Simultaneous Conversions with the Residue Number System using Linear Algebra

Javad Doliskani, Éric Schost
University of Waterloo

Pascal Giorgi, Romain Lebreton
Université de Montpellier

Journées Nationales du Calcul Formel – Luminy
January 25th 2018
How to perform arithmetic operations on ... ?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Multi-precision integer matrices $\mathcal{M}<em>{n \times n}(\mathbb{Z})$ or $\mathcal{M}</em>{n \times n}(\mathbb{Z}/N\mathbb{Z})$</td>
</tr>
<tr>
<td>2.</td>
<td>Multi-precision integer polynomials $\mathbb{Z}[X]$ or $\mathbb{Z}/N\mathbb{Z}$</td>
</tr>
</tbody>
</table>

Different approaches:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>Direct algorithm:</strong></td>
<td>matrix arithmetic then multi-precision integer arithmetic</td>
</tr>
<tr>
<td></td>
<td>$\leadsto$ Best for matrix/polynomial size $\ll$ integer bitsize</td>
</tr>
<tr>
<td>2. <strong>Modular approach:</strong></td>
<td>Split the big integer matrix into many small integer matrices</td>
</tr>
<tr>
<td></td>
<td>$\leadsto$ Best for matrix/polynomial size $\gg$ integer bitsize</td>
</tr>
</tbody>
</table>

Change of representation of integers (RNS conversion) is frequently a bottleneck.
### How to perform arithmetic operations on ... ?

1. **Multi-precision integer matrices**
   \[ \mathcal{M}_{n \times n} (\mathbb{Z}) \text{ or } \mathcal{M}_{n \times n} (\mathbb{Z}/N\mathbb{Z}) \]

2. **Multi-precision integer polynomials**
   \[ \mathbb{Z}[X] \text{ or } \mathbb{Z}/N\mathbb{Z} \]

### Different approaches:

1. **Direct algorithm:**
   Matrix arithmetic then multi-precision integer arithmetic
   \[ \rightsquigarrow \text{Best for matrix/polynomial size } \ll \text{ integer bitsize} \]

2. **Modular approach:**
   Split the big integer matrix into many small integer matrices
   \[ \rightsquigarrow \text{Best for matrix/polynomial size } \gg \text{ integer bitsize} \]

Change of representation of integers (RNS conversion) is frequently a bottleneck.
1. Context

2. Preliminaries on the Residue Number System
   a. Euclidean division
   b. Conversion with the Residue Number System

3. Conversion with RNS using linear algebra
   a. Algorithm
   b. Implementation details
   c. Timings and comparison
   d. Extension for larger moduli
**Problem:** If \( a, m \in \mathbb{N} \), compute \( q = a \text{ quo } m \) and \( r = a \text{ rem } m \).

**Note:** \((n := \text{ bitsize}(a)) \geq (t := \text{ bitsize}(m))\)

### Algorithms for Euclidean division:

1. \( n = 2t \) in time \( \mathcal{O}(I(t)) \) — **Balanced case**  
   **Idea:** Newton iteration to approximate \( 1/m \) then \( q = \lfloor a/m \rfloor \), finally \( r = a - qm \)  
   \[\text{[Cook '66], [Barrett '86]}\]

2. \( n \geq 2t \) in time \( \mathcal{O}\left(n \frac{I(t)}{t}\right)\),  
   **Idea:** Iterate base case reduction

3. \( n \leq 2t \) in time \( \mathcal{O}\left(t \frac{I(n-t)}{n-t}\right)\)  
   **Idea:** Compute \( q \) only at precision \( n - t \)  
   \[\text{[Giorgi et al, '13]}\]
**Problem:** If $a, m \in \mathbb{N}$, compute $q = a \div m$ and $r = a \mod m$.

**Note:** $(n := \text{bitsize}(a)) \geq (t := \text{bitsize}(m))$

<table>
<thead>
<tr>
<th>Algorithms for Euclidean division:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $n = 2t$ in time $\mathcal{O}(l(t))$ – <strong>Balanced case</strong> [Cook ’66], [Barrett ’86]</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>2. $n \geq 2t$ in time $\mathcal{O}\left( n \frac{l(t)}{t} \right)$,</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>3. $n \leq 2t$ in time $\mathcal{O}\left( t \frac{l(n-t)}{n-t} \right)$ [Giorghi et al, ’13]</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Euclidean division

**Problem:** If \( a, m \in \mathbb{N} \), compute \( q = a \text{ quo} m \) and \( r = a \text{ rem} m \).

**Note:** \( (n := \text{bitsize}(a)) \geq (t := \text{bitsize}(m)) \)

---

### Algorithms for Euclidean division:

1. \( n = 2t \) in time \( \mathcal{O}(l(t)) \) – Balanced case
   
   **Idea:** Newton iteration to approximate \( 1/m \) then \( q = \lfloor a/m \rfloor \), finally \( r = a - qm \)

2. \( n \geq 2t \) in time \( \mathcal{O}\left( n \frac{l(t)}{t} \right) \),

   **Idea:** Iterate base case reduction

3. \( n \leq 2t \) in time \( \mathcal{O}\left( t \frac{l(n-t)}{n-t} \right) \)

   **Idea:** Compute \( q \) only at precision \( n - t \)

---

[Cook ’66], [Barrett ’86], [Giorgi et al., ’13]
The Residue Number System

Residue Number System:

Represent integer $0 \leq a \leq m_1 \cdots m_s$ by its residues $(a \mod m_1, \ldots, a \mod m_s)$

Chinese remainder theorem:

$$\mathbb{Z} / (m_1 \cdots m_s)\mathbb{Z} \xrightarrow{\text{Reduction}} \prod_{i=1}^{s} \mathbb{Z} / m_i \mathbb{Z}$$

$$a \mod (m_1 \cdots m_s) \quad \mapsto \quad (a \mod m_1, \ldots, a \mod m_s)$$

From now on, assume that $m_i$ are bounded (e.g. 64 bits machine-word) and pairwise coprime.

Let’s recall the algorithms for reduction and reconstruction
**Problem:** Compute \((a \mod m_i)_{i=1}^s\) where \(a < M\) and \(M = m_1 \cdots m_s\).

**Naive Algorithm**

Apply \(a \mod m_i\) in the case \((t = 1) \ll (n = s)\) \(\leadsto\) Quadratic time \(O(s^2)\)

**Note:** Small constant factor in the big \(O\), faster for small \(s\)

**Divide-and-conquer algorithm**

Sub-product tree followed by Divide-and-conquer reductions

Quasi-linear complexity: \(O(l(s) \log s)\)

**Note:** Similar to polynomial case of multi-point evaluation
**Problem:** Given the evaluations \( a_i = a(m_i) = a(X) \rem (X - m_i) \), reconstruct \( a(X) \) using

\[
a(X) = \sum_i a_i \prod_{j \neq i} \frac{(X - m_j)}{(m_i - m_j)} = \sum_i \frac{a_i}{M_i(m_i)} M_i(X) \quad \text{where} \quad M_i(X) = M(X) / (X - m_i)
\]

---

**Divide-and-conquer algorithm**

1. Compute \( M_i(m_i) \)'s using \( M_i(m_i) = M'(m_i) \) so multi-point evaluation of \( M'(X) \)

2. Let \( b_i = \frac{a_i}{M_i(m_i)} \) and recursively compute \( a(X) = \sum_i b_i M_i(X) \):

---

**Idea:** If \( M^0(X) = \prod_{j=1}^{s/2} (X - m_j) \) and \( M^1(X) = \prod_{j=s/2+1}^{s} (X - m_j) \), note that

\[
a(X) = \sum_{i=1}^{s} b_i \frac{M}{(X - m_i)} = \left[ \sum_{i=1}^{s/2} b_i \frac{M^0}{(X - m_i)} \right] M^1 + \left[ \sum_{i=s/2+1}^{s} b_i \frac{M^1}{(X - m_i)} \right] M^0
\]

---

**Complexity:** \( \mathcal{O}(M(s) \log s) \)
**Problem:** Given the reductions \( a_i = a \mod m_i \), reconstruct \( a \).

**Integer equivalent of Lagrange interpolation**

\[
\mathbb{Z}/M\mathbb{Z} \longrightarrow (\mathbb{Z}/m_1\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_s\mathbb{Z})
\]

- \( M_i \mapsto (0, \ldots, M_i \mod m_i, \ldots, 0) \)
- \( u_i M_i \mapsto (0, \ldots, 1, \ldots, 0) \)
- \( u_i a_i M_i \mapsto (0, \ldots, a_i, \ldots, 0) \)
- \( \sum_{i=1}^{s} u_i a_i M_i \mapsto (a_1, \ldots, a_i, \ldots, a_s) \)

So \( a \) given by

\[
a := \left( \sum_{i=1}^{s} u_i a_i M_i \right) \mod M = \left( \sum_{i=1}^{s} ((u_i a_i) \mod m_i) M_i \right) \mod M
\]

is the reconstruction of \((a_1, \ldots, a_s)\).

**Remark:** \(((u_i a_i) \mod m_i) M_i\) corresponds to \(\frac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}\) in Lagrange formula.
**Problem**: Given the reductions \( a_i = a \mod m_i \), reconstruct \( a \).

**Integer equivalent of Lagrange interpolation**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbb{Z} / M \mathbb{Z} )</td>
<td>( \mathbb{Z} / m_1 \mathbb{Z} \times \cdots \times (\mathbb{Z} / m_i \mathbb{Z}) \times \cdots \times (\mathbb{Z} / m_s \mathbb{Z}) )</td>
</tr>
<tr>
<td>( M_i )</td>
<td>( (0, \ldots, M_i \mod m_i, \ldots, 0) )</td>
</tr>
<tr>
<td>( u_i M_i )</td>
<td>( (0, \ldots, 1, \ldots, 0) )</td>
</tr>
<tr>
<td>( u_i a_i M_i )</td>
<td>( (0, \ldots, a_i, \ldots, 0) )</td>
</tr>
<tr>
<td>( \sum_{i=1}^{s} u_i a_i M_i )</td>
<td>( (a_1, \ldots, a_i, \ldots, a_s) )</td>
</tr>
</tbody>
</table>

So \( a \) given by

\[
a := \left( \sum_{i=1}^{s} u_i a_i M_i \right) \mod M = \left( \sum_{i=1}^{s} \left( (u_i a_i) \mod m_i \right) M_i \right) \mod M
\]

is the reconstruction of \((a_1, \ldots, a_s)\).

**Remark**: \( (u_i a_i) \mod m_i \) \( M_i \) corresponds to \( a_i \prod_{j \neq i} \frac{X - m_j}{m_i - m_j} \) in Lagrange formula.
**Problem:** Given the reductions \( a_i = a \text{ rem } m_i \), reconstruct \( a \).

### Integer equivalent of Lagrange interpolation

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbb{Z} / M \mathbb{Z} )</td>
<td>( \times \cdots \times \mathbb{Z} / m_i \mathbb{Z} \times \cdots \times \mathbb{Z} / m_s \mathbb{Z} )</td>
</tr>
<tr>
<td>( ? )</td>
<td>( (a_1, \ldots, a_i, \ldots, a_s) )</td>
</tr>
<tr>
<td>( M_i )</td>
<td>( (0, \ldots, M_i \text{ rem } m_i, \ldots, 0) )</td>
</tr>
<tr>
<td>( u_i M_i )</td>
<td>( (0, \ldots, 1, \ldots, 0) )</td>
</tr>
<tr>
<td>( u_i a_i M_i )</td>
<td>( (0, \ldots, a_i, \ldots, 0) )</td>
</tr>
<tr>
<td>( \sum_{i=1}^s u_i a_i M_i )</td>
<td>( (a_1, \ldots, a_i, \ldots, a_s) )</td>
</tr>
</tbody>
</table>

So \( a \) given by

\[
a := \left( \sum_{i=1}^s u_i a_i M_i \right) \text{ rem } M = \left( \sum_{i=1}^s \left( (u_i a_i) \text{ rem } m_i \right) M_i \right) \text{ rem } M
\]

is the reconstruction of \( (a_1, \ldots, a_s) \).

**Remark:** \( ((u_i a_i) \text{ rem } m_i) \) \( M_i \) corresponds to \( \frac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)} \) in Lagrange formula.
**Problem:** Given the reductions $a_i = a \mod m_i$, reconstruct $a$.

<table>
<thead>
<tr>
<th>Integer equivalent of Lagrange interpolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{Z} / M \mathbb{Z} \longrightarrow (\mathbb{Z} / m_1 \mathbb{Z}) \times \cdots \times (\mathbb{Z} / m_i \mathbb{Z}) \times \cdots \times (\mathbb{Z} / m_s \mathbb{Z})$</td>
</tr>
</tbody>
</table>

| $? \longrightarrow (a_1, \ldots, a_i, \ldots, a_s)$ |
| $M_i \longrightarrow (0, \ldots, M_i \mod m_i, \ldots, 0)$ |
| $u_i M_i \longrightarrow (0, \ldots, 1, \ldots, 0)$ |
| $u_i a_i M_i \longrightarrow (0, \ldots, a_i, \ldots, 0)$ |
|
| $\sum_{i=1}^{s} u_i a_i M_i \longrightarrow (a_1, \ldots, a_i, \ldots, a_s)$ |

So $a$ given by

$$a := \left( \sum_{i=1}^{s} u_i a_i M_i \right) \mod M = \left( \sum_{i=1}^{s} ((u_i a_i) \mod m_i) M_i \right) \mod M$$

is the reconstruction of $(a_1, \ldots, a_s)$.

**Remark:** $((u_i a_i) \mod m_i) M_i$ corresponds to $\frac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}$ in Lagrange formula.
Problem: Given the reductions \( a_i = a \) \( \text{rem} \) \( m_i \), reconstruct \( a \).

<table>
<thead>
<tr>
<th>Integer equivalent of Lagrange interpolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbb{Z} / M \mathbb{Z} \rightarrow (\mathbb{Z} / m_1 \mathbb{Z}) \times \cdots \times (\mathbb{Z} / m_i \mathbb{Z}) \times \cdots \times (\mathbb{Z} / m_s \mathbb{Z}) )</td>
</tr>
<tr>
<td>( ? ) ( \rightarrow ) (( a_1, \ldots, a_i, \ldots, a_s ))</td>
</tr>
<tr>
<td>( M_i ) ( \rightarrow ) (0, \ldots, ( M_i ) ( \text{rem} ) ( m_i ), \ldots, 0)</td>
</tr>
<tr>
<td>( u_i ) ( M_i ) ( \rightarrow ) (0, \ldots, 1, \ldots, 0)</td>
</tr>
<tr>
<td>( u_i ) ( a_i ) ( M_i ) ( \rightarrow ) (0, \ldots, ( a_i ), \ldots, 0)</td>
</tr>
<tr>
<td>( \sum_{i=1}^{s} u_i ) ( a_i ) ( M_i ) ( \rightarrow ) (( a_1, \ldots, a_i, \ldots, a_s ))</td>
</tr>
</tbody>
</table>

So \( a \) given by

\[
a := \left( \sum_{i=1}^{s} u_i \ a_i \ M_i \right) \ \text{rem} \ M = \left( \sum_{i=1}^{s} \ ((u_i \ a_i) \ \text{rem} \ m_i) \ M_i \right) \ \text{rem} \ M
\]

is the reconstruction of \( (a_1, \ldots, a_s) \).

Remark: \( ((u_i \ a_i) \ \text{rem} \ m_i) \ M_i \) corresponds to \( \frac{a_i \ \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)} \) in Lagrange formula.
Problem: Given the reductions $a_i = a \mod m_i$, reconstruct $a$.

**Integer equivalent of Lagrange interpolation**

\[
\mathbb{Z}/M\mathbb{Z} \rightarrow (\mathbb{Z}/m_1\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_i\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_s\mathbb{Z})
\]

- $? \mapsto (a_1, \ldots, a_i, \ldots, a_s)$
- $M_i \mapsto (0, \ldots, M_i \mod m_i, \ldots, 0)$
- $u_i M_i \mapsto (0, \ldots, 1, \ldots, 0)$
- $u_i a_i M_i \mapsto (0, \ldots, a_i, \ldots, 0)$
- $\sum_{i=1}^{s} u_i a_i M_i \mapsto (a_1, \ldots, a_i, \ldots, a_s)$

So $a$ given by

\[
a := \left( \sum_{i=1}^{s} u_i a_i M_i \right) \mod M = \left( \sum_{i=1}^{s} (u_i a_i \mod m_i) M_i \right) \mod M
\]

is the reconstruction of $(a_1, \ldots, a_s)$.

**Remark:** $(u_i a_i \mod m_i) M_i$ corresponds to $\frac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}$ in Lagrange formula.
### Divide-and-conquer algorithm

1. Compute $M_i \rem m_i$

2. Compute $u_i = 1 / M_i \mod m_i$

3. Compute $b_i := (u_i a_i) \rem m_i$

4. Compute $P = \sum_i b_i M_i$

Total complexity: $\mathcal{O}(I(s) \log s)$

### Trick for $M_i \rem m_i$

- Multireduce $M$ modulo $m_i^2$ instead of $M / m_i \mod m_i$
### RNS reconstruction – Algorithm

#### Divide-and-conquer algorithm

1. Compute $M_i \text{ rem } m_i$
2. Compute $u_i = 1 / M_i \mod m_i$
3. Compute $b_i := (u_i a_i) \text{ rem } m_i$
4. Compute $P = \sum_i b_i M_i$

$\sim  M_i(m_i) = M'(m_i)$? No derivative $M'$!

$\sim$ Same recursion $P = P^0 M^1 + P^1 M^0$

#### Trick for $M_i \text{ rem } m_i$

Multireduce $M$ modulo $m_i^2$ instead of $M / m_i \mod m_i$

**Total complexity:** $\mathcal{O}(l(s)\log s)$
## Divide-and-conquer algorithm

1. Compute $M_i \text{ rem } m_i$  
   $\leadsto M_i(m_i) = M'(m_i)$? No derivative $M'$!

2. Compute $u_i = 1 / M_i \mod m_i$

3. Compute $b_i := (u_i a_i) \text{ rem } m_i$

4. Compute $P = \sum_i b_i M_i$  
   $\leadsto$ Same recursion $P = P^0 M^1 + P^1 M^0$

### Trick for $M_i \text{ rem } m_i$

Multireduce $M$ modulo $m_i^2$ instead of $M / m_i$ modulo $m_i$

### Total complexity: $O(I(s) \log s)$
RNS reconstruction – Algorithm

<table>
<thead>
<tr>
<th>Divide-and-conquer algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Compute $M_i \text{ rem } m_i$</td>
</tr>
<tr>
<td>2. Compute $u_i = 1 / M_i \text{ mod } m_i$</td>
</tr>
<tr>
<td>3. Compute $b_i := (u_i a_i) \text{ rem } m_i$</td>
</tr>
<tr>
<td>4. Compute $P = \sum_i b_i M_i$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trick for $M_i \text{ rem } m_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multireduce $M$ modulo $m_i^2$ instead of $M / m_i$ modulo $m_i$</td>
</tr>
</tbody>
</table>

**Total complexity:** $\mathcal{O}(l(s)\log s)$
**Fact:** We don’t know how to improve one RNS conversion!

End of the talk? No!
**Fact:** We don’t know how to improve one RNS conversion!

End of the talk? No!

**Motivation:** Integer matrix multiplication needs many simultaneous RNS conversions

**Problem:** Given $a_1, \ldots, a_r < (M = m_1 \cdots m_s)$, compute $(a_i \text{ rem } m_j)_{i=1\ldots r, j=1\ldots s}$

---

**State-of-the-art complexities for simultaneous RNS conversions**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Naive algorithms:</td>
<td>$O(rs^2)$</td>
</tr>
<tr>
<td>2. DAC algorithms:</td>
<td>$O(r \log s)$</td>
</tr>
</tbody>
</table>

---

**Our contribution**

[DOLISKANI, GIORGI, LEBRETON, SCHOST ’17]

Simultaneous conversions from/to RNS in time $O(r s^\omega - 1)$

(using precomputation of time $O(s^2)$)
1. Context

2. Preliminaries on the Residue Number System
   a. Euclidean division
   b. Conversion with the Residue Number System

3. Conversion with RNS using linear algebra
   a. Algorithm
   b. Implementation details
   c. Timings and comparison
   d. Extension for larger moduli
Problem: Given \( a = (a_1, ..., a_r) \in \mathbb{Z}^r \), compute \((a_i \text{ rem } m_j)_{i=1...r, j=1...s}\)

Note: \([_]_\ell = \_ \mod m_\ell \) and \( t \) such that \( m_i \leq 2^t \)

Idea:

1. Decompose \( a_i \) in base \( 2^t \): \( a_i = \sum_{j=0}^{s-1} a_{i,j} 2^{jt} \)

2. If \( d_{i,\ell} := (\sum_{j=0}^{s-1} a_{i,j} [2^{jt}]_\ell) \) then

\( d_{i,\ell} \) is a pseudo-reduction of \( a_i \) modulo \( m_\ell \), i.e. \( a_i = d_{i,\ell} \mod m_\ell \) and \( d_{i,\ell} \leq 2^t \)
### Algorithm

1. **Precompute** \([2^{j_t}]_i \leq i, j \leq s\) \(O(s^2)\)

2. **Matrix multiplication with small integer entries** \(O(\text{MM}(s, s, r))\)

\[
\begin{bmatrix}
    d_{1,1} & \cdots & d_{r,1} \\
    \vdots & \ddots & \vdots \\
    d_{1,s} & \cdots & d_{r,s}
\end{bmatrix}
= \begin{bmatrix}
    1 & [2^t]_1 & [2^{2t}]_1 & \cdots & [2^{(s-1)t}]_1 \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    1 & [2^t]_s & [2^{2t}]_s & \cdots & [2^{(s-1)t}]_s
\end{bmatrix}
\times
\begin{bmatrix}
a_{1,0} & \cdots & a_{r,0} \\
\vdots & \ddots & \vdots \\
a_{1,s-1} & \cdots & a_{r,s-1}
\end{bmatrix}
\]

3. **Final reduction** \(a_i = d_{i,\ell} \text{ rem } m_\ell\) with \(d_{i,\ell}\) small \(O(rs)\)

### Complexity:
\(O(rs^{\omega-1})\) when \(r \geq s\).

**Speed-up:** \(s^{3-\omega}\) compared to naive algorithm

**Slower** than DAC algorithm (asymptotically)
**Problem:** Given residues $a_{i,j} = (a_i \text{ rem } m_j)_{1 \leq i \leq r}$, reconstruct $(a_1, \ldots, a_r) \in [0; M]^r$

**Formula:** 

$$a_i = \left( \sum_{j=1}^{s} \left( \left( u_j a_{i,j} \right) \text{ rem } m_i \right) M_i \right) \text{ rem } M$$

**Idea:**

1. If $\ell_i := \sum_{j=1}^{s} \gamma_{i,j} M_j$, then

   $\ell_i$ is a pseudo-reconstruction of $(a_{i,j})_{1 \leq j \leq s}$, i.e. $\ell_i = a_{i,j} \text{ mod } m_j$ and $\ell_i < s M$. 
Problem: Given residues $a_{i,j} = (a_i \text{ rem } m_j)_{1 \leq i \leq r}$, reconstruct $(a_1, \ldots, a_r) \in [0; M]^r$ with $1 \leq j \leq s$

Formula: $a_i = \left( \sum_{j=1}^{s} \left( (u_j a_{i,j} \text{ rem } m_j) M_i \right) \text{ rem } M \right) \gamma_{i,j}$

Idea:

1. If $\ell_i := \sum_{j=1}^{s} \gamma_{i,j} M_j$, then

   $\ell_i$ is a *pseudo-reconstruction* of $(a_{i,j})_{1 \leq j \leq s}$, i.e. $\ell_i = a_{i,j} \text{ mod } m_j$ and $\ell_i < s \times M$.

2. Decompose in linear operation with small entries:
   a. Write $M_j = \sum_{k=0}^{s-1} \mu_{j,k} \times 2^{kt}$ in base $2^t$
   b. Compute $\ell_i = \sum_{k=0}^{s-1} \left( \sum_{j=1}^{s} \gamma_{i,j} \mu_{j,k} \right) 2^{kt}$  $\sim$ decomposition of $\ell_i$ in base $2^t$
## Algorithm

1. Precompute $M_j = M / m_j$ and $u_j = (1 / M_j \text{rem } m_j)$ \(\mathcal{O}(s^2)\)

2. Compute all $\gamma_{i,j} = (u_j a_{i,j}) \text{rem } m_i$ \(\mathcal{O}(rs)\)

3. Compute base $2^t$ pseudo-decomposition $(d_{i,j})$ of $\ell_i$:

   \[
   \begin{bmatrix}
   d_{1,0} & \cdots & d_{1,s-1} \\
   \vdots & \ddots & \vdots \\
   d_{r,0} & \cdots & d_{r,s-1}
   \end{bmatrix}
   =
   \begin{bmatrix}
   \gamma_{1,1} & \cdots & \gamma_{1,s} \\
   \vdots & \ddots & \vdots \\
   \gamma_{r,1} & \cdots & \gamma_{r,s}
   \end{bmatrix}
   \begin{bmatrix}
   \mu_{1,0} & \cdots & \mu_{1,s-1} \\
   \vdots & \ddots & \vdots \\
   \mu_{s,0} & \cdots & \mu_{s,s-1}
   \end{bmatrix}
   \]

4. Recover base $2^t$ exact decomposition of $\ell_i = \sum d_{i,k} 2^{kt}$ \(\mathcal{O}(rs)\)

5. Final reconstruction: $a_i = \ell_i \text{rem } M$ \(\mathcal{O}(rs)\)

### Complexity:
\(\mathcal{O}(rs^{s^{-1}})\) when $r \geq s$.

### Speed-up:
$s^{3-\omega}$ compared to naive algorithm
**Question**: How to choose modulus bitsize $t$ ?

**Constraints:**

1. Matrix entries bitsize:
   
   Use BLAS so all matrices integer entries should fit in double, i.e.
   
   $$s m_i 2^t < 2^{53}$$  \hspace{1cm} (1)

2. Limited number of primes of at most $t$ bits
   
   $$s \leq 2^t / (t \ln(2))$$  \hspace{1cm} (2)
Goal 1: Maximize reachable bitsize of $M$

Since $\log_2(M) \sim s t$, constraints give

1. $\log_2(M) \leq 2^{53-2t} t$
2. $\log_2(M) \leq 2^t / \ln(2)$

Todo: Find $t$ to maximize $M$

So take $t = 19$ and maximum $M$ has 76 KBytes

(2^{15}$ moduli of bitsize $\leq 19$)
Now if $M$ is less than 76 KBytes,

**Goal 2:** Maximize moduli bitsize $t$

Since $\log_2(M) \sim s\ t$, constraints give

1. $\log_2(M) \leq 2^{53-2t} t$
2. $\log_2(M) \leq 2^t / \ln(2)$

**Todo:** Maximize $t$ given $M$

**Example:** If $M$ is 128 Bytes, take 45 moduli of bitsize 23

(instead of 54 moduli of bitsize 19 $\Rightarrow$ speedup $(54/45)^{\omega-1} \approx 1.44$)
### Timings – RNS reduction

<table>
<thead>
<tr>
<th>RNS bitsize – $\log_2(M)$</th>
<th>Naive - MMX</th>
<th>DAC - FLINT</th>
<th>LinAlg - FFLAS [speedup]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^8$</td>
<td>0.34</td>
<td>0.17</td>
<td>0.06 [x 2.8]</td>
</tr>
<tr>
<td>$2^9$</td>
<td>0.75</td>
<td>0.35</td>
<td>0.13 [x 2.7]</td>
</tr>
<tr>
<td>$2^{10}$</td>
<td>1.77</td>
<td>0.84</td>
<td>0.27 [x 3.1]</td>
</tr>
<tr>
<td>$2^{11}$</td>
<td>4.26</td>
<td>2.73</td>
<td>0.75 [x 3.6]</td>
</tr>
<tr>
<td>$2^{12}$</td>
<td>11.01</td>
<td>7.03</td>
<td>1.92 [x 3.7]</td>
</tr>
<tr>
<td>$2^{13}$</td>
<td>29.86</td>
<td>17.75</td>
<td>5.94 [x 3.0]</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td>88.95</td>
<td>50.90</td>
<td>21.09 [x 2.4]</td>
</tr>
<tr>
<td>$2^{15}$</td>
<td>301.69</td>
<td>165.80</td>
<td>80.82 [x 2.0]</td>
</tr>
<tr>
<td>$2^{16}$</td>
<td>1055.84</td>
<td>506.91</td>
<td>298.86 [x 1.7]</td>
</tr>
<tr>
<td>$2^{17}$</td>
<td>3973.46</td>
<td>1530.05</td>
<td>1107.23 [x 1.4]</td>
</tr>
<tr>
<td>$2^{18}$</td>
<td>15376.40</td>
<td>4820.63</td>
<td>4114.98 [x 1.2]</td>
</tr>
<tr>
<td>$2^{19}$</td>
<td>59693.64</td>
<td>13326.13</td>
<td>15491.90 [none]</td>
</tr>
</tbody>
</table>

**Figure.** Simultaneous RNS reductions (time per integer in $\mu s$)
### Timings – RNS reconstruction

<table>
<thead>
<tr>
<th>RNS bitsize $\log_2(M)$</th>
<th>Naive - MMX</th>
<th>DAC - FLINT</th>
<th>LinAlg - FFLAS [speedup]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^8$</td>
<td>0.74</td>
<td>0.63</td>
<td>0.34 [x 1.8]</td>
</tr>
<tr>
<td>$2^9$</td>
<td>1.04</td>
<td>1.34</td>
<td>0.39 [x 3.4]</td>
</tr>
<tr>
<td>$2^{10}$</td>
<td>1.86</td>
<td>3.12</td>
<td>0.72 [x 4.3]</td>
</tr>
<tr>
<td>$2^{11}$</td>
<td>4.29</td>
<td>6.92</td>
<td>1.57 [x 4.4]</td>
</tr>
<tr>
<td>$2^{12}$</td>
<td>12.18</td>
<td>16.79</td>
<td>3.94 [x 4.3]</td>
</tr>
<tr>
<td>$2^{13}$</td>
<td>43.89</td>
<td>40.73</td>
<td>12.77 [x 3.2]</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td>144.57</td>
<td>113.19</td>
<td>43.13 [x 2.6]</td>
</tr>
<tr>
<td>$2^{15}$</td>
<td>502.18</td>
<td>316.61</td>
<td>161.44 [x 2.0]</td>
</tr>
<tr>
<td>$2^{16}$</td>
<td>2187.65</td>
<td>855.48</td>
<td>609.22 [x 1.4]</td>
</tr>
<tr>
<td>$2^{17}$</td>
<td>10356.08</td>
<td>2337.96</td>
<td>2259.84 [x 1.1]</td>
</tr>
<tr>
<td>$2^{18}$</td>
<td>39965.23</td>
<td>7295.26</td>
<td>8283.64 [none]</td>
</tr>
<tr>
<td>$2^{19}$</td>
<td>156155.06</td>
<td>18529.38</td>
<td>31382.81 [none]</td>
</tr>
</tbody>
</table>

**Figure.** Simultaneous RNS reconstruction (time per integer in $\mu s$)

**Note:** Our precomputations are more costly: we need $\sim 1000$ $a_i$’s to amortize them.
**Extension for larger moduli**

<table>
<thead>
<tr>
<th>Application to integer polynomial multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem:</strong> For multiplication in $\mathbb{Z}[x]$, we prefer Fourier primes ($\exists 2^k$-root for $k$ large)</td>
</tr>
<tr>
<td>But there are not so many Fourier primes $\leq 2^{19}$!</td>
</tr>
<tr>
<td>$\rightsquigarrow$ How can we extend our moduli bitsize limit?</td>
</tr>
</tbody>
</table>

**Recall:** When $m_i \leq 2^t$ and $a_i = \sum_{j=0}^{s-1} a_{i,j} 2^{jt}$, pseudo-reduction $d_{i,\ell} := (\sum_{j=0}^{s-1} a_{i,j} [2^{jt}]_\ell)$

$$
\begin{bmatrix}
  d_{1,1} & \ldots & d_{r,1} \\
  \vdots & \ddots & \vdots \\
  d_{1,s} & \ldots & d_{r,s}
\end{bmatrix} = \begin{bmatrix}
  1 & [2^t]_1 & [2^{2t}]_1 & \ldots & [2^{(s-1)t}]_1 \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & [2^t]_s & [2^{2t}]_s & \ldots & [2^{(s-1)t}]_s
\end{bmatrix} \times \begin{bmatrix}
  a_{1,0} & \ldots & a_{r,0} \\
  \vdots & \ddots & \vdots \\
  a_{1,s-1} & \ldots & a_{r,s-1}
\end{bmatrix}
$$

**Idea:** If $m_i \leq (2^t)^\kappa$, cut $B$ into $B = B_0 + B_1 2^t + \ldots + B_{\kappa-1} 2^{t(\kappa-1)}$ and compute $BA$ as

$$D = (B_0 A) + \ldots + 2^{(\kappa-1)t} (B_{\kappa-1} A)$$
Cost of the extension is about the same:

1. $s$ moduli of bitsize $t$:
   
   One multiplication of matrices $(s \times s) \times (s \times r)$

2. $s/\kappa$ moduli of bitsize $\kappa t$:
   
   $\kappa$ multiplications of matrices $(s/\kappa \times s) \times (s \times r)$

| Bitsize $\log_2(M)$ | RNS reduction | | | RNS reconstruction | | |
|---------------------|--------------|--------|-----------------|--------|----------|----------|----------|
|                     | FLINT $m_i < 2^{59}$ | FFLAS $\kappa = 1$ $m_i < 2^{19}$ | FFLAS $\kappa = 2$ $m_i < 2^{38}$ | FLINT $m_i < 2^{59}$ | FFLAS $\kappa = 1$ $m_i < 2^{19}$ | FFLAS $\kappa = 2$ $m_i < 2^{38}$ |
| $2^9$               | 0.35         | 0.13   | 0.24            | 1.34   | 0.39     | 0.70     |
| $2^{10}$            | 0.84         | 0.27   | 0.53            | 3.12   | 0.72     | 1.39     |
| $2^{11}$            | 2.73         | 0.75   | 1.20            | 6.92   | 1.57     | 2.46     |
| $2^{12}$            | 7.03         | 1.92   | 2.92            | 16.79  | 3.94     | 5.15     |
| $2^{13}$            | 17.75        | 5.94   | 8.01            | 40.73  | 12.77    | 14.98    |
| $2^{14}$            | 50.90        | 21.09  | 25.05           | 113.19 | 43.13    | 47.54    |
| $2^{15}$            | 165.80       | 80.82  | 85.38           | 316.61 | 161.44   | 167.93   |
| $2^{16}$            | 506.91       | 298.86 | 299.11          | 855.48 | 609.22   | 629.69   |
| $2^{17}$            | 1530.05      | 1107.23| 1099.52         | 2337.96| 2259.84  | 2375.98  |
| $2^{18}$            | 4820.63      | 4114.98| 4043.68         | 7295.26| 8283.64  | 8550.81  |
| $2^{19}$            | 13326.13     | 15491.90| 15092.94       | 18529.38| 31382.81| 33967.42|
Conclusions:

1. Our approach is complementary with asymptotically fast algorithms
   We improves run-times for small and medium size

2. We exploit the available optimized implementations of matrix multiplication (BLAS)
   Reach peak performance of processors, gain a significant constant

3. If our gain is only constant, its impact is substantial to many important applications
   multiplication in $\mathcal{M}_{u,v}(\mathbb{Z}), \mathbb{Z}[x]$, polynomial factorization...

4. When prime bitsize limitation is a problem, we are still able to reduce the computation to
   matrix multiplication with small entries

Perspectives:

1. Implement hybrid version of linear algebra and divide-and-conquer strategies

2. Use different cutting for large moduli to provide further improvement