ние программы позволит существенно увеличить скорость и эффективность обработки заказов. В результате сокращение запасов составит 173,0 тыс. руб., сокращение издержек по заказу – 5,6 тыс. руб., рост доходов благодаря сокращению дефицитов – 113,2 тыс. руб., прирост оборачиваемости – 8,4 дней, а рентабельности продаж – 3,6 п.п. Общий экономический эффект составит 264,8 тыс. руб.

Список использованной литературы

1.Обоснование направлений повышения эффективности функционирования предприятия на основе инновационной деятельности / Д. А. Глушанина [и др.]; науч. рук. О. Г. Довыдова // НИРС БГЭУ: сборник научных статей. Вып. 7 / [редкол: А.А. Быков (пред. и др.]; М-во образования Респ. Беларусь, Белорус. гос. экон. ун-т. – Минск: БГЭУ, 2018. – С. 212-216.

УДК 004.056.53

DESIGN OF A SPECIAL-PURPOSE MULTIPROCESSOR ARRAY FOR DATA PROTECTION AND DIGITAL SIGNATURE

Tiunchick A.A., PhD, assistant professor
Belarusian State Agrarian Technical University, Minsk

Ключевые слова: защита данных, ЭЦП, многопроцессорные устройства, модульное умножение Монтгомери.

Key words: data protection, digital signature, multiprocessor array, Montgomery modular multiplication.

Аннотация: Представлено специализированное многопроцессорное устройство для модульного умножения и возведения в степень по алгоритму Монтгомери. Устройство может быть реализовано на основе технологии СБИС или ПЛМ.

Summary**:** A special-purpose multiprocessor array for Montgomery modular multiplication and exponentiation is presented/ The array can be implemented via VLSI or FPGA technology/

Public-key cryptography plays an important role in digitalization of the economy. Modular multiplication is the basis of many encryption and digital signature algorithms. Efficient implementation of the operation is important to process larger numbers and increase cryptographic strength. Besides that, ehere exists a great demand for developing special-purpose hardware to speed up the computations.

A modular exponentiation operation $M^e \bmod n$ cannot be implemented in a naive fashion by first exponentiating M^e and then performing reduction modulo *n*, since the intermediate result M^e contains too many digits. Hence, the intermediate results of the exponentiation are to be reduced modulo *n* at each step. The straightforward reduction modulo *n* involves a number of arithmetic operations (division, subtraction, etc.), and is very time consuming. Therefore, special algorithms for modular operations are to be used.

P. L. Montgomery [1] proposed an algorithm for modular multiplication $AB \bmod m$ without trial division. Let *A*, *B* be elements of Z_m, where Z_m is the set of integers between 0 and $m-1$. Let *h* be an integer coprime to *m*, and $h > m$. *Montgomery modular multiplication (MM)* is an operation

$$A \overset{h,m}{\otimes} B = A \cdot B \cdot h^{-1} \bmod m. \qquad (1)$$

Implementation of this operation is much easier than a normal reduction modulo *m*; and is based on some facts from number theory. The use of MM does not result in the desirable speed-up immediately. To compute $AB \bmod m$, a computation of MM is to be performed twice:

$$C = Ah, mB = A \cdot B \cdot h^{-1} \bmod m, \text{ and}$$

$$Ch, m(h^2 \bmod m) = ABh^{-1} \cdot h^2 \cdot h^{-1} \bmod m = AB \bmod m,$$

where $h^2 \bmod m$ is computed in advance. The advantage of using two Montgomery multiplications instead of one operation of plain modular multiplication is uncertain.

An efficient way to compute $AB \bmod m$ using MM is by exploiting special representations of *A* and *B*. Let $\overline{\overline{X}}$ be called an *image* of *X* if $\overline{\overline{X}} = X \cdot h \bmod m$, $h > m$. If h and m are relatively prime, then there exists a one-to-one correspondence between *X* and $\overline{\overline{X}}$. MM of $\overline{\overline{A}}$ and $\overline{\overline{B}}$ is isomorphic to the modular multiplication of *A* and *B*. Indeed,

$$\overline{\overline{A}} \overset{h,m}{\otimes} \overline{\overline{B}} = (Ah \bmod m) \overset{h,m}{\otimes} (Bh \bmod m) = (AB)h \bmod m = \overline{\overline{A \cdot B}}.$$

The reduction of *X* to $\overline{\overline{X}}$ and vice versa can be carried out on the basis of MM:

$$X \overset{h,m}{\otimes} (h^2 \bmod m) = X \cdot h^2 h^{-1} \bmod m = \overline{\overline{X}}, \qquad (2)$$

$$\overline{\overline{X}} \overset{h,m}{\otimes} 1 = X \cdot h \bmod m \cdot 1 \cdot h^{-1} \bmod m = X. \qquad (3)$$

By virtue of the isomorphism of modular multiplication and MM, the use of the images is very convenient for exponentiation. Let $(\overset{h,m}{\otimes} X)^n$ denote $(n-1)$ MMs of X by itself. To compute $Y = X^n \bmod m$, we should perform three steps: first, convert X to $\overline{X}$ by (2); next, realize $\overline{Y} = (\overset{h,m}{\otimes} \overline{X})^n = \overline{X^n}$; and finally, convert $\overline{Y}$ to Y by (3).

Several algorithms suitable for hardware implementation of MM are known. In this paper, the design of a multiprocessor array is based on the algorithm described and analysed in [2]. Let numbers A, B and m be written with radix 2:

$$A = \sum_{i=0}^{N-1} a_i \cdot 2^i, \qquad B = \sum_{i=0}^{M} b_i \cdot 2^i, \qquad m = \sum_{i=0}^{M-1} m_i \cdot 2^i,$$

where a_i, b_i, $m_i \in \{0;1\}$, N and M are the numbers of digits in A and m, respectively. B satisfies condition $B < 2m$, and has at most $M+1$ digits. m is odd (to be coprime to the radix 2). Extend a definition of A with an extra zero digit $a_N = 0$. The algorithm for MM is given below.

$$\begin{aligned} &s : -0; \\ &\textbf{For } i := 0 \textbf{ to } N \textbf{ do} \\ &\textbf{Begin} \\ &\quad u_i := \big((s_0 + a_i * b_0) * w\big) \bmod r \\ &\quad s := (s + a_i * B + u_i * m) \operatorname{div} r \\ &\textbf{End} \end{aligned} \tag{4}$$

Initial condition $B < 2m$ ensures that intermediate and final values of s are bounded by $3m$. The use of an iteration with $a_N = 0$ ensures that the final value $s < 2m$. Hence, this value can be used for B input in a subsequent multiplication. Since 2 and m are relatively prime, we can precompute value $w = (2 - m_0)^{-1} \bmod 2$. An implementation of the operations div 2 and mod 2 is trivial (shifting and inspecting the lowest digit, respectively). Algorithm (4) returns either $s = A \cdot B \cdot 2^{-n-1} \bmod m$ or $s + m$ (because $s < 2m$). In any case, this extra m has no effect on subsequent arithmetics modulo m. It should be noted, that the number of iterations in (4) affects h. In our case, (4) presents the implementation of (1) with $h = 2^{N+1}$.

To design multiprocessor array, first we construct a data dependency graph (also referred as DG) for Algorithm (4). For N- and M-digit integers A and B, a graph consists of $N+2$ rows and $M+1$ columns. The i-th row represents the i-th iteration of (4). Arrows are associated with digits transferred along

indicated directions. Each vertex $v(j,i)$, $i \in \{0,...,N\}, j \in \{0,...,M\}$ is associated with the operation

$$s_{j-1}^{(i+1)} + 2 \cdot c_{out} := s_j^{(i)} + a_i \cdot b_j + u_i \cdot m_j + c_{in},$$

where $s_j^{(i)}$ denotes the j-th digit of the i-th partial product of s, c_{out} and c_{in} are the output and input carries. Rightmost starred vertices, i.e., vertices marked with «…», perform calculations of $u_i := ((s_0 + a_i * b_0) * w) \bmod 2$ besides an ordinary operation. Using standard notation, the vertex operations can be specified in terms of inputs/outputs as follows:

$$\begin{gathered} s_{out} + 2 \cdot c_{out} := s_{in} + a_{in} \cdot b_{in} + u_{in} \cdot m_{in} + c_{in}, \\ a_{out} := a_{in}, \qquad b_{out} := b_{in}, \\ u_{out} := u_{in}, \qquad m_{out} := m_{in}, \end{gathered} \tag{5}$$

for plain vertices, and

$$\begin{gathered} u_{out} := (s_{in} + a_{in} \cdot b_{in}) \cdot w_{in}, \\ c_{out} := maj_2(s_{in}, a_{in} \cdot b_{in}, u_{in} \cdot m_{in}), \\ a_{out} := a_{in}, \qquad b_{out} := b_{in}, \qquad m_{out} := m_{in}, \end{gathered} \tag{6}$$

for starred vertices, where $maj_2(s_{in}, a_{in} \cdot b_{in}, u_{in} \cdot m_{in})$ is 1 if at least two out of three entries are 1s; otherwise it is 0.

If instead of digits A we input digits B both, at the topmost and rightmost vertices of DG, then the graph model represents a calculation of

$$B \overset{h,m}{\otimes} B = (\overset{h,m}{\otimes} B)^2,$$

called M- *squaring*. To represent the computation of $(\overset{h,m}{\otimes} B)^3$, two graphs can be joined in a single graph by connecting s_j-outputs of the first graph with b_j-inputs of the next (identical) graph, in which rightmost inputs a_i get digits of B as before. To compute $(h,mB)^n$, we will need $n-1$ joined graphs. The resulting graph model consists of vertices located in a rectangular domain $V_1 = \{v(i,j) \mid 0 \le i \le n \times (M+2) - 1, 0 \le j \le M+1\}$. The graph is almost homogeneous, with exceptional starred vertices in the rightmost column.

However, a faster way to compute $B^n \bmod m$ is by reducing the computation to a sequence of modular squares and multiplications. Let $[n_0 \ldots n_k]$ be a binary representation of n, i.e., $n = n_0 + 2n_1 + \cdots + 2^k n_k$, $n_j \in \{0;1\}$, $k = \lfloor \log_2 n \rfloor$, $n_k = 1$. Let β denote a partial product. We start out with $\beta = B$ and run from n_{k-1}

to n_0 as follows: if $n_j = 0$, then $\beta := \beta^2$; if $n_j = 1$, then $\beta := \beta^2 * B$. Thus, we need at most $2k$ operations to compute B^n. This algorithm has an advantage over a low-to-high binary method of exponentiation since, when implemented in hardware, it requires only one set of storage registers for intermediate results as opposed to two for a low-to-high method.

To perform M-squaring the dependency graph for M-multiplication can be modified in such a way that all the b_j's inputs enter the graph only via the top-row vertices. This eliminates rightmost a_i-inputs entirely. To deliver all b_js to the rightmost vertices, we have to pump them through the graph in a direction determined by vector $(1,-1)$. To do it, additional arcs x_j's for propagation of b_j's digits have to be added to DG. Vertex operations are to be slightly modified to provide propagation of these digits: each non-starred vertex just transmits its *x*-input data to an *x*-output, while when arriving at the rightmost vertices, these data are "reflected" and propagated to the left as if they were ordinary a_i's input data. It is known that the output value $s < 2m$. Hence, we need at most $M+1$ rows for the "reflected" factor and an additional row for the extension with an extra zero digit. Using standard notation, the vertex operations for M-squaring can be specified in terms of inputs/outputs as follows:

$$\begin{gathered} s_{out} + 2 \cdot c_{out} := s_{in} + a_{in} \cdot b_{in} + u_{in} \cdot m_{in} + c_{in}, \\ a_{out} := a_{in}, \qquad x_{out} := x_{in}, \\ b_{out} := b_{in}, \qquad u_{out} := u_{in}, \qquad m_{out} := m_{in}, \end{gathered} \tag{7}$$

for plain vertices, and for starred vertices the operation is:

$$\begin{gathered} u := (s_{in} + x_{in} \cdot b_{in}) \cdot w_{in}, \\ s_{out} + 2 \cdot c_{out} := s_{in} + x_{in} \cdot b_{in} + u \cdot m_{in}, \\ c_{out} := maj_2(s_{in}, x_{in} \cdot b_{in}, u_{in} \cdot m_{in}), \\ u_{out} := u, \qquad a_{out} := x_{in}, \qquad b_{out} := b_{in}, \qquad m_{out} := m_{in}. \end{gathered} \tag{8}$$

DG for an exponentiation as a whole is constructed as a composition of graphs for M-multiplication and M-squaring by joining outputs of one graph with corresponding inputs of the consecutive graph. There are at most $2k$ graphs altogether, and the precise number of required graphs for M-multiplication and M-squaring and the order in which they occur in the composition is fully determined only by the binary representation of *n*. The vertices of the resulting graph constitute a rectangular domain $V_2 = \{v(i,j) \mid 0 \le i \le 2k \times (M+2), 0 \le j \le M+1\}$, where $k = \lfloor \log_2 n \rfloor$.

The next stage of the design is a space-time mapping of domain V_2 onto a one-dimensional domain of processing elements (PE). Spatial mapping is determined by a linear operator with matrix $P = (1\ 0)$, which maps an indefinitely long composition of the cohered DGs onto a linear processor array with $M+1$ processing elements: each column of vertices is mapped onto one PE. Hence, each PE has to be able to operate in two modes. To control the operation modes, a sequence of one-bit control signals τ is fed into the rightmost PE and propagated through the array. If $\tau = 0$ the PE implements an operation for M-multiplication, if $\tau = 1$, for M-squaring. The order in which control signals are input is determined by the binary representation of n. A timing function that provides a correct order of operations is $t(v) = 2i + j$. The total running time is thus at most $(4\lfloor \log_2 n \rfloor + 1)M + 8\lfloor \log_2 n \rfloor$ time units.

References

1. Montgomery, P.L. Modular multiplication without trial division // P.L. Montgomery Mathematics of Computations, 1985 (44) 519-521.
2. Walter, C.D. Systolic Modular Multiplication // IEEE Trans. on Comput., vol. 42 (1993), No. 3, pp. 376-378.
3. Tiountchik, A.A. Systolic modular exponentiation via Montgomery algorithm. // J. Electronics Letters. 1998. Vol. 34, No 9. pp. 874–875.

УДК 631.145

ПОКАЗАТЕЛИ УРОВНЯ МЕХАНИЗАЦИИ ПРОИЗВОДСТВА В ОТРАСЛЯХ АГРОПРОМЫШЛЕННОГО КОМПЛЕКСА

Цыганов В.А., к.ф.-м.н., доцент
УО «Белорусский государственный аграрный технический университет», г. Минск

Ключевые слова: агропромышленный комплекс, механизация, коэффициент механизации работ, коэффициент механизации труда
Key words: agro-industrial complex, mechanization, work mechanization coefficient, labor mechanization coefficient

Аннотация: в статье дана краткая характеристика этапов замены ручного труда машинным в отраслях агропромышленного комплекса, приведены соотношения для расчета коэффициентов механизации работ и труда.