Related work

There are two primary ways to reduce the time on the computation of modular exponentiation with large operators. One is to decrease the time to perform basic modular multiplication; the other is to reduce the number of modular multiplications used to compute x^E. We mainly focus on the latter way in his paper. Hence, the essential question is: what is the fewest number of modular multiplications necessary to compute x^E, given that the only operation permitted is multiplying two already-computed powers? This is equivalent to the question: What is the length of the shortest addition chain for the exponent E? An addition chain for E is a list of positive integers $a_1 = 1, a_2, \ldots, a_l = E$, such that, for each $r > 1$, there exist some i and j with $1 \leqslant i, j < r$ and $a_r = a_i + a_j$. Obviously, a short addition chain for E gives a fast algorithm for computing x^E through the line $x^{a_2}, x^{a_3}, \ldots, x^{a_{l-1}}, x^{a_l} = x^E$. See [1] for an excellent introduction about the addition chain. Unfortunately, Downey et al. [2] showed that the problem of finding the shortest addition chain is established to be an NP-complete problem. It means that directly finding the shortest addition chain is impractical. Naturally, many modular exponentiation methods have been proposed to achieve near optimal addition chain. The lower bound on the length of the addition chain was given by Erdös [3] using a counting argument, and the upper bound on it can be decided by the m-ary method [1].

The prime approach to compute modular exponentiation with large operators is commonly known as the square-and-multiply method. Knuth [1] discussed its history and gave references. Two versions of the square-and-multiply method are given in Fig. 1. The exponent E has k bits, where the least significant bit (LSB) is numbered 1, and the most significant bit (MSB) is numbered k. Algorithm A is the left-to-right binary algorithm, which starts at the most significant bit and works downward. Most of fast modular exponentiation methods are based on this left-to-right binary algorithm, e.g. the m-ary method [1], the adaptive method [4], and the window methods [5–8]. Algorithm B is the right-to-left binary algorithm, which starts at the least significant bit and works upward. It requires an extra data register S to store the middle variable. Note that modular multiplication and square in this right-to-left binary algorithm are independent of one another, and thus two operations at each loop can be parallelized. Provide that one multiplier and one squarer available, the running time of the right-to-left binary algorithm is bounded by the total time required for computing k modular squares.

The surveys [9,10] summarized the previous modular exponentiation methods.

```
Algorithm A (Left-to-Right)
Input: $x, E$
Output: $x^E$ contained in $C$
if( $k$th binary bit of $E$ is 1)
then $C = x$ else $C = 1$;
for $i = k-1$ downto 1
{
    $C = C \cdot C$;
    if ($i$th binary bit of $E$ is 1)
    then $C = C \cdot x$;
}
```

```
Algorithm B (Right-to-Left)
Input: $x, E$
Output: $x^E$ contained in $C$
$S = x$;
$C = 1$;
for $i = 1$ to $k$
{
    if ($i$th binary bit of $E$ is 1)
    then $C = C \cdot S$;
    $S = S \cdot S$;
}
```

Fig. 1. Classic square-and-multiply method.