UDC 004.421

THE RUNTIME ANALYSIS OF COMPUTATION OF MODULAR EXPONENTIATION

Prots'ko I. – Dr. Sc., Associate Professor, Department of Automated Control Systems, Lviv National Polytechnic University, Lviv, Ukraine.
Kryvinska N. – Dr. Sc., Professor, Department of Information System, Comenius University in Bratislava, Bratislava, Slovak Republic.
Gryshchuk O. – Software Developer, LtdC "SoftServe", Lviv, Ukraine.

ABSTRACT

Context. Providing the problem of fast calculation of the modular exponentiation requires the development of effective algorithmic methods using the latest information technologies. Fast computations of the modular exponentiation are extremely necessary for efficient computations in theoretical-numerical transforms, for provide high crypto capability of information data and in many other applications.

Objective – the runtime analysis of software functions for computation of modular exponentiation of the developed programs based on parallel organization of computation with using multithreading.

Method. Modular exponentiation is implemented using a 2^k-ary sliding window algorithm, where k is chosen according to the size of the exponent. Parallelization of computation consists in using the calculation of the remainders of numbers raised to the power of 2^i modulo, and their further parallel multiplications modulo.

Results. Comparison of the runtimes of three variants of functions for computing the modular exponentiation is performed. In the algorithm of parallel organization of computation with using multithreading provide faster computation of modular exponentiation for exponent values larger than 1K binary digits compared to the function of modular exponentiation of the MPIR library. The MPIR library with an integer data type with the number of binary digits from 256 to 2048 bits is used to develop an algorithm for computing the modular exponentiation with using multithreading.

Conclusions. In the work has been considered and analysed the developed software implementation of the computation of modular exponentiation on universal computer systems. One of the ways to implement the speedup of computing modular exponentiation is developing algorithms that can use multithreading technology on multi-cores microprocessors. The multithreading software implementation of modular exponentiation with increasing from 1024 the number of binary digit of exponent shows an improvement of computation time with comparison with the function of modular exponentiation of the MPIR library.

KEYWORDS: modular exponentiation, parallel computation, multithreading, big numbers.

ABBREVIATIONS

GMP is a GNU Multiple Precision Arithmetic library;
ME is a modular exponentiation;
MT is a multithreading;
MPIR is a Multiple Precision Integers and Rationals library.

NOMENCLATURE

a is a base integer value;
b is an integer value of the exponent;
b_k is a binary bit of b;
b.i is a binary digit;
c is an integer value;
exp is exponent;
k is a number of bits of b;
n is an integer value of the module;
mpz_t is a type of big numbers MPIR library;
quit_thread is a number of threads;
Ri is a result of the multiplication of exponent 2^i of the number a under modulo n;
s_cv is a variable of synchronizations;
s_mutex is an object for interacting;
x is a discrete logarithm;
y is an integer value of modular exponentiation.

INTRODUCTION

The task of developing an effective computational algorithm for ME for big numbers is relevant enough to solve the problems of modern asymmetric cryptography, for efficient computation of number-theoretic transforms, digital signatures and other applications [1].

The object of study is the process of analysis the developed software implementation of the computation of ME. The iteration impedes of the organization of parallel computations and does not provide high-speed computation of a ME in available computational resources of multicores computer systems. Therefore, one directions for speedup the computing of ME is to improve the software implementation using MT technology.

The subject of study is the parallel organization of computation of ME based on the use the bits of the binary exponent with MT execution of the computation.

The purpose of the work is to increase the speed of computation of ME based on MT technology of computer systems in comparison with the function of modular exponentiation of the MPIR library.

1 PROBLEM STATEMENT

The computation of directional function of ME is in the form

$$y = a^b \bmod n \ . \tag{1}$$

DOI 10.15588/1607-3274-2021-3-4

Many efficient algorithms for computing a discrete logarithm x use the determination of ME

$$a^x \equiv y(\bmod\ n)\ . \qquad (2)$$

After all a discrete logarithm is considered to be unidirectional function to ME, since it is difficult to compute it for a conditionally acceptable time.

There is an algorithm [2] for computing of ME, which uses the product of two integers modulo and the calculation of the square of an integer modulo. However, this algorithm to perform the computation of the integer power of a number modulo uses sequentially recursive operations of the product of two integers modulo and the square of an integer modulo.

One of the ways to implement the speedup of computation of ME is the parallelization of computation using MT technology in universal computer systems.

2 REVIEW OF THE LITERATURE

Many efficient algorithms use the determination of ME [3], which is characterized by considerable computational complexity and is compared with performing factorization of a number [4]. An analysis of the algorithms and their implementation for computing of ME shows the widespread use of iterative procedures among cryptographic software libraries designed for big numbers [5], which results in low efficiency in the use of available computational resources in multicores computer systems. In the book [2] an algorithm for computing a ME is described, that to use the binary form of representation of an integer number of exponent. Starting with the most significant bit of integer exponent, sequentially for each subsequent bit based on the computed values from the previous iteration is determined the result of the computation of the ME. However, iteration process impedes the organization of MT computations and does not provide high-speed determination of a ME.

The closest is the work [6], which was discussed the parallel algorithms to perform of ME and theoretical methods to obtain the optimal partition points that allows for a load balancing technique to optimize parallelization algorithms for the ME operation. The algorithms proposed in this work use the OpenMPI parallelization library as a basic tool.

Mathematical software libraries are used to implement the computation of ME. For example, the Pari/GP software library [10] contains a large set of programs for efficient computations of mathematical functions. The Pari/GP library also includes computation of the Mod(a, n)^b function for long numbers and other special numbers while using a small amount of memory during the computation process. The library uses a separate type of *t_INTMOD* to handle modulo numbers. Its peculiarity is the representation of numbers in a special form of Montgomery reduction, which simplifies the computation of division. You can use the Pari/GP library on Linux or MinGW environments. Compared to the Pari/GP library, the well-known MPIR library [7] is easier in use and can be compiled in Windows easily. A highly optimized modification of the well-known GMP or GNU Multiple Precision Arithmetic Library the MPIR library contains the function mpz_powm() to realize computation of ME. The MPIR library uses an optimized version of the ME algorithm, called the "Sliding-window method" [8], which reduce the average number of multiplication operations. However, the next iteration of the algorithm in the mpz_powm() function depends on the exponent bits that were previously analyzed in the sequential analysis of the window, which makes it impossible to divide the computations into independent flows.

3 MATERIALS AND METHODS

The parallelization of computation of ME in the form (1), based on the use of the exponent b as a binary number $b_{(k-1)}b_{(k-2)}\ldots b_2b_1b_0$, uses the application of the fundamental property of modularity [9]:

$$(a*c) \bmod n = ((a \bmod n)(c \bmod n)) \bmod\ n. \qquad (2)$$

Accordingly, the computation of the ME (1) takes the form

$$y = a^{b_{(k-1)}b_{(k-2)}\ldots b_2 b_1 b_0} \bmod\ n = (b_{k-1}a\,^\wedge 2^{k-1} \bmod\ n\ * \\ \ldots * b_1 a\,^\wedge 2^1 \bmod\ n * b_0 a\,^\wedge 2^0 \bmod\ n) \bmod\ n\ . \qquad (3)$$

In accordance with formula (3), a scheme for computing of the ME is shown on Figure 1.

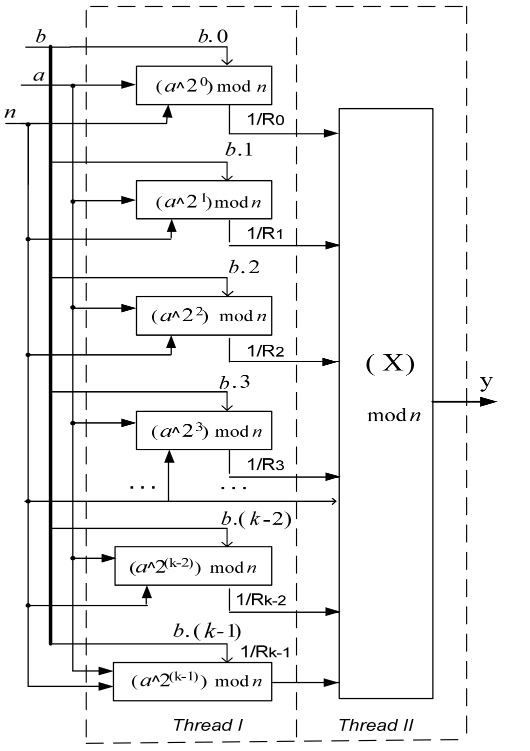

Figure 1 – The scheme for computing the ME

© Prots'ko I., Kryvinska N., Gryshchuk O., 2021
DOI 10.15588/1607-3274-2021-3-4

The scheme for computing the ME consists of the denotations:

a is the input of the base number; *n* is the input of the module;

(a^2^i) mod n are blocks of computation of the integer exponent of exponent 2^i of the number *a* by the module, $i = 0,1,2,\ldots, (k–1)$;

b is the input of an exponent with binary digits *b*.(k–1), *b*.(k–2),…,*b*.2, *b*.1, *b*.0;

(X) mod *n* is the block of multiplier under modulo *n*;

y is the output of the ME.

Consistently determined residuals of the number *a* raised to the power of 2^i under modulo *n* in blocks (a^2^i) mod *n*, for *i*=0, 1,…,*k*–1 are multiplied. In the case of presence of the exponent of the number of binary the value of zero in *i*-th binary digit (*b.i*) block (a^2^i) mod *n* is not execute the operation, otherwise is computed the multiplication of exponent 2^i of the number *a* under modulo *n*, which corresponds to the denotation at the output of the blocks (1/Ri). The determined product, in the final stage, forms in block (X)mod *n* the value *y* of the result of the computation of the ME.

For the organization of MT execution of the computation of the ME, in accordance with Figure 1, it is possible to consider the possibility of creating two threads. The *thread I* computes the numbers *a* raised to the power of 2^i under modulo *n*. The *thread II* computes the product of the results (a^2^i) mod n under modulo *n*, starting with the readiness of the first two values (a^2^i)mod *n*. Theoretical it is possible to reduce the computation time to 50% of the ME in comparison with single thread computation.

Therefore, due to the parallel execution of the computation of the ME, in accordance with Figure 1, consider the possibility of creating a thread of execution of multiplication under modulo operations and a thread by computation of the product of the residual values defined in the parallel previous thread. The only difficulty of organizing the computation with such threads is the need to synchronize the performance of threads to ensure the correct computation of the final value *y* of the ME.

4 EXPERIMENTS

The MPIR library, which is written in C and the assembler, is used to implement the algorithm for computing the ME, and provides an ability to compile its functions in Visual Studio C++. Accordingly, in the MPIR library, the *mpz_t* data type represents long numbers that are selected for the exponent *exp,* number *base,* modulo *mod* with number of binary digit from 256 to 2048 bits for testing. For implementation of the developed algorithm the MPIR library functions mpz_init_set(*mul, base*), mpz_sizeinbase(*exp*, 2), mpz_tstbit(*exp*, *i*), mpz_mul(*r*, *r*, *mul*) with long binary digit data are used.

Three versions of the algorithms are implemented as ME functions with aim to compare the performance execution of ME.

The function mod_exp(*mpz_t r*, *mpz_t base*, *mpz_t exp*, *mpz_t mod*) performs the basic iterative algorithm "Right-to-left binary exponentiation" [11]. The computation of the ME is carried out in one main thread. The result of the function is written to the variable *r*, and the computation time is fixed and averaged to output the average «single thread exponentiation» time in microseconds, which are showed in the Table 1.

The function of the MPIR library mpz_powm(*expected_result, base, exp, mod*) performs a ME, implementing a floating-window algorithm with Montgomery multiplication/reduction [12]. The result of the function is written to the *expected_result* variable and the computation time is recorded and averaged with the output of "mpz_powm average time" in microseconds, which are showed in the Table 1.

To organize MT computing according to the scheme in Figure 1, the thread functions thread_function(*bool* quit_thread*, *mpz_t*mod*) and parallel_mod_exp(*mpz_t base*, *mpz_t exp*, *mpz_t mod*) are used. Accordingly, these threads are created to perform the computation of the ME. The result of the function is written to the variable *s_thread_result*, and the computation time is fixed and averaged to output the average "parallel exponentiation" time in microseconds, which are showed in the Table 1.

The thread implemented in the function thread_function(*bool*quit_thread*, *mpz_t*mod*) computes the product under modulo over the values in the queue *s_thread_queue* to determine the ME. That is, from the queue will read the values (a^2^i)mod n which are computed by the function parallel_mod_exp(*mpz_t base*, *mpz_t exp*, *mpz_t mod*). Another thread function parallel_mod_exp(*mpz_t base*, *mpz_t exp*, *mpz_t mod*) computes the value of squaring by the module ($a^i a^i$) mod *n*, and writing the computed value under modulo to data

Table 1 – The average execution time of the functions of computing the ME based on the use of lib_mpir_gc

Bits of pseudorandom numbers / trials	256/2000	512/1000	1024/500	2048/100
Release /x64 – Core i7-3700X (AVX)				
single thread exponentiation (μs)	1276	7918	53543	383309
mpz_powm average time (μs)	824	5577	39312	279225
parallel exponentiation (μs)	1293	5970	**35182**	**244199**
Release /x64 – Core i9-7900X (AVX2/AVX512)				
single thread exponentiation (μs)	1190	7577	52091	375792
mpz_powm average time (μs)	767	5331	37907	271139
parallel exponentiation (μs)	1235	6114	**34961**	**240772**

queue *s_thread_queue*. In the loop of the function parallel_mod_exp(*mpz_t base, mpz_t exp, mpz_t mod*), binary digits of exponent *exp* are analyzed mpz_tstbit(*exp*, *i*) to determine whether to execute or not operation. That is, the result will or will not be written to the data queue *s_thread_queue* After analyzing the *b*.(k–1) binary digit in a final stage is computed the end value of the ME, which is written to the variable *s_thread_result*.

To protect the queue from simultaneous access of the threads is presented in Figure 2, where the *s_mutex* mutex with the basic lock() and unlock() methods is used. The *s_mutex* object is passed to our functions, which should use it for interacting. Before accessing the shared data queue *s_thread_queue*, the mutex is locked by the method lock(*s_mutex*) and is unlocked after the work with the shared data is completed.

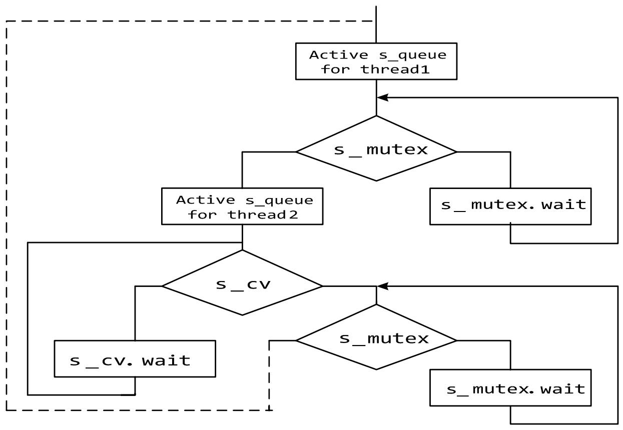

Figure 2 – The scheme of activation of threads when performing synchronization with a pair of mutex-conditional variables

The condition variable *s_cv* helps to synchronize the work of the threads. The variable *s_cv* is used for waiting of the thread of an event to occur in another thread (Figure 2). Running the s_cv.notify_one() method in the thread generates a readiness signal of the factor for another thread in the waiting state of s_cv.wait(*lock*). According-ly, even if a thread is unblocked by the *s_cv* conditional variable, it will not be able to continue immediately if another thread is already inside our critical section. In this case, the unblocked (on a conditional variable) thread changes its status from "blocked on a conditional varia-ble" (in which it was previously) to the status of "blocked on a mutex" and remains in it until the active flow is in a critical sections will not make the mutex free. The use of a combination a conditional variable and a mutex lock indicates the complexity of the synchronization the performance of threads for computing the ME.

5 RESULTS

To determine the average computation time of the ME in testing, the multiple functions called mod_exp(), mpz_powm(), parallel_mod_exp() were used. To compute the ME with a given number of trials the values of exponent *exp*, number *base* and *mod* were given by pseudo-random numbers with number of binary digit of 256, 512, 1024, 2048 bits. To reduce the total computation time on increasing the number of binary digits of long numbers the number of trials of latch-up of the computation time is decrease correspondent.

Numerical experiments were carried out on a computer system with a multi-core microprocessor with shared memory in a 64-bit Windows. Testing was performed on computer systems with an Intel Core i7-3770K that has 4 physical cores with 3.5GHz clock and an Intel Core i9-7900X processor that has 10 physical cores with 3.3GHz clock. According to hyper-threading technology, each physical core consists of two virtual ones.

The results are presented in Table 1, which contains the values of average execution time (μs microseconds) of computing the ME for pseudorandom data with a number of binary digits (256, 512, 1024, 2048bits) for correspondent the specified number of trials (2000, 1000, 500, 100).

6 DISCUSSION

The parallel algorithm in function parallel_mod_exp() for computing the ME improves (in Table 1, the best results are highlighted in bold) the computation time relative to the mpz_powm() function from the MPIR library with increasing the number of binary digits of data, starting from 1024 bits. The speedup of the computation time of the parallel algorithm with respect to the base time depends on the number of binary digits and their values in exponents. The value one in binary digits in exponent determine the operations of the multiplication by modulo in *thread I*. To computation the ME with a given number of trials, the values of the exponent exp, base number and mod were given pseudo-random numbers. Therefore, the obtained results of the average execution time do not depend on the number of ones in the binary value of the exponent.

The implementation of mpz_powm() uses a more optimal multiplication by modulo algorithm, the so-called Montgomery multiplication/reduction [12]. The modified Montgomery algorithms [13] can be used instead of operations in the *thread I* and *thread II* to speedup for computing the ME in the parallel algorithm.

The computation time of function parallel_mod_exp() with two threads improves the results (in Table 1) relative to the mod_exp() with single thread with increasing the number of binary digit of data starting from 512 bits. Thus, we can conclude that the using two threads sig-nificant improvement the execution time (Table 1) of ME of long numbers compared to single thread implementation.

Since parallelization is performed in two threads, the number of cores in computer systems does not accelerate the computation of ME. The execution time of ME on computer systems with Intel Core i7-3700X and Intel Core i9-7900X processors does not change significantly according to table 1. The use of the function parallel_mod_exp() is possible for the organization of parallel calculations of ME even larger big numbers.

CONCLUSIONS

The work compares and analyses the developed software implementation of the computation of ME and the software implementation of the function mpz_powm() of MPIR library. The computational scheme of the ME, the software implementation of the algorithm using single- and two threads for computing of ME, the run time results of the computation on multi-core microprocessors of universal computer systems have been described. As a result, the parallel organization of the computation speedups the execution of the computations of ME. MT software implementation with increasing the number of binary digits of exponent shows an improvement of computation time with regard to the MPIR function of computing ME. The execution time of the algorithms depends on the specific values of the exponent also.

The scientific novelty of obtained results lies in the implementation of parallelism using multithreading in the algorithm of computing the ME based on the use of the representation of exponent in the form of a binary number and the fundamental property of modularity.

The practical significance of the work lies in the fact that the obtained results can be successfully apply in the modern asymmetric cryptography, for efficient computation of number-theoretic transforms and other computational problems.

Prospects for further research are that the developed function using two streams can be used for the organization of multithreading computations of ME even larger big numbers.

ACKNOWLEDGEMENTS

The authors are grateful to Roman Rykmas of the team leader of Uniservice LtdC for participation in testing and discussing the results obtained by the functions of computing the ME.

The work is supported by the state budget scientific research project of Lviv National Polytechnic University “Experimental system of neural network cryptographic protection in real-time data transmission using codes imitating barker codes” (state registration number 0121U109503).

REFERENCES

1. Marouf I. Comparative Study of Efficient Modular Exponentiation Algorithms. / I. Marouf, M. M. Asad, Q. A. Al-Haija // COMPUSOFT, An international journal of advanced computer technology. – August-2017. – Vol. 6, Issue 8. – P. 2381–2392.
2. Cormen T. H. Introduction to Algorithms / [T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein]. – Cambridge, USA : MIT Press, 3rd edition, 2009. –1312 p.
3. Efficient modular exponentiation algorithms [Electronic resource]. – https://eli.thegreenplace.net/2009/03/28/ efficient-modular-exponentiation-algorithms
4. Prots’ko I. O. Computation factorization of number at chip multithreading mode / I. O. Prots’ko, O. V. Gryschuk // Radio Electronics, Computer Science, Control. – 2019. –№ 3. – P. 117–122. DOI: 10.15588/1607-3274-2019-3-13
5. Jakubski A. Review of General Exponentiation Algorithms / A. Jakubski, R. Perliński // Scientific Research of the Institute of Mathematics and Computer Science. –2011. – Vol. 2, № 10. – P. 87–98
6. Parallel modular exponentiation using load balancing without precomputation / [P. Lara, F. Borges, R. Portugal, N. Nedjah] // Journal of Computer and System Sciences. – 2012. – Vol. 78, № 2. – P. 575–582. https://doi.org/10.1016/j.jcss.2011.07.002
7. MPIR: Multiple Precision Integers and Rationals. [Electronic resource]. – Access mode: http://mpir.org/
8. Koç C.-K. Analysis of sliding window techniques for exponentiation / C.-K. Koç // Computers & Mathematics with Applications. – 1995. – № 30. – P. 17–24.
9. Childs L. N. A Concrete Introduction to Higher Algebra. Undergraduate texts in Mathematics. / L. N. Childs. – New York : Springer, Science+Business Media LLC, 3rd edition, 2009. DOI: 10.1007/978-0-387-74725-5
10. PARI/GP home. [Electronic resource]. – Access mode: http://pari.math.u-bordeaux.fr/
11. Sun D.-Z. How to compute modular exponentiation with large operators based on the right-to-left binary algorithm / Da-Zhi Sun, Zhen-Fu Cao, Yu Sun // Applied Mathematics and Computation. – 1 May 2006. – Vol. 176, Iss. 1, – P. 280–292 DOI: 10.1016/j.amc.2005.09.062
12. Montgomery P. Modular Multiplication without Trial Division / P. Montgomery // Mathematics of Computation. –1985. – Vol. 44, №170. – P. 519–521.
13. Algorithm design and theoretical analysis of a novel CMM modular exponentiation algorithm for large integers / A. Rezai, P. Keshavarzi // RAIRO – Theoretical Informatics and Applications. – July-September 2015. – Vol. 49, Num. 3. – P. 255 – 268. DOI: 10.1051/ita/2015007

Received 25.06.2021.
Accepted 15.08.2021.

УДК 004.421

ЧАСОВИЙ АНАЛІЗ ОБЧИСЛЕННЯ МОДУЛЬНОЇ ЕКСПОНЕНТИ

Процько І. – д-р техн. наук, доцент, кафедра автоматизованих систем управління, Національний університет «Львівська політехніка», Львів, Україна.

Кривінська Н. – д-р техн. наук, професор, кафедра інформаційних систем, Університет Коменіуса в Братиславі, Братислава, Словаччина.

Грищук О. – розробник програмного забезпечення, ТОВ «СофтСерв», Львів, Україна.

АНОТАЦІЯ

Актуальність. Постановка проблеми швидкого обчислення модульної експоненти вимагає розробки ефективних алгоритмічних методів з використанням новітніх інформаційних технологій. Швидкі обчислення модульної експоненти є надзвичайно необхідними для ефективних обчислень у теоретико-числових перетвореннях, для забезпечення високої стійкості криптоінформаційних даних та у багатьох інших додатках.

Мета – аналіз швидкості виконання функцій в програмному забезпеченні для обчислення модульної експоненти розроблених програм на основі паралельної організації обчислень з використанням багатопоточності.

DOI 10.15588/1607-3274-2021-3-4

Метод. Обчислення модульної експоненти реалізується за допомогою алгоритму 2^k-го ковзаючого вікна, де *k* вибирається відповідно до розміру показника степеня. Паралелізація обчислень полягає у використанні обчислення залишків чисел, піднесених до степеня 2^i за модулем, та їх подальшого паралельного множення за модулем.

Результати. Здійснено порівняння часу виконання трьох варіантів функцій для обчислення модульної експоненти. В алгоритмі паралельної організації обчислень з використанням багатопоточності забезпечується більш швидке обчислення обчислення модульної експоненти для значень показника степеня, що перевищує 1K двійкових цифр, порівняно з функцією обчислення модульної експоненти в бібліотеці MPIR. Бібліотека MPIR з цілочисельним типом даних з числом двійкових цифр від 256 до 2048 біт використовується для розробки алгоритму обчислення обчислення модульної експоненти з використанням багатопоточності.

Висновки. У роботі розглянуто та проаналізовано розроблену програмну реалізацію обчислення модульної експоненти на універсальних комп'ютерних системах. Одним із способів реалізації прискорення обчислень обчислення модульної експоненти є розробка алгоритмів, які можуть використовувати багатопотокову технологію на багатоядерних мікропроцесорах. Багатопотокова програмна реалізація обчислення модульної експоненти зі збільшенням від 1024 числа двійкових розрядів показника степеня показує поліпшення часу обчислення у порівнянні з функцією обчислення модульної експоненти бібліотеки MPIR.

КЛЮЧОВІ СЛОВА: модульна експонента, паралельні обчислення, багатопотоковість, великі числа.

УДК 004.421

ВРЕМЕННОЙ АНАЛИЗ ВЫЧИСЛЕНИЯ МОДУЛЬНОЙ ЭКСПОНЕНТЫ

Процько И. – д-р техн. наук, доцент кафедры автоматизированных систем управления, Национальный университет «Львивська политэхника», Львов, Украина.

Кривинська Н. – д-р техн. наук, профессор, кафедра информационных систем, Університет Комениуса в Братиславе, Братислава, Словакия.

Грищук О. – разработчик программного обеспечения, ООО «СофтСерв», Львов, Украина.

АННОТАЦИЯ

Актуальность. Решение задачи быстрого вычисления модульной экспоненты возведения требует разработки эффективных алгоритмических методов с использованием новейших информационных технологий. Быстрые вычисления модульной экспоненты возведения чрезвычайно необходимы для эффективных вычислений в теоретико-численных преобразованиях, для обеспечения высокой криптостойкости информационных данных и во многих других приложениях.

Цель – анализ исполнения функций в программном обеспечении для вычисления модульного возведения в степень разработанных программ на основе параллельной организации вычислений с использованием многопоточности.

Метод. Модульное возведение в степень реализуется с использованием алгоритма 2^k-арного скользящего окна, где *k* выбирается в соответствии с размером показателя степени. Распараллеливание вычислений заключается в использовании вычисления остатков чисел, возведенных в степень 2^i по модулю, и их дальнейших параллельных умножений по модулю.

Результаты. Проведено сравнение времени выполнения трех вариантов функций для вычисления модульной экспоненты. В алгоритме параллельной организации вычислений с использованием многопоточности обеспечивается более быстрое вычисление модульной экспоненты для значений показателя больше 1K двоичных разрядов по сравнению с функцией модульного возведения в степень библиотеки MPIR. Библиотека MPIR с целочисленным типом данных с количеством двоичных разрядов от 256 до 2048 бит используется для разработки алгоритма вычисления модульного возведения в степень с использованием многопоточности.

Выводы. В работе рассмотрена и проанализирована разработанная программная реализация вычисления модульной экспоненты на универсальных компьютерных системах. Одним из способов реализации ускорения вычислений модульной экспоненты является разработка алгоритмов, которые могут использовать технологию многопоточности на многоядерных микропроцессорах. Многопоточная программная реализация модульной экспоненты с увеличением с 1024 числа двоичных разрядов экспоненты показывает улучшение времени вычислений по сравнению с функцией модульного возведения в степень библиотеки MPIR.

КЛЮЧЕВЫЕ СЛОВА: модульная экспонента, параллельные вычисления, многопоточность, большие числа.

ЛІТЕРАТУРА / ЛИТЕРАТУРА

1. Marouf I. Comparative Study of Efficient Modular Exponentiation Algorithms. / I. Marouf, M. M. Asad, Q. A. Al-Haija // COMPUSOFT, An international journal of advanced computer technology. – August-2017. – Vol. 6, Iss. 8. – P. 2381–2392.
2. Cormen T. H. Introduction to Algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. – Cambridge, USA : MIT Press, 3rd edition, 2009. –1312 p.
3. Efficient modular exponentiation algorithms [Electronic resource]. – https://eli.thegreenplace.net/2009/03/28/ efficient-modular-exponentiation-algorithms
4. Процько І. О. Обчислення факторизації числа в мультипотоковому режимі на кристалі / І. О. Процько, О. В. Грищук // Радіоелектроніка, інформатика, управління – 2019. – № 3. – P. 117–122. DOI: 10.15588/1607-3274-2019-3-13
5. Jakubski A. Review of General Exponentiation Algorithms / A. Jakubski, R. Perliński // Scientific Research of the Institute of Mathematics and Computer Science. –2011. Vol. 2, №10. – P. 87–98
6. Lara P. Parallel modular exponentiation using load balancing without precomputation / P. Lara, F. Borges, R. Portugal, N. Nedjah // Journal of Computer and System Sciences. – 2012; Vol.78, № 2. – P. 575–582. https://doi.org/10.1016/j.jcss.2011.07.002
7. MPIR: Multiple Precision Integers and Rationals. [Electronic resource]. – Access mode: http://mpir.org/
8. Koç C.-K. Analysis of sliding window techniques for exponentiation / C.-K. Koç // Computers & Mathematics with Applications. – 1995. – № 30. – P. 17–24.
9. Childs L. N. A Concrete Introduction to Higher Algebra. Undergraduate texts in Mathematics. / L. N. Childs. – New York : Springer, Science+Business Media LLC, 3rd edition, 2009. DOI: 10.1007/978-0-387-74725-5
10. PARI/GP home. [Electronic resource]. – Access mode: http://pari.math.u-bordeaux.fr/
11. Sun D.-Z. How to compute modular exponentiation with large operators based on the right-to-left binary algorithm / Da-Zhi Sun, Zhen-Fu Cao, Yu Sun // Applied Mathematics and Computation. – 1 May 2006. – Vol. 176, Iss. 1, – P. 280–292 DOI: 10.1016/j.amc.2005.09.062
12. Montgomery P. Modular Multiplication without Trial Division / P. Montgomery // Mathematics of Computation. –1985. Vol. 44, №170, – P. 519–521.
13. Algorithm design and theoretical analysis of a novel CMM modular exponentiation algorithm for large integers / A. Rezai, P. Keshavarzi // RAIRO – Theoretical Informatics and Applications. – July-September 2015. – Vol. 49, Num. 3, P. 255 – 268. DOI: 10.1051/ita/2015007

DOI 10.15588/1607-3274-2021-3-4