Journal of Computer Science 6 (5): 519-524, 2010
ISSN 1549-3636

Methods of Fast Exponentiation

Mohammed Al-Maitah
Department of Computer Science, Alriadh Community College,
King Saud University, Saudi Arabia

Abstract: Problem statement: Modular exponentiation constitutes the basis of many well-known and widely used public key cryptosystems. **Approach:** A fast portable modular exponentiation algorithm considerably enhanced the speed and applicability of these systems, also an efficient implementation of this algorithm was the key to high performance of such system. **Results:** In this study, two main approaches for solving this problem were proposed. The proposed approaches involved calculations without usage of extra operational memory for saving constants and calculations with usage of preliminary calculated constants. **Conclusion/Recommendations:** The estimation of complexity of the speedup and effectiveness of proposed approaches for the data were presented.

Key words: Exponentiation, extra operational memory, consistent calculation, Fibonacci calculus

INTRODUCTION

Protection of information is one of the most important and complicated problems in computer equipment. Specialists face the task of creation of systems for transmission of information, which would maintain the high level of protection and security.

Nowadays the speediest systems of confidential information transmission are the systems with open keys. The majority of these systems use operation of modular exponentiation x^y mod z as basic for great numbers (Diffie and Hellman, 1976). The performance of given operation requires high expenses of machinery time. That is why the necessarily of this procedure optimization raises. The given problem can be decided with the help of fastening of exponentiation operation performance for numbers of high digit capacity.

There two main approaches solving this problem to be underlined: the calculation without usage of extra operational memory for saving of constants and calculation with usage of preliminary calculated constants, which will be saved in memory location. The purpose of the work is investigation of exactly these methods of exponentiation.

MATERIALS AND METHODS

Calculation without usage of extra operational memory for saving of constants: In systems of calculus with alphabet {0, 1} numbers are given in such way:

$$N = \sum_{i=0}^{n-1} a_i \cdot w_i \quad (1)$$

From this we have:

$$x^N = x^{\sum_{i=0}^{n-1} a_i w_i} = \prod_{i=0}^{n-1} x^{a_i w_i} \quad (2)$$

For binary calculus the expression (2) takes the form of:

$$x^N = \prod_{i=0}^{n-1} x^{a_i \cdot 2^i} \quad (3)$$

With the help of Horner's method the formula (3) can be represented by means of the following expression:

$$x^N = (...(((x^{a_{n-1}})^2 \cdot x^{a_{n-2}})^2 \cdot x^{a_{n-3}})^2 \cdot ... \cdot x^{a_1})^2 \cdot x^{a_0} \quad (4)$$

The expression (4) is basis for description of the "binary" method of quick exponentiation (Knuth, 1981) resulting in the fact that the process of exponentiation for the numbers represented in binary calculus comes to execution of the following calculations:

$$R_i = \begin{cases} R_{i-1}^2, \text{ if } a_{n-i} = 0; \\ R_{i-1}^2 \cdot x, \text{ if } a_{n-i} = 1 \end{cases}$$

where, R_i-intermediate result; i = 1, 2 …n.

The number of multiplication operations n_m required for the realization of the exponentiation can be calculated according to the formula:

$$n_m = n + n_1 \quad (5)$$

Where:
n = Number of digits in representation of the number
n_1 = Number of units in the code

As the formula (5) shows, the number of multiplication operations depends on the code capacity and probability of occurrence of a unity in site. There are two methods to reduce the number of units in code: by means of representation of numbers in calculus's with alphabet {0, 1, -1} or in redundant calculus's with alphabet {0, 1}, namely in Fibonacci Calculus (FC). At usage of the first method because of the presence of code digit Fig. 1 the necessity of execution of division operation, which requires high time spending (Knuth, 1981). That is why representation of numbers in FC is more reasonable.

Any natural number in FC is represented as the following (Stakhov and Luzhetsky, 1981):

$$N = a_{n-1}\varphi_p(n-1) + a_{n-2}\varphi_p(n-2) + ... + a_0\varphi_p(0)$$

Where:

$$\varphi_p(n) = \begin{cases} 0, & \text{if } n < 0, \\ 1, & \text{if } n = 0, \\ \varphi_p(n-1) + \varphi_p(n-p-1), & \text{if } n > 0 \end{cases}$$

p = 1, 2, 3…

Hence expression takes on the following form:

$$x^N = \prod_{i=0}^{n-1} x^{a_i w_i} = \prod_{i=0}^{n-1} x^{a_i \varphi_p(i)}$$

Process of exponentiation for numbers represented in FC comes to execution of the following calculations:

$$R_{i+p+1} = \begin{cases} R_i \cdot R_{i+p}, & \text{if } a_{n-p-i} = 0; \\ R_i \cdot R_{i+p} \cdot x, & \text{if } a_{n-p-i} = 1 \end{cases}$$

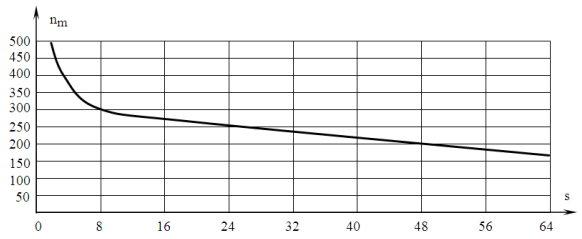

Fig. 1: Diagram of dependence of the number of multiplication operations upon the value of frame of calculus s

Reduction of probability of occurrence of unity in code leads to the increase of digit capacity of the code. For calculus with alphabet {0, 1} in equation is valid:

$$w_i \le \sum_{l=1}^{i} w_{i-l} + 1$$

Let:

$$w_i = \sum_{l=1}^{i} w_{i-l} + 1$$

Then according to expression (2):

$$\left(x^N\right)_1 = \prod_{i=0}^{n-1} x^{a_i\left(\sum_{l=1}^{i} w_{i-l}+1\right)}$$

Let:

$$w_i < \sum_{l=1}^{i} w_{i-l} + 1$$

Then:

$$\left(x^N\right)_2 = \prod_{i=0}^{n-1} x^{a_i\left(\sum_{l=1}^{i} w_{i-l}+1-r\right)}$$

If the number of digits n for representation of number N is same for different Calculus's, than the following in equation is valid:

$$\left(x^N\right)_2 < \left(x^N\right)_1$$

So, for maintenance of minimal number of multiplications it is reasonable to represent data in the calculus, for which the following equation is valid:

$$w_i = \sum_{l=1}^{i} w_{i-l} + 1$$

i.e., in binary calculus.

It is known that consistent calculation character is a principal lack of Horner's method. That is why for parallelization of calculations let us represent the other method of quick exponentiation described as follows:

$$x^N = \prod_{i=0}^{n-1} x^{a_i 2^i} = \prod_{i=0}^{n-1} \left(x^{2^i}\right)^{a_i}$$

So, the process of exponentiation comes to execution of the following calculations:

$$R_i = \begin{cases} R_{i-1}^2, & \text{if } a_{n-i} = 0; \\ R_{i-1}^2 \cdot R_{i-1}^2, & \text{if } a_{n-i} = 1 \end{cases} \quad (6)$$

According to expression (6) the number should be exponentiated in correspondence with the weights of code digits and the results with value of figure of code site equal unity should be multiplied. Peculiarity of this method for parallel calculation character lies in the fact that the number of units in code does not influence the duration of exponentiating process, as multiplication of grades occurs simultaneously with the formation of x^{2i} dependently on values a_i. In other words, the number of multiplication equals to the number of digits of binary code of a figure. The speedup S and effectiveness E of the given algorithm in consideration with «binary» is calculated according to the formulas:

$$S_{par.} = \frac{T_{seq.}}{T_{par.}}$$

Where:
$T_{seq.}$ = Time of fulfillment of sequential algorithm
$T_{per.}$ = Time of fulfillment of parallel algorithm:

$$E_{par.} = \frac{S_{par.}}{m}$$

where, m-number of processors.

So, taking into account the expression (5) speedup of parallel algorithm equals to:

$$S_{par.} = \frac{n + n_1}{n} = \frac{1.5n}{n} = 1.5$$

As far as it is necessary to have 2 processors for the fulfillment of parallel algorithm, its effectiveness amounts to:

$$E_{par.} = \frac{S_{par.}}{2} = 0.75$$

Calculation with usage of previously calculated constants: At usage of certain methods information transfer with open keys, where the calculation of value x^y mod z, specifically in Diffie-Helman's method, values x and z are considered as constants. It is also necessary to admit that relative cost of memory is considerably lower than relative cost of processor. That is why at fulfillment of operation of exponentiation it is necessary to use method of calculation with implementation of previously calculated constants $x^{a_i w_i}$, which will be saved in memory and skimmed if necessary. So, average number of operations of multiplication that are of sequential character will equal to the number of nonzero values in code digits.

Let us estimate the ratio of average number of operations of multiplication for figures represented in different Calculus's.

For Calculus's with frame $S = 2^t$, where t-whole positive figure, number of operations of multiplication in case of sequential fulfillment of operation of exponentiation is calculated according to the formula:

$$n_m = n_s \cdot w_s = n_s \cdot (1 - 1/s) = \\ = \frac{n}{]\log_2 s[} \cdot (1 - 1/s) = \frac{(s-1) \cdot n}{s \cdot]\log_2 s[}$$

Where:
n_s = Number of digits for representation of number in calculus with frame s
w_s = Probability of occurrence of nonzero value in i-digit of code
n = Number of digits in binary code of figure

Analysis of diagram of dependence of the number of multiplication operations upon the value of frame of calculus s (Fig. 1) shows that representation of data in calculus's of frame s = 4.16 has the highest advantage in number of operations.

For parallel process the fulfillment of exponentiation operation under the method of logarithmic summability for figures represented in calculus's with frame $s = 2^t$ the number of multiplication operations is determined according to the formula:

$$n_{m.par.} = \log_2 n_s$$

As far as at fulfillment of exponentiation operation only nonzero values are taken into account, the number of operations can be calculated with the help of the following expression:

$$n_{m.par.} = \log_2 n_s \cdot w_s = \log_2 n_{m.seq.}$$

Diagram of dependence of the number of multiplication operations upon the value of frame *s* at use of parallel algorithm is shown on Fig. 2.

In this case speedup of parallel algorithm equals to:

$$S_{par.} = \frac{n_{m.seq.}}{\log_2 n_{m.seq.}}$$

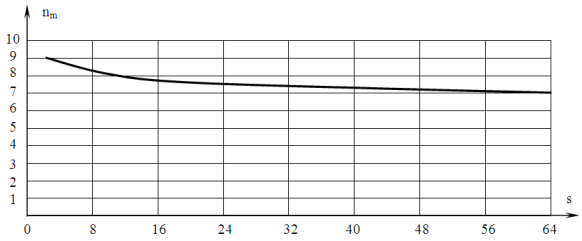

Fig. 2: Diagram of dependence of the number of multiplication operations upon the value of frame *s* at use of parallel algorithm

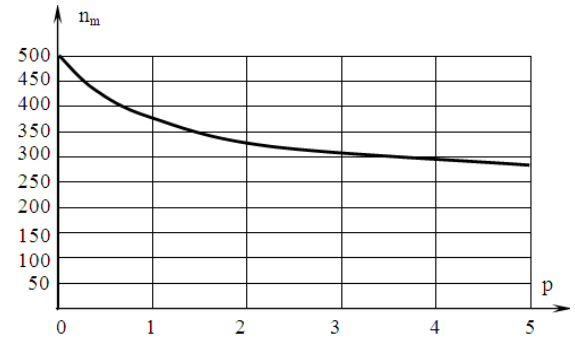

Fig. 3: Diagram of dependence of the number of multiplication's upon the value of p

Is necessary to have $(n_s w_s\text{-}1)$ processors to carry out the given algorithm. That is why its effectiveness amounts to:

$$E_{par.} = \frac{n_{m.seq.}}{(n_{m.seq.} - 1)\log_2 n_{m..seq}}$$

For FC average number of multiplication's for subsequent character of operation fulfillment is calculated according to the formula:

$$n_m = n_F \cdot w_F$$

Where:
n_F = Number of digits in Fibonacci code
w_F = Probability of occurrence of the unity in code digit

Then:

$$n_F = \frac{n + \log_2 k}{\log_2 \alpha_p} - 1$$

Where:

$$w_F = \frac{1}{k \cdot \alpha_p{}^p}$$

k = Coefficient
α_p = "golden" p-ration (Stakhov and Luzhetsky, 1981)

Table 1: The value of the coefficient k

p	0	1	2	3	4	5
k	2	2.2351	2.3979	2.5119	2.6363	2.6963

Table 2: The value of the coefficient α_p

p	0	1	2	3	4	5
α_p	2	1.618	1.63	1.381	1.325	1.287

Hence:

$$n_m = \frac{1}{k \cdot \alpha_p{}^p}\left(\frac{n + \log_2 k}{\log_2 \alpha_p} - 1\right) \quad (7)$$

The values of the k and a are listed in the Table 1 and 2 correspondingly.

On the basis of the expression (7) was created a diagram of dependence between the average number of the operations of multiplication and the value of p (Fig. 3). The diagram analysis shows that fulfillment of exponentiation over the data operation, represented in FC, requires greater number of the operations than over the data, represented in calculus systems with frame of $s = 2^t$.

In case of using parallel algorithm the amount of multiplication operations equals to:

$$n_{m.par.} = \log_2 n_F \cdot w_F = \log_2 n_{m.seq.}$$

Let's estimate speedup and effectiveness of parallel algorithm for the data represented in FC:

$$S_{par.} = \frac{n_{m.seq.}}{\log_2 n_{m.seq.}}$$

In case of using $(n_F \cdot w_F - 1)$ processors the effectiveness states:

$$E_{par.} = \frac{n_{m.seq.}}{(n_{m.seq.} - 1)\log_2 n_{m.seq.}}$$

Having analyzed the diagram of dependence of the number of multiplication operations upon the value of p and the value of speedup and effectiveness in case of using parallel algorithm for data Fig. 4, represented in FC, we may conclude the following: to provide the minimum amount of multiplication operations and to provide the greater effectiveness of parallel algorithm it's reasonable to represent the data in calculus systems with frame of $s = 2^t$.

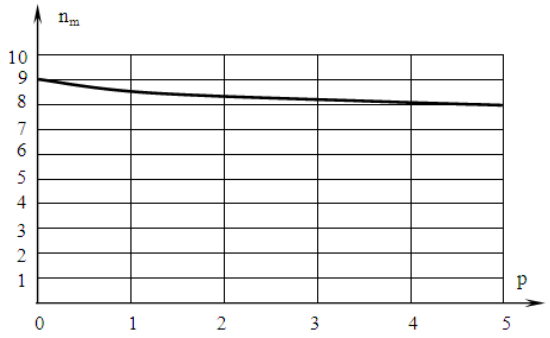

Fig. 4: Diagram of dependence of the number of multiplication operations upon the value of p in case of usage of parallel algorithm

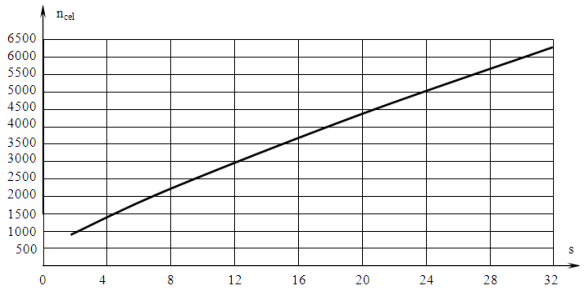

Fig. 5: Diagram of dependence of the number of cells upon the value of s

Except the calculation of time expenses the weight meaning has the memory volume, necessary for saving constants. That's why let's evaluate the memory expenses in the amount of cells for different calculus systems.

Diagram of dependence of the amount of cells to save constants upon the value of *s* is shown on Fig. 5.

The amount of cells for saving constants at representing data in calculus systems with frame of *s* we can be calculated according to the following formula:

$$n_{cel} = (s-1)\cdot n_s = (s-1)\cdot \frac{n}{]\log_2 s[}$$

Digit capacity of code of a figure represented in FC is R_F times as much as the digit capacity of binary code (Stakhov and Luzhetsky, 1981):

$$n_F = n \cdot R_F$$

where for $n_F \to \infty$:

$$R_F = 1 + \frac{1}{\log_2 \alpha_p}$$

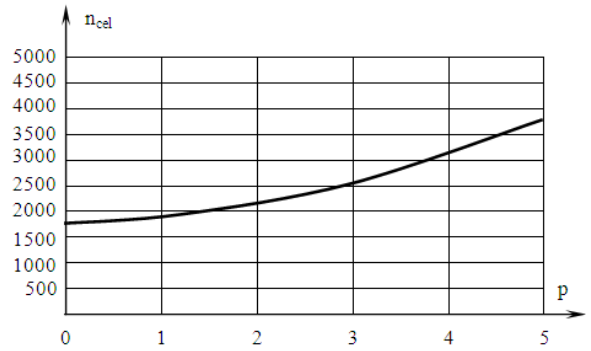

Fig. 6: Diagram of dependence of amount of memory cells upon p parameter

Considering that fact we have diagram of dependence of amount of memory cells upon the p parameter (Fig. 6).

Comparative analysis of diagrams of dependence (Fig. 5 and 6) shows that the fulfillment of exponentiation operation in case of representation of figure in FC requires the less amount of memory cells, than in case of representation of figures in calculus's with frame of $s = 2^t$.

RESULTS AND DISCUSSION

- Diagram of dependence of the number of multiplication operations upon the value of frame of calculus s has the highest advantage in number of operations
- Parallel algorithm with frame of $s=2^t$ provides the minimum amount of multiplication operations and provides the greater effectiveness
- Fulfillment of exponentiation operation in case of representation of figure in FC requires the less amount of memory cells, than in case of representation of figures in calculus's with frame of $s = 2^t$

CONCLUSION

- Exponentiation of figures with high digit capacity without use of extra random-access memory for storage of constants with minimum number of operations supports representation of figures in binary calculus
- Exponentiation of figures with high digit capacity with use of previously calculated constants is carried out with minimum number of operations with the highest effectiveness in case of representation of figures in calculus's with frame of $s = 2^t$ and representation of figures in FC requires the least memory volume

REFERENCES

Diffie, W. and M.E. Hellman, 1976. New directions in cryptography. http://www.cs.rutgers.edu/~tdnguyen/classes/cs671/presentations/Arvind-NEWDIRS.pdf

Knuth, D.E., 1981. The Art of Computer Programming: Seminumerical Algorithm. Addison-Wesley, pp: 2.

Stakhov, A.P. and V.A. Luzhetsky, 1981. Machine arithmetic of digital computers in Fibonacci codes and golden proportion. Scientific Council of Academy of Science of the USSR on Complex Problem. Cybernetics, pp: 64.