Académie de Montpellier

Université Montpellier II

- sciences et techniques du languedoc -

Diplôme d'études approfondies

- INFORMATIQUE -

Étude des changements de base en RNS

Pascal GIORGI

Date de soutenance : 4 juillet 2001

Directeur de stage : Jean-Claude BAJARD

Remerciements

Dans un premier temps je souhaite remercier Jean-Claude Bajard pour l'intérêt qu'il a apporté à mon travail et pour m'avoir communiquer son envie de travailler dans l'arithmétique des ordinateurs. Je remercie fortement Fabien Rico qui m'a apporté sa connaissance et son expérience du domaine lorsque j'en avais besoin. Je remercie aussi le groupe de travail « T.A.L.N » et en particulier Mathieu Lafourcade, Violaine Prince et Didier Schwab pour m'avoir permis d'effectuer mes simulations « extrêmement longues » sur leur machine. Enfin je remercie mes collègues de bureau qui m'ont apporté leur calme, leur gentillesse et leur sérénité tout au long de mon stage. Je tiens aussi à remercier particulièrement Philippe Elbaz-Vincent du département de mathématiques de l'université Montpellier 2 de m'avoir accepté dans son groupe de travail de cryptographie.

Résumé

Les systèmes modulaires de représentation des nombres ont la particularité de pouvoir effectuer en parallèle des calculs sur des valeurs résiduelles. Le changement de base de ce système est une opération inhérente à beaucoup d'algorithmes utilisant cette représentation. Nous montrons ici que la construction de bases particulières visant à améliorer cette opération est une tâche fastidieuse et que de telles bases sont en très faible quantité. Nous mettons en avant des opérateurs modulaires sur des nombres spécifiques, dont l'intégration aux systèmes modulaires permet l'optimisation de certains algorithmes de ce système.

Mots-clés : modulaire, représentation, extension, opérateur modulaires, constantes multiplicatives

Abstract

The main feature of the residue number system is to allow a parallelisation on small residue number. Most of algorithms using these systems need, in their implementation, a base extension. We show in this paper that base built around typical number to improve base extension are in too few quantity to be interesting. We present few modular operators which allow to improve algorithms based on a residue number system.

Keywords : modular, representation, extension, modular operators, multiplicative-constants

Table des matières

Introduction

Le codage est une notion que l'homme connaît depuis toujours. La représentation d'une entité sous forme d'un code est à l'origine de la communication entre les hommes, puis a évolué vers la communication hommes-machines et enfin machines-machines. Cette notion de codage est un concept qui est la base de l'informatique et des mathématiques. La représentation des nombres est l'une des fonctions prépondérantes dans la mise en oeuvre de concepts dans ces domaines. Depuis toujours on cherche à améliorer la représentation des nombres afin de faciliter les différents calculs qui peuvent leurs être appliqués.

L'arithmétique modulaire, basée sur des travaux datant de plus de 18 siècles, émerge dans la représentation des nombres en 1959 dans la thèse de doctorat de *Garner*[Gar59]. Son principe se base sur le fait de pouvoir représenter des nombres par leurs restes modulo d'autres nombres et de pouvoir effectuer des calculs en parallèle sur ces petites valeurs. Ce système de représentation trouve son intêret dans le domaine de la cryptographie et du traitement du signal. L'utilisation en cryptographie de très grands nombres devient de plus en plus nécessaire pour assurer la robustesse des systèmes utilisés. En effet on ne peut pas affirmer qu'un système cryptographique est inviolable car il dépend fondamentalement de la capacité qu'ont les ordinateurs à effectuer des calculs le plus rapidement possible.

Nous définissons le système modulaire de représentation des nombres dans la section 2.1, et nous présentons les opérations servant à ce système. Certaines de ces opérations sont difficiles à mettre en oeuvre et empêchent la généralisation d'un tel système. Dans la section 2.2 nous montrons les difficultés que l'on peut avoir lorsqu'on effectue un changement de base modulaire. La plupart des systèmes cryptographiques utilisent soit l'exponentiation modulaire, soit la multiplication modulaire. La première de ces fonctions utilise la deuxième dans son implémentation. Donc la multiplication modulaire est une opération importante dans la mise en place de tels systèmes. Nous présentons dans la section 2.3 deux algorithmes de multiplication modulaire. Le chapitre 3 montre nos travaux de recherche effectués pour améliorer le changement de base modulaire dans le but d'optimiser l'algorithme de multiplication modulaire RNS présenté dans la section 2.3.2. Ces travaux ont redirigé nos intêret de recherche vers des algorithmes sur des opérateurs modulaires. Différentes approches et résultats sur cette recherche sont présentés dans le chapitre 4.

Chapitre 1

Arithmétique modulaire

1.1 Système de représentation de nombres

Les systèmes les plus courants sont ceux à numération de position. La base 10 est l'un de ces systèmes. Le désavantage de ces systèmes est qu'ils peuvent générer une propagation de retenues lors de l'addition et de la multiplication. L'utilisation de l'arithmétique modulaire par un système de représentation de nombres est une alternative à ce problème. Elle permet d'effectuer des additions et des multiplications rapidement sans aucune propagation de retenues. Lors de la multiplication on ne calcule plus les produits partiels nécessaires à la représentation à numération de position.

Le principe de ce système de représentation, appelé RNS[1], est d'utiliser un ensemble d'entiers, tous premiers entre eux. On appelle cet ensemble « base modulaire » ou « base RNS ».

Un nombre est représenté dans ce système par ses restes modulo les entiers de la base modulaire du système. Toutes les opérations de ce système s'effectueront sur les restes modulaires du nombre représenté.

Soit la base modulaire $(m_1, m_2, ..., m_n)$, un nombre X sera représenté dans cette base par $(x_1, x_2, ..., x_n)$ tels que :

$$\begin{aligned} x_1 &= X \bmod m_1 \\ x_2 &= X \bmod m_2 \\ &\vdots \\ x_n &= X \bmod m_3 \end{aligned}$$

Ce système est basé sur le théorème des restes chinois qui fournit une condition d'unicité sur la représentation des nombres.

[1]RNS : Residue Number System

Théorèmes 1 **(Théorème des restes chinois)**

Soient $m_1, m_2, ..., m_n$ *des entiers positifs premiers entre eux. Soit* $M = \prod_{i=1}^{n} m_i$ *il existe un unique entier* X *vérifiant les conditions suivantes :*

$$0 \leq X < M$$

$$\forall i \in \{1..n\}\ x_i = X \ mod\ m_i$$

$$X = (\sum_{i=1}^{n} \alpha_i M_i) \ mod\ M$$

$$Avec\ \alpha_i = x_i (M_i)_{m_i}^{-1} \ mod\ m_i \ et\ M_i = \frac{M}{m_i}.$$

Où $(M_i)_{m_i}^{-1}$ représente l'inverse modulaire de M_i modulo m_i

$$(M_i^{-1})_{m_i} M_i \equiv 1 \bmod m_i \tag{1.1}$$

On remarque que :

$$(\alpha_i M_i) \bmod m_j = x_i \text{ si } j = i$$

$$(\alpha_i M_i) \bmod m_j = 0 \text{ sinon}$$

La représentation d'un nombre X dans ce système avec une base RNS $B_m^n = (m_1, m_2, ...m_n)$ est notée $X_{RNS} = (x_1, x_2, ..., x_n)$.

1.1.1 Opérations simples

Soient $X_{RNS} = (x_1, x_2, ..., x_n)$ et $Y_{RNS} = (y_1, y_2, ..., y_n)$ deux nombres représentés dans la base RNS $B_m^n = (m_1, m_2, ...m_n)$.

- Si $X + Y < M$ l'addition de ces deux nombres s'écrit :

$$X_{RNS} + Y_{RNS} = ((x_1 + y_1)_{m_1}, (x_2 + y_2)_{m_2}, ..., (x_n + y_n)_{m_n})_{RNS} \tag{1.2}$$

- Si $X \times Y < M$ leur multiplication s'écrit :

$$X_{RNS} \times Y_{RNS} = ((x_1 \times y_1)_{m_1}, (x_2 \times y_2)_{m_2}, ..., (x_n \times y_n)_{m_n}))_{RNS} \tag{1.3}$$

Dans ces deux opérations, les calculs opérant sur les différentes composantes du nombre représenté en RNS sont indépendants les uns des autres. Ils sont donc facilement parallèlisables sur un calculateur adapté. Cependant ces deux opérations sont restreintes par les conditions du théorème des restes chinois (Cf 1.1). Le dépassement de capacité étant difficile à détecter, nous nous intéressons par la suite à des opérations modulo des moduli inférieurs à M. Cela permet de conserver le résultat toujours inférieur à M.

1.1.2 Comparaisons et divisions

La comparaison en RNS est une opération difficile. Il n'existe aucun moyen de connaître directement une information sur l'ordre de grandeur d'un nombre représenté en RNS. Pour comparer deux nombres dans ce système il faut les convertir dans une représentation à numération de position. Sous cette forme cette opération est intrinsèque à la représentation elle même. Néanmoins, une telle conversion reste une opération très coûteuse. Il existe des algorithmes de comparaison basés sur des fonctions diagonales [DIP93] et sur l'approximation du signe du nombre [HP94] qui donnent des résultats satisfaisants.

A l'instar de la comparaison, la division est une opération difficile. Dans la plupart des cas elle se ramène à des comparaisons ou au calcul du PGCD. Il existe tout de même un cas simple où la division RNS se réduit à une multiplication RNS :

Si le diviseur est premier avec $M = \prod_{i=1}^{n} m_i$ et que le dividende est un multiple du diviseur, alors le résultat de cette division est un entier. Cette opération est donc facilement réalisable dans la représentation RNS.

Soient $X_{RNS} = (x_1, x_2, ..., x_n)$ et $Y_{RNS} = (y_1, y_2, ..., y_n)$ deux nombres RNS vérifiant ces conditions et $Y_M^{-1} = ((y_1)_{m_1}^{-1}, (y_2)_{m_2}^{-1}, ..., (y_n)_{m_n}^{-1})$ alors la division s'écrit :

$$\frac{X_{RNS}}{Y_{RNS}} = X_{RNS} \times Y_M^{-1} \tag{1.4}$$

Dans le cas général, il existe dans la littérature divers algorithmes : (complexité en temps : $O(n \log n)$)[HP94],[Did98],[Gam91].

1.1.3 Base Mixte : Base de Cantor

Le système MRS[2] ou Bases de Cantor est un système de représentation de nombres à numération de position. Cette représentation est liée naturellement à la représentation RNS car elle utilise une base d'entiers servant à représenter un nombre.

Pour représenter un entier X dans une représentation MRS, on prend un ensemble de n entiers $(m_1, m_2, \cdots, m_n)$ qui constituera la base de la représentation. Il suffit ensuite de calculer l'ensemble de chiffres $(x'_1, x'_2, \cdots, x'_n)$ tel que pour tout i, $x'_i < m_i$ et tel que le nombre X s'écrit :

$$X_{MRS} = x'_0 + x'_1 m_0 + x'_2 m_0 m_1 + ... + x'_{n-1} \prod_{i=0}^{n-2} m_i \tag{1.5}$$

Cette représentation est utile à la représentation modulaire. Le passage du RNS au MRS peut être utiliser pour comparer deux nombres représentés en RNS. Pour cela il suffit de prendre pour base MRS la même base que celle utilisée dans

[2]MRS : Mixed Radix System

la représentation RNS du nombre à convertir. En prenant pour base $B_{n,m} = (m_1, m_2, \cdots, m_n)$ les chiffres x'_i de la représentation MRS de X s'écrivent alors de la manière suivante (Cf [Did98]). Soit $m_{i,j}^{-1}$ l'inverse de m_i modulo m_j tel que

$$m_i m_{i,j}^{-1} \equiv 1 \bmod m_j \tag{1.6}$$

Alors

$$\begin{aligned}
x'_1 &= x_1 \bmod m_1 \\
x'_2 &= (x_2 - x'_1) m_{1,2}^{-1} \bmod m_2 \\
x'_3 &= ((x_3 - x'_1) m_{1,3}^{-1} - x'_2) m_{2,3}^{-1} \bmod m_3 \\
&\vdots \\
x'_n &= (\cdots (x_n - x'_1) m_{1,n}^{-1} - x'_2) m_{2,n}^{-1}) - \cdots - x'_{n-1}) m_{n-1,n}^{-1} \bmod m_n
\end{aligned} \tag{1.7}$$

Finalement

$$X_{MRS} = (x'_1, x'_2, x'_3, \cdots, x'_n) \text{ dans la base } B_{n,m}$$

$$\updownarrow$$

$$X = x'_1 + x'_2 m_1 + x'_3 m_1 m_2 + \cdots + x'_n m_1 m_2 m_3 \cdots m_{n-1} \tag{1.8}$$

Le passage du RNS au MRS est utile pour effectuer des opérations de conversion et de comparaison mais aussi pour le changement de base RNS.

1.2 Extension et Changement de base

De nombreux algorithmes utilisant le système RNS [Gam91],[BDKR00],[BDK97] nécessitent une extension ou un changement de base. En considérant un nombre RNS $X_{RNS} = (x_1, x_2, \cdots, x_n)$ dans la base $B_n^m = (m_1, m_2, \cdots, m_n)$ l'extension de base consiste à calculer le reste modulaire de X par un nouveau modulo p tel que p soit premier avec M. Le changement de base prend en compte plusieurs extensions.. Il existe deux types d'algorithmes effectuant cette opérations. L'un consiste à convertir le nombre RNS dans un système à numération de position de type MRS, et l'autre se base sur le théorème des restes chinois. On considérera dans la suite le changement de base de X, représenté dans la base RNS $B_m^n = (m_1, m_2, \cdots, m_n)$, vers la base RNS $B_p^t = (p_1, p_2, \cdots, p_t)$. avec :

$$M = \prod_{i=1}^{n} m_i \text{ et } P = \prod_{i=1}^{t} p_i$$

1.2.1 Extension de base de Szabo et Tanaka

Cette extension présentée dans [ST67], se divise en deux étapes.

Dans un premier temps on effectue la conversion du nombre X_{RNS} en représentation MRS (avec la même base B_m^n), puis on calcule les restes modulaires de X_{MRS} dans la nouvelle base modulaire B_p^t. On considère l'ensemble $(x'_1, x'_2, \cdots, x'_n)$ étant la représentation MRS de X dans la base B_m^n. Les restes $|X|_{p_i}$ s'écrivent alors :

$$|X|_{p_i} = \left| |x'_1|_{p_i} + |x'_2|_{p_i} |m_1|_{p_i} + |x'_3|_{p_i} |m_1 m_2|_{p_i} + \cdot + |x'_n|_{p_i} \left| \prod_{j=1}^{n-1} m_j \right|_{p_i} \right|_{p_i} \tag{1.9}$$

Ce type de changement de base est coûteux car il fait appel à une conversion de représentation et à n produits modulaires et n additions modulaires. On peut stocker la valeurs des produits $\left| \prod_{j=1}^{n-1} m_j \right|_{p_i}$ pour chaque moduli de la nouvelle base, afin de minimiser le nombre de calculs. soit $p \times (n-1)$valeurs de $\lfloor \log_2 m_i \rfloor$ chiffres.

1.2.2 Extension de base de Shenoy et Kumaresan

Cette méthode se base sur le théorème des restes chinois (Cf page 5) qui est l'axiome de la représentation modulaire. Ce théorème dit :

$$X = \sum_{i=1}^{n} |x_i |M_i|_{m_i}^{-1}|_{m_i} M_i - \alpha M \tag{1.10}$$

α correspond au quotient de la division Euclidienne de $X = \sum_{i=1}^{n} x_i M_i (M_i)_{m_i}^{-1}$ par M. α est donc compris entre 0 et n Le calcul des nouveaux restes dans la base B_p^t revient à calculer :

$$|X|_{p_i} = \left| \sum_{i=1}^{n} \left| \left| x_i |M_i|_{m_i}^{-1} \right|_{m_i} M_i \right|_{p_i} - |\alpha M|_{p_i} \right|_{p_i} \tag{1.11}$$

Ce calcul n'est réalisable que si la valeur de α est connue. La représentation RNS et le Théorème des restes chinois ne donnent aucune information sur la valeur de ce α. Afin de calculer cette valeur *Shenoy et Kumaresan* ont proposé dans [SK89] d'utiliser un modulo redondant m_{n+1}, où le reste $|X|_{m_{n+1}}$ est connu. On choisi ce modulo de tel sorte que les calculs modulaires l'utilisant soient simplifiés et tel que $gcd(m_{n+1}, M) = 1$ et $m_{n+1} > n$. On peut alors calculer la valeur de α de la manière suivante :

$$\alpha = \left| (M)_{m_{n+1}}^{-1} \left(\sum_{i=1}^{n} \left| \left| x_i |M_i|_{m_i}^{-1} \right|_{m_i} M_i \right|_{m_{n+1}} - |X|_{m_{n+1}} \right) \right|_{m_{n+1}} \tag{1.12}$$

α se calcule à partir des restes de X dans la base $B_m^n \cup \{m_{n+1}\}$ et de l'inverse modulaire de M modulo M_{n+1}. Aucune conversion vers un autre système de représentation n'est nécessaire.

Ces deux algorithmes n'utilisent que des additions modulaires et des multiplications modulaires. L'efficacité de ces deux méthodes de changement de base est fortement liée à la capacité que l'on aura à calculer les différentes multiplications modulaires inhérentes à cette opération.

1.3 Multiplication modulaire

Cette opération majeure de l'arithmétique modulaire joue un rôle important dans la mise en place de systèmes cryptographiques à clefs publiques tel que les algorithmes de cryptage RSA[ARS78],[Yan00] ou Diffie et Hellman[DH76],[Yan00]. Il existe un algorithme de multiplication modulaire qui se base sur la notion de réduction et sur une représentation des nombres en numération de position [Mon85]. Cette méthode est efficace car la réduction utilisée évite les divisions, inhérentes au calcul modulaire, simplement en effectuant des décalages.. Cette méthode , appelée multiplication de Montgomery, fait référence dans le domaine de la cryptographie.

Toutefois les systèmes à numération de position ne sont pas les plus adaptés au calcul sur les grands entiers. Les systèmes modulaires offrent des possibilités de performances accrues, des algorithmes de multiplication modulaires ont été proposés dans de tels systèmes. [PP95][BDK97],[BDK01]. Nous présentons ici l'algorithme de Montgomery [Mon85] et celui proposé en 1997 par Bajard et al [BDK01].

1.3.1 Algorithme de Montgomery (1985)

Le but de cet algorithme est d'éviter les divisions lors du calcul modulaire. Pour cela l' algorithme calcule un nombre qui est un multiple de la base de représentation. Soient X, Y et M représentés dans la base b. On calcule Q tel que $XY + QM$ soit un multiple de b^n avec $B^n > M$. On pose $R = b^n$. Ainsi on calcule $(XY + QM)R^{-1} \equiv XYR^{-1} \bmod M$. La division par R se fait par un décalage et le modulo se fait par une comparaison.

Pour calculer $X \times Y$ mod M il faut utiliser deux fois cette méthode. La première fois en passant comme paramètres X,Y et M on calcule $XYR^{-1} \bmod M$. Au second passage on passe comme paramètres $|XYR^{-1}|_M$, $|R^2|_M$ et M. Ce qui donne comme résultat :

$$((X \times Y \times R^{-1} \text{ mod M}) \times (R^2 \text{ mod M}) \times R^{-1}) \bmod M = X \times Y \bmod M$$

Algorithme 1: Multiplication de Montgomery

Données : $X = (x_{n-1} \cdots x_1 x_0)_b$, $Y = (y_{n-1} \cdots y_1 y_0)_b$,
$M = (m_{n-1} \cdots m_1 m_0)_b$ *représentés dans la base b*

avec $0 \leq X, Y < M$, $R = b^n$ *avec gcd(M,b)=1* , $M' = -M^{-1}$ *mod b.*

Résultat : XYR^{-1} mod M

$A \longleftarrow 0$ (avec $S = (s_n s_{n-1} \cdots s_1 s_0)_b$)

pour *i de 0 à (n-1)* **faire**

  $q_i \longleftarrow (s_0 + x_i y_0) M'$ mod b

  $S \longleftarrow (S + x_i Y + q_i M)/b$

fin

si $S \geq m$ **alors**

  $S \longleftarrow S - m$

fin

Retourner(S)

Cette méthode effectue le calcul de XYR^{-1} en effectuant les produits partiels sur les chiffres de X représentés dans la base b et en accumulant les résultat dans S. Cela permet à chaque itération de conserver les propriétés suivantes :

$$S \equiv \sum_{j=0}^{i} \frac{x_j \times Y}{b^i} \bmod M \tag{1.13}$$

$$0 \leq S < 2M \tag{1.14}$$

Les divisions par b reviennent à faire un simple décalage et le fait que $\frac{XY}{b^i}$ soit strictement inférieur à $2M$ nous permet de calculer $\frac{XY}{b^i}$ mod M avec une seule comparaison.

1.3.2 Algorithme de Multiplication Modulaire en RNS

Cet algorithme se base sur le principe de l'algorithme de Montgomery, présenté précédemment, pour calculer ABM^{-1} mod N, avec A,B et N représentés en RNS et tel que $N < M$. On considère un système modulaire principal utilisant la base $B_m^n = (m_1, m_2, \cdots, m_n)$ et un système modulaire secondaire utilisant comme base $\tilde{B}_{\tilde{p}}^t = (\tilde{p}_1, \tilde{p}_2, \cdots, \tilde{p}_t)$ telle que :

$$\forall i \in (1, \cdots, n) \ , \forall j \in (1, \cdots, t) \quad GCD(m_i, \tilde{p}_j) = 1$$
$$\tilde{P} = \prod_{i=1}^{t} \tilde{p}_i > M$$

A,B et N sont exprimés dans les deux systèmes modulaires de telle sorte que $R = ABM^{-1}$ mod N soit exprimé dans les deux systèmes :

$$A = (a_1, a_2, \cdots, a_n, \tilde{a}_1, \tilde{a}_2, \cdots, \tilde{a}_t)$$

$$\begin{aligned} B &= (b_1, b_2, \cdots, b_n, \tilde{b}_1, \tilde{b}_2, \cdots, \tilde{b}_t) \\ N &= (n_1, n_2, \cdots, n_n, \tilde{n}_1, \tilde{n}_2, \cdots, \tilde{n}_t) \\ R &= (r_1, r_2, \cdots, r_n, \tilde{r}_1, \tilde{r}_2, \cdots, \tilde{r}_t) \end{aligned}$$

L'algorithme calcule $R = ABM^{-1} \bmod N$ en utilisant une valeur intermédiaire Q de sorte que $A \times B + Q \times N$ soit un multiple de M et $Q < M$. La représentation de R dans B_m^n est donc constituée uniquement de zéros. De ce fait il est possible d'obtenir facilement la représentation de Q dans la base B_m^n. Pour chaque composante de cette représentation on a :
$(a_i \times b_i + q_i \times n_i) \bmod m_i = 0$. Donc

$$q_i = (m_i - a_i \times b_i) \times (n_i)_{m_i}^{-1} \bmod m_i \tag{1.15}$$

Comme $A \times B + Q \times N$ est un multiple de M il faut calculer le résultat final $R = (A \times B + Q \times N) \times M^{-1}$ dans la base $\tilde{B}_{\tilde{p}}^t$. Étant donné que l'on connaît déjà la représentation de A,B et N dans la base $\tilde{B}_{\tilde{p}}^t$, il suffit simplement de convertir Q dans ce système. Une fois que R est connu dans la base $\tilde{B}_{\tilde{p}}^t$ il reste à le convertir dans la base B_m^n On obtient donc l'algorithme suivant :

Algorithme 2: Multiplication Modulaire RNS

Données : une base RNS $B_m^n = (m_1, m_2, \cdots, m_n)$
une base RNS $\tilde{B}_{\tilde{p}}^t = (\tilde{p}_1, \tilde{p}_2, \cdots, \tilde{p}_n)$
un entier N (exprimé dans les deux systèmes)
un entier A (exprimé dans les deux systèmes)
un entier B (exprimé dans les deux systèmes)
avec $A \times B < M \times N$

Résultat : Un entier $R < 2N$ (exprimé dans les deux systèmes) tel que $R \equiv ABM^{-1} \pmod N$

Méthode : : $Q \longleftarrow (-A \times_{RNS} B) \times_{RNS} N^{-1}$ dans B_m^n
Conversion de la représentation de Q depuis B_m^n vers $\tilde{B}_{\tilde{p}}^t$
$R \longleftarrow (A \times_{RNS} B + Q \times_{RNS} N) \times_{RNS} M^{-1}$ dans $\tilde{B}_{\tilde{m}}^n$
conversion de la représentation de R depuis $\tilde{B}_{\tilde{p}}^t$ vers B_m^n

---

Pour calculer $AB \bmod M$ avec cette méthode il faut refaire un deuxième passage. La seconde fois on multiplie le résultat obtenu par M^2 avec la même méthode. On obtient ainsi :

$$\begin{aligned} (ABM^{-1} \bmod N) \times (M^2 \bmod N) \times (M^{-1}) \bmod N &= ABMM^{-1} \bmod N \\ &= AB \bmod N \end{aligned}$$

L'opération la plus coûteuse de cet algorithme est la conversion d'un système modulaire vers un autre. L'efficacité de cet algorithme dépend donc de la capacité à effectuer les conversions le plus rapidement possible. Le choix des systèmes modulaires est une notion prépondérante dans la mise en oeuvre de cet algorithme. Il est évident que plus les constantes multiplicatives utilisées lors des conversions sont « simples » plus on peut espérer de bonnes performances.

Chapitre 2

Recherches de Bases Intéressantes

Dans le but d'optimiser la multiplication modulaire en RNS, nous nous intéressons à l'amélioration de l'algorithme de Shenoy et Kumaresan (Cf section 1.2.2) par la recherche de bases intéressantes. Nous considérons les constantes multiplicatives $\mu_i^{p_j}$ et deux bases modulaires B_m^n, B_p^t telles que :

$$B_m^n = (m_1, m_2, ..., m_n)$$

$$B_p^t = (p_1, p_2, ..., p_t)$$

$$\mu_i^{p_j} = \left| M_i (M_i)^{-1}_{m_i} \right|_{p_j} \text{ pour tout } i \in \{1, \cdots, n\} \text{ et } j \in \{1, \cdots, t\}$$

Afin d'optimiser le changement de base de B_m^n vers B_p^t on cherche à minimiser les valeurs de toutes les constantes multiplicatives. Pour cela nous nous intéressons à la recherche de bases permettant d'obtenir des valeurs très faibles (en nombre de bits) pour les $\mu_i^{p_j}$. Si de telles bases existent, nous pourrons effectuer le changement de base très rapidement, car la multiplication par les $\mu_i^{p_j}$ se résumera à quelques additions. L'approche statistique semble la plus adaptée pour savoir si l'obtention de telles valeurs est possible.

2.1 Étude Statistique

Le but de cette étude est de savoir, grâce à des simulations (implémentées en MAPLE et C++), s'il existe des bases répondant à notre critère et de les récupérer. En fait nous cherchons deux bases telles que les valeurs des $\mu_i^{p_j}$ soient d'au plus 4 bits.

2.1.1 Étude exhaustive

Pour que cette étude soit exhaustive nous prenons en considération toutes les bases modulaires possédant quatre moduli codés sur 10 bits. Nous prenons comme moduli l'ensemble des nombres premiers compris entre 2^{10} et 2^{11}. Cela permet de respecter les conditions du théorème des restes chinois (Cf page 5). Nous nous limitons à de telles bases car la combinatoire générée reste acceptable et peut être facilement implémentée sur machine pour une étude exhaustive. Notre étude se fait sur un ensemble de $91 * 10^6$ bases différentes.. Nous étudions ensuite toutes les valeurs des constantes multiplicatives générées par l'extension de toutes les bases possibles de taille 4 sur l'ensemble des moduli. De ce test ressort une courbe de la distribution des valeurs moyennes des constantes multiplicatives pour une base modulaire utilisant quatre moduli..

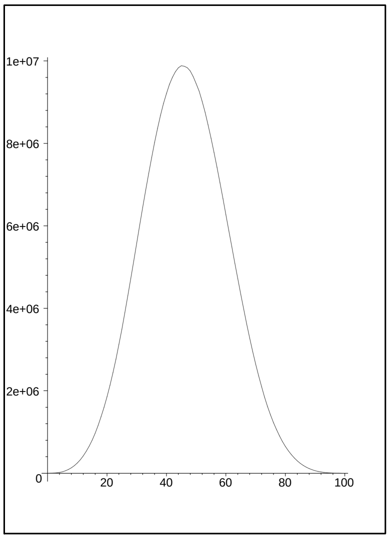


FIG. 2.1 – Distribution des valeurs moyennes des constantes multiplicatives (en % / max)

Les valeurs nous intéressant sont en très faible quantité, et donc pas suffisantes.

Le résultat suivant nous fournit deux base RNS, pouvant coder des nombres inférieur à 10^{12}, telles qu'elles génèrent les constantes multiplicatives les plus basses possible (en moyenne) pour un changement de base.

***Résultat simulation* 1** *$B_m^n = (1091, 1117, 1187, 1193)$ et $B_p^t = (1307, 1367, 1439, 1481)$ sont les deux meilleures bases de 4 moduli de 10 bits pour le changement de base. Elles fournissent les constantes multiplicatives suivantes :*

$$\mu^{1307} = (163, 8, 179, 17)$$
$$\mu^{1367} = (69, 29, 311, 133)$$
$$\mu^{1439} = (34, 64, 360, 128)$$

$$\mu^{1481} = (345, 25, 224, 39)$$

Le meilleur changement de base se fait avec des constantes multiplicatives bien trop élevées pour pouvoir obtenir une amélioration notable.

La courbe obtenue par ce test montre que les valeurs moyennes des constantes multiplicatives sont pratiquement toutes à la moitié du maximum. En fait nous obtenons une Gaussienne. Pour expliquer cela, il suffit de reprendre les données prises en compte dans le calcul de la distribution.

Nous évaluons la distribution de la moyenne de 4 constantes multiplicatives comprise entre 0 et au plus 2^{11}. La distribution des valeurs prises par les différentes constantes suit une loi uniforme. Cela est dû au fait que nous prenons en compte comme moduli un ensemble de nombres premiers (uniformité des restes modulaires). Le fait de prendre en considération la moyenne de 4 valeurs, ayant une distribution uniforme, nous ramène au théorème de la Limite Centrale. C'est-à-dire que la moyenne empirique des valeurs converge vers l'espérance des variables Dans notre cas les variables sont les constantes multiplicatives et d'après la loi des grands nombres l'espérance de ces variables est bien de $(\frac{max}{2})$.

Le résultat obtenu est bien la courbe d'une Gaussienne. En changeant quelques paramètres de notre test nous espérons obtenir une courbe plus proche d'une distribution uniforme. Cela permettrait de trouver des constantes ayant des valeurs très faibles. Pour cela il suffit d' augmenter la taille des moduli servant à représenter les bases. Cela augmente la combinatoire du nombre d'extensions possibles et de ce fait aplanit la courbe de la distribution. Le rapport entre la taille de la base et le nombre d'extensions possibles est un facteur prépondérant de la forme de la courbe de distribution.

2.1.2 Étude stochastique

Pour cette étude nous prenons en considération des bases de 10 moduli codés sur 32 bits. Nous sommes obligés pour des raisons de temps et de matériels d'effectuer cette étude sur des échantillons aléatoires car la combinatoire générée par une étude exhaustive est gigantesque. Le nombre de bases possibles est de l'ordre de 10^{97}. Cette étude reprend l'idée de la précédente mais le choix des moduli de la base se fait aléatoirement sur les nombres premiers compris entre 2^{32} et 2^{33}. Comme l'étude n'est pas exhaustive, un paramètre temporel est pris en compte dans cette simulation. Nous utilisons des temps proches de la dizaine de minutes afin de pouvoir effectuer plus de tests. Nous faisons varier au cours de ces simulations la taille de la base de façon décroissante afin d'aplanir la courbe de distribution. La meilleure courbe de distribution est obtenue pour une base de taille 2.

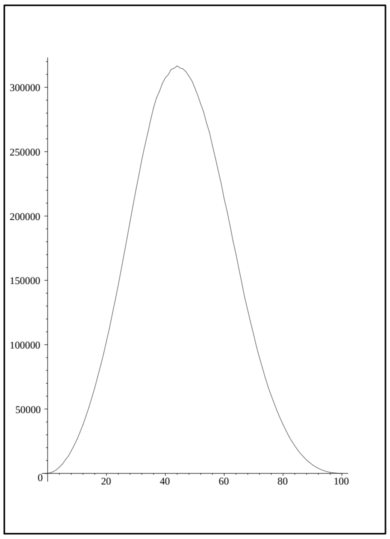


FIG. 2.2 – Distribution des valeurs moyennes des constantes multiplicatives (en % / max) pour des bases de taille 2

La courbe de distribution que nous obtenons possède toujours un maximum de valeurs au niveau de $\frac{max}{2}$. Nous avons tout de même réussi à aplanir un petit peu la forme de la courbe.

Le résultat suivant nous fournit une base RNS de 10 moduli de 32 bit, pouvant coder tous les nombres inférieur à 10^{97},et un moduli de 32 bits tel que l'extension de cette base vers ce moduli génère les plus petites constantes multiplicatives que nous avons pu tester (échantillonage aléatoire).

Résultat simulation 2 *La meilleure base que l'on a trouvée est :*
$Base = (7298095309, 4656564179, 6183656711, 4625970883, 6634405093, 4487632993, 6765757589, 5513626369, 6216182647, 5188670603)$
Le meilleur moduli pour étendre cette base est :
$Moduli = 4312403201$

Cette extension de base nous génère les constantes multiplicatives suivantes : $\mu^{4312403201} = (1505105449, 49, 628053581, 597329546, 1804107405, 3512147959, 271150152, 1145794824, 392376117, 1936184159)$

Nous constatons que les valeurs de ces constantes sont toutes plus proche de $\frac{max}{2} = 2^{16}$ *que de zéro. Cette extension de base, qui est la meilleure de nos tests, fournit des constantes multiplicatives trop importantes pour espérer un résultat satisfaisant.*

En continuant à faire varier les paramètres de nos test nous pouvons aplanir de plus en plus cette distribution. Pour cela il faut diminuer le rapport qu'il y a entre le nombre d'éléments de la base et le nombre de nombres premiers permettant de construire ces bases. Cela ne rentre pas dans le cadre de notre recherche car les paramètres nécessaires ne pourraient pas être utilisés par un système modulaire de représentation de nombres. Nous rappelons que le but d'un système modulaire est d'effectuer des calculs sur des petites valeurs. Ce qui n'est plus le cas quand on utilise des moduli trop grands.

2.1.3 Résultats

Nous ne pouvons pas affirmer qu'il n'existe pas des bases donnant des valeurs faibles aux constantes multiplicatives. Nous pouvons simplement dire que nous ne sommes pas en mesure de les générer automatiquement. De plus nous n'avons pas réussi lors de cette étude à mettre une seule base en valeur. La recherche de ces bases est une tâche complexe du fait qu'il en existe très peu. L'optique d'améliorer les changements de bases grâce à des bases générant des petites valeurs pour les constantes multiplicatives n'est pas une recherche intéressante.

Chapitre 3

Opérateurs Modulaires

Comme il est difficile de construire des bases améliorant le changement de base par la multiplication modulaire, nous nous intéressons dans ce chapitre, directement aux différents opérateurs modulaires possibles. Dans la suite nous nous intéressons à la multiplication modulaire en numération de position pour ces opérateurs. En prenant en compte des nombres particuliers, nous arrivons à effectuer la multiplication modulaire efficacement. Nous présentons ici quelques méthodes permettant d'améliorer le nombre d'opérations impliquées dans la multiplication modulaire pour les opérateurs modulaires que nous avons définis.

3.1 Multiplication modulaire avec précalcul

Cet algorithme travaille sur la représentation binaire des nombres impliqués dans la multiplication, ce qui permet de faire la multiplication "bit à bit" comme dans l'algorithme de multiplication modulaire de Montgomery [Mon85]. L'avantage majeur de cet algorithme, par rapport à celui de Montgomery en base 2, est qu'il ne prend en compte que les bits égaux à 1 dans la représentation binaire du multiplicateur. L'algorithme utilise le principe d'exponentiation rapide, c'est-à-dire que l'on se sert des calculs déjà effectués pour minimiser ceux qu'il reste à faire. De plus on introduit une notion de décalage modulaire avec précalcul.

3.1.1 Quelques définitions

Définition 1 *Le décalage modulaire est une multiplication modulaire par une puissance de la base, dans laquelle sont représentés les nombres.*

Définition 2 *La multiplication modulaire de deux nombres se réduit à une série de décalages modulaires et d'additions modulaires.*

Un exemple en base 2 :

$$|110 * 111|_{1010} = ||100 * 111|_{1010} + |10 * 111|_{1010}|_{1010}$$

Ce qui revient à faire deux décalages modulaires et une addition modulaire.

Définition 3 *La représentation d'un nombre A en binaire est de la forme* $(a_{n-1}a_{n-2}.......a_3a_2a_1a_0)$ *tel que :*

$$A = \sum_{i=0}^{n-1} a_i \times 2^i \text{ avec } n = \lfloor \log_2(A) \rfloor \text{ et } a_i \in \{0, 1\}.$$

3.1.2 Précalcul

L'algorithme a pour paramètres 3 nombres A, B et M et calcule le produit modulaire $|A * B|_M$. Nous considérerons par défaut que A est le multiplicateur et que B est le multiplicande.. Nous considérons aussi les nombres n_A, n_B, n_M qui sont (respectivement) la taille en nombre de bits de A ,B et M. . Nous calculons A' un tableau stockant de façon croissante les indices des a_i non nuls dans la représentation binaire de A. (ex $A' = [1, 2, 4]$ pour $A = 10110$). Soit n la taille de A', n correspond au nombre de bits égaux à 1 dans la représentation binaire de A. Ce tableau représente les valeurs des différents décalages à effectuer sur B dans la multiplication de B par A.

Nous calculons les 6 valeurs suivantes $Z_{111}, Z_{110}, Z_{101}, Z_{100}, Z_{11}, Z_{10}$ avec :

$$Z_{111} = |7 * 2^{n_M}|_M \text{ , } Z_{110} = |6 * 2^{n_M}|_M \text{ , } Z_{101} = |5 * 2^{n_M}|_M \text{ ,}$$

$$Z_{100} = |4 * 2^{n_M}|_M \text{ , } Z_{11} = |3 * 2^{n_M}|_M \text{ , } Z_{10} = |2 * 2^{n_M}|_M$$

Ces 6 valeurs seront utilisées lors des décalages modulaires de notre algorithme.

3.1.3 Principe de l'algorithme

Pour décaler modulairement B de A'[i] nous ne ne décalons pas simplement B de A'[i] puis calculons son reste modulaire par m. Nous bornons nos décalages de telle sorte que les nombres décalés aient au plus n_M+2 bits. Cela va nous permettre de calculer un décalage modulaire simplement par des additions, des décalages et une soustraction. Pour cela il suffit de considérer les $n_M + 2$ bits du nombre après le décalage et les de découper en deux parties. Une partie haute codée sur 3 bits correspondant aux bits n_M+2 , n_M+1 et n_M du nombre traité, et une partie basse correspondant aux $n_M - 1$ premiers bits de ce même nombre. Ainsi pour calculer son reste modulaire il suffit d'additionner le reste modulaire de sa partie haute, qui correspond à l'un de nos six nombres précalculés ($Z_{111}, Z_{110}, Z_{101} Z_{100}, Z_{11}, Z_{10}$), avec sa partie basse. Le résultat que nous obtenons est strictement inférieur à $2M$. Il suffit ensuite de soustraire, si nécessaire, M à ce résultat pour obtenir le résultat du décalage modulaire. Si le décalage modulaire fait que le nombre décalé possède au plus $n_M + 2$ bits, nous découpons ce décalage en plusieurs décalages consécutifs pour ne pas dépasser $n_M + 2$ bits. $n_M + 2$ bits. Nous ne nous soucions pas de calculer la soustraction à chaque fois, car le nombre obtenu est inférieur à $2M$. Il est donc strictement inférieur à 2^{n_M+1}. Nous pouvons donc continuer notre décalage sur ce nombre.

De plus pour calculer le décalage modulaire de B par A'[i], on se sert du résultat du décalage modulaire de B par $A'[i-1]$ que l'on décale modulairement de $A'[i] - A[i-1]$. Ce qui réduit la taille des décalages modulaires à effectuer. Ce résultat est dû au fait de la distributivité de la modularité par rapport à la multiplication et le fait que A' est ordonné de façon croissante.

Algorithme 3: Multiplication modulaire

Données : A , B , m

Résultat : $|A \times B|_m$

Précomputation : Z_{111} , Z_{110} , Z_{101} , Z_{100} , Z_{11} , Z_{10}

Initialisation : $S \longleftarrow 0$

$A' \longleftarrow Indice(A)$

$tmp \longleftarrow 0$

pour *i de 1 à Taille(A')* **faire**

$d \longleftarrow A'[i] - tmp$

$tmp \longleftarrow A'[i]$

tant que $d \neq 0$ **faire**

$B \longleftarrow$Décale(B, d, m)

si $B \geq m$ **alors**

$B \longleftarrow B - m$

$S \longleftarrow |S + B|_m$

Retourner S

Implémentation de la Fonction : : Décale

Données : B,d,m

Résultat : $B \times 2^d$ mod m

Initialisation : $n_b \longleftarrow NbrBit(B)$, $n_m \longleftarrow NbrBit(m)$

tant que $d + n_B > n_m$ **faire**

si $j + n_B = n_m + 1$ **alors**

$B \longleftarrow Sup_m(B \times 2^j) + Inf_m(B \times 2^j)$

$n_B \longleftarrow NbrBit(B)$

$d \longleftarrow 0$

sinon

$B \longleftarrow Sup_m(B \times 2^{n_m+2-n_B}) + Inf_m(B \times 2^{n_m+2-n_B})$

$n_B \longleftarrow NbrBit(B)$

$d \longleftarrow d - (n_m + 2 - n_B)$

si $d > 0$ **alors**

$B \longleftarrow |B \times 2^j|_m$

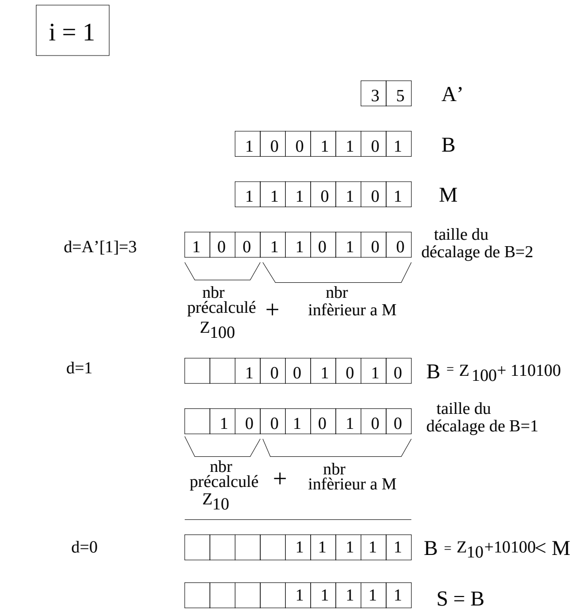
Retourner(B)

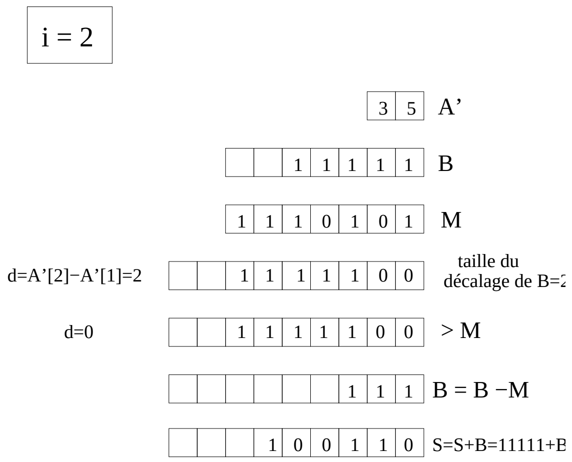
La fonction « Décale » calcule le décalage de B par d modulo m suivant la méthode énoncée dans le paragraphe 3.1.3.

Un exemple de déroulement de l'algorithme :

$$A = 40, B = 77, M = 117$$

$A' = [3,5]$, $Z_{111} = 97$, $Z_{110} = 35$, $Z_{101} = 86$, $Z_{100} = 22$, $Z_{11} = 75$, $Z_{10} = 11$





La solution est donc : $S = (100110)_2$
Soit $40 \times 77 \bmod 117 = 38$

3.1.4 Spécifications de l'algorithme

– La fonction Décale calcule un décalage modulaire suivant les conditions suivantes :

Soit elle peut décaler le nombre à traité jusqu'à $n_M + 2$ ou $n_M + 1$ bits et alors :

> Elle identifie la partie haute, notée Sup, avec l'une de ces six valeurs$(111, 110, 101, 100, 11, 10)$. Elle additionne ensuite le nombre précalculé Z_{Sup} avec la partie basse du nombre et calcule la taille du décalage qu'il reste à faire. Et tant que ce décalage n'est pas nul elle réitère cette opération.

Soit elle peut décaler le nombre jusqu'au plus n_M bits et alors elle s'arrête et retourne ce nombre.

- La fonction Sup_m utilisé dans l'algorithme prend en entrée un nombre codé sur n_m+2 bits, et identifie le nombre sur 3 bits correspondant au $(n_m + 2)^{ieme}$, $(n_m + 1)^{ieme}$ et au $(n_m)^{ieme}$ bits de ce nombre avec l'une des 6 valeurs $\{111$, 110 , 101 , 100 , 011 , 010 $\}$ et retourne le reste modulaire par m qui correspond respectivement à l'une des 6 valeurs précalculées $\{Z_{111}, Z_{110}, Z_{101}, Z_{100}, Z_{011}, Z_{010}\}$ qui sont strictement inférieures à M.
- La fonction Inf_m prend en entrée le même nombre que la fonction Sup_m, et retourne la partie du nombre codé sur les $n_m - 1$ premiers bits. Ce nombre est strictement inférieur à M.
- La fonction Indice utilisée dans les initialisations sert à calculer le tableau A' (qui est ordonné de façon croissante) correspondant aux valeurs des différents décalages à effectuer sur le multiplicande.

- La fonction Taille et la fonction NbrBit servent à récupérer respectivement la taille d'un tableau (nombres d'éléments) et la taille en nombre de bits d'un nombre.

A chaque début de boucle de l'algorithme on initialise la valeur du décalage avec la soustraction de A'[i-1] à A'[i]. Cela permet de récupérer la valeur du décalage qu'on doit effectuer sur le résultat obtenu dans la boucle précédente. De plus la valeur calculée en sortie de boucle est strictement inférieure à $2M$. Pour calculer le reste modulo M il suffit alors de faire une simple comparaison.

La complexité d'un tel algorithme va être complètement liée à la représentation binaire du multiplieur utilisé et à la valeur des nombres précalculés (Z_{val}). Tel que val prenne les valeurs suivantes $(111, 110, 101, 100, 11, 10)$. En effet plus le multiplicateur aura des bits égaux à un dans sa représentation binaire plus la complexité de l'algorithme augmentera. La complexité augmente aussi suivant les valeurs que prennent les valeurs des nombres précalculés. Plus ces valeurs vont être grandes (en nombre de bits) plus la taille du nombre à décaler sera grande et donc il y aura beaucoup plus de décalages « bornés » pour une même valeur de A'[i].

Néanmoins, cette complexité est dans le pire des cas en $\Theta(n)$. On effectue $2(n_A - 1)$ additions, $2(n_A - 1)$ comparaisons et $2(n_A - 1)$ soustractions. Ce qui nous fait $6n_A$ opérations simples. L'algorithme de Montgomery effectue la même opération avec $4n$ opérations simples. En prenant comme multiplicateur un nombre creux[1] la complexité de notre algorithme s'améliore. De plus nous savons par une étude théorique que les constantes multiplicatives sont en moyenne creuses à 50%. Ce résultat est du au théorème de la limite centrale. En prenant ce type de constantes notre nombre d'opérations se ramène à ($6n \times 50\%$). On calcule donc le résultat en 3n opérations. Ce qui est mieux qu'avec l'algorithme de Montgomery.

3.2 Algorithme sur des nombres de 32 Bits

Cet algorithme se base sur le fait que les calculateurs existants permettent d'effectuer des opérations rapides sur des nombres codés sur 32 bits. En effet la multiplication de deux nombres de 32 bits est l'équivalent de 5 additions. Cet algorithme profite de cette spécificité des calculateurs et utilise une réduction afin d'effectuer la multiplication modulaire.

3.2.1 Codage des nombres

Nous utilisons par la suite un nouveau codage de nombre basé sur une réduction modulaire. On représente un nombre A dans un système modulaire M (avec $0 \leq A < M < 2^{32}$) par :

$$[A]_M = A \times 2^{32} \bmod M$$

Ce système représente les nombres par leur décalage modulaire

L'addition dans un tel système peut se représenter ainsi :

$$[A + B]_M = |[A]_M + [B]_M|_M$$

La multiplication dans ce système se fait grâce à la réduction de Montgomery[Mon85]

[1] nombre dont l'écriture binaire possède très peu de 1

3.2.2 Algorithme de Réduction de Montgomery

Algorithme 4: Reduction modulaire de Montgomery : M_Reducte(T)

Données : M, M', R, R^{-1} et T

Conditions : : $gcd(M, R) = 1$

$R > M > 2$

$RR^{-1} - MM' = 1$

$0 \leq T < RM$

Résultat : TR^{-1} mod M

Méthode : : $Q \longleftarrow (T \bmod R)M') \bmod R$

$U \longleftarrow (T + QM) \text{ div } R$

Si $U > M$ alors $U \longleftarrow (U - M)$

Retourner u

A partir de cette réduction il est facile d'effectuer une multiplication modulaire. Soient $[A]_M$ et $[B]_M$ et $R = 2^{32}$ on a :

$$[AB]_M = M\_Reduce([A]_M \times [B])_M = AB \times 2^{32} \bmod M$$

Le but de ce changement de représentation de nombre est de pouvoir effectuer une multiplication modulaire simplement par une multiplication de deux nombre de 32 bits et une réduction de Montgomery. L'avantage que nous apporte cette réduction par rapport à une réduction modulaire normale est qu'elle remplace la division Euclidienne par la récupération des 32 bits de poids forts. Pour retrouver le resultat $ab \bmod m$ dans une représentation de nombre classsique il suffit simplement de calculer M-reduce($[ab]_m$).

De plus le calcul du changement de representation se fait par une multiplication et une réduction.

$$\begin{aligned}
[R]_M &= M_Reduce(2^{64}) \\
[A]_M &= M_Reduce([R]_m \times A) \\
&= A \times 2^{32} \bmod M \qquad (3.1)
\end{aligned}$$

Cette réduction fournit une nouvelle représentation de A et nous permet d'effectuer une multiplication modulaire rapidement dans ce système.

3.2.3 Idée

L'utilisation de ce système de codage des nombres permet d'effectuer des multiplications modulaires rapidement. Cette représentation est très intéressante pour effectuer des changements de bases en RNS En effet pour une base RNS utilisant n moduli inférieurs à 32 bits il suffit de connaître pour chaque extension de base la représentation dans ce système des n constantes multiplicatives. On calcule alors simplement l'extension de base comme suit :

$$|X|_p = \left| M\_reduce(\sum_{i=1}^{n} x_i \times [\mu_p^i]_p) - |\alpha M|_p \right|_p \tag{3.2}$$

Là où l'algorithme d'extension de base de *Shenoy et Kumaresan*(Cf page 9) effectue n multiplications modulaires sur 32 bits, nous effectuons avec avec notre système de codage de nombre seulement n multiplications sur 32 bits et une réduction de Montgomery. Le seul désanvatage de cette méthode est qu'il faut connaître les valeurs des constantes multiplicatives dans ce système de représentation de nombre. Pour remédier à cela il suffit de tabuler la représentation des constantes multiplicatives dans ce système. On obtient ainsi un algorithme efficace pour des changements de bases sur des moduli inférieurs à 32 bits.

3.3 Algorithme avec tables

Le système modulaire nous permet de représenter les nombres par des restes modulaires qui ont une taille (en nombre de bits) bien plus petite que le nombre à représenter. Dans cette méthode nous nous servons de ce fait pour permettre d'effectuer des multiplications modulaires par une constante multiplicative simplement avec des additions et des lectures de tables. Pour cela nous découpons les nombres à multiplier avec ces constantes en plusieurs mots. Ensuite nous stockons les résultats des multiplication modulaires par les constantes multiplicatives. Nous présentons dans la suite un exemple de cette méthode et une généralisation applicable au changement de base en RNS.

Exemple 1 *Soit 3 nombres A , B et M codés sur 32 bits.Où B est une constante multiplicative fixe. On cherche à calculer AB mod M. On représente le nombre A sous la forme suivante :*

$$A = A_0 + A_1 \times 2^8 + A_2 \times 2^{16} + A_3 \times 2^{24}$$

Avec A_0, A_1, A_2, A_3 4 mots binaires codés sur 8 bits.
On pose alors T une table stockant les valeurs suivantes :

$$A'_4 \times B \ mod\ M\ ,\ A'_4 \times 2^8 \times B\ mod\ M\ ,\ A'_4 \times 2^{16} \times B\ mod\ M\ ,\ A'_4 \times 2^{24} \times B\ mod\ M$$

A'_4 représente tous les mots binaires possible sur 8 bits. Ainsi cette table nécessite $4Ko$ d'espace mémoire par constante multiplicative.

La multiplication modulaire AB mod M s'écrit :

$$AB\ mod\ M = \left| \sum_{i=0}^{3} (A_i \times 2^{8i} \times B\ mod\ M) \right|_M$$

Si l'on se place dans le cas de de nombres de 1024 bits il faut considérer 32 moduli de 32 bits pour les représenter dans une base RNS. Nous obtenons ainsi 32 constantes multiplicatives par extension de base. il faut donc considérer 32 tables de 4 Ko chacune. Soit 128 Ko d'espace mémoire pour une extension de base avec un moduli de 32 bits.

Cet exemple nous montre que les multiplications inhérentes à une extension de base via l'algorithme de *Shenoy et Kumaresan* (Cf page 9) peuvent être réalisé à partir de 3×32 additions modulaires et 4×32 lectures de tables nécessitant un espace mémoire de 128 Ko au lieu de 32 produits modulaires. Nous proposons dans la suite d'étudier le rapport qu'il existe entre l'espace mémoire nécessaire à la table et le nombre d'additions modulaires.

3.3.1 Études du rapport taille table et nombre d'additions

Nous commençons cette étude en définissant la formule définissant l'espace mémoire nécessaire aux tables de stockage.

Définition 4 *Soit un nombre X représenté dans une base RNS de p moduli codés sur t bits. On a $X < p \times 2^t$. On se donne les nombres k et n tels qu'on découpe chaque moduli en k mots de n bits. L'espace nécessaire des tables par constante multiplicative est alors de :*

$$espace = k \times 2^n \times t \times p \times 2^{-3} Octets$$

Avec

- 2^n *: le nombre de mot binaire de n bits.*
- t *: la taille des mots à stocker.*

Nous effectuons une étude sur des nombres de 1024 bits représenté en RNS avec une base modulaire de 32 moduli de 32 bits. La taille de l'espace mémoire est donné par la définition précédente et le nombre d,addition est de $32 \times (k-1)$

Nbr de partitions	1	2	3	4	5	6	7	8
Taille des tables	512 Go	16 Mo	384 Ko	128 Ko	80 Ko	48 Ko	28 Ko	16 Ko
Nbr d'additions	0	32	64	96	128	160	192	224

Nous pouvons voir que si l'on ne veut aucune addition modulaire l'espace mémoire nécessaire pour la table est gigantesque. Si l'on veut que le rapport soit intéressant il faut partitionner les moduli en 6 ou 7 mots. Cela permet d'avoir un espace de stockage inférieur à 50 Ko et d'avoir moins de 200 additions.

Nous nous intéressons par la suite à minimiser le rapport entre l'espace mémoire nécessaire et le nombre d'additions.

3.3.2 Minimisation du rapport

Nous allons étudier les fonctions représentant l'espace mémoire multiplié par le nombre d'additions. Ce rapport peut être vu comme une optimisation de la place occupé par des composants sur un circuits électrique. On placera ces test dans le cadre d'un base modulaire de 32 moduli de 32 bits.

La première étude prend en compte la taille de l'espace mémoire multiplié par le nombre d'additions par rapport à la taille des mots utilisés.

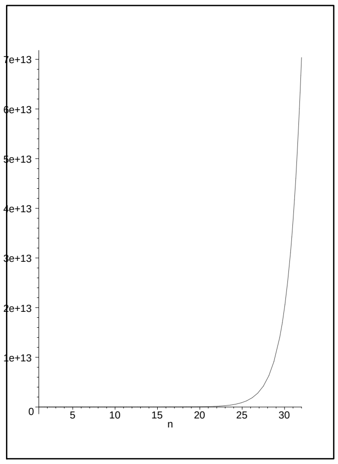


FIG. 3.1 – rapport entre l'espace mémoire multiplié par le nbr d'additions et la taille des mots utilisés

A partir de cette courbe nous n'arrivons pas à déterminer le minimum de ce rapport. Nous proposons donc d'étudier la dérivée de cette courbe pour connaître le minimum de celle-ci. La courbe de cette dérivée est la suivante :

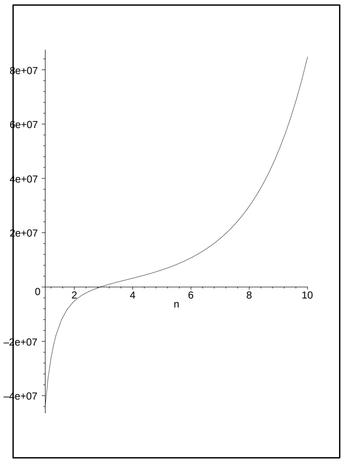


FIG. 3.2 – dérivée de la fonction

Cette courbe nous montre que la valeur minimale de la fonction définie précédemment est de 3. C'est-à-dire que le rapport optimal entre la taille de l'espace mémoire utilisé et le nombre d'additions est pour des mots de 3 bits. Cela nous donne donc un découpage des moduli de 32 bits en 11 mots de 3 bits. Ce rapport reste le même quelque soit la base modulaire choisi et la taille des moduli utilisé. Nous affirmons donc que la méthode de multiplication modulaire par table est optimale pour un partitionnement des moduli te taille n, en $\lceil \frac{n}{3} \rceil$ mots de 3 bits suivant le critère que nous avons énoncé au début de ces test.

3.3.3 Résultat

L'utilisation de tables est un moyen facile d'éviter des calculs trop important lorsqu'on doit effectuer plusieurs opérations avec une opérande fixe. Cette méthode permet donc de limiter le nombre de calcul à effectuer lorsqu'on doit effectuer un changement de base. De plus l'implémentation de ces tables dans un circuit électronique permet une certaine compression qui diminuera les tailles de celles-ci. Les données optimales pour cette méthode sont indépendantes du choix de la base modulaire utilisée et de la taille des moduli. Néanmoins, le critère choisi pour calculer cet optimal est prépondérant.

Chapitre 4

Conclusion

Nous avons présenté dans ce travail des méthodes de conversion ainsi que des opérateurs modulaires pour un système de représentation de nombres appelé « Residue Number System ». Ce système permet de représenter de grands entiers grâce à de simples valeurs résiduelles. De plus il apporte l'avantage de pouvoir effectuer des opérations simples en parallèle.

Nous avons mis en avant le fait qu'il est difficile d'améliorer le changement de base en cherchant à minimiser la valeurs des constantes multiplicatives, inhérentes à cette opération. La recherche de ces valeurs nous est difficile car la courbe de distribution de ces valeurs « en moyenne » est centrée sur la taille du nombre maximum, moins un.

Notre travail s'est porté par la suite sur l'étude de méthodes permettant d'optimiser les opérations modulaires. Il ressort de notre travail que nos opérateurs modulaires rendent efficace ces opérations pour certaines formes particulières d'opérandes. La première méthode qui utilise des valeurs précalculées, permet d'effectuer une multiplication modulaire rapidement sur des nombres creux. Ce travail est intéressant car il permet d'améliorer le changement de base RNS du fait que les nombres sont en général creux à 50%. Le deuxième algorithme que nous avons proposé dans ce travail se place dans le cadre de base modulaire utilisant des moduli inférieurs à 32 bits. Il utilise une nouvelle représentation de nombres lui permettant d'effectuer une réduction modulaire sans aucune division. De plus il tire profit du faible coût des multiplications sur des nombres inférieurs à 32 bis. L'utilisation de ce codage de nombres pour les constantes multiplicatives permet d'effectuer un changement de base RNS simplement par des multiplication 32 bits et une réduction. Enfin nous avons proposé une méthode basé sur l'utilisation de tables afin d'effectuer une multiplication modulaire simplement avec des additions modulaires. Nous avons montré que la taille des tables utilisées est directement liée au nombres d'additions nécessaire à cette méthode et que le rapport optimal est indépendant de la taille des moduli utilisés.

De ce travail ressort le fait que nous sommes en mesure de fournir des opérateurs modulaires permettant de rendre plus efficace le système RNS pour des bases données. De ce fait nous pouvons espérer introduire un tel système dans des applications cryptographiques. Afin de rendre ce système encore plus performant nous pourrions nous intéresser à la recherche de constantes multiplicatives fortement creuses (à 90%). Par la suite nous pourrions nous intéresser aux divers systèmes cryptographique existants tel les courbes elliptiques dans les corps finis de Galois $\mathrm{GF}(p^k)$. Nous pourrions voir si la mise en place de nouveaux opérateur permettraient une amélioration de ces systèmes. Enfin ce travail pourrait déboucher sur la création d'un cryptoprocesseur.

Bibliographie

[ARS78] L. Adleman, R. L. Rivest, and A. Shamir. A method for obtaining digital signature and public-key cryptosystems. *Communication of the ACM*, 21(2), 1978.

[BDK97] J-C. Bajard, L-S. Didier, and P. Kornerup. An RNS Montgomery Modular Multiplication Algorithm. In *Proc. 13th IEEE Symposium on Computer Arithmetic*, pages 234–239. IEEE Computer Society, July 1997. A revised version published in IEEE Transactions on Computers, Vol.46(7), July 1998.

[BDK01] J-C. Bajard, L-S. Didier, and P. Kornerup. An RNS Montgomery Modular Multiplication Algorithm. In *Proc. 15th IEEE Symposium on Computer Arithmetic*, pages 59–65. IEEE Computer Society, June 2001.

[BDKR00] J.-C. Bajard, L.-S. Didier, P. Kornerup, and F. Rico. Some improvements on rns montgomery modular multiplication. In F. T. Luk, editor, *SPIE's International Symposium on Optical Science and Technology : Advanced Signal Processing Algorithms, Architectures, and Implementations X*, volume 4116, pages 214–225, San Diego, CA, USA, August 2000.

[DH76] W. Diffie and M. E. Hellman. Multiuser cryptographic techniques. In *Proc. AFIPS 1976 National Computer Conference*, pages 109–112, Montvale, N.J., 1976. AFIPS.

[Did98] L. S. Didier. *Division et Multiplication Modulaire en Représentation Modulaire*. PhD thesis, Université de Provence, 1998.

[DIP93] G. Dimauro, S. Impedovo, and G. Pirlo. A new technique for fast number comparison in the residue number system. *IEEE Transactions on Computers*, 42(5) :608–612, May 1993.

[Gam91] D. Gamberger. New Approach to Integer Division in Residue Number System. In P. Kornerup and D. W. Matula, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 84–91, Grenoble, France, 1991. IEEE Computer Society Press, Los Alamitos, CA.

[Gar59] H. L. Garner. The residue number system. *IRE Transactions on Electronic Computers*, EC-8 :140–147, 1959.

[HP94] C. Y. Hung and B. Parhami. An approximate sign detection method for residue numbers and its application to rns division. *Computer Math. Applic.*, 27(4) :23–35, 1994.

[Mon85] P. L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44(170) :519–521, April 1985.

[PP95] Karl C. Posch and Reinhard Posch. Modulo réduction in residue number systems. *IEEE Transactions on Parallel and distributed systems*, 6(5) :449–454, may 1995.

[SK89] A. P. Shenoy and Kumaresan. Fast base extension using a redundant modulus in rns. *IEEE Transactions on Computers*, 38(2) :292–296, 1989.

[ST67] N. Szabo and R. I. Tanaka. *Residue Arithmetic and its application to Computer Technology.* McGraw-Hill, 1967.

[Yan00] S.Y Yan. *Number Theory for Computing* . Springer, Birmingham, UK, 2000.

Résumé :

Les systèmes modulaires de représentation des nombres ont la particularité de pouvoir effectuer en parallèle des calculs sur des valeurs résiduelles. Le changement de base de ce sytème est une opération inhérente à beaucoup d'algorithmes utilisant cette représentation. Nous montrons ici que la construction de bases particulières visant à améliorer cette opèration est une tâche fastidieuse et que de telles bases sont en très faible quantité. Nous mettons en avant des opèrateurs modulaires sur des nombres spécifiques, dont l'intégration aux systèmes modulaires permet l'optimisation de certains algorithmes de ce système.

Mots clés : modulaire, représentation, extension, opérateur modulaires, constantes multiplicatives

Abstract :

The main feature of the residue number system is to allow a parallelisation on small residue number. Most of algorithms using these systems need, in their implementation, a base extension. We show in this paper that base built around typical number to improve base extension are in too few quantity to be interesting. We present few modular operators which allow to improve algorithms based on a residue number system.

keywords : modular, representation, extension, modular operators, multiplicative-constants