# 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*

# Context

**How to perform arithmetic operations on ... ?**

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

**Different approaches:**

1. **Direct algorithm:** matrix arithmetic then multi-precision integer arithmetic

   $\rightsquigarrow$ Best for matrix/polynomial size $\ll$ integer bitsize

2. **Modular approach:** Split the big integer matrix into many small integer matrices

   $\rightsquigarrow$ Best for matrix/polynomial size $\gg$ integer bitsize

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

# Context

**How to perform arithmetic operations on ... ?**

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

**Different approaches:**

1. **Direct algorithm:** matrix arithmetic then multi-precision integer arithmetic

   $\rightsquigarrow$ Best for matrix/polynomial size $\ll$ integer bitsize

2. **Modular approach:** Split the big integer matrix into many small integer matrices

   $\rightsquigarrow$ Best for matrix/polynomial size $\gg$ integer bitsize

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

# Outline

# Euclidean division



**Problem**: If $a, m \in \mathbb{N}$ , compute $q = a \,\mathsf{quo}\, m$ and $r = a \,\mathrm{rem}\, m$.

**Note**: $(n := \mathrm{bitsize}(a)) \geqslant (t := \mathrm{bitsize}(m))$

**Algorithms for Euclidean division**:

1. $n = 2t$ in time $\mathcal{O}(\mathsf{I}(t))$ – **Balanced case** [Cook '66], [Barrett '86]

   **Idea**: Newton iteration to approximate $1/m$ then $q = \lfloor a/m \rfloor$, finally $r = a - qm$

2. $n \geqslant 2t$ in time $\mathcal{O}\left(n \frac{\mathsf{I}(t)}{t}\right)$,

   **Idea**: Iterate base case reduction

3. $n \leqslant 2t$ in time $\mathcal{O}\left(t \frac{\mathsf{I}(n-t)}{n-t}\right)$ [Giorgi et al, '13]

   **Idea:** Compute $q$ only at precision $n - t$

# Euclidean division



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

**Note**: $(n := \mathrm{bitsize}(a)) \geqslant (t := \mathrm{bitsize}(m))$

**Algorithms for Euclidean division**:

1. $n = 2t$ in time $\mathcal{O}(\mathsf{I}(t))$ – **Balanced case** [Cook '66], [Barrett '86]

   **Idea**: Newton iteration to approximate $1/m$ then $q = \lfloor a/m \rfloor$, finally $r = a - qm$

2. $n \geqslant 2t$ in time $\mathcal{O}\left(n \frac{\mathsf{I}(t)}{t}\right)$,

   **Idea**: Iterate base case reduction

3. $n \leqslant 2t$ in time $\mathcal{O}\left(t \frac{\mathsf{I}(n-t)}{n-t}\right)$ [Giorgi et al, '13]

   **Idea:** Compute $q$ only at precision $n - t$

# Euclidean division 

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

**Note**: $(n := \mathrm{bitsize}(a)) \geqslant (t := \mathrm{bitsize}(m))$

**Algorithms for Euclidean division**:

1. $n = 2t$ in time $\mathcal{O}(\mathsf{I}(t))$ – **Balanced case** [Cook '66], [Barrett '86]

   **Idea**: Newton iteration to approximate $1/m$ then $q = \lfloor a/m \rfloor$, finally $r = a - qm$

2. $n \geqslant 2t$ in time $\mathcal{O}\left(n \frac{\mathsf{I}(t)}{t}\right)$,

   **Idea**: Iterate base case reduction

3. $n \leqslant 2t$ in time $\mathcal{O}\left(t \frac{\mathsf{I}(n-t)}{n-t}\right)$ [Giorgi et al, '13]

   **Idea:** Compute $q$ only at precision $n - t$

**Residue Number System:**

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

**Chinese remainder theorem:**

$$\begin{array}{ccc} \mathbb{Z}/(m_1 \cdots m_s)\mathbb{Z} & \underset{\text{Reconstruction}}{\overset{\text{Reduction}}{\rightleftarrows}} & \prod_{i=1}^{s} \mathbb{Z}/m_i\mathbb{Z} \\ a \bmod (m_1 \cdots m_s) & \longmapsto & (a \bmod m_1, \ldots, a \bmod m_s) \end{array}$$

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**

# RNS reduction



**Problem**: Compute $(a \operatorname{rem} m_i)_{i=1\dots s}$ where $a < M$ and $M = m_1 \cdots m_s$.

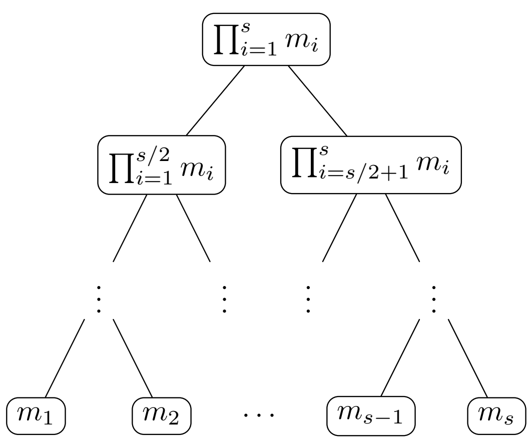
## Naive Algorithm

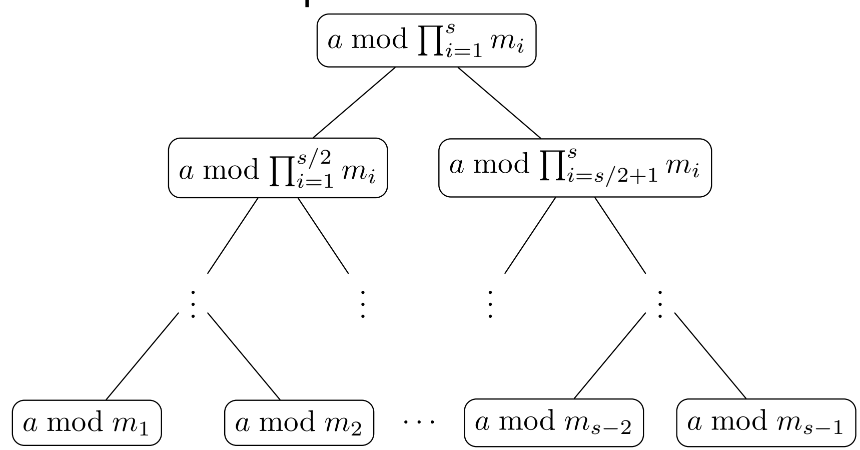
Apply $a \operatorname{rem} m_i$ in the case $(t=1) \ll (n=s)$ $\rightsquigarrow$ Quadratic time $\mathcal{O}(s^2)$

**Note**: Small constant factor in the big $\mathcal{O}$, faster for small $s$

## Divide-and-conquer algorithm

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




**Quasi-linear complexity**: $\mathcal{O}(\mathsf{I}(s) \log s)$

**Note**: Similar to polynomial case of multi-point evaluation

**Problem**: Given the evaluations $a_i = a(m_i) = a(X) \operatorname{rem} (X - m_i)$, reconstruct $a(X)$ using

$$a(X) = \sum_i a_i \prod_{j \neq i} \frac{(X - m_j)}{(m_i - m_j)} = \sum_i \underbrace{{}^{a_i}/_{M_i(m_i)}}_{b_i} M_i(X) \quad \text{where} \quad M_i(X) = M(X)/(X - m_i)$$

### 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 $b_i = {}^{a_i}/_{M_i(m_i)}$ 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

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

**Complexity**: $\mathcal{O}(\mathsf{M}(s) \log s)$

# RNS reconstruction – Maths



**Problem**: Given the reductions $a_i = a \operatorname{rem} m_i$, reconstruct $a$.

**Integer equivalent of Lagrange interpolation**

$$\mathbb{Z}/M\,\mathbb{Z} \longrightarrow (\mathbb{Z}/m_1\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_i\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_s\mathbb{Z})$$

$$\begin{array}{rcll}
? & \longmapsto & (a_1, \ldots, a_i, \ldots, a_s) & \\
M_i & \longmapsto & (0, \ldots, M_i \operatorname{rem} m_i, \ldots, 0) & M_i = m_1 \cdots \widehat{m_i} \cdots m_s \\
u_i\, M_i & \longmapsto & (0, \ldots, 1, \ldots, 0) & u_i = 1/M_i \bmod m_i \\
u_i\, a_i\, M_i & \longmapsto & (0, \ldots, a_i, \ldots, 0) & \\
\sum_{i=1}^{s} u_i\, a_i\, M_i & \longmapsto & (a_1, \ldots, a_i, \ldots, a_s) &
\end{array}$$

So $a$ given by

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

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

**Remark**: $((u_i\, a_i) \operatorname{rem} m_i)\, M_i$ corresponds to $\frac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}$ in Lagrange formula

# RNS reconstruction – Maths

**Problem**: Given the reductions $a_i = a \operatorname{rem} m_i$, reconstruct $a$.

### Integer equivalent of Lagrange interpolation

$$
\begin{array}{rcl}
\mathbb{Z}/M\,\mathbb{Z} & \longrightarrow & (\mathbb{Z}/m_1\,\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_i\,\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_s\,\mathbb{Z}) \\
 & & \\
? & \longmapsto & (a_1, \ldots, a_i, \ldots, a_s) \\
M_i & \longmapsto & (0, \ldots, M_i \operatorname{rem} m_i, \ldots, 0) \qquad M_i = m_1 \cdots \widehat{m_i} \cdots m_s \\
u_i\, M_i & \longmapsto & (0, \ldots, 1, \ldots, 0) \qquad u_i = 1/M_i \bmod m_i \\
u_i\, a_i\, M_i & \longmapsto & (0, \ldots, a_i, \ldots, 0) \\
\sum_{i=1}^{s} u_i\, a_i\, M_i & \longmapsto & (a_1, \ldots, a_i, \ldots, a_s)
\end{array}
$$

So $a$ given by

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

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

**Remark**: $((u_i\, a_i) \operatorname{rem} m_i)\, M_i$ corresponds to $\dfrac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}$ in Lagrange formula

# RNS reconstruction – Maths

**Problem**: Given the reductions $a_i = a \operatorname{rem} m_i$, reconstruct $a$.

## Integer equivalent of Lagrange interpolation

$$\mathbb{Z}/M\mathbb{Z} \longrightarrow (\mathbb{Z}/m_1\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_i\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_s\mathbb{Z})$$

$$? \longmapsto (a_1, \ldots, a_i, \ldots, a_s)$$

$$M_i \longmapsto (0, \ldots, M_i \operatorname{rem} m_i, \ldots, 0) \qquad M_i = m_1 \cdots \widehat{m_i} \cdots m_s$$

$$u_i M_i \longmapsto (0, \ldots, 1, \ldots, 0) \qquad u_i = 1/M_i \bmod m_i$$

$$u_i a_i M_i \longmapsto (0, \ldots, a_i, \ldots, 0)$$

$$\sum_{i=1}^{s} u_i a_i M_i \longmapsto (a_1, \ldots, a_i, \ldots, a_s)$$

So $a$ given by

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

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

**Remark**: $((u_i a_i) \operatorname{rem} m_i) M_i$ corresponds to $\frac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}$ in Lagrange formula

# RNS reconstruction – Maths 

**Problem**: Given the reductions $a_i = a \operatorname{rem} m_i$, reconstruct $a$.

**Integer equivalent of Lagrange interpolation**

$$\mathbb{Z}/M\,\mathbb{Z} \longrightarrow (\mathbb{Z}/m_1\,\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_i\,\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_s\,\mathbb{Z})$$

$$\begin{array}{rcll} ? & \longmapsto & (a_1, \ldots, a_i, \ldots, a_s) & \\ M_i & \longmapsto & (0, \ldots, M_i \operatorname{rem} m_i, \ldots, 0) & M_i = m_1 \cdots \widehat{m_i} \cdots m_s \\ u_i\,M_i & \longmapsto & (0, \ldots, 1, \ldots, 0) & u_i = 1/M_i \bmod m_i \\ u_i\,a_i\,M_i & \longmapsto & (0, \ldots, a_i, \ldots, 0) & \\ \sum_{i=1}^{s} u_i\,a_i\,M_i & \longmapsto & (a_1, \ldots, a_i, \ldots, a_s) & \end{array}$$

So $a$ given by

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

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

**Remark**: $((u_i\,a_i) \operatorname{rem} m_i)\,M_i$ corresponds to $\dfrac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}$ in Lagrange formula

# RNS reconstruction – Maths

**Problem**: Given the reductions $a_i = a \operatorname{rem} m_i$, reconstruct $a$.

**Integer equivalent of Lagrange interpolation**

$$\mathbb{Z}/M\mathbb{Z} \longrightarrow (\mathbb{Z}/m_1\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_i\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_s\mathbb{Z})$$

$$\begin{aligned}
? &\longmapsto (a_1, \ldots, a_i, \ldots, a_s) \\
M_i &\longmapsto (0, \ldots, M_i \operatorname{rem} m_i, \ldots, 0) && M_i = m_1 \cdots \widehat{m_i} \cdots m_s \\
u_i M_i &\longmapsto (0, \ldots, 1, \ldots, 0) && u_i = 1/M_i \bmod m_i \\
u_i a_i M_i &\longmapsto (0, \ldots, a_i, \ldots, 0) \\
\textstyle\sum_{i=1}^s u_i a_i M_i &\longmapsto (a_1, \ldots, a_i, \ldots, a_s)
\end{aligned}$$

So $a$ given by

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

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

**Remark**: $((u_i a_i) \operatorname{rem} m_i) M_i$ corresponds to $\dfrac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}$ in Lagrange formula

# RNS reconstruction – Maths

**Problem**: Given the reductions $a_i = a \operatorname{rem} m_i$, reconstruct $a$.

**Integer equivalent of Lagrange interpolation**

$$\mathbb{Z}/M\,\mathbb{Z} \longrightarrow (\mathbb{Z}/m_1\,\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_i\,\mathbb{Z}) \times \cdots \times (\mathbb{Z}/m_s\,\mathbb{Z})$$

$$\begin{array}{rcll}
? & \longmapsto & (a_1, \ldots, a_i, \ldots, a_s) & \\
M_i & \longmapsto & (0, \ldots, M_i \operatorname{rem} m_i, \ldots, 0) & M_i = m_1 \cdots \widehat{m_i} \cdots m_s \\
u_i\,M_i & \longmapsto & (0, \ldots, 1, \ldots, 0) & u_i = 1/M_i \bmod m_i \\
u_i\,a_i\,M_i & \longmapsto & (0, \ldots, a_i, \ldots, 0) & \\
\sum_{i=1}^{s} u_i\,a_i\,M_i & \longmapsto & (a_1, \ldots, a_i, \ldots, a_s) &
\end{array}$$

So $a$ given by

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

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

**Remark**: $((u_i\,a_i) \operatorname{rem} m_i)\,M_i$ corresponds to $\dfrac{a_i \prod_{j \neq i} (X - m_j)}{\prod_{j \neq i} (m_i - m_j)}$ in Lagrange formula

# RNS reconstruction – Algorithm

## Divide-and-conquer algorithm

1. Compute $M_i \operatorname{rem} m_i$ $\rightsquigarrow M_i(m_i) = M'(m_i)$? No derivative $M'$ !
2. Compute $u_i = 1 / M_i \bmod m_i$
3. Compute $b_i := (u_i\, a_i) \operatorname{rem} m_i$
4. Compute $P = \sum_i b_i\, M_i$ $\rightsquigarrow$ Same recursion $P = P^0\, M^1 + P^1\, M^0$

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

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

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

# RNS reconstruction – Algorithm

**Divide-and-conquer algorithm**

1. Compute $M_i \,\mathrm{rem}\, m_i$ $\rightsquigarrow M_i(m_i) = M'(m_i)$? No derivative $M'$ !
2. Compute $u_i = 1 / M_i \bmod m_i$
3. Compute $b_i := (u_i\, a_i) \,\mathrm{rem}\, m_i$
4. Compute $P = \sum_i b_i\, M_i$ $\rightsquigarrow$ Same recursion $P = P^0\, M^1 + P^1\, M^0$

**Trick for $M_i \,\mathrm{rem}\, m_i$**

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

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

# RNS reconstruction – Algorithm

## Divide-and-conquer algorithm

1. Compute $M_i \operatorname{rem} m_i$ $\rightsquigarrow M_i(m_i) = M'(m_i)$? No derivative $M'$ !
2. Compute $u_i = 1 / M_i \bmod m_i$
3. Compute $b_i := (u_i \, a_i) \operatorname{rem} m_i$
4. Compute $P = \sum_i b_i \, M_i$ $\rightsquigarrow$ Same recursion $P = P^0 \, M^1 + P^1 \, M^0$

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

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

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

## Divide-and-conquer algorithm

1. Compute $M_i \operatorname{rem} m_i$ $\rightsquigarrow M_i(m_i) = M'(m_i)$? No derivative $M'$ !
2. Compute $u_i = 1 / M_i \bmod m_i$
3. Compute $b_i := (u_i\, a_i) \operatorname{rem} m_i$
4. Compute $P = \sum_i b_i\, M_i$ $\rightsquigarrow$ Same recursion $P = P^0\, M^1 + P^1\, M^0$

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

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

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

# Our contribution



**Fact**: We don't know how to improve one RNS conversion !

End of of the talk ? No !

# Our contribution

**Fact**: We don't know how to improve one RNS conversion !

End of 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 $(a_i \operatorname{rem} m_j)_{i=1\ldots r, j=1\ldots s}$

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

1. Naive algorithms: $\mathcal{O}(r\, s^2)$
2. DAC algorithms: $\mathcal{O}(r\, \mathsf{I}(s) \log s)$

**Our contribution** [Doliskani, Giorgi, Lebreton, Schost '17]

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

(using precomputation of time $\mathcal{O}(s^2)$)

# Outline

# RNS reduction using linear algebra - Idea

**Problem**: Given $\boldsymbol{a} = (a_1, \ldots, a_r) \in [\![0; M[\![^r$, compute $(a_i \operatorname{rem} m_j)_{i=1\ldots r, j=1\ldots s}$

**Note:** $[\_]_\ell = \_ \bmod m_\ell$ and $t$ such that $m_i \leqslant 2^t$

**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. $a_i = d_{i,\ell} \bmod m_\ell$ and $d_{i,\ell} \leqslant 2^{4t}$

# RNS reduction using linear algebra - Complexity



**Algorithm**

1. Precompute $([2^{jt}]_i)_{1 \leqslant i,j \leqslant s}$ $\mathcal{O}(s^2)$
2. Matrix multiplication with small integer entries $\mathcal{O}(\mathsf{MM}(s,s,r))$

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

3. Final reduction $a_i = d_{i,\ell} \operatorname{rem} m_\ell$ with $d_{i,\ell}$ small $\mathcal{O}(r\,s)$

**Complexity**: $\mathcal{O}(r\,s^{\omega-1})$ when $r \geqslant s$.

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

**Slower** than DAC algorithm (asymptotically)

**Problem**: Given residues $a_{i,j} = (a_i \operatorname{rem} m_j)_{\substack{1 \leqslant i \leqslant r \\ 1 \leqslant j \leqslant s}}$, reconstruct $(a_1, \ldots, a_r) \in [\![0; M[\![^r$

**Formula**: $a_i = \left( \sum_{j=1}^{s} \underbrace{((u_j\, a_{i,j}) \operatorname{rem} m_i)}_{\gamma_{i,j}} M_i \right) \operatorname{rem} M$

**Idea**:

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

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

**Problem**: Given residues $a_{i,j} = (a_i \operatorname{rem} m_j)_{\substack{1 \leqslant i \leqslant r \\ 1 \leqslant j \leqslant s}}$, reconstruct $(a_1, \ldots, a_r) \in [\![0; M[\![^r$

**Formula**: $a_i = \left( \sum_{j=1}^{s} \underbrace{((u_j\, a_{i,j}) \operatorname{rem} m_i)}_{\gamma_{i,j}} M_i \right) \operatorname{rem} M$

**Idea**:

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

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

2. Decompose in linear operation with small entries :

   a. Write $M_j = \sum_{k=0}^{s-1} \mu_{j,k}\, 2^{kt}$ in base $2^t$

   b. Compute $\ell_i = \sum_{k=0}^{s-1} \left( \sum_{j=1}^{s} \gamma_{i,j}\ \mu_{j,k} \right) 2^{kt}$ $\quad\simeq$ decomposition of $\ell_i$ in base $2^t$

# RNS reconstruction using linear algebra - Complexity 

**Algorithm**

1. Precompute $M_j = M / m_j$ and $u_j = (1 / M_j \operatorname{rem} m_j)$ — $\mathcal{O}(s^2)$
2. Compute all $\gamma_{i,j} = (u_j a_{i,j}) \operatorname{rem} m_i$ — $\mathcal{O}(r\, s)$
3. Compute base $2^t$ pseudo-decomposition $(d_{i,j})$ of $\ell_i$ : — $\mathcal{O}(\mathsf{MM}(r, s, s))$

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

4. Recover base $2^t$ exact decomposition of $\ell_i = \sum d_{i,k} 2^{kt}$ — $\mathcal{O}(r\, s)$
5. Final reconstruction: $a_i = \ell_i \operatorname{rem} M$ — $\mathcal{O}(r\, s)$

**Complexity**: $\mathcal{O}(r\, s^{\omega - 1})$ when $r \geqslant s$.

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

# Implementation details



**Question**: How to choose modulus bitsize $t$ ?

**Constraints**:

1. Matrix entries bitsize:

   Use BLAS so all matrices integer entries should fit in $\mathrm{double}$, i.e.

$$s\, m_i\, 2^t < 2^{53} \qquad (1)$$

2. Limited number of primes of at most $t$ bits

$$s \leqslant 2^t / (t \ln(2)) \qquad (2)$$

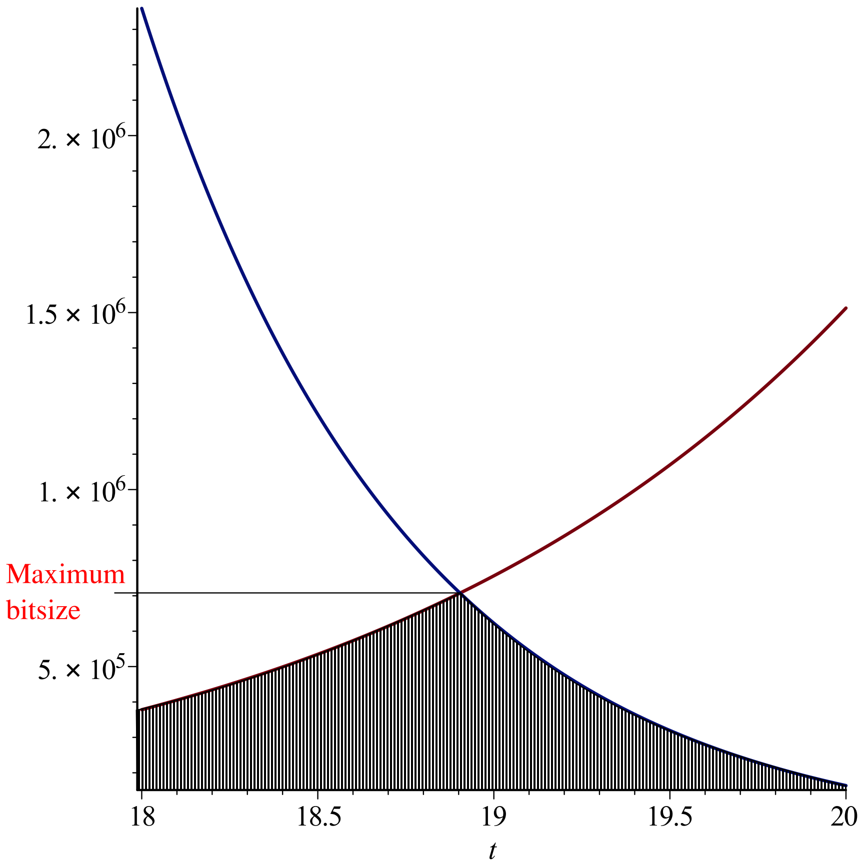
# Implementation details – Maximum bitsize

**Goal 1**: Maximize reachable bitsize of $M$

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

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

**Todo**: Find $t$ to maximize $M$




So take $t = 19$ and maximum $M$ has $76$ KBytes ($2^{15}$ moduli of bitsize $\leqslant 19$)

# Implementation details – Optimize $t$



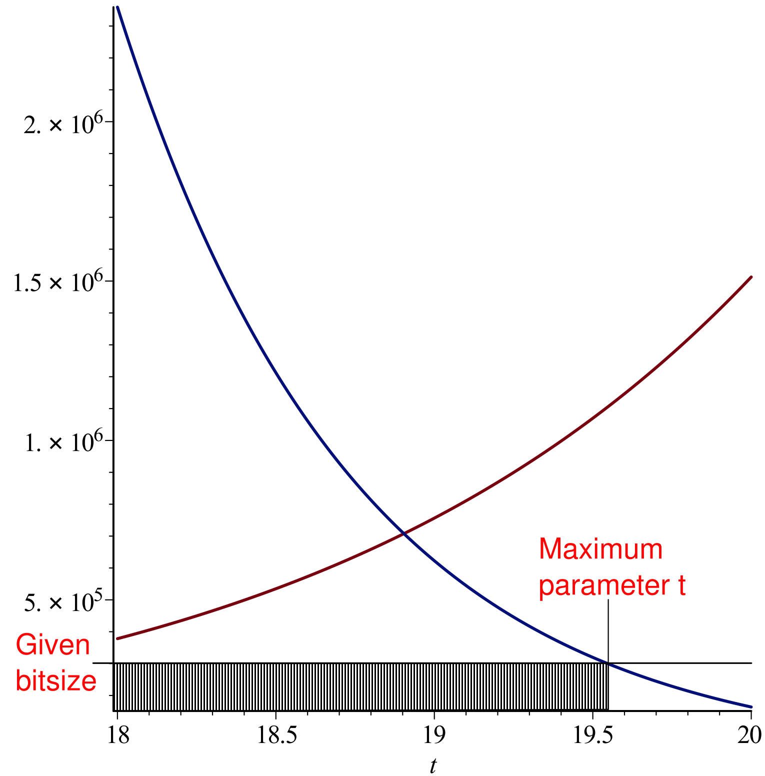
Now if $M$ is less than $76$ KBytes,

**Goal 2:** Maximize moduli bitsize $t$ ($=$ Reduce number $s$ of moduli)

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

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

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




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

(instead of $54$ moduli of bitsize $19 \rightsquigarrow$ speedup $(54/45)^{\omega-1} \simeq 1.44$)

# Timings – RNS reduction

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

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

# Timings – RNS reconstruction



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

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

**Note**: Our precomputations are more costly: we need $\simeq 1000$ $a_i$'s to amortize them.

**Application to integer polynomial multiplication**

**Problem**: For multiplication in $\mathbb{Z}[x]$, we prefer Fourier primes ($\exists\ 2^k$-root for $k$ large)

But there are not so many Fourier primes $\leqslant 2^{19}$ !

$\rightsquigarrow$ **How can we extend our moduli bitsize limit ?**

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

$$\underbrace{\begin{bmatrix} d_{1,1} & \dots & d_{r,1} \\ \vdots & & \vdots \\ d_{1,s} & \dots & d_{r,s} \end{bmatrix}}_{D} = \underbrace{\begin{bmatrix} 1 & [2^t]_1 & [2^{2t}]_1 & \dots & [2^{(s-1)t}]_1 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & [2^t]_s & [2^{2t}]_s & \dots & [2^{(s-1)t}]_s \end{bmatrix}}_{B} \times \underbrace{\begin{bmatrix} a_{1,0} & \dots & a_{r,0} \\ \vdots & & \vdots \\ a_{1,s-1} & \dots & a_{r,s-1} \end{bmatrix}}_{A}$$

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

$$D = (B_0\, A) + \ldots + 2^{(\kappa-1)t}\,(B_{\kappa-1}\, A)$$

# Complexity and timings of extended moduli variant

## 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 $(s/\kappa \times s) \times (s \times r)$

| | RNS reduction | | | RNS reconstruction | | |
|---|---|---|---|---|---|---|
| **Bitsize** $\log_2(M)$ | **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 |

# Conclusion 

**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