Simultaneous conversions with the Residue Number System using linear algebra
Application to integer matrix multiplication over integers or polynomials

...

Matrix multiplication with exact results (i.e. integers, finite fields, polynomials) is important in computer algebra. Since today's processors do not provide such structures, careful implementation must be done to get high performance.

In this paper, we focus on matrix multiplication over integers and polynomials over a finite field. We will discuss the use of CRT and Kronecker substitution to reduce to multiplication with small entries.

Our contribution is an algorithm that reduces most of the computations of a conversion with the Residue Number System to linear algebra. This allows us to benefit from the peak performance of linear algebra implementations to speed up simultaneous conversions with the Residue Number System on a wide range of input sizes. Note that we do not improve the best asymptotic complexity of conversions but dramatically improve the constant of a sub-optimal algorithm.

Our main application is matrix multiplication over integers and polynomials over a finite field. Our speed-up of the conversions to and from the Residue Number System improves significantly the overall time of matrix multiplication.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Algorithm design and analysis; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*computations in finite fields.*

General Terms: Algorithms, Experimentation, Performance.

Additional Key Words and Phrases:

1. INTRODUCTION

biblio ? checker Bernstein - quel papier ?

Talk about mixed radices litterature ? Who discovered quasi-linear conversions to RNS ?

Importance of matrix multiplication in computer algebra as a source of implementations at peak performances.

We will use the bit complexity model throughout this paper. The integer β will represent the bit-length of a machine word in the complexities. We will attribute a unit cost to operations such as addition, subtraction, multiplication, quotient and remainder between single precision integers.

1.0.0.1 *Notations.* Let $\mathsf{M}(n)$ denotes the bit complexity of the multiplication of two n-bits integers. In order to provide a fair etimate of practical complexities, we will consider that we are able to perform any arithmetic operations on t-bits integers at a constant cost. Therefore, let $n = st$ we will consider that the complexity of multiplying n-bits integer is $\mathsf{M}(s)\mathsf{M}(t)$ and thus $\mathsf{M}(s)$.

We denote resp. by $(a \operatorname{rem} p)$ and $(a \operatorname{quo} p)$ the remainder and quotient of the

Euclidean division of $a \in \mathbb{Z}$ by $p \in \mathbb{N}$ where $0 \leqslant (a \operatorname{rem} b) < b$. We extend the notation $(a \operatorname{rem} p)$ to any rational $a = n/d \in \mathbb{Q}$ such that $\gcd(d, p) = 1$ to be the unique residue of $(a \bmod p)$ satisfying $0 \leqslant (a \operatorname{rem} b) < b$.

For any $a = n/d$ with d coprime to p, we let $[a]_p$ be the unique representative in $\{0, \ldots, p-1\}$ of a modulo p.

We call informally a pseudo-reduction of a modulo p the computation of b such that $a = b \bmod p$ and b "not too big" compared to p. In practice, we often will have that $b = \mathcal{O}(p^2)$.

Say that our complexity model is bit complexity.

Should we take β (*e.g.* 2^{32} or 2^{64}) as a constant and simplify complexity ? Yes and no : we will give complexities in $\mathcal{O}(\ldots \mathsf{I}(\log \beta))$ and then specify that the term $\mathsf{I}(\log \beta))$ can be removed when β is a constant.

Only give costs for the typical case $p_i \simeq \beta, s \simeq B, r \gg s$ because of $\beta^B < p_1 \cdots p_s$, which implies that $B < s$?

Assume that $n, s \leqslant \beta$! Indeed we can assume that we won't have integers of more than β machine words since it is the typical memory limit for a system whose address are encoded on a single precision integer.

We make this restriction to ensure the number of primes s necessary to encode the product of two $N \times N$ matrices of integers of size n is still $\Theta(n)$ (the equation is $N\beta^{2n} \leqslant \beta^s$).

Introduce $\mathsf{MM}(n, k, m)$ as the cost of unbalanced matrix multiplication.

1.0.0.2 *Bibliography.* Bibliography on reductions and pseudo-reductions. Recall Barett, Montgomery results ?

1.0.0.3 *Complexity results.* Let $a \in \mathbb{Z}$ and $p \in \mathbb{N}$ such that $|a| < 2^n$ and $p < 2^m$. Cost of $a \bmod p$ when :

(1) $n = \Theta(m)$. **Cas classique, on en a vraiment besoin –

$$\mathcal{O}\left(\mathsf{M}(m)\right) \tag{1}$$

(2) $n \gg \Theta(m)$ **Servira à prouver l'algo naïf de multi-réduction –

$$\mathcal{O}\left(\frac{n}{m}\mathsf{M}(m)\right) \tag{2}$$

(3) $n - m \ll \Theta(m)$ **Sert à montrer que la finalisation des pseudos-réductions est peu couteuse**
**Polynomial analog suggests

$$\mathcal{O}\left(\mathsf{M}(n-m)\frac{m}{n-m}\right) \tag{3}$$

2. CONVERSIONS WITH RESIDUE NUMBER SYSTEM

A classic way to represent a positive integer is to use a positional number system according to a base β. Indeed, any integer $a \in \mathbb{N}$ can be encoded as $(a_{n-1}, \ldots, a_1, a_0)_\beta$ since it is uniquely determined by $a = \sum_{i=0}^{n-1} a_i \beta^i$ assuming $0 \leq a_i < \beta$. One advantage of such a representation is that it allows to represent infinitely many integers and compare or add/subtract integers in $\mathcal{O}(n)$ operations on the digits, where n is the number of digits of the operands.

However, multiplication with this representation is more complex and requires $\mathcal{O}(n^2)$ operations on the digits. Many other number systems allow to improve the complexity of the multiplication but most of them loose the benefit to represent infinitely many integers.

In this article, we do not intend to present all possible representations and let the reader refer to [Brent and Zimmermann 2010; Bernstein 2008] for a survey on fast integer arithmetic. Instead, we only present the multi-modular representation that will allow linear complexity for addition and multiplication of integers and then describe our method to multiply integer matrices.

2.1 Residue Number System

The Residue Number System is non-positional number system that allow to represent a finite subset of integers. Let $M = m_1 \times m_2 \times \ldots \times m_s \in \mathbb{N}$ where $m_i \in \mathbb{N}$ such that $\forall i \neq j$, $m_i \perp m_j$, then any integers $a \in [0, \ldots, M-1]$ can be uniquely determined by its residue $([a]_{m_1}, [a]_{m_2}, \ldots, [a]_{m_s})$. The uniqueness of this residual representation is ensured by the the ring isomorphism

$$\mathbb{Z}_M \simeq \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_s} \tag{4}$$

when the m_i's are coprime. This ring isomorphism is often called the Chinese Remainder Theorem (CRT)[Gathen and Gerhard 2003, Section 5.4].

This residual representation, also called Residue Number System (RNS), provides groups isomorphism for addition and multiplication which allow to perform these operations individually on each residual. If the m_i are fixed, let us note $[a]_i := [a]_{m_i}$ so that $([a]_1, [a]_2, \ldots, [a]_s)_{RNS}$ is the representation of the integer $a \in \mathbb{N}$.

Let $a, b, c, d \in \mathbb{N}$ such that $c \equiv a + b \bmod M$ and $d \equiv a \times b \bmod M$ then

$$\begin{aligned} ([c]_1, [c]_2, \ldots, [c]_s)_{RNS} &= ([a]_1 + [b]_1 \bmod m_1, \ldots, [a]_s + [b]_s \bmod m_s)_{RNS} && (5) \\ ([d]_1, [d]_2, \ldots, [d]_s)_{RNS} &= ([a]_1 \times [b]_1 \bmod m_1, \ldots, [a]_s \times [b]_s \bmod m_s)_{RNS} && (6) \end{aligned}$$

It is obvious from equations 5 and 6 that addition and multiplication in RNS require $\mathcal{O}(s)$ operations on the residuals. One may note that division in RNS is possible in the same way (individually on each residual) only when the division is exact.

In order to benefit from RNS representation, one often need to convert back and forth between classic positional number system and RNS. Of course, these conversions are costly and must be avoided when possible. However, when the number of operations in RNS is high enough, these conversions can be neglected and the RNS approach yields the most efficient solution.

2.2 Naive approach

2.2.1 *Conversion to RNS.* In order to convert an integer a to a residue number system $(m_1, \ldots m_s)$, one can of course apply Euclidean division of a by m_i for $i \in \{1, \ldots, s\}$. In this section, we want to reduce an integer $a \in \mathbb{Z}$ modulo each $m_1, \ldots, m_s \in \mathbb{N}$. Let us assume the particular case when $m_1, \ldots, m_s < \beta$ and $a < \beta^n$, where $n = \Theta(s)$ and $\beta = 2^t$ with t related to machine wordsize. Indeed, this case is representative of computations that takes advantage of RNS arithmetic on nowadays computers. Remark: if $n \gg \Theta(s)$, it suffices to compute $\tilde{a} = a \bmod M$

where $M = \Pi_{i=1}^{s} m_i$ to get back to the case $n = \Theta(s)$. This will add $\mathcal{O}(\frac{n}{s}\mathsf{M}(s)\mathsf{M}(t))$ bit operations to the complexity.

Algorithm 1: Modular reduction

```
Input: $a = \sum_{i=0}^{n-1} a_i\beta^i, m_j \in \mathbb{N}$
Output: $a \bmod m_j$
$c = a_{n-1}$
for $i = n-2$ to $0$ do
    $r = c\beta \bmod m_j$
    $c = r + a_i$
return $c \bmod m_j$;
```

Algorithm 1 presents the naive approach to perform a modular reduction. The bit complexity for computing $a \bmod m_j$ is $\mathcal{O}(n\mathsf{M}(t))$. Therefore, conversion to a RNS basis with s moduli costs $\mathcal{O}(sn\mathsf{M}(t))$ bit operations. When t is a constant the cost becomes $\mathcal{O}(sn)$.

The case when the m_j's are almost of the same size as a, i.e. $m_1, \ldots, m_s \in \beta^{\Theta(n)}$, is more classical and can be handled through integer multiplications using Barrett or Montgomery algorithms [Barrett 1986; Montgomery 1985]. Then, the cost of modular reduction becomes $\mathcal{O}(\mathsf{M}(nt)$ bit operations and the cost of conversion to RNS is $\mathcal{O}(s\mathsf{M}(nt))$. This boils down to $\mathcal{O}(s\mathsf{M}(n))$ when t is fixed.

2.2.2 *Conversion from RNS.* Assuming that an integer a is given by its RNS representation $([a]_1, \ldots, [a]_s)$ with the base $(m_1, \ldots, m_s)$. In order to retrieve the value of a written in base β, i.e. $a = \sum_{i=0}^{n-1} a_i\beta^i$, one need to solve the following congruence equations in the a_i's:

$$\begin{aligned} [a]_1 &\equiv \sum_{i=0}^{n-1} a_i\beta^i \bmod m_1 \\ &\vdots \\ [a]_s &\equiv \sum_{i=0}^{n-1} a_i\beta^i \bmod m_s \end{aligned} \tag{7}$$

A solution to this system of congruence is given by the Chinese Remainder Theorem recalled in Equation 4. Let us denote $M = \Pi_{i=1}^{s} m_i$ and $M_i = M/m_i$. The solution can be computed using the following equation

$$a \equiv \sum_{i=1}^{s} M_i([a]_i M_i^{-1} \bmod m_i) \bmod M \tag{8}$$

Let us assume, without loss of genericity, that M, the M_i and the $M_i^{-1} \bmod m_i$ are precomputed. Recalling that $m_i < \beta = 2^t$, then each multiplication $a_i' := ([a]_i M_i^{-1} \bmod m_i)$ costs $\mathcal{O}(\mathsf{M}(t))$ and each unbalanced multiplication $M_i a_i'$ costs $\mathcal{O}(s\mathsf{M}(t))$. Since $a_i' < m_i$, by definition of M_i we have $M_i a_i' < M < \beta^s$. This implies the following bound on the sum

$$\sum_{i=1}^{s} M_i a'_i < s\beta^s$$

and its reduction cost is $\mathcal{O}(ts \log s)$ (using formula (3)), which is not dominant. Therefore the total cost is $\mathcal{O}(s^2\mathsf{M}(t))$ bit operations.

Concerning the precomputation costs, M can be computed with s unbalanced multiplications at a cost of $\mathcal{O}(s^2\mathsf{M}(t))$ bit operations. Then each unbalanced exact division M/m_i has bit complexity $\mathcal{O}(s\mathsf{M}(t))$, and its corresponding reduction modulo m_i also costs $\mathcal{O}(s\mathsf{M}(t))$ using formula (2). Finally, computing each $M_i^{-1} \bmod m_i$ cost $\mathcal{O}(\mathsf{M}(t)\log t)$. Altogether, the precomputation costs $\mathcal{O}(s^2\mathsf{M}(t))$ bit operations.

2.3 Quasi-linear approach

In this section, we briefly recall the quasi-linear algorithm based on binary multiplication tree to convert to RNS. We follow the exposition given in [Gathen and Gerhard 2003][Section 10.3].

2.3.1 *Conversion to RNS.* The first step of the algorithm is a computation of the binary multiplication tree of the moduli m_i. Assume for the sake of simplicity that $s = 2^\kappa$. Starting from $m_1, \ldots, m_s$, we compute the first level of the tree made of the products $m_1m_2, m_3m_4, \ldots, m_{s-1}m_s$, then the second level made of the products $m_1m_2m_3m_4, \ldots, m_{s-3}\cdots m_s$ from the first level. And so on up to the last level of height κ made of the full product $M = m_1m_2\cdots m_s$.

The cost of computing this binary multiplication tree is $\mathcal{O}(\mathsf{M}(s)\log s\ \mathsf{M}(t))$ bit operations. This boils down to $\mathcal{O}(\mathsf{M}(s)\log s)$ when t is a constant.

The second step of the algorithm is recursive and use a divide-and-conquer scheme. In order to reduce an integer a modulo $m_1, \ldots, m_s$, one need to compute

$$a_l = a \bmod (m_1 \cdots m_{s/2}) \text{ and } a_h = a \bmod (m_{s/2+1} \cdots m_s).$$

Then recursively reduce a_l modulo $m_1, \ldots, m_{s/2}$ and a_h modulo $m_{s/2+1}, \ldots, m_s$ and you are done. Note that the products $m_1 \cdots m_{s/2}$ and $m_{s/2+1} \cdots m_s$ were precomputed, and this holds for every products required in the recursive calls.

The cost $C(s)$ of the reduction then satisfies $C(s) = 2C(s/2) + \mathcal{O}(\mathsf{M}(s)\mathsf{M}(t))$ which yields $C(s) = \mathcal{O}(\mathsf{M}(s)\log s\ \mathsf{M}(t))$ as before.

Therefore, the complexity of this approach for conversion to RNS is $\mathcal{O}(\mathsf{M}(s)\log s\ \mathsf{M}(t))$ bit operations and this becomes $\mathcal{O}(\mathsf{M}(s)\log s)$ when t is constant.

2.3.2 *Conversion from RNS.* The backwards conversion is a bit trickier. It still follows Formula (8) together with a divide-and-conquer approach.

The first step is to precompute $M_i^{-1} \bmod m_i$ for all $1 \leqslant i \leqslant s$. This can be achieved in time $\mathcal{O}(\mathsf{M}(s)\log s\ \mathsf{M}(t))$. For this matter, it suffices to use the quasi-linear conversion to RNS of previous section to compute M modulo $m_1^2, \ldots, m_s^2$. Then one can recover $M_i \bmod m_i = \left(M \bmod m_i^2\right)/m_i$ with an exact integer division. It remains to perform the inversions of $(M_i \bmod m_i)$ modulo m_i which amounts to $\mathcal{O}(s\mathsf{M}(t)\log t)$ bit operations.

Now let $a'_i := [a]_i M_i^{-1} \bmod m_i$. Formula (8) gives $a \equiv \sum_{i=1}^{s} a'_i (M/m_i) \bmod M$ which can be computed recursively on the number s of moduli as follow.

Let $M_l := m_1 \cdots m_{s/2}$ and $M_h := m_{s/2+1} \cdots m_s$. Compute recursively

$$\begin{aligned} a_l &:= \sum_{i=1}^{s/2} a'_i (M_l/m_i) \bmod M \\ a_h &:= \sum_{i=s/2+1}^{s} a'_i (M_h/m_i) \bmod M \end{aligned}$$

and finally recover a using $a = M_h \cdot a_l + M_l \cdot a_h$.
This recursive algorithm costs $C(s) = 2C(s/2) + \mathcal{O}(\mathsf{M}(s)\mathsf{M}(t))$ which yields $C(s) = \mathcal{O}(\mathsf{M}(s) \log s \; \mathsf{M}(t))$ as before.

2.4 Summary of complexities

In the following table, we recall all the complexities for conversions with RNS representation. We assume that the RNS basis is given as $(m_1, \dots, m_s)$ such that each $m_i < 2^t$ and integers to convert have n-bits with $n = \Theta(st)$.

	conversion from RNS	conversion to RNS
naive approach	$\mathcal{O}(s^2\mathsf{M}(t))$	$\mathcal{O}(s^2\mathsf{M}(t))$
fast approach	$\mathcal{O}(\mathsf{M}(s) \log s \; \mathsf{M}(t))$	$\mathcal{O}(\mathsf{M}(s) \log s \; \mathsf{M}(t))$

3. SIMULTANEOUS RNS CONVERSIONS

In this section, we want to simultaneously reduce the integers $a_1, \dots, a_r \in \mathbb{Z}$ modulo each of the positive coprime integers $m_1, \dots, m_s \in \mathbb{N}$. As before, we assume that $m_1, \dots, m_s < 2^t$ and that $a_j < 2^{nt}$ with $n = \Theta(s)$. In order to simplify the notations and the complexity estimate, we will use $\beta = 2^t$ and $n = s$ throughout this section.

3.1 Straightforward

The previous algorithms of Section 2 directly apply to simultaneous conversions with RNS. Of course, the precomputations of these algorithms are independent of the integer a to reduce. Therefore, they are done once and for all. The bit complexity of simultaneous conversions with RNS is $\mathcal{O}\left(rs^2\mathsf{M}(t)\right)$ for the naive approach and $\mathcal{O}\left(r\mathsf{M}(s) \log s \mathsf{M}(t)\right)$ for the quasi-linear approach.

Note that, in term of implementation, simultaneous reductions using the naive algorithms can benefit from vectorized (SIMD) instructions to lower the constant and can be easily parallelized. On the other hand, simultaneous reductions using the quasi-linear approach do not benefit from SIMD instructions (or at least not straightforwardly).

3.2 Linear Algebra

When one convert an integer a given in base β (i.e. $a = \sum_{i=0}^{s-1} a_i \beta^i$,) to its RNS representation in base $(m_1, \dots, m_s)$, we cannot take benefit of precomputing every $\beta^i \bmod m_j$ since each precomputations will be used only once per moduli. However, this is not the case when reducing many integers with the same moduli. Indeed, in such case, the precomputations can serve for every integer you need to reduce. In this section, we provide a novel way to simultaneously convert to and from RNS that take advantage of such precomputations and introduce matrix multiplication into the complexity. In particular, we will achieve a complexity of $\mathcal{O}(rs^{\omega-1}\mathsf{M}(t))$ bit operations where ω is the exponent in the complexity of matrix multiplication, best value being $\omega = 2.3729$ [Le Gall 2014]

3.2.1 *Conversions to RNS.* The conversion to RNS is split up into three phases.

—First, one need to compute all the $|\beta^i|_{m_j} = \beta^i \bmod m_j$ for $1 \leq i, j \leq s$ and group them into the following matrix

$$B = \begin{bmatrix} 1 & |\beta|_{m_1} & |\beta^2|_{m_1} & \cdots & |\beta^{s-1}|_{m_1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & |\beta|_{m_s} & |\beta^2|_{m_s} & \cdots & |\beta^{s-1}|_{m_s} \end{bmatrix} \in \mathcal{M}_{s\times s}(\mathbb{Z}).$$

—Then, use this matrix to pseudo-reduce the r integers $[a_1, \ldots, a_r]$ modulo the m_i's. Here, pseudo-reduction means that we reduce integers of bitsize $\mathcal{O}(s)$ to bitsize $\mathcal{O}(\log s)$. For this purpose, we write the expansion in base β of

$$a_i = \sum_{j=0}^{s-1} \alpha_{i,j}\beta^j \quad \text{for } 1 \leqslant i \leqslant r$$

and group them into the following matrix

$$C = \begin{bmatrix} \alpha_{1,0} & \alpha_{2,0} & \alpha_{3,0} & \cdots & \alpha_{r,0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{1,s-1} & \alpha_{2,s-1} & \alpha_{3,s-1} & \cdots & \alpha_{r,s-1} \end{bmatrix} \in \mathcal{M}_{s\times r}(\mathbb{Z})$$

From the definition of matrices B and C we have

$$(BC)_{i,j} \equiv a_j \bmod m_i \tag{9}$$

and

$$0 \leq (BC)_{i,j} < sm_i\beta < s\beta^2. \tag{10}$$

This shows that the matrix product BC encode all the pseudo-reduction of the a_i's with the m_j's.

—The last step consist in reducing the i-th row of the matrix product BC modulo m_i for $1 \leqslant i \leqslant r$.

The complexity of this approach is $\mathcal{O}(\mathsf{MM}(s, s, r)\mathsf{M}(t))$ bit operations. Indeed, the dominant cost is achieved for the matrix product BC where each entries of both matrices are bounded by $\beta = 2^t$. The construction of matrix B only costs $\mathcal{O}(s^2\mathsf{M}(t))$ bit operations while the last step to reduce the entries in BC costs $\mathcal{O}(sr\mathsf{M}(t))$ bit operations since by assumption $s < \beta$.

Assuming $s \leq r$, this gives a complexity of $\mathcal{O}(rs^{\omega-1}\mathsf{M}(t))$. This boils down to $\mathcal{O}(r^{2(\omega-2)}s^2\mathsf{M}(t))$ when $s > r$.

3.2.2 *Conversions from RNS.*

Simultaneous pseudo-reconstructions. Recall that we denote $M = \Pi_{i=1}^{s} m_i$ and $M_i = (M/m_i) \operatorname{rem} m_i$ for $1 \leqslant i \leqslant s$. Let $l_i := \sum_{j=1}^{s} a_{i,j} M_j [M_j^{-1}]_{m_j}$ so that $a_i = l_i \bmod M$ (see Formula (8)) and $l_i < M\beta$. We say that l_i are pseudo-reconstructions of $(a_{i,\ell})$ modulo $m_1, \ldots, m_s$.

Let us precompute $M_\ell [M_\ell^{-1}]_{m_\ell}$ for all $1 \leqslant \ell \leqslant s$. If we have precomputed the M_ℓ before (*e.g.* during the reduction step), then we can compute $[M_\ell]_{m_\ell}$ from M_ℓ in time $\mathcal{O}(s\mathsf{I}(\log \beta))$, then $[M_\ell^{-1}]_{m_\ell}$ in time $\mathcal{O}(\mathsf{I}(\log \beta) \log\log \beta)$ and finally $M_\ell [M_\ell^{-1}]_{m_\ell}$ in time $\mathcal{O}(s\mathsf{I}(\log \beta))$. When β is a constant, this comes down to $\mathcal{O}(s)$ bit operations.

Now from the expansion of $M_\ell [M_\ell^{-1}]_{m_\ell}$ in base β

$$M_j [M_j^{-1}]_{m_j} = \sum_{k=0}^{s-1} e_{j,k} \beta^k.$$

we perform the computation of l_i using linear algebra using

$$l_i = \sum_{j=1}^{s} a_{i,j} M_j [M_j^{-1}]_{m_j} = \sum_{j=1}^{s} a_{i,j} \sum_{k=0}^{s-1} e_{j,k} \beta^k = \sum_{k=0}^{s-1} \left(\sum_{j=1}^{s} a_{i,j} \cdot e_{j,k} \right) \beta^k.$$

Let $(d_{i,j}) \in \mathcal{M}_{r\times s}(\mathbb{Z})$ be the product of the matrices $(a_{i,j}) \in \mathcal{M}_{r\times s}(\mathbb{Z})$ and $(e_{j,k}) \in \mathcal{M}_{s\times s}(\mathbb{Z})$. Then $l_i = \sum_{k=0}^{s-1} d_{i,j} \beta^k$. Note that $(d_{i,j})$ are not the exact coefficients of the β-expansion of l_i. But we can still compute the l_is in time $\mathcal{O}(rs\mathsf{I}(\log \beta))$ since $d_{i,j} \leqslant s\beta^2$ and $s \leqslant \beta$ by assumption. The matrix product to compute $(d_{i,j})$ can be done in bit complexity $\mathcal{O}\left(rs^{\omega-1}\mathsf{I}(\log(\beta))\right)$.

3.2.2.1 *Simultaneous reconstructions.* The final step of the reconstruction consists in reducing l_i modulo M. This step is relatively cheap since l_i is almost reduced. Using $s < \beta$ and the modular reduction complexity results (**see Introduction**) when $\log(a/p) \ll \Theta(\log(p))$ in our case $l_i = \mathcal{O}(s\beta^{s+2}) = \mathcal{O}(\beta^{s+2})$ and $M = \mathcal{O}(\beta^s)$, the last reduction step costs $\mathcal{O}\left(s\mathsf{I}(\log \beta)\right)$ per l_i. Thus a total cost of $\mathcal{O}(rs\mathsf{I}(\log \beta))$.

4. MATRIX MULTIPLICATION WITH MULTI-PRECISION INTEGER COEFFICIENTS

4.1 Matrix multiplication modulo composite primes

Of course, this ideal case of application of matrix multiplication using RNS is for matrices over $\mathbb{Z}/M\mathbb{Z}$ with $M = m_1 \cdots m_s$ and $m_i < \beta$. Then, the multiplication of two matrices $A \in \mathcal{M}_{n,k}(\mathbb{Z}/M\mathbb{Z})$, $B \in \mathcal{M}_{k,m}(\mathbb{Z}/M\mathbb{Z})$ is done classically by converting them to RNS, multiply each of the reductions and convert from RNS. The cost is

$$\mathcal{O}(\left[\mathsf{MM}(n,k,m)s + (nk+km)s^2\right] \mathsf{I}(\log \beta))$$

using the naive approach for RNS,

$$\mathcal{O}(\left[\mathsf{MM}(n,k,m)s + (nk+km)s^{\omega-1}\right] \mathsf{I}(\log \beta))$$

using linear algebra approach and

$$\mathcal{O}([\mathsf{MM}(n,k,m)s + (nk+km)\mathsf{I}(s)\log s]\, \mathsf{I}(\log \beta))$$

using the quasi-optimal approach.

4.2 Integer matrix multiplication

In this section, the matrices $A \in \mathcal{M}_{n,k}(\mathbb{Z})$, $B \in \mathcal{M}_{k,m}(\mathbb{Z})$ have entries of absolute value less than β^n. Therefore, their product $C := A \cdot B$ have entries of absolute value less than $k\beta^{2n}$.

We will compute C modulo $M = m_1 \cdots m_s$ such that $2k\beta^{2n} < M$ and $m_i < \beta$. Therefore, the entries of C coincide with their unique residue modulo M whose absolute value is less than $\frac{M-1}{2}$.

Since we make the simplifying assumption that $k < \beta$ and assuming there exists enough primes slightly smaller than β, we will necessarily have $s = \Theta(n)$. Therefore, we can state the costs:

$$\mathcal{O}(\left[\mathsf{MM}(n,k,m)n + (nk+km)n^2\right] \mathsf{I}(\log\beta))$$

using the naive approach for RNS,

$$\mathcal{O}(\left[\mathsf{MM}(n,k,m)n + (nk+km)n^{\omega-1}\right] \mathsf{I}(\log\beta))$$

using linear algebra approach and

$$\mathcal{O}([\mathsf{MM}(n,k,m)n + (nk+km)\mathsf{I}(n)\log n] \, \mathsf{I}(\log\beta))$$

using the quasi-optimal approach.

Aarrrg ! We have the same notation : n is both the matrix size and its bit size

4.3 Matrix multiplication modulo any prime

In the case of matrix multiplication over $\mathbb{Z}/M\mathbb{Z}$ when M do not decompose into the product of several small primes, we actually perform an integer matrix multiplication and reduce the result modulo M. The costs remain the same than in Section 4.2 with the bit size $n := \log_\beta M$. Note that each final reduction costs $\mathcal{O}(\mathsf{I}(n)\mathsf{I}(\log\beta))$ (see Section 2.2.1), which is negligible.

5. IMPLEMENTATION

All the codes are available in the FFLAS_FFPACK Library (see https://github.com/linbox-team/fflas-ffpack).

- The simultaneous reduction/reconstruction's code is available in the file: `fflas-ffpack/fflas-ffpack/field/rns-double.inl`
- The matrix multiplication code is available in the file: `fflas-ffpack/fflas-ffpack/fflas/fflas_fgemm/fgemm_classical_mp.inl`

5.1 Reduction to word-size matrix multiplication

how to store intermediate matrices (Kronecker matrix) in order to optimize cache efficiency (transpose is delayed to the matrix multiplication). We always want the Kronecker matrix to be stored β-adic major. How to choose β and the m_i to guarantee the result to fit into a word-size register.

5.2 Kronecker substitution

β is chosen to be 2^{16} in order to do not require any arithmetic operation. Can we do better with a larger beta ? What is the compromise between β and the m_i ?

Reconstruction from the almost β-adic to GMP integer using a splitting of the *beta*-adic expansion into four GMP integer. ? Can we do better do reduce the result mod $\prod m_i$.

5.2.1 *From integer to β-adic.* using gmp structure we can do this for free. only cast mpz data to uint16.

5.3 Linear storage for multi-modular matrix

explication lda, rda , RNSMajor ou MatrixMajor

6. BENCHMARKS

7. TODO

1) parallelisme 2) algo hybride (rapide/la)

REFERENCES

Barrett, P. 1986. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology, CRYPTO'86*. LNCS, vol. 263. Springer, 311–326.

Bernstein, D. J. 2008. *Algorithmic Number Theory*. Vol. 44. MSRI Publications, Chapter Fast multiplication and its applications, 325–384.

Brent, R. and Zimmermann, P. 2010. *Modern Computer Arithmetic*. Cambridge University Press, New York, NY, USA.

Gathen, J. V. Z. and Gerhard, J. 2003. *Modern Computer Algebra*, 3 ed. Cambridge University Press, New York, NY, USA.

Le Gall, F. 2014. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. ISSAC '14. ACM, New York, NY, USA, 296–303.

Montgomery, P. L. 1985. Modular multiplication without trial division. *Mathematics of Computation 44,* 170 (Apr.), 519–521.