Introduction and Previous Work

Given two $N$ bit integers $a, b$ computing the product $ab$ is an important algorithmic problem. Standard brute-force algorithm runs in time $O(N^2)$. First non-trivial algorithm for the problem is given by Karatsuba and Ofman [KO63] which runs in time $O(N^{\log_2 3})$. The idea in Karatsuba et al algorithm is to simply consider numbers $a$ and $b$ as $a = a_0 + a_1 2^{N/2}$ and $b = b_0 + b_1 2^{N/2}$ and product is computed as $ab = a_0 b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1)2^{N/2} + a_1 b_1 2^N$. By doing this we save a multiplication of two $N/2$ bit numbers while computing the expression in the bracket at the cost of some extra additions. Since addition is less costly (can be done in time linear in number of bits) this gives the claimed improvement. In an important work Schonhage and Strassen [SS71] gave two algorithms for addition is less costly (can be done in time linear in number of bits) this gives the claimed improvement. In an important work Schonhage and Strassen [SS71] gave two algorithms for integer multiplication problem. First one runs in time $O(N \log N \log \log N \cdots 2^{\log^* N})$ and uses complex number arithmetic, whereas the second one runs in time $O(N \log N \log \log N)$ and uses modular arithmetic. Recently M. Furer [Fur07] gave a $O(N \log N 2^{\log^* N})$ time algorithm for the problem using complex number arithmetic. The algorithm using modular arithmetic are preferred over that use complex number manipulations, since in the later case one has to perform truncated computations and the error analysis involved is usually hard. So natural question is to see if we can solve the problem using modular arithmetic and with similar running time as that of [Fur07]. In the current work ([DKSS08]) exactly same question is addressed. The algorithm described in [DKSS08] runs in time $O(N \log N 2^{\log^* N})$ and uses modular arithmetic.

General Approach for Integer Multiplication

Karatsuba et. al.’s method [KO63] suggests a general approach for multiplying two integers. Instead of splitting $a, b$ in blocks of $N/2$ bits as in [KO63], can we split $a, b$ in blocks of $r$ bits for suitably chosen $r$ and possibly hope to compute the product $ab$ more efficiently? Fast Fourier Transform gives the way to do this. Consider $a = \sum_{i=0}^{M-1} a_i q^i$ and $b = \sum_{i=0}^{M-1} b_i q^i$, where $q = 2^r$ and $N = rM$. Think of $a, b$ as polynomials $a(x), b(x)$ over a suitable ring $R$, i.e. $a(x) = a_0 + a_1 x + \ldots + a_{M-1} x^{M-1}$ and $b(x) = b_0 + b_1 x + \ldots + b_{M-1} x^{M-1}$. Let $c(x) = a(x)b(x) \in R[x]$. Once $c(x)$ is obtained use it to recover product of integers $ab$. First consider $\zeta$ be the $2M^{th}$ root of unity in $R$ ($R$ is chosen in such a way that it has $2M^{th}$ root of unity). The $i^{th}$ Fourier coefficient of $a$ is given by $a_i' = a(\zeta^i)$. The polynomial $c(x)$ can be obtained simply by taking point-wise multiplication of Fourier coefficients of $a(x), b(x)$ and then taking the inverse Fourier transform. The most expensive part in this process is to compute Fourier transform of $a(x)$ and $b(x)$ apart from this we need to compute $N$ pointwise multiplication of elements in $R$. If $\zeta$ is some “special” root of unity FFT algorithm can efficiently compute the Fourier transform. So to work with this approach, one needs
first to choose suitable ring $R$ and then get hold of “special” $2M^{th}$ root of unity in $R$ which speed-ups the FFT computation, then one needs to choose parameter $M$ appropriately to get best possible running time.

**Choosing Appropriate Ring**

[SS71] gives two different algorithm for the problem. In first ring $R$ is chosen to be ring of complex numbers $\mathbb{C}$ and parameter $M$ is chosen approximately of size $N/\log^2 N$, i.e. $N$ bit integers $a, b$ are seen as polynomials over $\mathbb{C}$ of degree around $N/\log^2 N$. This lead to an algorithm for the problem with time complexity $O(N \log N \log \log N \ldots 2^{\log^* N})$. The second algorithm by Strassen et. al. uses the ring $\mathbb{Z}_{2^M+1}$ with $M = \sqrt{N}$. Note that the ring $\mathbb{Z}_{2^M+1}$ contains $2M^{th}$ root of unity. This gives $O(N \log N \log \log N)$ time algorithm for the problem. Furer [Fur07] gives an algorithm with better running time by viewing integers $a, b$ in the ring $\mathbb{C}[\alpha]/(\alpha^m + 1)$ for appropriate choice of $m$. The question we want to address is can we give an algorithm as faster as that of Furer but without doing computations in ring of complex numbers?

By analogy to Furer’s algorithm, a natural choice for the ring would be $\mathbb{F}_p[\alpha]/(\alpha^m + 1)$, with $M$ roughly $N/\log^2 N$. The prime $p$ should be chosen such that $2M$ divides $p - 1$, which ensures existence of $2M^{th}$ root of unity. The existence of such a prime is assured by Dirichlet’s theorem on primes in arithmetic progression. But the question is how to get hold of such a prime? Unfortunately the best known upper bound on $p$ is $O(M^{5.5})$([Lin44], [HB92]) which makes it infeasible for deterministic sieving algorithms to find such a $p$. The crucial idea of [DKSS08] is to get a smaller upper bound on $p$ at the cost of representing integers as multivariate polynomials. Due to which one needs to work with general notion of Fourier transforms on abelian groups.

**The Algorithm**

First we describe encoding of an integer as a multivariate polynomial. Let $a$ is $N$ bit integer. Choose $M$ roughly $N/\log^2 N$ and $m$ is roughly $\log N$. First break integer $a$ into $M^k$ blocks where $k$ is a constant greater that 7. So we have $a = a_0 + \ldots + a_{M^k} q^{M^k - 1}$ for $q = 2^{N/M^k}$.

Write each term $a_i q^i$ as a monomial $a_i X_1^{i_1} \ldots X_k^{i_k}$ where $i = i_1 + i_2 M + \ldots i_k M^{k-1}$. Further break each $a_i$ into $m/2$ equal blocks and write it as a polynomial in $\alpha$. This gives a $k$-variate polynomial $a(X)$ over $\mathbb{Z}[\alpha]/(\alpha^m + 1)$. Similarly represent $N$ bit integer $b$ as a $k$-variate polynomial $b(X)$.

Now to work in modular ring we need to choose prime $p$ such that $2M$ divides $p - 1$. By choice of $k$, we have $M = o(N^{1/7})$ combining with ([Lin44], [HB92]) this gives $o(N)$ upper bound on size of smallest possible $p$. Hence such $p$ can be easily computed by standard sieving procedure in time $o(N)$. Choose appropriate constant $c$ such that $p^c$ is greater that any coefficient of polynomial $a(X) b(X)$. This make us enable to work in the ring $R = \mathbb{Z}_{p^c}[\alpha]/(\alpha^m + 1)$. It is not difficult to see that we can retrieve the product of integers $ab$ by evaluating polynomial $a(X) b(X)$ at suitable points. To compute product $a(X) b(X)$ efficiently we first find Fourier transform of $a(X)$ and $b(X)$(Section 4 [DKSS08]).
Choosing suitable root of unity: To compute Fourier Transform of $a(X)$ and $b(X)$, we need to find principal-$2M^{th}$ root of unity $\delta$ (Definition 1 in [DKSS08]) in the ring $R$. Moreover, to make the FFT computations efficient we need this root to satisfy the property $\delta^{M/m} = \alpha$ in the ring $R$. To accomplish this we start with computing a generator of $\mathbb{F}_p^*$, which can be efficiently calculated since $p$ is small. By using Hensel lifting technique we obtain a primitive $p - 1^{th}$ root of unity $\zeta$ in $\mathbb{Z}_p$ using this generator. A principal $2M^{th}$ root of unity in $\mathbb{Z}_p$ is obtained by computing $\zeta^{p^{-1}/2M}$. This root of unity in $\mathbb{Z}_p$ can indeed be used to compute desired principal-$2M^{th}$ root of unity with the property mentioned using technique used in [Fur07]. For details refer to section 2.3 of [DKSS08].
Bibliography


