A Modular Multiplication Algorithm with Triangle Additions

Naofumi TAKAGI
Department of Information Science
Kyoto University
Kyoto 606-01, Japan

Abstract

A new algorithm for multiple-precision modular multiplication is proposed. In the algorithm, first the upper half triangle of the whole partial products is added up, and then the residue of the sum is calculated. Next, the sum of the lower half triangle of the whole partial products is added to the residue, and then the residue of the total amount is calculated. A new efficient procedure for residue calculation is also proposed, which accelerates the algorithm. Since it is both fast and uses a small amount of main memory, the algorithm is efficient for implementation on small computers, such as card computers, and is useful for application of a public-key cryptosystem to such computers.

1 Introduction

A sharp increase in the use of card computers, i.e., so-called smart cards, is expected. These computers will be the basis of various services, such as prepaid cards, banking cards, credit cards and so on. The success of the public-key cryptosystems, such as RSA [1] and ElGamal [2], could mean that these methods can also be applied to the card computers to solve the issue of security. In encryption and decryption of such public-key cryptosystems, modular multiplication with a large modulus (longer than 500 bits) is the main operation. Since card computers have only a small amount of main memory, the key to implementing a public-key cryptosystem on these computers would be to develop an algorithm for multiple-precision modular multiplication which is both fast and uses a small amount of main memory.

Various algorithms have been proposed for modular multiplication. Most of them are classified into two methods, i.e., "division-after-multiplication" and "division-during-multiplication" method. In an n-word modular multiplication by the former method, an ordinary n-word by n-word multiplication is carried out first and then a 2n-word by n-word division for residue calculation is performed. In the latter method, each subtraction step for the division for residue calculation is embedded in the repeated multiply-addition [3]. The former requires more amount of memory space than the latter does [4]. Namely, the former requires about 2n-word memory space for storing the intermediate results, while the latter requires about only (n+1)-word memory space for it. On the other hand, the latter requires more operations for residue calculation than the former does [5]. Namely, the latter requires a possible n-word subtraction, as well as an (n+1)-word by n-word division, for each residue calculation step, while the former does not require such subtraction.

In this paper, we propose a new fast algorithm for multiple-precision modular multiplication which uses a small amount of main memory. It is a completely new algorithm and belongs to neither the division-after-multiplication nor the division-during-multiplication method. In the algorithm, we first add up the upper half triangle of the whole partial products, and then calculate the residue of the sum. Next, we add the sum of the lower half triangle of the whole partial products to the residue, and then calculate the residue of the total amount. The algorithm requires about the same amount of memory space as the division-during-multiplication method does, and at the same time, it requires about the same number of operations as the division-after-multiplication method does. Furthermore, we propose a new efficient procedure for residue calculation, which accelerates the algorithm. It calculates the residue of an (n+1)-word number P with respect to modulo N. We keep the residue in the range of $[-N, N)$ and determine the quotient from only the most significant two words of P and those of N.

In the next section, we give assumptions on our computation model and describe the notations to be used through the paper. We propose a new algorithm

1063-6889/93 $03.00 © 1993 IEEE

for multiple-precision modular multiplication, in Section 3. We also propose an efficient procedure for residue calculation which accelerates the algorithm in Section 4, and prove its correctness in Appendix. Section 5 is a conclusion.

2 Assumptions and Notations

We consider a modular multiplication of $A \times B$ mod N, where the modulus N is an n-word number and the multiplicand A and the multiplier B are also n-word numbers satisfying $0 \leq A, B < N$. We assume the radix of each word is r. For example, when each word consists of 8 bits, $r = 256$. We assume that $n \leq r$. The i-th word ($i = 0, 1, ..., n-1$) of N is denoted by N_i. Namely, $N = \sum_{i=0}^{n-1} N_i \cdot r^i$. Similarly, $A = \sum_{i=0}^{n-1} A_i \cdot r^i$ and $B = \sum_{i=0}^{n-1} B_i \cdot r^i$.

We assume that our computer has the following operations as in [6].

1. Addition or subtraction of two unsigned single-word operands with a carry, giving the unsigned single-word sum and the carry.
2. Complementation of a single-word operand, giving the one's complement of the operand. $\bar{x}$ denotes the one's complement of x. ($\bar{x} = r - x - 1$.)
3. Multiplication of two unsigned single-word operands, giving the unsigned double-word product. The upper and the lower word of the product of x and y are denoted by $(x \cdot y)_{high}$ and $(x \cdot y)_{low}$, respectively.
4. Division of an unsigned double-word dividend by an unsigned single-word divisor, giving the unsigned single-word quotient and the unsigned single-word remainder. The dividend is assumed to be smaller than r times the divisor. We refer to the quotient and the remainder of x/y as $DIV(x, y)$ and $REM(x, y)$, respectively.
5. Comparison of two single-word numbers.

When x and y are single-word numbers, $x \| y$ is the concatenation of x and y, i.e., the double-word number $x \cdot r + y$.

3 A Multiple-Precision Modular Multiplication Algorithm

We consider a modular multiplication of $A \times B$ mod N. We first add up the upper half triangle of the whole partial products, i.e., $(A_i \cdot r^i) \cdot (B_j \cdot r^j)$'s such that $i + j \geq n - 1$, and then calculate the residue of the sum. Next, we add the sum of the lower half triangle, i.e., $(A_i \cdot r^i) \cdot (B_j \cdot r^j)$'s such that $i + j < n - 1$, to the residue, and then calculate the residue of the total amount. The algorithm is based on the fact that $A \times B \bmod N = \sum_{i,j} A_i \cdot B_j \cdot r^{i+j} \bmod N = ((\sum_{i+j \geq n-1} A_i \cdot B_j \cdot r^{i+j}) \bmod N + \sum_{i+j<n-1} A_i \cdot B_j \cdot r^{i+j}) \bmod N$.

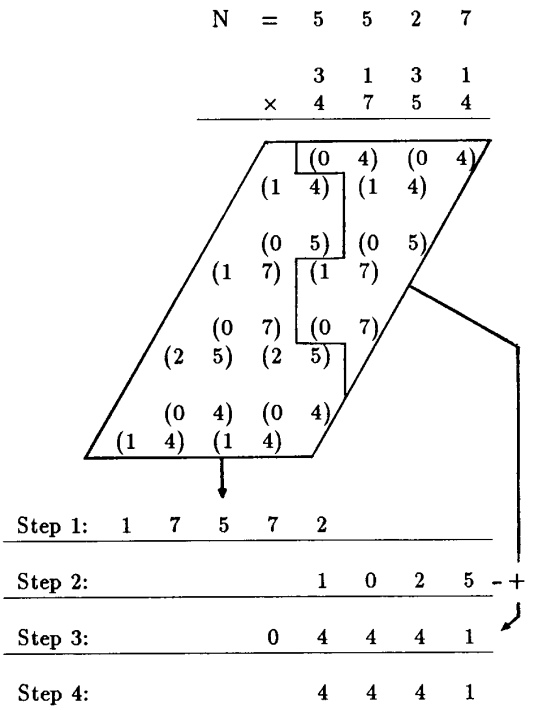

Figure 1: An example of calculation by [MODMUL] (Numbers are in the octal representation.)

The algorithm is as follows.

Algorithm [MODMUL]

Step 1: $P := \sum_{i+j \geq n-1} A_i \cdot B_j \cdot r^{i+j-n+1}$;
Step 2: $P := (P \cdot r^{n-1}) \bmod N$;
Step 3: $P := P + \sum_{i+j<n-1} A_i \cdot B_j \cdot r^{i+j}$;
Step 4: $P := P \bmod N$; □

Fig. 1 shows an example of a multiple-precision modular multiplication, i.e., $[3131]_8 \times [4754]_8$ mod $[5527]_8$, by Algorithm [MODMUL]. ($r = 8$ and $n = 4$. $[\cdots]_8$ is an octal number.)

In Step 2, we calculate $(P \cdot r^{n-1}) \bmod N$ by iterative calculations of the residue of an (n+1)-word number with respect to modulo N. We will propose an efficient

procedure for this calculation in the next section.

Through Algorithm [MODMUL], we require only (n+1)-word memory space for storing the intermediate result. This is about the same as that required by the division-during-multiplication method and is much smaller than that required by the division-after-multiplication method.

In Step 1, we require $n(n+1)/2$ single-word multiplications for generating double-word partial products, and about n^2+2n-2 single-word additions for adding them up in an adequate order. (Note that we may use only n+1 words.) In Step 3, we require $(n-1)n/2$ single-word multiplications and about n^2+2n-5 single-word additions. The number of operations required in Steps 1 and 3 in total is about the same as that required for an n-word by n-word multiplication.

The number of operations required in Steps 2 and 4 in total is about the same as that required for a 2n-word by n-word division.

Thus, the number of operations required by [MODMUL] is about the same as that required for the division-after-multiplication method, and is fewer than that required for the division-during-multiplication method.

We reduce the number of operations required in Steps 2 and 4, using an efficient residue calculation procedure to be proposed in the next section.

4 A Residue Calculation Procedure

We propose a new efficient procedure for residue calculation. The procedure calculates $P \bmod N$, where N is an n-word number and P is an (n+1)-word number satisfying $-N \cdot r \leq P < N \cdot r$. We assume $r^n/2 \leq N < r^n$ as in [6] and [7]. This assumption is acceptable in public-key cryptosystems [7]. We assume $r \geq 8$ and $r \geq 2n$. This assumption is also acceptable. For example, when the modulus is 512-bit and $r = 256$, $n = 64$ and this assumption is satisfied.

We represent P as an (n+1)-word number in two's complement form with a sign ps. When ps is 0, P is positive and otherwise (i.e., when ps is 1), P is negative. Namely, $P = -ps \cdot r^{n+1} + \sum_{i=0}^{n} P_i \cdot r^i$.

We refer to the result of the residue calculation procedure as $MOD(P,N)$. $MOD(P,N)$ satisfies $MOD(P,N) \equiv P \pmod N$ and $-N \leq MOD(P,N) < N$. Namely, we keep the residue in the range of $[-N, N)$. This is one of the key points to achieving the efficient calculation.

The procedure is as follows.

Function $MOD(P,N)$
if $ps = 0$ then do begin
  if $P_n = N_{n-1}$ then $q := r-1$
  else do begin
    $\hat{q} := DIV(P_n \| P_{n-1}, N_{n-1})$;
    $rem := REM(P_n \| P_{n-1}, N_{n-1})$;
    if $(\hat{q} \cdot N_{n-2})_{high} - rem > \lfloor N_{n-1}/2 \rfloor$
      then $q := \hat{q} - 1$ else $q := \hat{q}$;
  end;
end;
else do begin
  if $\bar{P}_n = N_{n-1}$ then $q := r-1$
  else do begin
    $\hat{q} := DIV(\bar{P}_n \| \bar{P}_{n-1}, N_{n-1})$;
    $rem := REM(\bar{P}_n \| \bar{P}_{n-1}, N_{n-1})$;
    if $(\hat{q} \cdot N_{n-2})_{high} - rem > \lfloor N_{n-1}/2 \rfloor$
      then $q := \hat{q} - 1$ else $q := \hat{q}$;
  end;
end;
return $P - (-1)^{ps} \cdot q \cdot N$; □

We determine q from only the most significant two words of P and those of N. The function returns an n-word number with a sign. The n-word is the lower n-word of the result of the final (n+1)-word addition/subtraction. We let the sign be 0 if the n-th word (the most significant word) of the result is 0 and be 1 otherwise (i.e., if the n-th word is $r-1$).

We will prove that $MOD(P,N) \equiv P \pmod N$ and $-N \leq MOD(P,N) < N$ hold, in Appendix.

As an example, let us consider calculation of $MOD([(0)17572]_8, [5527]_8)$, where $r = 8$ and $n = 4$. First $\hat{q} = DIV([17]_8, [5]_8) = [3]_8$ and $rem = REM([17]_8, [5]_8) = [0]_8$ are calculated. Since $([3]_8 \cdot [5]_8)_{high} - [0]_8 \leq [2]_8$, $q = \hat{q} = [3]_8$. Hence, $[(0)17572]_8 - [3]_8 \cdot [5527]_8 = [(1)6565]_8$ is calculated and $[(1)6565]_8$ is returned. ((0) or (1) in an octal number denotes its sign.)

The function can be performed by a couple of operations for determining q, a single-word by n-word multiplication and an (n+1)-word addition/subtraction. We interleave the multiplication with the addition/subtraction. Note that we require only one (n+1)-word addition/subtraction, but do not require any additional n-word addition which is required in the conventional multiple-precision division algorithm [6].

We can perform Step 2 of [MODMUL] using this procedure iteratively, as follows.

Step 2(a): $P := MOD(P,N)$;
Step 2(b): for $k := n-2$ down to 0 do
  $P := MOD(P \cdot r, N)$; □

			N	=	5	5	2	7
	1	7	5	7	2			
(a):	(1)	6	5	6	5			
		(0)	1	1	2	6		
(b):			(0)	3	5	3	1	
				(0)	1	0	2	5

Figure 2: An example of calculation in Step 2 of [MODMUL] by Function $MOD(P,N)$ (Numbers are in the octal representation.)

Since $0 \leq \sum_{i+j \geq n-1} A_i \cdot B_j \cdot r^{i+j-n+1} < N^2 \cdot r^{-n+1} < N \cdot r$, the P obtained by Step 1 of [MODMUL] satisfies $0 \leq P < N \cdot r$. Hence, we can apply $MOD(P,N)$ to this P in Step 2(a). Since $-N \leq MOD(P,N) < N$, we can apply the procedure iteratively in Step 2(b).

Fig. 2 shows an example of a residue calculation in Step 2 of [MODMUL], i.e., $([17572]_8 \cdot 8^3) \bmod [5527]_8$, by the iterative use of Function $MOD(P,N)$.

We can perform Step 4 of [MODMUL] by using Function $MOD(P,N)$, as follows.

Step 4(a): $P := MOD(P,N)$;
Step 4(b): if $P < 0$ then $P := P + N$; □

The P obtained by Step 2 satisfies $-N \leq P < N$. Since $0 \leq \sum_{i+j<n-1} A_i \cdot B_j \cdot r^{i+j} < (n-1) \cdot r^n$ and we have assumed $N \geq r^n/2$ and $r \geq 2n$, the P obtained by Step 3 satisfies $-N \leq P < N \cdot r$. Hence, we can apply $MOD(P,N)$ to this P in Step 4(a). (If we relax the restriction of $r \geq 2n$, we have to apply $MOD(P,N)$ more than once in Step 4(a). If $r < n$, we need more than n+1 words for storing the P obtained by Step 3.) Since $MOD(P,N)$ may be negative, we need Step 4(b) for correction.

5 Conclusion

We have proposed a new multiple-precision modular multiplication algorithm. Since it is both fast and uses a small amount of main memory, the algorithm is efficient for implementation on small computers, such as card computers, and is useful for application of a public-key cryptosystem to such computers.

We have also proposed a new efficient procedure for residue calculation which accelerates the proposed algorithm. In cooperation with this procedure, the algorithm achieves competitive speed to Montgomery's algorithm [8]. Note that although it is fast, Montgomery's algorithm requires pre- and post-processing for conversion to and from Montgomery's representation.

The proposed residue calculation procedure can also be used in the division-after-multiplication method for accelerating it. It may also accelerate multiple-precision division.

References

[1] R. L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Commun. ACM, vol. 21, no. 2, pp. 120-126, Feb. 1978.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Trans. Information Theory, vol. IT-31, no. 4, pp. 469-472, July 1985.

[3] E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography", D. Chaum et al Eds., 'Advances in Cryptology, Proceedings of CRYPTO 82', pp. 51-60, Plenum Press, New York, 1983.

[4] H. Morita, "A fast modular-multiplication algorithm based on a higher radix", Lecture Notes in Computer Science, vol. 435, G. Brassard Ed., 'Advances in Cryptology - CRYPTO'89 Proceedings', pp. 387-399, Springer-Verlag, 1990.

[5] A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer and P. G. A. Jespers, "A new carry-free division algorithm and its application to a single-chip 1024-b RSA processor", IEEE J. Solid-State Circuits, vol. 25, no. 3, June 1990.

[6] D. E. Knuth, 'The art of computer programming, vol. 2: Seminumerical algorithms, second edition', "Section 4.3: Multiple-precision arithmetic", Addison-Wesley, 1981.

[7] H. Morita and C. H. Yang, "A modular multiplication algorithm using lookahead determination", IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences, vol. E76-A, no. 1, pp. 70-77, Jan. 1993.

[8] P. L. Montgomery, "Modular multiplication without trial division", Mathematics of Computation, vol. 44, no. 170, pp. 519-521, 1985.

Appendix: Correctness of $MOD(P, N)$

We prove the correctness of Function $MOD(P, N)$. Namely, we show that $MOD(P, N) \equiv P \pmod N$ and $-N \leq MOD(P, N) < N$ hold.

(Proof)

Since $MOD(P, N) = P - (-1)^{ps} \cdot q \cdot N$ for a certain integer q, $MOD(P, N) \equiv P \pmod N$ holds.

To prove $-N \leq MOD(P, N) < N$, we have to consider the following six cases. Hereafter, P' denotes the part of P less than r^{n-1} and N' denotes the part of N less than r^{n-2}. Namely, $P = -ps \cdot r^{n+1} + P_n \cdot r^n + P_{n-1} \cdot r^{n-1} + P'$ and $N = N_{n-1} \cdot r^{n-1} + N_{n-2} \cdot r^{n-2} + N'$.

Case 1: $P \geq 0$

Case 1-1: $P_n = N_{n-1}$

$q = r - 1$

$MOD(P, N) = P - (r-1) \cdot N$

$MOD(P, N) \geq N_{n-1} \cdot r^n - (r-1) \cdot N$

$> r^{n-1} \cdot (N_{n-1} - r + 1) > -N$

$MOD(P, N) < r \cdot N - (r-1) \cdot N = N$

Case 1-2: $P_n < N_{n-1}$

$rem = P_n \cdot r + P_{n-1} - \hat{q} \cdot N_{n-1}$

$0 \leq rem < N_{n-1}$

Case 1-2a: $(\hat{q} \cdot N_{n-2})_{high} - rem \leq \lfloor N_{n-1}/2 \rfloor$

$q = \hat{q}$

$MOD(P, N) = P - \hat{q} \cdot N$

$= rem \cdot r^{n-1} + P' - \hat{q} \cdot (N_{n-2} \cdot r^{n-2} + N')$

$MOD(P, N) > (rem - (\hat{q} \cdot N_{n-2})_{high} - 2) \cdot r^{n-1}$

$\geq -(\lfloor N_{n-1}/2 \rfloor + 2) \cdot r^{n-1} \geq -N$

$MOD(P, N) \leq rem \cdot r^{n-1} + P' < N$

Case 1-2b: $(\hat{q} \cdot N_{n-2})_{high} - rem > \lfloor N_{n-1}/2 \rfloor$

$q = \hat{q} - 1$

$MOD(P, N) = P - (\hat{q} - 1) \cdot N$

$= rem \cdot r^{n-1} + P' - \hat{q} \cdot (N_{n-2} \cdot r^{n-2} + N') + N$

$MOD(P, N) \geq -\hat{q} \cdot (N_{n-2} \cdot r^{n-2} + N') + N$

$> -(r-1) \cdot r^{n-1} + N > -N$

$MOD(P, N) \leq (rem - (\hat{q} \cdot N_{n-2})_{high}) \cdot r^{n-1} + P' + N$

$< (-\lfloor N_{n-1}/2 \rfloor + 1) \cdot r^{n-1} + N < N$

Case 2: $P < 0$

Case 2-1: $\bar{P}_n = N_{n-1}$

$q = r - 1$

$MOD(P, N) = P + (r-1) \cdot N$

$MOD(P, N) \geq -r \cdot N + (r-1) \cdot N = -N$

$MOD(P, N) < -r^n \cdot N_{n-1} + (r-1) \cdot N$

$< r^{n-1} \cdot (r - 1 - N_{n-1}) < N$

Case 2-2: $\bar{P}_n < N_{n-1}$

$rem = (r^2 - P_n \cdot r - P_{n-1} - 1) - \hat{q} \cdot N_{n-1}$

$0 \leq rem < N_{n-1}$

Case 2-2a: $(\hat{q} \cdot N_{n-2})_{high} - rem \leq \lfloor N_{n-1}/2 \rfloor$

$q = \hat{q}$

$MOD(P, N) = P + \hat{q} \cdot N$

$= (-r^{n+1} + P_n \cdot r^n + P_{n-1} \cdot r^{n-1} + P')$

$+\hat{q} \cdot (N_{n-1} \cdot r^{n-1} + N_{n-2} \cdot r^{n-2} + N')$

$= (-r^2 + P_n \cdot r + P_{n-1} + \hat{q} \cdot N_{n-1}) \cdot r^{n-1}$

$+P' + \hat{q} \cdot (N_{n-2} \cdot r^{n-2} + N')$

$= -(rem + 1) \cdot r^{n-1} + P' + \hat{q} \cdot (N_{n-2} \cdot r^{n-2} + N')$

$MOD(P, N) \geq -(rem + 1) \cdot r^{n-1}$

$\geq -N_{n-1} \cdot r^{n-1} \geq -N$

$MOD(P, N)$

$= (-rem - 1 + (\hat{q} \cdot N_{n-2})_{high}) \cdot r^{n-1}$

$+(\hat{q} \cdot N_{n-2})_{low} \cdot r^{n-2} + P' + \hat{q} \cdot N'$

$< (\lfloor N_{n-1}/2 \rfloor + 2) \cdot r^{n-1} < N$

Case 2-2b: $(\hat{q} \cdot N_{n-2})_{high} - rem > \lfloor N_{n-1}/2 \rfloor$

$q = \hat{q} - 1$

$MOD(P, N) = P + (\hat{q} - 1) \cdot N$

$= -(rem + 1) \cdot r^{n-1} + P'$

$+\hat{q} \cdot (N_{n-2} \cdot r^{n-2} + N') - N$

$MOD(P, N)$

$\geq (-rem - 1 + (\hat{q} \cdot N_{n-2})_{high}) \cdot r^{n-1} - N$

$> (\lfloor N_{n-1}/2 \rfloor - 1) \cdot r^{n-1} - N > -N$

$MOD(P, N)$

$\leq -r^{n-1} + P' + \hat{q} \cdot (N_{n-2} \cdot r^{n-2} + N') - N$

$< r^n - N \leq N$

Thus, in any case, $-N \leq MOD(P, N) < N$ holds.

Therefore, the correctness of Function $NOD(P, N)$ has been proven. □