Fast Integer Multiplication Using Modular Arithmetic

Anindya De, Piyush P Kurur*, Chandan Saha
Dept. of Computer Science and Engineering
Indian Institute of Technology, Kanpur
Kanpur, UP, India, 208016
{anindya,ppk,csaha}@cse.iitk.ac.in

Ramprasad Saptharishi†
Chennai Mathematical Institute
Plot H1, SIPCOT IT Park
Padur PO, Siruseri, India, 603103
and
Dept. of Computer Science and Engineering
Indian Institute of Technology, Kanpur
Kanpur, UP, India, 208016
ramprasad@cmi.ac.in

April 4, 2017

Abstract

We give an $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ algorithm for multiplying two N-bit integers that improves the $O(N \cdot \log N \cdot \log\log N)$ algorithm by Schönhage-Strassen [SS71]. Both these algorithms use modular arithmetic. Recently, Fürer [Für07] gave an $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ algorithm which however uses arithmetic over complex numbers as opposed to modular arithmetic. In this paper, we use multivariate polynomial multiplication along with ideas from Fürer's algorithm to achieve this improvement in the modular setting. Our algorithm can also be viewed as a p-adic version of Fürer's algorithm. Thus, we show that the two seemingly different approaches to integer multiplication, modular and complex arithmetic, are similar.

1 Introduction

Computing the product of two N-bit integers is an important problem in algorithmic number theory and algebra. A naive approach leads to an algorithm that uses $O(N^2)$ bit operations. Karatsuba [KO63] showed that some multiplication operations of such an algorithm can be replaced by less costly addition operations which reduces the overall running time of the algorithm to $O(N^{\log_2 3})$ bit operations. Shortly afterwards this result was improved by Toom [Too63] who showed that for any $\varepsilon > 0$, integer multiplication can be done in $O(N^{1+\varepsilon})$ time. This led to the question as

*Research supported through Research I Foundation project NRNM/CS/20030163
†Research done while visiting IIT Kanpur under Project FLW/DST/CS/20060225

to whether the time complexity can be improved further by replacing the term $O(N^\epsilon)$ by a poly-logarithmic factor. In a major breakthrough, Schönhage and Strassen [SS71] gave two efficient algorithms for multiplying integers using fast polynomial multiplication. One of the algorithms achieved a running time of $O(N \cdot \log N \cdot \log\log N \ldots 2^{O(\log^* N)})$ using arithmetic over complex numbers (approximated to suitable precision), while the other used arithmetic modulo carefully chosen integers to improve the complexity further to $O(N \cdot \log N \cdot \log\log N)$. Despite many efforts, the modular algorithm remained the best until a recent remarkable result by Fürer [Für07]. Fürer gave an algorithm that uses arithmetic over complex numbers and runs in $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ time. Till date this is the best time complexity result known for integer multiplication.

Schönhage and Strassen introduced two seemingly different approaches to integer multiplication – using complex and modular arithmetic. Fürer's algorithm improves the time complexity in the complex arithmetic setting by cleverly reducing some costly multiplications to simple shift. However, the algorithm needs to approximate the complex numbers to certain precisions during computation. This introduces the added task of bounding the total truncation errors in the analysis of the algorithm. On the contrary, in the modular setting the error analysis is virtually absent. In addition, modular arithmetic gives a discrete approach to a discrete problem like integer multiplication. Therefore it is natural to ask whether we can achieve a similar improvement in time complexity of this problem in the modular arithmetic setting. In this paper, we answer this question affirmatively. We give an $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ algorithm for integer multiplication using modular arithmetic, thus matching the improvement made by Fürer.

Overview of our result

As is the case in both Schönhage-Strassen's and Fürer's algorithms, we start by reducing the problem to polynomial multiplication over a ring $\mathcal{R}$ by properly encoding the given integers. Polynomials can be multiplied efficiently using Discrete Fourier Transforms (DFT), which uses special roots of unity. For instance, to multiply two polynomials of degree less than M using the Fourier transform, we require a *principal* $2M$-th root of unity (see Definition 2.1 for *principal root*). An efficient way of computing the DFT of a polynomial is through the Fast Fourier Transform (FFT). In addition, if multiplications by these roots are efficient, we get a faster algorithm. Since multiplication by 2 is a shift, it would be good to have a ring with 2 as a root of unity. One way to construct such a ring in the modular setting is to consider rings of the form $\mathcal{R} = \mathbb{Z}/(2^M + 1)\mathbb{Z}$ as is the case in Schönhage and Strassen [SS71]. However, this makes the size of $\mathcal{R}$ equal to 2^M, which although works in case of Schönhage and Strassen's algorithm, is a little too large to handle in our case. We would like to find a ring whose size is bounded by some polynomial in M and which also contains a principal $2M$-th root of unity. In fact, it is this choice of ring that poses the primary challenge in adapting Fürer's algorithm and making it work in the discrete setting. In order to overcome this hurdle we choose the ring to be $\mathcal{R} = \mathbb{Z}/p^c\mathbb{Z}$, for a prime p and a constant c such that $p^c = poly(M)$. The ring $\mathbb{Z}/p^c\mathbb{Z}$, has a principal $2M$-th root of unity if and only if $2M$ divides $p - 1$, which means that we need to search for a prime p in the arithmetic progression $\{1 + i \cdot 2M\}_{i>0}$. To make this search computationally efficient, we need the degree of the polynomials M to be sufficiently small compared to the input size. It turns out that this can be achieved by considering multivariate polynomials instead of univariate polynomials. We use enough variables to make sure that the search for such a prime does not affect the overall running time; the number of variables finally chosen is a constant as well. In fact, the use of multivariate polynomial multiplications and a small ring are the main steps where our algorithm differs from earlier algorithms by Schönhage-Strassen and Fürer.

The use of *inner* and *outer* DFT plays a central role in both Fürer's as well as our algorithm. Towards understanding the notion of inner and outer DFT in the context of multivariate polynomials, we present a group theoretic interpretation of Discrete Fourier Transform (DFT). Arguing along the line of Fürer [Für07] we show that repeated use of efficient computation of inner DFT's using some special roots of unity in $\mathcal{R}$ makes the overall process efficient and leads to an $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ time algorithm.

2 The Ring, the Prime and the Root of Unity

We work with the ring $\mathcal{R} = \mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$ for some m, a constant c and a prime p. Elements of $\mathcal{R}$ are thus $m-1$ degree polynomials over α with coefficients from $\mathbb{Z}/p^c\mathbb{Z}$. By construction, α is a $2m$-th root of unity and multiplication of any element in $\mathcal{R}$ by any power of α can be achieved by shifting operations — this property is crucial in making some multiplications in the FFT less costly (Section 4.2).

Given an N-bit number a, we encode it as a k-variate polynomial over $\mathcal{R}$ with degree in each variable less than M. The parameters M and m are powers of two such that M^k is roughly $\frac{N}{\log^2 N}$ and m is roughly $\log N$. The parameter k will ultimately be chosen a constant (see Section 5). We now explain the details of this encoding.

2.1 Encoding Integers into multivariate Polynomials

Given an N-bit integer a, we first break these N bits into M^k blocks of roughly $\frac{N}{M^k}$ bits each. This corresponds to representing a in base $q = 2^{\frac{N}{M^k}}$. Let $a = a_0 + \ldots + a_{M^k-1}q^{M^k-1}$ where $a_i < q$. The number a is converted into a polynomial as follows:

1. Express i in base M as $i = i_1 + i_2 M + \cdots + i_k M^{k-1}$.
2. Encode each term $a_i q^i$ as the monomial $a_i \cdot X_1^{i_1} \cdots X_k^{i_k}$. As a result, the number a gets converted to the polynomial $\sum_{i=0}^{M^k-1} a_i \cdot X_1^{i_1} \cdots X_k^{i_k}$.

Further, we break each a_i into $\frac{m}{2}$ equal sized blocks where the number of bits in each block is $u = \frac{2N}{M^k \cdot m}$. Each coefficient a_i is then encoded as polynomial in α of degree less than $\frac{m}{2}$. The polynomials are then padded with zeroes to stretch their degrees to m. Thus, the N-bit number a is converted to a k-variate polynomial $a(X)$ over $\mathbb{Z}[\alpha]/(\alpha^m + 1)$.

Given integers a and b, each of N bits, we encode them as polynomials $a(X)$ and $b(X)$ and compute the product polynomial. The product $a \cdot b$ can be recovered by substituting $X_s = q^{M^{s-1}}$, for $1 \leq s \leq k$, and $\alpha = 2^u$ in the polynomial $a(X) \cdot b(X)$. The coefficients in the product polynomial could be as large as $M^k \cdot m \cdot 2^{2u}$ and hence it is sufficient to do arithmetic modulo p^c where $p^c > M^k \cdot m \cdot 2^{2u}$. Therefore, $a(X)$ can indeed be considered as a polynomial over $\mathcal{R} = \mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$. Our choice of the prime p ensures that c is in fact a constant (see Section 5).

2.2 Choosing the prime

The prime p should be chosen such that the ring $\mathbb{Z}/p^c\mathbb{Z}$ has a *principal* $2M$-th root of unity, which is required for polynomial multiplication using FFT. A principal root of unity is defined as follows.

Definition 2.1. *An n-th root of unity $\zeta \in \mathcal{R}$ is said to be* primitive *if it generates a cyclic group of order n under multiplication. Furthermore, it is said to be* principal *if n is coprime to the characteristic of $\mathcal{R}$ and ζ satisfies $\sum_{i=0}^{n-1} \zeta^{ij} = 0$ for all $0 < j < n$.*

In $\mathbb{Z}/p^c\mathbb{Z}$, a $2M$-th root of unity is principal if and only if $2M \mid p-1$ (see also Section 6). As a result, we need to choose the prime p from the arithmetic progression $\{1 + i \cdot 2M\}_{i>0}$, which is the main bottleneck of our approach. We now explain how this bottleneck can be circumvented.

An upper bound for the least prime in an arithmetic progression is given by the following theorem [Lin44]:

Theorem 2.2 (Linnik)**.** *There exist absolute constants ℓ and L such that for any pair of coprime integers d and n, the least prime p such that $p \equiv d \bmod n$ is less than ℓn^L.*

Heath-Brown [HB92] showed that the *Linnik constant* $L \leq 5.5$. Recall that M is chosen such that M^k is $\Theta\left(\frac{N}{\log^2 N}\right)$. If we choose $k = 1$, that is if we use univariate polynomials to encode integers, then the parameter $M = \Theta\left(\frac{N}{\log^2 N}\right)$. Hence the least prime $p \equiv 1 \pmod{2M}$ could be as large as N^L. Since all known deterministic sieving procedures take at least N^L time this is clearly infeasible (for a randomized approach see Section 5.1). However, by choosing a larger k we can ensure that the least prime $p \equiv 1 \pmod{2M}$ is $O(N^\varepsilon)$ for some constant $\varepsilon < 1$.

Remark 2.3. If k is any integer greater than $L+1$, then $M^L = O\left(N^{\frac{L}{L+1}}\right)$ and hence the least prime $p \equiv 1 \bmod 2M$ can be found in $o(N)$ time.

2.3 The Root of Unity

We require a principal $2M$-th root of unity in $\mathcal{R}$ to compute the Fourier transforms. This root $\rho(\alpha)$ should also have the property that its $\left(\frac{M}{m}\right)$-th power is α, so as to make some multiplications in the FFT efficient (Lemma 4.9). Such a root can be computed by interpolation in a way similar to that in Fürer's algorithm [Für07, Section 3], but we briefly sketch the procedure for completeness.

We first obtain a $(p-1)$-th root of unity ζ in $\mathbb{Z}/p^c\mathbb{Z}$ by lifting a generator of $\mathbb{F}_p^*$. The $\left(\frac{p-1}{2M}\right)$-th power of ζ gives us a $2M$-th root of unity ω. A generator of $\mathbb{F}_p^*$ can be computed by brute force, as p is sufficiently small. Having obtained a generator, we can use Hensel Lifting [NZM91, Theorem 2.23].

Lemma 2.4. *Let ζ_s be a primitive $(p-1)$-th root of unity in $\mathbb{Z}/p^s\mathbb{Z}$. Then there exists a unique primitive $(p-1)$-th root of unity ζ_{s+1} in $\mathbb{Z}/p^{s+1}\mathbb{Z}$ such that $\zeta_{s+1} \equiv \zeta_s \pmod{p^s}$. This unique root is given by $\zeta_{s+1} = \zeta_s - \frac{f(\zeta_s)}{f'(\zeta_s)}$ where $f(X) = X^{p-1} - 1$.*

We need the following claims to compute the root $\rho(\alpha)$.

Claim 2.5. *Let ω be a principal $2M$-th root of unity in $\mathbb{Z}/p^c\mathbb{Z}$.*

(a) If $\sigma = \omega^{\frac{M}{m}}$, then σ is a principal $2m$-th root of unity.

(b) The polynomial $x^m + 1 = \prod_{i=1}^{m}(x - \sigma^{2i-1})$ in $\mathbb{Z}/p^c\mathbb{Z}$. Moreover, for any $0 \leq i < j \leq 2m$, the ideals generated by $(x - \sigma^i)$ and $(x - \sigma^j)$ are comaximal in $\mathbb{Z}[x]/p^c\mathbb{Z}$.

(c) *The roots* $\left\{\sigma^{2i-1}\right\}_{1\leq i\leq m}$ *are distinct modulo* p *and therefore the difference of any two of them is a unit in* $\mathbb{Z}[x]/p^c\mathbb{Z}$.

We then, through interpolation, solve for a polynomial $\rho(\alpha)$ such that $\rho(\sigma^{2i+1}) = \omega^{2i+1}$ for all $1 \leq i \leq m$. Then,

$$\begin{aligned} \rho(\sigma^{2i+1}) &= \omega^{2i+1} \quad 1 \leq i \leq m \\ \implies \left(\rho(\sigma^{2i+1})\right)^{M/m} &= \omega^{(2i+1)M/m} = \sigma^{2i+1} \\ \implies (\rho(\alpha))^{M/m} &= \alpha \pmod{\alpha - \sigma^{2i+1}} \quad 1 \leq i \leq m \\ \implies (\rho(\alpha))^{M/m} &= \alpha \pmod{\alpha^m + 1} \end{aligned}$$

The first two parts of the claim justify the Chinese Remaindering. Finally, computing a polynomial $\rho(\alpha)$ such that $\rho(\sigma^{2i+1}) = \omega^{2i+1}$ can be done through interpolation.

$$\rho(\alpha) = \sum_{i=1}^{m} \omega^{2i+1} \frac{\prod_{j\neq i}(\alpha - \sigma^{2j+1})}{\prod_{j\neq i}(\sigma^{2i+1} - \sigma^{2j+1})}$$

The division by $(\sigma^{2i+1} - \sigma^{2j+1})$ is justified as it is a unit in $\mathbb{Z}/p^c\mathbb{Z}$ (part (c) of Claim 2.5).

3 The Integer Multiplication Algorithm

We are given two integers $a, b < 2^N$ to multiply. We fix constants k and c whose values are given in Section 5. The algorithm is as follows:

1. Choose M and m as powers of two such that $M^k \approx \frac{N}{\log^2 N}$ and $m \approx \log N$. Find the least prime $p \equiv 1 \pmod{2M}$ (Remark 2.3).
2. Encode the integers a and b as k-variate polynomials $a(X)$ and $b(X)$ respectively over the ring $\mathcal{R} = \mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$ (Section 2.1).
3. Compute the root $\rho(\alpha)$ (Section 2.3).
4. Use $\rho(\alpha)$ as the principal $2M$-th root of unity to compute the Fourier transforms of the k-variate polynomials $a(X)$ and $b(X)$. Multiply component-wise and take the inverse Fourier transform to obtain the product polynomial.
5. Evaluate the product polynomial at appropriate powers of two to recover the integer product and return it (Section 2.1).

The only missing piece is the Fourier transforms for multivariate polynomials. The following section gives a group theoretic description of FFT.

4 Fourier Transform

A convenient way to study polynomial multiplication is to interpret it as multiplication in a *group algebra*.

Definition 4.1 (Group Algebra)**.** *Let G be a group. The* group algebra *of G over a ring R is the set of formal sums $\sum_{g \in G} \alpha_g g$ where $\alpha_g \in R$ with addition defined point-wise and multiplication defined via convolution as follows*

$$\left(\sum_g \alpha_g g\right)\left(\sum_h \beta_h h\right) = \sum_u \left(\sum_{gh=u} \alpha_g \beta_h\right) u$$

Multiplying univariate polynomials over R of degree less than n can be seen as multiplication in the group algebra $R[G]$ where G is the cyclic group of order $2n$. Similarly, multiplying k-variate polynomials of degree less than n in each variable can be seen as multiplying in the group algebra $R[G^k]$, where G^k denotes the k-fold product group $G \times \ldots \times G$.

In this section, we study the Fourier transform over the group algebra $R[E]$ where E is an *additive abelian group*. Most of this, albeit in a different form, is well known but is provided here for completeness. [Sha99, Chapter 17]

In order to simplify our presentation, we will fix the base ring to be $\mathbb{C}$, the field of complex numbers. Let n be the *exponent* of E, that is the maximum order of any element in E. A similar approach can be followed for any other base ring as long as it has a principal n-th root of unity.

We consider $\mathbb{C}[E]$ as a vector space with basis $\{x\}_{x \in E}$ and use the Dirac notation to represent elements of $\mathbb{C}[E]$ — the vector $|x\rangle$, x in E, denotes the element $1.x$ of $\mathbb{C}[E]$.

Definition 4.2 (Characters)**.** *Let E be an additive abelian group. A* character *of E is a homomorphism from E to $\mathbb{C}^*$.*

An example of a character of E is the trivial character, which we will denote by 1, that assigns to every element of E the complex number 1. If χ_1 and χ_2 are two characters of E then their product $\chi_1.\chi_2$ is defined as $\chi_1.\chi_2(x) = \chi_1(x)\chi_2(x)$.

Proposition 4.3. *[Sha99, Chapter 17, Theorem 1] Let E be an additive abelian group of exponent n. Then the values taken by any character of E are n-th roots of unity. Furthermore, the characters form a* multiplicative abelian group *$\hat{E}$ which is isomorphic to E.*

An important property that the characters satisfy is the following [Isa94, Corollary 2.14].

Proposition 4.4 (Schur's Orthogonality)**.** *Let E be an additive abelian group. Then*

$$\sum_{x \in E} \chi(x) = \begin{cases} 0 & \text{if } \chi \neq 1, \\ \#E & \text{otherwise} \end{cases}$$

$$\sum_{\chi \in \hat{E}} \chi(x) = \begin{cases} 0 & \text{if } x \neq 0, \\ \#E & \text{otherwise.} \end{cases}$$

It follows from Schur's orthogonality that the collection of vectors $|\chi\rangle = \sum_x \chi(x) |x\rangle$ forms a basis of $\mathbb{C}[E]$. We will call this basis the *Fourier basis* of $\mathbb{C}[E]$.

Definition 4.5 (Fourier Transform)**.** *Let E be an additive abelian group and let $x \mapsto \chi_x$ be an isomorphism between E and $\hat{E}$. The* Fourier transform *over E is the linear map from $\mathbb{C}[E]$ to $\mathbb{C}[E]$ that sends $|x\rangle$ to $|\chi_x\rangle$.*

Thus, the Fourier transform is a change of basis from the point basis $\{|x\rangle\}_{x\in E}$ to the Fourier basis $\{|\chi_x\rangle\}_{x\in E}$. The Fourier transform is unique only up to the choice of the isomorphism $x \mapsto \chi_x$. This isomorphism is determined by the choice of the principal root of unity.

Remark 4.6. Given an element $|f\rangle \in \mathbb{C}[E]$, to compute its Fourier transform it is sufficient to compute the *Fourier coefficients* $\{\langle\chi|f\rangle\}_{\chi\in\hat{E}}$.

4.1 Fast Fourier Transform

We now describe the Fast Fourier Transform for general abelian groups in the character theoretic setting. For the rest of the section fix an additive abelian group E over which we would like to compute the Fourier transform. Let A be any subgroup of E and let $B = E/A$. For any such pair of abelian groups A and B, we have an appropriate Fast Fourier transformation, which we describe in the rest of the section.

Proposition 4.7. *1. Every character λ of B can be "lifted" to a character of E (which will also be denoted by λ) defined as follows $\lambda(x) = \lambda(x + A)$.*

2. Let χ_1 and χ_2 be two characters of E that when restricted to A are identical. Then $\chi_1 = \chi_2\lambda$ for some character λ of B.

3. The group $\hat{B}$ is (isomorphic to) a subgroup of $\hat{E}$ with the quotient group $\hat{E}/\hat{B}$ being (isomorphic to) $\hat{A}$.

We now consider the task of computing the Fourier transform of an element $|f\rangle = \sum f_x |x\rangle$ presented as a list of coefficients $\{f_x\}$ in the point basis. For this, it is sufficient to compute the Fourier coefficients $\{\langle\chi|f\rangle\}$ for each character χ of E (Remark 4.6). To describe the Fast Fourier transform we fix two sets of cosets representatives, one of A in E and one of $\hat{B}$ in $\hat{E}$ as follows.

1. For each $b \in B$, b being a coset of A, fix a coset representative $x_b \in E$ such $b = x_b + A$.
2. For each character φ of A, fix a character χ_φ of E such that χ_φ restricted to A is the character φ. The characters $\{\chi_\varphi\}$ form (can be thought of as) a set of coset representatives of $\hat{B}$ in $\hat{E}$.

Since $\{x_b\}_{b\in B}$ forms a set of coset representatives, any $|f\rangle \in \mathbb{C}[E]$ can be written uniquely as $|f\rangle = \sum f_{b,a} |x_b + a\rangle$.

Proposition 4.8. *Let $|f\rangle = \sum f_{b,a} |x_b + a\rangle$ be an element of $\mathbb{C}[E]$. For each $b \in B$ and $\varphi \in \hat{A}$ let $|f_b\rangle \in \mathbb{C}[A]$ and $|f_\varphi\rangle \in \mathbb{C}[B]$ be defined as follows.*

$$\begin{aligned} |f_b\rangle &= \sum_{a\in A} f_{b,a} |a\rangle \\ |f_\varphi\rangle &= \sum_{b\in B} \overline{\chi}_\varphi(x_b)\langle\varphi|f_b\rangle |b\rangle \end{aligned}$$

Then for any character χ of E, which can be expressed as $\chi = \lambda \cdot \chi_\varphi$, the Fourier coefficient $\langle\chi|f\rangle = \langle\lambda|f_\varphi\rangle$.

Proof. Recall that $\lambda(x+A) = \lambda(x)$, and φ is a restriction of the χ to the subgroup A.

$$\begin{aligned}
\langle \chi | f \rangle &= \sum_b \sum_a \overline{\chi}(x_b + a) f_{b,a} \\
&= \sum_b \overline{\lambda}(x_b + a) \sum_a \overline{\chi}_\varphi(x_b + a) f_{b,a} \\
&= \sum_b \overline{\lambda}(b) \overline{\chi}_\varphi(x_b) \sum_a \overline{\varphi}(a) f_{b,a} \\
&= \sum_b \overline{\lambda}(b) \overline{\chi}_\varphi(x_b) \langle \varphi | f_b \rangle \\
&= \langle \lambda | f_\varphi \rangle
\end{aligned}$$

□

We are now ready to describe the Fast Fourier transform given an element $|f\rangle = \sum f_x |x\rangle$.

1. For each $b \in B$ compute the Fourier transforms of $|f_b\rangle$. This requires $\#B$ many Fourier transforms over A.
2. As a result of the previous step we have for each $b \in B$ and $\varphi \in \hat{A}$ the Fourier coefficients $\langle \varphi | f_b \rangle$. Compute for each φ the vectors $|f_\varphi\rangle = \sum_{b \in B} \overline{\chi}_\varphi(x_b) \langle \varphi | f_b \rangle |b\rangle$. This requires $\#\hat{A}.\#B = \#E$ many multiplications by roots of unity.
3. For each $\varphi \in \hat{A}$ compute the Fourier transform of $|f_\varphi\rangle$. This requires $\#\hat{A} = \#A$ many Fourier transforms over B.
4. Any character χ of E is of the form $\chi_\varphi \lambda$ for some $\varphi \in \hat{A}$ and $\lambda \in \hat{B}$. Using Proposition 4.8 we have at the end of Step 3 all the Fourier coefficients $\langle \chi | f \rangle = \langle \lambda | f_\varphi \rangle$.

If the quotient group B itself has a subgroup that is isomorphic to A then we can apply this process recursively on B to obtain a divide and conquer procedure to compute the Fourier transform. In the standard FFT we use $E = \mathbb{Z}/2^n\mathbb{Z}$. The subgroup A is $2^{n-1}E$ which is isomorphic to $\mathbb{Z}/2\mathbb{Z}$ and the quotient group B is $\mathbb{Z}/2^{n-1}\mathbb{Z}$.

4.2 Analysis of the Fourier Transform

Our goal is to multiply k-variate polynomials over $\mathcal{R}$, with the degree in each variable less than M. This can be achieved by embedding the polynomials into the algebra of the product group $E = \left(\frac{\mathbb{Z}}{2M \cdot \mathbb{Z}}\right)^k$ and multiplying them as elements of the algebra. Since the exponent of E is $2M$, we require a principal $2M$-th root of unity in the ring $\mathcal{R}$. We shall use the root $\rho(\alpha)$ (as defined in Section 2.3) for the Fourier transform over E.

For every subgroup A of E, we have a corresponding FFT. We choose the subgroup A as $\left(\frac{\mathbb{Z}}{2m \cdot \mathbb{Z}}\right)^k$ and let B be the quotient group E/A. The group A has exponent $2m$ and α is a principal $2m$-th root of unity. Since α is a power of $\rho(\alpha)$, we can use it for the Fourier transform over A. As multiplications by powers of α are just shifts, this makes Fourier transform over A efficient.

Let $\mathcal{F}(2M, k)$ denote the complexity of computing the Fourier transform over $\left(\frac{\mathbb{Z}}{2M \cdot \mathbb{Z}}\right)^k$. We have

$$\mathcal{F}(2M, k) = \left(\frac{M}{m}\right)^k \mathcal{F}(2m, k) + M^k \mathcal{M}_{\mathcal{R}} + (2m)^k \mathcal{F}\left(\frac{M}{m}, k\right) \tag{1}$$

where $\mathcal{M}_{\mathcal{R}}$ denotes the complexity of multiplications in $\mathcal{R}$. The first term comes from the $\#B$ many Fourier transforms over A (Step 1 of FFT), the second term corresponds to the multiplications by roots of unity (Step 2) and the last term comes from the $\#A$ many Fourier transforms over B (Step 3).

Since A is a subgroup of B as well, Fourier transforms over B can be recursively computed in a similar way, with B playing the role of E. Therefore, by simplifying the recurrence in Equation 1 we get:

$$\mathcal{F}(2M,k) = O\left(\frac{M^k \log M}{m^k \log m}\mathcal{F}(2m,k) + \frac{M^k \log M}{\log m}\mathcal{M}_{\mathcal{R}}\right) \tag{2}$$

Lemma 4.9. $\mathcal{F}(2m,k) = O(m^{k+1} \log m \cdot \log p)$

Proof. The FFT over a group of size n is usually done by taking 2-point FFT's followed by $\frac{n}{2}$-point FFT's. This involves $O(n \log n)$ multiplications by roots of unity and additions in base ring. Using this method, Fourier transforms over A can be computed with $O(m^k \log m)$ multiplications and additions in $\mathcal{R}$. Since each multiplication is between an element of $\mathcal{R}$ and a power of α, this can be efficiently achieved through shifting operations. This is dominated by the addition operation, which takes $O(m \log p)$ time, since this involves adding m coefficients from $\mathbb{Z}/p^c\mathbb{Z}$. □

Therefore, from Equation 2,

$$\mathcal{F}(2M,k) = O\left(M^k \log M \cdot m \cdot \log p + \frac{M^k \log M}{\log m}\mathcal{M}_{\mathcal{R}}\right) \tag{3}$$

5 Complexity Analysis

The choice of parameters should ensure that the following constraints are satisfied:

1. $M^k = \Theta\left(\frac{N}{\log^2 N}\right)$ and $m = O(\log N)$.
2. $M^L = O(N^\varepsilon)$ where L is the Linnik constant (Theorem 2.2) and ε is any constant less than 1. Recall that this makes picking the prime by brute force feasible (see Remark 2.3).
3. $p^c > M^k \cdot m \cdot 2^{2u}$ where $u = \frac{2N}{M^k m}$. This is to prevent overflows during modular arithmetic (see Section 2.1).

It is straightforward to check that $k > L+1$ and $c > 5(k+1)$ satisfy the above constraints. Heath-Brown [HB92] showed that $L \leq 5.5$ and therefore $c = 42$ clearly suffices.

Let $T(N)$ denote the time complexity of multiplying two N bit integers. This consists of:

(a) Time required to pick a suitable prime p,

(b) Computing the root $\rho(\alpha)$,

(c) Encoding the input integers as polynomials,

(d) Multiplying the encoded polynomials,

(e) Evaluating the product polynomial.

As argued before, the prime p can be chosen in $o(N)$ time. To compute $\rho(\alpha)$, we need to lift a generator of $\mathbb{F}_p^*$ to $\mathbb{Z}/p^c\mathbb{Z}$ followed by an interpolation. Since c is a constant and p is a prime of $O(\log N)$ bits, the time required for Hensel Lifting and interpolation is $o(N)$.

The encoding involves dividing bits into smaller blocks, and expressing the exponents of q in base M (Section 2.1) and all these take $O(N)$ time since M is a power of 2. Similarly, evaluation of the product polynomial takes linear time as well. Therefore, the time complexity is dominated by the time taken for polynomial multiplication.

Time complexity of Polynomial Multiplication

From Equation 3, the complexity of Fourier transform is given by

$$\mathcal{F}(2M,k) = O\left(M^k \log M \cdot m \cdot \log p + \frac{M^k \log M}{\log m}\mathcal{M}_{\mathcal{R}}\right)$$

Proposition 5.1. *[Sch82] Multiplication in the ring $\mathcal{R}$ reduces to multiplying $O(\log^2 N)$ bit integers and therefore $\mathcal{M}_{\mathcal{R}} = T\left(O(\log^2 N)\right)$.*

Proof. Elements of $\mathcal{R}$ can be seen as polynomials in α over $\mathbb{Z}/p^c\mathbb{Z}$ with degree at most m. Given two such polynomials $f(\alpha)$ and $g(\alpha)$, encode them as follows: Replace α by 2^d, transforming the polynomials $f(\alpha)$ and $g(\alpha)$ to the integers $f(2^d)$ and $g(2^d)$ respectively. The parameter d is chosen such that the coefficients of the product $h(\alpha) = f(\alpha)g(\alpha)$ can be recovered from the product $f(2^d) \cdot g(2^d)$. For this, it is sufficient to ensure that the maximum coefficient of $h(\alpha)$ is less than 2^d. Since f and g are polynomials of degree m, we would want 2^d to be greater than $m \cdot p^{2c}$, which can be ensured by choosing $d = \Theta(\log N)$. The integers $f(2^d)$ and $g(2^d)$ are bounded by 2^{md} and hence the task of multiplying in $\mathcal{R}$ reduces to $O(\log^2 N)$ bit integer multiplication. □

Multiplication of two polynomials involve a Fourier transform followed by component-wise multiplications and an inverse Fourier transform. Since the number of component-wise multiplications is only M^k, the time taken is $M^k \cdot \mathcal{M}_{\mathcal{R}}$ which is clearly subsumed in $\mathcal{F}(M,k)$. Therefore, the time taken for multiplying the polynomials is $O(\mathcal{F}(M,k))$. Thus, the complexity of our integer multiplication algorithm $T(N)$ is given by,

$$\begin{aligned} T(N) &= O(\mathcal{F}(M,k)) \\ &= O\left(M^k \log M \cdot m \cdot \log p + \frac{M^k \log M}{\log m}\mathcal{M}_{\mathcal{R}}\right) \\ &= O\left(N \log N + \frac{N}{\log N \cdot \log\log N} T(O(\log^2 N))\right) \end{aligned}$$

The above recurrence leads to the following theorem.

Theorem 5.2. *Given two N bit integers, their product can be computed in $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ time.*

5.1 Choosing the Prime Randomly

To ensure that the search for a prime $p \equiv 1 \pmod{2M}$ does not affect the overall time complexity of the algorithm, we considered multivariate polynomials to restrict the value of M; an alternative is to use randomization.

Proposition 5.3. *Assuming ERH, a prime p such that $p \equiv 1 \pmod{2M}$ can be computed by a randomized algorithm with expected running time $\tilde{O}(\log^3 M)$.*

Proof. Titchmarsh [Tit30] (see also Tianxin [Tia90]) showed, assuming ERH, that the number of primes less than x in the arithmetic progression $\{1 + i \cdot 2M\}_{i>0}$ is given by,

$$\pi(x, 2M) = \frac{Li(x)}{\varphi(2M)} + O(\sqrt{x} \log x)$$

for $2M \leq \sqrt{x} \cdot (\log x)^{-2}$, where $Li(x) = \Theta(\frac{x}{\log x})$ and φ is the Euler totient function. In our case, since M is a power of two, $\varphi(2M) = M$, and hence for $x \geq 4M^2 \cdot \log^6 M$, we have $\pi(x, 2M) = \Omega\left(\frac{x}{M \log x}\right)$. Therefore, for an i chosen uniformly randomly in the range $1 \leq i \leq 2M \cdot \log^6 M$, the probability that $i \cdot 2M + 1$ is a prime is at least $\frac{d}{\log x}$ for a constant d. Furthermore, primality test of an $O(\log M)$ bit number can be done in $\tilde{O}(\log^2 M)$ time using Rabin-Miller primality test [Mil76, Rab80]. Hence, with $x = 4M^2 \cdot \log^6 M$ a suitable prime for our algorithm can be found in expected $\tilde{O}(\log^3 M)$ time. □

6 A Different Perspective

Our algorithm can be seen as a p-adic version of Fürer's integer multiplication algorithm, where the field $\mathbb{C}$ is replaced by $\mathbb{Q}_p$, the field of p-adic numbers (for a quick introduction, see Baker's online notes [Bak07]). Much like $\mathbb{C}$, where representing a general element (say in base 2) takes infinitely many bits, representing an element in $\mathbb{Q}_p$ takes infinitely many p-adic digits. Since we cannot work with infinitely many digits, all arithmetic has to be done with finite precision. Modular arithmetic in the base ring $\mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$, can be viewed as arithmetic in the ring $\mathbb{Q}_p[\alpha]/(\alpha^m + 1)$ keeping a precision of $\varepsilon = p^{-c}$.

Arithmetic with finite precision naturally introduces some errors in computation. However, the nature of $\mathbb{Q}_p$ makes the error analysis simpler. The field $\mathbb{Q}_p$ comes with a norm $|\cdot|_p$ called the p-adic norm, which satisfies the stronger triangle inequality $|x + y|_p \leq \max\left(|x|_p, |y|_p\right)$ [Bak07, Proposition 2.6]. As a result, unlike in $\mathbb{C}$, the errors in computation do not compound.

Recall that the efficiency of FFT crucially depends on a special principal $2M$-th root of unity in $\mathbb{Q}_p[\alpha]/(\alpha^m + 1)$. Such a root is constructed with the help of a primitive $2M$-th root of unity in $\mathbb{Q}_p$. The field $\mathbb{Q}_p$ has a primitive $2M$-th root of unity if and only if $2M$ divides $p - 1$ [Bak07, Theorem 5.12]. Also, if $2M$ divides $p - 1$, a $2M$-th root can be obtained from a $(p-1)$-th root of unity by taking a suitable power. A primitive $(p-1)$-th root of unity in $\mathbb{Q}_p$ can be constructed, to sufficient precision, using Hensel Lifting starting from a generator of $\mathbb{F}_p^*$.

7 Conclusion

There are two approaches for multiplying integers, one using arithmetic over complex numbers, and the other using modular arithmetic. Using complex numbers, Schönhage and Strassen [SS71] gave an $O(N \cdot \log N \cdot \log\log N \ldots 2^{O(\log^* N)})$ algorithm. Fürer [Für07] improved this complexity to $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ using some special roots of unity. The other approach, that is modular arithmetic, can be seen as arithmetic in $\mathbb{Q}_p$ with certain precision. A direct adaptation of the Schönhage-Strassen's algorithm in the modular setting leads to an $O(N \cdot \log N \cdot \log\log N \ldots 2^{O(\log^* N)})$ algorithm. In this paper, we showed that by choosing an appropriate prime and a special root of unity, a running time of $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ can be achieved through modular arithmetic as well. Therefore, in a way, we have unified the two paradigms.

Acknowledgement

We thank V. Vinay, Srikanth Srinivasan and the anonymous referees for many helpful suggestions that improved the overall presentation of this paper.

References

[Bak07] Alan J. Baker. An introduction to p-adic numbers and p-adic analysis. *Online Notes*, 2007. http://www.maths.gla.ac.uk/∼ajb/dvi-ps/padicnotes.pdf.

[Für07] Martin Fürer. Faster integer multiplication. *Proceedings of the* 39^{th} *ACM Symposium on Theory of Computing*, pages 57–66, 2007.

[HB92] D. R. Heath-Brown. Zero-free regions for Dirichlet L-functions, and the least prime in an arithmetic progression. In *Proceedings of the London Mathematical Society, 64(3)*, pages 265–338, 1992.

[Isa94] I. Martin Isaacs. *Character theory of finite groups*. Dover publications Inc., New York, 1994.

[KO63] A Karatsuba and Y Ofman. Multiplication of multidigit numbers on automata. *English Translation in Soviet Physics Doklady*, 7:595–596, 1963.

[Lin44] Yuri V. Linnik. On the least prime in an arithmetic progression, I. The basic theorem, II. The Deuring-Heilbronn's phenomenon. *Rec. Math. (Mat. Sbornik)*, 15:139–178 and 347–368, 1944.

[Mil76] G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.

[NZM91] Ivan Niven, Herbert S. Zuckerman, and Hugh L. Montgomery. *An Introduction to the Theory of Numbers*. John Wiley and Sons, Singapore, 1991.

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[Sch82] Arnold Schönhage. Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients. In *Computer Algebra, EUROCAM*, volume 144 of *Lecture Notes in Computer Science*, pages 3–15, 1982.

[Sha99] Igor R. Shafarevich. *Basic Notions of Algebra*. Springer Verlag, USA, 1999.

[SS71] A Schönhage and V Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[Tia90] Cai Tianxin. Primes representable by polynomials and the lower bound of the least primes in arithmetic progressions. *Acta Mathematica Sinica, New Series*, 6:289–296, 1990.

[Tit30] E. C. Titchmarsh. A divisor problem. *Rend. Circ. Mat. Palerme*, 54:414–429, 1930.

[Too63] A L. Toom. The complexity of a scheme of functional elements simulating the multiplication of integers. *English Translation in Soviet Mathematics*, 3:714–716, 1963.