Big integer multiplication

Svyatoslav Covanov

April 10, 2016

1. Fast Fourier Transform
2. Fürer
3. Using generalized Fermat primes

1. Fast Fourier Transform
2. Fürer
3. Using generalized Fermat primes

Naive multiplication

How to multiply two N-bit integers a and b ?

Naive multiplication

How to multiply two N-bit integers a and b ?

Schoolbook multiplication: $O(N^2)$ bit complexity.

Karatsuba:

- $O(N^{\log_2 3})$ bit complexity.
- Transformation of integers into polynomials.

Multiplying integer using polynomials

Input: 2 numbers a and b of N bits.
Output: 2 polynomials $A = \sum_i a_i x^i$ and $B = \sum_i b_i x^i$ of degree $n-1$.

$$a = a_0 + 2^k \times a_1 + \cdots + a_{n-1} \times 2^{(n-1)k} = A(2^k)$$
$$b = b_0 + 2^k \times b_1 + \cdots + b_{n-1} \times 2^{(n-1)k} = B(2^k)$$

Multiplying integer using polynomials

Input: 2 numbers a and b of N bits.
Output: 2 polynomials $A = \sum_i a_i x^i$ and $B = \sum_i b_i x^i$ of degree $n-1$.

$$a = a_0 + 2^k \times a_1 + \cdots + a_{n-1} \times 2^{(n-1)k} = A(2^k)$$
$$b = b_0 + 2^k \times b_1 + \cdots + b_{n-1} \times 2^{(n-1)k} = B(2^k)$$

- $\mathcal{R}$ is a commutative ring.
- $\begin{array}{l} A \longrightarrow \tilde{A} \in \mathcal{R}[x] \\ B \longrightarrow \tilde{B} \in \mathcal{R}[x] \end{array}$.
- $C \longrightarrow \tilde{C} = \tilde{A} \cdot \tilde{B}$ is injective:

$$\forall j, |c_j| = |\sum_{i=0}^{j} a_i \cdot b_{j-i}| < (j+1) \cdot 2^{2k} \leq n \cdot 2^{2k}.$$

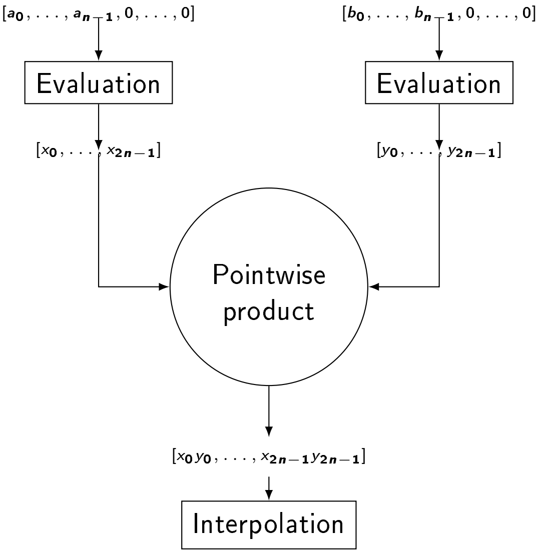
- We choose $2n-1$ distinct points w_i of $\mathcal{R}$.
- Computation of $A(w_i)$ and $B(w_i)$: equivalent to the product

$$\begin{pmatrix} 1 & w_0 & \dots & w_0^{2n-1} \\ 1 & w_1 & \dots & w_1^{2n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & w_{2n-1} & \dots & w_{2n-1}^{2n-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} A(w_0) \\ \vdots \\ A(w_i) \\ \vdots \\ A(w_{2n-1}) \end{pmatrix}.$$

- Pointwise products $A(w_i) \cdot B(w_i) = C(w_i)$.
- Lagrange interpolation of C from the $2n$ points $A(w_i) \cdot B(w_i)$:

$$\begin{pmatrix} 1 & w_0 & \dots & w_0^{2n-1} \\ 1 & w_1 & \dots & w_1^{2n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & w_{2n-1} & \dots & w_{2n-1}^{2n-1} \end{pmatrix}^{-1} \cdot \begin{pmatrix} A(w_0)B(w_0) \\ \vdots \\ A(w_{2n-1})B(w_{2n-1}) \end{pmatrix}.$$

Evaluation-Interpolation scheme



Discrete Fourier Transform (DFT)

If $\mathcal{R}$ is a ring containing a $2n$-th principal root of unity ω:
let

$$M_{2n}(\omega) = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{2n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2n-1} & \dots & (\omega^{2n-1})^{2n-1} \end{pmatrix}.$$

For $A \in \mathcal{R}[x]$,

$$\begin{pmatrix} A(1) \\ A(\omega) \\ \vdots \\ A(\omega^{2n-1}) \end{pmatrix}$$

is the discrete Fourier transform of A.

Definition

Let $\mathcal{R}$ be a **ring** containing a $2n$-th root of unity ω. The root ω is said to be a $2n$-th principal root of unity if

$$\forall i \in [1, 2n-1], \sum_{j=0}^{2n-1} \omega^{ij} = 0.$$

Weaker notion: primitive root of unity if

$$\forall i \in [1, 2n-1], \omega^{i} \neq 1.$$

Primitive and principal is the same thing on a field.

If $\mathcal{R}$ contains a $2n$-th principal root of unity ω, then

$$M_{2n}(\omega)^{-1} = \frac{1}{2n} M_{2n}(\omega^{-1}).$$

$\Rightarrow$ An efficient algorithm for the evaluation gives an efficient algorithm for the interpolation...

Fast Fourier Transform (FFT)

Cooley-Tukey FFT in radix 2:

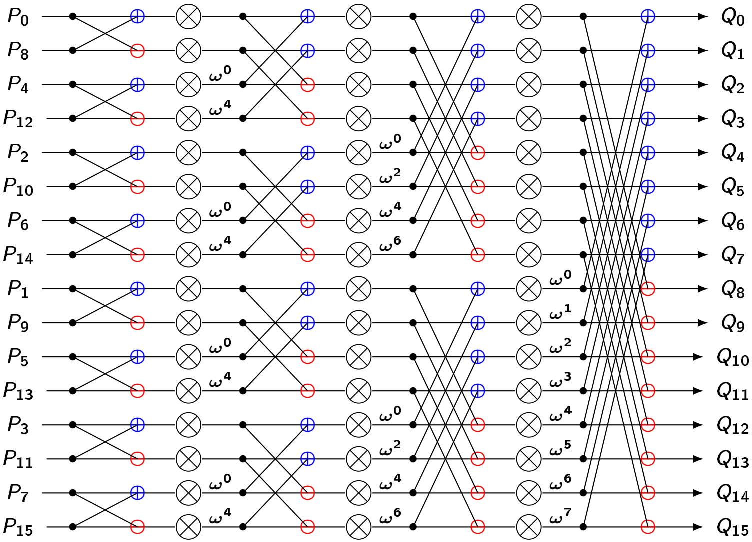
$$\begin{aligned} A(\omega^j) &= \sum_{i\in[0,2n-1]} a_i\omega^{ij} \\ &= \sum_{i\in[0,n-1]} a_{2i+1}\omega^{(2i+1)j} + \sum_{i\in[0,n-1]} a_{2i}\omega^{2ij} \\ &= \omega^j A_{odd}(\omega^{2j}) + A_{even}(\omega^{2j}) \end{aligned}$$

- We compute 2 DFT of n points (for A_{odd} and A_{even}).
- We multiply n points by ω^j (twiddle factors): n multiplications.
- We compute n DFT of 2 points:

$$\pm\omega^j A_{odd}(\omega^{2j}) + A_{even}(\omega^{2j}).$$

FFT(A, ω, $2n$)

if $n = 2$ **then**
    return $A_0 + A_1 + X(A_0 - A_1)$
end if
$A_{even} \leftarrow (A_{2i})_i$
$A_{odd} \leftarrow (A_{2i+1})_i$
$\hat{A}_{even} \leftarrow \mathrm{FFT}(A_{even}, \omega^2, n)$ $\triangleright$ $\hat{A}_{even} = \sum_{i \in [0,n-1]} A_{even}(\omega^{2i})X^i$
$\hat{A}_{odd} \leftarrow \mathrm{FFT}(A_{odd}, \omega^2, n)$ $\triangleright$ $\hat{A}_{odd} = \sum_{i \in [0,n-1]} A_{odd}(\omega^{2i})X^i$
$\hat{A} \leftarrow \hat{A}_{odd}(X) + \hat{A}_{even}(\omega X) + X^n \cdot (\hat{A}_{odd}(X) - \hat{A}_{even}(\omega X))$
return $\hat{A}$



$\Rightarrow 2n = 16$ points, $\log(2n) = 4$ levels, $n(\log(2n) - 1) = 24$ multiplications.

Choice of the ring

1. N: # bits of the integers that we multiply
2. $n-1$: degree of the polynomials A and B used to represent a and b
3. k: # bits used to encode the coefficients of A and B: $a = A(2^k)$, $b = B(2^k)$ and $n \cdot k = N$.

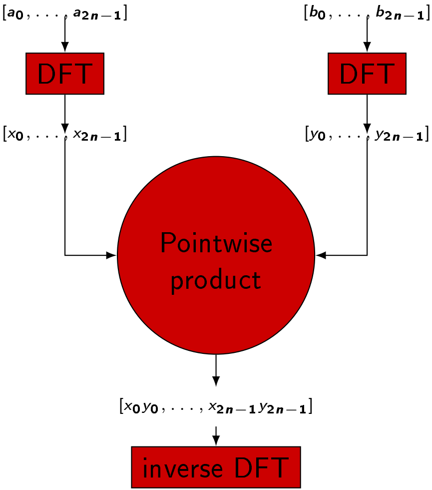
Choice of the ring

1. N: # bits of the integers that we multiply
2. $n - 1$: degree of the polynomials A and B used to represent a and b
3. k: # bits used to encode the coefficients of A and B: $a = A(2^k)$, $b = B(2^k)$ and $n \cdot k = N$.

Examples: (Schönhage-Strassen algorithms)

- $\mathcal{R} = \mathbb{C}$: $\omega = \exp(i\pi/n)$, provided that we allow enough precision .
- $\mathcal{R} = \mathbb{Z}/(2^e + 1)\mathbb{Z}$: $\omega = 2^j$ is a $2e/j$-th principal root of unity .

Complex Case



- $O(n \log n)$ expensive multiplications during the FFT
- $2n$ expensive multiplications during the pointwise product

We choose $n = O(\frac{N}{\log N})$ and $k = O(\log N)$.

Thus, writing the inductive equation for the complexity, we get

$$\mathcal{M}_N = \underbrace{O(N \log N)}_{\text{linear cost}} + \underbrace{(3n \log n + n)}_{\text{3 DFT + pointwise product}} \mathcal{M}_{O(\log N)}$$
$$\leq 4n \log n \cdot \mathcal{M}_{O(\log N)} \leq 4N \cdot \mathcal{M}_{O(\log N)})$$

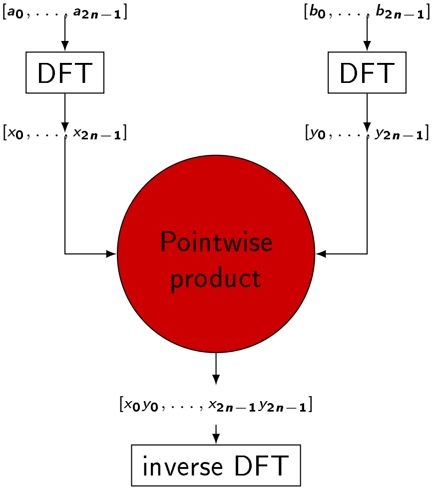
Expanding the equation, we get the following complexity

$$\mathcal{M}_N = N \log(N) \log\log(N) \log\log\log(N) \cdots 2^{O(\log^* N)}.$$

$\log^* N$: iterated logarithm of N,

$$\begin{cases} \log^* N = 1 & \text{if } N \leq 1, \\ \log^* N = 1 + \log^*(\log N) & \text{otherwise.} \end{cases}$$

Modular Case



- $O(n \log n)$ trivial multiplications during the FFT
- $2n$ expensive multiplications during the pointwise product

We choose $2n = \sqrt{N}$ and $e \approx 2\sqrt{N}$.

Thus, writing the inductive equation for the complexity, we get

$$\mathcal{M}_N = \underbrace{O(\sqrt{N}\log N \cdot \sqrt{N})}_{\text{additions, subtractions, shifts in the FFT}} + \underbrace{\sqrt{N}\mathcal{M}_{2\sqrt{N}}}_{\text{pointwise products}} .$$

Expanding the equation, we have

$$\begin{aligned}
\mathcal{M}_N &= O(N\log N) + \sqrt{N}(O(2\sqrt{N}\log 2\sqrt{N}) + \cdots) \\
&= O(N\log N) + O(N\log N) + \cdots \\
&= O(\log\log N \cdot N\log N)
\end{aligned}$$

Some remarks

Case	Degree	Mult. by a root	Recursion	Complexity
$\mathbb{C}$	$O(N/\log N)$	expensive	$O(\log N)$	$N\ \log N\ \log\log N \cdots 2^{O(\log^* N)}$
$\mathbb{Z}/(2^e+1)\mathbb{Z}$	$O(\sqrt{N})$	cheap	$O(\sqrt{N})$	$N\ \log N\ \log\log N$

In $\mathbb{C}$, computing an FFT in $\{1, -1, i, -i\}$ is quite easy. But less obvious for superior orders...

1 Fast Fourier Transform

2 Fürer

3 Using generalized Fermat primes

Cooley-Tukey

- $2n$-point DFT computed with radix-2 FFT:

$$2 \cdot \mathrm{DFT}(n) + \text{Twiddle factors} + n \cdot \mathrm{DFT}(2).$$

- $2n$-point DFT computed with radix-4 FFT:

$$4 \cdot \mathrm{DFT}(n/2) + \text{Twiddle factors} + n/2 \cdot \mathrm{DFT}(4).$$

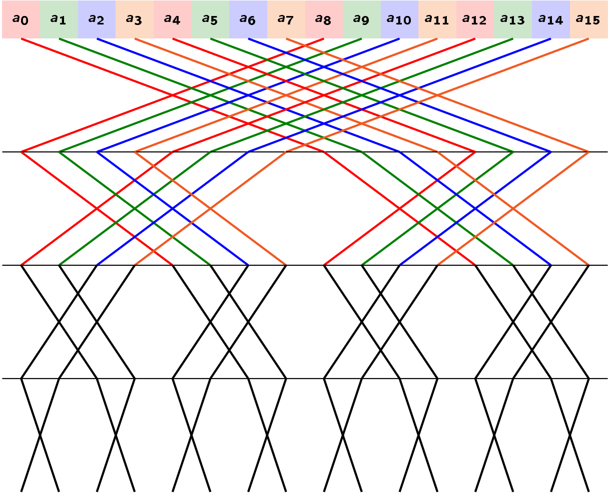
- $2n$-point DFT computed with radix-$2m$ FFT ($2m$ divides $2n$):

$$2m \cdot \mathrm{DFT}(n/m) + \text{Twiddle factors} + n/m \cdot \mathrm{DFT}(2m).$$

$$\mathrm{DFT}(mn) = m \cdot \mathrm{DFT}(n) + \text{Twiddle factors} + n \cdot \mathrm{DFT}(m).$$

Radix-4 Cooley-Tukey

$4 \cdot \mathrm{DFT}(4)$:

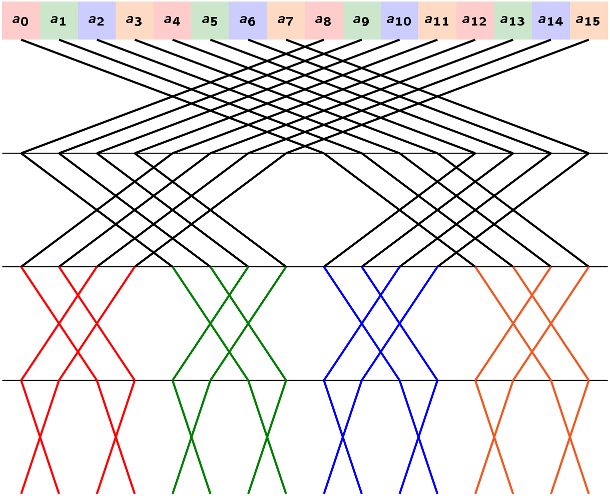


Matrix point of view:

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \\ a_{12} & a_{13} & a_{14} & a_{15} \end{pmatrix} \cdot M_4^T(\omega^4)$$

Radix-4 Cooley-Tukey

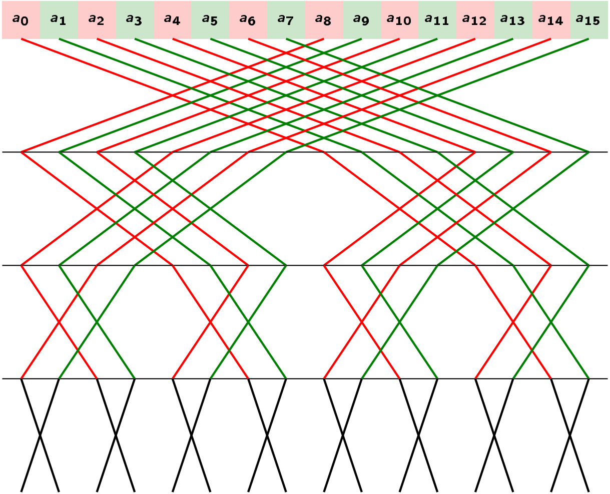
$4 \cdot \mathrm{DFT}(4)$:



Matrix point of view:

$$M_4(\omega^4) \cdot \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \\ a_{12} & a_{13} & a_{14} & a_{15} \end{pmatrix}$$

Radix-8 Cooley-Tukey



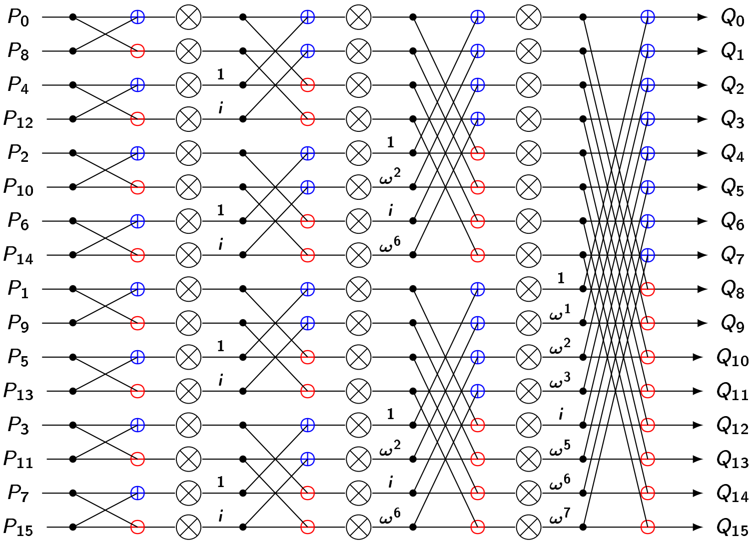
Matrix point of view:

$$M_8(\omega^2) \cdot \begin{pmatrix} a_0 & a_1 \\ a_2 & a_3 \\ a_4 & a_5 \\ a_6 & a_7 \\ a_8 & a_9 \\ a_{10} & a_{11} \\ a_{12} & a_{13} \\ a_{14} & a_{15} \end{pmatrix}$$

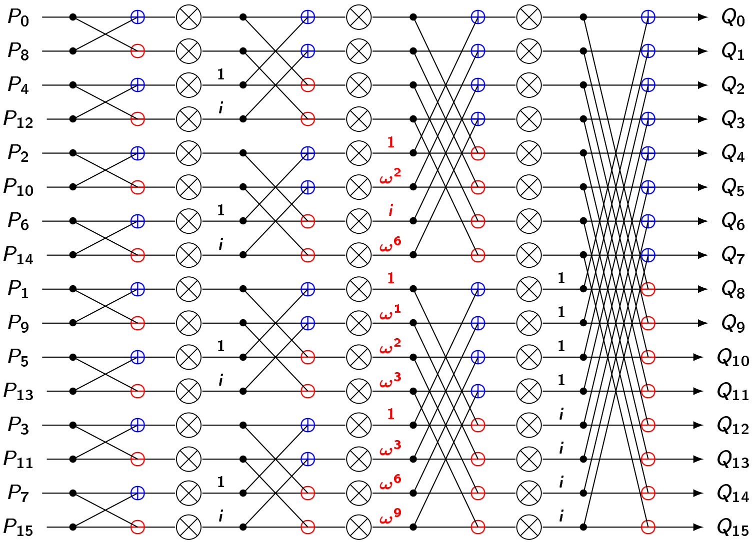
Cooley-Tukey FFT algorithm for $2n$ points:

- Radix-2 $\Rightarrow \log_2(2n)$ levels of recursion with $\frac{2n}{2}$ 2-point FFT on each level.
- Radix-4 $\Rightarrow \log_4(2n)$ levels of recursion with $\frac{2n}{4}$ 4-point FFT on each level.
- Radix-$2m$ $\Rightarrow \log_{2m}(2n)$ levels of recursion with $\frac{n}{m}$ $2m$-point FFT on each level.

An example in Complex Field: radix-2 FFT



An example in Complex Field: radix-4 FFT



Fürer's algorithm

- $\mathcal{R}$ is the ring $\mathcal{R} = \mathbb{C}[x]/(x^P + 1)$ (P divides $2n$).
  $\Rightarrow$There exists a $2n$-th root of unity ρ such that $\rho^{n/P} = x$.
- Computation of $2n$-point DFT with radix-$2P$ FFT.
- $\log_{2P} 2n$ levels of recursion:

$$\underbrace{\log_{2P}(2n)}_{\text{nb. of levels}} \cdot \underbrace{2n}_{\text{mult. per level}} \cdot \underbrace{\mathcal{M}_{\mathcal{R}}}_{\text{cost of a mult. in } \mathcal{R}}$$

  expensive multiplications.

Remark:
We use Lagrange interpolation to find ρ in $\mathcal{R}$:

$$\forall j \in [0, 2P-1], \rho(e^{\frac{2ji\pi}{n}}) = e^{\frac{ji\pi}{P}}.$$

Size of coefficients

- We choose $P = O(\log N)$.
  For $R = \sum_{i=0}^{P-1} r_i x^i \in \mathcal{R}$, $\log |r_i| \leq t = \Theta(\log N)$.
- An integer a is cut in pieces a_i of size k.
  $\Rightarrow$ A polynomial $\hat{A} = \sum_{i \in [0,n-1]} a_i Y^i$.
- Each a_i is transformed in an element of $\mathcal{R}$ (cut in $P/2$ pieces).
  $\Rightarrow$ A polynomial $A = \sum_{i \in [0,n-1]} \left(\sum_{j \in [0,P/2-1]} a_{ij} X^j \right) Y^i$.

We must have ($C = A \cdot B \longrightarrow \tilde{C} = \tilde{A} \cdot \tilde{B}$ is injective)

$$\log n + \log P + 2 \cdot (2k/P) \leq t.$$

Size of coefficients

- We choose $P = O(\log N)$.
  For $R = \sum_{i=0}^{P-1} r_i x^i \in \mathcal{R}$, $\log |r_i| \leq t = \Theta(\log N)$.
- An integer a is cut in pieces a_i of size k.
  $\Rightarrow$ A polynomial $\hat{A} = \sum_{i \in [0,n-1]} a_i Y^i$.
- Each a_i is transformed in an element of $\mathcal{R}$ (cut in $P/2$ pieces).
  $\Rightarrow$ A polynomial $A = \sum_{i \in [0,n-1]} \left(\sum_{j \in [0,P/2-1]} a_{ij} X^j \right) Y^i$.

We must have ($C = A \cdot B \longrightarrow \tilde{C} = \tilde{A} \cdot \tilde{B}$ is injective)

$$\log n + \log P + 2 \cdot (2k/P) \leq t.$$

Case	Degree	Mult. by a root	Recursion	Complexity
$\mathbb{C}$	$O(N/\log N)$	expensive	$O(\log N)$	$N \log N \log\log N \cdots 2^{O(\log^* N)}$
$\mathbb{Z}/(2^e+1)\mathbb{Z}$	$O(\sqrt{N})$	cheap	$O(\sqrt{N})$	$N \log N \log\log N$
$\mathbb{C}[x]/(x^P+1)$	$O(N/\log^2 N)$	it depends	$O(\log^2 N)$	$N \log N\, 2^{O(\log^* N)}$

In 2014, Harvey, Lecerf and Van Der Hœven proved that the exact complexity is

$$N \ \log N \ 16^{\log^* N}.$$

With Bluestein's Chirp transform, they reach unconditionally $N \ \log N \ 8^{\log^* N}$.

By using a conjecture on Mersenne primes, they even have $N \ \log N \ 4^{\log^* N}$.

1. Fast Fourier Transform
2. Fürer
3. Using generalized Fermat primes

What can we improve?

- We cut an N-bit integer in pieces of size $k \Rightarrow n = \frac{N}{k}$ pieces. The elements of $\mathcal{R}$ are encoded on Pt bits and t satisfies

$$\log n + \log P + 4k/P \leq t.$$

$\Rightarrow$ The cost of $\mathcal{M}_{\mathcal{R}}^{KS}$ is at least

$$\mathcal{M}_{Pt} \geq \mathcal{M}_{4k}$$

(multiplication of $4k$-bit integers).

- A multiplication in $\mathcal{R}$ requires padding (Kronecker substitution):

$$\mathcal{M}_{\mathcal{R}}^{KS} \text{ is at least } \mathcal{M}_{2tP} \geq \mathcal{M}_{8k}.$$

Number-theoretic transform

1. N: # bits of the integers that we multiply
2. $n-1$: degree of the polynomials A and B used to represent a and b
3. k: # bits used to encode the coefficients of A and B: $a = A(2^k)$ and $b = B(2^k)$

Instead of computing FFT over $\mathbb{C}$, we can choose $\mathcal{R} = \mathbb{Z}/q\mathbb{Z}$.
The prime q must satisy $2n \mid q-1$ (there exists a $2n$-th principal root of unity).
A choice of q such that $\log q = O(\log N)$ is optimal.

Number-theoretic transform

1. N: # bits of the integers that we multiply
2. $n-1$: degree of the polynomials A and B used to represent a and b
3. k: # bits used to encode the coefficients of A and B: $a = A(2^k)$ and $b = B(2^k)$

Instead of computing FFT over $\mathbb{C}$, we can choose $\mathcal{R} = \mathbb{Z}/q\mathbb{Z}$.
The prime q must satisy $2n \mid q-1$ (there exists a $2n$-th principal root of unity).
A choice of q such that $\log q = O(\log N)$ is optimal.

We cut the N-bit integers in pieces of size $k \approx \frac{1}{2}\log q$:

$$\log n + 2k \leq \log q.$$

$\Rightarrow \mathcal{M}_{\mathcal{R}}^{KS} \geq \mathcal{M}_{2k}$.

A Fürer-like number theoretic transform

- q is chosen such that $q = r^P + 1$: this is a generalized Fermat prime.
  Conjecturally, there exists r such that $r < P \cdot (\log P)^2 \Rightarrow \log_2 q \approx P \log P$.
- Let ρ be a $2n$-th root of unity in $\mathbb{Z}/q\mathbb{Z}$ such that $\rho^{n/P} = r$.

A Fürer-like number theoretic transform

- q is chosen such that $q = r^P + 1$: this is a generalized Fermat prime.
  Conjecturally, there exists r such that $r < P \cdot (\log P)^2 \Rightarrow \log_2 q \approx P \log P$.
- Let ρ be a $2n$-th root of unity in $\mathbb{Z}/q\mathbb{Z}$ such that $\rho^{n/P} = r$.

Working in radix r is like working with "polynomials" of degree P whose coefficients are bounded by r:

$$\mathcal{M}_{\mathcal{R}} \leq \mathcal{M}_{\mathbb{Z}_P[X]}.$$

How to multiply in $\mathcal{R}$

Instead of Kronecker substitution, we directly compute an FFT.

How to multiply in $\mathcal{R}$

Instead of Kronecker substitution, we directly compute an FFT.

- $x \in \mathbb{Z}/q\mathbb{Z}$ and $y \in \mathbb{Z}/q\mathbb{Z}$ are represented by polynomials over $\mathbb{Z}$:

  $$X(r) = x_0 + x_1 \cdot r + x_2 \cdot r^2 \cdots x_{P-1} \cdot r^{P-1}$$

  and

  $$Y(r) = y_0 + y_1 \cdot r + y_2 \cdot r^2 \cdots y_{P-1} \cdot r^{P-1}.$$

- We choose $Q = O(\log \log P)$ and we represent x and y in radix r^Q.
  $\Rightarrow$ We get $\tilde{X}$ and $\tilde{Y}$ polynomials modulo $X^{P/Q} + 1$ with coefficients $\leq r^Q$.
  $\Rightarrow$ We compute a P/Q-points FFT.
- We get $x \cdot y \in \mathbb{Z}/q\mathbb{Z}$ with reductions modulo r.

How to multiply in $\mathcal{R}$

Instead of Kronecker substitution, we directly compute an FFT.

- $x \in \mathbb{Z}/q\mathbb{Z}$ and $y \in \mathbb{Z}/q\mathbb{Z}$ are represented by polynomials over $\mathbb{Z}$:

$$X(r) = x_0 + x_1 \cdot r + x_2 \cdot r^2 \cdots x_{P-1} \cdot r^{P-1}$$

  and

$$Y(r) = y_0 + y_1 \cdot r + y_2 \cdot r^2 \cdots y_{P-1} \cdot r^{P-1}.$$

- We choose $Q = O(\log\log P)$ and we represent x and y in radix r^Q.
  $\Rightarrow$ We get $\tilde{X}$ and $\tilde{Y}$ polynomials modulo $X^{P/Q} + 1$ with coefficients $\leq r^Q$.
  $\Rightarrow$ We compute a P/Q-points FFT.
- We get $x \cdot y \in \mathbb{Z}/q\mathbb{Z}$ with reductions modulo r.

We do not have anymore the zero-padding due to Kronecker substitution.

Steps of the algorithm

- Find a prime $q = r^{\log N} + 1$ sufficiently large for multiplying integers of size N.
- Cut the integers a and b into pieces of size $k = O(\log N \log \log N)$, that are the coefficients of A and B.
- Represent the pieces as elements of $\mathbb{Z}/q\mathbb{Z}$ in radix r.
- Compute the FFT, the componentwise product, the inverse FFT.
- Switch from radix r to the regular representation of elements of $\mathbb{Z}/q\mathbb{Z}$.
- Transform the polynomial $C = A \cdot B$ into an integer c by evaluating it at 2^k.

Comparison of complexities

Using generalized Fermat primes we get the following data:

Case	Degree	Mult. by a root	Recursion	Complexity
$\mathbb{C}$	$O(N/\log N)$	expensive	$O(\log N)$	$N \cdot \log N \cdot \log\log N \cdots 2^{O(\log^* N)}$
$\mathbb{Z}/(2^e+1)\mathbb{Z}$	$O(\sqrt{N})$	cheap	$O(\sqrt{N})$	$N \cdot \log N \cdot \log\log N$
$\mathbb{C}[x]/(x^P+1)$	$O(N/\log^2 N)$	it depends	$O(\log^2 N)$	$N \cdot \log N\ 16^{\log^* N}$
$\mathbb{Z}/(r^P+1)\mathbb{Z}$	$O(N/(\log N \cdot \log\log N))$	it depends	$O(\log N \cdot \log\log N)$	$N \cdot \log N\ 4^{\log^* N}$

Some estimations

	Schönhage-Strassen algorithm		
bitsize	nb. mult.	mult. bitsize	estimated time (s)
2^{30}	2^{16}	$\approx 2^{16}$	9.96
2^{36}	2^{18}	$\approx 2^{18}$	$2.60 \cdot 10^2$
2^{40}	2^{21}	$\approx 2^{21}$	$2.36 \cdot 10^4$
2^{46}	2^{24}	$\approx 2^{24}$	$2.17 \cdot 10^6$
2^{50}	2^{26}	$\approx 2^{26}$	$4.10 \cdot 10^7$
2^{56}	2^{29}	$\approx 2^{29}$	$2.94 \cdot 10^9$

	Generalized Fermat primes			
bitsize	nb. mult.	prime	KS. bitsize	estimated time (s)
2^{30}	$2^{24} \cdot 13$	$562^{32} + 1$	800	$3.57 \cdot 10$
2^{36}	$2^{30} \cdot 16$	$562^{32} + 1$	800	$3.35 \cdot 10^3$
2^{40}	$2^{34} \cdot 19$	$562^{32} + 1$	800	$6.26 \cdot 10^4$
2^{46}	$2^{40} \cdot 22$	$884^{32} + 1$	800	$4.64 \cdot 10^6$
2^{50}	$2^{44} \cdot 25$	$884^{32} + 1$	800	$7.91 \cdot 10^7$
2^{56}	$2^{50} \cdot 28$	$884^{32} + 1$	800	$5.67 \cdot 10^9$

- nb. mult.: $2n \cdot (3 \cdot \lceil \log_{2P} 2n \rceil + 1)$.

Conclusion

Avoiding the padding due to a modular ring and the Kronecker substitution improves on the complexity of the algorithm: we reach $N \log N \cdot 4^{\log^* N}$.

The complexity is conjectural: related to "Hypothesis H" and lower bounds on r such that $P(r)$ is prime for a polynomial P.

In practice, we do not expect this algorithm to improve on Schönhage-Strassen for sizes $\leq 2^{40}$ bits.

It is possible to improve the arithmetic in $\mathbb{Z}/q\mathbb{Z}$ by choosing $q = b^P + 1$ with a special b (sparse?): a lot of generalized Fermat primes.