Electronic Colloquium on Computational Complexity, Report No. 23 (2008)

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 603103, India
ramprasad@cmi.ac.in

November 18, 2007

Abstract

We give an $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ algorithm for multiplying two N-bit integers using modular computation that matches the running time of the current best algorithm due to Fürer [Fur07]. Fürer uses arithmetic over complex numbers whereas the best known algorithm using modular computation has a complexity of $O(N \cdot \log N \cdot \log \log N)$ [SS71]. Hence, in the modular setting, our algorithm is an improvement over Schönage-Strassen [SS71]. We also argue that our algorithm can be viewed as a p-adic version of Fürer's algorithm. Thus, the two seemingly different paradigms of computation, modular and complex arithmetic, are essentially similar in the context of integer multiplication.

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

ISSN 1433-8092

1 Introduction

Given two N-bit integers a and b, computing their product is an important algorithmic problem in number theory and algebra. Karatsuba and Ofman [KO63] gave the first non-trivial integer multiplication algorithm with a running time of $O(N^{\log_2 3})$. Shortly afterwards, Toom [Too63] showed that for any $\varepsilon > 0$, integer multiplication can be done in $O(N^{1+\varepsilon})$ time.

In a major breakthrough, Schönage and Strassen [SS71] devised an $O(N \cdot \log N \cdot \log\log N)$ algorithm for integer multiplication. Despite many efforts, this remained the best until Fürer [Fur07] gave an algorithm which runs in $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ time. Though the main ingredient in both these algorithms is a reduction to polynomial multiplication and subsequent use of Fast Fourier transform (FFT), they differ in their choice of the base ring. While the former used polynomials over $\mathbb{Z}/(2^k + 1)\mathbb{Z}$ and modular arithmetic, the later used polynomials over rings of the form $\mathbb{C}[\alpha]/(\alpha^m + 1)$ with approximate arithmetic over $\mathbb{C}$.

Our Contribution

In this paper, we give an $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ algorithm that uses modular arithmetic. An advantage of using modular arithmetic over complex arithmetic is that the error analysis is simplified (Section 6). However, there is a slight complication in choosing a suitable modulus as explained below.

We follow the general paradigm of reducing the task to polynomial multiplication. The integers are encoded as polynomials over the ring $\mathcal{R} = \mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$ for appropriate c, m and a prime p, and multiplied as polynomials. Multiplication of polynomials using FFT requires a suitable root of unity in $\mathcal{R}$. This imposes a constraint $p \equiv 1 \pmod{2M}$, where the degrees of the polynomials are less than M. A naive search for such a prime will impose an overhead of $O(N^{1+\varepsilon})$ for some $\varepsilon > 0$. Therefore, it is not sufficient to just replace the ring $\mathbb{C}[\alpha]/(\alpha^m + 1)$, used in Fürer's algorithm, by $\mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$.

One possible way to avoid this overhead is to pick p randomly. In fact, using the Extended Riemann Hypothesis (ERH), one can give such a randomized algorithm with the same expected running time (Section 5.1). However, this is unsatisfactory because the algorithm is both randomized and conditional. We circumvent this difficulty of picking the prime p by considering multivariate polynomials and using an appropriate FFT.

Our algorithm crucially depends on the the choice of the base ring with the help of a special prime, and computation of roots of unity in the ring. The following section is devoted to the study of these objects.

2 The Ring, the Prime and the Roots of Unity

Given an N-bit integer a, we encode it as a multivariate polynomial over the ring $\mathcal{R} = \mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$ for $m = O(\log N)$, a constant c and a prime p. Elements of $\mathcal{R}$ are thus $m-1$ degree polynomials over α each of whose coefficients are elements of $\mathbb{Z}/p^c\mathbb{Z}$. 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 less costly. The number a is converted into a k-variate polynomial over $\mathcal{R}$ with degree in each variable less than M. The parameter M is chosen such that the total degree M^k of the polynomial is $\Theta\left(\frac{N}{\log^2 N}\right)$.

2.1 Encoding Integers into k-variate Polynomials

The number $a < 2^N$, given in binary, is first converted into base p. The choice of parameters (see Section 5) ensures that the total number of digits would be $\frac{tm}{2} \cdot M^k$ where t is a constant. We divide these digits into M^k blocks of $\frac{tm}{2}$ digits. This corresponds to a representation of a in base $q = p^{\frac{tm}{2}}$. Let $a = a_0 + \ldots + a_{M^k-1}q^{M^k-1}$ where $a_i < q$. Every q^i is converted into a monomial as follows:

1. Express i in base M as $i = i_1 + i_2M + \cdots + i_kM^{k-1}$.
2. Encode each term a_iq^i as the monomial $a_i \cdot X_1^{i_1}X_2^{i_1} \cdots X_k^{i_k}$. As a result, the number a gets converted to the polynomial $a(X) = \sum_{i=0}^{M^k-1} a_i \cdot X_1^{i_1} \cdots X_k^{i_k}$.

The next step is to convert each a_i into an element in the ring $\mathcal{R}$. This is done by representing each a_i in base p^t and interpreting the number as a polynomial in α of degree less than $m/2$ evaluated at $\alpha = p^t$. The polynomials are then padded with zeroes to stretch their degrees to less than m.

Our algorithm proceeds as follows: Given integers a and b, each of N bits, we encode them as polynomials $a(X)$ and $b(X)$ and compute the product polynomial using FFT. The product $a \cdot b$ can be recovered by substituting X_s by $q^{M^{s-1}}$, for $1 \leq s \leq k$, and α by p^t in the polynomial $a(X) \cdot b(X)$.

The coefficients in the product polynomial could be as large as $p^{2t} \cdot \frac{m}{2} \cdot M^k$ and hence to avoid overflows we consider them as elements of $\mathbb{Z}/p^c\mathbb{Z}$ instead of $\mathbb{Z}/p^t\mathbb{Z}$ for some constant $c > t$. The precise values of the parameters are given in Section 5.

Polynomial multiplication using FFT requires a *principal* $2M$-th root of unity in $\mathcal{R}$.

Definition 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.

2.2 Finding the Prime

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

Theorem 2.1 (Linnik)**.** *There exist absolute constants ℓ and L such that for any pair of coprime integers d and n, the least prime such that $p \equiv d \bmod n$ is less than ℓn^L.*

Heath-Brown [HB92] showed that $L \leq 5.5$. 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, which implies that a naive search is infeasible. 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 2.2. If $k \geq L + 1$, then $M^L = O\left(N^{\frac{L}{L+1}}\right)$ and hence the least prime $p \equiv 1 \pmod{2M}$ can be found in $o(N)$ time.

2.3 The Root of Unity

We require a principal $2M$-th root of unity $\rho(\alpha)$ in $\mathcal{R}$ to compute the Fourier transforms. This root $\rho(\alpha)$ should also have the property that an appropriate power of it is α so as to make some multiplications in the FFT efficient. The root $\rho(\alpha)$ can be computed by interpolation in a way similar to that in

Fürer's algorithm [Fur07, Section 3], except that we need a principal $2M$-th of unity ω in $\mathbb{Z}/p^c\mathbb{Z}$ to start with. To obtain such a root, 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 the required $2M$-th root of unity ω. A generator of $\mathbb{F}_p^*$ can be computed by bruteforce, as p is sufficiently small. Having obtained a generator, we use Hensel Lifting [NZM91, Theorem 2.23].

Lemma 2.3. *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 $f(X) = X^{p-1} - 1$.*

3 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 2 such that $M^k = \Theta\left(\frac{N}{\log^2 N}\right)$ and $m \approx 2\log N$. Find the least prime $p \equiv 1 \pmod{2M}$ (Remark 2.2).

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 pricipal $2M$-th root to compute the Fourier transform 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 p 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 2 (Group Algebra)**.** *Let G be any 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 in the literature 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. Then 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 Hilbert space with orthonormal 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 3 (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. Let χ_1 and χ_2 be 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 4.1. *[Sha99, Chapter 17, Theorem 1] Let E be an additive abelian group with exponent n. Then the values taken by any character of E is an n-th root 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 4.2 (Schur's Orthogonality)**.** *Let E be an additive abelian group. Then*

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

It follows from Schur's orthogonality that the collection of vectors $|\chi\rangle = \frac{1}{\sqrt{\#E}} \sum_x \chi(x) \, |x\rangle$ forms a orthonormal basis of $\mathbb{C}[E]$. We will call the basis $|\chi\rangle$ the *Fourier basis* of $\mathbb{C}[E]$.

Definition 4 (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 (in fact unitary) map from $\mathbb{C}[E]$ to $\mathbb{C}[E]$ that sends $|x\rangle$ to $|\chi_x\rangle$.*

Thus Fourier transform is a change of basis from the point basis $\{|x\rangle\}_{x \in E}$ to the Fourier basis $\{|\chi_x\rangle\}_{x \in E}$.

Remark 4.3. The Fourier transform is unique only upto the choice of the isomorphism $x \mapsto \chi_x$. 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. We need the following property about characters of an abelian group.

Proposition 4.4.

1. *Every character λ of B can be "lifted" uniquely 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 4.3). To describe the Fast Fourier transform we fix two sets of cosets representatives, one of B in E and one of $\hat{A}$ 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 resricted to A is the character φ. The characters $\{\chi_\varphi\}$ form (can be thought of as) a set of coset representatives of the the quotient group $\hat{A} = \hat{E}/\hat{B}$ 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 4.5. *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.*

$$|f_b\rangle = \sum_{a\in A} f_{b,a} |a\rangle \; ; \; |f_\varphi\rangle = \sum_{b\in B} \overline{\chi}_\varphi(x_b)\langle\varphi|f_b\rangle |b\rangle \tag{1}$$

Then for any character $\chi = \chi_\varphi\lambda$ of E the Fourier coefficient $\langle\chi|f\rangle = \langle\lambda|f_\varphi\rangle$.

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

1. Compute for each $b \in B$ 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 $|f_\varphi\rangle$. 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 4.5 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 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 can use $\rho(\alpha)$ as the principal root for the Fourier transform over E.

For every subgroup A of E, we have a corresponding FFT. We choose the subgroup A as $\left(\frac{\mathbb{Z}}{2m\cdot\mathbb{Z}}\right)^k$, which has exponent $2m$. Let B be the quotient group E/A. 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 $\mathcal{F}(M,k)$ denote the complexity of computing the Fourier transform over $E = \left(\frac{\mathbb{Z}}{2M\cdot\mathbb{Z}}\right)^k$. We have

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

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 we get:

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

Lemma 4.6. $\mathcal{F}(m,k) = O(m^{k+1}\log m \cdot \log p)$

Proof. The FFT over a group of size n is traditionally done by taking 2-point FFT's followed by $\frac{n}{2}$-point FFT's. This involves $O(n \log n)$ operations in the base ring. Using this method, Fourier transforms over A can be computed with $O(m^k \log m)$ multiplications and additions in $\mathcal{R}$. Each multiplication is between an element of $\mathcal{R}$ and a power of α, which 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 $\mathbb{Z}/p^c\mathbb{Z}$. $\square$

Therefore,

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

5 Complexity Analysis

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

1. $M^k = \Theta\left(\frac{N}{\log^2 N}\right)$ and $m = O(\log N)$. This is to ensure that we get the desired time complexity.
2. $M^L = O(N^\varepsilon)$ for some constant $\varepsilon < 1$. Recall that this makes picking the prime by brute force feasible (see Remark 2.2).
3. $m = 2\log N$ and $2^N \leq p^{M^k \cdot t \cdot \frac{m}{2}}$. This is required to encode N-bit integers (see Section 2.1).
4. $p^c > p^{2t} \cdot M^k \cdot \frac{m}{2}$. This is to prevent overflows during modular arithmetic (see Section 2.1).

It is straightforward to check that $k = \lfloor L+1 \rfloor$, $t = k+1$ and $c = 3t$ satisfy the three constraints.

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

1. Time required to pick a suitable prime p,
2. Computing the root $\rho(\alpha)$,
3. Computing the Fourier transforms.

As argued before, 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 poly-logarithmic. Both these terms are dominated by the time required for computing Fourier transform.

Time complexity of Fourier transform

In Section 4 we showed that the complexity of Fourier transform is given by

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

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

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. ☐

Therefore, the complexity of our 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 $p \equiv 1 \pmod{M}$ 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 $p \equiv 1 \pmod{M}$ can be computed by a randomized algorithm with expected running time $\tilde{O}(\log^3 M)$.*

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

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

for $M \leq \sqrt{x} \cdot (\log x)^{-2}$, where $Li(x) = \Theta(\frac{x}{\log x})$ and φ is the Euler totient function. In our case, $\varphi(M) = M/2$ since M is a power of 2, and hence for $x \geq M^2 \cdot \log^6 M$, we have $\pi(x, M) = \Omega\left(\frac{x}{M \log x}\right)$. Therefore, for any uniformly randomly chosen i in the range $1 \leq i \leq M \cdot \log^6 M$, the probability that $iM+1$ 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 $x = M^2 \cdot \log^6 M$ 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 error 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. This makes the precision argument relatively straightforward.

Recall that FFT crucially depends upon a special kind of 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 an $2M$-th primitive root of unity if and only if $2M$ divides $p - 1$ [Bak07, Theorem 5.12], which gives an alternate reason for choosing $p \equiv 1 \pmod{2M}$. 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 Conclusions

There are two paradigms for multiplying integers, one using arithmetic over complex numbers, and the other using modular arithmetic. Using complex numbers, Schönage and Strassen [SS71] gave an $O(N \cdot \log N \cdot \log\log N \ldots 2^{O(\log^* N)})$ algorithm. Fürer [Fur07] improved this complexity to $O(N \cdot \log N \cdot 2^{O(\log^* N)})$ using some special roots of unity. The other paradigm, modular arithmetic, can be seen as arithmetic in $\mathbb{Q}_p$ with certain precision. A direct adaptation of Schönage-Strassen algorithm in the modular paradigm leads to an $O(N \cdot \log N \cdot \log\log N \ldots 2^{O(\log^* N)})$ algorithm. However in the same paper, Schönage-Strassen also gave a modular algorithm with time complexity $O(N \cdot \log N \cdot \log\log N)$. 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 also be achieved through modular arithmetic. Therefore, in a way, we have unified the two paradigms.

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.

[Fur07] Martin Furer. Faster Integer Multiplication. *Proceedings of the 48^{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. Multipication 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.

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

[SS71] A Schonage 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.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de/