Effectively multiplication- and division-free residue number systems

Eugene Zima

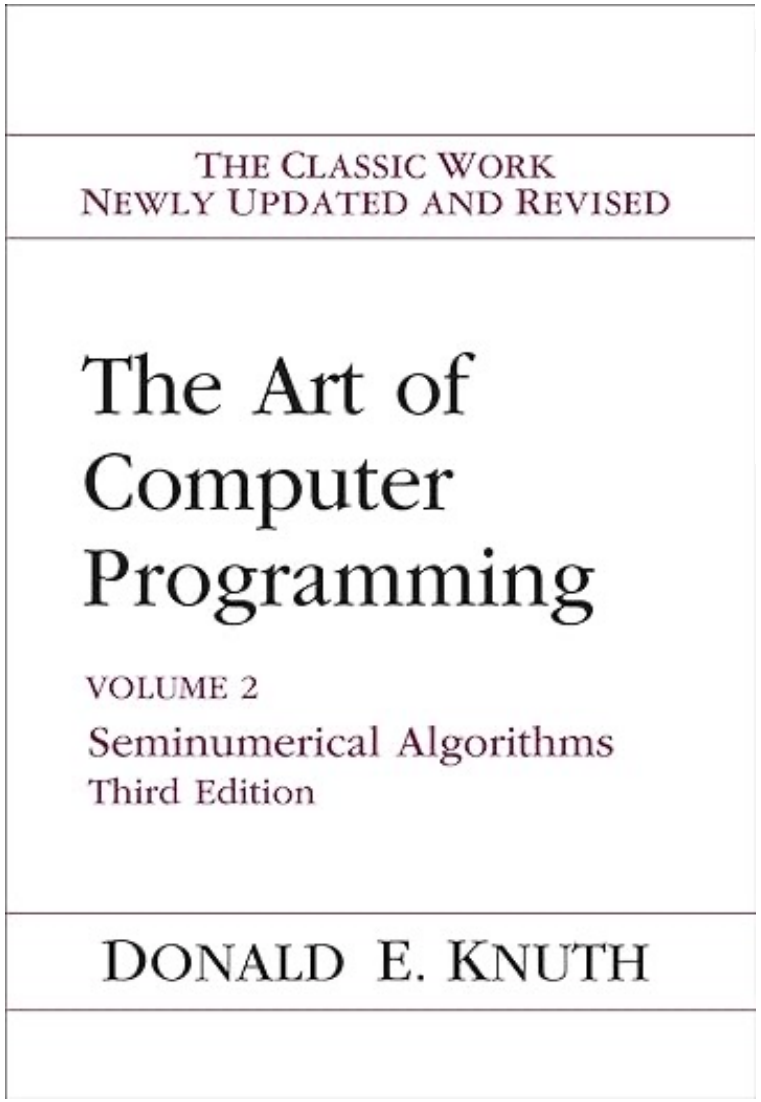
Physics and Computer Science Department

Wilfrid Laurier University, Waterloo

e-mail: ezima@wlu.ca

Joint work with:

Alex Stewart (2005-2007), Yu Li (2018), Benjamin Chen (2022-2024)



*4.3.2. Modular Arithmetic

Another interesting alternative is available for doing arithmetic on large integer numbers, based on some simple principles of number theory. The idea is to have several *moduli* m_1, m_2, ..., m_r that contain no common factors, and to work indirectly with *residues* $u \bmod m_1$, $u \bmod m_2$, ..., $u \bmod m_r$ instead of directly with the number u.

*4.3.3. How Fast Can We Multiply?

- Intro and generic requirements to the moduli
- Mersenne type of moduli
- Fermat type of moduli
- Sparse balanced binary number system
- Three-term moduli (trinomials)
- Many small primes & two-level RNS (application)
- Conclusion

Intro

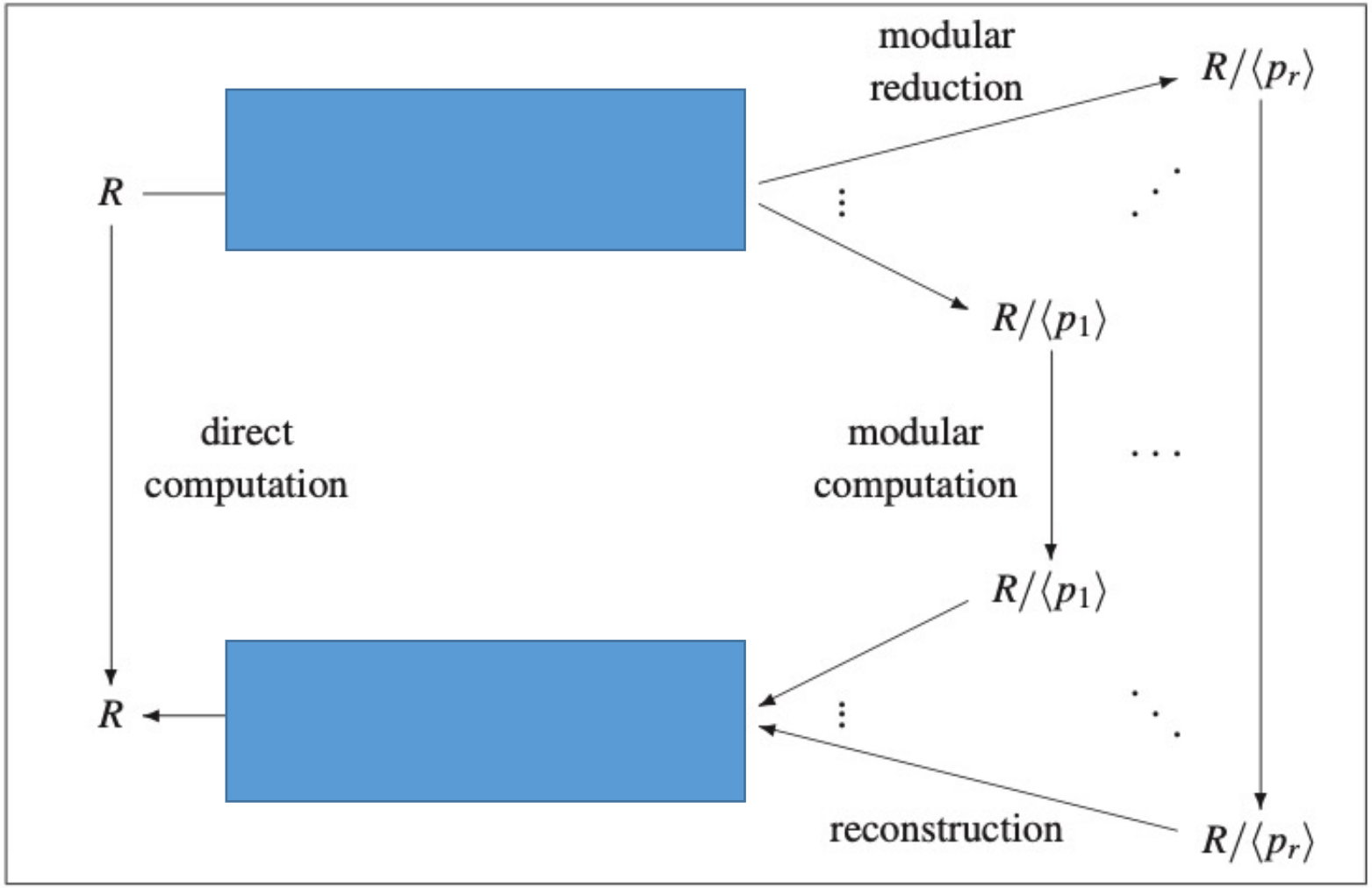


FIGURE 5.2: General scheme for small primes modular algorithms.

Intro

If bit-complexity of direct computations is $M(n)$ the "modular" approach reduces it to $rM\left(\frac{n}{r}\right)$ plus some "overhead".

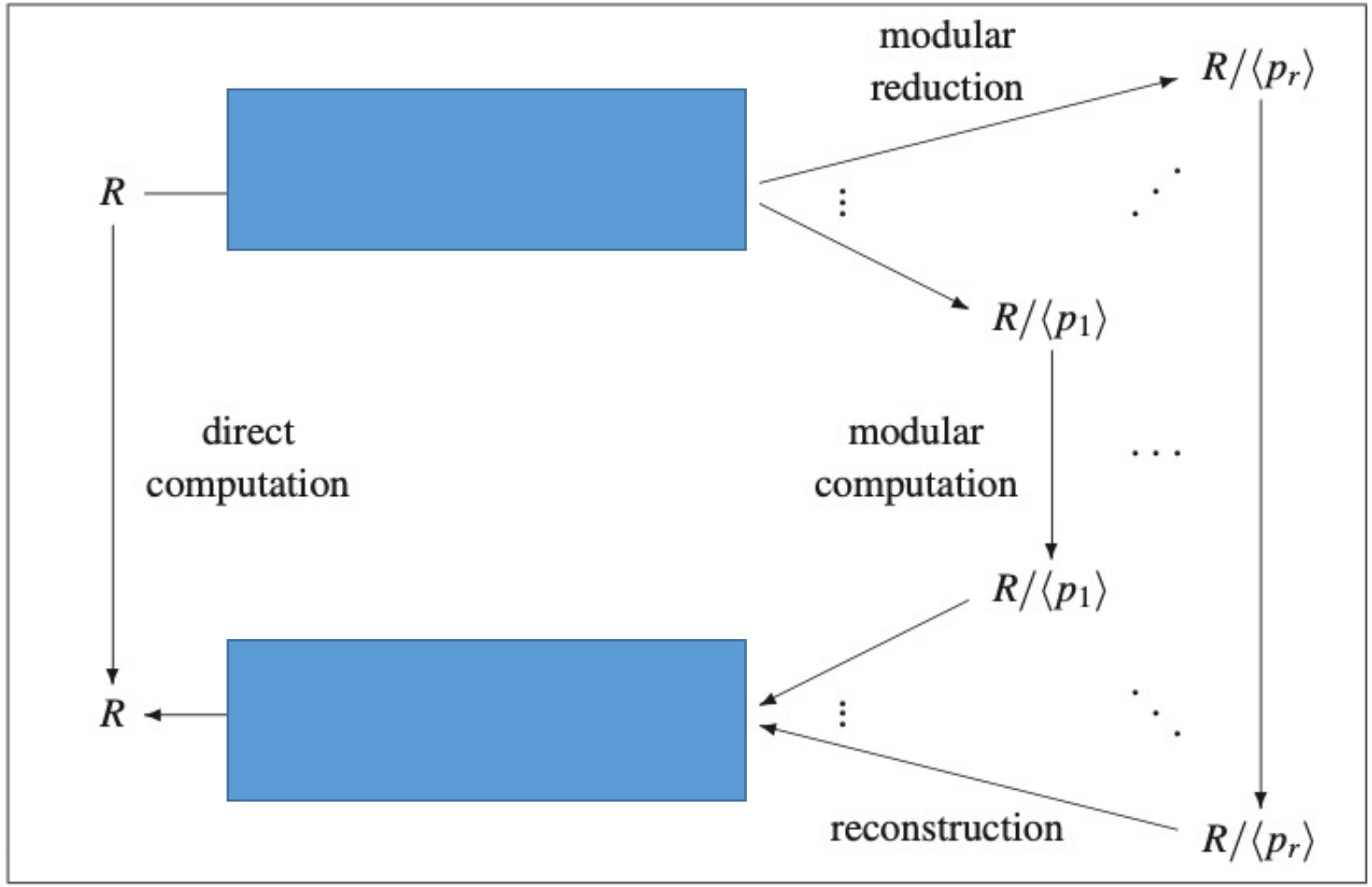


FIGURE 5.2: General scheme for small primes modular algorithms.

Intro

If bit-complexity of direct computations is $M(n)$ the "modular" approach reduces it to $rM\left(\frac{n}{r}\right)$ plus some "overhead".

Additionally

- the size of intermediate results is controlled
- computations modulo $p_i, i = 1, \ldots, r$ can be done in parallel

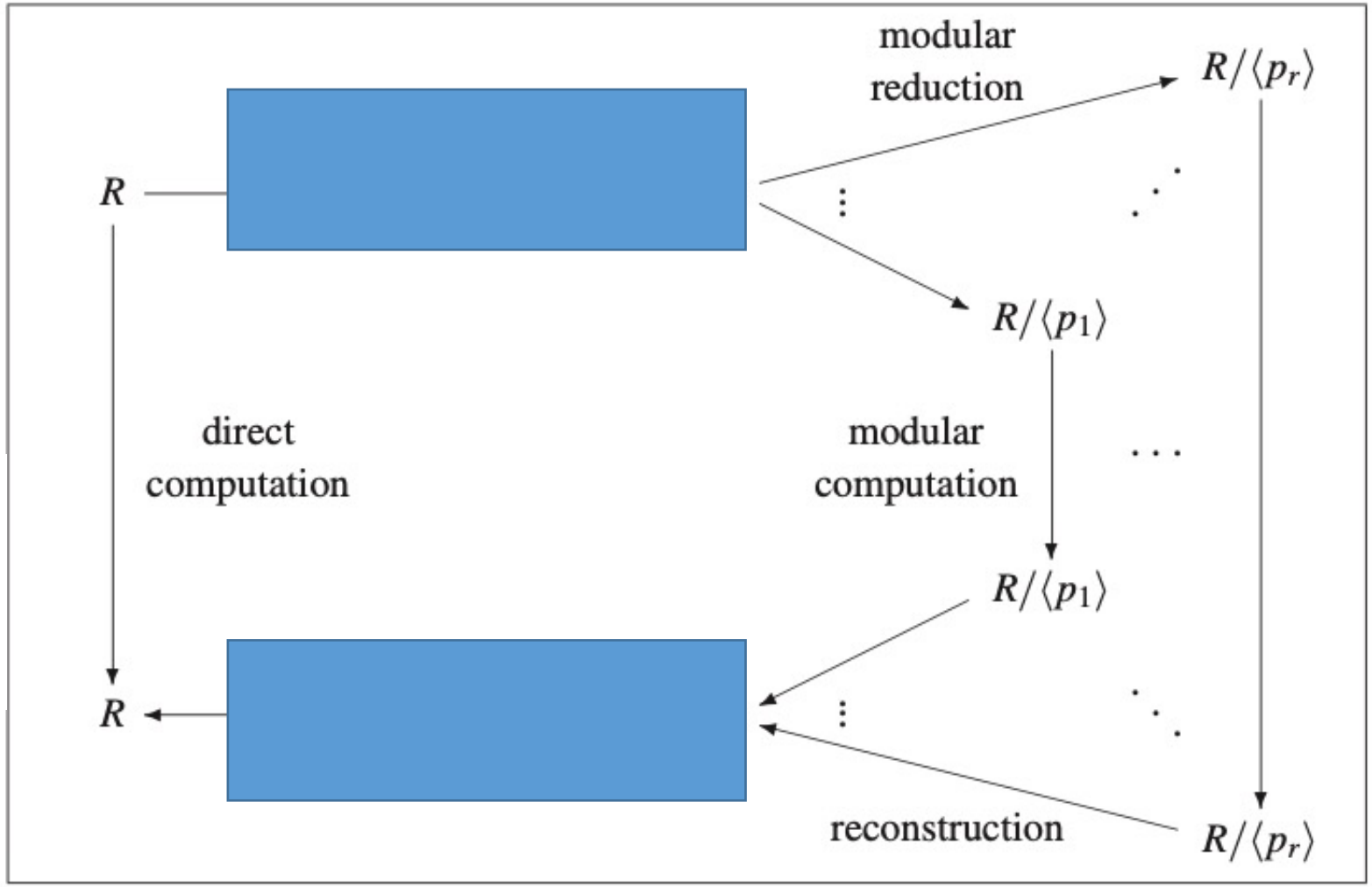


FIGURE 5.2: General scheme for small primes modular algorithms.

Similarity with evaluation and interpolation

reduction == evaluation
reconstruction == interpolation

Compare Garner’s reconstruction and Newton interpolation…

Selection of different evaluation points was studied extensively.
See for example

MULTIDIGIT MULTIPLICATION FOR MATHEMATICIANS

DANIEL J. BERNSTEIN

ABSTRACT. This paper surveys techniques for multiplying elements of various commutative rings. It covers Karatsuba multiplication, dual Karatsuba multiplication, Toom multiplication, dual Toom multiplication, the FFT trick, the twisted FFT trick, the split-radix FFT trick, Good's trick, the Schönhage-Strassen trick, Schönhage's trick, Nussbaumer's trick, the cyclic Schönhage-Strassen trick, and the Cantor-Kaltofen theorem. It emphasizes the underlying ring homomorphisms.

Selection of different evaluation points was studied extensively.
See for example

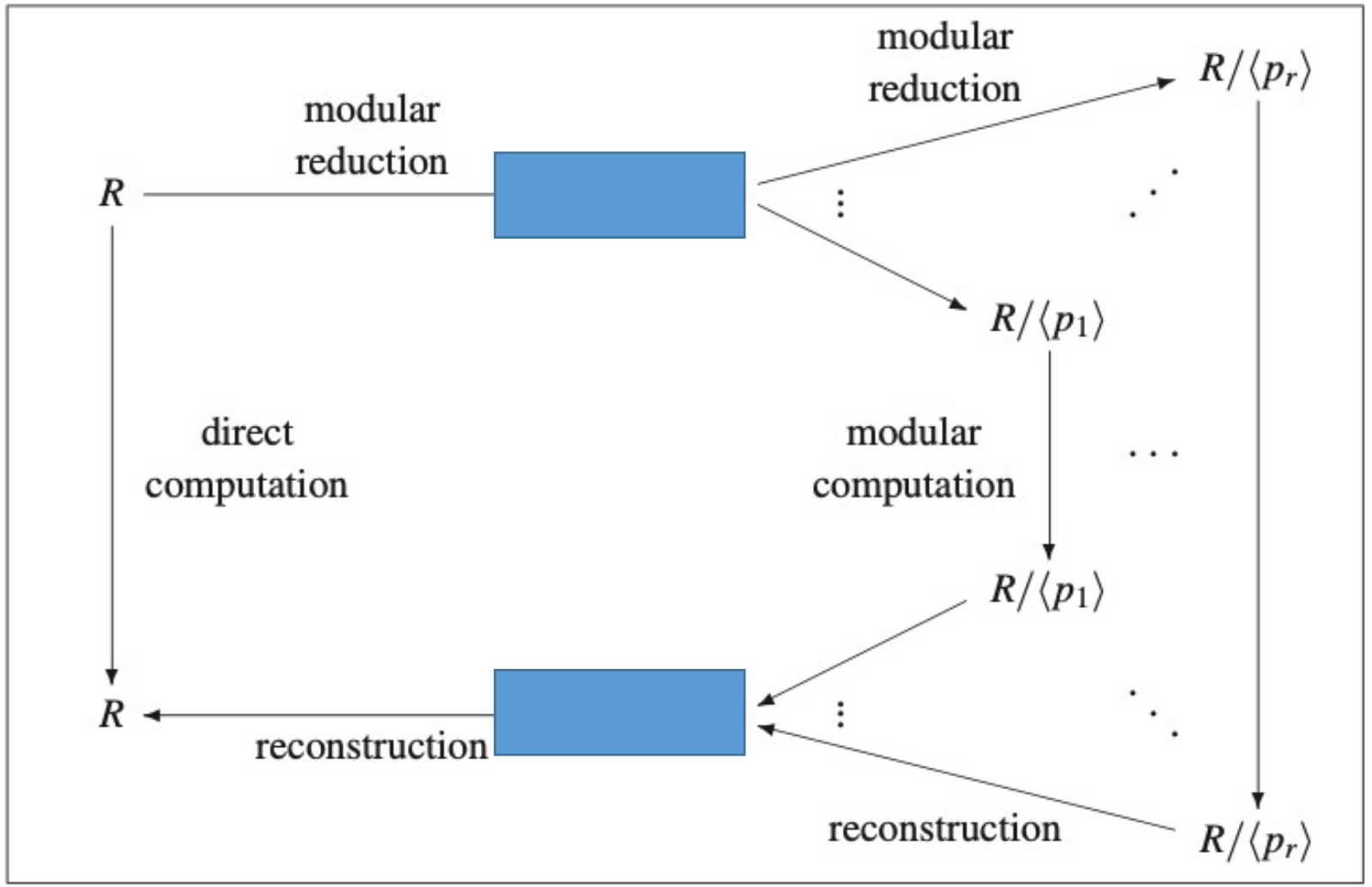
MULTIDIGIT MULTIPLICATION FOR MATHEMATICIANS

DANIEL J. BERNSTEIN

ABSTRACT. This paper surveys techniques for multiplying elements of various commutative rings. It covers Karatsuba multiplication, dual Karatsuba multiplication, Toom multiplication, dual Toom multiplication, the FFT trick, the twisted FFT trick, the split-radix FFT trick, Good's trick, the Schönhage-Strassen trick, Schönhage's trick, Nussbaumer's trick, the cyclic Schönhage-Strassen trick, and the Cantor-Kaltofen theorem. It emphasizes the underlying ring homomorphisms.

Selection of different moduli is not that popular, but still deserves investigation.

Requirements to the moduli



- relative primality
- “fast” reduction
- “fast” reconstruction
- balance in size
- scalability

Most ”complex” arithmetic operations involved in reduction/reconstruction:

- integer **division** with remainder (red)
- computation of modular **inverses** (rec)
- **multiplication** by moduli (rec)
- **multiplication** by the inverses (rec)

Some variations of reconstruction use

$$M_{ij} = m_i^{-1} \mod m_j, \quad (i < j)$$

Algorithm 5 Garner(r, m)

Require: $\forall i \in [0, N-1]: 0 \leq r_i < m_i$
Ensure: $a \equiv r_i \bmod m_i$

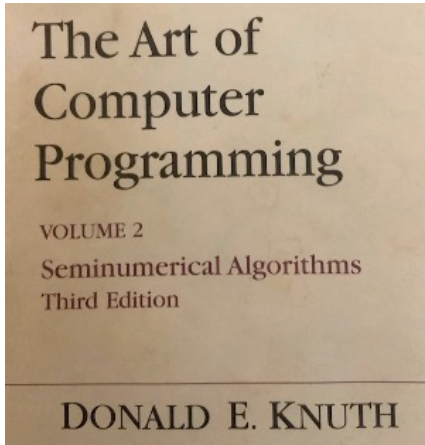
```
1: M ← 1
2: for i = 1 to N − 1 do
3:     M ← M m_{i−1}
4:     M_i ← M^{−1} mod m_i
5: end for
6: a_0 ← r_0
7: for i = 1 to N − 1 do
8:     t ← a_{i−1}
9:     for j = i − 2 downto 0 do
10:        t ← t m_j
11:        t ← t + a_j
12:    end for
13:    t ← r_i − t
14:    a_i ← t M_i mod m_i
15: end for
16: a ← a_{N−1}
17: for i = N − 1 downto 0
18:    a ← a m_i
19:    a ← a + a_i
20: end for
21: return a
```

How to chose the moduli making both conversion to RNS and reconstruction as efficient as possible?

Mersenne type of moduli

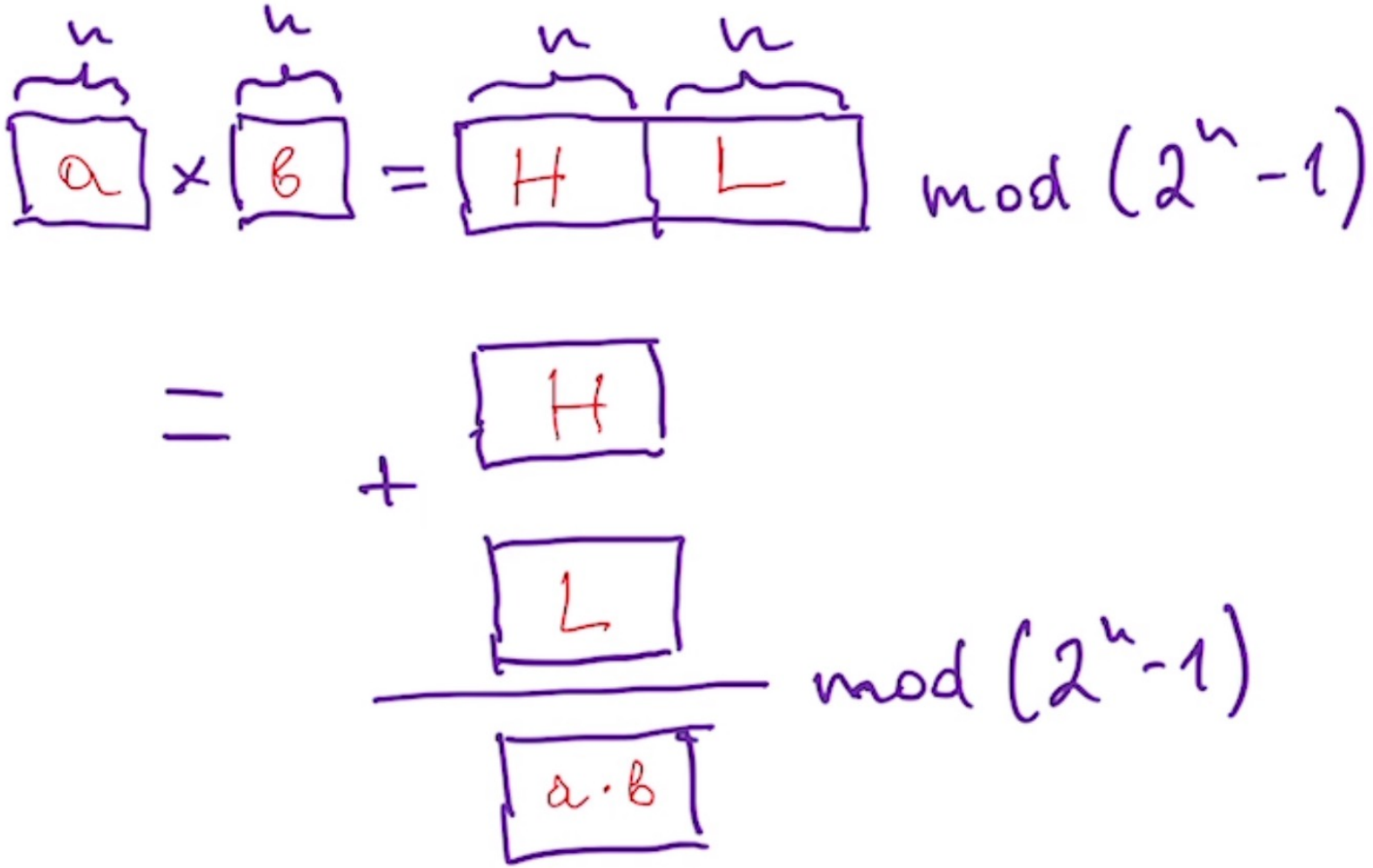
Several relatively prime moduli of the form 2^n-1 are selected (using $\gcd(2^n-1, 2^m-1) = 1$ if and only if $\gcd(n, m) = 1$).

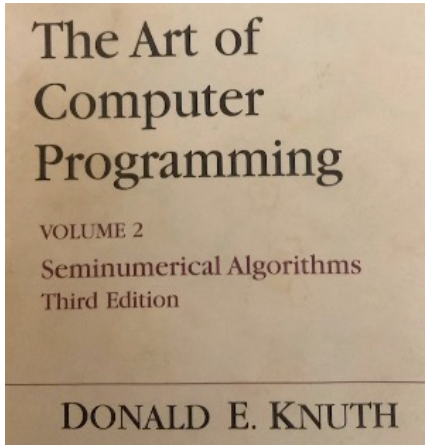
This replaces division with remainder in the residue computation by shift and addition operations that are much simpler (using that $2^n \equiv 1 \bmod (2^n - 1)$, the remainder from division of x by $2^n - 1$ can be obtained by splitting x into several numbers of bit-length n from right to left and adding them modulo $2^n - 1$).



Mersenne type of moduli

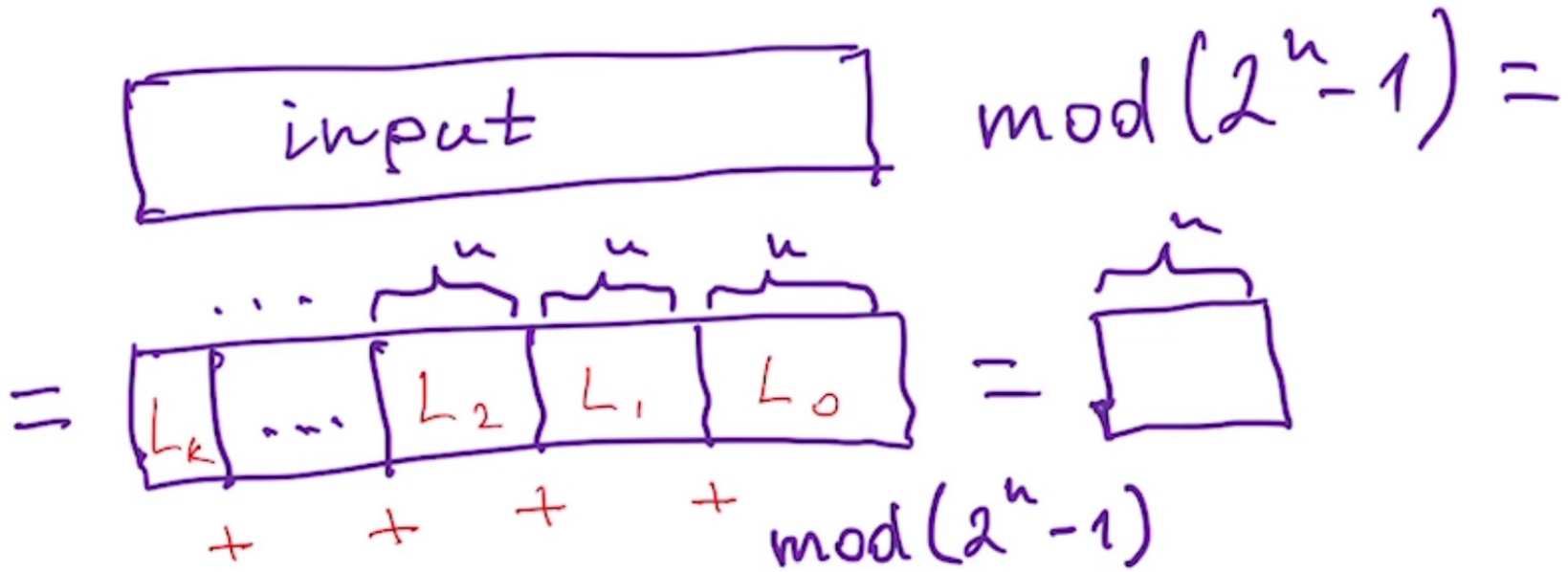
$$2^n \equiv 1 \mod (2^n - 1)$$

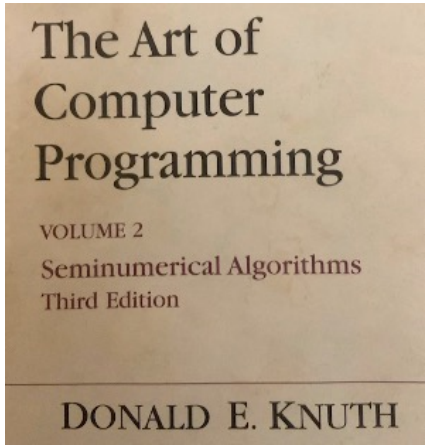




Mersenne type of moduli

$$2^n \equiv 1 \mod (2^n - 1)$$





Mersenne type of moduli

- relative primality

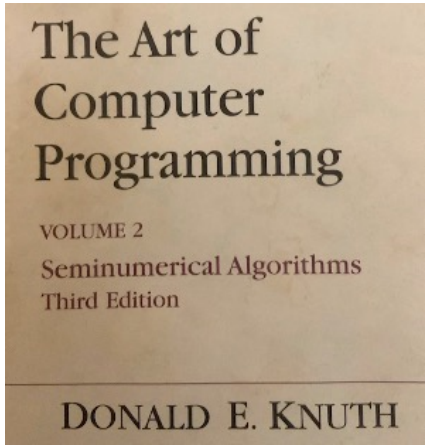
$$\gcd(2^e - 1,\ 2^f - 1) = 2^{\gcd(e,f)} - 1$$

- reduction is fast

- ~~reconstruction is fast~~ (multiplications by moduli are fast, multiplication by “precomputed” inverses are not)

- balance in size

- scala~~bility~~y



Mersenne type of moduli

- relative primality

$$\gcd(2^e - 1,\ 2^f - 1) = 2^{\gcd(e,f)} - 1$$

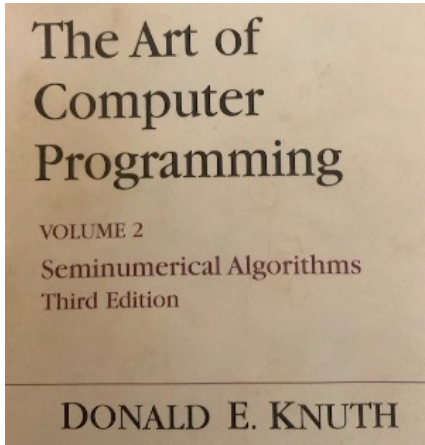
- reduction is fast

$$2^n \equiv 1 \mod (2^n - 1)$$

- ~~reconstruction is fast~~ (multiplications by moduli are fast, multiplication by “precomputed” inverses are not)

- balance in size

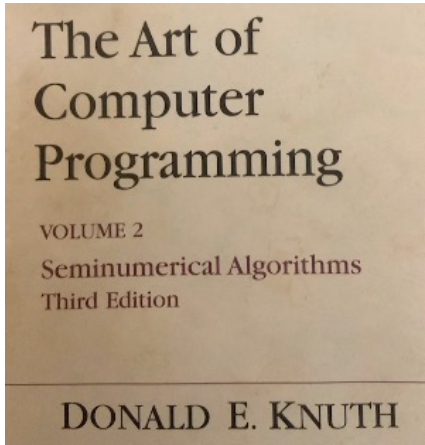
- scala~~bility~~y



Mersenne type of moduli (scalability)

One of the earliest (semi-successful use of Mersenne type of moduli in modular arithmetic):

[10] Schönhage A. Multiplikation großer Zahlen. *Computing* vol. 1, 1966, pp. 182–196.



Mersenne type of moduli (scalability)

One of the earliest (semi-successful use of Mersenne type of moduli in modular arithmetic):

In order to understand the essential mechanism of Schönhage's method, we shall look at a special case. Consider the sequence defined by the rules

$$q_0 = 1, \qquad q_{k+1} = 3q_k - 1, \tag{22}$$

so that $q_k = 3^k - 3^{k-1} - \cdots - 1 = \frac{1}{2}(3^k + 1)$. We will study a procedure that multiplies p_k-bit numbers, where $p_k = (18q_k + 8)$, in terms of a method for multiplying p_{k-1}-bit numbers. Thus, if we know how to multiply numbers having $p_0 = 26$ bits, the procedure to be described will show us how to multiply numbers of $p_1 = 44$ bits, then 98 bits, then 260 bits, etc., eventually increasing the number of bits by almost a factor of 3 at each step.

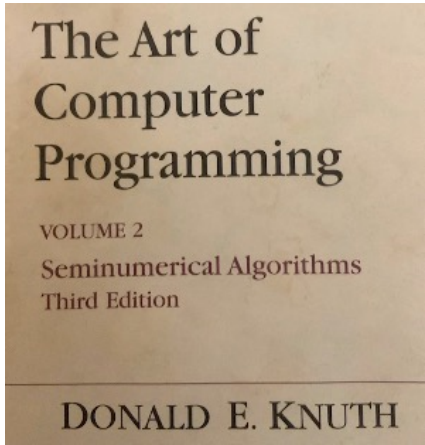
When multiplying p_k-bit numbers, the idea is to use the six moduli

$$\begin{aligned} m_1 &= 2^{6q_k-1} - 1, & m_2 &= 2^{6q_k+1} - 1, & m_3 &= 2^{6q_k+2} - 1, \\ m_4 &= 2^{6q_k+3} - 1, & m_5 &= 2^{6q_k+5} - 1, & m_6 &= 2^{6q_k+7} - 1. \end{aligned} \tag{23}$$

These moduli are relatively prime, by Eq. 4.3.2–(19), since the exponents

$$6q_k - 1, \quad 6q_k + 1, \quad 6q_k + 2, \quad 6q_k + 3, \quad 6q_k + 5, \quad 6q_k + 7 \tag{24}$$

are always relatively prime (see exercise 6). The six moduli in (23) are capable of representing numbers up to $m = m_1m_2m_3m_4m_5m_6 > 2^{36q_k+16} = 2^{2p_k}$, so there is no chance of overflow in the multiplication of p_k-bit numbers u and v.



Mersenne type of moduli (scalability)

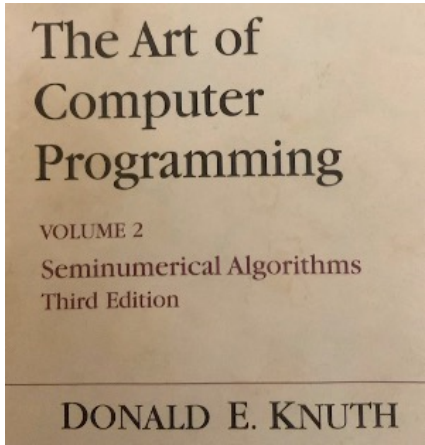
One of the earliest (semi-successful use of Mersenne type of moduli in modular arithmetic):

The reader will find it instructive to study the ingenious method represented by (32) and (33) very carefully. Similar techniques are discussed in Section 4.6.3.

Schönhage's paper [*Computing* **1** (1966), 182–196] shows that these ideas can be extended to the multiplication of n-bit numbers using $r \approx 2^{\sqrt{2 \lg n}}$ moduli, obtaining a method analogous to Algorithm T. We shall not dwell on the details here, since Algorithm T is always superior; in fact, an even better method is next on our agenda.

$$T(n) = (2r)\,T\left(\frac{n}{r}\right) + \tilde{O}(n)$$

$$T(n) = (2r-1)\,T\left(\frac{n}{r}\right) + \tilde{O}(n)$$



Mersenne type single modulus

Side note (from 1960):

The Use of Index Calculus and Mersenne Primes for the Design of a High-Speed Digital Multiplier*

AVIEZRI S. FRAENKEL

University of California, Los Angeles

Introduction

In recent years there has been an interest in unconventional number representations for computer number systems [1, 2]. The present paper considers a system in which the two numbers entering a multiplication are transformed into *indices*. These indices are *added* in their own number system. The sum, when converted back, gives the product.

As with logarithms, multiplication is thus replaced by the faster process of addition. In fact, a^n may be computed by the single operation of multiplying the index of a by n. Using indices that are integers the product is exact, which is not generally true for logarithms. However, the difference of two indices corresponds to the quotient only when the latter is an integer. No easy way has been found for performing division in other cases.

Properties of indices corresponding to *Mersenne Primes* are derived and used to save mechanization or storage requirements for the conversion of numbers into indices and vice versa. For a computer with a large word length, the required storage is still quite extensive.

Fermat type of moduli

Different strategies of selecting the moduli of the form $2^n + 1$ were considered in [12]. Relative primality of such moduli guaranteed by the proper choice of exponents driven by the following fact:

$$\gcd(2^m + 1, 2^n + 1) = 1 \iff v_2(m) \neq v_2(n),$$

where $v_2(x)$ is the binary valuation of x (the number of trailing zeros in the binary representation of x).

Fermat type of moduli

Different strategies of selecting the moduli of the form $2^n + 1$ were considered in [12]. Relative primality of such moduli guaranteed by the proper choice of exponents driven by the following fact:

$$\gcd(2^m + 1, 2^n + 1) = 1 \iff v_2(m) \neq v_2(n),$$

where $v_2(x)$ is the binary valuation of x (the number of trailing zeros in the binary representation of x).

This is a corollary of a more general result: for any positive integers $a, n, m, a > 1$ [4],

4. Cade, J.J., Kee-Wai, Lau, Pedersen, A., and Lossers, O.P., Problem E3288. Problems and Solutions, *The Am. Math. Monthly*, 1990, vol. 97, no. 4, pp. 344–345.

$$\mathrm{GCD}(a^m + 1, a^n + 1) = \begin{cases} a^{\mathrm{GCD}(m,n)} + 1, & \text{if } \nu_2(m) = \nu_2(n) \\ 1, & \text{if } \nu_2(m) \neq \nu_2(n) \text{ and } a \text{ is even} \\ 2, & \text{if } \nu_2(m) \neq \nu_2(n) \text{ and } a \text{ is odd.} \end{cases}$$

Fermat type of moduli

Different strategies of selecting the moduli of the form $2^n + 1$ were considered in [12]. Relative primality of such moduli guaranteed by the proper choice of exponents driven by the following fact:

$$\gcd(2^m + 1, 2^n + 1) = 1 \iff v_2(m) \neq v_2(n),$$

where $v_2(x)$ is the binary valuation of x sentation of x).

In the book [5] devoted to Fermat numbers [5] (which appeared 11 years later after publication [4]), only a particular case of this result is proven:

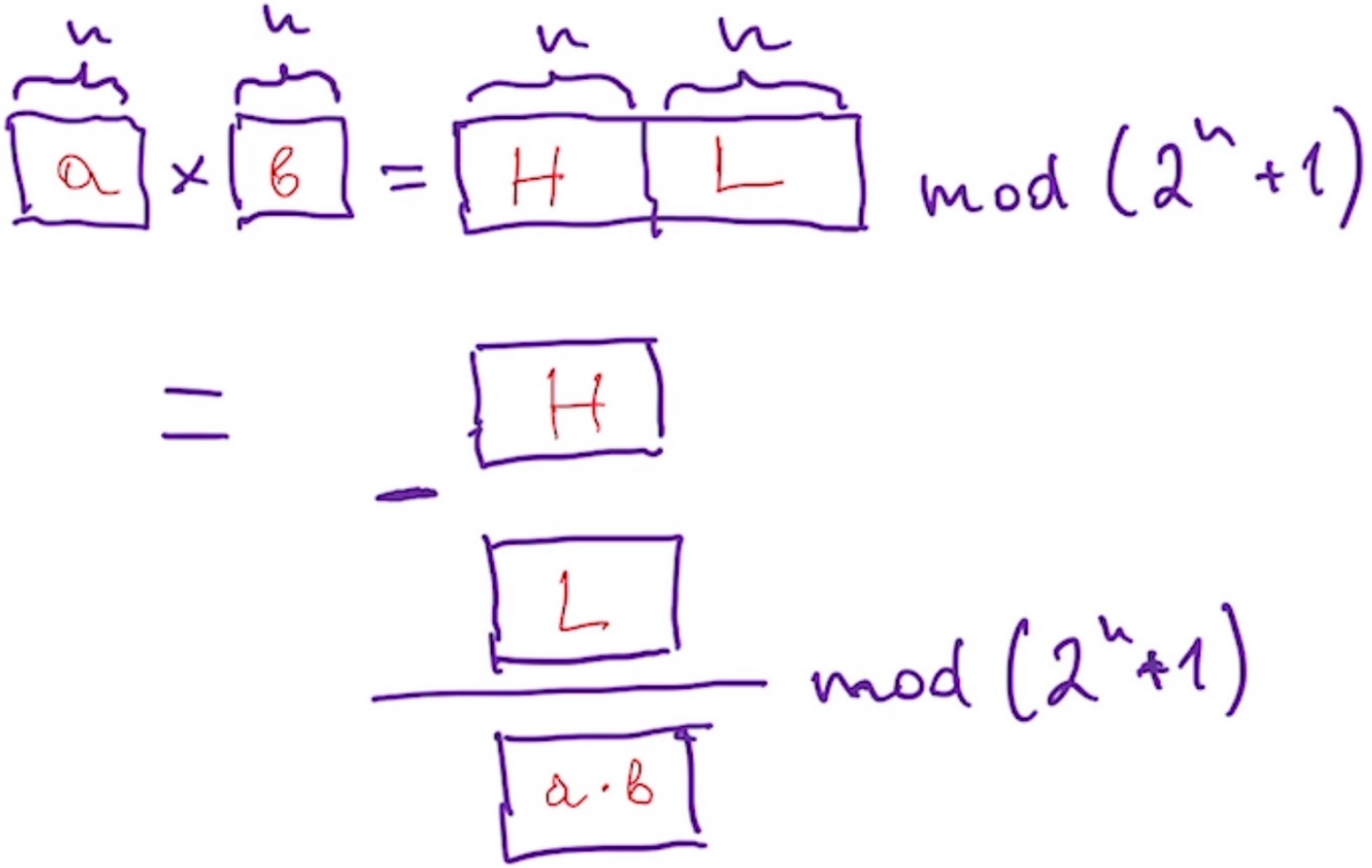
$$\mathrm{GCD}(2^m + 1, 2^{mn} + 1) = \begin{cases} 1, & \text{if } n \text{ is even} \\ 2^m + 1, & \text{if } n \text{ is odd.} \end{cases}$$

5. Křížek, M., Luca, F., and Somer, L., *17 Lectures on Fermat Numbers: From Number Theory to Geometry.* New York: Springer, 2001.

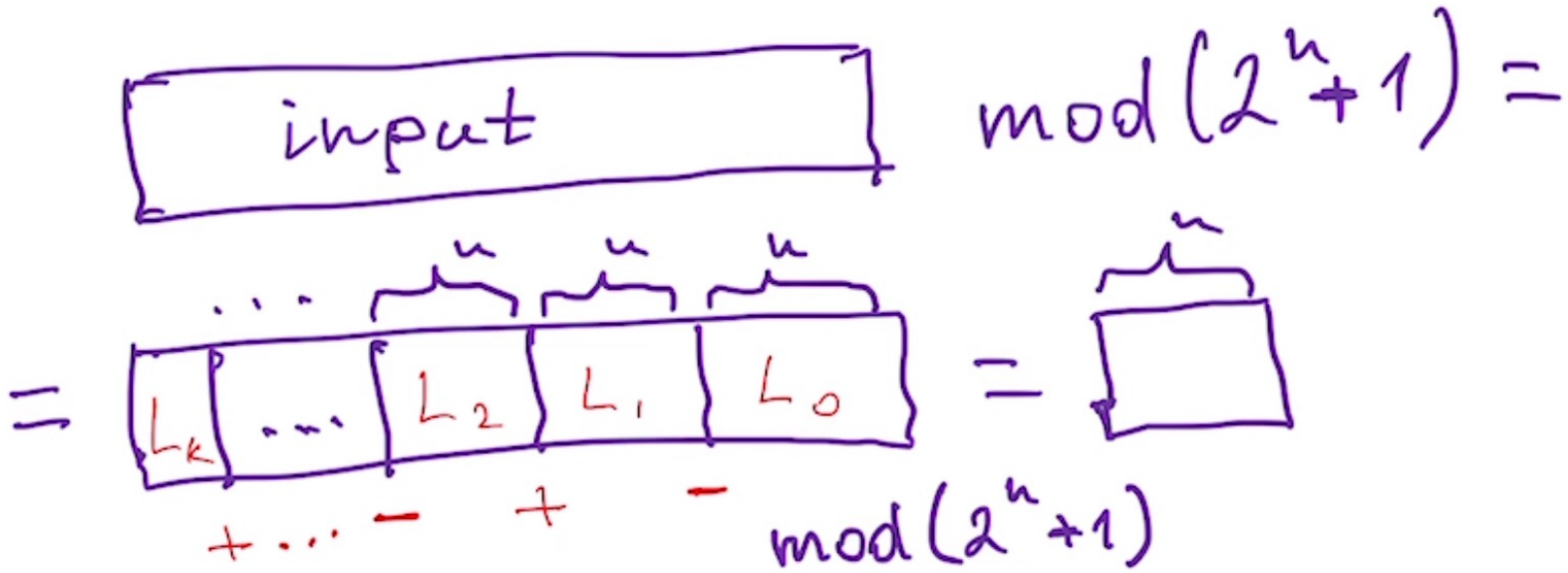
Fermat type of moduli

Using $2^n \equiv -1 \bmod (2^n + 1)$, the remainder from division of x by $2^n + 1$ can be obtained by splitting x into several numbers of bit-length n from right to left and subtracting/adding them modulo $2^n + 1$ (see [12] for details).

Fermat type of moduli



Fermat type of moduli



Fermat type of moduli

Block strategy

$$m_i = 2^{2^n - 2^i} + 1, \quad i = 0, 1, \ldots, k$$

$$n > k$$

Shift strategy

$$m_i = 2^{a2^i} + 1, \quad i = 0, 1, \ldots, k$$

Space complexity comment …

Fermat type of moduli (shift strategy)

Consider moduli of the form $m_i = 2^{a2^i} + 1, \quad i = 0, 2, \ldots, k$, where a is an arbitrary positive integer, and products $M_i = \prod_{j=0}^{i-1} m_j = \prod_{j=0}^{i-1} \left(2^{a2^j} + 1\right), \quad i = 0, 1, \ldots, k-1$. Then

$$M_i^{-1} \bmod m_i = 2^{a2^i - 1} - 2^{a-1} + 1, \quad i = 1, 2, ..., k. \tag{1}$$

Fermat type of moduli (shift strategy)

Consider moduli of the form $m_i = 2^{a2^i} + 1, \quad i = 0, 2, \ldots, k$, where a is an arbitrary positive integer, and products $M_i = \prod_{j=0}^{i-1} m_j = \prod_{j=0}^{i-1} \left(2^{a2^j} + 1\right), \quad i = 0, 1, \ldots, k-1$. Then

$$M_i^{-1} \bmod m_i = 2^{a2^i - 1} - 2^{a-1} + 1, \quad i = 1, 2, ..., k. \tag{1}$$

With this choice of moduli there is no need to (pre-)compute and to store inverses. Inverse is defined by the value of a (which is the same for all moduli) and the index i. This allows reconstruction to become essentially multiplication-free: multiplication by the sparse inverse is just 2 shifts, 1 addition, and 1 subtraction. When $a = 1$ (i.e., the moduli are consecutive Fermat numbers), $M_i^{-1} \bmod m_i = 2^{2^i - 1}, \quad i = 1, 2, ...$ and multiplication by the inverse requires shift only.

Fermat type of moduli (shift strategy)

Consider moduli of the form $m_i = 2^{a2^i} + 1, \quad i = 0, 2, \ldots, k$, where a is an arbitrary positive integer, and products $M_i = \prod_{j=0}^{i-1} m_j = \prod_{j=0}^{i-1} \left(2^{a2^j} + 1\right), \quad i = 0, 1, \ldots, k-1$. Then

$$M_i^{-1} \bmod m_i = 2^{a2^i - 1} - 2^{a-1} + 1, \quad i = 1, 2, ..., k. \tag{1}$$

With this choice of moduli there is no need to (pre-)compute and to store inverses. Inverse is defined by the value of a (which is the same for all moduli) and the index i. This allows reconstruction to become essentially multiplication-free: multiplication by the sparse inverse is just 2 shifts, 1 addition, and 1 subtraction. When $a = 1$ (i.e., the moduli are consecutive Fermat numbers), $M_i^{-1} \bmod m_i = 2^{2^i - 1}, \quad i = 1, 2, ...$ and multiplication by the inverse requires shift only.

However, such choice of moduli does not satisfy balance requirement. In fact, the bit length of m_i is larger than the bit-length of product $m_0 m_1 \ldots m_{i-1}$.

Fermat type of moduli (other bases)

A more general result is valid for an arbitrary numerical system with an even base B. Consider moduli of type $m_i = B^{2^i} + 1$ $(i = 0, 1, \ldots, k)$ and products

$$M_i = \prod_{j=0}^{i-1} m_j = \prod_{j=0}^{i-1} (B^{2^j} + 1) \quad (i = 1, 2, \ldots, k).$$

Proposition 2.

$$M_i^{-1} \bmod m_i = \frac{B^{2^i} - B + 2}{2}, \quad i = 1, 2, \ldots, k.$$

Fermat type of moduli

Block strategy

- relative primality

$$\gcd(2^m + 1, 2^n + 1) = 1 \iff v_2(m) \neq v_2(n)$$

- reduction is fast

$$2^n \equiv -1 \mod (2^n + 1)$$

- ~~reconstruction is fast~~ (same as Mersenne)

- balance in size

- scalability ?

Shift strategy

- relative primality

- reduction is fast

- reconstruction is fast

- ~~balance in size~~

- scalability

Simple *scalability* is in place if multiplying all the exponents by the same natural a preserves relative primality, and probably some other interesting properties.

Sparse balanced binary numbers

Consider $x \in \mathbb{Z}$ represented as

$$x = \sum_{i=0}^{n} b_i 2^i \text{ with } b_i \in \{-1, 0, 1\}. \qquad (2)$$

This representation looks similar to the *balanced ternary representation* [TAoCP] but uses base 2 instead of base 3. We call (2) *balanced binary representation* and note that it shares many useful properties with balanced ternary representation. For example,

(a) most significant digit in (2) defines the sign of number x.

(b) $x = 0$ if and only if $b_i = 0, i = 0, 1, \ldots n$. Equivalently, if there is a value of index i such that $b_i \neq 0$ then $x \neq 0$.

Representation (2) is not unique. However, with little additional effort this can be fixed.

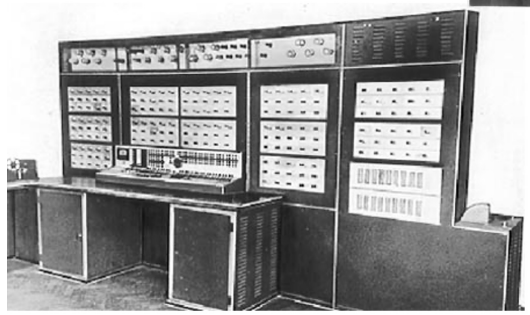
$$30 = 11110_{(2)}$$

$$30 = 100\bar{1}10$$

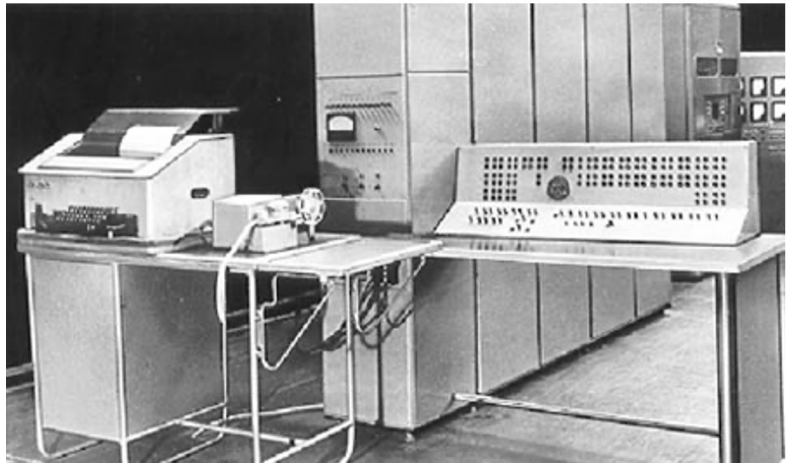
$$30 = 1000\bar{1}0$$

Николай Петрович Брусенцов

Сетунь, экспериментальная модель 1958



Сетунь, серийная модель, выпущено 50 шт.



Sparse balanced binary numbers

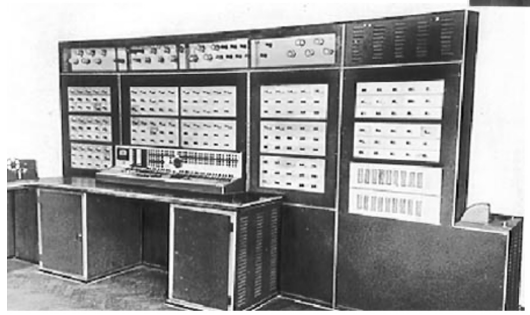
We say that a number $x \in \mathbb{Z}$ is in *sparse balanced binary representation* if

$$x = \sum_{i=0}^{n} b_i 2^i \text{ with } b_i \in \{-1, 0, 1\} \text{ and } b_i \cdot b_{i+1} = 0 \text{ for } i = 0, 1, \ldots, n-1, \quad (3)$$

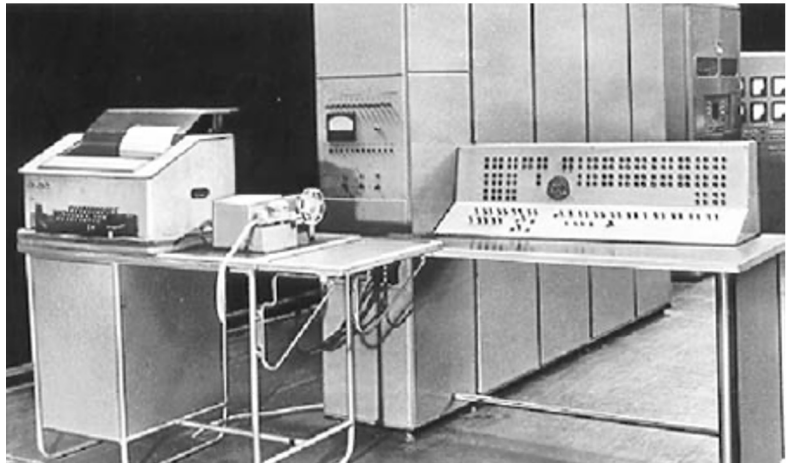
i.e., it is represented as in (2) with extra requirement: no two consecutive bits are set.

Николай Петрович Брусенцов

Сетунь, экспериментальная модель 1958



Сетунь, серийная модель, выпущено 50 шт.



Sparse balanced binary numbers

We say that a number $x \in \mathbb{Z}$ is in *sparse balanced binary representation* if

$$x = \sum_{i=0}^{n} b_i 2^i \text{ with } b_i \in \{-1, 0, 1\} \text{ and } b_i \cdot b_{i+1} = 0 \text{ for } i = 0, 1, \ldots, n-1, \quad (3)$$

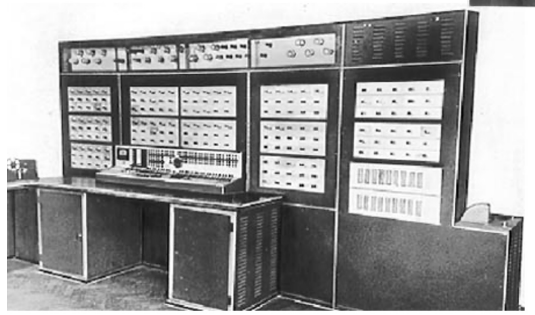
i.e., it is represented as in (2) with extra requirement: no two consecutive bits are set.

1. Every number $x \in \mathbb{Z}$ has unique sparse balanced binary representation.

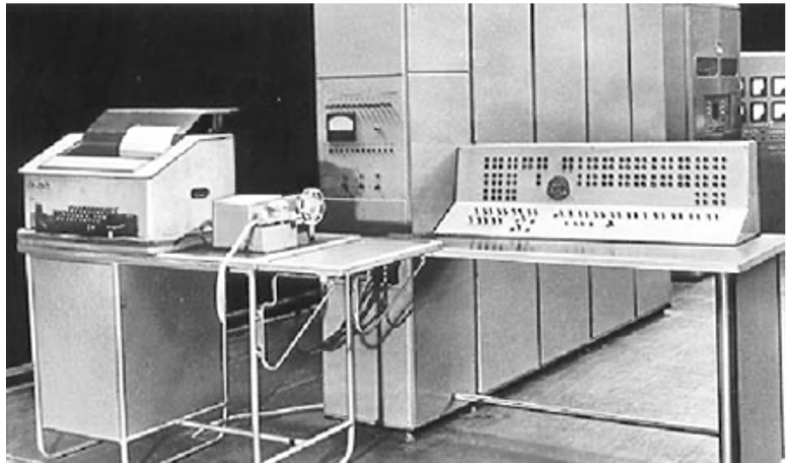
2. Sparse balanced binary representation of $x \in \mathbb{Z}$ is the sparsest one: it has minimal number of bits set among all possible balanced binary representations of x.

Николай Петрович Брусенцов

Сетунь, экспериментальная модель 1958



Сетунь, серийная модель, выпущено 50 шт.



Sparse balanced binary numbers

Given x as in (3) and natural u,

$$\mathrm{Scal}(u, x) = \sum_{i=0}^{n} b_i 2^{ui}.$$

This is the same as change of base from 2 to 2^u.

Fermat type moduli scale (preserve relative primality under scaling), Mersenne type moduli do not scale

$$2^m + 1 \perp 2^n + 1 \iff \nu_2(m) \neq \nu_2(n) \qquad \gcd(2^{um} - 1, 2^{un} - 1) \neq 1 \text{ for } u > 1$$

$$2^{um} + 1 \perp 2^{un} + 1 \iff \nu_2(um) \neq \nu_2(un)$$

$$\nu_2(uv) = \nu_2(u) + \nu_2(v)$$

Sparse balanced binary numbers

"Some other interesting property" is computed inverses preserve sparsity under scaling operation

For example, for $a = 2^{224} + 1$ and $b = 2^{192} + 1$ we have

$$a^{-1} \mod b = 2^{191} + 2^{159} + 2^{127} + 2^{95} + 2^{63} + 2^{31} + 1.$$

Now, if we scale moduli by factor $u = 100$ and consider $\tilde{a} = 2^{22400} + 1$ and $\tilde{b} = 2^{19200} + 1$, then

$$\tilde{a}^{-1} \mod \tilde{b} = 2^{19199} + 2^{15999} + 2^{12799} + 2^{9599} + 2^{6399} + 2^{3199} + 1.$$

Sparse balanced binary numbers

"Some other interesting property" is computed inverses preserve sparsity under scaling operation

For example, for $a = 2^{224} + 1$ and $b = 2^{192} + 1$ we have

$$a^{-1} \mod b = 2^{191} + 2^{159} + 2^{127} + 2^{95} + 2^{63} + 2^{31} + 1.$$

Now, if we scale moduli by factor $u = 100$ and consider $\tilde{a} = 2^{22400} + 1$ and $\tilde{b} = 2^{19200} + 1$, then

$$\tilde{a}^{-1} \mod \tilde{b} = 2^{19199} + 2^{15999} + 2^{12799} + 2^{9599} + 2^{6399} + 2^{3199} + 1.$$

$$m_i = 2^{2^n - 2^i} + 1, \quad i = 0, 1, \ldots, k$$

$$n > k$$

… start with i>1 and sparsity of inverses will be preserved under scaling

Sparse balanced binary numbers

"Some other interesting property" is computed inverses preserve sparsity under scaling operation

2 Some basic facts

Let u and v be natural with $u > v$. Write $u = qv + r,\ (0 \le r < v)$.
Then

1) $2^u + 1 = Q(2^v - 1) + R,\ (0 \le R < 2^v - 1)$ with

$$R = 2^r + 1, \quad Q = 2^{u-v} + 2^{u-2v} + \cdots + 2^{u-qv}.$$

2) $2^u - 1 = Q(2^v - 1) + R,\ (0 \le R < 2^v - 1)$ with

$$R = 2^r - 1, \quad Q = 2^{u-v} + 2^{u-2v} + \cdots + 2^{u-qv}.$$

3) $2^u + 1 = Q(2^v + 1) + R,\ (0 \le R < 2^v + 1)$ with

$$R = 2^r + 1, \quad Q = 2^{u-v} - 2^{u-2v} + \cdots - 2^{u-qv}$$

for even q, and

$$R = 2^v - (2^r - 1) + 1, \quad Q = 2^{u-v} - 2^{u-2v} + \cdots + 2^{u-qv} - 1$$

for odd q.

4) $2^u - 1 = Q(2^v + 1) + R,\ (0 \le R < 2^v + 1)$ with

$$R = 2^r - 1, \quad Q = 2^{u-v} - 2^{u-2v} + \cdots - 2^{u-qv}$$

for even q, and

$$R = 2^v - 2^r, \quad Q = 2^{u-v} - 2^{u-2v} + \cdots + 2^{u-qv} - 1$$

for odd q.

Sparse balanced binary numbers

"Some other interesting property" is computed inverses preserve sparsity under scaling operation

2 Some basic facts

Let u and v be natural with $u > v$. Write $u = qv + r,\ (0 \le r < v)$.
Then

1) $2^u + 1 = Q(2^v - 1) + R,\ (0 \le R < 2^v - 1)$ with

$$R = 2^r + 1,\quad Q = 2^{u-v} + 2^{u-2v} + \cdots + 2^{u-qv}.$$

2) $2^u - 1 = Q(2^v - 1) + R,\ (0 \le R < 2^v - 1)$ with

$$R = 2^r - 1,\quad Q = 2^{u-v} + 2^{u-2v} + \cdots + 2^{u-qv}.$$

3) $2^u + 1 = Q(2^v + 1) + R,\ (0 \le R < 2^v + 1)$ with

$$R = 2^r + 1,\quad Q = 2^{u-v} - 2^{u-2v} + \cdots - 2^{u-qv}$$

for even q, and

$$R = 2^v - (2^r - 1) + 1,\quad Q = 2^{u-v} - 2^{u-2v} + \cdots + 2^{u-qv} - 1$$

for odd q.

4) $2^u - 1 = Q(2^v + 1) + R,\ (0 \le R < 2^v + 1)$ with

$$R = 2^r - 1,\quad Q = 2^{u-v} - 2^{u-2v} + \cdots - 2^{u-qv}$$

for even q, and

$$R = 2^v - 2^r,\quad Q = 2^{u-v} - 2^{u-2v} + \cdots + 2^{u-qv} - 1$$

for odd q.

Consider two moduli from a group of block-Fermat numbers: $2^{2^n-2^l}+1$ and $2^{2^n-2^m}+1$ with $m > l$. "Remainder" sequence for those looks like
$2^{2^n-2^l}+1$,
$2^{2^n-2^m}+1$,
after subtraction we get
$2^{2^n-2^m}(2^{2^m-2^l}-1)$ and after cancelling power of 2 we get
$2^{2^m-2^l}-1$.
Next reminder is from division of $2^{2^n-2^m}+1$ by $2^{2^m-2^l}-1$.
...

Sparse balanced binary numbers

"Some other interesting property" is computed inverses preserve sparsity under scaling operation

2 Some basic facts

Let u and v be natural with $u > v$. Write $u = qv + r,\ (0 \le r < v)$.
Then

1) $2^u + 1 = Q(2^v - 1) + R,\ (0 \le R < 2^v - 1)$ with

$$R = 2^r + 1,\quad Q = 2^{u-v} + 2^{u-2v} + \cdots + 2^{u-qv}.$$

2) $2^u - 1 = Q(2^v - 1) + R,\ (0 \le R < 2^v - 1)$ with

$$R = 2^r - 1,\quad Q = 2^{u-v} + 2^{u-2v} + \cdots + 2^{u-qv}.$$

3) $2^u + 1 = Q(2^v + 1) + R,\ (0 \le R < 2^v + 1)$ with

$$R = 2^r + 1,\quad Q = 2^{u-v} - 2^{u-2v} + \cdots - 2^{u-qv}$$

for even q, and

$$R = 2^v - (2^r - 1) + 1,\quad Q = 2^{u-v} - 2^{u-2v} + \cdots + 2^{u-qv} - 1$$

for odd q.

4) $2^u - 1 = Q(2^v + 1) + R,\ (0 \le R < 2^v + 1)$ with

$$R = 2^r - 1,\quad Q = 2^{u-v} - 2^{u-2v} + \cdots - 2^{u-qv}$$

for even q, and

$$R = 2^v - 2^r,\quad Q = 2^{u-v} - 2^{u-2v} + \cdots + 2^{u-qv} - 1$$

for odd q.

Consider two moduli from a group of block-Fermat numbers: $2^{2^n-2^l}+1$ and $2^{2^n-2^m}+1$ with $m > l$. "Remainder" sequence for those looks like
$2^{2^n-2^l}+1$,
$2^{2^n-2^m}+1$,
after subtraction we get
$2^{2^n-2^m}(2^{2^m-2^l}-1)$ and after cancelling power of 2 we get
$2^{2^m-2^l}-1$.
Next reminder is from division of $2^{2^n-2^m}+1$ by $2^{2^m-2^l}-1$.
...

The closed-form expression for $a^{-1} \mod b$ when $m = l + 1$ is

$$2^{2^n-2^m-1} + 2^{2^n-2^m-2^l-1} + \cdots + 2^{2^n-2^m-(q-1)\cdot 2^l-1} + 1$$

Alternatively,

$$2^{2^n-2^m-1} + 2^{2^n-2^m-2^l-1} + \cdots + 2^{2^l-1} + 1$$

Sparse balanced binary numbers

Repeat for scaled numbers:

For $a = 2^{(2^n - 2^l)u} + 1$ and $b = 2^{(2^n - 2^m)u} + 1$
the extended gcd sequence for general m, n:

s	t	g
1	0	$2^{(2^n-2^l)u} + 1$
0	1	$2^{(2^n-2^m)u} + 1$
1	-1	$2^{(2^n-2^m)u}(2^{(2^m-2^l)u} - 1)$
-1	$2^{(2^m-2^l)u}$	$2^{(2^m-2^l)u} - 1$

$$m = l + 1$$

The closed form expression for $a^{-1} \mod b$ is

$$2^{(2^n-2^m)u-1} + 2^{(2^n-2^m-2^l)u-1} + \cdots + 2^{(2^n-2^m-(q-1)\cdot 2^l)u-1} + 1$$

… or use Maple to choose ”satisfactory” group of moduli…

Notes:

Note that SBB-number is evaluation of a sparse polynomial from $\mathbb{Z}_3[x]$ at $x = 2$ (when symmetric representatives for $\mathbb{Z}_3$ are used); also scaling operation with scaling factor u is evaluation of the same polynomial at $x = 2^u$.

Moduli size grow fast with the number of moduli required.

Block Fermat numbers not perfectly balanced in size.

Three-term moduli

$$m_i = 2^n \pm 2^{k_i} \pm 1, \quad k_i < n$$

Three-term moduli

$$m_i = 2^n \pm 2^{k_i} \pm 1, \quad k_i < n$$

Balance is guaranteed by default

Three-term moduli

$$m_i = 2^n \pm 2^{k_i} \pm 1, \quad k_i < n$$

Balance is guaranteed by default

Simple Power Analysis on Fast Modular Reduction with NIST Recommended Elliptic Curves

Yasuyuki Sakai[1] and Kouichi Sakurai[2]

Three-term moduli

$$m_i = 2^n \pm 2^{k_i} \pm 1, \quad k_i < n$$

Balance is guaranteed by default

2.1 Generalized Mersenne Prime

In FIPS 186-2 NIST provides 5 recommended prime fields [2]. The order of the fields are shown below.

P-192: $p_{192} = 2^{192} - 2^{64} - 1$
P-224: $p_{224} = 2^{224} - 2^{96} + 1$
P-256: $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
P-384: $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
P-521: $p_{521} = 2^{521} - 1$

These recommended primes have a special form, which are referred to as generalized Mersenne prime. This form permits fast modular reduction. Solinas gave fast reduction algorithms for such the prime [4,15]. The following Algorithms 1, 2, 3, 4 and 5 show the dedicated reduction algorithms for p_{192} p_{224}, p_{256}, p_{384} and p_{521}, respectively.

Three-term moduli

$$m_i = 2^n \pm 2^{k_i} \pm 1, \quad k_i < n$$

Balance is guaranteed by default

GENERALISED MERSENNE NUMBERS REVISITED

ROBERT GRANGER AND ANDREW MOSS

ABSTRACT. Generalised Mersenne Numbers (GMNs) were defined by Solinas in 1999 and feature in the NIST (FIPS 186-2) and SECG standards for use in elliptic curve cryptography. Their form is such that modular reduction is extremely efficient, thus making them an attractive choice for modular mul-

Three-term moduli

Consider $m = 2^n - 2^k + 1$ and a $2n$-bit number x. Using $2^n \equiv 2^k - 1 \bmod m$ one can compute (in division/multiplication-free manner (see also [9])) $r = \mathrm{rem}(x, 2^n)$, $q = \mathrm{quo}(x, 2^n)$, $y = r + q \cdot (2^k - 1)$, obtaining number y of length about $n + k$ bits with $x \bmod m = y \bmod m$. This process can be continued until we get residue of x. If $k \leq cn$ for fixed constant c: $0 < c < 1$, then the number of iterations in this process is bounded by $\lceil \frac{1}{1-c} \rceil$, i.e., effectively bounded by constant and does not depend on n.

Three-term moduli

Algorithm 2 Theoretic-Reduce(a, m) — Reduce a by $m = 2^n - 2^k + 1$

Require: $0 \leq a$

Ensure: The number returned r satisfies $0 \leq r \leq 2^n - 1$

1: $aa \leftarrow a$
2: $r \leftarrow \text{rem}(aa, 2^n)$
3: $q \leftarrow \text{quo}(aa, 2^n)$
4: **while** $q \neq 0$ **do**
5: $\quad aa \leftarrow r + q \cdot (2^k - 1)$
6: $\quad r \leftarrow \text{rem}(aa, 2^n)$
7: $\quad q \leftarrow \text{quo}(aa, 2^n)$
8: **end while**
9: **if** $r \geq 2^n - 2^k + 1$ **then** $r \leftarrow r - (2^n - 2^k + 1)$
10: **end if**
11: **return** r

If k ~ 3/4 n the while loop has at most 4 iterations; If k ~ 1/2 n the while loop has at most 2 iterations;

Three-term moduli

Algorithm 2 Theoretic-Reduce(a, m) — Reduce a by $m = 2^n - 2^k + 1$

Require: $0 \leq a$

Ensure: The number returned r satisfies $0 \leq r \leq 2^n - 1$

1: $aa \leftarrow a$
2: $r \leftarrow \text{rem}(aa, 2^n)$
3: $q \leftarrow \text{quo}(aa, 2^n)$
4: **while** $q \neq 0$ **do**
5: $\quad aa \leftarrow r + q \cdot (2^k - 1)$
6: $\quad r \leftarrow \text{rem}(aa, 2^n)$
7: $\quad q \leftarrow \text{quo}(aa, 2^n)$
8: **end while**
9: **if** $r \geq 2^n - 2^k + 1$ **then** $r \leftarrow r - (2^n - 2^k + 1)$
10: **end if**
11: **return** r

2^2^17 – 2^2^10 + 1

rand[0..n]	Regular GMP (s)	Simple Reduce (s)
2^2^18	2.04586	0.0336324
2^2^19	6.96865	0.0816615
2^2^20	13.5465	0.243587
2^2^21	26.9513	0.885439
2^2^22	54.3707	3.504

If k ~ 3/4 n the while loop has at most 4 iterations; If k ~ 1/2 n the while loop has at most 2 iterations;

Reduction is fast for both – original arbitrary large input and in RNS (1st line in the table above).

Three-term moduli

Consider moduli $m_1 = 2^n - 2^\ell + 1, m_2 = 2^n - 2^k + 1, n > k > \ell$.

A very simple sufficient condition of co-primality of m_1, m_2 is:
if $n \bmod (k-\ell) = k \bmod (k-\ell)$ or $k \bmod (k-\ell) = 0$ then $\gcd(m_1, m_2) = 1$. This follows from the inspection of remainder sequence for m_1, m_2 while applying combined steps of binary and regular Euclidean algorithm: $2^n - 2^\ell + 1, 2^n - 2^k + 1, 2^\ell(2^{k-\ell} - 1), \ldots$ and the equality

$$(2^n - 2^k + 1) \bmod (2^{k-\ell} - 1) = 2^{n \bmod (k-\ell)} - 2^{k \bmod (k-\ell)} + 1.$$

Similar conditions hold for different choice of $+/-$ signs between terms of moduli.

Three-term moduli

Now, to satisfy requirement 3 one needs to search for moduli in advance using careful inspection of application of binary and regular extended Euclidean algorithm to m_1, m_2 with fixed n and variable k, ℓ. This search is to be performed only once, and produces moduli and inverses that can be re-scaled and reused for different sizes of input. The scalability follows from simple properties of remainder sequences: if $\gcd(2^n - 2^\ell + 1, 2^n - 2^k + 1) = 1$ then for any natural a also $\gcd(2^{an} - 2^{a\ell} + 1, 2^{an} - 2^{ak} + 1) = 1$ (the remainder sequence for scaled moduli will be the same as original with all exponents scaled by the factor a). Also, if for $k > \ell + 1$ the inverse $m_2^{-1} \bmod m_1$ has sparse bit pattern, then scaling moduli the by same factor a preserves the bit pattern (again, remainder sequence in binary and regular extended Euclidean algorithm remains the same with all exponents scaled). For example, $(2^{100} - 2^{60} + 1)^{-1} \bmod (2^{100} - 2^{50} + 1) = 2^{40} + 2^{30} + 2^{20} + 2^{10} + 1$ and scaling by arbitrary natural a gives $(2^{100a} - 2^{60a} + 1)^{-1} \bmod (2^{100a} - 2^{50a} + 1) = 2^{40a} + 2^{30a} + 2^{20a} + 2^{10a} + 1$.

Three-term moduli

Note, that after three-terms moduli satisfying requirements 1–4 are selected, one can add 2^n and $2^n + 1$ to the set of moduli, as these new moduli are relatively prime to previously selected, and also $(2^n - 2^k + 1)^{-1} \bmod 2^n = 2^k + 1$, $(2^n - 2^k + 1)^{-1} \bmod (2^n + 1) = 2^{n-k}$ and $(2^n + 1)^{-1} \bmod 2^n = 1$, i.e., inverses are sparse and scalable.

Many small primes

Choose many moduli that fit machine word and use hardware arithmetic for the simultaneous reduction/reconstruction

Simultaneous Conversions with the Residue Number System Using Linear Algebra

JAVAD DOLISKANI, Institute for Quantum Computing, University of Waterloo
PASCAL GIORGI and ROMAIN LEBRETON, LIRMM CNRS - University of Montpellier
ERIC SCHOST, University of Waterloo

We present an algorithm for simultaneous conversions between a given set of integers and their Residue Number System representations based on linear algebra. We provide a highly optimized implementation of the algorithm that exploits the computational features of modern processors. The main application of our algorithm is matrix multiplication over integers. Our speed-up of the conversions to and from the Residue Number System significantly improves the overall running time of matrix multiplication.

Many small primes (FFLAS-FFPACK)

Choose many moduli that fit machine word and use hardware arithmetic for the simultaneous reduction/reconstruction

- relative primality

- reduction is fast

- reconstruction is fast

- balance in size

- ~~scalability~~

Simultaneous Conversions with the Residue Number System Using Linear Algebra

JAVAD DOLISKANI, Institute for Quantum Computing, University of Waterloo
PASCAL GIORGI and ROMAIN LEBRETON, LIRMM CNRS - University of Montpellier
ERIC SCHOST, University of Waterloo

We present an algorithm for simultaneous conversions between a given set of integers and their Residue Number System representations based on linear algebra. We provide a highly optimized implementation of the algorithm that exploits the computational features of modern processors. The main application of our algorithm is matrix multiplication over integers. Our speed-up of the conversions to and from the Residue Number System significantly improves the overall running time of matrix multiplication.

Two-level RNS (application)

In [4] an algorithm for simultaneous conversions between a given set of integers and their modular representations based on linear algebra is described. Authors provide a highly optimized implementation of the algorithm that exploits the computational features of modern processors. This implementation performance on the standard benchmark of matrix multiplication starts to deteriorate when the size of entries of randomly selected integer matrices becomes very large (2^{18} or more bits).

Two-level RNS (application)

In [4] an algorithm for simultaneous conversions between a given set of integers and their modular representations based on linear algebra is described. Authors provide a highly optimized implementation of the algorithm that exploits the computational features of modern processors. This implementation performance on the standard benchmark of matrix multiplication starts to deteriorate when the size of entries of randomly selected integer matrices becomes very large (2^{18} or more bits).

To improve this two layer experimental modular approach was implemented by Yu Li and Benjamin Chen (University of Waterloo).

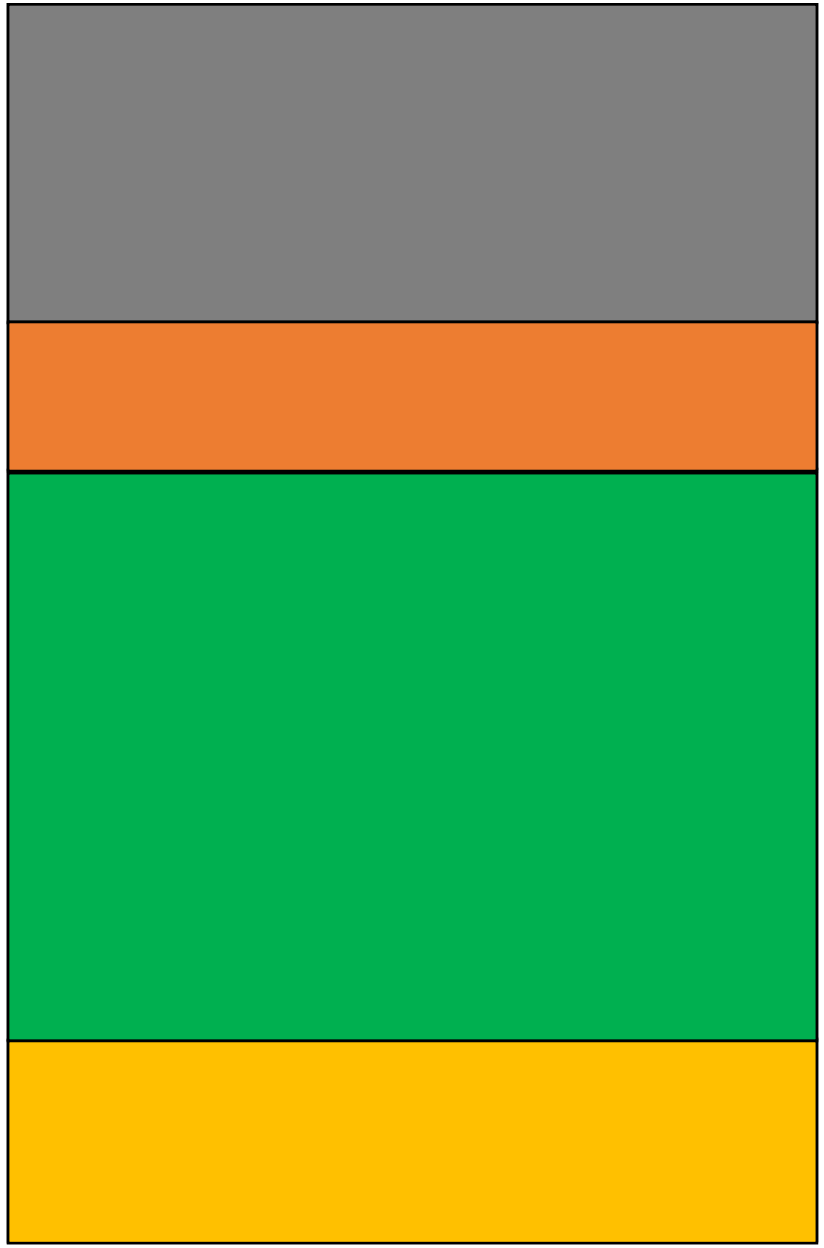
The idea is to select large moduli discussed in previous section on the first layer, and reduce the problem to several problems with entries bit-size amenable for FFLAS-FFPACK. On the second layer simultaneous conversion [4] is used. Result from multiple calls to FFLAS-FFPACK are used to reconstruct the final answer using accelerated reconstruction with specially selected moduli.

Two-level RNS (application)

Benchmark is matrix multiplication with large integer entries

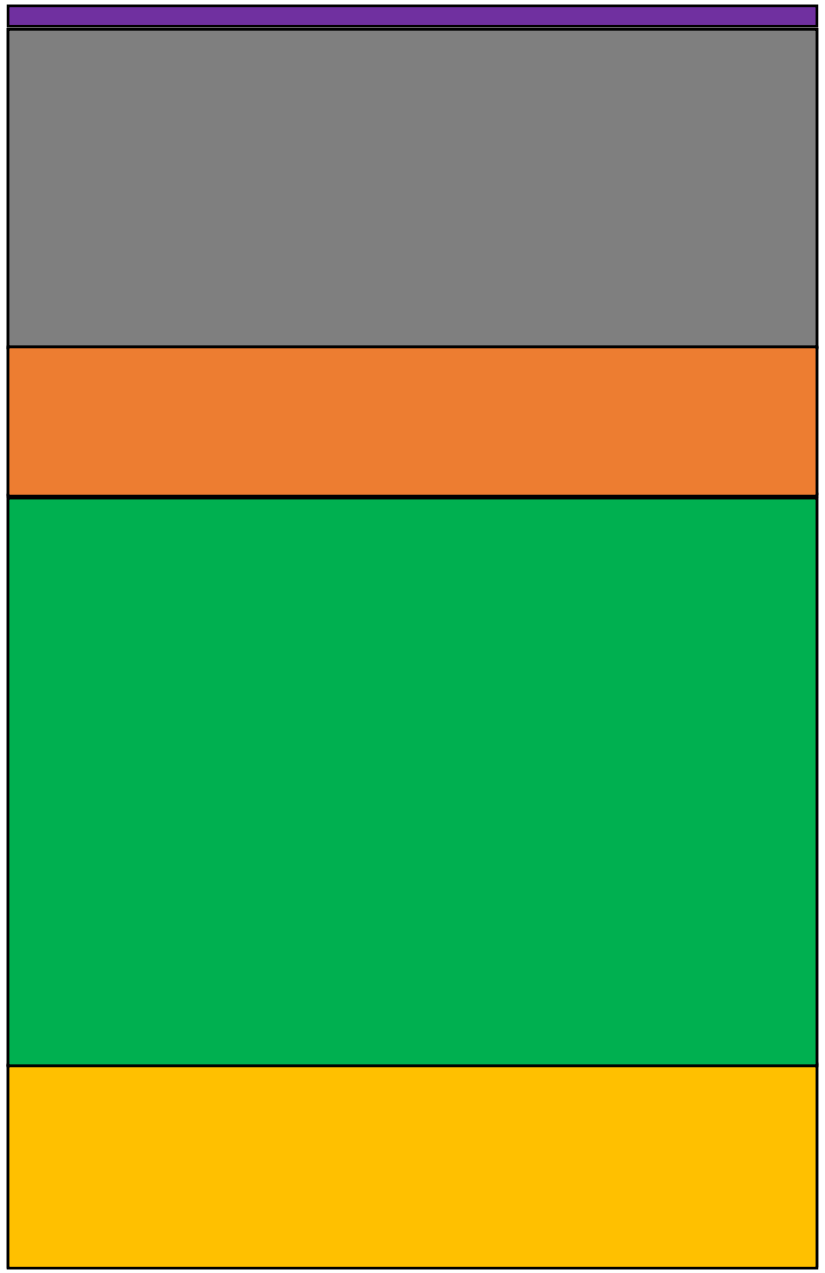
Dim	Size	FFLAS	Marge (overh)	Trinom (overh)
8	$\approx 2^{15}$	1.09	0.22 (0.05)	0.41 (0.03)
-	$2^{18} - 2^{19}$	102.69	11.73 (0.88)	11.52 (0.52)
-	$\approx 2^{19}$	406.58	43.64 (2.07)	43.20 (1.16)
-	$\approx 2^{20}$	1627.49	169.45 (5.17)	168.78 (2.57)
-	$\approx 2^{21}$	DNF	385.71 (7.95)	382.80 (4.05)
16	$\approx 2^{15}$	1.18	0.44 (0.18)	0.49 (0.12)
-	$2^{18} - 2^{19}$	105.31	16.96 (3.37)	15.84 (1.92)
-	$\approx 2^{19}$	416.94	59.42 (8.16)	56.20 (4.44)
-	$\approx 2^{20}$	1656.10	222.08 (20.54)	212.76 (10.09)
-	$\approx 2^{21}$	DNF	482.94 (31.7)	480.15 (16.07)
32	$\approx 2^{15}$	1.61	1.33 (0.62)	1.11 (0.39)
-	$2^{18} - 2^{19}$	124.33	42.63 (13.43)	38.29 (7.57)
-	$\approx 2^{19}$	479.36	128.12 (32.56)	114.51 (17.65)
-	$\approx 2^{20}$	2003.69	450.47 (82.15)	408.52 (40.35)
-	$\approx 2^{21}$	DNF	946.7 (126.96)	900.89 (63.93)

Table 1: Timing (in seconds) of square integer matrix multiplication benchmark:
Dim - matrix dimension, Size - bitsize of entries, FFLAS - direct use of FFLAS-FFPACK implementation, Marge - first layer with 7 moduli of the form $2^n - 1$, Trinom - first layer with 7 moduli of the form $2^n - 2^k + 1$, (overh) represents the time spent for conversion to and from RNS in the first layer (note, that in column Marge overhead includes time to compute inverses, while in column Trinom inverses are not computed and obtained from scaling). Hardware used is AMD EPYC 7502P 32C @ 2.5 GHz with 503GB of RAM. Entries DNF mean "did not finish in 10 hours".



FFLAS-FFPACK timing

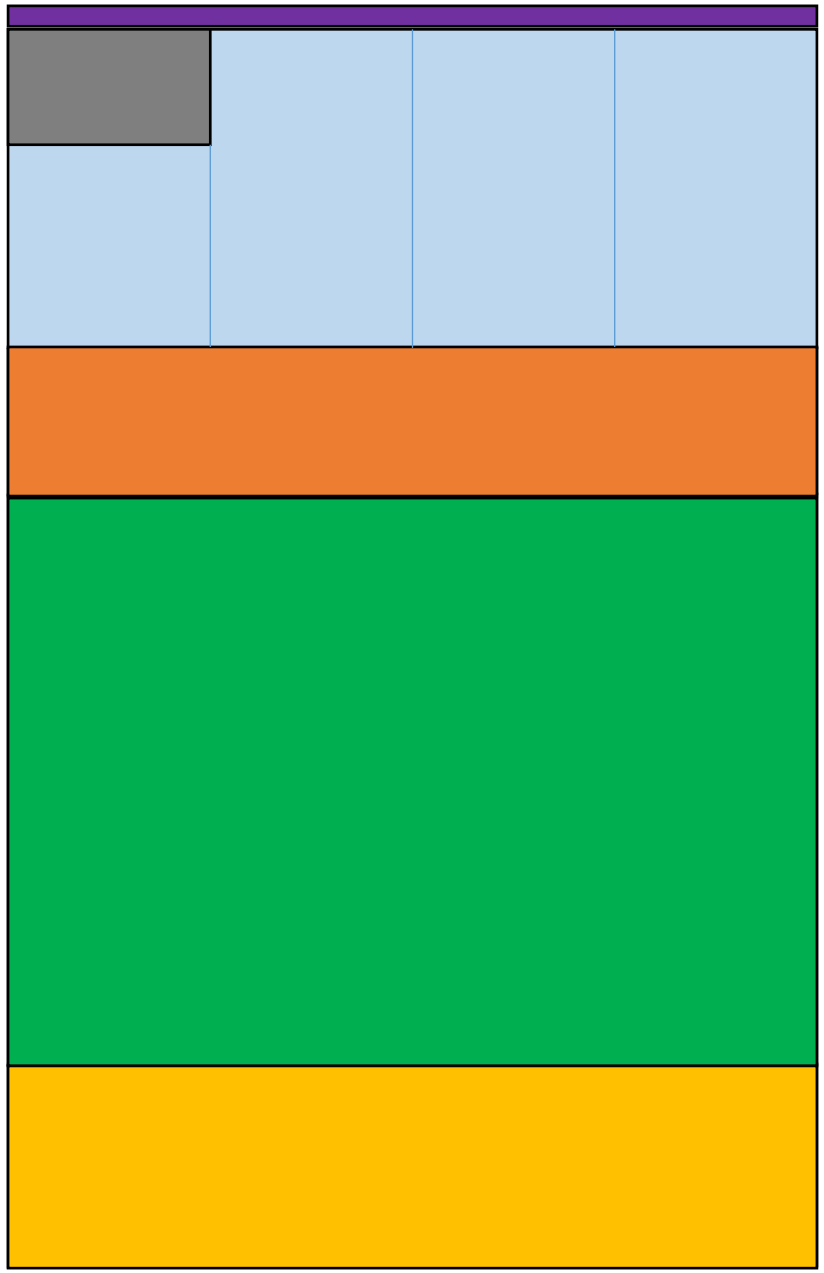
- Preprocessing
- Simultaneous conversion to RNS
- Multiplication in all RNS
- Reconstruction of result from modular images



FFLAS-FFPACK timing

Preprocessing
Simultaneous conversion to RNS
Multiplication in all RNS
Reconstruction of result from modular images

Upper layer conversion to RNS

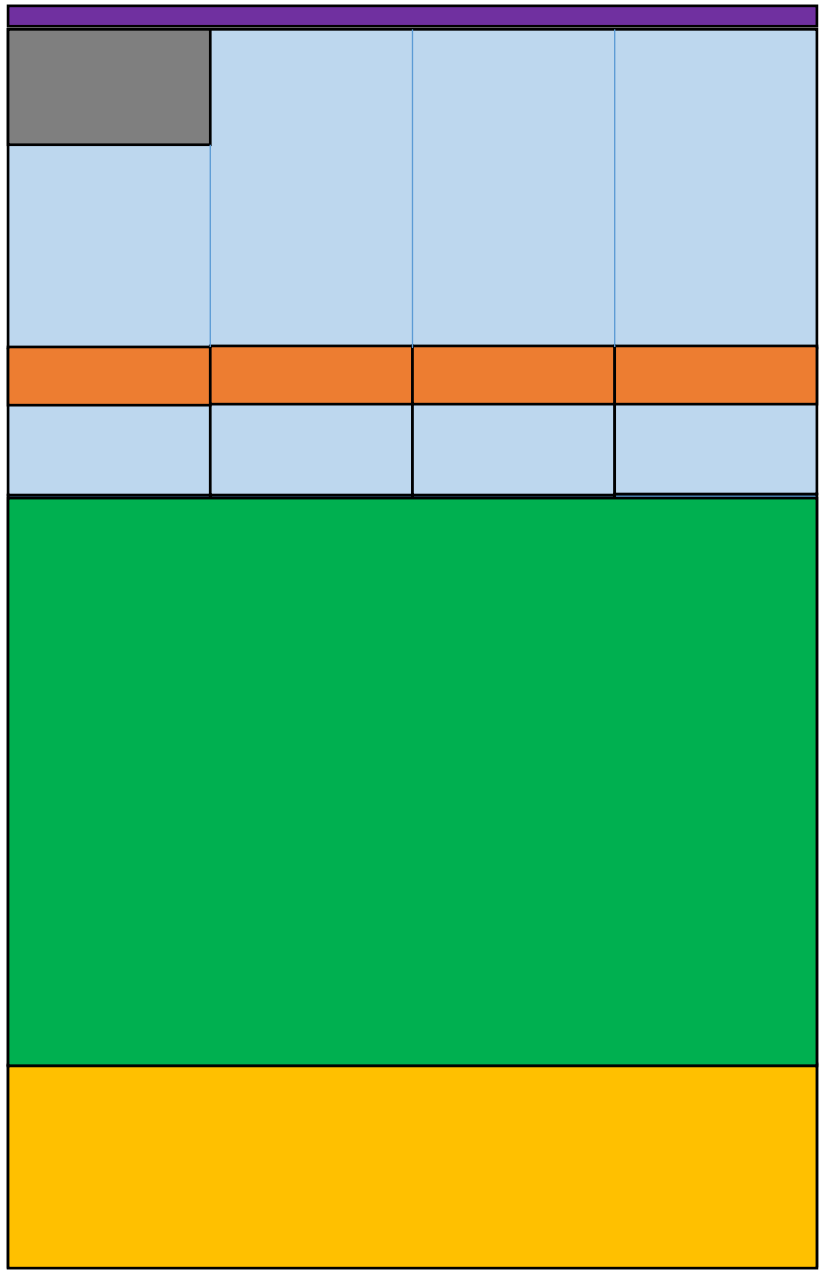


FFLAS-FFPACK timing

Preprocessing
Simultaneous conversion to RNS
Multiplication in all RNS
Reconstruction of result from modular images

Saved time

Upper layer conversion to RNS

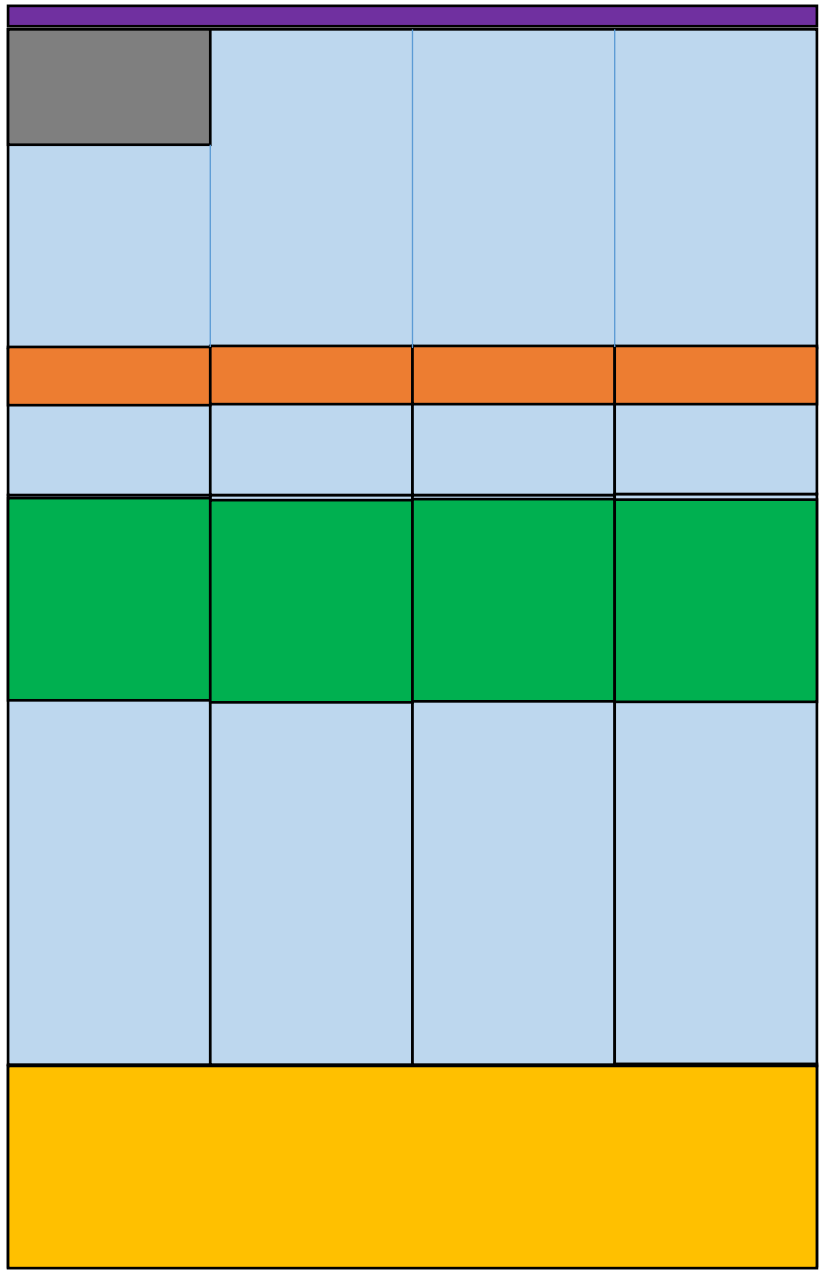


FFLAS-FFPACK timing

Preprocessing
Simultaneous conversion to RNS
Multiplication in all RNS
Reconstruction of result from modular images

Saved time

Upper layer conversion to RNS

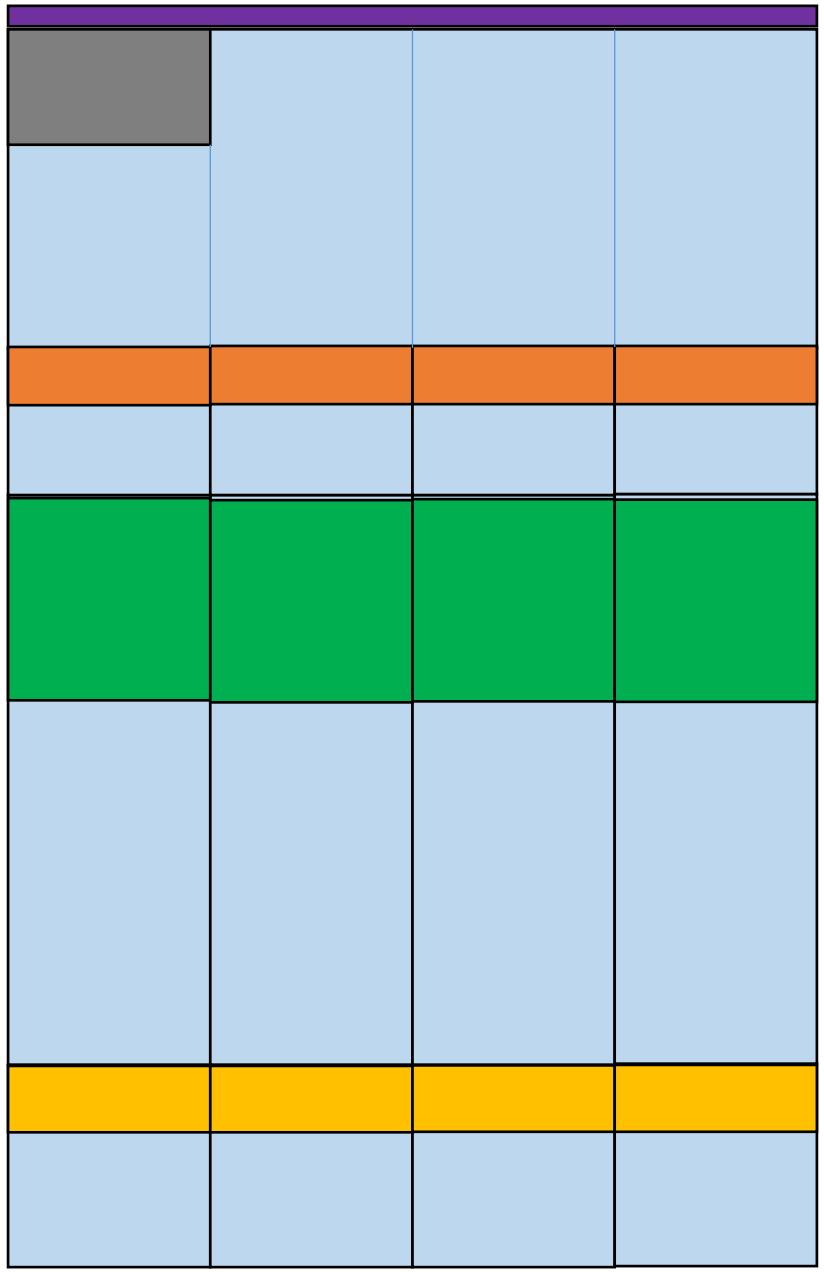


FFLAS-FFPACK timing

Preprocessing
Simultaneous conversion to RNS
Multiplication in all RNS
Reconstruction of result from modular images

Saved time

Upper layer conversion to RNS

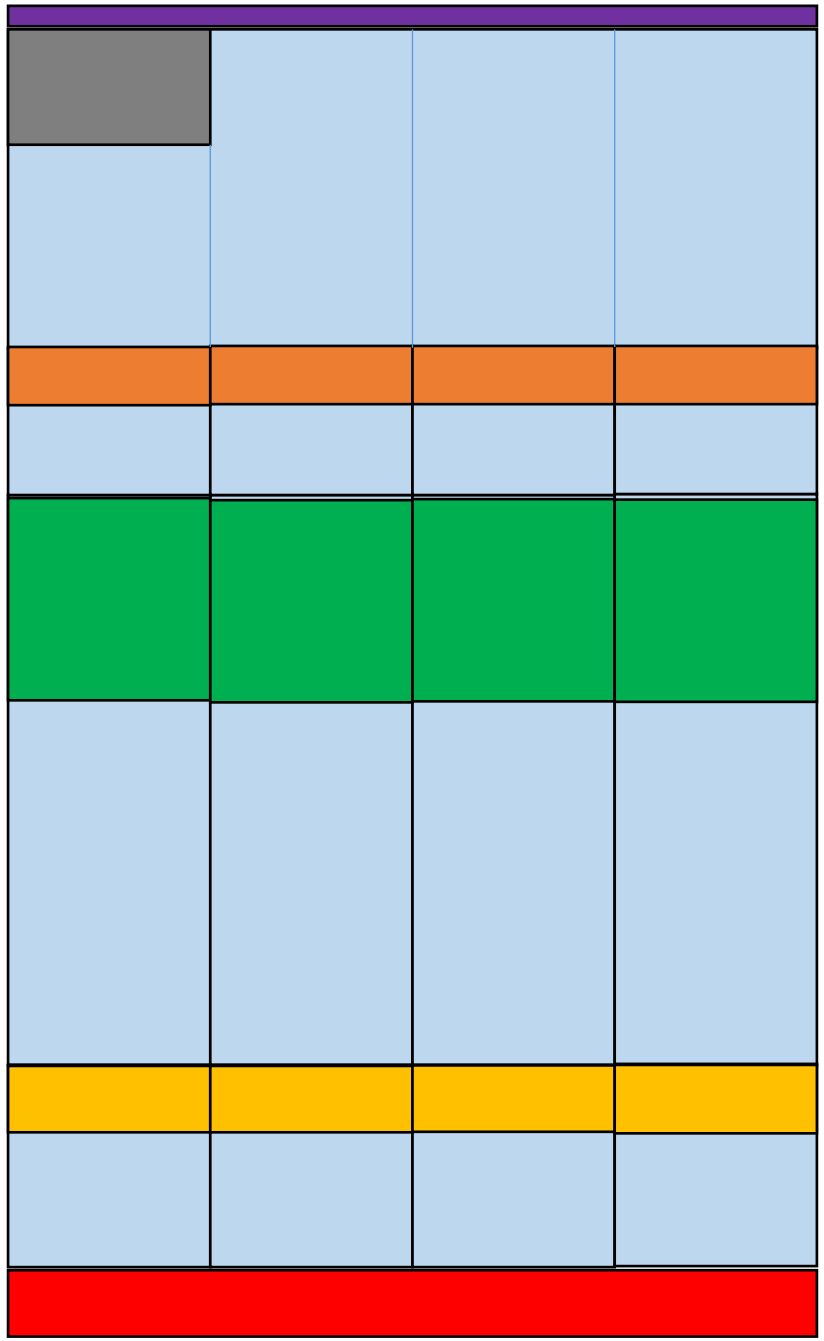


FFLAS-FFPACK timing

Preprocessing
Simultaneous conversion to RNS
Multiplication in all RNS
Reconstruction of result from modular images

Saved time

Upper layer conversion to RNS



FFLAS-FFPACK timing

- Preprocessing
- Simultaneous conversion to RNS
- Multiplication in all RNS
- Reconstruction of result from modular images

- Saved time

- Upper layer conversion to RNS
- Upper layer reconstruction

Two-level RNS (application)

Benchmark is matrix multiplication with large integer entries

Dim	Size	FFLAS	Marge (overh)	Trinom (overh)
8	$\approx 2^{15}$	1.09	0.22 (0.05)	0.41 (0.03)
-	$2^{18} - 2^{19}$	102.69	11.73 (0.88)	11.52 (0.52)
-	$\approx 2^{19}$	406.58	43.64 (2.07)	43.20 (1.16)
-	$\approx 2^{20}$	1627.49	169.45 (5.17)	168.78 (2.57)
-	$\approx 2^{21}$	DNF	385.71 (7.95)	382.80 (4.05)
16	$\approx 2^{15}$	1.18	0.44 (0.18)	0.49 (0.12)
-	$2^{18} - 2^{19}$	105.31	16.96 (3.37)	15.84 (1.92)
-	$\approx 2^{19}$	416.94	59.42 (8.16)	56.20 (4.44)
-	$\approx 2^{20}$	1656.10	222.08 (20.54)	212.76 (10.09)
-	$\approx 2^{21}$	DNF	482.94 (31.7)	480.15 (16.07)
32	$\approx 2^{15}$	1.61	1.33 (0.62)	1.11 (0.39)
-	$2^{18} - 2^{19}$	124.33	42.63 (13.43)	38.29 (7.57)
-	$\approx 2^{19}$	479.36	128.12 (32.56)	114.51 (17.65)
-	$\approx 2^{20}$	2003.69	450.47 (82.15)	408.52 (40.35)
-	$\approx 2^{21}$	DNF	946.7 (126.96)	900.89 (63.93)

Table 1: Timing (in seconds) of square integer matrix multiplication benchmark:
Dim - matrix dimension, Size - bitsize of entries, FFLAS - direct use of FFLAS-FFPACK implementation, Marge - first layer with 7 moduli of the form $2^n - 1$, Trinom - first layer with 7 moduli of the form $2^n - 2^k + 1$, (overh) represents the time spent for conversion to and from RNS in the first layer (note, that in column Marge overhead includes time to compute inverses, while in column Trinom inverses are not computed and obtained from scaling). Hardware used is AMD EPYC 7502P 32C @ 2.5 GHz with 503GB of RAM. Entries DNF mean "did not finish in 10 hours".

Two-level RNS (application)

1st level arithmetic comparison

Size	R.Red	M.Red	T.Red	R.Rec	M.Rec	T.Rec
2^{18}	3.90	0.10	0.20	3.41 + 6.68	2.39 + 2.59	2.48
2^{19}	9.96	0.20	0.37	8.51 + 16.82	5.94 + 6.42	6.26
2^{20}	24.70	0.40	0.64	20.22 + 39.17	14.38 + 15.71	14.86
2^{21}	58.45	0.79	1.24	51.35 + 88.67	38.45 + 36.57	34.43
2^{22}	128.40	1.59	2.43	111.44 + 195.43	84.07 + 78.96	77.14

Table 2: Timing (in seconds) for the input 32 by 32 matrix conversion to RNS and immediate reconstruction from RNS. Column Size represents bitsize of matrix entries, R.Red – time for the regular (division-based) reduction, M.Red - reduction by moduli of the form 2^n-1, T.Red - reduction by moduli of the form $2^n - 2^k + 1$. Column R.Rec represents time of the regular reconstruction [4] (time to compute inverses + reconstruction time), M.Rec – reconstruction time for moduli of the form $2^n - 1$ (time to compute inverses + reconstruction time), T.Rec – reconstruction time for moduli of the form $2^n - 2^k + 1$. Hardware used is AMD EPYC 7502P 32C @ 2.5 GHz with 503GB of RAM.

Conclusion

Careful selection of moduli with fixed bit-pattern provides practical improvement to the standard modular algorithms. This selection (satisfying requirements 1–4) uses search with back-tracking and is based on inspection of remainder sequences in combined binary and regular extended Euclidean algorithm. There is a similarity between few-terms moduli discussed here and polynomials with few terms (such as trinomials or pentanomials) over the integers. For example, given natural $n > k > \ell$, if $n \bmod (k - \ell) = k \bmod (k - \ell)$ or $k \bmod (k - \ell) = 0$ then polynomials $x^n - x^\ell + 1$ and $x^n - x^k + 1$ are relatively prime. It is anticipated that the inspection of the structure of polynomial remainder sequences for fewnomials with unit coefficients over integers can help in the search of balanced moduli with three, five, or generally "few" terms, satisfying requirements 1–4. Note, that fewnomials over finite fields were studied extensively (see, for example [1]). However, it seems that the structure of polynomial remainder sequences of fewnomials over the ring of integers deserves additional study.

References

[1] Banegas G., Custódio R., and Panario D. A new class of irreducible pentanomials for polynomial-based multipliers in binary fields. *J Cryptogr. Eng.*, vol. 9, 2019, pp. 359–373.

Conclusion

It is expected that choosing moduli with 4, 5 or even more terms in bit pattern satisfying requirements 1–4 might provide further improvement. There is a trade-off: adding more terms slows down conversion to RNS and RNS arithmetic slightly, but gives more flexibility in finding relatively prime moduli with scalable sparse inverses, which accelerates reconstruction phase. This trade-off deserves further investigation.