Dual-Residue Montgomery Multiplication

Anding Wang, Yier Jin, and Shiju Li

College of Information Science & Engineering
Zhejiang University, Hangzhou, China
anding_704@hotmail.com, jinyier@gmail.com

Abstract. The paper introduces a new approach based on dual residue system to compute Montgomery multiplication. The novelty of this proposal is that we import an extra Montgomery residue system with new transformation constant beside the normal one. In this way, one of the multiplicand can be divided into two parts and both higher and lower parts are calculated in parallel to speed up computation. Then two implementations in hardware are proposed for the algorithm. In parallel architecture, the proposed algorithm can perform nearly twice speedup compared to normal Montgomery method. And in pipeline architecture, the computation speed can be even faster. Besides speeding up calculation the extra merit of our proposal is that the multiplier can partial replace Montgomery multiplier used nowadays without any changes on top architecture.

Keywords: Dual residue system, Montgomery algorithm, Parallelism.

1 Introduction

Modular multiplication is one of the basic computations which are widely used in public-key cryptography, especially in RSA algorithm [11], Diffie-Hellman key exchange algorithm [3], elliptic curve cryptography(ECC) and digital signature algorithm(DSA) [14]. Compared to other computation such as modular addition, modular multiplication costs much more time, so low-complexity algorithms and their high-efficient implementations both in hardware and software have been studied for a long time.

In the aspect of algorithm improvement, several methods are proposed to simplify divisions or to avoid trial divisions. Among them, two approaches are of major concentration: One is based on the interleaved modular multiplication algorithm, and the other is based on the Montgomery algorithm [5]. The former's strategy is to simplify the division by finding the quotient early and then to get modular result. Barrett modified this algorithm by getting approximate result first according to a finely selected radix $b = 2^L$(L equals to two times of the length of modulus), and then subtract modulus in less than two times as Barrett's algorithm [2]. The latter's strategy avoids division by adding multiples of modulus in order to make lower bits of product zero. It leverages pre-computation which transforms operators into residue system to simplify the operation.

K. Li et al. (Eds.): NPC 2007, LNCS 4672, pp. 267–276, 2007.

Among these algorithms, it has been proved that the Montgomery method is better in most cases. However, each step in Montgomery Algorithm depends on the previous step's result that means this algorithm cannot be operated in parallel. Although Koc proposed a new architecture which can improve parallelism in reconfigurable hardware implementation, this modification just change the implementation form not the algorithm itself [16].

In this paper, we proposes a method that explores the parallelism of Montgomery algorithm itself to further boost speed. The point of this improvement in parallel comes from splitting normal residue system into dual-residue system modulo M. Since n-word modulus M is always an odd integer in cryptography application, an integer R which is coprime to M in normal Montgomery algorithm is often set $R_n = r^n$ where r is an s-bit word. Then we choose appropriate integers a and b which fulfill $a + b = n$. The novelty of our proposal is that we import an extra residue system with transformation constant $R_b = r^b$ beside the formal residue system where $R_n = r^n$. As $b < n$, the transformation constant R_b is less than the modulus M, similar to Kaihara and Takagi's new representation [5]. This extra residue system enables the splitting of one multiplier into higher part and lower part and each part can be computed in parallel. Both operations are processed under Montgomery algorithm except for different residues. Since parameters a and b can be chosen from 0 to $n - 1$, the variation of b covers all possibilities of the new residues with different performances. In the result showed below, we find when b is slightly larger than $\lceil n/2 \rceil$, the speed up is close to twice than that of normal Montgomery algorithm.

Further advantage is that the inputs and outputs on the top level of this new method are the same as normal Montgomery algorithm, so the implementation of this method no matter in hardware or in software can take place of Montgomery multiplier without any changes in high level. And even advanced, our proposal can make use of all improvements made for Montgomery algorithm up to date.

2 Montgomery Algorithm

Montgomery modular multiplication was first proposed by P.L. Montgomery in 1985 [4] and was a powerful modular algorithm to deal with arbitrary modulus M. It does not compute modular multiplication straightforward, instead, it leverage some pre-computation to transform multiplicands into residue system by a transformation constant R coprimed to modulus M. The M-residue representation of integers $A, B < M$ are defined as $X = A \cdot R (\mathrm{mod}\ M)$, $Y = B \cdot R (\mathrm{mod}\ M)$. Although this algorithm works for any R that is coprime to M and is larger than M, it is more useful when R is a power of 2 both in hardware and software implementation. To facilitate the implementation on word based system, we use s-bit word r as radix and let R be a power of r. Because M is often an odd integer in reality, the relative prime condition is fulfilled initially. Let $R = r^n$, the Montgomery multiplication algorithm computes

$$\begin{aligned} C = A \otimes_n B &= ABR^{-1} \bmod M \\ &= ABr^{-n} \bmod M \end{aligned} \tag{1}$$

for n-word integers satisfying the following condition:

$$0 \leq A, B < M.$$

If the inputs are X, Y,the result will be

$$C' = XYR^{-1} = A \cdot R \cdot B \cdot R \cdot R^{-1} = ABR \bmod M.$$

The key idea of the Montgomery algorithm is to add an appropriate multiple of M to make the lowest n words of AB equal to 0. As $M = \sum_{i=0}^{n-1} m_i \cdot r^i$, $A = \sum_{i=0}^{n-1} a_i \cdot r^i$, $B = \sum_{i=0}^{n-1} b_i \cdot r^i$ and the word-level Montgomery multiplication algorithm is described below.

Algorithm 1. Word-level Montgomery Modular Multiplication

Input	A, B, M, $(0 \leq A, B < M)$, $(r^{n-1} < M < r^n), (R = r^n), m_0^{-1} (\bmod\ r)$
Output	$C = A \otimes_n B = ABr^{-n} \bmod M$
	$C := 0$ For $i = 0$ to $n-1$ $t_i = (c_0 + a_i b_0) m_0^{-1} \bmod r$ $C = (c_i + a_i B + t_i M)/r$; if $(C > M)$ then $C := C - M$ return C;

The transformation between normal representation and the M-residue representation can also be performed through Montgomery multiplication. X can be computed by multiplying A with $R^2 \bmod M$ in Montgomery algorithm as $X = A \otimes_n R^2 \bmod M = A \cdot R^2 \cdot R^{-1} \bmod M = A \cdot R \bmod M$. The backwards transformation can be performed between X and constant 1 which can be presented as $A = X \otimes_n 1 = A \cdot R \cdot R^{-1}$.

3 Dual-Residue Montgomery Algorithm

In order to speed up the Montgomery multiplication, a new fast method which improves the parallelism of this algorithm is proposed. The novelty of the new algorithm comes from the splitting of M-residue representation into a normal residue representation and a new residue representation. In the new residue representation, the constant R is less than modulus M, a situation that was forbidden in normal Montgomery algorithm. We denote the new constant as R_b which equals to r^b with $b < n$. The computation of modular multiplication can be processed in two residue systems in parallel to improve the processing speed.

According to the Montgomery modular multiplication of two integers with transformation constant $R = r^n$ mentioned in Section 2, the algorithm can be rewritten here with X, Y, the images of A, B, respectively:

$$X \otimes_n Y = XYr^{-n} \bmod M.$$

Our proposal is to achieve this goal with new method other than that in Algorithm 1. First, we split multiplier Y into two parts Y_H and Y_L according to a parameter a, $a < n$, i.e., $Y = Y_H \cdot r^a + Y_L$. Then we define another parameter b equal to $n - a$ to construct a new residue modulus M with transformation constant r^b. It is obviously that in this new residue system the constant r^b is less than modulus M as $b < n$ which is not the case required by Montgomery algorithm. However, we will show the effectiveness of the Montgomery computation in the new residue system later. With these pre-definitions, the Montgomery multiplication modulo M of images X, Y can be computed as follows:

$$X \otimes_n Y = (X \otimes_b Y_H + X \otimes_n Y_L) \bmod M \tag{2}$$

In Equation 2, the left term, $X \otimes_b Y_H$, is calculated using Montgomery algorithm where transformation constant is r^b. The right term, $X \otimes_n Y_L$, is calculated using Montgomery algorithm where the constant is r^n normally. These two calculations are performed in parallel. The split multiplicands Y_H and Y_L are both shorter than Y in length, so they can be performed faster than the NORMAL Montgomery method with unsplit operands. The computation details of $X \otimes_b Y_H$ and $X \otimes_n Y_L$ is described as follows and the computation process of the new Montgomery algorithm is shown in Figure 1.

The correctness of Equation 2 can be defined as following:

$$\begin{aligned} & (X \otimes_b Y_H + X \otimes_n Y_L) \bmod M \\ =& (X \cdot Y_H \cdot r^{-b} + X \cdot Y_L \cdot r^{-n}) \bmod M \\ =& X \cdot (Y_H \cdot r^{-n} \cdot r^a + Y_L \cdot r^{-n}) \bmod M \\ =& X \cdot (Y_H \cdot r^a + Y_L) \cdot r^{-n} \bmod M \\ =& X \cdot Y \cdot r^{-n} \bmod M \end{aligned} \tag{3}$$

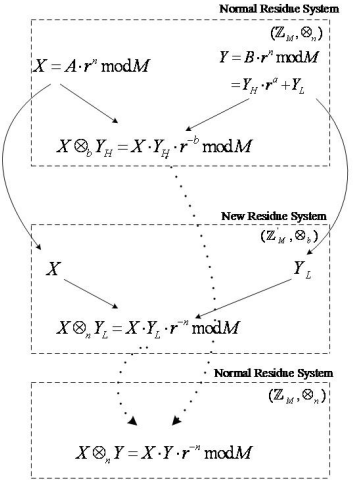

Fig. 1. The Computation process of New Montgomery Algorithm

Here we concentrate on the first term $X \otimes_b Y_H$ where the parameter b should fulfill $(X \cdot Y_H + m \cdot M)/r^b < 2M$, we have$X \cdot Y_H + m \cdot M < M \cdot r^{r-a} + M \cdot 2^b = M \cdot (r^b + r^b) = 2 \cdot M \cdot r^b$, so the only limitation of a and b is that $a + b = r$ and this condition is fulfilled initially.

Since the implementations of Montgomery algorithm can affect the computation performances and in our method, the operands are not of normal form, we pay attention to the selection of algorithms. In [9], Koc introduced several methods to compute Montgomery algorithm in word level and among these approaches, Coarsely Integrated Operand Scanning (CIOS) algorithm is proved to be least complex. Furthermore, CIOS algorithm is much more appropriate than others to implement our proposed dual residue Montgomery algorithm.

For the left term $X \otimes_b Y_H$, operand X is n words long, operand Y_H is b words long and the transformation constant is r^b. So we implement CIOS algorithm by reducing the iteration times in order to simplify the computation using the characteristic of short operand. The CIOS algorithm is showed in Figure 2.

Algorithm 2. CIOS Algorithm with constant r^b

Input	$X, Y_H, M, M'[0] = M^{-1}[0] \bmod r, r^b$
Output	$T = X \cdot Y_H \cdot r^{-b} \bmod M$
	$For\ i = 0\ to\ b-1\{$
	$C := 0$
	$For\ j = 0\ to\ n-1\{$
	$(C, S) := T[j] + X[j] * Y_H[i] + C$
	$T[j] := S$
	$\}$
	$(C, S) := T[s] + C$
	$T[s] := S$
	$T[s+1] := C$
	$C := 0$
	$m := T[0] * M'[0] \bmod r$
	$(C, S) := T[0] + m * M'[0]$
	$for\ j = 1\ to\ n-1\{$
	$(C, S) := T[j] + m * M[j] + C$
	$T[j-1] := S$
	$\}$
	$(C, S) := T[s] + C$
	$T[s-1] := S$
	$T[s] := T[s+1] + C$
	$\}$

For the right term $X \otimes_r Y_L$, operand X is n words long, operand Y_H is a words long and the constant is r^n. Although one of the operand is shorter than normal operand in Montgomery algorithm, the constant remains r^n which means we cannot directly implement CIOS algorithm mentioned above because the normal form cannot strictly effectively make use of the short-operand and the improvement of speed will be quite restricted. If so, the parallelism of our algorithm will be consumed by poor performance of term $X \otimes_n Y_L$. To avoid

this restriction, we modify CIOS algorithm to fit our proposal and the modified CIOS algorithm is listed in Algorithm 3.

Algorithm 3. Modified CIOS Algorithm with constant r^b

Input	$X, Y_L, M, M'[0] = M^{-1}[0] \bmod r, r^b$
Output	$T = X \cdot Y_L \cdot r^{-b} \bmod M$

```
For i = 0 to b − 1{
  C := 0
  For j = 0 to n − 1{
    (C, S) := T[j] + X[j] ∗ Y_L[i] + C
    T[j] := S
  }
  (C, S) := T[s] + C
  T[s] := S
  T[s + 1] := C
  C := 0
  m := T[0] ∗ M'[0] mod r
  (C, S) := T[0] + m ∗ M'[0]
  for j = 1 to n − 1{
    (C, S) := T[j] + m ∗ M[j] + C
    T[j − 1] := S
  }
  (C, S) := T[s] + C
  T[s − 1] := S
  T[s] := T[s + 1] + C
  }
For i = 0 to a − 1{
  C := 0
  m := T[0] ∗ M'[0] mod r
  (C, S) := T[0] + m ∗ M'[0]
  for j = 1 to n − 1{
    (C, S) := T[j] + m ∗ M[j] + C
    T[j − 1] := S
  }
  (C, S) := T[s] + C
  T[s − 1] := S
  T[s] := T[s + 1] + C
  }
```

The complete computation of modular multiplication are listed below as Algorithm 4. In this algorithm, X and Y indicate the multiplicands of modular multiplication and Y_H, Y_L are higher and lower parts of Y, respectively. $TMP1$ and $TMP2$ are internal variables which are used to store partial products. $CIOS(X, Y_H)$ is a function whose detail is described in Algorithm 2 and $Modified\_CIOS(X, Y_L)$ make use of algorithm mentioned in Algorithm 3. The initial stage Step 1 transfers multiplicands from memory to register and flushes all the internal variables. Step 2 computes $TMP1$ and $TMP2$ in parallel. The complexity of our algorithm depends on this step and the parallelism depth

decides the performance improvement of our proposal compared to normal Montgomery algorithm. Step 3 merges two internal variables together through a modular adder.

Algorithm 4. Dual Residue Montgomery Multiplication	
Input	X, Y, M, $(0 \leq X, Y < M)$, $(r^{n-1} < M < r^n), (R = r^n), M_0^{-1} (\text{mod } r)$
Output	$C = X \otimes_n Y = XYr^{-n} \bmod M$
step 1	$TMP1 := 0;\ TMP2 := 0;\ Y_H = Y/r^a;\ Y_L = Y \bmod r^a;$
step 2	$TMP1 = CIOS(X, Y_H, r^b)$ $TMP2 = \text{Modified\_CIOS}(X, Y_L, r^n)$
step 3	$C = (TMP1 + TMP2) \bmod M;$

4 Hardware Implementation

4.1 Parallel Architecture

The diagram of multiplier with parallel architecture based on the new Montgomery algorithm are showed in figure 2. It includes six registers, two operation units(CIOS and Modified CIOS) and one modular adder. These six n-word long registers are used to store multiplicands X and Y, modulus M, internal variables $TMP1$ and $TMP2$ and final product C. The CIOS algorithm architecture can be find in [6], although we can make use of any other designs which may be of higher efficiency. In fact, this is one advantage of our algorithm that can make use of any available improvement of Montgomery algorithm no matter in hardware or in software. In the multiplier, input registers which store multiplicands X and Y, modulus M are all barrel registers of n-word width.

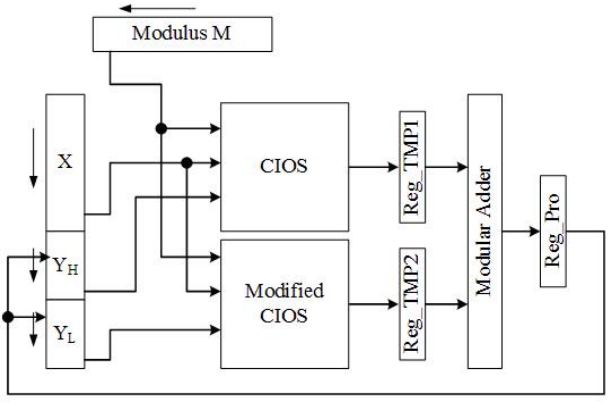

Fig. 2. The diagram of parallel architecture multiplier

In the aspect of area cost, this proposed multiplier requires one more Montgomery multiplier with CIOS architecture, two extra register for internal variables and a modular adder. As the CIOS architecture mentioned in [6] has an internal state machine and the hardware cost is less than other implementations, the extra hardware cost is insignificant.

Transformation between ordinary integers and their residue class representation is also performed in the same multiplier. The forward transformation is to compute modular multiplication of multiplicand A and $R^2(\mathrm{mod}M)$ which can be divided into higher and lower part to fit this multiplier. And backward transformation between X and constant 1 is less complex as the higher part is zero and only the lower part computation is required.

In reality, the division of multiplicand Y according to parameter a is case specific and the process element of CIOS algorithm architecture can be changed in variable environment. For example, the CIOS and Modified CIOS algorithm are of same radix and compute in same sequence, a should be set slightly smaller than $\lceil n/2 \rceil$ in order to achieve most significant performance speedup because the Modified CIOS algorithm requires more computation steps than CIOS algorithm. a should be chosen around $\lceil n/3 \rceil$ when CIOS algorithm runs at radix-2 while the Modified CIOS algorithm based on radix-4 [15].

This architecture is of high attractive when used in cryptography application such as public key cipher RSA. Because the use of our multiplier can reach nearly twice speedup with sequential output which cannot be produced through several independent Montgomery multipliers. Another advantage is that the product of our multiplier is the same as normal Montgomery algorithm although the computation process is different. That means our multiplier can partial replace Montgomery multiplier used nowadays without any change on top architectures. This characteristic provides significant flexibility on time and space trade-off. In the time critical paths, our multiplier can be used to speed up computation while in the area critical occasions normal Montgomery multiplier is used.

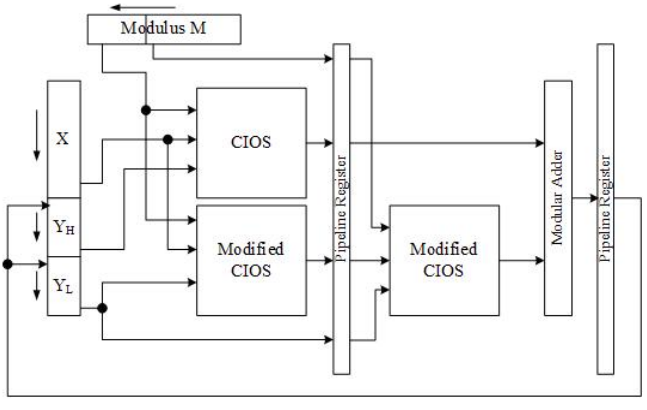

Fig. 3. The diagram of pipeline architecture multiplier

4.2 Pipeline Architecture

In other cryptography applications such as Elliptic Curve Cryptosystem(ECC) [14], the product of multiplier is not used in the next round immediately which differs from that in RSA system. In order to further improve multiplication speed, we introduce an two stages pipeline architecture showed in figure 3.

Compared to parallel architecture, the pipelined one require one more register as pipeline register. The modified CIOS algorithm is performed in two steps which is distributed in two stages. With this architecture, the bipartite multiplier can achieve even higher computation speed, i.e., we set a around $\lceil n/3 \rceil$, the time cost is only one third of normal Montgomery multiplication.

5 Conclusion

In this paper, we proposed a dual-residue method to compute Montgomery algorithm. In this approach, we define two residue systems with one normal transformation constant and another smaller one which assures the splitting of multiplicand and compute partial modular multiplication in parallel under CIOS algorithm or other Montgomery algorithms with low complexity. The implementation in hardware is then presented and two architectures are developed fit for different applications, in which the pipelined one can achieve three times speedup with little extra hardware cost. Further work will be concentrate on converting this method from GF(p) to GF(2^m).

References

1. Knuth, D.E.: The Art of Computer Programming — Seminumerical Algorithm, 3rd edn., vol. 2. Addison-Wesley, Reading (1998)
2. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on astandard digital signal processor. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 311–323. Springer, Heidelberg (1987)
3. Diffie, W., Hellman, M.E.: New Directions in Cryptography. IEEE Trans. Infomation Theory 22(11), 644–654 (1976)
4. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
5. Kaihara, M.E., Takagi, N.: Bipartite Modular Multiplication. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 201–210. Springer, Heidelberg (2005)
6. McLoone, M., McIvor, C., McCanny, J.V.: Coarsely integrated operand scanning (CIOS) architecture for high-speed Montgomery modular multiplication. In: IEEE International Conference on Field-Programmable Technology(ICFPT'04), pp. 185–191 (2004)
7. Walter, C.D.: Space/Time Trade-Offs for Higher Radix Modular Multiplication Using Repeated Addition. IEEE Trans. Computers 46(2) (1997)
8. Manochehri, K., Pourmozafari, S.: Modified radix-2 Montgomery modular multiplication to make it faster and simpler. In: International Conference on Information Technology: Coding and Computing(ITCC'04), vol. 1, pp. 598–602 (2005)

9. Ko ç, Ç.K., Acar, T., Kaliski Jr., B.S.: Analyzing and comparing Montgomery multiplication algorithms. IEEE Micro 16(3), 26–33 (1996)
10. Chiou-Yng, L., Jenn-Shyong, H., I-Chang, J., Erl-Huei, L.: Low-complexity bit-parallel systolic Montgomery multipliers for special classes of GF(2^m). IEEE Trans. Computers 54(9), 1061–1070 (2005)
11. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commum. ACM 21(2), 120–126 (1978)
12. Hars, L.: Long Modular Multiplication for Cryptographic Applications. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 45–61. Springer, Heidelberg (2004)
13. Yanik, T., Savas, E., Koç, Ç.K.: Incomplete reduction in modular arithmetic. IEE Proceedings-Computers and Digital Techniques 149(2), 46–52 (2002)
14. IEEE Standard Specifications for Public-Key Cryptography, IEEE Std 1363-2000 (2000)
15. Tawalbeh, L.A., Tenca, A.F., Koç, Ç.K.: A radix-4 scalable design. IEEE Potentials 24(2), 16–18 (2005)
16. Tenca, A.F., Koç, Ç.K.: A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm. IEEE Trans. Computers 52(9) (2003)