RNS bases and conversions

Jean-Claude Bajard and Thomas Plantard
LIRMM UMR 5506, University of Montpellier 2, France, {bajard,plantard}@lirmm.fr

ABSTRACT

Residue Number Systems (RNS) allow the distribution of large dynamic range computations over small modular rings, which allows the speed up of computations. This feature is well known, and already used in both DSP and cryptography. Most of implementations use RNS bases of three elements to reduce the complexity of conversions, but if can increase the number of RNS modular computational channels, then we are able to compute over smaller rings and thus further increase the speed of computation. In this paper, we deal with conversion from RNS to RNS or RNS to standard representations of numbers. We find, in the literature, two classes of conversion algorithms: those directly based on the chinese remainder theorem and those which use an intermediate Mixed Radix representation. We analyze these two different methods, show where the choice of the base is important and discuss the base selection criteria. We deduce that MRS conversions offer more possibilities than the CRT conversions. We provide features of RNS bases which provide low complexity of both RNS computation and conversion. We introduce some examples of bases well suited for cryptography applications.

Keywords: RNS, MRS, modular reduction, division by a constant, bases conversion.

1. INTRODUCTION

Original works on modular arithmetic are very old. The Chinese Remainder Theorem was first proposed around the fifth century by Sun Tsŭ.[1] But the use of this arithmetic to represent numbers was introduced only in 1959 by H.L. Garner.[2]

Theorem 1.1 (Chinese Remainder Theorem).

We consider a n-uple of coprime numbers $(m_1, m_2, \ldots, m_n)$. *We note* $M = \prod_{i=1}^{n} m_i$*, If we consider the n-uple* $(x_1, x_2, \ldots, x_n)$ *of integer such that* $x_i < m_i$*. Then there exits an unique* X *which verifies:*

$$0 \leq X < M \quad \textit{and} \quad x_i = X \bmod m_i = |X|_{m_i} \quad \textit{pour} \;\; 1 \leq i \leq n$$

The n-uple $(m_1, m_2, \ldots, m_n)$ of coprimes is generally called RNS basis.

The main interest of the Residue Number Systems is to distribute integer operations on evaluations with the residues values. Thus an operation with large integers is made on the residues which are small numbers and where computations can be executed independently for each modulo allowing a complete parallelization of the calculus.

Example 1. *We consider the coprime n-uple* $(255, 256, 257)$*, thus* $M = 16776960$

$a = 10000$ *is represented by* $(55, 16, 234)$ *and* $b = 300$ *by* $(45, 44, 43)$

We can verify that $a \times b = 3000000$ *is represented by* $(180, 192, 39)$ *where* $180 = 55 * 45 \bmod 255$*,* $192 = 16 * 44 \bmod 256$ *and* $39 = 234 * 43 \bmod 257$.

We find, in the literature, different kinds of applications which use RNS. Most of the papers are dedicated to signal processing.[3–6] In fact, the transformations used in this domain are based on additions and multiplications which are efficient in RNS. They just need three moduli to represent the values manipulated during the evaluation. Use of RNS appears also in discret geometry[7] or in theoretical approaches.[8] The last domain where those systems are helpful, is cryptography[9].[10] The numbers used are huge, 160 bits for Elliptic Curves Cryptography (ECC) and 1024 bits for RSA. These systems offer a good alternative for implementing parallel computing for such numbers.

Though addition and multiplication are easily parallelizable with RNS, some operations, like division or modular multiplication, need conversions from RNS to binary or RNS to an other RNS. This transformation is not trivial, and most of the algorithms depend of it. It is the reason why we focus on this operation, in particular from RNS to RNS which is used in cryptography,[11] analyzing the influence of the choice of the coprime n-uple defining the RNS basis.

We first present the classical conversion algorithms from RNS to RNS or other representations. Then, we analyze these methods through the point of view of the RNS bases. We propose some criteria of bases selection to obtain efficient conversions. And we end, in the conclusion, with a discussion around RNS.

2. SOME GENERALITIES ON THE CONVERSION

When we propose to use RNS to represent numbers, the first question is: What is the cost of the conversion to and from these systems? It is clear that the translation from a classical representation to RNS is identical to a modular reduction. One of the most trivial way is to store all the powers of the radix modulo each element of the RNS basis.[12–14] Thus, if we consider the RNS basis $(m_1, m_2, \ldots, m_n)$ with $M = \prod_{i=1}^{n} m_i < \beta^T$ where β is the radix of the classical representation, then we have to store Tn values of size T/n digits, in other words T^2 digits, for using the following formula:

$$x_i = |X|_{m_i} = \left| \sum_{j=1}^{T} \chi_j \times \beta^j \right|_{m_i} = \left| \sum_{j=1}^{T} \chi_j \times \left|\beta^j\right|_{m_i} \right|_{m_i} \tag{1}$$

We assume that the modular multiplier-adder is the basic operator of RNS computing. Hence, the conversion from β representation to RNS can be done in T iterations. Now if we consider a full parallelization, with n modular multiplier-adders, this conversion can be logarithmic (ie $O(\log(T))$). But in this paper we focus our attention to the conversion from RNS to RNS or to β representation.

To simplify our future discussions, we consider that $\beta = 2$, that all the m_i have the same number t of bits, and for the RNS to RNS conversion the two bases have the same number of moduli.

We note $\mathcal{B} = (m_1, m_2, \ldots, m_n)$ and $\widetilde{\mathcal{B}} = (\widetilde{m_1}, \widetilde{m_2}, \ldots, \widetilde{m_n})$ the two RNS bases which we will use.

2.1. From the Chinese Remainder Theorem

In the proof of the Chinese Remainder Theorem we can show, that a solution of the following system:

$$\left\{ \begin{array}{ccccc} X & \equiv & x_1 & \pmod{m_1} \\ \ldots & \ldots & & \ldots \\ X & \equiv & x_n & \pmod{m_n} \end{array} \right. \tag{2}$$

can be constructed from the formula:

$$X = \left| x_1 \left|M_1\right|_{m_1}^{-1} M_1 + x_2 \left|M_2\right|_{m_2}^{-1} M_2 + \ldots + x_n \left|M_n\right|_{m_n}^{-1} M_n \right|_M \tag{3}$$

Where, $M_i = \dfrac{M}{m_i}$ for $1 \le i \le n$ and $|M_i|_{m_i}^{-1}$ represents the inverse of M_i modulo m_i .

It is obvious that the evaluation of a such expression is heavy. To simplify a little bit the equation (3) we consider $\alpha_i = x_i \left|M_i\right|_{m_i}^{-1} \bmod m_i$, so we obtain a new expression for X:

$$X = \left(\sum_{i=1}^{n} \alpha_i M_i \right) \bmod M \tag{4}$$

Despite this transformation, each term stays a great value of the same range than M. Hence we deduce that the cost of the conversion from RNS to binary representation is equivalent to n modular products modulo an m_i, n products of a large number by a small one, and $n - 1$ additions of large numbers and a reduction modulo M.

Here again, we can parallelize to obtain a logarithmic time complexity, but the operators will be huge and not realistic. Now, if the number n of moduli is small, like for some DSP, the equation (4) can offer a good solution.[15, 16]

The conversion RNS to RNS is similar, we will make the same kind of operations on each modulo of the new basis $\widetilde{\mathcal{B}}$, the equation (3) becomes for $j \in \{1, ..., n\}$:

$$X = \left| x_1 \left|M_1\right|_{m_1}^{-1} \left|M_1\right|_{\widetilde{m_j}} + x_2 \left|M_2\right|_{m_2}^{-1} \left|M_2\right|_{\widetilde{m_j}} + \ldots + x_n \left|M_n\right|_{m_n}^{-1} \left|M_n\right|_{\widetilde{m_j}} \right|_{\widetilde{m_j}} \qquad (5)$$

Thus, this evaluation needs n modular products modulo an m_i, and for each modulo of the new RNS basis n modular products (of small numbers), and $n-1$ modular additions of residues and a reduction modulo M. So if we do not take into account the final reduction, we need $n(n+1)$ modular products and $n(n-1)$ modular additions on residues. We have to notice that we can have a great degree of parallelization.

Now, the reduction modulo M is the main drawback of this approach. In fact most of the method proposed to find α such that:

$$X + \alpha M = \left(\sum_{i=1}^{n} \alpha_i M_i \right) \qquad (6)$$

Hence, when α is determined, αM can be subtracted from the second member of equation (6). This remark is also available for the RNS to RNS conversion.

A. P. Shenoy and R. Kumaresan have proposed to use an extra modulo to find this factor.[17] In this method all the computing is done with integers. An other point of view is to find an approximation of α with floating point cell.[9, 18, 19]

2.1.1. Shenoy Kumaresan

This method is based on the knowledge of the residue modulo an extra co-prime m_e. So X is define by its RNS representation $(x_1, x_2, \ldots, x_n)$, and we know $x_e = X \bmod m_e$. Hence, from expression (6) we can deduce χ the residue of $X + \alpha M$ modulo m_e:

$$\chi = (X + \alpha M) \bmod m_e = \left| \sum_{i=1}^{n} \alpha_i \left|M_i\right|_{m_e} \right|_{m_e} \qquad (7)$$

Thus, we have

$$\alpha = (\chi - x_e) \times \left|M\right|_{m_e}^{-1} \bmod m_e \qquad (8)$$

This evaluation needs n modular products and $n-1$ modular additions on residues. Then the final reduction is done with a product by α of M ($\left|M\right|_{\widetilde{m_j}}$ for RNS to RNS) and a subtraction.

But this condition on the extra modulo cannot be always satisfied. It is not obvious of maintaining the knowledge of the residue modulo m_e during the different calculuses. So, we must consider an approach available just with the RNS representation.

2.1.2. Floating point approach

In this approach we factorize the equation (6), where appears a floating point value.

$$X + \alpha M = \left(\sum_{i=1}^{n} \alpha_i \frac{1}{m_i} \right) M \qquad (9)$$

Thus α is the integer part of $\sum_{i=1}^{n} \alpha_i \frac{1}{m_i}$. But this evaluation cannot be realized with exact calculuses, and the different references[9, 18, 19] obtain $\alpha \pm 1$. The complexities are close to those of Shenoy-Kumaresan, but with a specific floating point cell. So, if we want a pure integer implementation we will use an auxiliary representation like mixed radix systems.

2.2. Via a mixed radix system

Mixed Radix Systems MRS can be seen as multi-radixes representation. Like for RNS, we define as a MRS basis a set of integers $(m_1, m_2, \dots, m_n)$ (here they are not necessary coprime), and an integer X, inferior to M, is represented by a n-uple $(x'_1, ..., x'_n)$, with $x'_i < m_i$ for $i \in \{1, ..., n\}$, such that:

$$X = x'_1 + x'_2 m_1 + x'_3 m_1 m_2 + ... + x'_n m_1 ... m_{n-1} \tag{10}$$

The conversion from RNS to MRS is a classical greedy algorithm where we consider the residue modulo the radix and then we iterate this process with the quotient. We present two different ways to obtain this representation one which can be parallelized and an other purely sequential.

We notice that the construction of X from the MRS representation can be done with a classical Horner scheme, with n products by an m_i and $n-1$ additions (for a conversion RNS to RNS we get the same complexity with modular operations on each modulus of the new RNS basis).

2.2.1. Parallel algorithm

A naive approach[20] consists in applying to the RNS representation of X , the extraction of a residue and the division by the radix (which is different at each step).

We note $m_{i,j}^{-1}$ the inverse of m_i modulo m_j such that: $m_i.m_{i,j}^{-1} \equiv 1[m_j]$. Thus the conversion can be described by the following equations:

$$\begin{cases} x'_1 = x_1 \bmod m_1 \\ x'_2 = (x_2 - x'_1) m_{1,2}^{-1} \bmod m_2 \\ x'_3 = ((x_3 - x'_1) m_{1,3}^{-1} - x'_2) m_{2,3}^{-1} \bmod m_3 \\ \vdots \\ x'_n = (\cdots (x_n - x'_1) m_{1,n}^{-1} - x'_2) m_{2,n}^{-1}) - \cdots - x'_{n-1}) m_{n-1,n}^{-1} \bmod m_n \end{cases} \tag{11}$$

We can remark that x_1, the residue modulo m_1, is subtracted from the other residues which are then divided by m_1 (ie multiplied by the inverse of m_1 for each modulo), this operation is iterated with x'_2 etc.

The complexity of this conversion RNS to MRS is equivalent to $n(n-1)/2$ modular subtractions and products, which can be parallelized to obtain the result in $n-1$ steps.

The drawback of this approach is due to the storage of the $n(n-1)/2$ inverses needed for the conversion. An other method, purely sequential, allows to store only $n-1$ values.

2.2.2. Sequential approach

In the equations (11) we observe that we can factorize all the inverses.[1] Thus those equations becomes:

$$\begin{cases} x'_1 = x_1 \bmod m_1 \\ x'_2 = (x_2 - x'_1) \left|m_2\right|_{m_2}^{-1} \bmod m_2 \\ x'_3 = ((x_3 - x'_1) - x'_2 m_1) \left|m_1 m_2\right|_{m_3}^{-1} \bmod m_3 \\ \vdots \\ x'_n = ((x_n - x'_1) - m_1(x'_2 - m_2(x'_3 - \cdots - m_{n-3}(x'_{n-2} - m_{n-2} x'_{n-1})...) \left|m_1 ... m_{n-1}\right|_{m_n}^{-1} \bmod m_n \end{cases} \tag{12}$$

The global complexity is the same than in the previous approach, but here we observe that each evaluation can begin only when all the previous are ended. It is not possible to parallelize.

3. BASIS SELECTION CRITERIA AND ALGORITHMS

After those few recalls, we arrive to our main problem which is, how can we define a "good" RNS basis. It is clear that, if we just need a basis with few elements, the easiest is to choose a set like $\{2^t - 1, 2^t, 2^t + 1\}$ which is very used in digital signal processing. But, in the literature,[15, 21] most of the operators use the CRT approach. We are not convinced, that it is the best choice of algorithm. If we observe the equation (11) we can consider $m_1 = 2^t + 1$, $m_2 = 2^t - 1$, and $m_3 = 2^t$, then , since $m_1 \bmod m_2 = 2$, $m_1 \bmod m_3 = 1$ and $m_2 \bmod m_3 = -1$, we deduce that $m_{1,2}^{-1} = 2^{n-1}$ $m_{1,3}^{-1} = 1$ and $m_{2,3}^{-1} = -1$. Thus the conversion to MRS can be reduced to one shift and four additions, then conversion to binary needs at most two shifts and four additions.

3.1. About MRS to a new RNS basis

The first observation we can made looking equation (10), is that if we use the Horner scheme, we will multiply by an $m_i \bmod \widetilde{m_j}$, for $i, j \in \{1, ..., n\}$. We recall that the elements of the RNS bases of this study are t bits-integers, thus the value $|m_i - \widetilde{m_j}|$ is less than $\widetilde{m_j}$ and can be used for the evaluation of:

$$X \bmod \widetilde{m_j} = \widetilde{x_j} = [x'_1 + (m_1 - \widetilde{m_j})(x'_2 + (m_2 - \widetilde{m_j})(x'_3 + ... + (m_{n-1} - \widetilde{m_j})x'_n)...)] \bmod \widetilde{m_j} \qquad (13)$$

Hence, we replace a classical modular product by one with an operand of the size of the maximal difference between two RNS bases elements.

Now the problem is to find RNS bases such that those differences are minimal. To simplify this study we can consider that the two bases are ordered such that: $m_1 < m_2 < ... < m_n < \widetilde{m_1} < \widetilde{m_2} < ... < \widetilde{m_n}$. Thus the maximal difference is given by $\widehat{d} = \widetilde{m_n} - m_1$. So, the question is: What is the minimal interval of t-bits integers, which contains $2n$ coprime numbers? Or, how many coprimes can we get in an interval of size $\widehat{d}$?

We know that the function $\pi(m)$ which gives the number of prime numbers less than m, can be approximated by $\frac{m}{\log(m)}$. Thus, as showed in table 1 the expression $\frac{\widehat{d}}{\log(\widehat{d})}$ gives a good idea of the range of the number of coprimes between m and $m+\widehat{d}$. We note that,if m is chosen equal to a power of 2, the two bases can be considered composed

$\widehat{d}$	2^8	2^{10}	2^8	2^{10}	2^8	2^{10}
m	2^{32}	2^{32}	2^{64}	2^{64}	2^{128}	2^{128}
$\frac{\widehat{d}}{\log(\widehat{d})}$	46	147	46	147	46	147
# coprimes found	39	117	44	124	41	121

Table 1. coprimes found between m and $m + d$ with a trivial algorithm

of pseudo-mersennes which are fine for the modular operations. But, we precise that the number of coprimes does not depend of this particularity, we can verify in table 1 that this value only depends of d. Furthermore, the main interest is that the modular products of equation (13) have one factor with at most $\log(\widehat{d})$ bits. Aftermath, those operations can be more than $t/\log(\widehat{d})$ times faster than a classical modular product. This remark is true as well for conversion to binary than for conversion to RNS. To have a realistic idea of the benefit we can get, we consider an example coming from cryptolography which uses huge numbers of for example 1024 bits. In the RNS modular algorithms for cryptography[11,22] we need 68 moduli of 32 bits, we can assume a maximal difference $\widehat{d}$ less than 2^{10}, thus we can expect a speed up of 3 which is consequent.

Example 2. *We can generate a set of* 68 *coprime numbers from* 4294967296 *to* 4294967783 *where the difference is bound by* 487*. This set is defined by the first value and the differences with it.*

{4294967296 1 3 5 7 9 13 15 21 25 27 33 37 55 61 63 75 81 85 91 93 97 105 111 123 133 145 153 163 175 177 181 201 207 211 217 231 243 247 253 261 265 267 273 295 301 307 313 321 327 331 343 351 357 375 385 387 391 397 405 415 427 445 453 463 481 483 487 }

Thus, in expression (13), the maximal value of the constant factors is inferior to 487 *which is a nine bits integer.*

Now, we will see if such bases have also some interests for conversion from RNS to MRS.

3.2. From RNS to MRS

It is obvious that if we want to exploit the properties of the previous choice of bases, we will want to use the sequential algorithm defined by equations (12), where each occurrence of m_i can be replaced, for an evaluation modulo m_j, by the difference $(m_i - m_j)$ which is a small value (ie less than $\widehat{d}$). In this case we will obtain the previous speed up for the $n(n-1)/2$ modular products by an m_i, but we keep the classical modular products for the $n - 1$ multiplications by an inverse.

Now if we consider the equations (11) of the parallel conversion, we can remark that we needs to multiply by the inverses of small numbers (which can be negative). But we know a reduction algorithm found by Montgomery,[23] which gives the modular product by an inverse.

So, the operations, which we have to evaluate, have the following form $y = x * |d|_m^{-1} \bmod m$ where x and m are two t-bits integers and d a small value (ie t/c bits number). The method proposed by P.l. Montgomery is based on the construction of a multiple of a value d by adding to x a multiple of m, thus this result can be divided by d to obtain a reduced value less than m.

Hence, we must construct k such that: $k < d$ and $x + km$ is a multiple of d, then, as the division of $x + km$ by d is exact, we obtain the expected value y. This is recapitulated in the following algorithm:

```
Algorithm 1 : Modular Division
Require: two integers $d, m$ with $0 < d << m$ and $gcd(m, d) = 1$
    three precomputed values $r_m, q_m$ and $I_m$ such that:
    $m = r_m + q_m d$ with $r_m < d$ and $I_m = (-m)^{-1} \bmod d$
    and an integer $x$, $0 \le x < m$
Ensure: an integer $y$ with $y = x/d \bmod m$
 1: find $r_x$ and $q_x$ such that: $x = r_x + q_x \times d$ with $0 \le r_x < d$
 2: $k \leftarrow r_x \times I_x \bmod d$
 3: $y \leftarrow (r_x + k \times r_m)/d + q_x + k \times q_m$
```

Now, we analyze the complexity of this algorithm step by step, we note $2^{\delta-1} \le d < 2^{\delta}$:

1. r_x and q_x are the results of an euclidian division where the divisor d is a small constant(δ-bits number) and the dividend is a t-bits integer.
2. r_x and I_x are smallest than d, so k can be obtained with a classical Barret modular multiplication[24] with δ-bits numbers.
3. Then we remark that $q_x + k \times q_m < m$ and $(r_x + k \times r_m) < d^2$ is a multiple of d. Hence, we need
    - one addition of t-bits numbers, one product of a δ-bits by a $(t - \delta)$-bits numbers,
    - one δ-bits multiplication, one 2δ-bits addition and one division of a 2δ-bits by a δ-bits number (we can use Barrett reduction algorithm),
    - and, to end one t-bits addition

If we use this algorithm for a conversion RNS to MRS, we must store r_m, q_m and I_m for each element of the basis, which will need $n^2 \times (t + \widehat{\delta})$ bits, where $\widehat{\delta}$ represents the maximal value of the δ.

3.2.1. Division by a small constant

We want to compute r_x and q_x such that: $x = r_x + q_x \times d$ with $0 \le r_x < d$.

As, d is a small constant, we can use a division algorithm adapted from Barrett method. The main idea of this approach is given by the following equations, where $f(y)$ is the fractional part of y:

$$\begin{aligned} x2^{k+1} &= d\tfrac{2^{k+\delta}}{d}\tfrac{x}{2^{\delta-1}} \\ &= d\left(\left\lfloor \tfrac{2^{k+\delta}}{d}\right\rfloor + f(\tfrac{2^{k+\delta}}{d})\right)\left(\left\lfloor \tfrac{x}{2^{\delta-1}}\right\rfloor + f(\tfrac{x}{2^{\delta-1}})\right) \\ &= d\left(\left\lfloor \tfrac{2^{k+\delta}}{d}\right\rfloor \left\lfloor \tfrac{x}{2^{\delta-1}}\right\rfloor\right) + d\left(f(\tfrac{2^{k+\delta}}{d})\left\lfloor \tfrac{x}{2^{\delta-1}}\right\rfloor + \left\lfloor \tfrac{2^{k+\delta}}{d}\right\rfloor f(\tfrac{x}{2^{\delta-1}}) + f(\tfrac{x}{2^{\delta-1}})f(\tfrac{2^{k+\delta}}{d})\right) \end{aligned} \tag{14}$$

Now, we remark that:

$$\begin{aligned}\left(f(\tfrac{2^{k+\delta}}{d})\left\lfloor\tfrac{x}{2^{\delta-1}}\right\rfloor + \left\lfloor\tfrac{2^{k+\delta}}{d}\right\rfloor f(\tfrac{x}{2^{\delta-1}}) + f(\tfrac{x}{2^{\delta-1}})f(\tfrac{2^{k+\delta}}{d})\right) &< \tfrac{x}{2^{\delta-1}} + \tfrac{2^{k+\delta}}{d} + 1 \\ &< 2^{k+1}\left(\tfrac{x}{2^{\delta+k}} + \tfrac{2^{\delta-1}}{d} + \tfrac{1}{2^{k+1}}\right)\end{aligned} \tag{15}$$

Hence, we deduce from (14) and (15) that if $x < 2^{\delta+k}$, then:

$$x2^{k+1} - d\left(\left\lfloor\frac{2^{k+\delta}}{d}\right\rfloor\left\lfloor\frac{x}{2^{\delta-1}}\right\rfloor\right) < 2^{k+1}\left(2 + \frac{1}{2^{k+1}}\right)d \tag{16}$$

Thus, we obtain the Barrett equation:

$$x - d\left\lfloor\frac{\left(\left\lfloor\frac{2^{k+\delta}}{d}\right\rfloor\left\lfloor\frac{x}{2^{\delta-1}}\right\rfloor\right)}{2^{k+1}}\right\rfloor < 3d \tag{17}$$

The Barrett algorithm $Barrett_k(x,d)$ returns (q,r) such that $q = \left\lfloor\frac{\left(\left\lfloor\frac{2^{k+\delta}}{d}\right\rfloor\left\lfloor\frac{x}{2^{\delta-1}}\right\rfloor\right)}{2^{k+1}}\right\rfloor$ and $r = x - dq$, for $x < 2^{\delta+k}$ and $2^{\delta-1} \leq d < 2^{\delta}$.

Algorithm 2 : $Barrett_k(x,d)$

```
Require: $x < 2^{\delta+k}$ and $2^{\delta-1} \leq d < 2^{\delta}$
    and a precomputed value $\left\lfloor\frac{2^{k+\delta}}{d}\right\rfloor$
Ensure: $(q,r)$ such that $r = x - dq < 3d$
 1: $q \leftarrow \left(\left\lfloor\frac{2^{k+\delta}}{d}\right\rfloor\left\lfloor\frac{x}{2^{\delta-1}}\right\rfloor\right)$
 2: $q \leftarrow \left\lfloor\frac{q}{2^{k+1}}\right\rfloor$
 3: $r \leftarrow x - qd \bmod 2^{\delta+1}$ (as , $r < 3d$)
```

The complexity analysis gives:

1. one product of two k-bits numbers,
2. one shift,
3. one product of $(\delta+1)$-bits numbers, and one subtraction of $(\delta+1)$-bits numbers.

Thus, if we suppose that $k = t - \delta$, as this algorithm gives q and r such that: $r = x - dq < 3d$, for determining r_x and q_x, we must compare r to d and, if r is greater than d, then we must subtract d to r and increment q, this operation can be useful twice, to obtain $r < d$, and thus we get $r_x = r$ and $q_x = q$.

If we consider that the product is quadratic (at least more than linear) in number of binary operations, this algorithm 2 uses at most $k^2 + \delta^2 + 3\delta + 2$ binary operations. Hence, it can be useful to split the input x to get less operations.

In algorithm 3, we have fixed k, and we consider R_1 the $k+\delta$ most significant digits of $R = R_1 2^{T-k-\delta} + R_0$, where R and T are respectively initialized to x and t. We apply $Barrett_k(R_1, d)$ which returns (r_{R_1}, q_{R_1}) such that $R_1 - dq_{R_1} = r_{R_1} < 3d$. R is set to $r_{R_1}2^{T-k-\delta} + R_0$ and Q to $q_{R_1} + Q2^{k-2}$. Then T is decreased of $k-2$, and we iterate again with the $k+\delta$ most significant digits of R to obtain $R < 3d$.

At the end of each path in the first "while" loop, we verify that $Qd2^{T-2-\delta} + R = x$. Hence when the algorithm ends, we obtain (r_x, q_x) with $r_x + q_x d = x$ and $0 \leq r_x < d$.

Algorithm 3 : Division by a small constant $DIV(x, d)$

```
Require: two integers $x < 2^t$ and $2^{\delta-1} \leq d < 2^\delta$
Ensure: two integers $r_x, q_x$ with $r_x + q_x d = x$ and $0 \leq r < d$
 1: $R \leftarrow x$
 2: $Q \leftarrow 0$
 3: $T \leftarrow t$
 4: while $T > \delta + 2$ do
 5:   if $T \geq k + \delta$ then
 6:     $R = R_1 2^{T-k-\delta} + R_0$
 7:     $(r_{R_1}, q_{R_1}) \leftarrow Barrett_k(R_1, d)$
 8:     $R \leftarrow r_{R_1} 2^{T-k-\delta} + R_0$
 9:     $Q \leftarrow q_{R_1} + Q2^{k-2}$
10:     $T \leftarrow T - k + 2$
11:   else
12:     $R = R_1$
13:     $(r_{R_1}, q_{R_1}) \leftarrow Barrett_k(R_1, d)$
14:     $R \leftarrow r_{R_1}$
15:     $Q \leftarrow q_{R_1} + Q2^{k-2}$
16:     $T \leftarrow \delta + 2$
17:   end if
18: end while
19: while $R \geq d$ do
20:   $R \leftarrow R - d$
21:   $Q \leftarrow Q + 1$
22: end while
23: $r_x \leftarrow R$, $q_x \leftarrow Q$
```

The number of iterations needed is equal to $\left\lceil \dfrac{(t-\delta-2)}{(k-2)} \right\rceil$. Thus, if we refer to the complexity of $Barrett_k(R_1, d)$, algorithm 3, which represents the first step of algorithm 1, needs $\left\lceil \dfrac{(t-\delta-2)}{(k-2)} \right\rceil (k^2 + \delta^2 + 3\delta + 4 + \log(t)) + 2\delta$ binary operations.

3.2.2. Analysis of this conversion from RNS to MRS

Other steps of algorithm 1 represent $3\delta^2 + \delta t + 10\delta + 2t$ binary operations (considering the complexity of Barrett algorithm[25]). The total complexity of our method to obtain the modular product by an inverse of a small number, is equivalent to $\left\lceil \dfrac{(t-\delta-2)}{(k-2)} \right\rceil (k^2 + \delta^2 + 3\delta + 4 + \log(t)) + 3\delta^2 + (t+12)\delta + 2t$.

We show in table 2 a comparison of the number of binary operations of our approach (we suppose $k = \delta$), with the Barrett algorithm in $2t^2 + 4t$ operations for the multiplication by a stored inverse.

d	2^8	2^{10}	2^8	2^{10}	2^8	2^{10}
m	2^{32}	2^{32}	2^{64}	2^{64}	2^{128}	2^{128}
Our approach	1252	1521	2386	2868	4219	4234
Barrett algo	2176	2176	8448	8448	33280	33280

Table 2. comparison of our approach to Barret algorithm (number of binary operations)

The ratio of the complexities of Barrett by our method is linear in function of t, for a fixed d, this means that the efficiency of our approach increases if we use large moduli. The complete conversion using equation (11) requires $\frac{n(n-1)}{2}$ calls to a multiplication by an inverse, so the speed-up obtained with our algorithm becomes really consequent.

4. CONCLUSION

We have seen in this paper that choosing successive coprime numbers can allow to construct efficient RNS implementations. The numbers of coprimes we can expect in an interval is close to the quotient of the size of this interval with the logarithm of this size. As all the values of an RNS bases are close, we can choose one reference element and define the others by their differences to it, thus we just use small values. With this choice of moduli, the storage is reduced to $nlog_2(d) + t$ bits for the m_i and the complexities of conversions are more efficient. In this paper we do not have taken into account the possibility of selecting randomized bases[22] which is interesting in cryptography. It is a new very interesting use of RNS, which needs to be studied in future works.

REFERENCES

1. D. Knuth, *Seminumerical Algorithms*, vol. 2 of *The Art of Computer Programming*, Addison-Wesley, 2 ed., 1981.
2. H. L. Garner, "The residue number system," *IRE Transactions on Electronic Computers* **EL-8**, pp. 140–147, June 1959.
3. M. H. Etzel and W. K. Jenkins, "The design of specialized residue classes for efficient recursive digital filter realization," *IEEE Transactions on Acoustics, Speech and Signal Processing* **ASSP-30**(3), pp. 370–380, 1982.
4. M. Soderstrand, W. Jenkins, G. Jullien, and F. Taylor, eds., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, (New-York), IEEE Press, 1986.
5. M. A. Bayoumi, G. A. Jullien, and W. C. Miller, "A look-up table VLSI design methodology for RNS structures used in DSP applications," *IEEE Transactions on Circuits and Systems* **CAS-34**(6), pp. 604–616, 1987.
6. W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory* **IT-22**(6), pp. 644–54, 1976.
7. H. Brönnimann, I. Z. Emiris, V. Pan, and S. Pion, "Computing exact geometric predicates using modular arithmetic with single precision," in *Proceedings of the 13th Annual Symposium on Computational Geometry*, pp. 174–182, ACM, (Nice, France), 1997.
8. P. W. Beame, S. A. Cook, and H. J. Hoover, "Log depth circuits for division and related problems," in *25th Annual Symposium on Fundations of Computer Science*, pp. 1–6, IEEE Computer Society Press, 1984.
9. K. C. Posch and R. Posch, "Modulo reduction in residue number systems," *IEEE Transaction on Parallel and Distributed Systems* **6**(5), pp. 449–454, 1995.
10. J.-C. Bajard, L.-S. Didier, and P. Kornerup, "An RNS montgomery modular multiplication algorithm," *IEEE Transactions on Computers* **47**(7), pp. 766–776, 1998.
11. J.-C. Bajard and L. Imbert, "A full rns implementation of rsa," *IEEE Transactions on Computers* **53**(6), pp. 769–774, 2004.
12. G. Alia and E. Martinelli, "On the lower bound to the VLSI complexity of number conversion from weighted to residue representation," *IEEE Transactions on Computers* **42**(8), pp. 962–967, 1993.
13. B. Parhami, "Optimal table-lookup schemes for binary-to-residue and residue-to-binary conversions," in *27th Asilomar Conference on Signals, Systems and Computers*, **1**, pp. 812–816, IEEE Computer Society Press, (Pacific Grove, USA), 1993.
14. C. N. Zhang, B. Shirazi, and D. Y. Y. Yun, "An efficient algorithm and parallel implementations for binary and residue number systems," *Journal of Symbolic Computation* **15**(4), pp. 451–462, 1993.
15. S. J. Piestrak, "Design of high-speed residue-to-binary number system converter based on chinese remainder theorem," in *ICCD 1994*, pp. 508–511.
16. R. Conway and J. Nelson, "New crt-based rns converter using restricted moduli set," *IEEE Transactions on Computers* **52**(5), pp. 572–578, 2003.
17. A. P. Shenoy and R. Kumaresan, "Fast base extension using a redundant modulus in RNS," *IEEE Transactions on Computer* **38**(2), pp. 292–296, 1989.
18. C. Y. Hung and B. Parhami, "An approximate sign detection method for residue numbers and its application to RNS division," *Computers and Mathematics with Applications* **27**, pp. 23–35, Feb. 1994.

19. J.-C. Bajard, L.-S. Didier, and J.-M. Muller, “A new euclidian division algorithm for residue number systems,” *Journal of VLSI Signal Processing, Image and Video Technology* . A paraître.
20. N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*, McGraw-Hill, 1967.
21. R. Conway and J. Nelson, “Fast converter for 3 moduli rns using new property of crt,” *IEEE Transactions on Computers* **48**(8), pp. 852–860, 1999.
22. J.-C. Bajard, L. Imbert, P. Liardet, and Y. Teglia, “Leak resistant arithmetic,” in *CHES 2004*,
23. P. Montgomery, “Modular multiplication without trial division,” *Mathematic of Computation* **44**, pp. 519–521, Apr. 1985.
24. P. Barrett, “Implementing the rivest, shamir and adleman public key encryption algorithm on a standard digital processor,” in *Advances in Cryptology, Proceedings of Crypto’86*, A. M. Odlyzko, ed., pp. 311–323, 1986.
25. A. Bosselaers, R. Govaerts, and J. Vandewalle, “Comparison of three modular reduction functions,” in *Advances in Cryptology, Proceedings Crypto’93, LNCS 773, D. Stinson, Ed., Springer-Verlag, 1994*, pp. 175–186.