Computing 7, 281–292 (1971)

Fast Multiplication of Large Integers

A. Schönhage, Konstanz
und
V. Strassen, Zürich[1]

(Received on July 8, 1970)

English translation by Ryan Landay
May 22, 2023

Summary

Fast Multiplication of Large Numbers. An algorithm is given for computing the product of two N-digit binary numbers by $O(N \lg N \lg\lg N)$. Two ways of implementing the algorithm are considered: multitape Turing machines and logical nets (with step = binary logical element.)

1 Introduction

The school method for multiplying two decimal numbers can be easily adapted for the multiplication of N-digit binary numbers using a Turing machine with multiple tapes or in a logical network (constructed from two-digit logical elements). In both cases, the computational effort is of the order of N. Here, we define the effort of a network as the number of its elements.

For a Turing machine that performs multiplication of numbers of arbitrary length, the effort of multiplying N-digit numbers is defined as the maximum number of head movements over all input pairs of length N.

The prevailing and intuitively plausible belief that the effort required by the school method cannot be significantly reduced was refuted in 1962 by A. Karatsuba [4], who constructed a network with

$$O\left(N^{\lg 3}\right)$$

elements ($\lg 3 \approx 1.58$). The method easily extends to Turing machines without difficulty.

[1]Part of the research of the second author was done at the Department of Statistics, University of California, Berkeley. He wishes to thank the National Science Foundation for their support (NSF GP-7454).

Another surprising result was presented in the following year by Toom [7], in which a network for multiplying N-digit binary numbers with

$$O(N2^{\,\mathrm{const}\sqrt{\lg N}})$$

logical elements was proposed.

Independently of Toom and using a completely different method, Schönhage [6] showed in 1966 that N-digit numbers can be multiplied on a Turing machine with an effort of

$$O\left(N2^{\sqrt{2\lg N}}(\lg N)^{3/2}\right)$$

Subsequently, the Toom algorithm was also adapted to Turing machines (Cook [1]), albeit with an increased effort of

$$O\left(N2^{\sqrt{2\lg N}}(\lg N)\right)$$

(Cook [1] and Knuth [5], page 273).

The significantly different methods of Toom and Schönhage thus practically yield the same effort, which has led to speculations about its near optimality.

The interested reader is recommended to refer to Chapter 4 of Knuth's brilliant one-man encyclopedia, "The Art of Computer Programming," where the results mentioned here are presented and proven in detail.

In this work, two methods for multiplying N-digit binary numbers are presented, which can be implemented using both logical networks and Turing machines. The effort of one method is:

$$O(N\lg N(\lg\lg N)^{1+\varepsilon}),$$

whilst the effort of the other method is

$$O(N\lg N\lg\lg N).$$

Both methods utilize the fast Fourier transformation (Cooley and Tukey [3]; independently, D. Knuth also had the idea of utilizing the fast Fourier transformation for multiplying large numbers). Its usage is suggested by the fact that the multiplication of two numbers, apart from carrying out the carries, is a convolution. Therefore, if the two binary numbers to be multiplied are arranged as strings of suitable length and these strings are interpreted as elements of a ring R, which allows for performing the necessary calculations with the strings in a faithful representation and also includes the required primitive roots of unity, the desired "large" multiplication can be decomposed into the Fourier transformation of the two string sequences, component-wise multiplication of the transformed sequences, inverse transformation, and carrying out the carries. The resulting "small" multiplications are treated analogously. This leads to a recursive nesting of routines of the described nature.

In our first method, we take $R = \mathbb{C}$ as the field of complex numbers, the string length is approximately $\lg N$, and we achieve an effort of $O(N\lg N(\lg\lg N)^{1+\varepsilon})$ by nesting three times.

In our second method, we use the residue class ring $\mathbb{Z}_{F_n}$ of $\mathbb{Z}$ modulo a Fermat number $2^{2^n}+1$, where the string length is $\approx \sqrt{N}$, and we nest approximately $\lg\lg N$ levels. The crucial advantage of Fermat numbers F_n is that 2 is a primitive 2^{n+1}-th root of unity modulo F_n, and its power residues have extremely simple binary representations, so the multiplication with these roots of unity is negligible.

The implementation of the second method using a logical network can be done with a depth (which determines the time complexity) of $O(\lg N)$. However, we do not go into further detail on this. The order of magnitude of $\lg N$ for the depth is naturally the best possible.

We do not believe that the order of magnitude $N \lg N \lg(\lg N)$ for the effort is optimal, but we suspect that the order of magnitude $N \lg N$ is optimal (cf. the deeper result of [2]). Unfortunately, the online restriction for logical networks is unacceptable, and for computation on Turing machines, it is too strict in any case, as none of the mentioned methods, except for the school method, operates online).

In the next section, we introduce the fast Fourier transformation in the form we will need it later. In the third section, we outline the simpler method using $R = \mathbb{C}$, and in the fourth section, we extensively discuss the method using $R = \mathbb{Z}_{F_n}$.

2 Fast Fourier Transform

Let R be a commutative ring with 1, $w_n \in R$ be a 2^n-th root of unity with $w_n^{2^{n-1}} = -1$, and $2 = 1+1$ be a unit in R.

Then the transformation

$$\hat{a}_k = \sum_{j=0}^{2^n-1} a_j w_n^{jk} \quad (0 \le k < 2^n) \tag{2.1}$$

which assigns to each vector $a \in R^{2n}$ its Fourier transform $\hat{a}$, can be decomposed into individual steps according to the following scheme.

For k and j, the dual representations

$$k = \sum_{v=0}^{n-1} k_v 2^v, \quad j = \sum_{v=0}^{n-1} j_v 2^{n-1-v} \quad (k_v, j_v = 0 \text{ or } 1) \tag{2.2}$$

are used. Starting from

$$A_0(j_0, \ldots, j_{n-1}) = a_j \quad (0 \le j < 2^n) \tag{2.3}$$

$A_1, \ldots, A_n$ are defined recursively using

$$A_{v+}(k_0, \ldots, k_v, j_{v+1}, \ldots, j_{n-1}) =$$

$$= \sum_{j_v=0}^{1} A_v(k_0, \ldots, k_{v-1}, j_v, \ldots, j_{n-1}) w_n^{j_v 2^{n-1-v}(k_v 2^v + \ldots + k_0 2^0)} \tag{2.4}$$

or in a more detailed form, taking into account $w_n^{2^{n-1}} = -1$

$$\left.\begin{aligned} &A_{v+1}(\ldots, k_{v-1}, 0, j_{v+1}, \ldots) = \\ &= A_v(\ldots, k_{v-1}, 0, j_{v+1}, \ldots) + A_v(\ldots, k_{v-1}, 1, j_{v+1}, \ldots) w_n^{\varkappa} \\ &A_{v+1}(\ldots, k_{v-1}, 1, j_{v+1}, \ldots) = \\ &= A_v(\ldots, k_{v-1}, 0, j_{v+1}, \ldots) - A_v(\ldots, k_{v-1}, 1, j_{v+1}, \ldots) w_n^{\varkappa} \end{aligned}\right\} \tag{2.5}$$

with $\varkappa = 2^{n-1-v}(k_{v-1}2^{v-1} + \ldots + k_0 2^0)$.

From (2.3) and (2.4) and considering

$$w_n^{(j_0 2^{n-1} + \ldots + j_{v-1} 2^{n-v})} k_v 2^v = 1$$

one obtains the closed form representation through induction

$$A_{v+1}\left(k_0, \ldots, k_v, j_{v+1}, \ldots, j_{n-1}\right) =$$

$$\sum_{j_v=0}^{1} \ldots \sum_{j_0=0}^{1} a_j w_n^{(j_0 2^{n-1} + \ldots + j_v 2^{n-1-v})(k_v 2^v + \ldots + k_0 2^0)}.$$

According to (2.1) and (2.2), it follows in particular that

$$A_n(k_0, \ldots, k_{n-1}) = \hat{a}_k. \tag{2.6}$$

The rule for the inverse transformation, mapping $\hat{a}$ to a, is obtained by solving (2.5):

$$\left.\begin{aligned} &A_v(\ldots, k_{v-1}, 0, j_{v+1}, \ldots) = \\ &= 2^{-1}\left(A_{v+1}\left(\ldots, k_{v-1}, 0, j_{v+1}, \ldots\right) + A_{v+1}\left(\ldots, k_{v-1}, 1, j_{v+1}, \ldots\right)\right) \\ &A_v(\ldots, k_{v-1}, 1, j_{v+1}, \ldots) = \\ &= 2^{-1} \cdot w_n^{-\varkappa}\left(A_{v+1}\left(\ldots, k_{v-1}, 0, j_{v+1}, \ldots\right) - A_{v+1}\left(\ldots, k_{v-1}, 1, j_{v+1}, \ldots\right)\right) \end{aligned}\right\} \tag{2.7}$$

with $\varkappa = 2^{n-1-v}(k_{v-1}2^{v-1} + \ldots + k_0 2^0)$.

At a later point, the following fact is essential: according to (2.3) and (2.4), specifically for $0 \le j < 2^{n-1}$, we have:

$$\begin{aligned} A_1(1, j_1, \ldots, j_{n-1}) = A_0(0, j_1, , \ldots, j_{n-1}) - A_0(1, j_1, \ldots, j_{n-1}) = \\ = a_j - a_{j+2^{n-1}} \end{aligned} \tag{2.8}$$

Furthermore, the subsequent steps of the recursion (2.5), limited to $k_0 = 1$, lead from these differences to the $\hat{a}_k$ with odd values of k. Similarly, the differences (2.8) can be recovered solely from the 2^{n-1} $\hat{a}_k$ with odd k, using (2.7) (restricted to $k_0 = 1$).

3 Multiplication using complex numbers

In this section, we describe a method for constructing fast multiplication algorithms using the fast Fourier transformation in the ring $R = \mathbb{C}$ of complex numbers. Starting with the elementary method V_0 based on the traditional approach, we use this construction to obtain successively faster algorithms $V_1, V_2, \ldots$ Assuming the existence of V_m, we need to specify how the $2N$-digit product $c = ab$ is computed in the N-digit binary representation for positive integers $a \geq 0$, $b \geq 0$ when using the method V_{m+1}.

Starting with natural numbers l and n, which will later be chosen appropriately depending on N while satisfying the condition

$$l \cdot 2^n \geq 2N \tag{3.1}$$

the factors a and b are decomposed as

$$a = \sum_{j=0}^{2^n-1} a_j 2^{jl}, \quad b = \sum_{j=0}^{2^n-1} b_j 2^{jl} \tag{3.2}$$

$$\text{where } 0 \leq a_j < 2^l,\ 0 \leq b_j < 2^l \text{ and } a_j = b_j = 0 \text{ for } j \geq 2^{n-1}$$

into sections of length l. The product is then expressed as

$$\begin{aligned} c = ab &= \sum_{\tau=0}^{2^n-1} c_\tau 2^{\tau l} \\ \text{with } c_\tau &= \sum_{\varrho+\sigma=\tau} a_\varrho b_\sigma = \sum_{\varrho+\sigma\equiv\tau \pmod{2^n}} a_\varrho b_\sigma, \end{aligned} \tag{3.3}$$

This means that c_τ is obtained by folding the a_ϱ with the b_σ. The Fourier transformation described in section 2, with $w_n = e^{2\pi i \cdot 2^{-n}}$, converts this folding into 2^n multiplications, namely

$$\hat{a}_k \hat{b}_k = \left(\sum_{\varrho=0}^{2^n-1} a_\varrho w_n^{\varrho k} \right) \left(\sum_{\sigma=0}^{2^n-1} b_\sigma w_n^{\sigma k} \right) = \sum_{\tau=0}^{2^n-1} c_\tau w_n^{\tau k} = \hat{c}_k \quad (0 \leq k < 2^n). \tag{3.4}$$

The procedure V_{m+1} can be outlined in broad terms as follows:

(3.5) Transition from the a_ϱ to the $\hat{a}_k$ and correspondingly from the b_σ to the $\hat{b}_k$ according to (2.5);

(3.6) Execution of the 2^n multiplications $\hat{a}_k \hat{b}_k = \hat{c}_k$;

(3.7) Transition from the $\hat{c}_k$ to the c_τ according to (2.7);

(3.8) Position-wise addition of the c_τ according to (3.3).

To realize this plan in a finite manner, it is necessary to replace the complex numbers involved in steps (3.5) to (3.7) with sufficiently accurate numerical approximations. The rounding errors must be kept small enough so that the integer values c_τ can be computed up to an error of less than $\frac{1}{2}$ and determined exactly through rounding. In addition to the $A_v(\ldots)$ from step 2, corresponding quantities $B_v(\ldots)$ and $C_v(\ldots)$ arise when transitioning from b_j to $\hat{b}_k$ and from c_k to c_τ. From $a_j < 2^l$ and $b_j < 2^l$, it follows according to (2.5) and (2.7)

$$|A_v(\ldots)| < 2^{l+v}, |B_v(\ldots)| < 2^{l+v}, |\hat{a}_k| < 2^{l+n}, |\hat{b}_k| < 2^{l+n},$$
$$|\hat{c}_k| < 2^{2l+2n}, |C_v(\ldots)| < 2^{2l+2n}.$$

In order to perform the calculation using complex fixed-point arithmetic with one digit before the decimal point and a suitable number s of digits after the decimal point, we use s-digit

$$\omega_{n,\varkappa} \text{ with } |\omega_{n,\varkappa} - w_n^{\varkappa}| < 2^{-s} \text{ for } |\varkappa| < 2^{n-1}$$

with scaled approximation values

$$\alpha_v(\ldots) \approx 2^{-l-v} A_v(\ldots), \quad \beta_v(\ldots) \approx 2^{-l-v} B_v(\ldots),$$
$$\gamma_v(\ldots) \approx 2^{-2l-2n} C_v(\ldots).$$

From (2.5), (3.6), and (2.7), it is evident how to compute α_v, β_v, and γ_v considering these scalings. The complex multiplications involved are implemented using 4 multiplications of real s-digit numbers following the V_m method, along with additional rounding-based additions to s digits after the decimal point. The estimation of rounding errors yields:

$$\left|2^{2l+2n}\gamma_0(j_0, \ldots, j_{n-1}) - c_j\right| \leq \text{const} \cdot n \cdot 2^{2l+2n-s} < \frac{1}{2},$$

provided that we choose:

$$s \geq 2l + 2n + \lg n + \text{const} \tag{3.9}$$

Now we can estimate the effort $M_{m+1}(N)$ of the V_{m+1} method by referring to $M_m(s)$ and using the complexity $O(s)$ for addition of s-digit numbers. The $\omega_{n,\varkappa}$ values can be permanently incorporated when using logical networks. When using multi-tape Turing machines, they need to be calculated beforehand. They are given by

$$w_1 = -1, \quad w_2 = i, \quad w_{v+1} = \frac{1 + w_v}{|1 + w_v|} \text{ for } v \geq 2,$$
$$w_{v+1}^{2\varkappa} = w_v^{\varkappa}, \quad w_{v+1}^{2\varkappa+1} = w_{v+1} \cdot w_v^{\varkappa}$$

using $O(2^n)$ operations such as addition, multiplication, division, and square root, and achieving accuracy to s digits using elementary methods requires at most $O(2^n \cdot s^2)$ steps.

The steps (8.5) and (3.7) in the previously described implementation incur a complexity of $O(2^n \cdot n(M_m(s)+s))$, (3.6) costs $O(2^n(M_m(s)+s))$, and finally (3.8) adds $O(2^n(2l+n))$. Thus, we have

$$M_{m+1}(N) = O(2^n(s^2 + nM_m(s) + ns + 2l + n))$$

and by choosing the smallest possible s under the condition (3.9), we obtain

$$M_{m+1}(N) = O(2^n((l+n)^2 + nM_m(3(l+n)))). \tag{3.10}$$

For the school method V_0, we have $M_0(s) = O(s^2)$, so

$$M_1(N) = O(2^n \cdot n(l+n)^2).$$

By choosing, while satisfying (3.1),

$$l = \lceil \lg N \rceil, \quad n = \left\lceil \lg \left(\frac{2N}{l} \right) \right\rceil$$

we obtain for the method V_1 the complexity

$$M_1(N) = O(N(\lg N)^2). \tag{3.11}$$

Now we use this result in (3.10) for $m = 1$, with the same choice of l and n, resulting in

$$M_2(N) = O(N \lg N (\lg \lg N)^2).$$

Continuing in this manner, we obtain

$$M_3(N) = O(N \lg N \lg \lg N (\lg \lg \lg N)^2)$$

and so on. We refrain from choosing the nesting degree m in terms of n since we will describe an even faster method in the next section.

4 Use of Fermat numbers

Instead of complex numbers, we now use residue class rings $\mathbb{Z}_{F_n}$ with respect to Fermat numbers

$$F_n = 2^{2^n} + 1.$$

Since the multiplication of N-digit binary numbers $a, b \in \mathbb{Z}$ $(a, b \geq 0)$ can be understood without distortion as multiplication in $\mathbb{Z}_{F_n}$, provided that

$$2N \leq 2^n \tag{4.1}$$

holds and thus $c = ab$ is uniquely determined by

$$c \equiv ab \pmod{F_n} \text{ and } 0 \leq c < F_n \tag{4.2}$$

we can focus on the multiplication in these residue class rings.

The elements of $\mathbb{Z}_{F_n}$ are conveniently represented by fixed-length binary numbers of length 2^{n+1}, i.e., by integers x of the form

$$x = \sum_{j=0}^{2^{n+1}-1} x_j 2^j (x_j = 0 \text{ or } = 1) \tag{4.3}$$

Although this representation is not unique, it has other advantages based on the congruence

$$2^{2^{n+1}} \equiv 1 \pmod{F_n} \tag{4.4}$$

Due to $2^{2^n} \equiv -1 \pmod{F_n}$, the ring $R = \mathbb{Z}_{F_n}$ satisfies the conditions in 2., here with $n+1$ instead of n, and $w_{n+1} = 2$ or with $w_n = 4$. The multiplications with $w_{n+1}^{\varkappa} = 2^{\varkappa}$ required in the Fourier transformation in $\mathbb{Z}_{F_n}$ can be easily realized by cyclically shifting x by $\varkappa$ positions:

$$2^{\varkappa} \cdot x = \sum_{j=0}^{2^{n+1}-1} x_j \cdot 2^{j+\varkappa} \equiv \sum_{k=0}^{2^{n+1}-1} y_k 2^k \pmod{F_n}$$

with $x_j = y_k$ for $j + \varkappa \equiv k \pmod{2^{n+1}}$ The addition modulo F_n of numbers in the form (4.3) also occurs cyclically, meaning an overflow $2^{2^{n+1}}$ is accounted for as a $+1$ in the 0th position. Subtraction can be reduced to addition by cyclically shifting the subtrahend by 2^n positions, thanks to $2^{2^n} \equiv -1 \pmod{F_n}$. These operations require at most a computational effort of $O(2^n)$. Furthermore, we need to solve the following subtasks: reducing

$$x = u + v \cdot 2^{2^n}, \quad 0 \le u < 2^{2^n}, \quad 0 \le v < 2^{2^n} \tag{4.5}$$

to the representation with the minimal non-negative remainder

$$\xi \equiv x \pmod{F_n}, \quad 0 \le \eta \le 2^{2^n}$$

which is achieved by the rule

$$\xi := \begin{cases} u - v, & \text{if } v \le u, \\ 2^{2^n} + 1 + u - v, & \text{if } v > u \end{cases} \tag{4.6}$$

with a computational effort of $O(2^n)$.

The task of computing the unique integer z determined by

$$\begin{cases} z \equiv \xi \pmod{F_n}, & 0 \le \xi \le 2^{2^n}, \\ z \equiv \eta \pmod{2^{n+2}}, & 0 \le \eta < 2^{n+2}, \\ 0 \le z < 2^{n+2} F_n \end{cases} \tag{4.7}$$

can also be solved with a computational effort of $O(2^n)$ by first calculating

$$\delta \equiv \eta - \xi \pmod{2^{n+2}} \text{ with } 0 \le \delta < 2^{n+2}$$

and then computing

$$z = \xi + \delta\left(2^{2^n} + 1\right).$$

After these preparations, we now describe a method for reducing the multiplication in $\mathbb{Z}F_m$ to multiplications in $\mathbb{Z}_{F_n}$, distinguishing the cases

$$m = 2n - 1 \text{ or } m = 2n - 2 \tag{4.8}$$

To ensure $n < m$, let $m \geq 3$.

First, we consider the case $m = 2n - 1$.

The elements to be multiplied from $\mathbb{Z}_{F_n}$ are given in the form (4.3) as 2^{m+1}-digit numbers a and b. By decomposing them as

$$a = \sum_{\varrho=0}^{2^{n+1}-1} a_\varrho 2^{\varrho \cdot 2^{n-1}}, \quad b = \sum_{\sigma=0}^{2^{n+1}-1} b_\sigma 2^{\sigma \cdot 2^{n-1}}, \quad 0 \leq a_\varrho, b_\sigma < 2^{2^{n-1}} \tag{4.9}$$

into 2^{n+1} sections of length 2^{n-1} each, the product takes the form

$$ab \equiv \sum_{\tau=0}^{2^{n+1}-1} c_\tau 2^{\tau 2^{n-1}} \pmod{F_m}$$

where

$$c_\tau = \sum_{\substack{\varrho+\sigma \equiv \tau \pmod{2^{n+1}} \\ 0 \leq \varrho, \sigma < 2^{n+1}}} a_\varrho b_\sigma < 2^{n+1+2^n} \tag{4.10}$$

Due to

$$2^{2^n} \cdot 2^{n-1} = 2^{2^m} \equiv -1 \pmod{F_m}$$

we also have

$$ab \equiv \sum_{j=0}^{2^n-1} \left(c_j - c_{j+2^n} + 2^{n+1+2^n}\right) 2^{j \cdot 2^{n-1}} + \sum_{j=2^n}^{2^{n+1}-1} 2^{n+1+2^n} \cdot 2^{j \cdot 2^{n-1}} \pmod{F_m}$$

where

$$z_j = \begin{cases} c_j - c_{j+2^n} + 2^{n+1+2^n} & \text{for } 0 \leq j < 2^n \\ 2^{n+1+2^n} & \text{for } 2^n \leq j < 2^{n+1} \end{cases} \tag{4.11}$$

thus

$$ab \equiv \sum_{j=0}^{2^{n+1}-1} z_j 2^{j \cdot 2^{n-1}} \pmod{F_m}. \tag{4.12}$$

The term 2^{n+1+2^n} was added with respect to (4.10), so that $0 \leq z_j < 2^{n+2}F_n$ holds.

Since 2^{n+2} and F_n are coprime, it is sufficient to calculate z_j modulo 2^{n+2} and modulo F_n. The calculation of z_j modulo 2^{n+2} can be done easily by the following trick: Using

$$\alpha_j \equiv a_j, \quad \beta_j \equiv b_j \pmod{2^{n+2}}, \quad 0 \leq \alpha_j, \beta_j < 2^{n+2}$$

the numbers

$$u = \sum_{\varrho=0}^{2^{n+1}-1}, \quad v = \sum_{\sigma=0}^{2^{n+1}-1} \beta_\sigma 2^{\sigma(3n+5)},$$

are formed, where

$$u, v < 2^{2^{n+1}(3n+5)}$$

The product of these numbers contains disjoint sections of length $3n+5$ as the sums

$$\gamma_\tau = \sum_{\varrho+\sigma=\tau} \alpha_\varrho \beta_\sigma < 2^{n+1} \cdot \left(2^{n+2}\right)^2, \quad 0 \le \tau < 2^{n+2}.$$

According to (4.10), for $0 \le \tau < 2^{n+1}$ we have

$$c_\tau \equiv \gamma_\tau + \gamma_{\tau+2^{n+1}} \pmod{2^{n+2}},$$

and according to (4.11), we have

$$z_j \equiv \eta_j \pmod{2^{n+2}} \text{ for } 0 \le j < 2^n, \tag{4.14}$$

where

$$\eta_j \equiv \gamma_j - \gamma_{j+2^n} + \gamma_{j+2\cdot 2^n} - \gamma_{j+3\cdot 2^n} \pmod{2^{n+2}} \text{ and } 0 \le \eta_j < 2^{n+2}.$$

The computation of η_j requires at most $O(2^{2n})$ operations, and the multiplication $u \cdot v$ of numbers with a length of at most $2^{n+1}(3n+5)$ has a complexity of

$$M_1(2^{n+1}(3n+5)) = O(2^n \cdot n^3) \le O(2^{2n})$$

as estimated in (3.11). Here, we have used (3.11) for convenience, but any bound of the form $O(N^{2-\varepsilon})$ would suffice, such as the one provided by Karatsuba (see Knuth [5], pages 258–259).

We perform the computation of $z_j \mod F_n$ using the Fourier transform in $\mathbb{Z}_{F_n}$ with $w_{n+1} = 2$, following steps (3.5) to (3.7). However, it is important to note that instead of computing the values of c_j as in (4.11), we only need to compute the 2^n differences $c_j - c_{j+2^n} \pmod{F_n}$. Analogous to (2.8), these are the 2^{n+1}-digit numbers

$$C_1(1, j_1, \ldots, j_n) \equiv c_j - c_{j+2^n} \pmod{F_n} \text{ with } j = j_1 \cdot 2^{n-1} + \ldots + j_n \cdot 2^0,$$

and as mentioned in the conclusion of step 2, for their computation, we only need the 2^n multiplications

$$\hat{a}_k \hat{b}_k \equiv \hat{c}_k \pmod{F_n} \text{ for odd } k < 2^{n+1}$$

The Fourier transform in $\mathbb{Z}_{F_n}$ requires $O(n \cdot 2^n)$ steps, each with a complexity of $O(2^n)$, resulting in a total cost of $O(n \cdot 2^{2n})$. The addition of 2^{n+1+2n} and reduction modulo F_n according to (4.6) yields, with a complexity of $O(2^{2n})$ for $0 \le j < 2^n$, the remainders

$$\xi_j \equiv C_1(1, j_1, \ldots, j_n) + 2^{n+1+2^n} \equiv z_j \pmod{F_n}, \quad 0 \le \xi_j \le 2^{2^n}.$$

Together with (4.14), we have congruences of the form (4.7), from which the 2^n values of z_j ($j < 2^n$) can be computed with a complexity of $O(2^{2n})$. Finally, the digit-wise addition of the z_j in (4.12) requires an additional complexity of $O(2^{2n})$.

In total, we have shown that multiplication in $\mathbb{Z}_{F_{2n-1}}$ can be realized using 2^n multiplications in $\mathbb{Z}_{F_n}$ and an additional complexity of $O(n \cdot 2^{2n})$.

The case of even $m = 2n - 2$ can be treated analogously. The factors a and b are divided into 2^n sections of length 2^{n-1}. Their convolution is reduced to multiplications in $\mathbb{Z}_{F_n}$ using Fourier transformation in $\mathbb{Z}_{F_n}$, now with $w_n = 4$. Once again, only the multiplications for odd k are needed, which in this case are 2^{n-1} in total. Therefore, multiplication in $\mathbb{Z}_{F_{2n-2}}$ can be realized with 2^{n-1} multiplications in $\mathbb{Z}_{F_n}$ and an additional complexity of $O(n \cdot 2^{2n})$.

Now we want to estimate the complexity of our method in closed form. Let $M(n)$ denote the minimum size of logical networks for multiplication in $\mathbb{Z}_{F_n}$ (using the representation (4.3)). According to the previous results, with a sufficiently large γ_0, we have:

$$\begin{cases} M(2n-2) \le 2^{n-1} M(n) + \gamma_0 (n-1) 2^{2n-1} \\ M(2n-1) \le 2^n M(n) + \gamma_0 (n-1) 2^{2n} \end{cases} \quad \text{for } n \ge 3. \tag{4.15}$$

On the other hand, these inequalities can also be interpreted as time estimates for a suitably organized multi-tape Turing machine that works recursively using the method described earlier, where $M(n)$ represents the maximum number of steps required for multiplication in $\mathbb{Z}_{F_n}$.

Let

$$\gamma = \max\{M(1), M(2), M(3), \gamma_0\}$$

then, according to (4.15), by induction on k, we have:

$$M(n) \le \gamma k \cdot 2^{k+n} \text{ for } n \le 2^k + 1,$$

which implies

$$M(n) = O(2^n n \lg n).$$

When applied to the multiplication of N-digit binary numbers, we choose, in accordance with (4.1):

$$n = \lceil \lg(2N) \rceil$$

The reduction modulo F_n required after multiplication in $\mathbb{Z}_{F_n}$, according to (4.6), costs only $O(2^n) = O(N)$. Therefore, multiplication of N-digit numbers can be accomplished with a complexity of:

$$O(N \lg N \lg \lg N)$$

References

[1] Cook, S. A.: *On the Minimum Computation Time of Functions.* Dissertation, Harvard University (1966).

[2] Cook, S. A., and S. O. Aanderaa: *On the Minimum Computation Time of Functions.* Trans. AMS 142, 291–314 (1969).

[3] Cooley, J. W., and J. W. Tukey: *An Algorithm for the Machine Calculation of Complex Fourier Series.* Math. Comp. 19, 297–301 (1965).

[4] Karatsuba, A., and J. Ofman: *Multiplication of Many-Digital Numbers by Automatic Computers (Russian).* Dokl. Akad. Nauk SSSR 145, 293–294 (1962).

[5] Knuth, D. E.: *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms, Chapter 4: Arithmetic.* Addison-Wesley, 1969.

[6] Schönhage, A.: *Multiplikation großer Zahlen.* Computing 1, 182–196 (1966).

[7] Toom, A. L.: *The complexity of a scheme of functional elements realizing the multiplication of integers.* Dokl. Akad. Nauk SSSR 150, 496–498 (1963).