Recent results on fast multiplication

David Harvey

MACAO Collaborative Workshop, University of Wollongong, 26th November 2019

University of New South Wales
Joint work with Joris van der Hoeven (École Polytechnique, Palaiseau)

Two multiplication problems

Integer multiplication

$$M(n) := \text{cost of multiplying integers with at most } n \text{ digits}$$

- “digits” means in some fixed base (e.g. binary or decimal).
- “cost” means “bit complexity”
  (e.g. # steps on multi-tape Turing machine, or # gates in Boolean circuit).

Integer multiplication

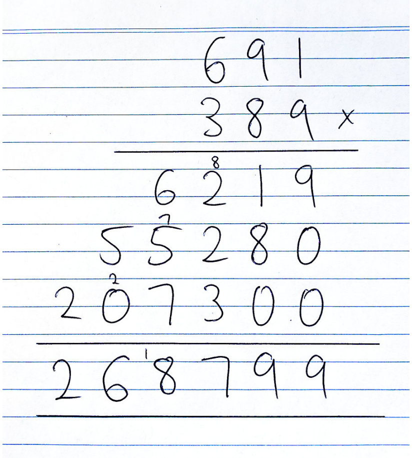
$$M(n) := \text{cost of multiplying integers with at most } n \text{ digits}$$

- "digits" means in some fixed base (e.g. binary or decimal).
- "cost" means "bit complexity"
  (e.g. # steps on multi-tape Turing machine, or # gates in Boolean circuit).

Polynomial multiplication over finite fields

$$M_q(n) := \text{cost of multiplying polynomials in } \mathbb{F}_q[x] \text{ of degree at most } n$$

- $\mathbb{F}_q$ = field with q elements, q a fixed prime power.
- "cost" means bit complexity, or # ring operations in $\mathbb{F}_q$.



Goes back at least to ancient Egypt — probably much older.

Complexity is $M(n) = O(n^2)$.

Same algorithm for polynomials: $M_q(n) = O(n^2)$.

Conjecture (Kolmogorov, around 1956)

$$M(n) = \Theta(n^2).$$

Conjecture (Kolmogorov, around 1956)

$$M(n) = \Theta(n^2).$$

According to Karatsuba (1995),

> *"Probably, [the conjecture's] appearance is based on the fact that throughout the history of mankind people have been using [the algorithm] whose complexity is $O(n^2)$, and if a more economical method existed, it would have already been found."*

1962	Karatsuba	$n^{\log 3/\log 2}$ ($\approx n^{1.58}$)
1969	Knuth	$n\,2^{\sqrt{2\log n/\log 2}}\log n$
1971	Schönhage–Strassen	$n \log n \log\log n$
2007	Fürer	$n \log n\, K^{\log^* n}$ for some $K > 1$
2019	H.–van der Hoeven†	$n \log n$

Brief history of bounds for $M(n)$.

†: not yet published.

1962	Karatsuba	$n^{\log 3/\log 2}$ $(\approx n^{1.58})$
1969	Knuth	$n\, 2^{\sqrt{2\log n/\log 2}} \log n$
1971	Schönhage–Strassen	$n \log n \log\log n$
2007	Fürer	$n \log n\, K^{\log^* n}$ for some $K > 1$
2019	H.–van der Hoeven†	$n \log n$

Brief history of bounds for $M(n)$.
†: not yet published.

Conjecture (Schönhage–Strassen, 1971)

$$M(n) = \Theta(n \log n).$$

Faster multiplication

1977	Schönhage	$n \log n \log \log n$
2017	H.–van der Hoeven–Lecerf	$n \log n \, 8^{\log^* n}$
2019	H.–van der Hoeven	$n \log n \, 4^{\log^* n}$
2019	H.–van der Hoeven†	$n \log n$

Brief history of bounds for $\mathsf{M}_q(n)$.

†: not yet published; depends on unproved number-theoretic hypothesis.

1977	Schönhage	$n \log n \log \log n$
2017	H.–van der Hoeven–Lecerf	$n \log n \, 8^{\log^* n}$
2019	H.–van der Hoeven	$n \log n \, 4^{\log^* n}$
2019	H.–van der Hoeven†	$n \log n$

Brief history of bounds for $\mathsf{M}_q(n)$.
†: not yet published; depends on unproved number-theoretic hypothesis.

Unsolved problem

Can we get $\mathsf{M}_q(n) = O(n \log n)$ unconditionally?

Expected answer: **yes**, because the unproved hypothesis is extremely plausible.

1. Complex DFTs and FFTs
2. Reductions between integer and polynomial multiplication
3. Multidimensional DFTs
4. Conditional $O(n \log n)$ multiplication for integers and polynomials
5. Unconditional $O(n \log n)$ integer multiplication

Complex DFTs and FFTs

The discrete Fourier transform (DFT)

Let $n \geqslant 1$ and $\zeta := e^{2\pi i/n} \in \mathbb{C}$. The roots of $x^n - 1$ are $1, \zeta, \ldots, \zeta^{n-1}$.

The **DFT of length *n*** over $\mathbb{C}$ is the linear map (in fact ring isomorphism)

$$\mathbb{C}[x]/(x^n - 1) \longrightarrow \mathbb{C}^n, \qquad F \longmapsto (F(1), F(\zeta), \ldots, F(\zeta^{n-1})).$$

The discrete Fourier transform (DFT)

Let $n \geqslant 1$ and $\zeta := e^{2\pi i/n} \in \mathbb{C}$. The roots of $x^n - 1$ are $1, \zeta, \ldots, \zeta^{n-1}$.

The **DFT of length *n*** over $\mathbb{C}$ is the linear map (in fact ring isomorphism)

$$\mathbb{C}[x]/(x^n - 1) \longrightarrow \mathbb{C}^n, \qquad F \longmapsto (F(1), F(\zeta), \ldots, F(\zeta^{n-1})).$$

Example for $n = 4$

$$\text{The DFT of} \quad F_0 + F_1 x + F_2 x^2 + F_3 x^3 \quad \text{is} \quad \begin{pmatrix} F_0 + F_1 + F_2 + F_3 \\ F_0 + iF_1 - F_2 - iF_3 \\ F_0 - F_1 + F_2 - F_3 \\ F_0 - iF_1 - F_2 + iF_3 \end{pmatrix} \in \mathbb{C}^4.$$

The naive algorithm to evaluate the DFT requires $O(n^2)$ operations in $\mathbb{C}$.

The discrete Fourier transform (DFT)

DFTs can be used to compute **cyclic convolutions**, i.e. multiply in $\mathbb{C}[x]/(x^n - 1)$.

Given as input $F, G \in \mathbb{C}[x]/(x^n - 1)$:

1. use DFT to compute $a_j := F(\zeta^j)$ and $b_j := G(\zeta^j)$ for $j = 0, \ldots, n - 1$
2. compute pointwise products $c_j := a_j \cdot b_j$
3. use inverse DFT to find $H \in \mathbb{C}[x]/(x^n - 1)$ such that $H(\zeta^j) = c_j$ for all j

Output is $H = FG \pmod{x^n - 1}$.

The fast Fourier transform (FFT)

The simplest case of the **Cooley–Tukey FFT** (1965) reduces the complexity of the DFT from $O(n^2)$ to $O(n \log n)$ operations in the case $n = 2^k$.

Example for $n = 8$

$$
\begin{tikzcd}[row sep=small]
F \bmod x^8 - 1 \arrow[r] \arrow[ddddr] & F \bmod x^4 - 1 \arrow[r] \arrow[ddr] & F \bmod x^2 - 1 \arrow[r] \arrow[dr] & F \bmod x - 1 = F(\zeta^0) \\
 & & & F \bmod x + 1 = F(\zeta^4) \\
 & & F \bmod x^2 + 1 \arrow[r] \arrow[dr] & F \bmod x - \zeta^2 = F(\zeta^2) \\
 & & & F \bmod x + \zeta^2 = F(\zeta^6) \\
 & F \bmod x^4 + 1 \arrow[r] \arrow[ddr] & F \bmod x^2 - \zeta^2 \arrow[r] \arrow[dr] & F \bmod x - \zeta = F(\zeta^1) \\
 & & & F \bmod x + \zeta = F(\zeta^5) \\
 & & F \bmod x^2 + \zeta^2 \arrow[r] \arrow[dr] & F \bmod x - \zeta^3 = F(\zeta^3) \\
 & & & F \bmod x + \zeta^3 = F(\zeta^7)
\end{tikzcd}
$$

More generally, the Cooley–Tukey algorithm reduces a transform of any length n to DFTs whose lengths are the prime factors of n.

How do we handle a DFT whose length is a **large prime**?

More generally, the Cooley–Tukey algorithm reduces a transform of any length n to DFTs whose lengths are the prime factors of n.

How do we handle a DFT whose length is a **large prime**?

Rader's algorithm (1968)

A DFT of prime length n may be reduced to a cyclic convolution of length $n - 1$, together with $O(n)$ additions in $\mathbb{C}$.

The convolution of length $n - 1$ may be evaluated by various methods (e.g. FFTs).

Example: DFT of length 5.

Given $a_0, \ldots, a_4 \in \mathbb{C}$, want to compute

$$
\begin{aligned}
&a_0 + a_1 \quad + a_2 \quad + a_3 \quad + a_4 \\
&a_0 + a_1\zeta^1 + a_2\zeta^2 + a_3\zeta^3 + a_4\zeta^4 \\
&a_0 + a_1\zeta^2 + a_2\zeta^4 + a_3\zeta^1 + a_4\zeta^3 \\
&a_0 + a_1\zeta^3 + a_2\zeta^1 + a_3\zeta^4 + a_4\zeta^2 \\
&a_0 + a_1\zeta^4 + a_2\zeta^3 + a_3\zeta^2 + a_4\zeta^1,
\end{aligned}
$$

where $\zeta = e^{2\pi i/5}$.

Example: DFT of length 5.

Given $a_0, \ldots, a_4 \in \mathbb{C}$, want to compute

$$\begin{aligned}
&a_0 + a_1 \quad\; + a_2 \quad\; + a_3 \quad\; + a_4 \\
&a_0 + a_1\zeta^1 + a_2\zeta^2 + a_3\zeta^3 + a_4\zeta^4 \\
&a_0 + a_1\zeta^2 + a_2\zeta^4 + a_3\zeta^1 + a_4\zeta^3 \\
&a_0 + a_1\zeta^3 + a_2\zeta^1 + a_3\zeta^4 + a_4\zeta^2 \\
&a_0 + a_1\zeta^4 + a_2\zeta^3 + a_3\zeta^2 + a_4\zeta^1,
\end{aligned}$$

where $\zeta = e^{2\pi i/5}$.

Apart from a few additions, this is equivalent to computing

$$\begin{aligned}
&a_1\zeta^1 + a_2\zeta^2 + a_4\zeta^4 + a_3\zeta^3 \\
&a_1\zeta^2 + a_2\zeta^4 + a_4\zeta^3 + a_3\zeta^1 \\
&a_1\zeta^4 + a_2\zeta^3 + a_4\zeta^1 + a_3\zeta^2 \\
&a_1\zeta^3 + a_2\zeta^1 + a_4\zeta^2 + a_3\zeta^4.
\end{aligned}$$

Example: DFT of length 5.

Given $a_0, \ldots, a_4 \in \mathbb{C}$, want to compute

$$
\begin{aligned}
&a_0 + a_1 \quad + a_2 \quad + a_3 \quad + a_4 \\
&a_0 + a_1\zeta^1 + a_2\zeta^2 + a_3\zeta^3 + a_4\zeta^4 \\
&a_0 + a_1\zeta^2 + a_2\zeta^4 + a_3\zeta^1 + a_4\zeta^3 \\
&a_0 + a_1\zeta^3 + a_2\zeta^1 + a_3\zeta^4 + a_4\zeta^2 \\
&a_0 + a_1\zeta^4 + a_2\zeta^3 + a_3\zeta^2 + a_4\zeta^1,
\end{aligned}
$$

where $\zeta = e^{2\pi i/5}$.

Apart from a few additions, this is equivalent to computing

$$
\begin{aligned}
&a_1\zeta^1 + a_2\zeta^2 + a_4\zeta^4 + a_3\zeta^3 \\
&a_1\zeta^2 + a_2\zeta^4 + a_4\zeta^3 + a_3\zeta^1 \\
&a_1\zeta^4 + a_2\zeta^3 + a_4\zeta^1 + a_3\zeta^2 \\
&a_1\zeta^3 + a_2\zeta^1 + a_4\zeta^2 + a_3\zeta^4.
\end{aligned}
$$

This in turn is equivalent to computing the length-4 cyclic convolution of

$$(a_1, a_2, a_4, a_3) \quad \text{and} \quad (\zeta^3, \zeta^4, \zeta^2, \zeta^1).$$

Reductions between integer and polynomial multiplication

Can reduce integer to polynomial multiplication using **Kronecker segmentation**.

Example: suppose we want the product of $u = 314159265$ and $v = 271828182$.

Can reduce integer to polynomial multiplication using **Kronecker segmentation**.

Example: suppose we want the product of $u = 314159265$ and $v = 271828182$.

Step 1. Rewrite u and v in base 10^3, encode them as polynomials

$$F(x) = 314x^2 + 159x + 265, \qquad G(x) = 271x^2 + 828x + 182,$$

i.e., so that $F(10^3) = u$ and $G(10^3) = v$.

Reducing integer multiplication to polynomial multiplication

Can reduce integer to polynomial multiplication using **Kronecker segmentation**.

Example: suppose we want the product of $u = 314159265$ and $v = 271828182$.

Step 1. Rewrite u and v in base 10^3, encode them as polynomials

$$F(x) = 314x^2 + 159x + 265, \qquad G(x) = 271x^2 + 828x + 182,$$

i.e., so that $F(10^3) = u$ and $G(10^3) = v$.

Step 2. Compute the polynomial product

$$H(x) = F(x)G(x) = 85094x^4 + 303081x^3 + 260615x^2 + 248358x + 48230.$$

Can reduce integer to polynomial multiplication using **Kronecker segmentation**.

Example: suppose we want the product of $u = 314159265$ and $v = 271828182$.

Step 1. Rewrite u and v in base 10^3, encode them as polynomials

$$F(x) = 314x^2 + 159x + 265, \qquad G(x) = 271x^2 + 828x + 182,$$

i.e., so that $F(10^3) = u$ and $G(10^3) = v$.

Step 2. Compute the polynomial product

$$H(x) = F(x)G(x) = 85094x^4 + 303081x^3 + 260615x^2 + 248358x + 48230.$$

Step 3. Substitute $x = 10^3$ to get $uv = H(10^3) = 85397341863406230$.

Reducing integer multiplication to polynomial multiplication

Application: the **first Schönhage–Strassen algorithm** (1971).

To multiply n-bit integers:

1. Rewrite integers in base 2^b where $b \approx \log n$. Encode as polynomials $F, G \in \mathbb{Z}[x]$, coefficient size b bits, degree around $n/\log n$.
2. Multiply polynomials in $\mathbb{C}[x]$ using complex FFTs, with working precision $O(\log n)$ bits. Round result to get correct product in $\mathbb{Z}[x]$.
3. Substitute $x = 2^b$ to get product in $\mathbb{Z}$.

Application: the **first Schönhage–Strassen algorithm** (1971).

To multiply n-bit integers:

1. Rewrite integers in base 2^b where $b \approx \log n$. Encode as polynomials $F, G \in \mathbb{Z}[x]$, coefficient size b bits, degree around $n/\log n$.
2. Multiply polynomials in $\mathbb{C}[x]$ using complex FFTs, with working precision $O(\log n)$ bits. Round result to get correct product in $\mathbb{Z}[x]$.
3. Substitute $x = 2^b$ to get product in $\mathbb{Z}$.

Complexity analysis:

$$\mathsf{M}(n) = O\Big(\tfrac{n}{\log n} \log\big(\tfrac{n}{\log n}\big)\, \mathsf{M}(\log n)\Big) = O(n\,\mathsf{M}(\log n))$$

Reducing integer multiplication to polynomial multiplication

Application: the **first Schönhage–Strassen algorithm** (1971).

To multiply n-bit integers:

1. Rewrite integers in base 2^b where $b \approx \log n$. Encode as polynomials $F, G \in \mathbb{Z}[x]$, coefficient size b bits, degree around $n/\log n$.
2. Multiply polynomials in $\mathbb{C}[x]$ using complex FFTs, with working precision $O(\log n)$ bits. Round result to get correct product in $\mathbb{Z}[x]$.
3. Substitute $x = 2^b$ to get product in $\mathbb{Z}$.

Complexity analysis:

$$\begin{aligned} \mathsf{M}(n) &= O\left(\tfrac{n}{\log n} \log\left(\tfrac{n}{\log n}\right) \mathsf{M}(\log n)\right) = O(n\, \mathsf{M}(\log n)) \\ &= O(n \log n\, \mathsf{M}(\log\log n)) = O(n \log n \log\log n\, \mathsf{M}(\log\log\log n)) = \cdots \end{aligned}$$

Reducing polynomial multiplication to integer multiplication

Can reduce polynomial to integer multiplication using **Kronecker substitution**.

Example: suppose we want to multiply

$$F(x) = 314x^2 + 159x + 265, \qquad G(x) = 271x^2 + 828x + 182.$$

Reducing polynomial multiplication to integer multiplication

Can reduce polynomial to integer multiplication using **Kronecker substitution**.

Example: suppose we want to multiply

$$F(x) = 314x^2 + 159x + 265, \qquad G(x) = 271x^2 + 828x + 182.$$

Step 1. Encode them as integers:

$$u = F(10^7) = 314000015900000265, \qquad v = G(10^7) = 271000082800000182.$$

Can reduce polynomial to integer multiplication using **Kronecker substitution**.

Example: suppose we want to multiply

$$F(x) = 314x^2 + 159x + 265, \qquad G(x) = 271x^2 + 828x + 182.$$

Step 1. Encode them as integers:

$$u = F(10^7) = 314000015900000265, \qquad v = G(10^7) = 271000082800000182.$$

Step 2. Compute the integer product

$$uv = 85094030308102606150248358004823 0.$$

Can reduce polynomial to integer multiplication using **Kronecker substitution**.

Example: suppose we want to multiply

$$F(x) = 314x^2 + 159x + 265, \qquad G(x) = 271x^2 + 828x + 182.$$

Step 1. Encode them as integers:

$$u = F(10^7) = 314000015900000265, \qquad v = G(10^7) = 271000082800000182.$$

Step 2. Compute the integer product

$$uv = 85094030308102606150248358004823 0.$$

Step 3. Read off the desired polynomial product

$$F(x)G(x) = 85094x^4 + 303081x^3 + 260615x^2 + 248358x + 48230.$$

Application: **multiplication in $\mathbb{F}_p[x]$.**

1. Lift polynomials to $\mathbb{Z}[x]$.
2. Multiply in $\mathbb{Z}[x]$ using Kronecker substitution (i.e. via *integer* multiplication).
3. Reduce result modulo p to obtain product in $\mathbb{F}_p[x]$.

Application: **multiplication in $\mathbb{F}_p[x]$.**

1. Lift polynomials to $\mathbb{Z}[x]$.
2. Multiply in $\mathbb{Z}[x]$ using Kronecker substitution (i.e. via *integer* multiplication).
3. Reduce result modulo p to obtain product in $\mathbb{F}_p[x]$.

This method is efficient provided p is not too small compared to n.

Unfortunately for fixed p this method is **inefficient** due to “zero-padding”:

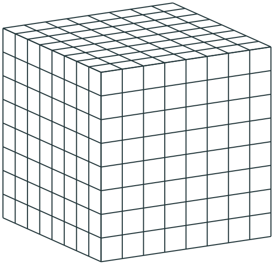
$$\mathsf{M}_p(n) = \mathsf{M}(O(n \log n)) = O(n \log^2 n).$$

Multidimensional DFTs

Example: let $F \in \mathbb{C}[x, y, z]/(x^8 - 1, y^8 - 1, z^8 - 1)$.

We may represent F with $8^3 = 512$ coefficients:

$$F = \sum_{j=0}^{7} \sum_{k=0}^{7} \sum_{l=0}^{7} F_{j,k,l} x^j y^k z^l.$$



Suppose we want to evaluate

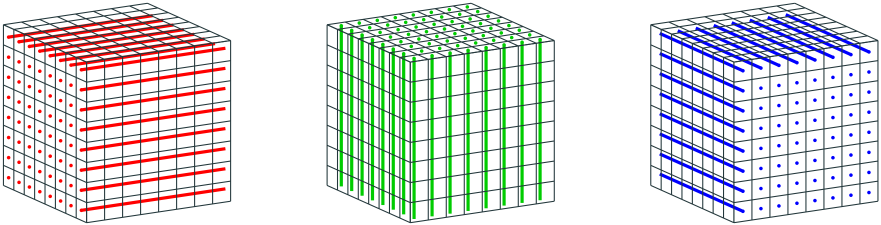
$$F(\zeta^j, \zeta^k, \zeta^l) \qquad \text{for } j, k, l = 0, \ldots, 7$$

where $\zeta = e^{2\pi i/8}$.

This is a 3-dimensional DFT of size $8 \times 8 \times 8$.

Multidimensional DFTs

Standard method for d-dimensional DFT: evaluate in each variable separately.



For a transform of size $n_1 \times \cdots \times n_d$, total cost (operations in $\mathbb{C}$) is

$$\frac{n}{n_1} O(n_1 \log n_1) + \cdots + \frac{n}{n_d} O(n_d \log n_d) = O(n \log n), \qquad n := n_1 \cdots n_d.$$

Nussbaumer's algorithm (late 1970s) instead does the following.

Suppose the input is a polynomial

$$F \in \mathbb{C}[x, y, z]/(x^8 - 1, y^8 - 1, z^8 - 1).$$

Nussbaumer's method for multidimensional DFTs

Nussbaumer's algorithm (late 1970s) instead does the following.

Suppose the input is a polynomial

$$F \in \mathbb{C}[x, y, z]/(x^8 - 1, y^8 - 1, z^8 - 1).$$

First reduce modulo $z^4 - 1$ and $z^4 + 1$ just like the first step of the usual FFT:

$$F \bmod z^4 - 1 \in \mathbb{C}[x, y, z]/(x^8 - 1, y^8 - 1, z^4 - 1),$$
$$F \bmod z^4 + 1 \in \mathbb{C}[x, y, z]/(x^8 - 1, y^8 - 1, z^4 + 1).$$

We may handle the first problem recursively (in general: split in half along the "longest" dimension).

Let's concentrate on the second problem.

So now our problem is to evaluate a polynomial

$$G \in \mathbb{C}[x, y, z]/(x^8 - 1, y^8 - 1, z^4 + 1)$$

at the roots of $x^8 - 1$, $y^8 - 1$, $z^4 + 1$.

So now our problem is to evaluate a polynomial

$$G \in \mathbb{C}[x, y, z]/(x^8 - 1, y^8 - 1, z^4 + 1)$$

at the roots of $x^8 - 1$, $y^8 - 1$, $z^4 + 1$.

Key observation

The roots of $z^4 + 1$ are exactly the primitive 8-th roots of unity,
so z itself behaves like a primitive 8-th root of unity in $\mathbb{C}[z]/(z^4 + 1)$.

We may evaluate x and y **at the powers of z** (instead of at the powers of ζ).

In other words, we evaluate

$$G(z^j, z^k, z) \in \mathbb{C}[z]/(z^4 + 1), \qquad j, k = 0, \ldots, 7.$$

We may evaluate x and y **at the powers of z** (instead of at the powers of ζ).

In other words, we evaluate

$$G(z^j, z^k, z) \in \mathbb{C}[z]/(z^4 + 1), \qquad j, k = 0, \ldots, 7.$$

We do this with the usual Cooley–Tukey FFT algorithm, but whenever we would usually multiply by some ζ^s, we multiply by z^s instead!

Multiplying by z^s is easy: just involves moving coefficients around.

We may evaluate x and y **at the powers of z** (instead of at the powers of ζ).

In other words, we evaluate

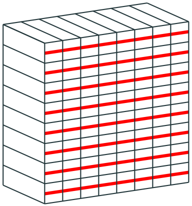
$$G(z^j, z^k, z) \in \mathbb{C}[z]/(z^4 + 1), \qquad j, k = 0, \ldots, 7.$$

We do this with the usual Cooley–Tukey FFT algorithm, but whenever we would usually multiply by some ζ^s, we multiply by z^s instead!

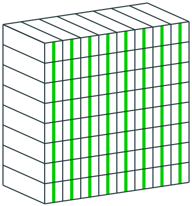
Multiplying by z^s is easy: just involves moving coefficients around.

Finally, for each j and k, use the usual complex FFT to evaluate $G(z^j, z^k, z)$ at the genuine complex roots of $z^4 + 1$ (powers of ζ).

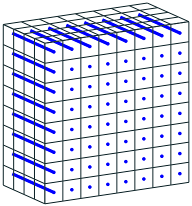
Nussbaumer's method for multidimensional DFTs



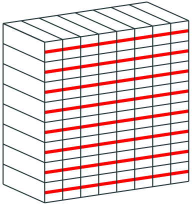
evaluate x at z^j: easy
(no multiplications)



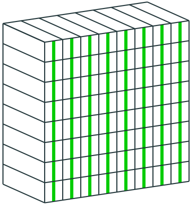
evaluate y at z^k: easy
(no multiplications)



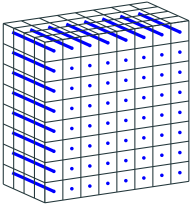
evaluate z at ζ^l: harder
(some multiplications)



evaluate x at z^j: easy
(no multiplications)



evaluate y at z^k: easy
(no multiplications)



evaluate z at ζ^l: harder
(some multiplications)

Analysis for d-dimensional DFT **(assuming all n_i powers of two)**:

- $O(n \log n)$ additions in $\mathbb{C}$ (just like standard algorithm), but
- only $O\left(\dfrac{n \log n}{d}\right)$ multiplications in $\mathbb{C}$ **(save a factor of d)**.

Conditional $O(n \log n)$ multiplication for integers and polynomials

I will illustrate for integers with $n = 10^{14}$ bits (around 11 TB).

I will illustrate for integers with $n = 10^{14}$ bits (around 11 TB).

Step 1. Cut integers into chunks of 46 ($\approx \log_2 10^{14}$) bits.

Encode into polynomials in $\mathbb{Z}[t]$, with 46-bit coefficients, and degree less than

$$\lceil n/46 \rceil = 2\,173\,913\,043\,479.$$

I will illustrate for integers with $n = 10^{14}$ bits (around 11 TB).

Step 1. Cut integers into chunks of 46 ($\approx \log_2 10^{14}$) bits.

Encode into polynomials in $\mathbb{Z}[t]$, with 46-bit coefficients, and degree less than

$$\lceil n/46 \rceil = 2\,173\,913\,043\,479.$$

It suffices to multiply the polynomials in the ring $\mathbb{Z}[t]/(t^N - 1)$ where

$$N = 5\,509\,236\,183\,041 = p_1 p_2 p_3, \qquad p_1 = 15361,\ p_2 = 18433,\ p_3 = 19457.$$

This suffices to recover the product in $\mathbb{Z}[t]$ because $N > 2 \times 2\,173\,913\,043\,479$.

Step 2. Using CRT, there is an isomorphism (Agarwal–Cooley 1977):

$$\mathbb{Z}[t]/(t^{5\,509\,236\,183\,041} - 1) \cong \mathbb{Z}[x, y, z]/(x^{15361} - 1, y^{18433} - 1, z^{19457} - 1),$$
$$t \longmapsto xyz.$$

Can be computed efficiently in either direction (just rearrange coefficients).

So we have reduced to a 3-dimensional cyclic convolution of size $p_1 \times p_2 \times p_3$.

Step 3. To multiply in

$$\mathbb{Z}[x,y,z]/(x^{15361}-1, y^{18433}-1, z^{19457}-1),$$

we use the same strategy as the Schönhage–Strassen algorithm:

1. Compute (multidimensional) DFTs of both polynomials over $\mathbb{C}$.
2. Multiply pointwise in $\mathbb{C}$.
3. Perform inverse DFT to get approximate product in

$$\mathbb{C}[x,y,z]/(x^{15361}-1, y^{18433}-1, z^{19457}-1).$$

4. Round resulting coefficients to nearest integer.
   (Working precision throughout is a small multiple of 46 bits.)

How do we efficiently perform a DFT of size $15361 \times 18433 \times 19457$?

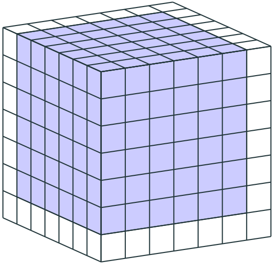
Nussbaumer's trick doesn't work directly, because the p_i are not powers of two.

How do we efficiently perform a DFT of size $15361 \times 18433 \times 19457$?

Nussbaumer's trick doesn't work directly, because the p_i are not powers of two.

Step 4. Using multidimensional variant of Rader's trick, reduce to multiplication in

$$\mathbb{C}[x, y, z]/(x^{15360} - 1, y^{18432} - 1, z^{19456} - 1).$$



Key observation

Convolution lengths are reduced from p_i to $p_i - 1$.

I picked the primes very carefully: notice that

$$
\begin{aligned}
p_1 - 1 &= 15360 = 15 \times 2^{10}, \\
p_2 - 1 &= 18432 = 18 \times 2^{10}, \\
p_3 - 1 &= 19456 = 19 \times 2^{10}.
\end{aligned}
$$

I picked the primes very carefully: notice that

$$p_1 - 1 = 15360 = 15 \times 2^{10},$$
$$p_2 - 1 = 18432 = 18 \times 2^{10},$$
$$p_3 - 1 = 19456 = 19 \times 2^{10}.$$

Step 5. Reduce to:

- "nice" DFTs of size $2^{10} \times 2^{10} \times 2^{10}$ (use Nussbaumer), and
- "annoying" DFTs of size $15 \times 18 \times 19$.

What happens for general n? We get

- "nice" DFTs of size
$$2^k \times \cdots \times 2^k.$$
Using Nussbaumer, the first $d-1$ of the dimensions cost $O(n \log n)$. May take $d \approx 10^6$ (independently of n) to control the cost of the last dimension.

What happens for general n? We get

- "nice" DFTs of size

  $$2^k \times \cdots \times 2^k.$$

  Using Nussbaumer, the first $d-1$ of the dimensions cost $O(n \log n)$. May take $d \approx 10^6$ (independently of n) to control the cost of the last dimension.
- "annoying" DFTs of size

  $$\frac{p_1 - 1}{2^k} \times \cdots \times \frac{p_d - 1}{2^k}.$$

  This is where the complexity analysis becomes conditional.

To make the "annoying" DFTs cheap enough, we need to prove existence of small primes in the arithmetic progression $p = 1 \pmod{2^k}$.

To make the “annoying” DFTs cheap enough, we need to prove existence of small primes in the arithmetic progression $p = 1 \pmod{2^k}$.

Linnik's theorem (1944)

There exists a constant $L > 1$ such that for any relatively prime integers a and $m \gg 0$, there exists a prime $p = a \pmod{m}$ with $p < m^L$.

A **Linnik constant** is a value of L for which the above statement holds.

Best published Linnik constant is currently $L = 5.18$ (Xylouris, 2011).

Linnik's theorem is **embarrassingly weak**!

Example: consider $p = 1 \pmod{2^{10}}$. The first few primes are

$$p = 12289,\ 13313,\ 15361,\ 18433,\ 19457,\ 25601,\ 37889,\ 39937, \ldots.$$

But Linnik's theorem (with the best known L) only guarantees that

$$p < (2^{10})^{5.18} \approx 4 \times 10^{15}.$$

Linnik's theorem is **embarrassingly weak**!

Example: consider $p = 1 \pmod{2^{10}}$. The first few primes are

$$p = 12289,\ 13313,\ 15361,\ 18433,\ 19457,\ 25601,\ 37889,\ 39937, \ldots.$$

But Linnik's theorem (with the best known L) only guarantees that

$$p < (2^{10})^{5.18} \approx 4 \times 10^{15}.$$

Under GRH, can prove that any $L > 2$ is a Linnik constant (Heath-Brown 1992).

This is still hopeless: we get

$$p < (2^{10})^2 \approx 10^6.$$

Widely-believed conjecture

Any $L > 1$ is a Linnik constant.

Widely-believed conjecture

Any $L > 1$ is a Linnik constant.

Theorem (H.–van der Hoeven 2019)

If there exists a Linnik constant $L < 1 + \frac{1}{303}$, then the cost of the “annoying” DFTs can be controlled, and the algorithm sketched in this talk achieves

$$M(n) = O(n \log n).$$

We can probably weaken the bound for L a bit, but we have no idea how to get anywhere near $L = 2$.

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

1. Choose small primes $p_1, \ldots, p_d = 1 \pmod{2^k}$ for suitable k

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

1. Choose small primes $p_1, \ldots, p_d = 1 \pmod{2^k}$ for suitable k
2. Construct extension $\mathbb{F}_{q^s}/\mathbb{F}_q$ containing p_i-th and $(p_i - 1)$-th roots of 1

Conditional $O(n \log n)$ multiplication in $\mathbb{F}_q[x]$

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

1. Choose small primes $p_1, \ldots, p_d = 1 \pmod{2^k}$ for suitable k
2. Construct extension $\mathbb{F}_{q^s}/\mathbb{F}_q$ containing p_i-th and $(p_i - 1)$-th roots of 1
3. Reduce to multiplication in $\mathbb{F}_{q^s}[x]$ (i.e. cut into chunks of size s)

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

1. Choose small primes $p_1, \ldots, p_d = 1 \pmod{2^k}$ for suitable k
2. Construct extension $\mathbb{F}_{q^s}/\mathbb{F}_q$ containing p_i-th and $(p_i - 1)$-th roots of 1
3. Reduce to multiplication in $\mathbb{F}_{q^s}[x]$ (i.e. cut into chunks of size s)
4. Reduce to multiplication in $\mathbb{F}_{q^s}[x_1, \ldots, x_d]/(x_1^{p_1} - 1, \ldots, x_d^{p_d} - 1)$

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

1. Choose small primes $p_1, \ldots, p_d = 1 \pmod{2^k}$ for suitable k
2. Construct extension $\mathbb{F}_{q^s}/\mathbb{F}_q$ containing p_i-th and $(p_i - 1)$-th roots of 1
3. Reduce to multiplication in $\mathbb{F}_{q^s}[x]$ (i.e. cut into chunks of size s)
4. Reduce to multiplication in $\mathbb{F}_{q^s}[x_1, \ldots, x_d]/(x_1^{p_1} - 1, \ldots, x_d^{p_d} - 1)$
5. Reduce to DFTs of size $p_1 \times \cdots \times p_d$ over $\mathbb{F}_{q^s}$

Conditional $O(n \log n)$ multiplication in $\mathbb{F}_q[x]$

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

1. Choose small primes $p_1, \ldots, p_d = 1 \pmod{2^k}$ for suitable k
2. Construct extension $\mathbb{F}_{q^s}/\mathbb{F}_q$ containing p_i-th and $(p_i - 1)$-th roots of 1
3. Reduce to multiplication in $\mathbb{F}_{q^s}[x]$ (i.e. cut into chunks of size s)
4. Reduce to multiplication in $\mathbb{F}_{q^s}[x_1, \ldots, x_d]/(x_1^{p_1} - 1, \ldots, x_d^{p_d} - 1)$
5. Reduce to DFTs of size $p_1 \times \cdots \times p_d$ over $\mathbb{F}_{q^s}$
6. Reduce to multiplication in $\mathbb{F}_{q^s}[x_1, \ldots, x_d]/(x_1^{p_1-1} - 1, \ldots, x_d^{p_d-1} - 1)$ (Rader)

Conditional $O(n \log n)$ multiplication in $\mathbb{F}_q[x]$

A similar idea works for multiplying in $\mathbb{F}_q[x]$, with various additional technicalities (especially in characteristic 2):

1. Choose small primes $p_1, \ldots, p_d = 1 \pmod{2^k}$ for suitable k
2. Construct extension $\mathbb{F}_{q^s}/\mathbb{F}_q$ containing p_i-th and $(p_i - 1)$-th roots of 1
3. Reduce to multiplication in $\mathbb{F}_{q^s}[x]$ (i.e. cut into chunks of size s)
4. Reduce to multiplication in $\mathbb{F}_{q^s}[x_1, \ldots, x_d]/(x_1^{p_1} - 1, \ldots, x_d^{p_d} - 1)$
5. Reduce to DFTs of size $p_1 \times \cdots \times p_d$ over $\mathbb{F}_{q^s}$
6. Reduce to multiplication in $\mathbb{F}_{q^s}[x_1, \ldots, x_d]/(x_1^{p_1-1} - 1, \ldots, x_d^{p_d-1} - 1)$ (Rader)
7. Use Nussbaumer to do synthetic FFTs in $d - 1$ dimensions, etc etc.

Conditional $O(n \log n)$ multiplication in $\mathbb{F}_q[x]$

Theorem (H.–van der Hoeven 2019)

If there exists a Linnik constant $L < 1 + 2^{-1162}$, then

$$M_q(n) = O(n \log n).$$

Can probably improve 2^{-1162}, but we don't know by how much.

Unconditional $O(n \log n)$ integer multiplication

Again take $n = 10^{14}$ bits.

As before, reduce to multiplying polynomials of degree 2 173 913 043 479 with 46-bit coefficients.

Again take $n = 10^{14}$ bits.

As before, reduce to multiplying polynomials of degree 2 173 913 043 479 with 46-bit coefficients.

This time we choose primes

$$p_1 = 16381, \qquad p_2 = 16369, \qquad p_3 = 16363.$$

Notice they are all **just below $2^{14} = 16384$**.

(Easy to find such primes. No arithmetic progressions involved.)

It suffices to multiply in $\mathbb{Z}[t]/(t^N - 1)$ where

$$N = p_1 p_2 p_3 = 4\,387\,584\,457\,807 > 2 \times 2\,173\,913\,043\,479.$$

It suffices to multiply in $\mathbb{Z}[t]/(t^N - 1)$ where

$$N = p_1 p_2 p_3 = 4\,387\,584\,457\,807 > 2 \times 2\,173\,913\,043\,479.$$

As before, reduce to complex DFTs of size $16381 \times 16369 \times 16363$.

Unconditional $O(n \log n)$ integer multiplication

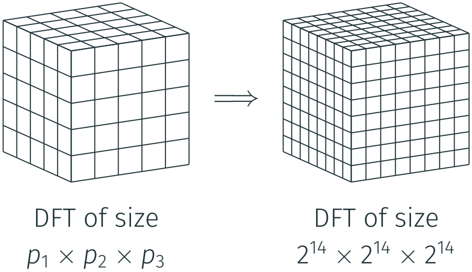
It suffices to multiply in $\mathbb{Z}[t]/(t^N - 1)$ where

$$N = p_1 p_2 p_3 = 4\,387\,584\,457\,807 > 2 \times 2\,173\,913\,043\,479.$$

As before, reduce to complex DFTs of size $16381 \times 16369 \times 16363$.

But instead of using Rader's algorithm, we use a new technique called **Gaussian resampling** to directly reduce to a DFT of size $2^{14} \times 2^{14} \times 2^{14}$.

Then we win by using Nussbaumer's method to evaluate this last DFT.



Example: given input $u \in \mathbb{C}^{13}$, suppose we want to compute DFT $\hat{u} \in \mathbb{C}^{13}$.

Suppose however that we only know how to compute DFTs of length 16.

Example: given input $u \in \mathbb{C}^{13}$, suppose we want to compute DFT $\hat{u} \in \mathbb{C}^{13}$.

Suppose however that we only know how to compute DFTs of length 16.

We will convert length 13 to length 16 via a certain **resampling map**

$$S\colon \mathbb{C}^{13} \to \mathbb{C}^{16}.$$

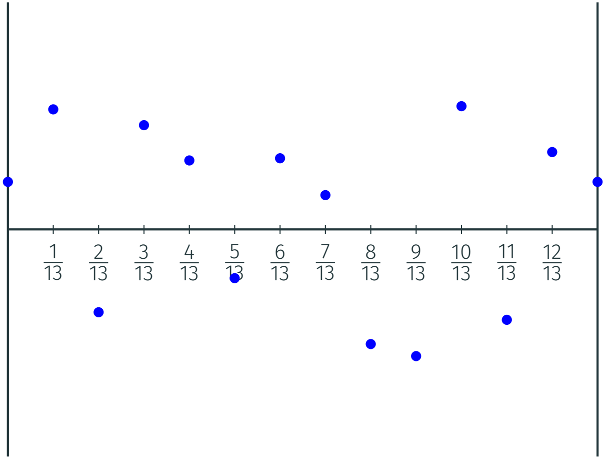
I will show how to construct S over the next few slides.

The diagram shows a typical input vector $u \in \mathbb{C}^{13}$.

For simplicity we assume $u_i \in \mathbb{R}$.

The blue points are $(\frac{i}{13}, u_i)$ for $i = 0, \ldots, 12$.

Notice the x-axis wraps around from left to right (i.e., the x-values live in $\mathbb{R}/\mathbb{Z}$).



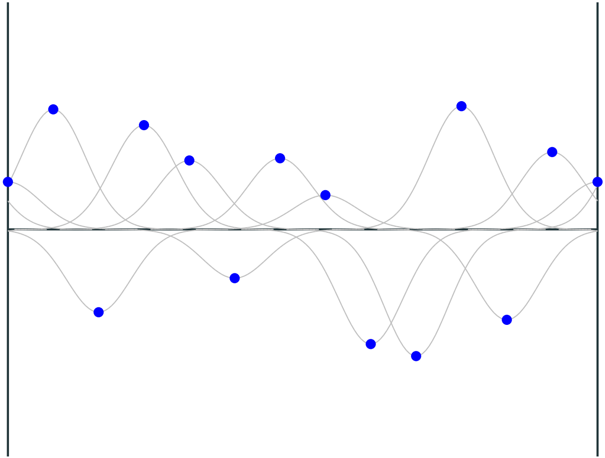
The resampling map

Draw a Gaussian curve centred around each data point.

The equation for the i-th point is

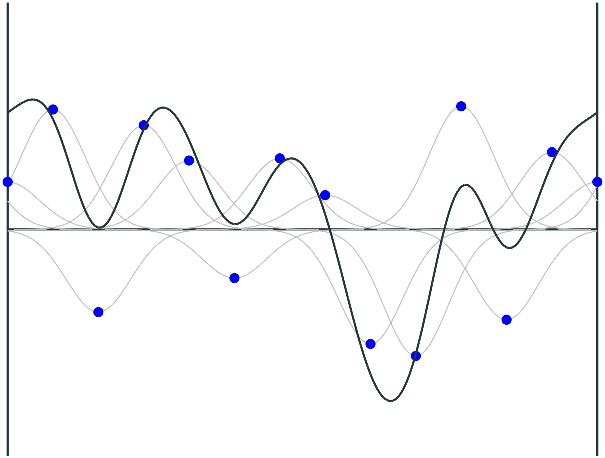
$$y = u_i e^{-13^2 (x - \frac{i}{13})^2}.$$

The "height" of the curve is u_i and the "width" is $\frac{1}{13}$.



Add up all the Gaussians to get a nice smooth 1-periodic curve:

$$f(x) = \sum_{i=0}^{12} u_i e^{-13^2(x-\frac{i}{13})^2}.$$

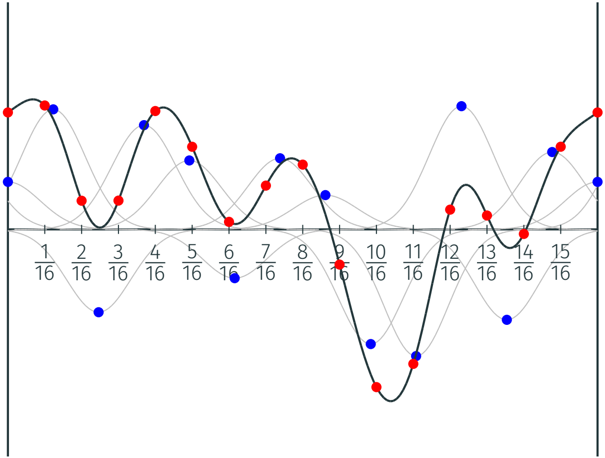


Add up all the Gaussians to get a nice smooth 1-periodic curve:

$$f(x) = \sum_{i=0}^{12} u_i e^{-13^2 (x - \frac{i}{13})^2}.$$

The resampled vector $v = S(u)$ is defined by evaluating $f(x)$ at 16 equally-spaced points:

$$v_j = f(\tfrac{j}{16}), \qquad j = 0, \ldots, 15.$$



$$
\begin{bmatrix}
1.000 & 0.368 & 0.018 & & & & & & & & & 0.018 & 0.368 \\
0.517 & 0.965 & 0.244 & 0.008 & & & & & & & & & 0.037 \\
0.071 & 0.677 & 0.869 & 0.151 & 0.004 & & & & & & & & 0.001 \\
0.003 & 0.127 & 0.826 & 0.729 & 0.087 & 0.001 & & & & & & & \\
 & 0.006 & 0.210 & 0.939 & 0.570 & 0.047 & 0.001 & & & & & & \\
 & & 0.014 & 0.323 & 0.996 & 0.415 & 0.023 & & & & & & \\
 & & & 0.030 & 0.465 & 0.984 & 0.282 & 0.011 & & & & & \\
 & & & 0.001 & 0.058 & 0.623 & 0.907 & 0.179 & 0.005 & & & & \\
 & & & & 0.002 & 0.105 & 0.779 & 0.779 & 0.105 & 0.002 & & & \\
 & & & & & 0.005 & 0.179 & 0.907 & 0.623 & 0.058 & 0.001 & & \\
 & & & & & & 0.011 & 0.282 & 0.984 & 0.465 & 0.030 & & \\
 & & & & & & & 0.023 & 0.415 & 0.996 & 0.323 & 0.014 & \\
 & & & & & & & 0.001 & 0.047 & 0.570 & 0.939 & 0.210 & 0.006 \\
0.003 & & & & & & & & 0.001 & 0.087 & 0.729 & 0.826 & 0.127 \\
0.071 & 0.001 & & & & & & & & 0.004 & 0.151 & 0.869 & 0.677 \\
0.517 & 0.037 & & & & & & & & & 0.008 & 0.244 & 0.965
\end{bmatrix}
$$

Matrix of resampling map $S\colon \mathbb{C}^{13} \to \mathbb{C}^{16}$.

Each “output” coordinate depends mainly on the nearby “input” coordinates.

Fun fact #1. The Fourier transform of a Gaussian is again a Gaussian. This leads to a commutative diagram

$$\begin{tikzcd}
u \arrow[r, "\text{DFT of length } s"] \arrow[d, "\substack{\text{Resample}\\ S\colon \mathbb{C}^s \to \mathbb{C}^t}"'] & \hat{u} \arrow[d, "\substack{\text{Resample}\\ \hat{S}\colon \mathbb{C}^s \to \mathbb{C}^t}"] \\
v \arrow[r, "\text{DFT of length } t"] & \hat{v}
\end{tikzcd}$$

In our example, $s = 13$ and $t = 16$.

The map $\hat{S}$ is defined almost exactly the same way as S; it differs by some straightforward scaling factors and data reindexing.

Fun fact #2. Due to the rapid decay of the Gaussians, the resampling map can be evaluated efficiently.

If the target transform length is t, the cost is

$$O(t\sqrt{\log t})$$

operations in $\mathbb{C}$ (assuming working precision $O(\log t)$ bits).

This is **asymptotically negligble** compared to $O(t \log t)$ cost of the FFT.

Fun fact #3. The resampling map is injective.

This follows more or less from the "diagonal" structure of the matrix of S.

Moreover, there is a **deconvolution algorithm** that recovers u from $v = S(u)$ using

$$O(t\sqrt{\log t})$$

operations in $\mathbb{C}$.

(Note: we do not actually prove this in the paper. For technical reasons we do something a bit different.)

Conclusion: we can compute the map $u \mapsto \hat{u}$ (a DFT of length 13) by traversing the diagram as follows:

$$
\begin{tikzcd}
u \arrow[r, dashed, "\text{DFT of length 13}"] \arrow[d, "\begin{array}{c}\text{Resample}\\ \mathbb{C}^{13} \to \mathbb{C}^{16}\end{array}"'] & \hat{u} \\
v \arrow[r, "\text{DFT of length 16}"] & \hat{v} \arrow[u, "\begin{array}{c}\text{Deconvolve}\\ \mathbb{C}^{16} \to \mathbb{C}^{13}\end{array}"']
\end{tikzcd}
$$

The cost of the vertical arrows is asymptotically negligible.

Combining a multidimensional version of Gaussian resampling with everything else from before, we get:

Theorem (H.–van der Hoeven 2019)

$$M(n) = O(n \log n).$$

Final words

Combining a multidimensional version of Gaussian resampling with everything else from before, we get:

Theorem (H.–van der Hoeven 2019)

$$M(n) = O(n \log n).$$

Unsolved problem

Can we get $M_q(n) = O(n \log n)$ unconditionally?

Unfortunately, Gaussian resampling does not seem to work over $\mathbb{F}_q$.

Is there some other way of “changing the transform length” over $\mathbb{F}_q$???

Thank you!