Explain the algorithm | Marcus Östling
DD2467 Individual Project in TCS | June 1, 2022

1 Algorithm

A lot of the text is taken from https://www.youtube.com/watch?v=FKGRc867j10

1. Reduce the integer multiplication problem to polynomial muliplication in $\mathbb{Z}[x]$.
2. Reduce to mutlidimensional DFTs.
3. Gaussian resampling to obtain "convenient" transform lengths. Dimensions of powers of 2's
4. Apply Nussbaumer's trick to calculate DFT
5. Run algorithm recursively for large multiplication in the DFT.
6. Multiply point-wise
7. inverse

1.1 reduction to mutlidimensional DFTs

1. Choose a dimension parameter d and a bunch of distinct primes $s_1, ..., d_d$ so that:
    - the s_i are all rouhly the same size
    - $s_1 \cdots s_d$ is bigger (but not too much bigger) than the degree of the product polynomial. (Volume of hypercube about same size as length of the product)
    - and each s_i is slightly smaller than a power of two t_i.
    - Compute the polynomial product in

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

(cyclic convolution of length $s_1 \cdots s_d$)

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

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

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!)

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 $s_1 \times \cdots \times s_d$.

1.2 Gaussian resampling

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.

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

1. Compute a "resampled" vector $v \in \mathbb{C}^t$

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

   Each v_k is a linear combination of the u_j's. The weight follow a Gaussian law.

   (Technical note in the paper there is a parameter α controlling the width of the Gaussian)

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

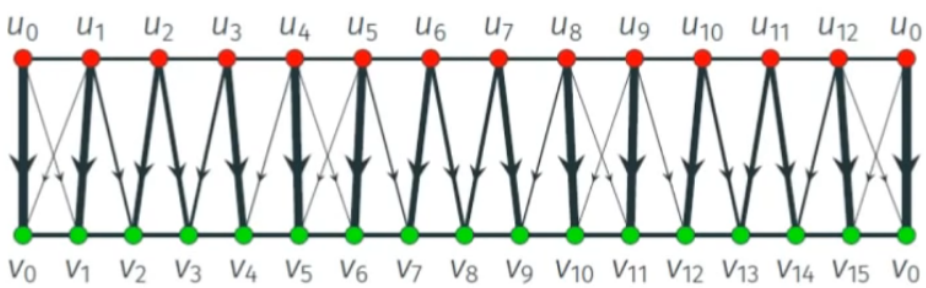

Figure 1: Gaussian resampling, input u

   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.

   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.

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.

   Useful fact: The Furier transform of a Gaussian is a Gaussian.
   This implies the following relation between $\hat{v}$ and $\hat{u}$:

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

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

3. "deconvolve" $\hat{v}$ to obtain $\hat{u}$
   And we're finished!

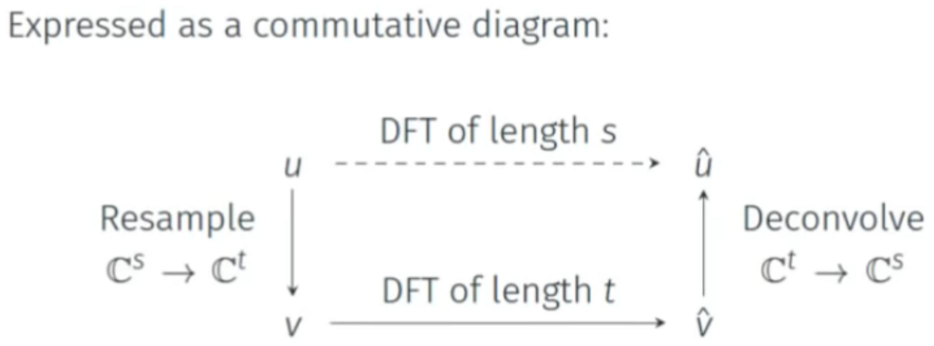

Figure 2:

1.3 back to interger 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 $t_1 \times \cdots \times t_d$

1.3.1 Example

Target degree was 400000
take $d = 3$ and $(s_1, s_2, s_3) = (59, 61, 127)$.
Notice that $s_1 s_2 s_3 = 457074 > 400000$ (twice as big/ constant times bigger)
The corresponding powers of two are $(t_1, t_2, t_3) = (64, 64, 128)$.

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

Compute 3-dimensional convultion 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 Nussbumer's trick... but we can't, because the lengths s_i are not powers of two!

multidimensional transforms:
We reduced 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.
(Then we can use the fake roots of unity in 128 for the 64's?)

1.4 Schönhage-Strassen

FFT, O(nlogn), but inside the FFT the coefficients get to large and you need to call the algorithm recursively O(nlog n log log n ...).

1.5 Nussbaumer's trick

In the ring

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

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

Nussbaumer showed that in some cases you can use "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 of power x_i than by a genuine complex t_i-th root of unity!

Important special case: suppose $t_1, ..., t_d$ are all powers of two. (Nussbaumer's trick applies).
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 $\mathbb{C}$.

A normal FFT uses O(tlogt) multiplications and O(tlogt) additions/subtraction.
Using Nussbaumer's trick the additions/subtractions stay the same but the number of multiplications are only $O(\frac{tlogt}{d})$ (t's are about the same size).

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

2 Complexity analysis

We get an O(nlogn) 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 interger multiplication problems of size roughtly $n^{1/d}$.

Marcus Östling
June 1, 2022

This leads to the recurrence inqualty

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

The factor 1728 can be improved. (double the length of the paper)

Now we take and fixed $d > 1728$ (for example $d = 1729$)
The recurrence then easily leads to $M(n) = O(n \log n)$

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

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

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

3 Final comments

For what n do we win?

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

This can probably be improved.

The algorithm is currently optimizied to recude the size of the paper.