INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 4, Issue 3 JULY 2015

ANALYSIS OF RNS MONTGOMERY EXPONENTIATION ALGORITHM FOR CRYPTOGRAPHY

[1]**CH. Mahesh**
PG Scholar in VLSI System Design,
[2]**S. Koteshwar Rao**
M.TECH, ASST. PROFESSOR, ECE Department,
[1]chekurimahesh92@gmail.com,
Avanthi Institute of Engineering & Technology, Hyderabad, Telangana

Abstract:

The hardware implementation of modular exponentiation for very large integers is a well-known topic in digital arithmetic. An effective approach for obtaining parallel and carry-free implementations consists in using the Montgomery exponentiation algorithm and executing the necessary operations in RNS. Two efficient methods for performing the RNS Montgomery exponentiation have been proposed by Kawamura et al. and by Bajard and Imbert. The above approaches mainly differ in the algorithm used for implementing the base extension. This paper presents a modified RNS Montgomery exponentiation algorithm, where several multiplications are moved outside the main execution loop and replaced by an effective pre-processing stage producing a significant saving on the overall delay with respect to stateof-the-art approaches. Since the proposed modification should be applied to both of the above algorithms, two versions are specifically discussed. The overall comparison shows that with the proposed approach, a 18:5% speedup can be achieved for an implementation over 1024 bits, without any significant area overhead.

Keywords— RNS, Montgomery reduction, Modular exponentiation, Modular multiplication.

I. Introduction

The computation of the Montgomery exponentiation (ME) in the Residue Number System (RNS) [1] allows limiting the delay due to carry propagation and reaching a high degree of parallelism [2]. This approach mainly requires the execution of a set of Montgomery multiplications (MMs) [3]. However, in RNS, some operations (e.g. division, comparison, modulo) are natively difficult to execute. Hence, several approaches have been proposed in order to fully exploit the potential of RNS for modular exponentiation, by minimizing the impact of related drawbacks. A key element of these approaches is the Base Extension (BE), which calculates a number on a different RNS base.

The graphics processing unit (GPU) has enjoyed a large increase in floating point performance compared with the CPU in the last number of years. The traditional CPU has leveled off in terms of clock frequency as power and heat concerns increasingly become dominant restrictions. The latest GPU from Nvidia's GT200 series reports a peak throughput of almost 1 TeraFlop, whereas the latest Intel CPUs reported throughput is in the order of 100 GigaFlops [1]. This competitive advantage of the GPU comes at the price of a decreased applicability to general purpose computing. The latest generation of graphics processors, which are DirectX 10 [2] compliant, support integer processing and give more control over the processor's threading and memory model compared to previous GPU generations. We use this new generation of GPU to accelerate public key cryptography. In particular we use an Nvidia 8800GTX GPU with CUDA [3] to investigate the possibility of high speed 1024-bit RSA decryption. We focus on 1024-bit RSA decryption as it shows a high arithmetic intensity, ratio of arithmetic to IO operations, and also allows easy comparison with CPU implementations. W e exploit the new GPU's flexibility to support a GPU sliding window [4]

exponentiation implementation, based on Montgomery exponentiation [5] using both radix and residue number system (RNS) representations. We investigate both types of number representation showing how GPU occupancy and inter thread communication plays a central role to performance. Regarding the RNS implementations, we exploit the GPU's flexibility to use a more optimized base extension approach than was previously possible. We also explore various GPU implementations of single precision modular multiplication for use within the exponentiation approaches based on RNS.

A significant number of applications including cryptography, error correction coding, computer algebra, DSP, etc., rely on the efficient realization of arithmetic over finite fields of the form GF (2^n), where n € Z and n ≥1,or the GF(p) form ,where p a prime. Cryptographic applications form a special case, since, for security reasons, they require large integer operands efficient field multiplication with large operands is crucial for achieving a satisfying cryptosystem performance, since multiplication is the most time- and area-consuming operation. Therefore, there is a need for increasing the speed of cryptosystems employing modular arithmetic with the least possible area penalty. An obvious approach to achieve this would be through parallelization of their operations. In recent years, RNS and PRNS have enjoyed renewed scientific interest due to their ability to perform fast and parallel modular arithmetic. Using RNS/PRNS, a given path serving a large data range is replaced by parallel paths of smaller dynamic ranges, with no need for exchanging information between paths. As a result, the use of residue systems can offer reduced complexity and power consumption of arithmetic units with large word lengths. On the other hand, RNS/PRNS implementations bear the extra cost of input converters to translate numbers from a standard binary format into residues and output converters to translate from RNS/PRNS to binary representations.

A new methodology for embedding residue arithmetic in a dual-field Montgomery modular multiplication algorithm for integers in GF(p) and for polynomials in GF (2^n) is presented in this paper. The mathematical conditions that need to be satisfied for a valid RNS/PRNS incorporation are examined. The derived architecture is highly parallelizable and versatile, as it supports binary-to-RNS/PRNS and

have been used extensively to reduce the RNS/PRNS-to-binary conversions, Mixed Radix Conversion (MRC) for integers and polynomials, dual-field Montgomery multiplication, and dual-field modular exponentiation and inversion in the same hardware.

Residue Number System

In recent years, we have experienced a great development in the field of digital communication technologies which brought together a great concern about security in computers and communications systems. Several public-key cryptosystems were proposed in order to enable the encryption of messages using a public encryption key 'e' without a prior communication of a secret key. The secrecy relies on the fact that decryption key is computationally infeasible to deduce from the public encryption key. Then, the only person who can decrypt the cipher-text is the receiver, who knows the secret decryption key'd'.

Public-key cryptography plays an important role in digital communication and storage systems. Processing public-key cryptosystems requires huge amount of computation, and, there is therefore, a great demand for developing dedicated hardware to speed up the computations. Speeding up the computation using specialized hardware enables the use of larger keys in public-key cryptosystems. This is translated into an increase of the security of the system. Also, this enables the speedup of a secure link between two distant points using an insecure channel, which is critical in real-time systems. The reduction of the hardware amount is another important aspect when implementing in dedicated hardware because it allows for the miniaturization of portable devices and reduces fabrication costs

The Residue Number System (RNS) is a non-weighted number system that can map large numbers to smaller residues, without any need for carry propagations .Its most important property is that additions, subtractions, and multiplications are inherently carry-free. These arithmetic operations can be performed on residue digits concurrently and independently. Thus, using residue arithmetic, would in principle, increase the speed of computations RNS has shown high efficiency in realizing special purpose applications such as digital filters , image processing , RSA cryptography and specific applications for which only additions, subtractions and multiplications are used and the number dynamic range is specific. Special moduli sets

hardware complexity in the implementation of converters and arithmetic operations. Among which the triple moduli set $\{2^n+1,2^n,2^n-1\}$ have some benefits. Since the operation of multiplication is of major importance for almost all kinds of processors, efficient implementation of multiplication modulo 2^n-1 is important for the application of RNS.

II. Literature survey

In [4], Kawamura et al. proposed an RNS ME technique applied to RSA, and a new BE. The BE is characterized by a summation that provides a result modulo a small multiple of the base, which is corrected after the sum of each element. The proposed approach has been detailed in [5], where an architecture has also been presented, showing that its performance are faster than not RNS approaches.

In [6], Bajard et al. proposed an implementation of the MM based on both the RNS and the Mixed Radix Number System (MRS), where the MRS corresponds to a weighted system associated with the RNS. Then, in [7], the same authors proposed a Montgomery multiplication method fully implemented in RNS. This approach employs an approxi-mated BE and the algorithm proposed in [8], where the result is approximated and corrected by using an extra modulo. Finally in [9], Bajard and Imbert detailed the application of the previous ME approach in the context of RSA.

The RNS MM has also been studied in different contexts, out of the exponentiation. In [10], an implementation for elliptic curve cryptography on FPGA is presented. This implementation employs the algorithm proposed in [4] with some pre-computations suitable to the particular architecture presented in the paper. In [2], an implementation of RNS MM on GPU is presented. Experimental results achieved in the above context show that the algorithm described in [7] provides the best performance.

This paper proposes a modified ME algorithm for RNS. The novelties of the algorithm are in the pre-processing stage, which is used to reduce the number of multiplications performed in the loop of the exponentiation algorithm. It is worth observing that, even though in the specific case the ME is considered, the basic approach used

The reconstruction expression is:

here is more general and can be easily applied in other contexts. In par-ticular, the analysis of the designed pre-processing method for generic modular multiplication is presented in [11].

The differences with respect to previous approaches con-sist in the values that are pre-computed, and in the new ME and MM algorithms. The modifications discussed in the current work are applicable to both the state-of-the-art MM algorithms [4], [7], which mainly differ for the BE correction methods. Hence, in the following, the proposed modifications will be presented in two versions specifically tailored to the characteristics of these methods.

III. Residue Arithmetic

A. Residue Number System

RNS consists of a set of ,L pair-wise relatively prime integers A= (m1.m2,……,m_L) (called the base) and the range of the RNS is computed as $A = \prod_{i=1}^{L} m_i$. Any integer Z €[0,A-1] has a unique RNS representation Z_A given by Z_A=(Z1,Z2,….ZL) = ((Z)m1, (Z)m2, (Z)m3, (Z)m4…….. (Z)mL), where (Z)mi denotes the operation Z mod mi. Assuming two integers a,b in RNS format, i.e., a_A=(a1,a2,….aL) and, b_A=(b1,b2,…bL) then one can perform the operations in parallel by

$$a_A \otimes b_A = \left(\langle a_1 \otimes b_1 \rangle_{m_1}, \langle a_2 \otimes b_2 \rangle_{m_2}, \ldots, \langle a_L \otimes b_L \rangle_{m_L} \right)$$

To reconstruct the integer from its residues, two methods may be employed [14]. Thefirst is through the CRT according to

$$z = \sum_{i=1}^{L} \left\langle z_i \cdot A_i^{-1} \right\rangle_{m_i} \cdot A_i - \gamma A$$

Where Ai =A/ai and A^1i is the multiplicative inverse of Ai on ai . The result of ∑ki=1((xi A1i) mod ai) Ai is equal to x + A, where < k.

Using the Chinese Remainder Theorem (CRT) [12], it is possible to convert a value x from an RNS base to a radix system, achieving a high parallelism.

$$x = (\sum_{i=1}^{k}((x_i A_i^{-1}) \bmod a_i)A_i) \bmod A \qquad (1)$$

where $A_i = \frac{A}{a_i}$ and A_i^{-1} is the multiplicative inverse of A_i on a_i. The result of $\sum_{i=1}^{k}((x_i A_i^{-1}) \bmod a_i)A_i$ is equal to $x + \lambda A$, where $\lambda < k$.

B. Montgomery Exponentiation (ME)

The comparison between the state-of-the-art RNS ME algorithm (the main part of the ME algorithm is common to both [4] and [9]) and the proposed algorithm is shown in Fig. 1. Since R = B, B mod N and B2 mod N must be precomputed. Step 1 in Algorithm 1 [4], [9] calculates x, as previously described. Steps 2 and 3 initialize the exponentiation process, which is executed in the loop from step 4 to step 9. The proposed RNS ME algorithm (Algorithm 2) executes 3 multiplication steps more than Algorithm 1 out of the loop, in order to execute less multiplication steps in the MM algorithm. Before executing the loop, all the values on base A are multiplied by A1j (steps 1 and 4). The multiplication by A1j is used in the state-of-the-art BE algorithm, in order to extend the values from A to B. In the proposed algorithm both the input values in A of the RNS MM are pre-multiplied by A1j so a multiplication by Aj in the RNS MM is required to reach the value for that BE.

In contrast to the original multiplication, the correction can be merged to another multiplication, decreasing the number of modular multiplications in the loop. The values that are multiplied by A1j in A are represented with a d hat accent. In order to reach the correct result of the exponentiation, another multiplication by Aj is required after the loop of RNS MMs (step 12).

C. RNS Montgomery Multiplication

In general, in RNS the MM [4], [7] is performed on two RNS bases, A = (a1; :::; ak) and B = (b1; :::; bk), such that

Algorithm 1: State-of-the-art RNS Montgomery exponentiation [4], [9]

Input: x in $\mathcal{A} \cup \mathcal{B} \cup a_r$ and $e = (e_{g-1}...e_1e_0)_b$
Output: $z = (x^e) \bmod N$ in $\mathcal{A} \cup \mathcal{B}$
Precomputation: $B^2 \bmod N$, $B \bmod N$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$

1: $\bar{x} \leftarrow \text{MM}\ (x, B^2 \bmod N)$
2: $z = B \bmod N$ in $\mathcal{B} \cup a_r$
3: $z = B \bmod N$ in $\mathcal{A}$
4: **for** i from $g-1$ down to 0 **do**
5: $\quad z \leftarrow \text{MM}\ (z, z)$
6: $\quad$ **if** $e_i = 1$ **then**
7: $\quad\quad z \leftarrow \text{MM}(z, \bar{x})$
8: $\quad$ **end if**
9: **end for**
10: $z \leftarrow \text{MM}(z, 1)$

Algorithm 2: Proposed RNS Montgomery exponentiation

Input: x in $\mathcal{A} \cup \mathcal{B} \cup a_r$ and $e = (e_{g-1}...e_1e_0)_b$
Output: $z = (x^e) \bmod N$ in $\mathcal{A} \cup \mathcal{B}$
Precomputation: $\hat{B}^2 \bmod N$, $B \bmod N$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$, $A_j \bmod a_j$ and $A_j^{-1} \bmod a_j$ for $j = 1...k$

1: $\hat{x}_{aj} = x_{aj} A_j^{-1} \bmod a_j$ for $j = 1...k$
2: $\hat{\bar{x}} \leftarrow \text{MM}\ (\hat{x}, \hat{B}^2 \bmod N)$
3: $z = B \bmod N$ in $\mathcal{B} \cup a_r$
4: $\hat{z}_{aj} = (B \bmod N)_{a_j} A_j^{-1} \bmod a_j$ for $j = 1...k$
5: **for** i from $g-1$ down to 0 **do**
6: $\quad \hat{z} \leftarrow \text{MM}\ (\hat{z}, \hat{z})$
7: $\quad$ **if** $e_i = 1$ **then**
8: $\quad\quad \hat{z} \leftarrow \text{MM}(\hat{z}, \hat{\bar{x}})$
9: $\quad$ **end if**
10: **end for**
11: $\hat{z} \leftarrow \text{MM}(\hat{z}, \hat{1})$
12: $z_{aj} = \hat{z}_{a_j} A_j \bmod a_j$ for $j = 1...k$

Figure 1. Comparison between the proposed RNS Montgomery exponentiation and the state-of-the-art algorithm

Algorithm 3: State-of-the-art RNS Montgomery multiplication [4], [7]

Input: x, y, and N in $\mathcal{A} \cup \mathcal{B} \cup a_r$, such that[1] $A = \prod_{j=1}^{k} a_j$, $B = \prod_{i=1}^{k} b_i$, and $\gcd(A, B) = 1$, $\gcd(N, B) = 1$, $0 \leq xy < NB$, $B > 4N$, and $A > 2N$
Output: $w \equiv xyB_N^{-1} \pmod N$, $w < 2N$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$
Precomputation: B_A^{-1}, N in $\mathcal{A} \cup a_r$; N^{-1} in $\mathcal{B}$

1: $s = xy$ in $\mathcal{B}$
2: $s = xy$ in $\mathcal{A} \cup a_r$
3: $u = s(-N^{-1})$ in $\mathcal{B}$
4: u in $\mathcal{A} \cup a_r \Leftarrow \text{BE1}(u \text{ in } \mathcal{B})$
5: $t = uN$ in $\mathcal{A} \cup a_r$
6: $v = s + t$ in $\mathcal{A} \cup a_r$
7: $w = vB_A^{-1}$ in $\mathcal{A} \cup a_r$
8: w in $\mathcal{B} \Leftarrow \text{BE2}(w \text{ in } \mathcal{A} \cup a_r)$

With BBEs $B > (k+1)^2 N$ and $A > (k+1)N$

Algorithm 4: Proposed RNS Montgomery multiplication

Input: $\hat{x}$, $\hat{y}$, and N in $\mathcal{A} \cup \mathcal{B} \cup a_r$, such that[1] $A = \prod_{j=1}^{k} a_j$, $B = \prod_{i=1}^{k} b_i$, and $\gcd(A, B) = 1$, $\gcd(N, B) = 1$, $0 \leq xy < NB$, $B > 4N$, and $A > 2N$

Output: $w \equiv xyB_N^{-1} \pmod{N}, w < 2N$ in $\mathcal{B}$, $\hat{w} \equiv xyB_N^{-1}A_j^{-1} \pmod{N}$ in $\mathcal{A} \cup a_r$

Precomputation:

1: $s = xy$ in $\mathcal{B}$

2: $\hat{s} = \hat{x}\hat{y}$ in $\mathcal{A} \cup a_r$

-

3: $\hat{w}$ in $\mathcal{A} \cup a_r \Leftarrow$ BE1(s in $\mathcal{B}$, $\hat{s}$ in $\mathcal{A}$)

-

-

-

4: w in $\mathcal{B} \Leftarrow$ BE2($\hat{w}$ in $\mathcal{A} \cup a_r$)

Figure 2. Comparison between the proposed RNS Montgomery multiplication and the state-of-the-art algorithm

The relevant characteristics of MM implementation in RNS (Fig. 2) are described as the following. Step 3 is only performed on B, so that the modular reduction by B does not require additional operations. After the modular reduction, the BE to A of step 4 is required, since a subsequent step requires a different base in order to perform a division by B and gcd (A; B) = 1. The multiplication in step 5 and the addition in step 6 are only performed on A, since the result of the multiplication in B is equal to the additive inverse on B of the result of step 1 and so the result of the addition in B is 0. The multiplication by B1 in step 7 is only performed on A, since gcd (B; B) 6 = 1. The last operation is the BE to B, so that the result can be used as input for other MMs.

In [7] and [4], the respective authors have proposed two different techniques to perform the BE. Both techniques are based on (1), but avoid the last modular reduction by A in order to save computational effort. In [4], the final modular reduction of both BEs required by the RNS MM are replaced by an approximation and a correction. In [7], the final modular reduction of the first BE is not executed, so B is added to the correct result. However, the second BE gives the exact value due to a correction executed at the end of the summation of multiplications. The adopted correction technique has been presented in [8], and it requires that all the values calculated on A are also calculated on an additional base element ar.

IV. Algorithm Analysis

In this section, the proposed approach is evaluated and compared with the state-of-the-art algorithms. The analysis is focused on the MM, which requires the majority of the total computational time.

A. Number of modular multiplications

Both the approaches in [4] and [9] have been evaluated by the respective authors according to the number of modular multiplications required. Table II reports the comparison of the proposed RNS MM algorithm with the previous ones. It can be easily seen that the algorithm presented in [9] achieves a reduction of k modular multiplications with respect to [4], whereas the proposed algorithm allows a further saving of 3k modular multiplications.

B. Analysis and classification of the required multiplications

As shown in Table II, the described algorithms require $2k^2$ and between 5k to 9k modular multiplications. All the necessary multiplications (not considering the BE correction) are listed and classified in Table III. Since each operation is performed on k base elements and only one multiplication is performed on a larger base, up to k cells can perform in parallel the required operations. Thus, the multiplications shown in Table III are organized in 2k + 8 multiplication steps, composed by k parallel multiplications. The multiplication steps are classified according to the opportunity of parallelization and pipelining. These aspects are of paramount importance, since different multiplication steps can require a different number of execution steps. The identified types of multiplication steps are:

- full, where the beginning of the operation must wait for the completion of the previous operation that calculates an input value; the number of required execution steps is p, where p corresponds to the number of pipeline stages;
- parallelizable, where a group of operations can be executed in parallel, or the first operation can be executed as full, and the subsequent ones can be

- pipelined; full parallelizable, where an operation can be executed in parallel to the previous and/or to the subsequent one requiring 0 execution steps.

V. Conclusion

In this paper a novel RNS Montgomery exponentiation algorithm is proposed. The algorithm is presented in two versions, targeted to the BE approaches adopted in [4] and in [9], respectively. The architecture proposed in [5], that is compliant with the approach in [4], has been used as a reference point for the design of a new architecture suitable for the method adopted in [9]. An algorithmic analysis has shown that the proposed approach is capable of providing a reduction of 4p b1=(p + M 1)c steps over the 2dk=M e 2 + b1=(p + M 1)c +8p required by each RNS MM (without considering the BE correction). Then, an architectural analysis has shown that, with the BE proposed in [4], the total number of steps and the reduction are the same, whereas with the BE approach used in [9], the reduction is the same but the MM requires one additional step. According to the algorithmic characteristics described in [4], [9], and to the architectural features described in [5], the delay reduction is equal to 13:6% or 13:4% depending on whether they BE adopted in [4] or in [9] is used, respectively.

REFERENCES

[1] N. Szabo and R. Tanaka, Residue arithmetic and its applications to computer technology. McGraw-Hill, 1967.

[2] R. Szerwinski and T. G ¨ uneysu, "Exploiting the power of GPUs for asymmetric cryptography," Cryptographic Hardware and Embedded Systems–CHES 2008, pp. 79–99, 2008.

[3] P. Montgomery, "Modular multiplication without trial division," Mathematics of computation, vol. 44, no. 170, pp. 519– 521, 1985.

[4] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel Montgomery multiplication," in Advances in Cryptology EUROCRYPT 2000, ser. LNCS. Springer Berlin / Heidelberg, 2000, pp. 523–538.

[5] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, "Implementation of RSA algorithm based on RNS Montgomery multiplication," in Cryptographic Hardware and Embedded Systems CHES 2001, ser. LNCS. Springer Berlin / Heidelberg, 2001, pp. 364–376.

[6] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "An RNS Montgomery modular multiplication algorithm," Computers, IEEE Transactions on, vol. 47, no. 7, pp. 766 – 776, jul. 1998.

[7] J.-C. Bajard, L. S. Didier, and P. Kornerup, "Modular multiplication and base extensions in residue number systems," in Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on, 2001, pp. 59–65.

[8] A. Shenoy and R. Kumaresan, "Fast base extension using a redundant modulus in RNS," Computers, IEEE Transactions on, vol. 38, no. 2, pp. 292–297, Feb 1989.

[9] J.-C. Bajard and L. Imbert, "A full RNS implementation of RSA," Computers, IEEE Transactions on, vol. 53, no. 6, pp. 769–774, June 2004.

[10] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over Fp," in Workshop on Cryptographic Hardware and Embedded Systems 2010 (CHES 2010), ser. LNCS. Springer Berlin / Heidelberg, 2010, pp. 48–64.

[11] F. Gandino, F. Lamberti, J.-C. Bajard, and P. Montuschi, "Preprocessing in RNS Montgomery multiplication," Tech. Rep., 2010.

[12] M. Pohst and H. Zassenhaus, Eds., Algorithmic algebraic number theory. New York, NY, USA: Cambridge UniversityPress, 1989, ch. 2.2.5.

[13] J. Bajard, N. Meloni, and T. Plantard, "Efficient RNS bases for cryptography," in IMACS'05 : World Congress: Scientific Computation, Applied Mathematics and Simulation, July 2005.

[14] D. Gajski, Principles of Digital Design. Prentice-Hall, 1997.