Applied Mathematics and Computation 292 (2017) 406–416

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Parallel algorithms for modular multi-exponentiation

Fábio Borges [a,b,*], Pedro Lara [b], Renato Portugal [b]

[a] Technische Universität Darmstadt, Darmstadt 64289, Germany
[b] Laboratório Nacional de Computação Científica, Petrópolis 25651-075, Brazil

ARTICLE INFO

Keywords:
Modular exponentiation
Modular multi-exponentiation
Parallelization
Algorithms
Cryptography

ABSTRACT

Modular exponentiation is a time-consuming operation widely used in cryptography. Modular multi-exponentiation, a generalization of modular exponentiation also used in cryptography, deserves further analysis from the algorithmic point of view. The parallelization of modular multi-exponentiation can be obtained by generalizing methods used to parallelize modular exponentiation. In this paper, we present a new parallelization method for the modular multi-exponentiation operation with two optimizations. The first one searches for the fastest solution without taking into account the number of processors. The second one balances the load among the processors and finds the smallest number of processors that achieves the fastest solution. In detail, our algorithms compute the product of i modular exponentiations. They split up each exponent in j blocks and start j threads. Each thread processes together i blocks from different exponents. Thus, each block of an exponent is processed in a different thread, but the blocks of different exponents are processed together in the same thread. Using addition chains, we show the minimum number of threads with load balance and optimal running time. Therefore, the algorithms are optimized to run with the minimum time and the minimum number of processors.

1. Introduction

Many algorithms in cryptography need to use the modular-exponentiation operation, such as the famous Diffie–Hellman key exchange protocol [1], which is widely employed in communication channels. In an early stage, cryptographers observed the necessity to compute the product of two modular exponentiations in the ElGamal signature scheme [2]. The importance of multi-exponentiation became more evident with the Digital Signature Algorithm (DSA) standardized by the National Institute of Standards and Technology (NIST). Moreover, this operation is used in privacy-enhancing protocols [3–5], and it is known that many people are currently concerned about privacy issues. Multi-exponentiation can also be employed in other algebraic structures such as the ones used in elliptic-curve cryptosystems.

Algorithms for modular multi-exponentiation are stable for non-parallel computers [6]. However, the scenario for hardware development is changing. Companies that manufacture computer chips are not increasing the clock frequency; instead, they are increasing the number of cores inside the processors. Moreover, graphical processing units have a large number of cores that are available for general processing. Algorithms that parallelize the usual modular exponentiation operation have been developed [7], which can be used to speed up many other algorithms in the context of communication protocols. To

* Corresponding author at: Laboratório Nacional de Computação Científica, 25651-075 Petrópolis, Brazil.
E-mail addresses: borges@lncc.br (F. Borges), pedro.lara@cefet-rj.br (P. Lara), portugal@lncc.br (R. Portugal).
URL: http://www.lncc.br/~borges/ (F. Borges), http://www.lncc.br/~portugal/ (R. Portugal)

http://dx.doi.org/10.1016/j.amc.2016.07.036
0096-3003/

further enhance the performance of such algorithms and protocols, it is interesting to develop parallel algorithms for modular multi-exponentiation [8]. The case of the product of two exponentiations is especially interesting, because it is used in many cryptographic algorithms such as DSA, Paillier Cryptosystem, and others [2–5].

In this work, we present new parallel algorithms to compute the modular multi-exponentiation operation by re-arranging the exponents in binary notation. This analytical manipulation generalizes the one used for a single exponentiation and can be parallelized with similar techniques. The main advantage is to reduce the number of squarings in comparison with a straightforward calculation. We analyze the optimal load balance and obtain analytical expressions for the partition of the exponents. Our algorithms can be used to compute a single modular exponentiation and can be applied to the Diffie–Hellman [1] and RSA [9] protocols. They have better performance than parallel algorithms developed to a single modular exponentiation such as the ones in [7,10]. We analyze the performance of our algorithms and present simulations that confirm our results.

The present work can be improved by adding other known techniques, such as pre-computation, sliding windows, or NAF notation [11–13]. The processing time can be reduced by using hardware properties [14] or by implementing the algorithms in hardware [15]. Pre-computation only speeds up when the bases are fixed. Similarly, sliding windows and NAF notation have advantage only when the exponents obey some restrictions, which depend on their Hamming weight. Despite the good performance, implementations in hardware are costly. Our solution does not depend on fixed bases, Hamming weight, or hardware architecture. We start by describing the basic square and multiply algorithm until achieving new parallel algorithms, which are asymptotically faster than related work.

The structure of this paper is as follows. In Section 2, we describe the arithmetic background. In Section 3, we present the new parallelization method. In Section 4, we analyze the performance of the algorithms. In Section 5, we describe how the load balance is achieved. In Section 6, we compare our results with related ones in Literature. In Section 7, we present the simulations. In Section 8, we draw our conclusions.

2. Background

Given three-integer numbers b, e, and m, modular exponentiation is the operation that computes b raised to the power of e modulo m, namely $b^e \mod m$. Similarly, modular multi-exponentiation is the operation that computes

$$\prod_{i=1}^{n} b_i^{e_i} \mod m, \tag{1}$$

for integers $b_1, b_2, \ldots, b_n$ and $e_1, e_2, \ldots, e_n$. As we discussed in the introduction, the most interesting values of n are one and two. However, this technique can be applied to speed up the computation of Diophantine equations [16] for large n. A fast way to compute $b^{e_i} \mod m$ is to write the exponent in its binary form

$$e_i = \sum_{j=1}^{l} 2^{j-1} e_{i_j}, \tag{2}$$

where $e_{i_j} \in \{0, 1\}$ and l is the number of bits of e_i, which is usually called as bit-length. Notice that the first index indicates an exponent and the second indicates its bit position. Then, to compute $b^{e_i} \mod m$ we employ

$$b^{e_i} \mod m = \left(\cdots \left(\left(b^{e_{i_l}} \right)^2 b^{e_{i_{l-1}}} \right)^2 \cdots \right)^2 \cdot b^{e_{i_1}} \mod m. \tag{3}$$

We can rewrite Eq. (3) as a recursive function

$$\mathrm{MExp}(L) = \begin{cases} b^{e_1} & \text{if } L = 1 \\ b^{e_{l-L+1}} \cdot \mathrm{MExp}(L-1)^2 & \text{if } L > 1 \end{cases} \tag{4}$$

that starts with $\mathrm{MExp}(l)$.

Analogously, we have

$$\prod_{i=1}^{n} b_i^{e_i} \mod m = \left(\cdots \left(\left(\prod_{i=1}^{n} b_i^{e_{i_l}} \right)^2 \prod_{i=1}^{n} b_i^{e_{i_{l-1}}} \right)^2 \cdots \right)^2 \cdot \prod_{i=1}^{n} b_i^{e_{i_1}} \mod m. \tag{5}$$

We can rewrite Eq. (5) as the following recursive function:

$$\mathrm{MMExp}(L) = \begin{cases} \prod_{i=1}^{n} b_i^{e_i} & \text{if } L = 1 \\ \prod_{i=1}^{n} b_i^{e_{i_{l-L+1}}} \cdot \mathrm{MMExp}(L-1)^2 & \text{if } L > 1 \end{cases}, \tag{6}$$

where L starts with the bit-length of the largest exponent e_i, *i.e.*,

$$L \leftarrow \lceil \max(\log_2 e_1, \ldots, \log_2 e_n) \rceil. \tag{7}$$