MULTIPLICATION*

S. B. Gashkov†, I. S. Sergeev‡(Moscow)

Abstract

The paper provides a survey of the modern state of the theory of fast multiplication of numbers and polynomials. We consider the development of multiplication methods from the initial block algorithms of 1960s due to Karatsuba and Toom to the methods of 1970s based on the Discrete Fourier transform (DFT) and further to the novel methods invented in 2007–2009[1]. Modern multiplication methods combine exploiting of special algebraic structures, the use of approximate computations, special forms of Fourier transforms, namely, multidimensional DFT, additive analogue of DFT. These and other concepts essential for the fast multiplication algorithms are thoroughly discussed in the present survey. In particular, we provide an introduction to the theory of DFT with derivation of facts necessary for the exposition. The final part of the survey contains a brief discussion of results on parallel multiplication algorithms, accurate complexity bounds of the basic methods, online multiplication algorithms, multiplicative complexity of the multiplication of polynomials over finite fields. We mention computational models where multiplication has either linear, or quadratic complexity.

Keywords: fast computations, multiplication, complexity, discrete Fourier transform, additive DFT.

Bibliography: 89 titles.

Translated version. Originally published in: S. B. Gashkov, I. S. Sergeev, 2020, "Multiplication", *Chebyshevskii sbornik*, vol. 21, no. 1, pp. 101–134. (in Russian)

1 Introduction

Interest in methods of fast calculations arose even among ancient mathematicians: it is enough to recall Euclid's algorithm for the greatest common divisor. But only at the beginning of the 20th century did a mathematical theory appear that allows to strictly formulate and give precise answers to questions like "how fast does this algorithm work?" or "which algorithm is faster?" The central concept of this theory is *complexity*.

Complexity theory, which can be attributed to mathematical logic, discrete mathematics or mathematical cybernetics[2], arose in the works of the American theoretical engineer C. Shannon in the 1930–40s.

The technological revolution of the mid-20th century, which led to the birth of modern electronics, provoked the rapid development of related scientific disciplines: computer science, cybernetics, computational methods, the theory of algorithms and complexity. In addition to Shannon, A. N. Kolmogorov made a significant contribution to the creation of the mathematical

*The research was supported by RFBR (project 19-01-00294a).

†**Gashkov Sergey Borisovich** — doctor of science, professor, Lomonosov Moscow State University (Moscow). *e-mail: sbgashkov@gmail.com*

‡**Sergeev Igor Sergeevich** — candidate of science; head of laboratory, Federal State Unitary Enterprise "Scientific and Research Institute "Kvant" (Moscow). *e-mail: isserg@gmail.com*

[1]Note from 2025: actually, we discuss methods appeared by 2020.

[2]Note from 2025: or rather to computer science, in the Western classification.

apparatus of these theories (see, e.g., [57]). He also actively supported research in new directions in the USSR and abroad.

Fast algorithms for individual arithmetic operations began to appear without waiting for either the final formulation of the theory or emergence of powerful computing tools. Just a few examples. In 1938, D. Lehmer constructed an accelerated version of the Euclidean GCD algorithm, interestingly, specially adapted for computer applications. Lehmer's method is the basis of modern fast GCD algorithms. A year later, A. Brauer published an optimal method for raising to a natural power in terms of the number of multiplications — it is approximately twice as fast as the trivial binary method.

In the 1950s, engineers and mathematicians were already actively developing efficient designs of circuits or programs for the efficient implementation of arithmetic. Among the theoretical results, we can note the final solution to the problem of the complexity of evaluating a given polynomial at a point, obtained in the late 1950s by students of A. G. Vitushkin (who, in turn, was a student of Kolmogorov) V. Ya. Pan and E. G. Belaga. A little earlier in the USA, this problem was studied by T. Motzkin. In particular, they constructed fast algorithms for evaluating a polynomial, using approximately two times fewer multiplications of coefficients than in Horner's scheme.

However, it was with the advent of A. A. Karatsuba's fast method of multiplying numbers in 1960 (or rather, with its publication in 1962) that the formation of a new scientific direction is associated – the theory of fast computations. The extraordinary influence that Karatsuba's result had on the development of the theory is explained by the combined effect of several circumstances. On the one hand, the result was sensational (many believed that the school method of multiplication was optimal). On the other hand, Karatsuba's method demonstrates the efficiency of a general technique for reducing a large problem to smaller problems, which was then successfully applied to construct fast algorithms for many other arithmetic operations. Thirdly, multiplication is a fundamental operation – a building block for various arithmetic operations, which, as a result, are automatically accelerated by accelerating the multiplication algorithm.

As a result, both objective factors (the fundamental nature of the multiplication operation) and subjective factors (the effect produced by Karatsuba's method) led to the problem of studying the complexity of multiplication becoming central for the theory of fast computations.

By the time the first fast multiplication algorithms (Karatsuba's and Toom's) appeared in the early 1960s, the apparatus of complexity theory had fully formed, and the central question of the complexity of implementing an arbitrary Boolean function was solved in the asymptotic sense for several basic computational models by O. B. Lupanov.

One can notice the similarity in the development of fast methods for computing Boolean functions and fast methods for multiplication. The first nontrivial estimate of the complexity of computing a function is given by Shannon's cascade method. Using the formula

$$f(x_1, x_2, \ldots, x_n) = f(1, x_2, \ldots, x_n) \cdot x_1 \vee f(0, x_2, \ldots, x_n) \cdot \overline{x_1}$$

it reduces the original problem to two similar problems of smaller size — implementations of subfunctions (here $\overline{x}$ denotes the negation of variable x). Similarly, Karatsuba's method reduces the original multiplication to three multiplications of smaller size, only the formula used in it is less obvious. Lupanov's asymptotically optimal method nontrivially develops the idea of the cascade method and consists in reducing to the computation of a function in local domains. The principle underlying the method is called the local coding principle. In the multiplication problem, this principle corresponds to the idea of interpolation. All fast multiplication methods, starting with Toom's method, are based on interpolation.

Along with numerical multiplication, we consider the multiplication of polynomials of one variable. This is motivated, firstly, by the fact that the methods of multiplying numbers and

polynomials are closely interrelated, and secondly, the importance of polynomial arithmetic continues to grow as cryptography develops. Perhaps especially in demand are methods for fast multiplication of polynomials over finite fields of characteristic 2.

Formally, the multiplication of n-digit numbers can be interpreted as a Boolean operator whose inputs and outputs are the binary digits of the numbers being multiplied and the product, respectively. To estimate the complexity of Boolean operators, it is convenient to use a computational model of Boolean circuits. *Boolean circuits* are built from functional elements with two inputs and one output, implementing all possible Boolean operations. *Complexity* of a circuit is the number of functional elements in it. The latter concept is very close to the concept of *bit complexity* of a computation, and the concept of a circuit itself is very close to the concept of *a straight-line program*. In theoretical works on computer arithmetic, computations are often simulated on (multi-tape) *Turing machines*, the operating principle of which is to read and rewrite information on a tape using a movable head that functions as an automaton. In practice, Turing machines in their pure form are not used.

To study methods of multiplying polynomials over some ring K, it is natural to use the model of *arithmetic circuits*, otherwise called *circuits over the ring* K. Unlike Boolean circuits, arithmetic circuits are built from elements of addition, multiplication, and multiplication by constants of the ring K.

A more detailed explanation of computational models and basic concepts of complexity theory can be found, for example, in [3, 56, 86, 33, 14].

Below, $M(n)$ denotes the complexity of multiplying n-digit numbers when implemented by Boolean circuits (or straight-line programs). $M_K(n)$ denotes the complexity of multiplying polynomials of degree $< n$ over K when implemented by circuits over K. M_K denotes the complexity of multiplication in the ring K when implemented by Boolean circuits.

2 Karatsuba's method

The standard (machine) method of multiplication is based on a well-known school trick: one of the factors is successively multiplied by the digits of the other, the correspondingly shifted results are written under each other and then added together. The multiplication of numbers 26 and 21 in this interpretation would look like this:

					1	1	0	1	0
					1	0	1	0	1
					1	1	0	1	0
				0	0	0	0	0	
			1	1	0	1	0		
		0	0	0	0	0			
	1	1	0	1	0				
1	0	0	0	1	0	0	0	1	0

Fig. 1

The complexity of multiplying n-bit numbers in this way, as is easy to check, has the order of n^2 operations. If we calculate more accurately, a complexity estimate of $6n^2 - 8n$ may be obtained: the multiplication circuit contains n^2 bit multiplication elements and $n - 1$ standard adders to compute the sum of n numbers.

Such estimate can also be derived recursively. Let's split $2n$-bit numbers A and B into blocks of n digits: $A = A_1 2^n + A_0$, $B = B_1 2^n + B_0$. Now the multiplication of the original numbers is performed using four multiplications of "halves" and several additions according to the formula

$$AB = A_1 B_1 2^{2n} + (A_0 B_1 + A_1 B_0) 2^n + A_0 B_0. \tag{1}$$

Reflecting on the problem of the complexity of multiplication posed by A. N. Kolmogorov, in 1960 A. A. Karatsuba (at that time a graduate student at Moscow State University) discovered a more economical formula than (1),

$$AB = A_1B_1 2^{2n} + (A_1B_1 + A_0B_0 - (A_0 - A_1)(B_0 - B_1))2^n + A_0B_0, \tag{2}$$

which requires only three multiplications of n-digit numbers and several additions and subtractions. For the complexity $K(n)$ of the method, we obtain the recurrence relation:

$$K(2n) \le 3K(n) + O(n),$$

which is resolved as $K(n) = O\left(n^{\log 3}\right)$. (Here and further in the text all logarithms are to the base 2.)

The author of the first fast multiplication method tells in detail about its history in [49, 50].

A similar method and result are valid for polynomials. The polynomial analogy allows us to see the idea of interpolation at the heart of Karatsuba's method. Thus, the multiplication of degree-1 polynomials is performed in three steps:

i) Evaluating polynomials at three points[3]: 0, -1, ∞.

ii) Multiplying the values point-wise.

iii) Recovering a polynomial with given values at the interpolation points, see formula (2).

The development of the interpolation idea led to the emergence of theoretically faster methods of multiplication.

3 Block multiplication methods of the 1960s

The first such method was proposed by A. L. Toom (at that time a student of the Mechanics and Mathematics Department of Moscow State University) in 1963 [83]. In Toom's method, each multiplier is divided into r blocks:

$$A = \sum_{i=0}^{r-1} A_i 2^{iq}, \qquad B = \sum_{i=0}^{r-1} B_i 2^{iq}, \qquad A_i, B_i < 2^q. \tag{3}$$

The product of numbers is restored from the product of polynomials $A(X) = \sum A_i X^i$ and $B(X) = \sum B_i X^i$ by substituting $X = 2^q$. To multiply polynomials, interpolation is performed at $2r - 1$ points: $-r + 1, \ldots, r - 1$.

Note that multiplications in Toom's method occur not only at the interpolation points, but also when evaluating the polynomials $A(X)$ and $B(X)$ at these points, and when reconstructing the polynomial $A(X)B(X)$ from the values at the points. In the original method, the reconstruction stage is considered as solving a system of linear equations for the coefficients of the desired polynomial.

Toom's method is practical for small values of r. Choosing $r = 2$ simply yields a variant of Karatsuba's method. To this day, Karatsuba's methods and the ternary ($r = 3$) variant of Toom's method are commonly used to multiply numbers in the range from a few tens to 1000 binary digits. But to derive a theoretical complexity estimate, one should choose r to grow with n, and the optimal choice is $r = 2^{\Theta(\sqrt{\log n})}$.

The author's estimate of the complexity of the method is $O\left(n2^{5\sqrt{\log n}}\right)$. A more thorough analysis by D. Knuth (see [53]) resulted in a refinement of the estimate to $O\left(n2^{\sqrt{2\log n}} \log n\right)$.

A version of Toom's method adapted for multiplying polynomials (indicated by S. Cook [17]) suggests choosing polynomials of small degrees with coefficients 0 and 1 as interpolation points.

[3]The value of a polynomial at point ∞ is assumed to be equal to the leading coefficient.

This choice is universal: it allows multiplying binary polynomials as well. Record estimates of the complexity of multiplying binary polynomials of degrees within one or two hundred are obtained by specially optimized block methods, see, for example, [13, 63, 23].

More broadly, the idea of the transition to a modular representation can be seen as the basis of Toom's method, since interpolation is a special case of this idea. In a modular representation, a polynomial is written as a set of residues modulo some given polynomials $f_i(x)$. The Toom—Cook method, as described above, uses polynomials $x^q - \alpha_i$, where α_i are points: numbers or polynomials.

Developing the modular idea, A. Schönhage in 1966 [69] constructed a suitable system of modules of the form $2^{q_i} - 1$ and obtained another method of multiplying numbers with a complexity estimate $O\left(n2^{\sqrt{2\log n}}\log^{3/2} n\right)$.

A year before this, an article by J. Cooley and J. Tukey [18] appeared, in which the authors indicated a fast method for computing the discrete Fourier transform (DFT) and generally drew attention to the DFT as a tool for fast computations. Since then, all new fast multiplication algorithms are based on the DFT.

4 Discrete Fourier transform

The application of the DFT to the problem of multiplication practically ended the discussion of the optimal choice of interpolation points. Multiplication via the DFT uses a group of roots of unity as interpolation points. The following is some background on the DFT that is necessary for presenting fast multiplication methods.

Following [33], we give a formal definition. Let K be a commutative, associative ring with unity. An element $\zeta \in K$ is called a *primitive root of order* $N \in \mathbb{N}$ if $\zeta^N = 1$, and none of the elements $\zeta^{N/p} - 1$, where p is a prime divisor of N, is a zero divisor in K. (Recall that an element a is called a zero divisor if there is a nonzero element b such that $ab = 0$.)

The *discrete Fourier transform* (*DFT*) *of order* N is a $(K^N \to K^N)$-transform

$$\mathrm{DFT}_{N,\zeta}[K](\gamma_0, \ldots, \gamma_{N-1}) = (\gamma_0^*, \ldots, \gamma_{N-1}^*), \qquad \gamma_j^* = \sum_{i=0}^{N-1} \gamma_i \zeta^{ij}, \tag{4}$$

where ζ is a primitive root of order N.

The fundamental property of the DFT is formulated as follows:

Lemma 1. *Let the elements* γ_j^* *be determined from* (4). *Then*

$$\mathrm{DFT}_{N,\zeta^{-1}}[K](\gamma_0^*, \ldots, \gamma_{N-1}^*) = (N\gamma_0, \ldots, N\gamma_{N-1}),$$

where N *on the right-hand side of the formula is the sum of* N *units of the ring.*

As a consequence, we obtain that if the element $N = 1 + \ldots + 1 \in K$ is invertible, then the inverse transform to the DFT is defined as

$$\mathrm{DFT}_{N,\zeta}^{-1}[K] = N^{-1} \cdot \mathrm{DFT}_{N,\zeta^{-1}}[K].$$

4.1 Polynomial interpretation of the DFT

Consider a polynomial $\Gamma(x) = \gamma_0 + \ldots + \gamma_{N-1}x^{N-1}$. Then, by definition,

$$\mathrm{DFT}_{N,\zeta}[K](\gamma_0, \ldots, \gamma_{N-1}) = \left(\Gamma(\zeta^0), \ldots, \Gamma(\zeta^{N-1})\right),$$

i.e., the DFT evaluates the polynomial $\Gamma(x)$ at the points ζ^i. The meaning of the inverse transform $\mathrm{DFT}^{-1}_{N,\zeta}[K]$ is to restore the coefficients of the unique polynomial of degree $< N$ that has a given set of values at the points $\zeta^0, \ldots, \zeta^{N-1}$.

Formally, the relationship between the DFT and interpolation is described by the following lemma:

Lemma 2. *The transform* $\mathrm{DFT}_{N,\zeta}[K]$ *defines an isomorphism:* $K[x]/(x^N - 1) \to K^N$.

The above isomorphism leads to an efficient way of multiplying polynomials over K.

Theorem 1. *Let* $\mathrm{DFT}_{N,\zeta}[K]$ *and its inverse be defined in a ring* K*. Then the multiplication of two polynomials modulo* $x^N - 1$ *over* K *can be performed using two transforms* $\mathrm{DFT}_{N,\zeta}[K]$*, one* $\mathrm{DFT}^{-1}_{N,\zeta}[K]$*,* N *nonsclar multiplications in* K*, and* N *multiplications by constants* $N^{-1} \in K$*.*

If we represent polynomials as vectors of coefficients, then the multiplication modulo $x^N - 1$ coincides with the *cyclic convolution of order* N.

4.2 DFT computation

The efficiency of the DFT is ensured by a family of algorithms for its fast computation. These algorithms are called the fast Fourier transform (FFT). In general, FFT algorithms are those algorithms that employ the technique of reducing a DFT of composite order N to DFTs of order of factors of N. Sometimes the term FFT is applied to any algorithms of complexity $O(N \log N)$.

The most popular example of FFT is the Cooley—Tukey method [18] based on the following lemma.

Lemma 3 ([18])**.** *A DFT of order* ST *is implemented using* S *DFTs of order* T*,* T *DFTs of order* S*, and* $(S-1)(T-1)$ *multiplications by powers of* ζ *— a primitive root of order* ST*.*

The proof of the lemma is based on the decomposition of the DFT into "external" DFTs of order S and "internal" DFTs of order T. For any $s = 0, \ldots, S-1$ and $t = 0, \ldots, T-1$, the following holds:

$$\gamma^*_{sT+t} = \sum_{I=0}^{ST-1} \gamma_I \zeta^{I(sT+t)} = \sum_{i=0}^{T-1} \sum_{j=0}^{S-1} \gamma_{iS+j} \zeta^{(iS+j)(sT+t)} =$$

$$= \sum_{i=0}^{T-1} \sum_{j=0}^{S-1} \gamma_{iS+j} \zeta^{itS+jsT+jt} = \sum_{j=0}^{S-1} (\zeta^T)^{js} \cdot \zeta^{jt} \cdot \gamma_{(j),t}, \quad (5)$$

where

$$\gamma_{(j),t} = \sum_{i=0}^{T-1} \gamma_{iS+j} (\zeta^S)^{it}.$$

Given that the DFT of order 2 costs an addition and a subtraction, a recursive application of Lemma 3 yields

Lemma 4. *The DFT of order* 2^k *can be performed in* $k2^k$ *addition-subtraction operations and* $(k-2)2^{k-1} + 1$ *scalar multiplication operations.*

This upper bound is asymptotically the best known for the complexity of the DFT of order 2^k.

For the case where N is decomposed into relatively prime factors, there is a more efficient algorithm by I. Good [34] that does not require additional multiplications by powers of the primitive root. This algorithm is also called the Good—Thomas algorithm, since it was rediscovered by L. Thomas [82] several years after Good's work.

Lemma 5 ([34]). *If* $\mathrm{GCD}(S, T) = 1$*, then to compute a DFT of order* ST *it is sufficient to perform* S *DFTs of order* T *and* T *DFTs of order* S*.*

The proof is based on the following decomposition. Let m, n be the Bezout coefficients from the equality $nS + mT = 1$. First, for $I = 0, \ldots, ST-1$ denote $\gamma_I = \gamma_{i,\,j}$, where $i = I \bmod T$ and $j = I \bmod S$, and for $K = 0, \ldots, ST-1$ denote $\gamma^*_K = \gamma^*_{s,\,t}$, where $s = mK \bmod S$ and $t = nK \bmod T$. Note that $I = (inS + jmT) \bmod ST$ and $K = (sT + tS) \bmod ST$. Then for any K,

$$\gamma^*_K = \gamma^*_{s,\,t} = \sum_{I=0}^{ST-1} \gamma_I \zeta^{IK} = \sum_{j=0}^{S-1} \sum_{i=0}^{T-1} \gamma_{i,\,j} \zeta^{(inS+jmT)(sT+tS)} =$$

$$= \sum_{j=0}^{S-1} \sum_{i=0}^{T-1} \gamma_{i,\,j} \zeta^{(intS^2+jmsT^2)} = \sum_{j=0}^{S-1} \sum_{i=0}^{T-1} \gamma_{i,\,j} (\zeta^S)^{it} (\zeta^T)^{js} = \sum_{j=0}^{S-1} (\zeta^T)^{js} \gamma_{(j),\,t}, \quad (6)$$

where

$$\gamma_{(j),\,t} = \sum_{i=0}^{T-1} \gamma_{i,\,j} (\zeta^S)^{it}.$$

Comparing the right-hand sides of (5) and (6), we see that multiplication by powers of ζ^{jt} can be avoided in the second case.

In the case when N is prime or when it has no small divisors, a DFT of order N can be reduced to a convolution by the method of L. Bluestein [10] or the method of C. Rader [66].

Bluestein's method requires a root of unity of degree $2N$ in the ring K and works as follows. Let, as above, ζ be a primitive root of order N. Put $\beta = \zeta^{(N-1)/2} \in K$. Note that $\beta^{-2} = \zeta$. Then

$$\gamma^*_j = \sum_{i=0}^{N-1} \gamma_i \zeta^{ij} = \beta^{-j^2} \sum_{i=0}^{N-1} \beta^{(j-i)^2} \beta^{-i^2} \gamma_i = \beta^{-j^2} \sum_{i+k \equiv j \bmod N} U_k V_i = \beta^{-j^2} C_j,$$

where $U_k = \beta^{k^2}$, $V_i = \beta^{-i^2} \gamma_i$. If we define polynomials $u(x) = \sum U_i x^i$, $v(x) = \sum V_i x^i$, $c(x) = \sum C_i x^i$, then $c(x) = u(x)v(x) \bmod (x^N - 1)$. In fact, it is proved

Lemma 6 ([10]). *A DFT of order* N *can be performed by a cyclic convolution of order* N *and* $2N - 2$ *multiplications by roots of unity in the ring* K*.*

An alternative to Bluestein's method is Rader's method. It is applicable in the case of prime N, but does not impose additional requirements on the ring. We retain the notation ζ for a primitive root of degree N in the ring K. Let $g \bmod N$ be a generator of the multiplicative group of the field $GF(N)$. For $k = 1, \ldots, N-1$ we denote $\gamma_{[k]} = \gamma_{g^k \bmod N}$, $\gamma^*_{[k]} = \gamma^*_{g^k \bmod N}$, and $\zeta^{[k]} = \zeta^{g^k}$ for any k. Then

$$\gamma^*_{[j]} = \gamma_0 + \sum_{i=1}^{N-1} \gamma_{[i]} \zeta^{[i+j]}.$$

The sums on the right-hand side of the formula are the components of the cyclic convolution of the vectors $\left(\gamma_{[1]}, \ldots, \gamma_{[N-1]}\right)$ and $\left(\zeta^{[N-1]}, \ldots, \zeta^{[1]}\right)$. The component γ^*_0 is calculated separately as $\sum \gamma_i$. Thus, we obtain

Lemma 7 ([66]). *A DFT of prime order* N *can be performed by a cyclic convolution of order* $N - 1$ *and* $2N - 2$ *additions in the ring* K*.*

Rader's method can be generalized to the case of odd primary numbers $N = p^n$ (p is a prime number). In this case, the DFT reduces to a cyclic convolution of order $(p-1)p^{n-1}$, see, e.g., [9, 59].

Another useful technique was proposed by R. Crandall and B. Fagin [19]. In some cases, it allows to reduce a DFT of "inconvenient" order to a DFT of "convenient" order. For example, multiplication modulo a Mersenne prime $2^p - 1$ coincides with a cyclic convolution of order p of the binary notation vectors of the numbers being multiplied. Therefore, it is implemented by the DFT of order p with a primitive root $2 \in \mathbb{Z}_{2^p-1}$. But p is also a prime number, and DFT of this order is usually not implemented very efficiently.

However, the problem can be reduced to an approximate calculation of the real DFT of arbitrary order N. We split the number $X = [x_{p-1}, \ldots, x_0]$ into N blocks of approximately equal length: $X = [X_{N-1}, \ldots, X_0]$. Let B_i be the starting position of the i-th block. Write X as

$$X = \sum_{i=0}^{N-1} X_i \cdot 2^{B_i} = \sum_{i=0}^{N-1} \left(X_i \cdot 2^{B_i - ip/N} \right) 2^{ip/N} = \sum_{i=0}^{N-1} X'_i \cdot 2^{ip/N}.$$

Now the multiplication of X by $Y = \sum_{i=0}^{N-1} Y'_i \cdot 2^{ip/N}$ can be performed via two DFTs of order N with primitive root $2^{p/N} \in (\mathbb{R} \bmod 2^p - 1)$ and one inverse DFT. From the resulting vector $[Z'_{N-1}, \ldots, Z'_0]$ the desired product is determined as

$$XY = \sum_{i=0}^{N-1} \left(Z'_i \cdot 2^{ip/N - B_i} \right) 2^{B_i} \bmod 2^p - 1.$$

When the block size is chosen close to the length of the machine word, the described method is efficiently implemented on standard computers.

4.3 Multidimensional DFT

The concepts of DFT and FFT algorithms are easily extended to multidimensional domains. Let DFTs of orders $n_1, \ldots, n_d$ be defined in a ring K with primitive roots $\zeta_1, \ldots, \zeta_d$, respectively. In the polynomial interpretation, the input of the d-dimensional transform is some polynomial $\Gamma(x_1, \ldots, x_d) \in K[x_1, \ldots, x_d]$, and the components of a DFT of order $n_1 \times \ldots \times n_d$ over K are all possible values

$$\Gamma\left(\zeta_1^{j_1}, \ldots, \zeta_d^{j_d}\right), \qquad j_1 \in \{0, \ldots, n_1 - 1\}, \ \ \ldots, \ \ j_d \in \{0, \ldots, n_d - 1\}.$$

It is easy to verify that the analogue of Lemma 2 holds: the DFT defines an isomorphism of the rings $K[x_1, \ldots, x_d] / \left((x_1^{n_1} - 1) \cdot \ldots \cdot (x_d^{n_d} - 1) \right)$ and $K^{n_1 \cdot \ldots \cdot n_d}$. The inverse DFT is obtained from the forward one by replacing the primitive roots ζ_i with ζ_i^{-1} and multiplying by $(n_1 \cdot \ldots \cdot n_d)^{-1}$ (the analogue of Lemma 1 takes place).

A straightforward way to compute a multidimensional DFT is as a composition of one-dimensional DFTs component-wise. In the case $d = 2$, write $\Gamma(x_1, x_2) = \sum_{j=0}^{n_2-1} \Gamma_j(x_1) \cdot x_2^j$. With the use of n_2 DFTs of order n_1, find all values $\Gamma_j(\zeta_1^i)$. Apply the DFT of order n_2 to each of the polynomials $\Gamma'_i(x_2) = \sum_{j=0}^{n_2-1} \Gamma_j(\zeta_1^i) \cdot x_2^j$. As a result, we will obtain all values $\Gamma(\zeta_1^i, \zeta_2^j)$. Generalization of the described method leads to the following result.

Lemma 8. *Let $N = n_1 \cdot \ldots \cdot n_d$. A DFT of order $n_1 \times \ldots \times n_d$ can be realized by N/n_1 DFTs of order n_1, N/n_2 DFTs of order n_2, ..., and N/n_d DFTs of order n_d.*

For more details on fast DFT computing, see, for example, [9, 84, 30].

5 Multiplication via FFT

Theorem 1 and Lemma 4 show that the complexity of multiplication of complex polynomials of degree $n = 2^k$ can be estimated as

$$M_{\mathbb{C}}(n) \le (9 + o(1))n \log n. \tag{7}$$

As Schönhage [73] noted, the same estimate is also valid for arbitrary n: in the proof, one should replace n with the nearest number $n' = 2^t r$ from above, where $r = O(\log n)$. In practice, it is preferable to use a DFT of order $2^{\lceil \log n \rceil}$ truncated to n components, see [44, 58, 75, 4].

Of course, bound (7) holds for multiplication over an arbitrary ring containing primitive roots of unity of any order. Special tricks were needed to apply the FFT to multiplication over rings that do not have suitable roots of unity, as well as to multiplication of integers.

A simple ring extension method works well for polynomials with real coefficients: it is enough to switch to calculations over the field $\mathbb{C}$. Moreover, since the complex Fourier transform of a real vector is completely determined by half of its components (the rest are complex conjugates), about half of the operations can be saved, see, e.g., [84, 30].

Since the 1960s, algorithms for computing a complex DFT of order 2^k with real complexity[4] $\sim 4k2^k$ have been known. Such an estimate was first obtained by R. Yavne [88] using a method in which all multiplications are performed by real constants (real-factor FFT), see also [84]. However, the split-radix FFT methods (see [84, 7]) have become more popular — in them, all multiplications are multiplications by roots of unity, which is favorable for error control of the computations. In 2004, J. van Buskirk improved the complexity bound to $\sim (34/9)k2^k$ (the method is published in [47, 55]) by a modification of the split-radix algorithm. Other practical algorithms were proposed in [42, 89], and the best bound to date, $\sim 3\frac{123}{160} \cdot k2^k$, is published in [79].

Thus, the complexity of multiplication of real polynomials is estimated as $M_{\mathbb{R}}(n) \le \left(11\frac{49}{160} + o(1)\right) n \log n$.

The methods mentioned above use $\Theta(k2^k)$ of both additive operations and scalar multiplications. However, it is known [87, 43] that $O(N)$ multiplications are sufficient to implement a DFT of arbitrary order N, but the corresponding algorithms have a higher overall complexity.

The theory of multiplication of real and complex polynomials is adapted to multiplication over the fields $GF(q)$ and $GF(q^2)$, where $q = 2^p - 1$ is a pseudo-Mersenne prime. The field $GF(q^2) \cong GF(q)[x]/(x^2 + 1)$ has primitive roots of order 2^k, $k \le p + 1$, hence, fast algorithms are possible for multiplication of polynomials of degree $< 2^k$. For more details, see [9, 29].

Immediately after the appearance of FFT algorithms, attempts were made to apply them to multiplication of numbers. According to [57], the first fast algorithm for integer multiplication based on the DFT was constructed by N. S. Bakhvalov — his method had complexity $O(n \log^3 n)$.

Then, in 1971, A. Schönhage and V. Strassen [70] published two fast methods of multiplication at once. The first of them exploits the natural idea of transition to calculations over the complex field $\mathbb{C}$, which admits a DFT of arbitrary order.

5.1 Complex approximation method

Let $2N = 2^n q$. In the first Schönhage—Strassen method [70] N-digit numbers to be multiplied are divided into blocks of length q, as in (3). The resulting polynomials are multiplied using a complex DFT of order 2^n. Calculations with complex numbers are performed with an accuracy of s digits after the binary point. The parameter s is chosen so that the coefficients of the product polynomial, which are actually integers, are computed with an error $< 1/2$; then

[4] The symbol "$\sim$" means asymptotic equality.

they can be restored by rounding. A simple analysis (see [70] or [53]) shows that it is sufficient to choose $s = 2n + 2q + \log n + O(1)$.

The main complexity of the algorithm is concentrated in the DFT: $O(2^n n)$ multiplications and additions-subtractions of $O(s)$-digit numbers. The recurrence relation for the complexity of the method has the form

$$M(N) \le O(2^n n(M(s) + s))$$

and is solved as $M(N) = O(N \log N \log\log N \log\log\log N \ldots)$. The same complexity estimate (and probably by the same method) was previously obtained by A. A. Karatsuba [50], but not published.

5.2 Multiplication in ring extensions

Essentially, the second Schönhage—Strassen method [70] is based on reducing the multiplication of polynomials over a ring K that does not have roots of unity of an appropriate order to multiplication in an extension ring. Namely, it is proposed to use the extension $K_{2,n}(x) = K[x]/(x^{2^n} + 1)$, where 2 is supposed to be invertible in K.

In the ring $K_{2,n}(x)$, the DFT of order 2^{n+1} with a primitive root x is defined (for clarity, here and below, instead of elements of the factor ring $K_{2,n}(x)$, which are classes of polynomials equivalent modulo $x^{2^n} + 1$, polynomials-representatives of the classes stand).

Lemma 4 implies

Lemma 9. *A DFT of order 2^k over a ring $K_{2,n}(x)$, $k \le n + 1$, can be implemented in $k2^{k+n}$ additive operations in K.*

For the proof, it suffices to note that multiplication by powers of a primitive root x in the ring $K_{2,n}(x)$ is implemented for free.

The ring $K_{2,n}(x)$ can be considered as an extension of a similar ring $K_{2,m}(y)$ of lower dimension:

Lemma 10. *Let $m < n$. There is an isomorphism*

$$K_{2,n}(x) \cong K_{2,m}(y)[x]/(x^{2^{n-m}} - y),$$

generated by the substitution $x^{2^{n-m}} = y$.

It is important to note that the indicated isomorphism is realized by a simple permutation of the coefficients. For example, the polynomial $x^3 + 2x^2 - 1 \in K_{2,2}(x)$ corresponds to the polynomial $yx + (2y - 1) \in K_{2,1}(y)[x]/(x^2 - y)$. Other isomorphisms can also be used to implement multiplication, see [7].

In the Schönhage—Strassen method for multiplication in the ring $K_{2,n}(x)$, an isomorphism of Lemma 10 is used with the choice of parameter $m = \lceil n/2 \rceil$. Via three DFTs of order $2^{n-m+1} = 2^{\lfloor n/2 \rfloor + 1}$ over the ring $K_{2,m}(y)$, the multiplication is reduced to multiplications in a smaller ring. Recursive application of this procedure leads to the following result.

Theorem 2. *Multiplication in the ring $K_{2,n}(x)$ can be performed using $3 \cdot 2^n n(\log_2 n + O(1))$ additive operations and $O(n2^n)$ multiplications in K.*

Multiplication of polynomials over K can now be performed by a suitable multiplication algorithm in $K_{2,n}(x)$.

To obtain an algorithm for multiplying numbers, instead of the ring $K_{2,n}(x)$, we consider the residue ring $\mathbb{Z}_{F_n}$ modulo the Fermat number $F_n = 2^{2^n} + 1$. Number 2 plays the role of the variable x — it is a primitive root of order 2^{n+1} in the Fermat ring.

The n-bit numbers to be multiplied are divided into blocks of length 2^{m-1}, which are interpreted as coefficients of integer polynomials. The product of the initial numbers is reconstructed from the product of these polynomials. The main part of the integer multiplication method follows the algorithm of Theorem 2. The polynomials are multiplied as polynomials over the ring $\mathbb{Z}_{F_m}$, and therefore the coefficients of the product of the original polynomials are not computed exactly, but modulo $F_m = 2^{2^m} + 1$. In fact, the coefficients of the product of the original polynomials can be of order $2^{n-m} \cdot 2^{2^m}$. Hence, in the method [70], the product of the original polynomials over the residue ring modulo $2^{n-m+O(1)}$ is additionally computed. The desired coefficients are then recovered via the Chinese remainder theorem.

The complexity of the method is determined by the complexity of multiplication over the ring $\mathbb{Z}_{F_m}$, so as a result, the same bound $O(n \log n \log\log n)$ as in Theorem 2 holds for it.

In the same 1971, J. Pollard [65] proposed an ideologically close, but simpler and more practical version of the algorithm. In it, the numbers are divided into 2^{k-1} blocks of length m, corresponding to the length of the machine word. The numerical product is reconstructed from the product of polynomials, performed over several prime moduli p_i such that $\prod p_i \geq 2^{2m+k}$. The moduli are chosen with the condition $p_i \equiv 1 \bmod 2^k$, then the multiplication of polynomials is performed via a DFT of order 2^k over $GF(p_i)$.

The original Schönhage—Strassen method does not allow multiplying polynomials over rings in which two is non-invertible, since there is no DFT of even order defined over such rings. In [71] Schönhage proposed a modified multiplication algorithm for rings in which three is invertible. For such a ring K, the extension $K_{3,n}(x) = K[x]/(x^{2\cdot 3^n} + x^{3^n} + 1)$ is considered — in it x is a primitive root of degree 3^{n+1}. The multiplication is performed in the spirit of Theorem 2, but using DFTs of order of the powers of three.

For the complexity of the specified multiplication method in the ring $K_{3,n}(x)$, an upper bound $13 \cdot 3^n n(\log n + O(1))$ holds, which in the case of a ring of characteristic 2 can be reduced to $10 \cdot 3^n n(\log n + O(1))$ operations in K [31].

The multiplication strategy in the case of $2^{-1}, 3^{-1} \notin K$ is indicated by the Cantor—Kaltofen method [12]. By the method of Theorem 2 and its ternary analogue, replacing the inverse transforms $\mathrm{DFT}^{-1}_{N,\zeta}$ with unnormalized transforms $\mathrm{DFT}_{N,\zeta^{-1}}$, we compute the "almost products"

$$2^{N_1} fg = 2^{N_1} fg \bmod (x^{2^{n_1}} + 1), \quad 3^{N_2} fg = 3^{N_2} fg \bmod (x^{2\cdot 3^{n_2}} + x^{3^{n_2}} + 1)$$

for suitable $n_i, N_i \in \mathbb{N}$, where f, g are the polynomials being multiplied. Finally, the product fg can be determined as $q2^{N_1} fg + s3^{N_2} fg$, where q, s are Bezout coefficients from the relation $q2^{N_1} + s3^{N_2} = 1$. However, the importance of developing fast multiplication algorithms over such rather exotic rings seems insignificant for now.

The applicability of the Schönhage—Strassen integer method has long been in doubt. However, the method is implemented in many modern fast arithmetic libraries and is usually called when the length of multipliers exceeds several thousand digits[5]. Moreover, in the last decade, applications have appeared that employ multiplication of numbers with millions of digits and more, in particular, fully homomorphic encryption (FHE). But at the same time, theoretically faster methods of multiplication have also emerged.

6 The latest methods of multiplication

For a long time, the methods of Schönhage and Strassen remained record-breaking, until in 2007 M. Fürer [26] presented a method for multiplying n-digit numbers with complexity $n \log n \cdot 2^{O(\log^* n)}$. In a certain sense, Fürer's method is a combination of two methods from [70].

[5]Perhaps the skepticism regarding the Schönhage—Strassen method was overcome by its implementation by Schönhage himself and his students on a multi-tape Turing machine emulator in the early 1990s [74].

A year later, a group of Indian mathematicians [20] proposed a modification of Fürer's method with a similar complexity estimate.

It is curious that Fürer back in 1989 [25] constructed a method that had such complexity under the assumption of a sufficiently high distribution density of Fermat primes — but today the hypothesis of finiteness of the set of such numbers is considered more plausible.

In 2014, D. Harvey, J. van der Hoeven, and G. Lecerf [36, 37] constructed a family of multiplication algorithms for both integers and polynomials, with theoretical complexity $O\left(n \log n \cdot 8^{\log^* n}\right)$. In subsequent works [38, 39], the complexity estimates were improved to $O\left(n \log n \cdot 4^{\log^* n}\right)$.

A single iteration of each of the new multiplication methods performs a logarithmic reduction in the problem size: from n to $2^{\log^{O(1)} \log n}$. This is achieved by switching to computations over a ring that admits a compact encoding of elements and contains primitive roots of high and smooth order: for example, a ring of complex polynomials of small degree or a finite field of suitable size.

Unlike previous multiplication methods, the main complexity of the new methods is concentrated in the Fourier transforms, not in the multiplications of Fourier images. More precisely, the complexity of the algorithms is concentrated in the scalar multiplications that arise when the FFT is decomposed into a composition of low-order transforms.

6.1 Fürer's method

Consider the ring $C_p = \mathbb{C}[x]/(x^{2^p}+1)$. In it, x is a primitive root of unity of order 2^{p+1}. The ring also contains primitive roots of unity of arbitrary degree $q2^{p+1}$, which are simultaneously roots (of degree q) of x.

One of such primitive roots $\rho_q(x)$ of degree $q2^{p+1}$ can be constructed as follows. Denote $\zeta = e^{\mathbf{i}\pi/2^p}$ and $\xi = e^{\mathbf{i}\pi/(q2^p)}$. We define the polynomial $\rho_q(x)$ of degree $< 2^p$ by its values at a set of points:

$$\rho_q(\zeta^{2k+1}) = \xi^{2k+1}, \quad k = 0, \ldots, 2^p - 1.$$

As can be verified, this polynomial has an additional important property: the moduli of the coefficients of its powers $\rho_q^m(x)$ in the ring C_p do not exceed 1.

Thus, in the ring C_p, a DFT of order $N = 2^{s(p+1)}$ with primitive root $\rho_* = \rho_{2^{(s-1)(p+1)}}(x)$ is defined, and when implemented by the Cooley-Tukey method, it has the following structure: s layers in which $N2^{-(p+1)}$ DFTs of order 2^{p+1} are performed in parallel alternate with layers in which N multiplications by powers of $\rho_*(x)$ are performed.

The circuit consists of $O(N \log N)$ "light" operations — additions-subtractions and multiplications by powers of x (cyclic shifts), with the help of which the internal DFTs are performed, and $(s-1)N$ "heavy" operations: multiplications by powers of the primitive root.

Let E binary digits be used to write the real and imaginary parts of the complex coefficients of polynomials from C_p. Then the complexity $M(N, E)$ of multiplying polynomials from $C_p[y]$ modulo $y^N - 1$ can be estimated as

$$M(N, E) \leq O(2^p EN \log N) + 3sN \cdot M(1, E).$$

Multiplication in the ring C_p naturally reduces to multiplying $O(2^p(E+p))$-digit numbers.

Due to the accumulation of error (absolute value of error) during the calculations, the accuracy with which the coefficients of the product are computed is, generally speaking, significantly worse than the accuracy of the original coefficients. However, using the boundedness of the coefficients $\rho_*^m(x)$, it is easy to verify that the error increases by a factor of $A^2 N^{O(1)}$, where A is the maximum of the absolute values of the complex coefficients of the original polynomials.

The Fürer method uses a choice of parameters[6] $E \asymp \log N$ and $2^p \asymp \log N$. The numbers

[6]The symbols "$\asymp$" and "$\preceq$" denote equality and inequality of orders.

to be multiplied are represented by polynomials over C_p of degree $< N/2$ as follows: they are divided into blocks of length $E/3$, which are interpreted as integers, each 2^{p-1} consecutive blocks are interpreted as coefficients of a polynomial from the ring $\mathbb{C}[x]/(x^{2^p}+1)$. The precision parameter E is chosen so that the coefficients of the product of the resulting polynomials can be reconstructed by simple rounding.

Thus, for the complexity of multiplication of n-digit numbers, we obtain the relation

$$M(n) \le O(2^p EN \log N) + 3sN \cdot M(O(2^p(E+p))),$$

where $N \asymp n/\log^2 n$, $E \asymp \log n$, $2^p \asymp \log n$, $s \asymp \log n/\log\log n$. It resolves as $M(n) \preceq n \log n \cdot 2^{O(\log^* n)}$.

The method [20] develops Fürer's idea by using suitable extensions of residue rings $\mathbb{Z}_{p^c}$ instead of C_p.

6.2 Harvey—van der Hoeven—Lecerf method

Note that neither Fürer's method [26] nor its p-adic version [20] can be adapted to multiplying polynomials. However, the family of methods proposed in [36, 37] includes both numerical and polynomial methods of multiplication. Let us consider the new approach on the example of multiplying binary polynomials.

Multiplication of binary polynomials can be interpreted as multiplication in the ring $GF(2)[x]/(x^n-1)$. By inserting an appropriate number of zero coefficients, such multiplication reduces to multiplication in the ring $GF(2^k)[y]/(y^N-1)$, where $kN \asymp n$, and $N = N_1 \cdot \ldots \cdot N_d \mid 2^k - 1$ is a smooth number.

The existence of a suitable smooth number N is guaranteed by the following number-theoretic result.

Lemma 11 ([1])**.** *The minimum number $\lambda(t)$ such that*

$$t \le \prod_{p\in\mathbb{P},\ (p-1)\mid\lambda(t)} p,$$

has magnitude $\lambda(t) = (\log t)^{\Theta(\log\log\log t)}$.

According to the lemma, for a given threshold t there exists a number $M \ge t$ — a product of primes $2 < p \le \lambda(t)+1$ with the property $p-1 \mid \lambda(t)$. The last condition, by Fermat's little theorem, means that $p \mid 2^{\lambda(t)}-1$. It can be verified that if $S > \lambda(t)$, then from M one can extract a product $N = N_1 \cdot \ldots \cdot N_d \in [t,\ (\lambda(t)+1)t]$ of mutually prime factors, $S \le N_i \le S^3$.

In the method under consideration, $t \asymp n$ and $S = 2^{\Theta(\log^2\log n)}$ are chosen. The parameter k is chosen to be a multiple of $\lambda(t)$ and has size of order $(\log n)^{\Theta(\log\log\log n)}$. By construction, the size of factors N_i is $2^{\Theta(\log^2\log n)}$.

Next, the multiplication in the ring $GF(2^k)[y]/(y^N-1)$ is performed using a DFT of order N over $GF(2^k)$. The DFT of order N by the Cooley—Tukey method (Lemma 3) or Good's method (Lemma 5) reduces to successively performing DFTs of orders $N_1, \ldots, N_d$. The DFT of order N_i by the Bluestein method (Lemma 6) reduces to the multiplication in the ring $GF(2^k)[y]/(y^{N_i}-1)$.

This multiplication, in turn, is reduced (by inserting zero coefficients) to multiplication in the ring $GF(2)[x]/(x^{n_i}-1)$, where $n_i \asymp kN_i$. Thus, the problem of multiplication of size n is reduced to multiplications of size $n_i = 2^{\Theta(\log^2\log n)}$.

Using the complexity estimate for a DFT of composite order N that follows from Lemma 5, we obtain for the complexity $M_{GF(2)}(n)$ of multiplication in the ring $GF(2)[x]/(x^n-1)$ the recurrence relation

$$M_{GF(2)}(n) \le N \sum_{i=1}^{d} \frac{M_{GF(2)}(n_i)}{N_i} + 2dN M_{GF(2^k)},$$

where $M_{GF(2^k)}$ is the complexity of multiplication in the field $GF(2^k)$ — we estimate it roughly as $O(k \log k \log\log k)$ (Schönhage's method [71]).

Denote $\mu(n) = \frac{M_{GF(2)}(n)}{n}$. Then, taking into account $d \asymp \log n / \log^2 \log n$, we have (for some constant c)

$$\mu(n) \le c \sum_{i=1}^{d} \mu(n_i) + O(d \log k \log\log k) = c \sum_{i=1}^{d} \mu(n_i) + o(\log n).$$

It is easy to verify that this relation is resolved as $\mu(n) = \log n \cdot 2^{O(\log^* n)}$.

To justify the bound $\mu(n) \preceq \log n \cdot 4^{\log^* n}$ in [37, 38], several additional techniques and a more careful analysis are proposed. In particular, the Crandall—Fagin method [19] adapted for finite field extensions is applied.

Similarly, one can perform multiplication over an arbitrary finite field and, with the help of the Cantor—Kaltofen idea [12] (see above), over an arbitrary ring. An accurate estimate [38] says that multiplication of polynomials of degree n over an arbitrary ring K may be performed using $O\left(n \log n \cdot 4^{\log^* n}\right)$ additive operations and $O\left(n \cdot 2^{\log^* n}\right)$ multiplicative operations in K.

Numerical versions [36, 39] of the method follow a general scheme close to the above. Instead of fields $GF(2^k)$, either residue rings $\mathbb{Z}_q$ with a special representation of the elements or a field of complex numbers, in which calculations are performed with controlled precision, and the order of the DFT is chosen to be a power of two, are used.

6.3 Multiplication with complexity $O(n \log n)$

In 2019, D. Harvey and J. van der Hoeven [40, 41] constructed multiplication methods of complexity $O(n \log n)$, and it is quite possible that this result is no longer improvable in order. The new methods are based on reduction to multivariate interpolation.

Indeed, from the multiplication of polynomials of degree $< n$ over a ring K, which admits only FFT of order $m \ll n$, it is easy to pass by Kronecker substitution $x_i = x^{r \cdot 2^i}$ to multiplication in the ring of polynomials $K[x_1, \ldots, x_d]$. For multiplication in this ring, d-dimensional FFT on m^d points is available. But in order for the inverse substitution to allow the product to be restored, the parameter r should be chosen slightly smaller than $m/2$. Therefore, the size of the polynomials being multiplied increases by at least 2^d times. Such growth is unacceptable for constructing theoretically fast methods of multiplication.

The growth of dimension can be avoided by passing to a multidimensional DFT with relatively prime components. Indeed, let $n = n_1 \cdot \ldots \cdot n_d$, where $n_1, \ldots, n_d$ are pairwise relatively prime. Then there is an isomorphism $K[x]/\left(x^n - 1\right) \cong K[x_1, \ldots, x_d]/\left(x_1^{n_1} - 1, \ldots, x_d^{n_d} - 1\right)$ generated by the substitution $x_1 = x^{n/n_1}, \ldots, x_d = x^{n/n_d}$. Thus, a one-dimensional DFT of order n coincides with a d-dimensional DFT of order $n_1 \times \ldots \times n_d$ up to a permutation of components. This connection was observed in [2], and the Good—Thomas method allows to reduce the computation of such a transform to DFTs of orders n_i. In this case, the stumbling block may be the inefficiency of the implementation of DFTs with various orders n_i.

Such way of transition to a multidimensional DFT is employed in the method [40, 41]. As in the Fürer method, n-digit numbers to be multiplied are first divided into $N = O(n/\log n)$ blocks of length of order $\log n$, interpreted as complex coefficients. The numbers are then considered as polynomials over the ring $C_p = \mathbb{C}[y]/(y^{2^p} + 1)$, where $p = \lceil (1/d) \log N \rceil$. Multiplication of such polynomials is performed as multiplication in the ring $C_p[x]/\left(x^{n_1 \cdot \ldots \cdot n_{d-1}} - 1\right)$, where $3N > n_1 \cdot \ldots \cdot n_{d-1} \cdot 2^p > 2N$, and all n_i are distinct primes. The last multiplication, as described above, reduces to computing a DFT of order $n_1 \times \ldots \times n_{d-1}$ and multiplications at the interpolation points. All calculations are performed with a budget of accuracy that can be specified as $O(\log n)$ digits. Let $E = O(\log n)$ digits be used to store the real and imaginary parts of a complex coefficient.

To overcome the problem of inefficiency of DFTs of prime orders n_i, the authors [40, 41] proposed two ways. In the first method, as parameters n_i, numbers of the form $n_i = k_i \cdot 2^p + 1$ are chosen, where the factors k_i are not very large. By a multidimensional version of the Rader method, a DFT of order $n_1 \times \ldots \times n_{d-1}$ is reduced to a cyclic convolution of a smoother order $(n_1 - 1) \times \ldots \times (n_{d-1} - 1)$. In this way, an upper bound $O(n \log n)$ of the complexity of multiplication can be obtained, provided that k_i can be chosen to be really small, say, $k_i < 2^{\varepsilon p}$, where ε is a suitable constant. But the possibility of such a choice relies on an unproven conjecture about the size of primes in an arithmetic progression. If the conjecture is confirmed, then the estimate $O(n \log n)$ of complexity for multiplication of polynomials of degree n will be proved at the same time, see [41].

The second method works only in the numerical case. The primes n_i are chosen to be slightly smaller than powers of two $t_i \in \{2^{p-1}, 2^p\}$, so that $T = t_1 \cdot \ldots \cdot t_{d-1} \leq 2N' = 2\, n_1 \cdot \ldots \cdot n_{d-1}$. The prime number distribution law allows this to be done. The key point of the method is to reduce a DFT of order $n_1 \times \ldots \times n_{d-1}$ to a DFT of order $t_1 \times \ldots \times t_{d-1}$: naturally, we are talking about constructing an approximation with sufficiently high accuracy. The possibility of quickly approximately calculating a DFT of one order using a DFT of another order is not obvious and little known. The issue was studied in [22] for one-dimensional transforms. The authors [40] generalized the method [22] to the multidimensional case and slightly improved it. As a result, it was possible to show that, given certain relations between n_i and t_i (the numbers should not be too close) and some other feasible conditions, the specified reduction can be performed with complexity $o(n \log n)$ and the required accuracy.

Recall that a DFT of order t_i over a ring C_p can be computed by the Cooley—Tukey method in $O(t_i \log t_i)$ operations in C_p, i.e. with the overall complexity $O(t_i \log t_i \cdot 2^p E)$. Then Lemma 8 allows us to estimate the complexity of a DFT of order $t_1 \times \ldots \times t_{d-1}$ as $O(N' \log N' \cdot 2^p E) = O(n \log n)$. Using the notation $M(N, E)$ from §6.1, we arrive at the relation

$$M(n) \leq M(N', E) + O(n) \leq N' \cdot M(1, E) + O(n \log n) \leq N' \cdot M(O(2^p(E + p))) + O(n \log n), \quad (8)$$

estimating the complexity $M(1, E)$ of (approximate) multiplication in the ring C_p as $M(O(2^p(E + p)))$, as in §6.1. Since $N' = O(n \cdot 2^{-p} / \log n)$ and $p \sim (1/d) \log n$, relation (8) is resolved as $M(n) = O(n \log n)$ for a sufficiently large constant parameter d.

This is the general scheme of the computation. The error analysis and the choice of the parameter E are performed in approximately the same way as in the Fürer method. The main difficulty of the proof lies in constructing an approximation of a multidimensional DFT. The multidimensional case is reduced to the one-dimensional one, i.e. computing a DFT of order s via a DFT of order $t > s$, where $(s, t) = 1$. For this, the matrix identity $R \cdot \Pi_s \cdot \Phi_s = \Pi_t \cdot \Phi_t \cdot R'$ is used, where Φ_z are matrices of order-z DFTs, Π_z are suitable permutation matrices of size $z \times z$, and R and R' are special matrices of size $t \times s$. For a fixed vector of arguments $x \in \mathbb{C}^s$, we obtain an overdetermined system of linear equations $A \cdot X = b$ with respect to the vector of unknowns $X = \Phi_s \cdot x$, where $A = R \cdot \Pi_s$ and $b = \Pi_t \cdot \Phi_t \cdot R' \cdot x$. The construction of an approximate solution of the system is facilitated by the peculiarity of the matrices R and R' — the rapid decrease of their coefficients with distance from a certain diagonal. For more details, see [40].

7 Additive DFT and multiplication

DFT-based methods generally do not allow fast multiplication of polynomials over finite fields, since a finite field rarely supports a DFT of sufficiently large smooth order. Theoretically fast methods do not work in a practical range of dimensions. However, it turns out that one can multiply quickly using interpolation at points of additive subgroups of a finite field, and methods based on this turn out to be practically efficient.

A special case of the additive multiplication method was proposed by Y. Wang and X. Zhu [85] in 1988. A year later, D. Cantor [11] constructed a fast multiplication method over fields of order q^{q^k}. Later, J. von zur Gathen and J. Gerhard [32] extended the method to fields of arbitrary order.

Let a basis $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ be chosen in a field $GF(q^k)$. Set $W_0 = \{0\}$ and denote $W_i = \langle \alpha_1, \ldots, \alpha_i \rangle$ — the linear span of the first i basis elements over $GF(q)$. Thus, W_i is an additive subgroup. By construction,

$$W_i = \bigcup_{c \in GF(q)} \{c\alpha_i + W_{i-i}\}.$$

The *additive DFT* (ADFT) of order q^i is defined as the set of values of a polynomial $\Gamma(x) \in GF(q^k)[x]$ at the points of the subgroup W_i. The inverse ADFT is defined as the set of coefficients of a unique polynomial of degree $< q^i - 1$ that takes the given values at W_i.

Define polynomials $S_i(x)$ as

$$S_i(x) = \prod_{\beta \in W_i} (x - \beta). \tag{9}$$

It is easy to prove

Lemma 12. *The following properties of the polynomials $S_i(x)$ hold:*

(i) $S_0(x) \mid S_1(x) \mid \ldots \mid S_k(x) = x^{q^k} - x$;

(ii) $S_{i+1}(x) = S_i^q(x) - S_i^{q-1}(\alpha_{i+1}) S_i(x)$;

(iii) A polynomial $S_i(x)$ is linearized over $GF(q)$, i.e. it has nonzero coefficients only at x^{q^j}, $0 \le j \le i$.

For linearized polynomials, in particular for polynomials $S_i(x)$, the linearity properties hold: $S_i(\beta + \gamma) = S_i(\beta) + S_i(\gamma)$ for any $\beta, \gamma \in GF(q^k)$, and $S_i(c\beta) = cS_i(\beta)$ for any $\beta \in GF(q^k)$ and $c \in GF(q)$.

It can be verified [32] that both the forward and inverse ADFT of order q^i over $GF(q^k)$ may be implemented by circuits over $GF(q^k)$ of complexity $O(i^2 q^{i+1})$.

7.1 Fast ADFT

The ADFT complexity can be reduced by a special choice of basis. Efficient bases for the ADFT over fields $GF\left(q^{q^k}\right)$ were proposed by D. Cantor [11]. Elements of Cantor bases[7] are defined by the relations $\alpha_1 = 1$ and $\alpha_{i+1}^q - \alpha_{i+1} = \alpha_i$ for $i > 1$. Such a choice of basis allowed Cantor to improve the complexity estimate to $O\left(i^{\log_q \binom{q+1}{2}} q^{i+1}\right)$ in the case of a prime number q.

For binary fields, which are of primary interest, faster algorithms have been constructed by S. Gao and T. Mateer [28, 58]. We present a fast ADFT algorithm over the field $GF\left(2^{2^k}\right)$.

The Cantor basis in a binary field is constructed according to the rules: $\alpha_1 = 1$ and $\alpha_{i+1}^2 + \alpha_{i+1} = \alpha_i$ for $i > 1$. Then the characteristic polynomials (9) satisfy the relation $S_{i+1}(x) = S_i^2(x) + S_i(x)$ according to Lemma 12.

The fast Gao—Mateer algorithm makes effective use of the fact that many polynomials S_i are binomials. Namely, $S_{2^i}(x) = x^{2^{2^i}} + x$ and $W_{2^i} \cong GF\left(2^{2^i}\right)$. In other words, the sequence $\{W_i\}$ contains a tower of fields

$$GF(2) \subset GF\left(2^2\right) \subset \ldots \subset GF\left(2^{2^i}\right) \subset \ldots \subset GF\left(2^{2^k}\right).$$

To show the analogy between the fast ADFT algorithm and the FFT, we rewrite the preimage polynomial in a special basis.

[7] The Cantor basis is a special case of the Artin—Schreier basis.

On the set of polynomials of degree $< 2^i$ over $GF(2^k)$ we define a basis $P_i = \{m_b(x) \mid 0 \le b < 2^i\}$ as follows: if in binary notation $b = [b_{i-1}, \ldots, b_0]$, then $m_b(x) = \prod_{j=0}^{i-1} S_j^{b_j}(x)$. By construction, $P_0 \subset P_1 \subset \ldots \subset P_k$. We have

Lemma 13 ([58]). *The transition between the representations of a polynomial of degree $< 2^m$ in the standard basis and the basis P_m in any direction can be performed by a circuit over $GF\left(2^{2^k}\right)$ of $\le 2^{m-2} m \log m$ additive operations.*

The algorithm for transition to the basis P_m involves only operations of division with remainder by binomials, which explains its low complexity. Let $2^i < m = 2^i + t \le 2^{i+1}$. A problem of size 2^m is reduced to 2^{2^i} problems of size 2^t and 2^t problems of size 2^{2^i}. The general scheme is as follows.

1) First, the polynomial $f(x)$ is rewritten as

$$\sum_{j=0}^{2^t-1} f_j(x) S_{2^i}^j(x),$$

$\deg f_j < 2^{2^i}$. This is done by the "bisection" method: first we divide the polynomial $f(x)$ by $S_{2^i}^{2^{t-1}}$, then the quotient and remainder by $S_{2^i}^{2^{t-2}}$, and so on.

2) The inner polynomials $f_j(x)$ are rewritten in the basis P_{2^i}.

3) Assuming $y = S_{2^i}(x)$, the result of the previous step can be written as

$$\sum_{b=0}^{2^{2^i}-1} m_b(x) g_b(y),$$

$\deg g_b < 2^t$. Then, we represent the polynomials g_b in the basis P_t (with respect to the variable y). Note that $S_{2^i+l}(x) = S_l(y)$, so the transition is complete.

When passing from the special basis P_m to the standard one, all steps are implemented in reverse order.

The proof of the following lemma is completely analogous to the proof of Lemma 4.

Lemma 14. *The evaluation of a polynomial $f(x) \in GF\left(2^{2^k}\right)[x]$, written in the basis P_m, at the points $\omega + W_m$, where $\omega \in GF\left(2^{2^k}\right)$, and the inverse problem of reconstructing the polynomial from the given values at these points, is performed by a circuit over $GF\left(2^{2^k}\right)$ containing $m2^m$ additions and $m2^{m-1}$ scalar multiplications. In the case $\omega = 0$, one can additionally save $2^m - 1$ additions and scalar multiplications.*

At the next step of the algorithm, a polynomial $f(x)$ of degree $2^m - 1$ is written as $f_1(x) S_{m-1}(x) + f_0(x)$. Hence,

$$f(\omega + W_{m-1}) = f'(\omega + W_{m-1}), \quad f(\omega + \alpha_m + W_{m-1}) = f''(\omega + \alpha_m + W_{m-1}),$$

where $f' = f_1 S_{m-1}(\omega) + f_0$, $f'' = f_1 S_{m-1}(\omega) + f_0 + f_1$.

When performing the inverse procedure, polynomials f_0, f_1 are restored using the formulas $f_1 = f'' + f'$, $f_0 = f' + S_{m-1}(\omega)(f'' + f')$.

7.2 Multiplication via ADFT

The ADFT allows fast multiplication of polynomials over fields $GF(q^k)$ if the degree of the product is less than q^k. If the field cardinality is small, then the computations are embedded in a field of larger size.

For example, to multiply binary polynomials, the polynomials are divided into blocks of length 2^{k-1}, which are interpreted as elements of the field $GF\left(2^{2^k}\right)$ for a suitable k.

As a result, the multiplication of polynomials of degree N is reduced to $\Theta(N)$ multiplications in a field of degree $O(\log N)$, so the theoretical complexity of the method is $O(N \log N \log^{1+o(1)} \log N)$ — somewhat higher than that of the Schönhage—Strassen method or the method from [37]. However, in practice, the ADFT method has priority. Another reason is that many multiplications in the fast ADFT algorithm are performed on elements of subfields.

In [8], it is shown that this method allows multiplying binary polynomials of degree of order of several hundred faster than any other previously applied method.

Quite recently [46, 54] it was discovered that an additional gain (in the limit about two times, but in practice less) can be obtained by exploiting the Frobenius identity $(a+b)^q = a^q + b^q$, which is valid for any $a, b \in GF(q)$. By virtue of this identity, the value of a polynomial with coefficients from $GF(q)$ at a point $\omega \in GF(q^k)$ automatically determines its values at all points ω^{q^l}. Indeed, if $f \in GF(q)[x]$ and $\omega \in GF(q^k)$, then $f\left(\omega^{q^l}\right) = (f(\omega))^{q^l}$.

The Frobenius automorphism $x \to x^{2^i}$ preserves the additive subgroups W_j defined for the Cantor basis: if $\omega \in W_j$, then $\omega^2 \in W_j$, as follows from the definition of the basis. It can be verified that the Frobenius automorphism group, when acting on an element $\omega \in W_j \setminus W_{j-1} \in GF\left(2^{2^k}\right)$, generates an equivalence class (orbit) $\{\omega^{2^i} \mid i = 0, \ \ldots, 2^k - 1\}$ of cardinality $2^{\lceil \log j \rceil}$. As a consequence, the set of ADFT points of order 2^m over $GF\left(2^{2^k}\right)$ splits into at most $2^m/(m - o(1))$ equivalence classes.

This observation dictates the following method for multiplying polynomials of total degree less than 2^m. The field $GF(2^{2^k})$, $k = \lceil \log m \rceil$, is used for the computations. First, each of the polynomials $f, g \in GF(2)[x]$ to be multiplied, interpreted as a polynomial over $GF\left(2^{2^k}\right)$, is represented in the basis P_m. Partitioning into blocks and insertion of zero blocks are not required. Next, each polynomial is evaluated on the set $\Sigma \subset W_m$ of representatives of the equivalence classes. The algorithm of Lemma 14, adapted for polynomials with coefficients from the subfield $GF(2)$, is applied. The inverse procedure reconstructs the product from the values $f(\omega)g(\omega)$, $\omega \in \Sigma$. For details, see [15].

Note the analogy with the multiplication of polynomials with real coefficients: the presence of the complex conjugation automorphism that preserves the subfield $\mathbb{R}$ and the set of roots of unity allows to compute the complex DFT with a real image or preimage approximately twice as fast as in the general case.

The idea of a reduced DFT over a finite field exploiting the Frobenius identity was proposed by J. van der Hoeven and R. Larrieu in [46]. The additive analogue of the method described above was constructed by a group of mathematicians from Taiwan in 2018 [54]. The upper bounds obtained with its help for the complexity of multiplication of binary polynomials of degree from several hundred and higher are the best of those published at the time of writing this survey.

8 Parallel multiplication circuits

Until now we have measured the quality of algorithms by the total number of operations performed. Now let's consider another important characteristic — the execution time of the

algorithm. In theoretical works, time is usually formalized by the concept of depth.

Depth is the maximum number of elements in a directed input-output chain of a circuit. If complexity corresponds to the size or area of a real electronic microcircuit, then depth corresponds to the speed or response time of the microcircuit. The depth $D(f)$ of a function f (i.e. the minimum depth of the circuit implementing the function) characterizes the time of its parallel computation. The depth of a function that essentially depends on n variables in a basis of binary operations cannot be less than $\log n$.

In addition to Karatsuba's method, the paper [48] presents another result due to Yu. P. Ofman. In modern language, it says that the multiplication of n-digit numbers can be implemented by a Boolean circuit of depth $O(\log n)$.

Multiplication of n-coefficient polynomials is obviously performed with depth $\lceil \log n \rceil + 1$ by the standard method, and this bound is tight. The problem of the integer multiplication depth is more complicated. In the numerical case, even the question of the addition depth is not trivial. However, V. M. Khrapchenko [51] solved it in the asymptotic sense, having constructed an n-digit adder of depth $(1 + o(1)) \log n$. The best, to date, upper bound on the depth of the adder is $\log n + \log \log n + O(1)$ and belongs to M. I. Grinchuk [35].

In the early 1960s, several papers, including [48], showed that the depth of multiplication of n-digit numbers is $O(\log n)$. The proof is based on a reduction to the problem of summing n bits. Indeed, if in the school method of multiplication (see Fig. 1) we perform the additions separately in each column and properly group the bits of the results, the multiplication will reduce to summing $\log n$ numbers. We can continue in the same spirit, reducing the number of terms to $\log \log n$, etc., until only two terms remain, in order to arrive at the relation

$$D(M_n) \le (D(C_n) + D(C_{\log n}) + \ldots) + D(A_{2n}) = D(C_n) + \log n + o(\log n), \qquad (10)$$

where M_n, A_n, and C_n denote the operators of multiplication and addition of n-digit numbers, respectively, and the operator of summation of n bits. The last transition is valid under the condition $D(C_n) = o(n)$. In practical terms, the described strategy is not ideal, but it allows to derive the asymptotically best known estimates of the multiplication depth.

Fast methods for performing multiple addition are based on the idea of compressors.

A binary $(k, l)-$*compressor*[8] of width 1 is a circuit implementing a Boolean operator $(x_1, \ldots, x_k) \to (y_1, \ldots, y_l)$ according to the rule $\sum 2^{a_i} x_i = \sum 2^{b_j} y_j$, where $k > l$, and $a_i, b_j \in \mathbb{Z}$. A $(k, l)-$compressor of arbitrary width is composed from parallel copies of width-1 compressors — it maps k multidigit numbers into l numbers, preserving the sum.

Indeed, k numbers $x^i = [x^i_{n-1}, ..., x^i_0]$, $1 \le i \le k$, by parallel transforms $(x^1_{j+a_1}, \ldots, x^k_{j+a_k}) \to (y^1_{j+b_1}, \ldots, y^l_{j+b_l})$, $j \in \mathbb{Z}$, are converted into l numbers $y^i = [y^i_{n+h}, ..., y^i_0]$, $1 \le i \le l$, where $h < \log k$, while preserving the sum (all undefined digits in the above expressions are assumed to be zero).

The simplest compressor is the $(3, 2)-$compressor. It implements the sum of three bits according to the rule $x_1 + x_2 + x_3 = 2y_2 + y_1$ by the formulas

$$y_2 = x_1(x_2 \oplus x_3) \oplus x_2 x_3, \qquad y_1 = x_1 \oplus x_2 \oplus x_3$$

of complexity 5 and depth 3.

An elementary way to implement the operator C_n in parallel is to construct a tree of $(3, 2)-$compressors of width[9] $\log n$. The tree has n inputs, two $\log n$-bit outputs, and $\lceil \log_{3/2}(n/2) \rceil$ compressor levels. It remains to attach a $\log n$-bit adder to the outputs. Clearly, the depth of the resulting circuit is $O(\log n)$.

[8] In Western literature, the term CSA (carry save adder) is often used.

[9] Note from 2025: here, width is related to compressors.

Methods for fast computation of C_n via compressors were developed and improved in the works of V. M. Khrapchenko [52], M. Paterson, N. Pippenger, U. Zwick [61, 62] (these works also provide a detailed history of the issue). In particular, in [61, 62] a method for optimal placement of given compressors in a circuit was established. Record-breaking results to date were obtained by one of the authors of this review: a constructive bound $D(M_n) \lesssim 4.34 \log n$ [76] and a non-constructive bound $D(M_n) \lesssim 4.02 \log n$ [78]. In the proof of these estimates, the sum of the bits is reconstructed from the remainders modulo 2^k and 3^l, and an approximate sum value calculated with suitable accuracy.

Of both applied and theoretical interest is the construction of parallel versions of fast multiplication methods that do not significantly increase complexity. The considered methods of polynomial multiplication already have logarithmic depth. At least two approaches to parallel rebuilding of numerical algorithms are known — both involve the use of special encoding of numbers in which additive operations are performed with depth $O(1)$. The first method introduces notation in the signed quaternary number system and has been known since the 1960s [5], see also [86]. In the second method, proposed by A. V. Chashkin [14], the number u is encoded by a pair of numbers (u_1, u_2), the difference of which yields the true value $u = u_1 - u_2$.

9 More on the complexity of basic multiplication methods

Despite the development of fast multiplication methods, the standard method and its modifications are still widely used. In practice, the numbers being multiplied are usually relatively small, up to several hundred digits — for such sizes, the standard method is efficient.

So, the essence of the standard method is to reduce multiplication to multiple addition of numbers. The complexity of n-fold addition of n-digit numbers is obviously of order n^2, due to the dimensionality of the problem.

Knowing that $5n-3$ operations are required to add two n-digit numbers [68], it seems natural that summing m numbers requires approximately $5(m-1)n$ operations. Even more unexpected was the result obtained in the work [21] of 2010 by a group of mathematicians from the St. Petersburg Branch of the MI RAS[10]: the trivial estimate for the complexity of m-fold addition can be improved already for $m = 3$. The circuit proposed by the authors is based on a new economical compressor design. Using this compressor, the upper bound for the complexity of the standard multiplication method can be refined to $5.5n^2 - 6.5n - 1 + (n \bmod 2)$ [77].

For the complexity of Karatsuba's method, in which multiple additions and subtractions dictated by formula (2) are optimized taking into account the result [21], the upper bound $K(2^k) \leq 25\frac{83}{405} \cdot 3^k - O(2^k)$ [77] holds.

10 Complexity: quadratic and linear

The initial assumption about the quadratic complexity of multiplication is justified in the computational model of cell circuits. Essentially, a planar cell circuit is a rectangle composed of square cell elements. On a side of a cell there is no more than one input or output, by means of which the cells are connected to each other. For example, a circuit can be composed of functional, switching and separating elements, shown in Fig. 2 (f symbolizes a binary, and g — an unary Boolean function). The inputs and outputs of a circuit are located along the perimeter of the rectangle. The complexity of a cell circuit is the number of elements in it. For more details, see, e.g., [80].

N. A. Shkalikova [80] proved that the order of complexity of multiplying n-digit numbers when implemented by planar cell circuits is n^2. The same is true for three-dimensional cell circuits.

[10] Mathematical Institute, Russian Academy of Sciences.

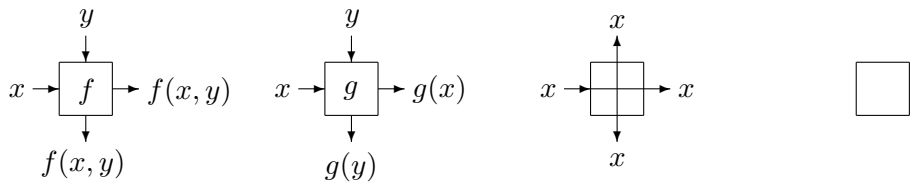

Fig. 2

It is clear that although cell circuits are to some extent similar to real electronic circuits, the relative size of conductors (switching elements) in the model is somewhat overestimated. The lower bound for the complexity of multiplication is obtained from the estimate of the number of conductors connecting the circuit fragments. However, in modern circuit engineering, the density of conductor placement and the density of functional elements are approaching each other.

On the contrary, one can specify computational models in which multiplication is provably performed exceptionally fast, unlike cell circuits. One such model is RAM-programs (alternative notation log-RAM). A RAM-program performs arithmetic operations with arguments of length $\log n$ in unit time, where n is the input size, for more details see [3, 81, 27]. The complexity is measured by the total time of instructions executed. The model is introduced to adequately reflect the process of computation on a general-purpose processor with memory, although, of course, a physical processor is not capable of supporting unit time for executing operations when $n \to \infty$.

A. O. Slisenko [81] and later M. Fürer [27] observed that already the first Schönhage—Strassen method, based on the complex DFT, is implemented by RAM-programs of linear complexity. Indeed, the method reduces the multiplication of n-digit numbers to the DFT of order $\Theta(n/\log n)$ with $O(\log n)$-digit arguments, which is performed in $O(n)$ arithmetic operations with such arguments.

However, the linear complexity of multiplication in the RAM-program model should not be misleading. The model fundamentally allows computations with sublinear complexity. For example, addition of n-digit numbers is implemented by RAM-programs in time $O(n/\log n)$.

Around 1979, A. Schönhage [72] obtained a linear complexity of multiplication in a more sophisticated computational model — storage modification machines, see also [53, 81].

11 Real-time multiplication

In some applications, there is a need for real-time (or online) multiplication algorithms. Real-time means that a k-th digit or coefficient of a multiplier is supplied to the input of an algorithm only after the $(k-1)$-th digit (coefficient) of the product has been computed. Such a situation naturally arises when the next input digits depend on the obtained output digits. This happens, in particular, when solving functional equations by an iterative procedure (see, e.g., [45]).

Let $M^{\mathrm{ONLINE}}(n)$ denote the complexity of real-time multiplication of n-digit numbers by straight-line programs[11]. M. Fischer and L. Stockmeyer [24] obtained the estimate

$$M^{\mathrm{ONLINE}}(2^k) \preceq \sum_{i=0}^{k-1} 2^{k-i} M(2^i) + k2^k$$

[11] Recall that a straight-line program is a circuit with a fixed linear order of operations.

in terms of the complexity of usual multiplication[12]. From this estimate, under the assumption of uniform growth of the function $M(n)$, it follows that $M^{\text{ONLINE}}(n) \preceq M(n)\log n$. If, for example, the Karatsuba method is used for the internal multiplications of the algorithm, then the resulting online version of the Karatsuba method will have the same order of complexity $n^{\log 3}$ as the original method. A similar result is true for the multiplication of polynomials.

J. van der Hoeven showed in a series of papers (see, for example, [45]) that the factor $\log n$ can be reduced under some additional assumptions about the properties of the ring over which the multiplication is performed.

A nontrivial lower bound $M^{\text{ONLINE}}n) = \Omega(n\log n)$ is due to M. Paterson, M. Fischer, and A. Meyer [60].

12 Conclusion

So, the pessimistic hypothesis $M(n) = \Omega(n^2)$ of the 1950s was initially replaced by optimistic expectations: some publications raised the question of the existence of a multiplication method of linear complexity. Today, we can state that, starting with Karatsuba's method and ending with the methods of 2014–2019, the main line of development of multiplication methods — problem size reduction using interpolation — remained unchanged. This path led to the conquest of the $O(n\log n)$ boundary in the integer version of the problem; there is no doubt that in the very near future a similar estimate will be proven for the complexity of polynomial multiplication. Many researchers are inclined to believe that the obtained result in terms of complexity can no longer be improved.

However, some hopes are associated with the development of methods of multiplication of linear multiplicative complexity. *Multiplicative complexity* of multiplication of polynomials from $K[x]$ is the number of nonscalar multiplications in K performed. The question of multiplicative complexity is nontrivial when the cardinality of the ring K is less than the number of coefficients in the product of polynomials, and is especially interesting in the extreme case $K = GF(2)$.

Around 1987, Chudnovsky brothers [16] proposed a theory for constructing multiplication methods over any finite field $GF(q)$ with linear multiplicative complexity. The research was continued in dozens of papers by other authors, see, e.g., the surveys [6, 67]. All the constructed methods use interpolation in groups of divisors on algebraic curves and have a very nontrivial justification[13].

So far, the main results here deal with the refinement of constants in estimates of multiplicative complexity. For example, it has been proven to date that the multiplicative complexity of multiplying polynomials of degree n over $GF(2)$ does not exceed $(35/3 + o(1))n$ [64]. But the overall complexity of the corresponding multiplication algorithms is apparently far from desirable.

On the other hand, nonlinear multiplicative complexity is one of the limiting factors for the known fast multiplication algorithms, since it determines the dimension of linear transforms in the bilinear structure of the algorithms. For comparison, in the methods of Schönhage and Strassen, of order $n\log n$ elementary nonscalar multiplications are performed, in the method [37] — about $n \cdot c^{\log^* n}$, and in the method [41] — about $n\log^c n$.

[12] In [24] the Turing machine computational model was considered, but the bound also holds for straight-line programs.

[13] The complexity of the theory is emphasized by the fact that in all the initial and in many subsequent papers on multiplication methods of linear multiplicative complexity, gaps in the proofs were found.

References

[1] Adleman, L. M., Pomerance, C. & Rumely, R. S. 1983, "On distinguishing prime numbers from composite numbers", The Annals of Mathematics, Second Series, vol. 117, no. 1, pp. 173–206.

[2] Agarwal R. C. & Cooley J. W. 1977, "New algorithms for digital convolution", IEEE Trans. on ASSP, vol. 25, no. 5, pp. 392–410.

[3] Aho, A. V., Hopcroft, J. E. & Ullman J. D. 1976, The design and analysis of computer algorithms, Addison–Wesley, Reading, Mass.

[4] Arnold, A. 2013, "A new truncated Fourier transform algorithm", Proc. Int. Symp. on Symbolic and Algebraic Comp. (Boston, 2013), ACM, NY, pp. 15–22.

[5] Avizienis, A. 1961, "Signed-digit number representations for fast parallel arithmetic", IRE Trans. on Electr. Computers, vol. EC10, pp. 389–400.

[6] Ballet, S., Chaumine, J., Pieltant, J. & Rolland, R. 2011. "On the tensor rank of multiplication in finite fields", arXiv:1107.1184, available at: http://arxiv.org/abs/1107.1184

[7] Bernstein, D. J. 2008, "Fast multiplication and its applications", Algorithmic Number Theory, MSRI Publ., vol. 44, pp. 325–384.

[8] Bernstein, D. J. & Chou, T. 2014, "Faster binary-field multiplication and faster binary-field MACs", Lecture Notes on Comp. Sci., vol. 8781, pp. 92–111.

[9] Blahut, R. E. 1985, Fast algorithms for digital signal processing, Addison–Wesley, Reading, Mass.

[10] Bluestein, L. I. 1970, "A linear filtering approach to the computation of discrete Fourier transform", ECCC Trans. on Audio and Electroacoustics, vol. 18, no. 4, pp. 451–455; 1968, NEREM record, vol. 10, pp. 218–219.

[11] Cantor, D. G. 1989, "On arithmetical algorithms over finite fields", J. Comb. Theory. Series A, vol. 50, pp. 285–300.

[12] Cantor, D. & Kaltofen, E. 1991, "On fast multiplication of polynomials over arbitrary algebras", Acta Inf., vol. 28, no. 7, pp. 693–701.

[13] Cenk, M. & Hasan, M. A. 2015, "Some new results on binary polynomial multiplication", J. Cryptographic Engineering, vol. 5, no. 4, pp. 289–303.

[14] Chashkin, A. V. 2012, Diskretnaya matematika [Discrete mathematics], Akademiya, Moscow, Russia, 2012 (in Russian).

[15] Chen, M.-S., Cheng, C.-M., Kuo, P.-C., Li, W.-D. & Yang, B.-Y. 2018, "Multiplying boolean polynomials with Frobenius partitions in additive fast Fourier transform", arXiv:1803.11301, available at: http://arxiv.org/abs/1803.11301

[16] Chudnovsky, D. V. & Chudnovsky, G. V. 1988, "Algebraic complexities and algebraic curves over finite fields", J. Complexity, vol. 4, pp. 285–316.

[17] Cook, S. 1966, On the minimum computation time of functions, Ph.D. Thesis, Harvard Univ., Cambridge, Mass.

[18] Cooley, J. W. & Tukey J. W. 1965, "An algorithm for the machine calculation of complex Fourier series", Math. Comp., vol. 19, no. 90, pp. 297–301.

[19] Crandall, R. & Fagin, B. 1994, "Discrete weighted transforms and large-integer arithmetic", Math. Comp., vol. 62, no. 205, pp. 305–324.

[20] De, A., Kurur, P. P., Saha, C. & Saptharishi, R. 2013, "Fast integer multiplication using modular arithmetic", SIAM J. Comput., vol. 42, no. 2, pp. 685–699.

[21] Demenkov, E., Kojevnikov, A., Kulikov, A. & Yaroslavtsev G. 2010, "New upper bounds on the Boolean circuit complexity of symmetric functions", Inf. Proc. Letters, vol. 110, no. 7, pp. 264–267.

[22] Dutt, A. & Rokhlin, V. 1993, "Fast Fourier transforms for nonequispaced data", SIAM J. Sci. Comput., vol. 14, no. 6., pp. 1368–1393.

[23] Find, M. G. & Peralta, R. 2019, "Better circuits for binary polynomial multiplication", IEEE Trans. on Comp., vol. 68, no. 4, pp. 624–630.

[24] Fischer, M. J. & Stockmeyer, L. 1974, "Fast on-line integer multiplication", J. Comput. and System Sci., vol. 9, pp. 317–331.

[25] Fürer, M. 1989, On the complexity of integer multiplication (extended abstract), Tech. Report CS-89-17, Pennsylvania State Univ.

[26] Fürer, M. 2009, "Faster integer multiplication", SIAM J. Comput., vol. 39, no. 3, pp. 979–1005.

[27] Fürer, M. 2014, "How fast can we multiply large integers on an actual computer?", Proc. Latin Amer. Symp. on Theor. Inf. (Montevideo, 2014), Lecture Notes on Comp. Sci., vol. 8392, pp. 660–670.

[28] Gao, S. & Mateer, T. 2010, "Additive fast Fourier transforms over finite fields", IEEE Trans. on Information Theory, vol. 56, pp. 6265–6272.

[29] Gashkov, S. B. 2000, "Remarks on the fast multiplication of polynomials, and Fourier and Hartley transforms", Diskret. mat., vol. 12, no. 3, pp. 124–153 (in Russian); 2000, Discrete Math. and Appl., vol. 10, no. 5, pp. 499–528.

[30] Gashkov, S. B. & Sergeev, I. S. 2009, "Fast Fourier transform algorithms", Diskretnaya matematika i ee prilozheniya [Discrete mathematics and its applications], vol. V, Keldysh Inst. Appl. Math., Moscow, Russia, pp. 3–23 (in Russian).

[31] Gashkov, S. B. & Sergeev, I. S. 2013, "On complexity and depth of Boolean circuits for multiplication and inversion over finite fields of characteristic 2", Diskret. mat., vol. 25, no. 1, pp. 3–32 (in Russian); 2013, Discrete Math. and Appl., vol. 23., no. 1, pp. 1–37.

[32] von zur Gathen, J. & Gerhard, J. 1996, "Arithmetic and factorization of polynomials over $\mathbb{F}_2$", Proc. Int. Symp. on Symbolic and Algebraic Comp. (Zürich, 1996), ACM, NY, pp. 1–9.

[33] von zur Gathen, J. & Gerhard, J. 1999, Modern computer algebra, Cambridge Univ. Press, Cambridge.

[34] Good, I. J. 1958, "The interaction algorithm and practical Fourier analysis", J. R. Statist. Soc. B., vol. 20, no. 2, pp. 361–372; 1960, vol. 22, no. 2, pp. 372–375.

[35] Grinchuk, M. I. 2008, "Sharpening an upper bound on the adder and comparator depths", Diskret. analiz i issled. operatsii. Ser. 1, vol. 15, no. 2, pp. 12–22 (in Russian); 2009, J. Appl. and Industr. Math., vol. 3, no. 1, pp. 61–67.

[36] Harvey, D., van der Hoeven, J. & Lecerf, G. 2016, "Even faster integer multiplication", J. Complexity, vol. 36, pp. 1–30.

[37] Harvey, D., van der Hoeven, J. & Lecerf, G. 2017, "Faster polynomial multiplication over finite fields", J. ACM, vol. 63, no. 6, Article 52.

[38] Harvey, D. & van der Hoeven, J. 2017, "Faster integer and polynomial multiplication using cyclotomic coefficient rings", arXiv:1712.03693, available at: http://arxiv.org/abs/1712.03693

[39] Harvey, D. & van der Hoeven, J. 2019, "Faster integer multiplication using short lattice vectors", Proc. 13th Algorithm. Number Theory Symp. (Madison, Wisconsin, 2018), Math. Sci. Publ., Berkeley, pp. 293–310.

[40] Harvey D. & van der Hoeven J. 2019, "Integer multiplication in time $O(n \log n)$", HAL Tech. report no. 02070778, available at: https://hal.archives-ouverts.fr/hal-02070778

[41] Harvey D. & van der Hoeven J. 2019, "Polynomial multiplication over finite fields in time $O(n \log n)$", HAL Tech. report no. 02070816, available at: https://hal.archives-ouverts.fr/hal-02070816

[42] Haynal, S. & Haynal, H. 2011, "Generating and searching families of FFT algorithms", J. Satisf., Boolean Modeling and Comput., vol. 7, no.4, pp. 145–187.

[43] Heideman, M. T. 1986, Applications of multiplicative complexity theory to convolution and the discrete Fourier transform, Ph.D. Thesis, Rice Univ., Houston, Texas.

[44] van der Hoeven, J. 2004, "The truncated Fourier transform and applications", Proc. Int. Symp. on Symbolic and Algebraic Comput. (Santander, 2004), ACM, NY, pp. 290–296.

[45] van der Hoeven, J. 2014, "Faster relaxed multiplication", Proc. Int. Symp. on Symbolic and Algebraic Comp. (Kobe, 2014), ACM, NY, pp. 405–412.

[46] van der Hoeven, J. & Larrieu, R. 2017, "The Frobenius FFT", Proc. Int. Symp. on Symbolic and Algebraic Comp. (Kaiserslautern, 2017), ACM, NY, pp. 437–444.

[47] Johnson, S. F. & Frigo, M. 2007, "A modified split-radix FFT with fewer arithmetic operations", IEEE Trans. Signal Process., vol. 55, no. 1, pp. 111–119.

[48] Karatsuba, A. A. & Ofman, Yu. P. 1962, "Multiplication of multidigit numbers on automata", Doklady Akad. nauk SSSR, vol. 145, no. 2, pp. 293–294 (in Russian); 1963, Soviet Physics Doklady, vol. 7, pp. 595–596.

[49] Karatsuba, A. A. 1995, "The complexity of computations", Trudy MIAN, vol. 211, pp. 186–202 (in Russian); 1995, Proc. Steklov Inst. Math., vol. 211, pp. 169–183.

[50] Karatsuba, A. A. 2013, "Comments to my works, written by myself", Sovrem. problemy matem., vol. 17, pp. 7–29 (in Russian); 2013, Proc. Steklov Inst. Math., vol. 282, suppl. 1, pp. S1–S23.

[51] Khrapchenko, V. M. 1967, "Asymptotic estimation of addition time of a parallel adder", Problemy kibernetiki [Problems of cybernetics], vol. 19, Nauka, Moscow, USSR, pp. 107–120 (in Russian); 1970, Systems Theory Research, vol. 19, pp. 105–122.

[52] Khrapchenko, V. M. 1978, "Some estimates for the multiplication time", Problemy kibernetiki [Problems of cybernetics], vol. 33, Nauka, Moscow, USSR, pp. 221–227 (in Russian).

[53] Knuth, D. E. 1998, The art of computer programming. Vol. 2. Seminumerical algorithms, 3rd ed., Addison–Wesley Longman, Reading, Mass.

[54] Li, W.-D., Chen, M.-S., Kuo, P.-C., Cheng, C.-M. & Yang, B.-Y. 2018, "Frobenius additive fast Fourier transform", Proc. Int. Symp. on Symbolic and Algebraic Comp. (New York, 2018), ACM, NY, pp. 263–270.

[55] Lundy, T. J. & van Buskirk, J. 2007, "A new matrix approach to real FFTs and convolutions of length 2^k", Computing, vol. 80, no. 1, pp. 23–45.

[56] Lupanov, O. B. 1984, Asimptoticheskie otsenki slozhnosti upravlyayuscshikh sistem [Asymptotic bounds on the complexity of control systems], Mosk. Gos. Univ., Moscow, USSR (in Russian).

[57] Lupanov, O. B. 2008, "A. N. Kolmogorov and the circuit complexity theory", Matematicheskie voprosy kibernetiki [Mathematical issues of cybernetics], vol. 17, Fizmatlit, Moscow, Russia, pp. 5–12 (in Russian).

[58] Mateer, T. 2008, Fast Fourier transform algorithms with applications, Ph.D. thesis, Clemson Univ., Clemson, South Car.

[59] Nussbaumer, H. J. 1982, Fast Fourier transform and convolution algorithms", 2nd ed., Springer, Berlin–Heidelberg.

[60] Paterson, M. S., Fischer, M. J. & Meyer, A. R. 1974, An improved overlap argument for on-line multiplication, Tech. Report 40, Project MAC, MIT, Boston, Mass.

[61] Paterson, M. S., Pippenger, N. & Zwick, U. 1992, "Optimal carry save networks", LMS Lecture Notes Series. Boolean function complexity, vol. 169, Cambridge Univ. Press, Cambridge, UK, pp. 174–201.

[62] Paterson, M. & Zwick, U. 1993, "Shallow circuits and concise formulae for multiple addition and multiplication", Comput. Complexity, vol. 3, pp. 262–291.

[63] De Piccoli, A., Visconti, A. & Rizzo, O. G. 2018, "Polynomial multiplication over binary finite fields: new upper bounds", Cryptology ePrint Archive, 2018/901, available at: https://eprint.iacr.org/2018/091.pdf

[64] Pieltant, J. & Randriam, H. 2015, "New uniform and asymptotic upper bounds on the tensor rank of multiplication in extensions of finite fields", Math. Comp., vol. 84, pp. 2023–2045.

[65] Pollard, J. M. 1971, "The Fast Fourier Transform in a finite field", Math. Comp., vol. 25, no. 114, pp. 365–374.

[66] Rader, C. M. 1968, "Discrete Fourier transforms when the number of data samples is prime", Proc. IEEE, vol. 56, pp. 1107–1108.

[67] Randriambololona, H. 2012, "Bilinear complexity of algebras and the Chudnovsky—Chudnovsky interpolation method", J. Complexity, vol. 28, no. 4, pp. 489–517.

[68] Red'kin, N. P. 1981, "On the minimal realization of the binary adder", Problemy kibernetiki [Problems of cybernetics], vol. 38, Nauka, Moscow, USSR, pp. 181–216 (in Russian).

[69] Schönhage, A. 1966, “Multiplikation großer Zahlen”, Computing, vol. 1, no. 3, pp. 182–196.

[70] Schönhage, A. & Strassen, V. 1971, “Schnelle Multiplikation großer Zahlen”, Computing, vol. 7, no. 3–4, pp. 271–282.

[71] Schönhage, A. 1977, “Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2”, Acta Inf., vol. 7, pp. 395–398.

[72] Schönhage, A. 1980, “Storage modification machines”, SIAM J. Comput., vol. 9, no. 3, pp. 490–508.

[73] Schönhage, A. 1982, “Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients”, Proc. Europ. Comput. Algebra Conf. (Marseille, 1982), Lecture Notes on Comp. Sci., vol. 144, pp. 3–15.

[74] Schönhage, A., Grotefeld, A. F. W. & Vetter, E. 1994, Fast algorithms: a multitape Turing machine implementation, BI-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich.

[75] Sergeev, I. S. 2011, “Regular estimates for the complexity of polynomial multiplication and truncated Fourier transform”, Prikladnaya diskretnaya matematika [Applied discrete mathematics], no. 4(14), pp. 72–88 (in Russian).

[76] Sergeev, I. S. 2013, “Upper bounds on the depth of symmetric Boolean functions”, Vestnik Mosk. univ. Ser. 15. Vychislit. matem. i kibern., no. 4, pp. 39–44 (in Russian); 2013, Moscow Univ. Comput. Math. and Cybernetics, vol. 37, no. 4, pp. 195–201.

[77] Sergeev, I. S. 2014, “On the circuit complexity of the standard and the Karatsuba methods of multiplying integers”, Trudy XXII Mezhdunar. nauchn.-tekhn. konf. “Informatsionnye sredstva i tekhnologii” [Proc. 22th Int. Sci.-Tech. Conf. “Information tools and technologies”], vol. 3, Izd. dom MEI, Moscow, Russia, pp. 180–187 (in Russian); 2016, arXiv:1602.02362, available at: http://arxiv.org/abs/1602.02362

[78] Sergeev, I. S. 2016, “Complexity and depth of formulas for symmetric Boolean functions”, Vestnik Mosk. univ. Ser. 1. Matem. Mekh., no. 3, pp. 53–57 (in Russian); 2016, Moscow Univ. Math. Bulletin, vol. 71, no. 3, pp. 127–130.

[79] Sergeev, I. S. 2017, “On the real complexity of a complex DFT”, Problemy peredachi informatsii, vol. 53, no. 3, pp. 90–99 (in Russian); 2017, Problems of Information Transmission, vol. 53, no. 3, pp. 284–293.

[80] Shkalikova, N. A. 1976, “On the complexity of realization of some functions by cell circuits”, Sborn. rabot po matem. kibern. [Selected papers on math. cybernetics], vol. 1, Izd. Vychisl. Ts. AN SSSR, Moscow, USSR, pp. 102–115 (in Russian).

[81] Slisenko A. O. 1981, “Complexity problems in computational theory”, Uspekhi mat. nauk, vol. 36, no. 6(222), pp. 21–103 (in Russian); 1981, Russian Math. Surveys, vol. 36, no. 6, pp. 23–125.

[82] Thomas, L. H. 1963, “Using a computer to solve problem in Physics”, Application in digital computers, Ginn and Co., Boston, Mass.

[83] Toom, A. L. 1963, “The complexity of a scheme of functional elements realizing the multiplication of integers”, Doklady Akad. nauk SSSR, vol. 150, no. 3, pp. 496–498 (in Russian); 1963, Soviet Math. Doklady, vol. 3, pp. 714–716.

[84] Vlasenko, V. A., Lappa, Yu. M. & Jaroslavsky L. P. 1990, Metody sinteza bystrykh algoritmov svertki i spektral'nogo analiza signalov [Methods to design the fast convolution algorithms and spectral signal analysis], Nauka, Moscow, USSR (in Russian).

[85] Wang, Y. & Zhu, X. "A fast algorithm for Fourier transform over finite fields and its VLSI implementation", IEEE Journal on Selected Areas in Communications, vol. 6, no. 3, pp. 572–577.

[86] Wegener, I. 1987, The complexity of Boolean functions, Wiley–Teubner, Stuttgart, FRG.

[87] Winograd, S. 1979, "On the multiplicative complexity of the discrete Fourier transform", Advances in Math., vol. 32, no.2, pp. 83–117.

[88] Yavne, R. 1968, "An economical method for calculating the discrete Fourier transform", Proc. Fall Joint Computer Conf. (San Francisco, 1968). Part I, ACM, NY, pp. 115–125.

[89] Zheng, W., Li, K. & Li, K. 2014, "Scaled radix-2/8 algorithm for efficient computation of length-$N = 2^m$ DFTs", IEEE Trans. Signal Process., vol. 62, no. 10, pp. 2492–2503.