Integer multiplication in time $O(n \log n)$

David Harvey

University of Newcastle, CARMA Colloquium
23rd May 2019

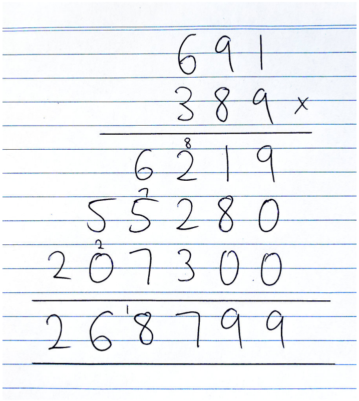
University of New South Wales
Joint work with Joris van der Hoeven (École Polytechnique, Palaiseau)

The basic problem

$\mathsf{M}(n) := $ time needed to multiply n-bit integers.

Complexity model: any reasonable notion of counting “bit operations”, e.g. multitape Turing machines or Boolean circuits.

Long multiplication



```
     6 9 1
     3 8 9  x
   ---------
    6 2 1 9
  5 5 2 8 0
2 0 7 3 0 0
-----------
2 6 8 7 9 9
-----------
```

Long multiplication: $M(n) = O(n^2)$.

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

Kolmogorov's lower bound conjecture

Conjecture (Kolmogorov, around 1956)

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

In other words: long multiplication is asymptotically optimal.

Progress in the 1960s

Luckily for us, Kolmogorov was wrong:

1962	Karatsuba	$n^{\log 3/\log 2}$ ($\approx n^{1.58}$)
1963	Toom	$n\,2^{5\sqrt{\log n/\log 2}}$
1966	Schönhage	$n\,2^{\sqrt{2\log n/\log 2}}(\log n)^{3/2}$
1969	Knuth	$n\,2^{\sqrt{2\log n/\log 2}}\log n$

Knuth's bound is $O(n^{1+\epsilon})$ for any $\epsilon > 0$, but not $O(n(\log n)^C)$ for any $C > 0$.

Schönhage and Strassen

1968: Cooley–Tukey publish the Fast Fourier transform (FFT).

Shortly afterwards:

Theorem (Schönhage–Strassen, 1971)

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

Schönhage and Strassen

1968: Cooley–Tukey publish the Fast Fourier transform (FFT).

Shortly afterwards:

Theorem (Schönhage–Strassen, 1971)

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

They also proposed their own conjecture:

Conjecture (Schönhage–Strassen, 1971)

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

Fürer's algorithm

Nothing happened for 36 years!

Then:

Theorem (Fürer, 2007)

$$M(n) = O(n \log n \, K^{\log^* n})$$

for some (unspecified) constant $K > 1$.

Here $\log^*$ is the iterated logarithm:

$$\log^* x = \begin{cases} 0 & x \leqslant 1, \\ \log^*(\log x) + 1 & x > 1. \end{cases}$$

Incremental progress over the last five years

		value of K
2014	H–vdH–Lecerf	8
2014	H–vdH–Lecerf	4 (†)
2015	Covanov–Thomé	4 (†)
2016	H	6
2016	H–vdH	4 (†)
2017	H–vdH	$4\sqrt{2} \approx 5.66$
2018	H–vdH	4

(†) indicates conditional algorithms (depend on various conjectures about prime numbers of a special form)

New result

Theorem (H. & van der Hoeven, 2019)

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

New result

Theorem (H. & van der Hoeven, 2019)

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

If the Schönhage–Strassen conjecture is correct, then the new algorithm is asymptotically optimal…

New result

Theorem (H. & van der Hoeven, 2019)

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

If the Schönhage–Strassen conjecture is correct, then the new algorithm is asymptotically optimal…

… but we shouldn't forget what happened to Kolmogorov's conjecture!

WARNING

The paper has not yet been officially peer-reviewed.

WARNING

The paper has not yet been officially peer-reviewed.

Prof. Joshua Cooper (University of South Carolina), quoted in New Scientist (29th March):

WARNING

The paper has not yet been officially peer-reviewed.

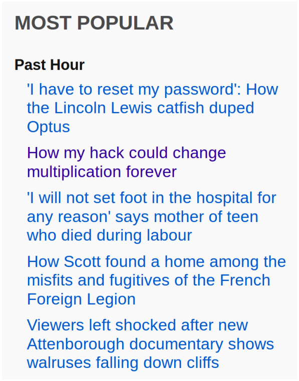
Prof. Joshua Cooper (University of South Carolina), quoted in New Scientist (29th March):

> *"It passes the smell test."*

Media circus: personal highlights

(1) when my article in The Conversation was briefly the 2nd most popular story on ABC News

(but I didn't write the headline)



MOST POPULAR

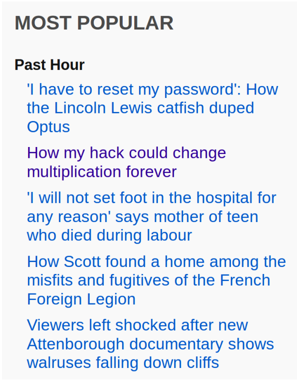
Past Hour

- 'I have to reset my password': How the Lincoln Lewis catfish duped Optus
- How my hack could change multiplication forever
- 'I will not set foot in the hospital for any reason' says mother of teen who died during labour
- How Scott found a home among the misfits and fugitives of the French Foreign Legion
- Viewers left shocked after new Attenborough documentary shows walruses falling down cliffs

Media circus: personal highlights

(1) when my article in The Conversation was briefly the 2nd most popular story on ABC News

(but I didn't write the headline)


MOST POPULAR

Past Hour

- 'I have to reset my password': How the Lincoln Lewis catfish duped Optus
- How my hack could change multiplication forever
- 'I will not set foot in the hospital for any reason' says mother of teen who died during labour
- How Scott found a home among the misfits and fugitives of the French Foreign Legion
- Viewers left shocked after new Attenborough documentary shows walruses falling down cliffs

(2) discussing logarithms (log-a-rhythms?) live on Triple J

Plan for the rest of the talk

1. DFTs and FFTs
2. The first Schönhage–Strassen algorithm
3. Multidimensional DFTs and Nussbaumer’s trick
4. The new algorithm: a first attempt
5. Gaussian resampling to the rescue
6. Final comments

DFTs and FFTs

The discrete Fourier transform (DFT)

Let $t \geqslant 1$, and set $\zeta := e^{2\pi i/t} \in \mathbb{C}$.

The *discrete Fourier transform* (DFT) of a vector

$$u = (u_0, \ldots, u_{t-1}) \in \mathbb{C}^t$$

is the vector

$$\hat{u} = (\hat{u}_0, \ldots, \hat{u}_{t-1}) \in \mathbb{C}^t$$

defined by

$$\hat{u}_k := \sum_{j=0}^{t-1} u_j \zeta^{jk}.$$

The discrete Fourier transform (DFT)

Example: take $t = 4$. Then

$$\begin{aligned}
\hat{u}_0 &= u_0 + \ u_1 + u_2 + \ u_3, \\
\hat{u}_1 &= u_0 + iu_1 - u_2 - iu_3, \\
\hat{u}_2 &= u_0 - \ u_1 + u_2 - \ u_3, \\
\hat{u}_3 &= u_0 - iu_1 - u_2 + iu_3.
\end{aligned}$$

The discrete Fourier transform (DFT)

Equivalently, the DFT evaluates a polynomial at the t-th roots of unity. If $f(x) = u_0 + u_1 x + \cdots + u_{t-1} x^{t-1}$, then

$$\hat{u} = (f(1), f(\zeta), \ldots, f(\zeta^{t-1})).$$

The discrete Fourier transform (DFT)

Equivalently, the DFT evaluates a polynomial at the t-th roots of unity. If $f(x) = u_0 + u_1 x + \cdots + u_{t-1}x^{t-1}$, then

$$\hat{u} = (f(1), f(\zeta), \ldots, f(\zeta^{t-1})).$$

You should think of $f(x)$ as living in the quotient ring

$$\mathbb{C}[x]/(x^t - 1).$$

The DFT explicitly evaluates the isomorphism

$$\mathbb{C}[x]/(x^t - 1) \longrightarrow \bigoplus_{k=0}^{t-1} \mathbb{C}[x]/(x - \zeta^k) \cong \mathbb{C}^t.$$

The fast Fourier transform (FFT)

Evaluating the DFT in the naive way requires $O(t^2)$ operations (additions, multiplications) in $\mathbb{C}$.

The *fast Fourier transform* (FFT) does it in $O(t \log t)$ operations.

The fast Fourier transform (FFT)

Evaluating the DFT in the naive way requires $O(t^2)$ operations (additions, multiplications) in $\mathbb{C}$.

The *fast Fourier transform* (FFT) does it in $O(t \log t)$ operations.

I don't have time to explain the FFT in detail.

Here is a nice picture of a butterfly instead.

The fast Fourier transform (FFT)

The FFT can be used to quickly multiply polynomials.

Given polynomials $f, g \in \mathbb{C}[x]$ of degree $< t$:

1. Use FFT to evaluate f and g at the t-th roots of unity.
2. Multiply their values at these points.
3. Use inverse FFT to interpolate at the t-th roots of unity.

Output is the product $f(x)g(x)$ modulo $x^t - 1$, i.e., a cyclic convolution of length t.

Total cost: $O(t \log t)$ operations in $\mathbb{C}$.

The first Schönhage–Strassen algorithm

The first Schönhage–Strassen algorithm

Schönhage and Strassen (1971) described two different integer multiplication algorithms.

The most famous is the second one, which achieved

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

The first one has slightly worse complexity. I will explain it in this section, as a warm-up for the new algorithm.

The first Schönhage–Strassen algorithm

We are given as input two n-bit integers u and v.

STEP #1: write $u = f(2^b)$ and $v = g(2^b)$ where $f, g \in \mathbb{Z}[x]$, coefficients have $b \approx \log n$ bits.

The first Schönhage–Strassen algorithm

We are given as input two n-bit integers u and v.

STEP #1: write $u = f(2^b)$ and $v = g(2^b)$ where $f, g \in \mathbb{Z}[x]$, coefficients have $b \approx \log n$ bits.

Running example (in decimal):

$$u = 3141592653 \ldots [\text{omit } 999980 \text{ digits}] \ldots 0577945815,$$
$$v = 2718281828 \ldots [\text{omit } 999980 \text{ digits}] \ldots 4769422818.$$

The first Schönhage–Strassen algorithm

We are given as input two n-bit integers u and v.

STEP #1: write $u = f(2^b)$ and $v = g(2^b)$ where $f, g \in \mathbb{Z}[x]$, coefficients have $b \approx \log n$ bits.

Running example (in decimal):

$$u = 3141592653\ldots[\text{omit } 999980 \text{ digits}]\ldots 0577945815,$$
$$v = 2718281828\ldots[\text{omit } 999980 \text{ digits}]\ldots 4769422818.$$

Write them as $u = f(10^5)$ and $v = g(10^5)$ where

$$f(x) = 31415x^{199999} + 92653x^{199998} + \cdots + 45815,$$
$$g(x) = 27182x^{199999} + 81828x^{199998} + \cdots + 22818.$$

The first Schönhage–Strassen algorithm

STEP #2: multiply $f(x)$ and $g(x)$ in $\mathbb{C}[x]$ using FFTs over $\mathbb{C}$, and round the coefficients of the result to obtain product in $\mathbb{Z}[x]$.

Running example: the product $f(x)g(x)$ is

$$853922530x^{399998} + 5089120466x^{399997} + \cdots + 1045406670.$$

The first Schönhage–Strassen algorithm

STEP #2: multiply $f(x)$ and $g(x)$ in $\mathbb{C}[x]$ using FFTs over $\mathbb{C}$, and round the coefficients of the result to obtain product in $\mathbb{Z}[x]$.

Running example: the product $f(x)g(x)$ is

$$853922530x^{399998} + 5089120466x^{399997} + \cdots + 1045406670.$$

During FFT:

- need to maintain working precision of $O(\log n)$ bits, and

The first Schönhage–Strassen algorithm

STEP #2: multiply $f(x)$ and $g(x)$ in $\mathbb{C}[x]$ using FFTs over $\mathbb{C}$, and round the coefficients of the result to obtain product in $\mathbb{Z}[x]$.

Running example: the product $f(x)g(x)$ is

$$853922530x^{399998} + 5089120466x^{399997} + \cdots + 1045406670.$$

During FFT:

- need to maintain working precision of $O(\log n)$ bits, and
- coefficient arithmetic in $\mathbb{C}$ reduces to integer arithmetic, which is handled recursively!

The first Schönhage–Strassen algorithm

STEP #3: substitute $x = 2^b$ into $f(x)g(x)$ to obtain uv. This amounts to adding coefficients with suitable overlap.

Running example: plug $x = 10^5$ into

$$f(x)g(x) = 853922530x^{399998} + 5089120466x^{399997} + \cdots$$

to get

$$uv = 8539734222\ldots[\text{omit } 1999980 \text{ digits}]\ldots7628606670.$$

The first Schönhage–Strassen algorithm

Complexity analysis: we have reduced an n-bit multiplication problem to $O(n)$ multiplications of size $O(\log n)$.

So we get

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

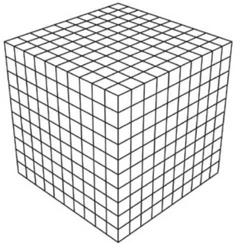
Multidimensional DFTs and Nussbaumer’s trick

Multidimensional DFTs

Let $t_1, \ldots, t_d \geqslant 1$ be integers.

I will write $\mathbb{C}^{t_1,\ldots,t_d}$ for the tensor product $\mathbb{C}^{t_1} \otimes_{\mathbb{C}} \cdots \otimes_{\mathbb{C}} \mathbb{C}^{t_d}$.

An element of $\mathbb{C}^{t_1,\ldots,t_d}$ is basically a d-dimensional array of complex numbers of size $t_1 \times \cdots \times t_d$.



Multidimensional DFTs

Let $u = (u_{j_1,\ldots,j_d}) \in \mathbb{C}^{t_1,\ldots,t_d}$ be such an array.

Set $\zeta_\ell := e^{2\pi i/t_\ell} \in \mathbb{C}$ for $\ell = 1, \ldots, d$.

The DFT of u is the array $\hat{u} \in \mathbb{C}^{t_1,\ldots,t_d}$ defined by

$$\hat{u}_{k_1,\ldots,k_d} := \sum_{j_1=0}^{t_1-1} \cdots \sum_{j_d=0}^{t_d-1} u_{j_1,\ldots,j_d} \zeta_1^{j_1 k_1} \cdots \zeta_d^{j_d k_d}.$$

Multidimensional DFTs

Multidimensional DFTs may be interpreted as evaluating *multivariate* polynomials at roots of unity.

If

$$f(x_1, \ldots, x_d) = \sum_{j_1,\ldots,j_d} u_{j_1,\ldots,j_d} x_1^{j_1} \cdots x_d^{j_d},$$

then

$$\hat{u}_{k_1,\ldots,k_d} = f(\zeta_1^{k_1}, \ldots, \zeta_d^{k_d}).$$

Multidimensional DFTs

Multidimensional DFTs may be interpreted as evaluating *multivariate* polynomials at roots of unity.

If

$$f(x_1, \ldots, x_d) = \sum_{j_1,\ldots,j_d} u_{j_1,\ldots,j_d} x_1^{j_1} \cdots x_d^{j_d},$$

then

$$\hat{u}_{k_1,\ldots,k_d} = f(\zeta_1^{k_1}, \ldots, \zeta_d^{k_d}).$$

The polynomial f naturally lives in

$$\mathbb{C}[x_1, \ldots, x_d]/(x_1^{t_1} - 1, \ldots, x_d^{t_d} - 1).$$

Multiplication in this ring corresponds to d-dimensional cyclic convolution.

Multidimensional FFTs

The simplest way to evaluate a DFT of size $t_1 \times \cdots \times t_d$ is to perform one-dimensional FFTs along each dimension.

For example, a DFT of size $t_1 \times t_2 \times t_3$ can be decomposed into

- $t_2 t_3$ transforms of length t_1 (evaluate x_1), plus
- $t_1 t_3$ transforms of length t_2 (evaluate x_2), plus
- $t_1 t_2$ transforms of length t_3 (evaluate x_3).

The total cost is

$$O(t_2 t_3(t_1 \log t_1) + t_1 t_3(t_2 \log t_2) + t_1 t_2(t_3 \log t_3)) = \\ O(t_1 t_2 t_3 \log(t_1 t_2 t_3)).$$

Multidimensional FFTs

In general, a transform of size $t_1 \times \cdots \times t_d$ can be performed in

$$O(t \log t)$$

operations in $\mathbb{C}$, where $t \coloneqq t_1 \cdots t_d$ is the total number of coefficients.

Multidimensional FFTs

In general, a transform of size $t_1 \times \cdots \times t_d$ can be performed in

$$O(t \log t)$$

operations in $\mathbb{C}$, where $t := t_1 \cdots t_d$ is the total number of coefficients.

Notice that this includes:

- $O(t \log t)$ additions/subtractions, and also
- $O(t \log t)$ multiplications.

Nussbaumer's trick

Nussbaumer made a crucial observation in the late 1970s (based on previous work of Schönhage and Strassen).

In the ring

$$\mathbb{C}[x_1, \ldots, x_d]/(x_1^{t_1} - 1, \ldots, x_d^{t_d} - 1),$$

the variable x_i behaves somewhat like a t_i-th root of unity.

Nussbaumer's trick

Nussbaumer made a crucial observation in the late 1970s (based on previous work of Schönhage and Strassen).

In the ring

$$\mathbb{C}[x_1, \ldots, x_d]/(x_1^{t_1} - 1, \ldots, x_d^{t_d} - 1),$$

the variable x_i behaves somewhat like a t_i-th root of unity.

Nussbaumer showed that in some cases you can use the "fake" root of unity x_i to speed up the transforms with respect to the *other* variables.

The point is that it's much easier to multiply by a power of x_i than by a genuine complex t_i-th root of unity!

Nussbaumer's trick

Important special case: *suppose $t_1, \ldots, t_d$ are all powers of two.*

Nussbaumer's trick

Important special case: *suppose $t_1, \ldots, t_d$ are all powers of two.*

Then we only need to perform genuine 1-dimensional complex FFTs for the "longest" of the d dimensions.

The transforms in the other $d - 1$ dimensions collapse entirely to a sequence of additions and subtractions in $\mathbb{C}$.

Nussbaumer's trick

Important special case: *suppose $t_1, \ldots, t_d$ are all powers of two.*

Then we only need to perform genuine 1-dimensional complex FFTs for the "longest" of the d dimensions.

The transforms in the other $d-1$ dimensions collapse entirely to a sequence of additions and subtractions in $\mathbb{C}$.

Altogether:

- still need to perform $O(t \log t)$ additions and subtractions,
- but only $O\left(\frac{t \log t}{d}\right)$ multiplications (assuming the t_i are all about the same size).

Nussbaumer's trick

Important special case: *suppose $t_1, \ldots, t_d$ are all powers of two.*

Then we only need to perform genuine 1-dimensional complex FFTs for the "longest" of the d dimensions.

The transforms in the other $d-1$ dimensions collapse entirely to a sequence of additions and subtractions in $\mathbb{C}$.

Altogether:

- still need to perform $O(t \log t)$ additions and subtractions,
- but only $O\left(\frac{t \log t}{d}\right)$ multiplications (assuming the t_i are all about the same size).

By taking d large enough, we can make an arbitrarily large fraction of the work into additions/subtractions.

The new algorithm: a first attempt

Reduction to multidimensional DFTs

We start just like Schönhage–Strassen: reduce the integer multiplication problem to polynomial multiplication in $\mathbb{Z}[x]$.

Reduction to multidimensional DFTs

We start just like Schönhage–Strassen: reduce the integer multiplication problem to polynomial multiplication in $\mathbb{Z}[x]$.

To take advantage of Nussbaumer’s trick, we then need to transport the problem to a multidimensional setting without “wasting” too much space.

Reduction to multidimensional DFTs

Choose a dimension parameter d and a bunch of distinct primes $s_1, \ldots, s_d$ so that

- the s_i are all roughly the same size,
- $s_1 \cdots s_d$ is bigger (but not too much bigger) than the degree of the product polynomial,
- and each s_i is slightly smaller than a power of two t_i.

Reduction to multidimensional DFTs

Choose a dimension parameter d and a bunch of distinct primes $s_1, \ldots, s_d$ so that

- the s_i are all roughly the same size,
- $s_1 \cdots s_d$ is bigger (but not too much bigger) than the degree of the product polynomial,
- and each s_i is slightly smaller than a power of two t_i.

Running example: the target degree was 400000.

Take $d = 3$ and $(s_1, s_2, s_3) = (59, 61, 127)$.

Notice that $s_1 s_2 s_3 = 457073 > 400000$.

The corresponding powers of two are $(t_1, t_2, t_3) = (64, 64, 128)$.

Reduction to multidimensional DFTs

It suffice to compute the polynomial product in

$$\mathbb{Z}[x]/(x^{s_1 \cdots s_d} - 1),$$

i.e., we may treat the problem as a cyclic convolution of length $s_1 \cdots s_d$.

In our running example this is just

$$\mathbb{Z}[x]/(x^{457073} - 1).$$

Reduction to multidimensional DFTs

Agarwal and Cooley pointed out (1977) that the Chinese remainder theorem induces an isomorphism

$$\mathbb{Z}[x]/(x^{s_1\cdots s_d}-1) \cong \mathbb{Z}[x_1,\ldots,x_d]/(x_1^{s_1}-1,\ldots,x_d^{s_d}-1).$$

In other words, a 1-dimensional cyclic convolution of length $s_1\cdots s_d$ is equivalent (after suitable data rearrangement) to a d-dimensional cyclic convolution of dimension $s_1\times\cdots\times s_d$.

Reduction to multidimensional DFTs

Agarwal and Cooley pointed out (1977) that the Chinese remainder theorem induces an isomorphism

$$\mathbb{Z}[x]/(x^{s_1 \cdots s_d} - 1) \cong \mathbb{Z}[x_1, \ldots, x_d]/(x_1^{s_1} - 1, \ldots, x_d^{s_d} - 1).$$

In other words, a 1-dimensional cyclic convolution of length $s_1 \cdots s_d$ is equivalent (after suitable data rearrangement) to a d-dimensional cyclic convolution of dimension $s_1 \times \cdots \times s_d$.

(Note: for this step it is crucial that the s_i are relatively prime!

A cyclic convolution of length 32 is definitely *not* equivalent to a 2-dimensional cyclic convolution of size 4×8!)

Reduction to multidimensional DFTs

To compute this d-dimensional convolution, we use the usual evaluate–multiply–interpolate strategy.

So we have reduced to the problem of computing a few d-dimensional DFTs of size $s_1 \times \cdots \times s_d$.

Reduction to multidimensional DFTs

To compute this d-dimensional convolution, we use the usual evaluate–multiply–interpolate strategy.

So we have reduced to the problem of computing a few d-dimensional DFTs of size $s_1 \times \cdots \times s_d$.

Running example: we want to compute a 3-dimensional convolution of size $59 \times 61 \times 127$, which reduces in turn to computing DFTs of size $59 \times 61 \times 127$.

Reduction to multidimensional DFTs

To compute this d-dimensional convolution, we use the usual evaluate–multiply–interpolate strategy.

So we have reduced to the problem of computing a few d-dimensional DFTs of size $s_1 \times \cdots \times s_d$.

Running example: we want to compute a 3-dimensional convolution of size $59 \times 61 \times 127$, which reduces in turn to computing DFTs of size $59 \times 61 \times 127$.

At this point we really want to apply Nussbaumer's trick... but we can't, because the lengths s_i are not powers of two!

Gaussian resampling to the rescue

Gaussian resampling in one dimension

Suppose that we want to compute a DFT of length s, where s is an "inconvenient" transform length (e.g., a prime number).

Our paper introduces a technique called *Gaussian resampling*, that reduces such a DFT to another DFT of length $t > s$.

We are free to choose t to be a "convenient" length, such as a power of two.

Gaussian resampling in one dimension

Suppose that we want to compute a DFT of length s, where s is an “inconvenient” transform length (e.g., a prime number).

Our paper introduces a technique called *Gaussian resampling*, that reduces such a DFT to another DFT of length $t > s$.

We are free to choose t to be a “convenient” length, such as a power of two.

Crucially, Gaussian resampling does not introduce any constant factor overhead, unlike alternatives such as Bluestein’s algorithm.

It may be viewed as a refinement of a special case of the Dutt–Rokhlin algorithm for non-equispaced FFTs (1993).

Gaussian resampling in one dimension

Given as input $u \in \mathbb{C}^s$, we want to compute $\hat{u} \in \mathbb{C}^s$ (i.e., DFT of length s).

Gaussian resampling in one dimension

Given as input $u \in \mathbb{C}^s$, we want to compute $\hat{u} \in \mathbb{C}^s$ (i.e., DFT of length s).

STEP #1: compute a "resampled" vector $v \in \mathbb{C}^t$:

$$v_k := \sum_{j=-\infty}^{\infty} e^{-\pi s^2 (\frac{j}{s} - \frac{k}{t})^2} u_{j \bmod s}, \qquad k = 0, 1, \dots, t-1.$$

Gaussian resampling in one dimension

Given as input $u \in \mathbb{C}^s$, we want to compute $\hat{u} \in \mathbb{C}^s$ (i.e., DFT of length s).

STEP #1: compute a “resampled” vector $v \in \mathbb{C}^t$:

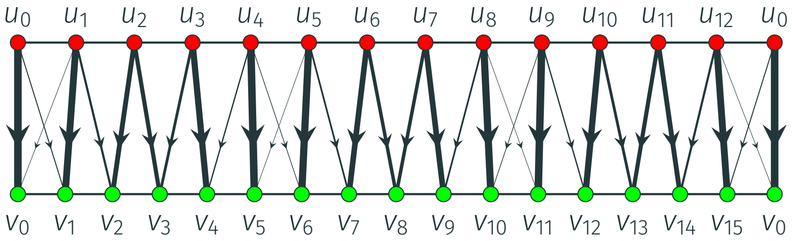
$$v_k := \sum_{j=-\infty}^{\infty} e^{-\pi s^2 (\frac{j}{s} - \frac{k}{t})^2} u_{j \bmod s}, \qquad k = 0, 1, \ldots, t-1.$$

Each v_k is a linear combination of the u_j’s. The weights follow a Gaussian law.

(Technical note: in the paper there is a parameter α controlling the width of the Gaussian. I will ignore it in this talk.)

Gaussian resampling in one dimension

Example for $s = 13$, $t = 16$ (thicker lines means larger weights):


u_0 u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 u_{10} u_{11} u_{12} u_0
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13} v_{14} v_{15} v_0

Due to the rapid Gaussian decay, each v_k depends mainly on the u_j's that are "nearby" in the circle.

This allows us to approximate v from u very efficiently.

Gaussian resampling in one dimension

It turns out that the resampling map $\mathbb{C}^s \to \mathbb{C}^t$ is injective (at least if the ratio t/s is not too close to 1).

In other words, the resampling process doesn't lose any information — it is possible to “deconvolve” v to recover u.

Gaussian resampling in one dimension

It turns out that the resampling map $\mathbb{C}^s \to \mathbb{C}^t$ is injective (at least if the ratio t/s is not too close to 1).

In other words, the resampling process doesn't lose any information — it is possible to "deconvolve" v to recover u.

Moreover, this deconvolution can be performed efficiently, again thanks to the rapid Gaussian decay.

Gaussian resampling in one dimension

STEP #2: compute DFT of $v \in \mathbb{C}^t$ to obtain $\hat{v} \in \mathbb{C}^t$.

This is a DFT of “convenient” length t, so presumably more efficient than original DFT of “inconvenient” length s.

Gaussian resampling in one dimension

Useful fact: the Fourier transform of a Gaussian is a Gaussian.

Gaussian resampling in one dimension

Useful fact: the Fourier transform of a Gaussian is a Gaussian.

This implies the following relation between $\hat{v}$ and $\hat{u}$:

$$\hat{v}_{-sk \bmod t} = \sum_{j=-\infty}^{\infty} e^{-\pi t^2 (\frac{k}{t} - \frac{j}{s})^2} \hat{u}_{tj \bmod s}.$$

In other words, after a suitable permutation of the coefficients, $\hat{v}$ is simply a "resampled" version of $\hat{u}$!

Gaussian resampling in one dimension

Useful fact: the Fourier transform of a Gaussian is a Gaussian.

This implies the following relation between $\hat{v}$ and $\hat{u}$:

$$\hat{v}_{-sk \bmod t} = \sum_{j=-\infty}^{\infty} e^{-\pi t^2 (\frac{k}{t} - \frac{j}{s})^2} \hat{u}_{tj \bmod s}.$$

In other words, after a suitable permutation of the coefficients, $\hat{v}$ is simply a “resampled” version of $\hat{u}$!

STEP #3: “deconvolve” $\hat{v}$ to obtain $\hat{u}$.

And we’re finished!

Gaussian resampling in one dimension

Expressed as a commutative diagram:

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

Back to integer multiplication

Gaussian resampling also works for multidimensional transforms: we just apply the 1-dimensional version in each dimension separately.

So a transform of “inconvenient” size $s_1 \times \cdots \times s_d$ can be reduced to a more convenient one of size $t_1 \times \cdots \times t_d$.

Back to integer multiplication

Gaussian resampling also works for multidimensional transforms: we just apply the 1-dimensional version in each dimension separately.

So a transform of "inconvenient" size $s_1 \times \cdots \times s_d$ can be reduced to a more convenient one of size $t_1 \times \cdots \times t_d$.

Running example: we reduce each DFT of size $59 \times 61 \times 127$ to a DFT of size $64 \times 64 \times 128$.

As the t_i are all powers of two, we can finally apply Nussbaumer's trick.

Sketch of complexity analysis

We get an $O(n \log n)$ contribution coming mainly from the additions/subtractions in the d-dimensional Nussbaumer FFTs (also covers cost of Gaussian resampling, etc).

The 1-dimensional transforms in the “long” dimension are handled recursively, by converting them to integer multiplication problems of size roughly $n^{1/d}$.

Sketch of complexity analysis

We get an $O(n \log n)$ contribution coming mainly from the additions/subtractions in the d-dimensional Nussbaumer FFTs (also covers cost of Gaussian resampling, etc).

The 1-dimensional transforms in the “long” dimension are handled recursively, by converting them to integer multiplication problems of size roughly $n^{1/d}$.

This leads to the recurrence inequality

$$\mathsf{M}(n) < 1728 \cdot \frac{n}{n'} \mathsf{M}(n') + O(n \log n), \qquad n' = n^{\frac{1}{d}+o(1)}.$$

Sketch of complexity analysis

We get an $O(n \log n)$ contribution coming mainly from the additions/subtractions in the d-dimensional Nussbaumer FFTs (also covers cost of Gaussian resampling, etc).

The 1-dimensional transforms in the “long” dimension are handled recursively, by converting them to integer multiplication problems of size roughly $n^{1/d}$.

This leads to the recurrence inequality

$$\mathsf{M}(n) < 1728 \cdot \frac{n}{n'} \mathsf{M}(n') + O(n \log n), \qquad n' = n^{\frac{1}{d}+o(1)}.$$

The factor 1728 is just a product of a bunch of small factors that show up at various stages of the analysis. Can probably be improved to $8 + o(1)$ with some work.

Sketch of complexity analysis

Now we take any fixed $d > 1728$ (for example $d = 1729$).

The recurrence then easily leads to $\mathsf{M}(n) = O(n \log n)$.

Sketch of complexity analysis

Now we take any fixed $d > 1728$ (for example $d = 1729$).

The recurrence then easily leads to $\mathsf{M}(n) = O(n \log n)$.

Another point of view: the total cost of the FFTs *decreases* by the constant factor $d/1728$ at each recursion level, so overall we get

$$n \log n + \frac{1728}{d} n \log n + \left(\frac{1728}{d}\right)^2 n \log n + \cdots = O(n \log n).$$

Sketch of complexity analysis

Now we take any fixed $d > 1728$ (for example $d = 1729$).

The recurrence then easily leads to $\mathsf{M}(n) = O(n \log n)$.

Another point of view: the total cost of the FFTs *decreases* by the constant factor $d/1728$ at each recursion level, so overall we get

$$n \log n + \frac{1728}{d} n \log n + \left(\frac{1728}{d}\right)^2 n \log n + \cdots = O(n \log n).$$

By contrast, in all previous integer multiplication algorithms, the total FFT cost *increases* (or stays constant) at each level.

Final comments

For what n do we win?

The argument in our paper only kicks in for

$$n \geqslant 2^{1729^{12}} \approx 10^{2148570911044552519406350450594173419 52}.$$

This can probably be improved (a lot).

For what n do we win?

The argument in our paper only kicks in for

$$n \geqslant 2^{1729^{12}} \approx 10^{214857091104455251940635045059417341952}.$$

This can probably be improved (a lot).

But actually this says nothing about the threshold for when our algorithm starts to beat existing algorithms.

The threshold could be much bigger than this, or much smaller.

We have no idea.

Advertisement for another paper

Two main results in *Polynomial multiplication over finite fields in time $O(n \log n)$*:

- An $O(n \log n)$ algorithm for multiplying polynomials of degree n in $\mathbb{F}_q[x]$.
- Another $O(n \log n)$ integer multiplication algorithm, which uses Rader's algorithm instead of Gaussian resampling.

Advertisement for another paper

Two main results in *Polynomial multiplication over finite fields in time $O(n \log n)$*:

- An $O(n \log n)$ algorithm for multiplying polynomials of degree n in $\mathbb{F}_q[x]$.
- Another $O(n \log n)$ integer multiplication algorithm, which uses Rader's algorithm instead of Gaussian resampling.

Unfortunately both algorithms depend on a conjecture about primes in arithmetic progressions (essentially that Linnik's theorem holds for an exponent L that is very close to 1).

Advertisement for another paper

Two main results in *Polynomial multiplication over finite fields in time* $O(n \log n)$:

- An $O(n \log n)$ algorithm for multiplying polynomials of degree n in $\mathbb{F}_q[x]$.
- Another $O(n \log n)$ integer multiplication algorithm, which uses Rader's algorithm instead of Gaussian resampling.

Unfortunately both algorithms depend on a conjecture about primes in arithmetic progressions (essentially that Linnik's theorem holds for an exponent L that is very close to 1).

Open question

Is there an unconditional $O(n \log n)$ multiplication algorithm for $\mathbb{F}_q[x]$?

Practical applications

Fredrik Johansson (INRIA Bordeaux) wrote on Reddit:

Practical applications

Fredrik Johansson (INRIA Bordeaux) wrote on Reddit:

“The result is of extreme practical importance.

Practical applications

Fredrik Johansson (INRIA Bordeaux) wrote on Reddit:

“The result is of extreme practical importance.

Not for actually multiplying integers (as usual with these algorithms it’s probably not faster than existing algorithms for integers that can be stored in the observable universe),

Practical applications

Fredrik Johansson (INRIA Bordeaux) wrote on Reddit:

“The result is of extreme practical importance.

Not for actually multiplying integers (as usual with these algorithms it’s probably not faster than existing algorithms for integers that can be stored in the observable universe), but for writing papers.

Practical applications

Fredrik Johansson (INRIA Bordeaux) wrote on Reddit:

“The result is of extreme practical importance.

Not for actually multiplying integers (as usual with these algorithms it’s probably not faster than existing algorithms for integers that can be stored in the observable universe), but for writing papers.

It has always been a hassle to write down the complexity of integer multiplication or algorithms based on integer multiplication by introducing soft-O notation, little-o exponents, epsilons greater than 0, or iterated logarithms.

Practical applications

Fredrik Johansson (INRIA Bordeaux) wrote on Reddit:

"The result is of extreme practical importance.

Not for actually multiplying integers (as usual with these algorithms it's probably not faster than existing algorithms for integers that can be stored in the observable universe), but for writing papers.

It has always been a hassle to write down the complexity of integer multiplication or algorithms based on integer multiplication by introducing soft-O notation, little-o exponents, epsilons greater than 0, or iterated logarithms.

From now on I can just write O(n log n) in my papers and be done with it!"

An unconvinced reader at The Conversation

An unconvinced reader at The Conversation

“Interesting article, looks good but subject to faith based belief.

An unconvinced reader at The Conversation

“Interesting article, looks good but subject to faith based belief.

We are told there are quicker methods to multiply but no explanation is given whatsoever as to how they work.

An unconvinced reader at The Conversation

“Interesting article, looks good but subject to faith based belief.

We are told there are quicker methods to multiply but no explanation is given whatsoever as to how they work.

Result is I do not feel informed or have had my knowledge advanced. I can’t really endorse their claims because there’s nothing to support them.

An unconvinced reader at The Conversation

“Interesting article, looks good but subject to faith based belief.

We are told there are quicker methods to multiply but no explanation is given whatsoever as to how they work.

Result is I do not feel informed or have had my knowledge advanced. I can’t really endorse their claims because there’s nothing to support them.

Just like a climate change paper, it could be true but then again it may not be.

An unconvinced reader at The Conversation

“Interesting article, looks good but subject to faith based belief.

We are told there are quicker methods to multiply but no explanation is given whatsoever as to how they work.

Result is I do not feel informed or have had my knowledge advanced. I can’t really endorse their claims because there’s nothing to support them.

Just like a climate change paper, it could be true but then again it may not be.

Just because it says it’s a mathematical paper —

An unconvinced reader at The Conversation

“Interesting article, looks good but subject to faith based belief.

We are told there are quicker methods to multiply but no explanation is given whatsoever as to how they work.

Result is I do not feel informed or have had my knowledge advanced. I can’t really endorse their claims because there’s nothing to support them.

Just like a climate change paper, it could be true but then again it may not be.

Just because it says it’s a mathematical paper —
well it may or may not be.”

Thank you!