# Toward High Performance Matrix Multiplication for Exact Computation

**Pascal Giorgi**

Joint work with Romain Lebreton (U. Waterloo)
Funded by the French ANR project HPAC

LIRMM
UNIVERSITÉ MONTPELLIER 2
SCIENCES ET TECHNIQUES
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

*Séminaire CASYS - LJK, April 2014*

# Motivations

- Matrix multiplication plays a central role in computer algebra.
  *algebraic complexity of* $O(n^\omega)$ *with* $\omega < 2.3727$ [Williams 2011]

- Modern processors provide many levels of parallelism.
  *superscalar, SIMD units, multiple cores*

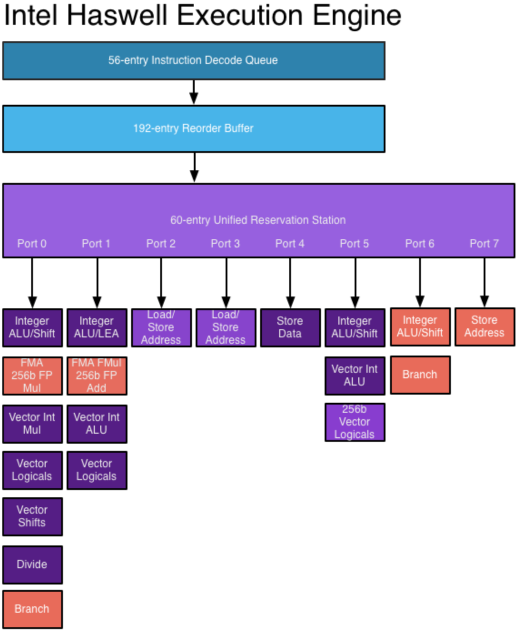
**High performance matrix multiplication**

✓ numerical computing $=$ classic algorithm $+$ hardware arithmetic

✗ exact computing $\neq$ numerical computing

- algebraic algorithm is not the most efficient ($\neq$ complexity model)
- arithmetic is not directly in the hardware (e.g. $\mathbb{Z}, \mathsf{F}_q, \mathbb{Z}[x], \mathbb{Q}[x, y, z]$).

# Motivation : Superscalar processor with SIMD


Hierarchical memory :

- L1 cache : $32kB$ - $4\ cycles$
- L2 cache : $256kB$ - $12\ cycles$
- L3 cache : $8MB$ - $36\ cycles$
- RAM : $32GB$ - $36\ cycles + 57ns$

# Motivations : practical algorithms

## High performance algorithms (rule of thumb)

- best asymptotic complexity is not alway faster : constants matter
- better arithmetic count is not always faster : caches matter
- process multiple data at the same time : vectorization
- fine/coarse grain task parallelism matter : multicore parallelism

# Motivations : practical algorithms

**High performance algorithms (rule of thumb)**

- best asymptotic complexity is not alway faster : constants matter
- better arithmetic count is not always faster : caches matter
- process multiple data at the same time : vectorization
- fine/coarse grain task parallelism matter : multicore parallelism

Our goal : try to incorporate these rules into exact matrix multiplications

# Outline

1. Matrix multiplication with small integers
2. Matrix multiplication with multi-precision integers
3. Matrix multiplication with polynomials

# Outline

# Matrix multiplication with small integers

This corresponds to the case where each integer result holds in one processor register :

$$A, B \in \mathbb{Z}^{n\times n} \text{ such that } ||AB||_\infty < 2^s$$

where $s$ is the register size.

**Main interests**

- ring isomorphism :
  $\rightarrow$ computation over $\mathbb{Z}/p\mathbb{Z}$ is congruent to $\mathbb{Z}/2^s\mathbb{Z}$ when $p(n-1)^2 < 2^s$.
- its a building block for matrix mutiplication with larger integers

# Matrix multiplication with small integers

Two possibilities for hardware support :

- use floating point mantissa, i.e. $s = 2^{53}$,
- use native integer, i.e. $s = 2^{64}$.

## Using floating point

historically, the first approach in computer algebra [Dumas, Gautier, Pernet 2002]

- ✓ out of the box performance from optimized BLAS
- ✗ handle matrix with entries $< 2^{26}$

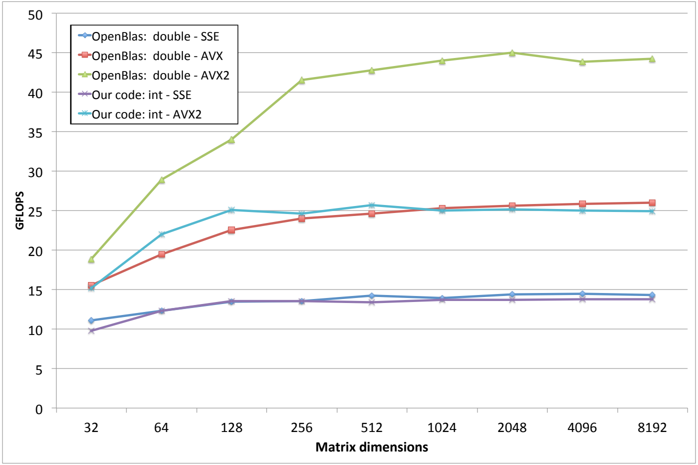
## Using native integers

- ✓ apply same optimizations as BLAS libraries [Goto, Van De Geijn 2008]
- ✓ handle matrix with entries $< 2^{32}$

# Matrix multiplication with small integers

| | | floating point | integers |
|---|---|---|---|
| Nehalem (2008) | SSE4 128-bits | 1 mul+1 add | 1 mul+2 add |
| Sandy Bridge (2011) | AVX 256-bits | 1 mul+1 add | |
| Haswell (2013) | AVX2 256-bits | 2 FMA | 1 mul+2 add |

# vector operations per cycle (pipelined)



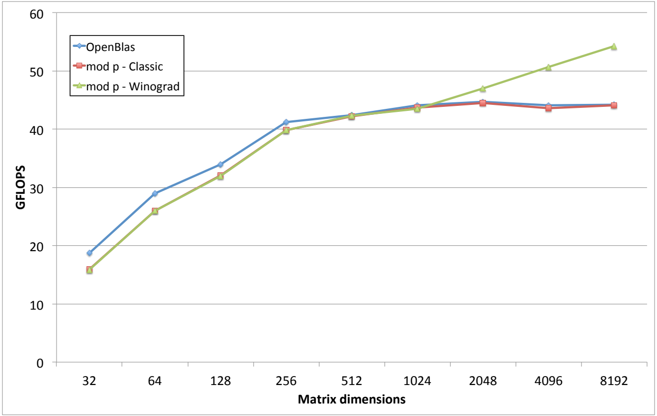

*benchmark on Intel i7-4960HQ @ 2.60GHz*

# Matrix multiplication with small integers

## Matrix multiplication modulo a small integer

Let $p$ such that $(p-1)^2 \times n < 2^{53}$

1. perform the multiplication in $\mathbb{Z}$ using BLAS
2. reduce the result modulo $p$




*benchmark on Intel i7-4960HQ @ 2.60GHz*

# Outline

# Matrix multiplication with multi-precision integers

**Direct approach**

Let $\mathsf{M}(k)$ be the bit complexity of $k$-bit integers multiplication and

$$A, B \in \mathbb{Z}^{n \times n} \text{ such that } ||A||_\infty, ||B||_\infty \in O(2^k).$$

Computing $AB$ using direct algorithm costs $n^\omega \mathsf{M}(k)$ bit operations.

- ✗ not best possible complexity, i.e. $\mathsf{M}(k)$ is super-linear
- ✗ not efficient in practice

Remark:

Use evaluation/interpolation technique for better performances ! ! !

# Multi-modular matrix multiplication

**Multi-modular approach**

$$||AB||_\infty < M = \prod_{i=1}^{k} m_i, \quad \text{with primes } m_i \in O(1)$$

then $AB$ can be reconstructed with the CRT from $(AB) \bmod m_i$.

1. for each $m_i$ compute $A_i = A \bmod m_i$ and $B_i = B \bmod m_i$
2. for each $m_i$ compute $C_i = A_i B_i \bmod m_i$
3. reconstruct $C = AB$ from $(C_1, \ldots, C_k)$

Bit complexity :
$O(n^\omega k + n^2 R(k))$ where $R(k)$ is the cost of reduction/reconstruction

# Multi-modular matrix multiplication

## Multi-modular approach

$$||AB||_\infty < M = \prod_{i=1}^{k} m_i, \quad \text{with primes } m_i \in O(1)$$

then $AB$ can be reconstructed with the CRT from $(AB) \bmod m_i$.

1. for each $m_i$ compute $A_i = A \bmod m_i$ and $B_i = B \bmod m_i$
2. for each $m_i$ compute $C_i = A_i B_i \bmod m_i$
3. reconstruct $C = AB$ from $(C_1, \ldots, C_k)$

Bit complexity :
$O(n^\omega k + n^2 R(k))$ where $R(k)$ is the cost of reduction/reconstruction

- $R(k) = O(M(k)\log(k))$ using divide and conquer strategy
- $R(k) = O(k^2)$ using naive approach

# Multi-modular matrix multiplication

**Improving naive approach with linear algebra**

reduction/reconstruction of $n^2$ data corresponds to matrix multiplication

- ✓ improve the bit complexity from $O(n^2k^2)$ to $O(n^2k^{\omega-1})$
- ✓ benefit from optimized matrix multiplication, i.e. SIMD

Remark :
A similar approach has been used by [Doliskani, Schost 2010] in a non-distributed code.

# Multi-modular reductions of an integer matrix

Let us assume $M = \prod_{i=1}^{k} m_i < \beta^k$ with $m_i < \beta$.

**Multi-reduction of a single entry**

Let $a = a_0 + a_1\beta + \ldots a_{k-1}\beta^{k-1}$ be a value to reduce $\bmod m_i$ then

$$\begin{bmatrix} |a|_{m_1} \\ \vdots \\ |a|_{m_k} \end{bmatrix} = \begin{bmatrix} 1 & |\beta|_{m_1} & \ldots & |\beta^{k-1}|_{m_1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & |\beta|_{m_k} & \ldots & |\beta^{k-1}|_{m_k} \end{bmatrix} \times \begin{bmatrix} a0 \\ \vdots \\ a_{k-1} \end{bmatrix} - Q \times \begin{bmatrix} m_1 \\ \vdots \\ m_k \end{bmatrix}$$

with $||Q||_\infty < k\beta^2$

Lemma : if $k\beta^2 \in O(1)$ than the reduction of $n^2$ integers modulo the $m_i$'s costs $O(n^2 k^{\omega-1}) + O(n^2 k)$ bit operations.

# Multi-modular reconstruction of an integer matrix

Let us assume $M = \prod_{i=1}^{k} m_i < \beta^k$ with $m_i < \beta$ and $M_i = M/m_i$

$$\text{CRT formulae : } a = (\sum_{i=1}^{k} |a|_{m_1} \cdot M_i |M_i^{-1}|_{m_i}) \bmod M$$

## Reconstruction of a single entry

Let $M_i|M_i^{-1}|_{m_i} = \alpha_0^{(i)} + \alpha_1^{(i)}\beta + \ldots \alpha_{k-1}^{(i)}\beta^{k-1}$ be the CRT constants, then

$$\begin{bmatrix} a_0 \\ \vdots \\ a_{k-1} \end{bmatrix} = \begin{bmatrix} \alpha_0^{(1)} & \ldots & \alpha_{k-1}^{(1)} \\ \vdots & \ddots & \vdots \\ \alpha_0^{(k)} & \ldots & \alpha_{k-1}^{(k)} \end{bmatrix} \times \begin{bmatrix} |a|_{m_1} \\ \vdots \\ |a|_{m_k} \end{bmatrix}$$

with $a_i < k\beta^2$ and $a = a_0 + \ldots + a_{k-1}\beta^{k-1} \bmod M$ the CRT solution.

<u>Lemma</u> : if $k\beta^2 \in O(1)$ than the reconstruction of $n^2$ integers from their images modulo the $m_i$'s costs $O(n^2 k^{\omega-1}) + \tilde{O}(n^2 k)$ bit operations.

# Matrix multiplication with multi-precision integers

**Implementation of multi-modular approach**

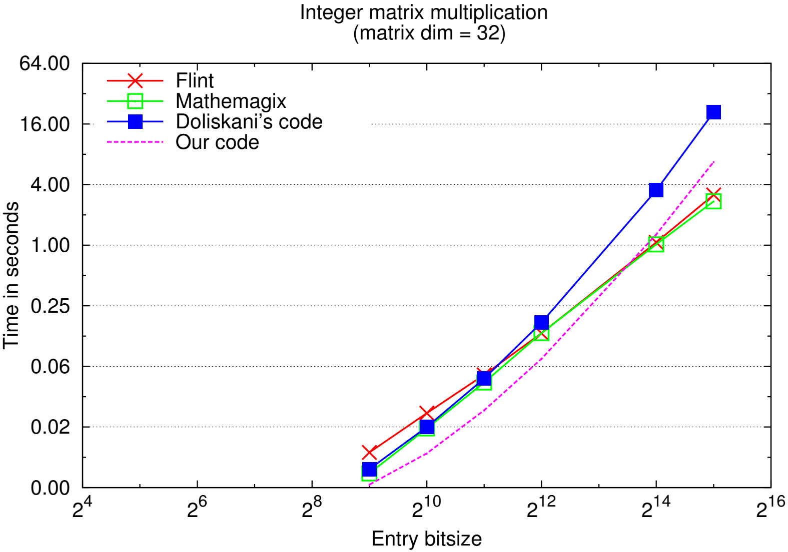
- choose $\beta = 2^{16}$ to optimize $\beta$-adic conversions
- choose $m_i$ s.t. $n\beta m_i < 2^{53}$ and use BLAS `dgemm`
- use a linear storage for multi-modular matrices

Compare sequential performances with :

- FLINT library [1] : uses divide and conquer
- Mathemagix library [2] : uses divide and conquer
- Doliskani's code [3] : uses `dgemm` for reductions only

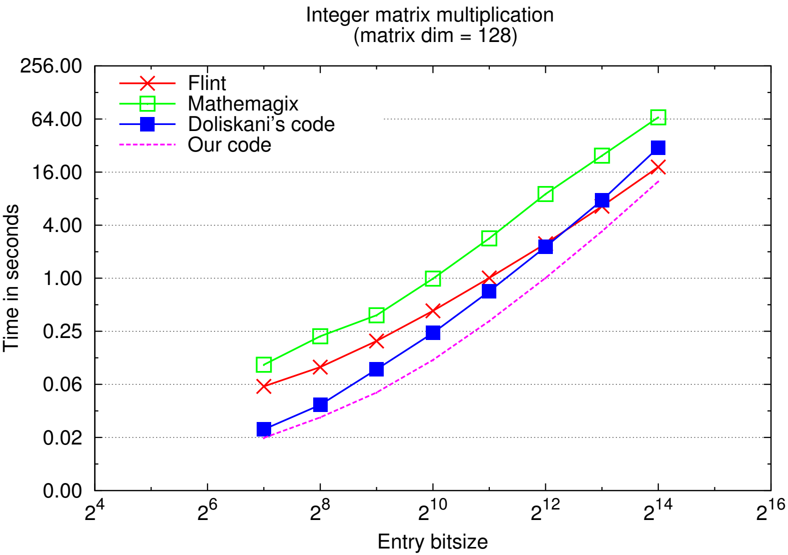
1. www.flintlib.org
2. www.mathemagix.org
3. courtesy of J. Doliskani

# Matrix multiplication with multi-precision integers



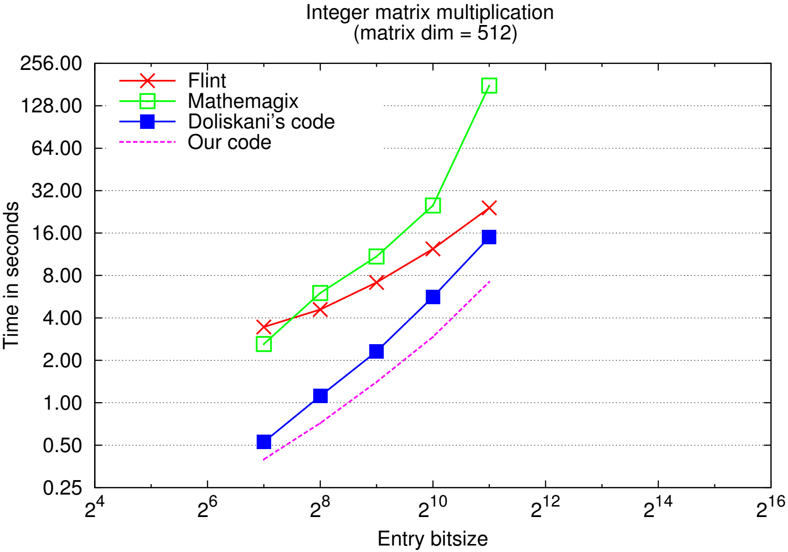

*benchmark on Intel Xeon-2620 @ 2.0GHz*

# Matrix multiplication with multi-precision integers



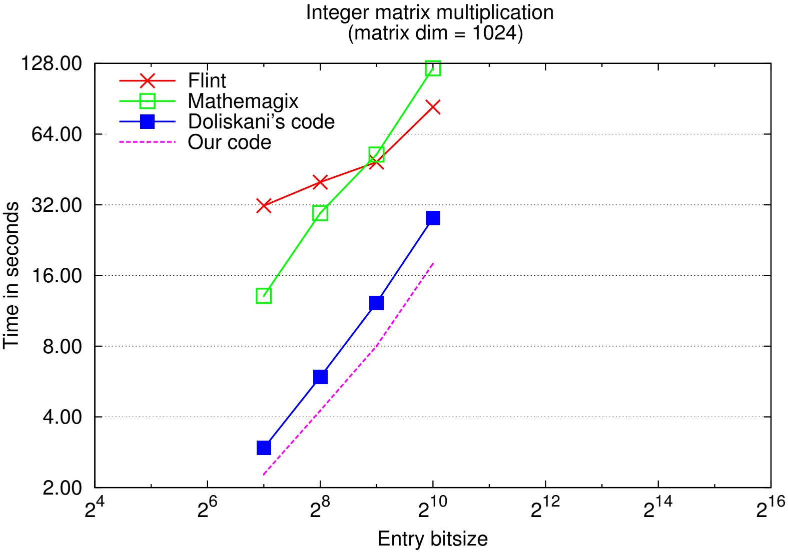

*benchmark on Intel Xeon-2620 @ 2.0GHz*

# Matrix multiplication with multi-precision integers




*benchmark on Intel Xeon-2620 @ 2.0GHz*

# Matrix multiplication with multi-precision integers




*benchmark on Intel Xeon-2620 @ 2.0GHz*

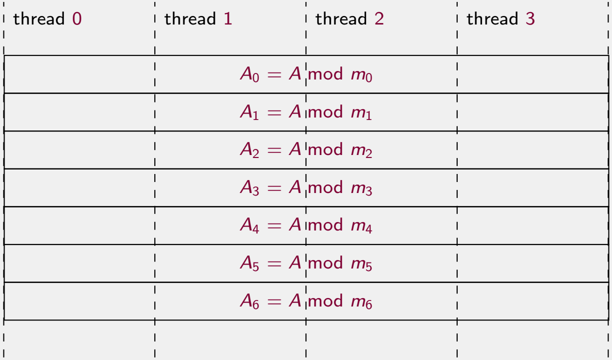
# Parallel multi-modular matrix multiplication

1. for $i = 1 \ldots k$ compute $A_i = A \bmod m_i$ and $B_i = B \bmod m_i$

3. reconstruct $C = AB$ from $(C_1, \ldots, C_k)$

**Parallelization of multi-modular reduction/reconstruction**

each thread reduces (resp. reconstructs) a chunk of the given matrix



# Parallel multi-modular matrix multiplication

2. for $i = 1 \dots k$ compute $C_i = A_i B_i \bmod m_i$

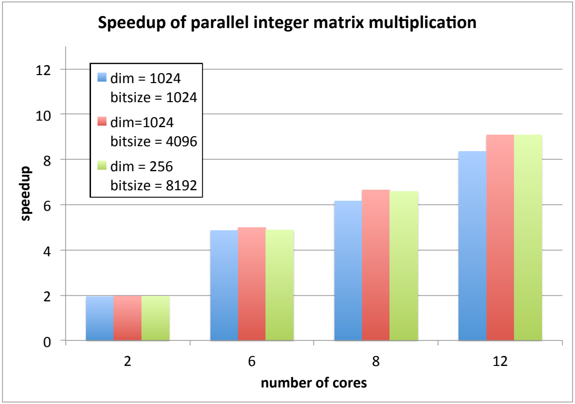
## Parallelization of modular multiplication

each thread computes a bunch of matrix multiplications $\bmod m_i$

| | |
|---|---|
| thread 0 | $C_0 = A_0 B_0 \bmod m_0$ |
| | $C_1 = A_1 B_1 \bmod m_1$ |
| thread 1 | $C_2 = A_2 B_2 \bmod m_2$ |
| | $C_3 = A_3 B_3 \bmod m_3$ |
| thread 2 | $C_4 = A_4 B_4 \bmod m_4$ |
| | $C_5 = A_5 B_5 \bmod m_5$ |
| thread 3 | $C_6 = A_6 B_6 \bmod m_6$ |

# Parallel Matrix multiplication with multi-precision integers

- based on OpenMP task
- CPU affinity (hwloc-bind), allocator (tcmalloc)
- still under progress for better memory strategy ! ! !




*benchmark on Intel Xeon-2620 @ 2.0GHz (2 NUMA with 6 cores)*

# Outline

# Matrix multiplication over $F_p[x]$

We consider the "easiest" case :

$$A, B \in F_p[x]^{n \times n} \text{ such that } \deg(AB) < k = 2^t$$

- $p$ is a Fourier prime, i.e. $p = 2^t q + 1$
- $p$ is such that $n(p-1)^2 < 2^{53}$

**Complexity**

$O(n^\omega k + n^2 k \log(k))$ op. in $F_p$ using evaluation/interpolation with FFT

# Matrix multiplication over $\mathrm{F_p[x]}$

We consider the "easiest" case :

$$A, B \in \mathrm{F_p[x]}^{\mathrm{n \times n}} \text{ such that } \mathrm{deg(AB)} < \mathrm{k} = 2^{\mathrm{t}}$$

- $p$ is a Fourier prime, i.e. $p = 2^t q + 1$
- $p$ is such that $n(p-1)^2 < 2^{53}$

**Complexity**

$O(n^{\omega} k + n^2 k \log(k))$ op. in $\mathrm{F_p}$ using evaluation/interpolation with FFT

Remark:

using Vandermonde matrix on can get a similar approach as for integers, i.e. $O(n^{\omega} k + n^2 k^{\omega - 1})$

# Matrix multiplication over $F_p[x]$

**Evaluation/Interpolation scheme**

Let $\theta$ a primitive $k$th root of unity in $F_p$.

1. for $i = 1 \dots k$ compute $A_i = A(\theta^{i-1})$ and $B_i = B(\theta^{i-1})$
2. for $i = 1 \dots k$ compute $C_i = A_i B_i \in F_p$
3. interpolate $C = AB$ from $(C_1, \dots, C_k)$

- steps 1 and 3 : $O(n^2)$ call to $\mathtt{FFT}_k$ over $F_p[x]$
- step 2 : $k$ matrix multiplications modulo a small prime $p$

# FFT with SIMD over $F_p$

**Butterly operation modulo p**

compute $X + Y \bmod p$ and $(X - Y)\theta^{2^i} \bmod p$.

- Barret's modular multiplication with a constant (NTL)
- calculate into $[0, 2p)$ to remove two conditionals [Harvey 2014]

Let $X, Y \in [0, 2p), W \in [0, p), p < \beta/4$ and $W' = \lceil W\beta/p \rceil$.

---

**Algorithm:** Butterfly(X,Y,W,W',p)

---

$X' := X + Y \bmod 2p$
$T := X - Y + 2p$
$Q := \lceil W'T/\beta \rceil$ — 1 high short product
$Y' := (WT - Qp) \bmod \beta$ — 2 low short products
return $(X', Y')$

---

# FFT with SIMD over $F_p$

**Butterly operation modulo p**

compute $X + Y \bmod p$ and $(X - Y)\theta^{2^i} \bmod p$.

- Barret's modular multiplication with a constant (NTL)
- calculate into $[0, 2p)$ to remove two conditionals [Harvey 2014]

Let $X, Y \in [0, 2p)$, $W \in [0, p)$, $p < \beta/4$ and $W' = \lceil W\beta/p \rceil$.

**Algorithm:** Butterfly(X,Y,W,W',p)

1: $X' := X + Y \bmod 2p$
2: $T := X - Y + 2p$
3: $Q := \lceil W'T/\beta \rceil$ — 1 high short product
4: $Y' := (WT - Qp) \bmod \beta$ — 2 low short products
5: return $(X', Y')$

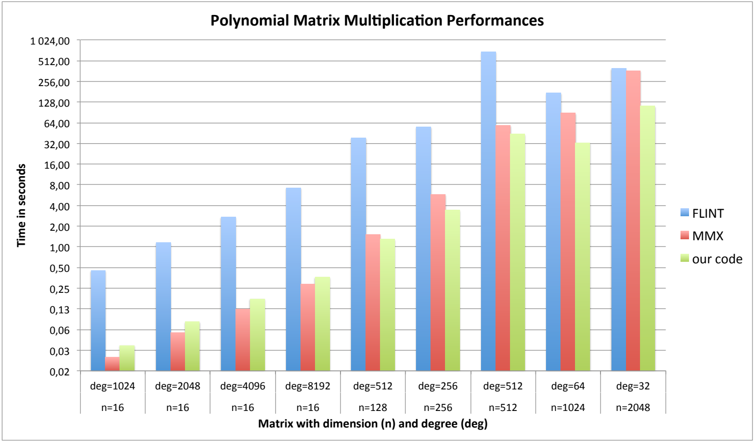
✓ SSE/AVX provide 16 or 32-bits low short product

✗ no high short product available (use full product)

# Matrix multiplication over $F_p[x]$

## Implementation

- radix-4 FFT with 128-bits SSE (29 bits primes)
- BLAS-based matrix multiplication over $F_p$ [FFLAS-FFPACK library]




*benchmark on Intel Xeon-2620 @ 2.0GHz*

# Matrix multiplication over $\mathbb{Z}[x]$

$$A, B \in \mathbb{Z}[x]^{n\times n} \text{ such that } \deg(AB) < d \text{ and } ||(AB)_i||_\infty < k$$

**Complexity**

- $\tilde{O}(n^\omega d \log(d) \log(k))$ bit op. using Kronecker substitution
- $O(n^\omega d \log(k) + n^2 d \log(d) \log(k))$ bit op. using CRT+FFT

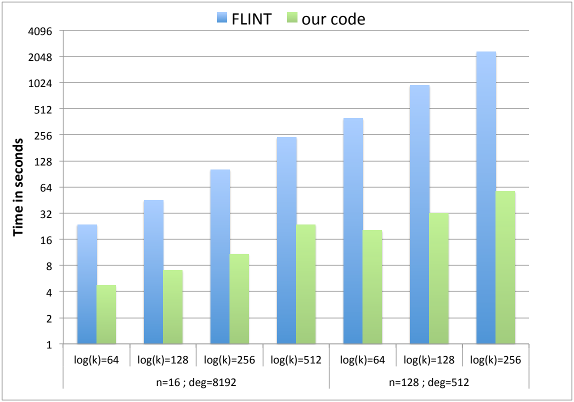
Remark:

if the result's degree and bitsize are not too large, CRT with Fourier primes might suffice.

# Matrix multiplication over $\mathbb{Z}[x]$

## Implementation

- use CRT with Fourier primes
- re-use multi-modular reduction/reconstruction with linear algebra
- re-use multiplication in $F_p[x]$




*benchmark on Intel Xeon-2620 @ 2.0GHz*

# Parallel Matrix multiplication over $\mathbb{Z}[x]$

**Very first attempt (work still in progress)**

- parallel CRT with linear algebra (same code as in $\mathbb{Z}$ case)
- perform each multiplication over $F_p[x]$ in parallel
- some part of the code still sequential

| n | d | $\log(k)$ | 6 cores | 12 cores | time seq |
|---:|---:|---:|---:|---:|---:|
| 64 | 1024 | 600 | $\times 3.52$ | $\times 4.88$ | 61.1s |
| 32 | 4096 | 600 | $\times 3.68$ | $\times 5.02$ | 64.4s |
| 32 | 2048 | 1024 | $\times 3.95$ | $\times 5.73$ | 54.5s |
| 128 | 128 | 1024 | $\times 3.76$ | $\times 5.55$ | 53.9s |

# Polynomial Matrix in LinBox (proposition)

## Generic handler class for Polynomial Matrix

```
template<size_t type, size_t storage, class Field>
class PolynomialMatrix;
```

## Specialization for different memory strategy

```
// Matrix of polynomials
template<class _Field>
class PolynomialMatrix<PMType::polfirst,PMStorage::plain,_Field>;

// Polynomial of matrices
template<class _Field>
class PolynomialMatrix<PMType::matfirst,PMStorage::plain,_Field>;

// Polynomial of matrices (partial view on monomials)
template<class _Field>
class PolynomialMatrix<PMType::matfirst,PMStorage::view,_Field>;
```

# Conclusion

**High performance tools for exact linear algebra :**

- matrix multiplication through floating points
- multi-dimensional CRT
- FFT for polynomial over wordsize prime fields
- adaptative matrix representation

We provide in the LinBox library (`www.linalg.org`)

- efficient sequential/parallel matrix multiplication over $\mathbb{Z}$
- efficient sequential matrix multiplication over $\mathrm{F}_p[x]$ and $\mathbb{Z}[x]$