Fast arithmetic for faster integer multiplication

Svyatoslav Covanov, Emmanuel Thomé

Fast arithmetic for faster integer multiplication

Svyatoslav Covanov
École Polytechnique
University of Lorraine
France

Emmanuel Thomé
INRIA
University of Lorraine
France

ABSTRACT

For almost 35 years, Schönhage-Strassen's algorithm has been the fastest algorithm known for multiplying integers, with a time complexity $O(n \cdot \log n \cdot \log\log n)$ for multiplying n-bit inputs. In 2007, Fürer proved that there exists $K > 1$ and an algorithm performing this operation in $O(n \cdot \log n \cdot K^{\log^* n})$. Recent work showed that this complexity estimate can be made more precise with $K = 8$, and conjecturally $K = 4$. We obtain here the same result $K = 4$ using simple modular arithmetic as a building block, and a careful complexity analysis. We rely on a conjecture about the existence of sufficiently many primes of a certain form.

Keywords

Integer multiplication, FFT, complexity, arithmetic

1. INTRODUCTION

Beyond the schoolbook algorithm, the first nontrivial algorithm improving the complexity for multiplying n-bit integers is Karatsuba's algorithm in the 1960's [8], which reaches the complexity $O(n^{\log_2 3})$, using a divide-and-conquer method. The Karatsuba algorithm can be viewed as a simple case of a more general evaluation-interpolation paradigm. Indeed, it consists in the evaluation of two polynomials in 0, 1 and ∞, followed by a component-wise multiplication, and an interpolation phase involving a multiplication by a 3×3 matrix. By generalizing this evaluation-interpolation approach, it is possible to improve the complexity to $O(n^{1+\epsilon})$ for any $\epsilon > 0$. This result is known as the Toom-Cook algorithm [11].

In [10], Schönhage and Strassen reached the $O(n \cdot \log n \cdot \log\log n)$ complexity using the fast Fourier transform (FFT), which is a divide-and-conquer algorithm allowing one to quickly evaluate a polynomial at the powers of a primitive root of unity [9]. The key to obtaining this complexity result is the appropriate choice of a base ring $\mathbf{R}$ in which evaluation is to be carried out, and in which recursive calls to the multiplication algorithms are also done. The most popular variant of the Schönhage-Strassen algorithm uses $\mathbf{R} = \mathbb{Z}/(2^t + 1)\mathbb{Z}$, where 2 is a primitive $2t$-th root of unity, t being chosen close to $\sqrt{n}$.

For almost 35 years, this complexity estimate remained unbeaten, until Fürer proposed in [5] a new algorithm also relying on the FFT, but using a different ring, namely $\mathbf{R} = \mathbb{C}[x]/(x^P + 1)$ for P a suitably chosen power of 2. This combines the benefits of the Schönhage-Strassen algorithm with the possibility to use larger transform length, thereby allowing recursive calls on shorter data. This eventually yields complexity $O(n \cdot \log n \cdot K^{\log^* n})$ for some constant K.

A variant was subsequently proposed in [3], using $\mathbf{R} = \mathbb{Q}_p[x]/(x^P + 1)$, and with similar complexity. Be it either in Fürer's original context with complex coefficients, or in this p-adic case, some work is needed to properly estimate the required complexity. However the p-adic case makes the analysis easier.

In [7], a new algorithm and a sharper complexity analysis allows one to make the complexity more explicit, namely $O(n \cdot \log n \cdot 8^{\log^* n})$ and even $O(n \cdot \log n \cdot 4^{\log^* n})$ using a conjecture on Mersenne primes.

We will show in this article a different strategy to reach the $O(n \cdot \log n \cdot 4^{\log^* n})$ by a few variations to the original algorithm proposed by Fürer. Our approach adapts an idea from [4] relying on the (unfortunately unlikely) assumption that there exists an infinite sequence of Fermat primes. In contrast, the assumption we use is heuristically valid. This idea, combined with a sharp complexity analysis, permits to reach the complexity $O(n \cdot \log n \cdot 4^{\log^* n})$.

This article is organized as follows. Section 2 describes the essential building blocks we use for our algorithm, and in particular Fürer's algorithm. In Section 3, we describe how to adapt this algorithm with a new ring. In Section 4 a sharp complexity analysis is given, leading to the announced complexity.

Throughout the article, $\log x$ denotes the logarithm in base 2, and $\log_e x$ denotes the classical logarithm. We use the notation $\log^{(m)}$ to denote the m-th iterate of the log function, so that $\log^{(m+1)} = \log \circ \log^{(m)}$.

2. FFT-BASED MULTIPLICATION

2.1 Integers to polynomials

Let a and b be n-bit integers. We intend to compute the integer product $c = ab$. The Kronecker substitution technique associates to a and b two univariate polynomials A and B such that $a = A(\eta)$ and $b = B(\eta)$, for η a suitable power of 2. Coefficients of A and B are obtained by the base η expansion of the integers a and b, and are therefore bounded by η. These polynomials are then interpreted in some ring $\mathbf{R}$, and multiplied modulo a polynomial, so that the result can be recovered as $c = C(\eta)$, where the coefficients of C are interpreted as integers. This expression is valid when η is suitably chosen, so that no overflow happens in the computation of the polynomial product.

The core of this procedure is the modular multiplication in $\mathbf{R}$, which is done with Algorithm 1 which multiplies modulo the minimal polynomial of the set of evaluation points $\mathcal{S}$.

Algorithm: PolynomialMultiply
Input: A, B two polynomials in $\mathbf{R}[x]$ whose product has degree less than N, and $\mathcal{S}$ a set of N evaluation points
Output: $C = A \cdot B$
$A = \mathsf{MultiEvaluation}(A, \mathcal{S})$
$B = \mathsf{MultiEvaluation}(B, \mathcal{S})$
$C = \mathsf{PointwiseProduct}(A, B)$
return $C = \mathsf{Interpolation}(C, \mathcal{S})$

Algorithm 1: Polynomial multiplication

Besides the cost of the MultiEvaluation and Interpolation routines, which will be discussed further, the cost of the PointwiseProduct step in Algorithm 1 is easily seen to be exactly N products in the ring $\mathbf{R}$.

Throughout the article, we use the notations which appeared above. Namely, the integer n denotes the bit size of the integers whose product we intend to compute (as a side note, the approach also applies for a integer middle product $n \times 2n \to n$ bits). Integers are represented by polynomials evaluated at some η as above. We use an evaluation-interpolation approach, using evaluation on $N = 2^k$ points, which are successive powers of an appropriately chosen N-th root of unity in a ring $\mathbf{R}$. The bit size used for representing elements in $\mathbf{R}$ is denoted by t.

2.2 Cooley-Tukey FFT

Let $\mathbf{R}$ be a ring containing a N-th principal root of unity ω. We recall that an N-th principal root of unity is such that $\sum_{j=0}^{N-1} \omega^{jk}$ is either N or 0 depending on whether k is or is not a multiple of N (in $\mathbb{C}$, $\omega = \exp(2i\pi/N)$ is a principal N-th root of unity).

The Fast Fourier Transform algorithm (FFT) evaluates polynomials at all the powers of ω: given $P \in \mathbf{R}[X]$, the FFT returns $P(1), P(\omega), P(\omega^2), \ldots, P(\omega^{N-1})$ (equivalently, we will identify this N-uple with the polynomial having these coefficients). The set of powers of ω will play the role of the set of evaluation points $\mathcal{S}$ mentioned in Algorithm 1.

Let us assume that N is a power of 2. We can describe an algorithm computing the Fourier transform of N points in $\mathbf{R}$ using a divide-and-conquer strategy. This algorithm is the Cooley-Tukey FFT [2], which corresponds to Algorithm 2.

Algorithm: Radix2FFT
Input: $P = \sum_{i=0}^{N-1} p_i X^i$ a polynomial in $R[X]$ of degree $N = 2^k$, ω an N-th root of unity
Output: $P(1) + P(\omega)X + \cdots + P(\omega^{N-1})X^{N-1}$
if $N = 1$ **then**
  return P
else
  $Q_0 = \sum_{j=0}^{N/2-1} p_{2j} X^i$
  $Q_1 = \sum_{j=0}^{N/2-1} p_{2j+1} X^i$
  $Q_0 = \mathsf{Radix2FFT}(Q_0, \omega^2)$
  $Q_1 = \mathsf{Radix2FFT}(Q_1, \omega^2)$
  $P = Q_0(X) + Q_1(\omega \cdot X) + X^{N/2}(Q_0(X) - Q_1(\omega \cdot X))$
  return P
end

Algorithm 2: Cooley-Tukey FFT algorithm

The complexity of Algorithm 2 can be expressed recursively. Each call to Radix2FFT involves 2 recursive calls on half-size inputs as well as $O(N)$ multiplications (the evaluation in $\omega \cdot x$) and additions in $\mathbf{R}$. We thus have

$$C(N) = 2C(N/2) + O(N)$$

from which it follows that $O(Nk) = O(N \log_2 N)$ operations in the ring $\mathbf{R}$ are required.

A more general form of the Cooley-Tukey FFT recursion exists. This starts by writing the transform length $N = 2^k$ as $N = N_1 N_2 = 2^{k_1+k_2}$, with $N_i = 2^{k_i}$. We organize the coefficients of the input polynomial P as the columns of an $N_1 \times N_2$ matrix, and then perform N_1 "row" transforms of length N_2, followed by N_2 "column" transforms of length N_1. This is Algorithm 3.

Algorithm: LargeRadixFFT
Input: $P = \sum_{i=0}^{N-1} p_i X^i \in \mathbf{R}[X]$ of degree $N = N_1 N_2$ with $N = 2^k$ and $N_i = 2^{k_i}$, and ω an N-th root of unity
Output: $P(1) + P(\omega)X + \cdots + P(\omega^{N-1})X^{N-1}$
Let $Q_i(X)$ be such that $P(X) = \sum_{i=0}^{N_1-1} Q_i(X^{N_1})X^i$
for $i = 0$ **to** $N_1 - 1$ **do**
  $Q_i = \mathsf{FFT}(Q_i, \omega^{N_1})$
  $Q_i = Q_i(\omega^i X)$
end
Let $S_j(Y)$ be such that $\sum_i Q_i Y^i = \sum_j S_j X^j$
for $j = 0$ **to** $N_2 - 1$ **do**
  $S_j(Y) = \mathsf{FFT}(S_j, \omega^{N_2})$
end
return $\sum_{j=0}^{N_2} S_j(X^{N_2})X^j$

Algorithm 3: General Cooley-Tukey FFT

One easily sees that Algorithm 3 specializes to Algorithm 2 when $k_1 = 1$. Algorithm 3 leaves unspecified which algorithm is used for the recursive calls denoted by FFT, or more precisely nothing is prescribed regarding how transform length are to be factored as $N = N_1 N_2$ in general.

This "matrix" form of the Cooley-Tukey FFT appeared several times in literature. It is often observed that effectively doing the transposition of the polynomial coefficients, and use Algorithm 3 for balanced transform lengths N_1, N_2 leads to a better performing implementation. As we observe below, this has a stronger impact in the context of Fürer's algorithm.

2.3 Complexity of integer multiplication

By combining the evaluation-interpolation scheme of §2.1 with FFT-based multi-evaluation (and interpolation, which is essentially identical and not discussed further), we obtain quasi-linear integer multiplication algorithms. We identify several tasks whose cost contribute to the bit complexity of such algorithms.

- converting the input integers to the polynomials in $\mathbf{R}[X]$;
- multiplications by roots of unity in the FFT computation;
- linear operations in the FFT computation (additions, etc);

- point-wise products involving elements of $\mathbf{R}$. Recursive calls to the integer multiplication algorithm are of course possible;
- recovering an integer from the computed polynomial.

The first and last items have linear complexity whenever the basis η from §2.1 is chosen as a power of 2 and the representation of elements of $\mathbf{R}$ is straightforward. Using notations described in §2.1, we have a bit complexity: $M(n) = O(M(N)\gamma'_{\mathbf{R}}) + O(N\gamma_{\mathbf{R}})$ where $\gamma_{\mathbf{R}}$ denotes the cost for the point-wise products in $\mathbf{R}$, while $\gamma'_{\mathbf{R}}$ denotes the cost for the multiplication by elements of $\mathbf{R}$ which occur within the FFT computation. The costs $\gamma_{\mathbf{R}}$ and $\gamma'_{\mathbf{R}}$ may differ slightly.

More precise bit complexity estimates depend of course on the choice of the base ring $\mathbf{R}$. We now discuss several possible choices.

2.4 Choice of the base ring

There are several popular options for choosing the ring $\mathbf{R}$, which were given in [10]. We describe their important characteristics. When it comes to roots of unity, choosing $\mathbf{R} = \mathbb{C}$ might seem natural. This needs careful analysis of the required precision. A precision $t = \Theta(\log n)$ bits is compatible with a transform length $N = O(\frac{n}{\log n})$. In this case, the cost $\gamma'_{\mathbf{R}}$ dominates, since multiplication by roots of unity are expensive. This leads to $\gamma'_{\mathbf{R}} = O(M(\log n))$, whence we obtain $M(n) = O(nM(\log n))$. This leads to

$$M(n) = 2^{O(\log^* n)} \cdot n \cdot \log n \cdot \log^{(2)} n \cdot \log^{(3)} n \cdot \ldots,$$

where $O(\log^* n)$ is the number of recursive calls.

Schönhage and Strassen proposed an alternative, namely to use the ring $\mathbf{R} = \mathbb{Z}/(2^t + 1)\mathbb{Z}$, with $t = \Theta(\sqrt{n})$. The ring $\mathbb{R}$ has "cheap" roots of unity, which minimizes the cost $\gamma'_{\mathbf{R}}$. The other term is more important in the recursion. This leads to the complexity equation $M(n) \leq O(n \log n) + 2\sqrt{n}M(\sqrt{n})$, which leads to the complexity $O(n \log n \log^{(2)} n)$.

2.5 Fürer's contribution

The two choices mentioned in §2.4 have orthogonal advantages and drawbacks. The complex option allows larger transform length, shorter recursion size, but suffers from expensive roots of unity, at least considerably more expensive than in the case of the $\mathbb{Z}/(2^t + 1)\mathbb{Z}$ option.

Fürer [6] proposed a ring with cheap roots of unity, yet allowing significantly larger transform length. This ring is $\mathbf{R} = \mathbb{C}[x]/(x^\ell + 1)$. The polynomial x is a natural 2ℓ-th root of unity in $\mathbf{R}$. However in this ring, we can also find roots of unity of larger order. For example an N-th root of unity may be obtained as the polynomial $\omega(x)$ meeting the conditions

$$\forall j \in [0, N[, \omega(e^{2ij\pi/N})^{N/2\ell} = x \circ (e^{2ij\pi/2\ell}) = e^{2ij\pi/2\ell}.$$

The actual computation of $\omega(x) \in \mathbf{R}$ can be done with Lagrange interpolation. A crucial observation is that $\omega^{\frac{N}{2\ell}} = x \in \mathbf{R}$, which means that a fraction $\frac{1}{2\ell}$ of the roots of unity involve a cheap multiplication in $\mathbf{R}$.

Let us now consider how an FFT of length N in $\mathbf{R}[X]$ can be computed with Algorithm 3, with $N_1 = 2\ell$ and $N_2 = \frac{N}{2\ell}$. The N_1 transforms of length N_1 will be performed recursively with LargeRadixFFT. As for the N_2 transforms of length $N_1 = 2\ell$, since $\omega^{N_2} = x$, all multiplication by roots of unity within these transforms are cheap.

bit size range	p
$2^{16} \leq n < 2^{32}$	$44^{16} + 1$
$2^{32} \leq n < 2^{64}$	$96^{32} + 1$
$2^{64} \leq n < 2^{128}$	$300^{64} + 1$
$2^{128} \leq n < 2^{256}$	$532^{128} + 1$
$2^{256} \leq n < 2^{512}$	$892^{256} + 1$
$2^{512} \leq n < 2^{1024}$	$1036^{512} + 1$

Table 1: Primes used in the algorithm

We wish to count how many expensive multiplications by roots of unity are involved in the FFT computation, taking into account the recursive calls to LargeRadixFFT. This number is easily written as

$$E(N) = 2\ell E(\frac{N}{2\ell}) + N,$$
$$\text{whence } E(N) = N(\lceil \log_{2\ell} N \rceil - 1).$$

Fürer defined $\mathbf{R}$ with $\ell = 2^{\lfloor \log^{(2)} n \rfloor}$ and proves that precision $O(\log n)$ is sufficient for the coefficients of the elements of $\mathbf{R}$ which occur in the computation. The integers to be multiplied are split into pieces of r bits, and blocks of $\ell/2$ such pieces are grouped together to form the coefficients of an element of $\mathbf{R}$. In other terms, we have $N \leq 2n/\log^2 n$. Finally, using Kronecker substitution, we embed elements of $\mathbf{R}$ in $\mathbb{Z}$ and we call recursively the algorithm to multiply integers. We get the following recursive formula for the binary complexity $M(n)$:

$$M(n) \leq N(3\lceil \log_{2\ell} N \rceil + 1) \cdot M(O(\log^2 n)) + O(N \log N \cdot \log^2 n) \quad (1)$$

The first product in the previous formula comes from the expensive multiplications involved in Fürer's algorithm and the second product describes the linear operations such as additions, subtractions, cheap multiplications. The integer 3 corresponds to the accumulation of two direct transforms, and one inverse transform for interpolation. Fürer proves that this recurrence leads to

$$M(n) \leq n \log n (2^{d \log^* \sqrt[4]{n}} - d')$$

for some $d, d' > 0$.

3. CONTRIBUTION

3.1 A new ring

The main contribution of this article is to propose a ring $\mathbf{R}$ which reduces the constant d in the complexity estimate above. Instead of working in $\mathbb{C}[x]/(x^\ell + 1)$, we work in $\mathbf{R} = \mathbb{Z}/p\mathbb{Z}$, where p is a prime chosen as $p = r^\ell + 1$ with r a multiple of 4.

The integers a and b to be multiplied are decomposed in some base η, as in §2.1. For the evaluation-interpolation scheme to be valid, the parameter η and the transform length N must be such that $\log N + 2\log \eta \leq \log p$.

The integer ℓ plays a role similar to ℓ in Fürer's construction. We therefore define it likewise, as the largest power of 2 below $\log n$. The integer r is chosen subject to the condition that $r \geq \log n$. For bit sizes in the foreseeable relevant practical range and well beyond, the primes p may be chosen as given by Table 3.1.

Given the form of p, elements x of $\mathbb{Z}/p\mathbb{Z}$ can be represented as $x = \sum_{i=0}^{\ell-1} x_i r^i$, with $0 \leq x_i < r$ (since the element $r^\ell = -1$ cannot be decomposed like the other elements, it has to be treated separately). In other terms, we write down the expansion of x in base r. We have a "cheap" root of unity in $\mathbf{R}$, namely r, which is such that $r^{2\ell} = 1$. As an example, the product of $x = \sum_{i\in[0,\ell-1]} x_i r^i$ with r writes as:

$$\begin{aligned} x \cdot r &= \sum_{i\in[1,\ell]} x_{i-1} r^i = -x_{\ell-1} + \sum_{i\in[1,\ell-1]} x_{i-1} r^i, \\ &= (-x_{\ell-1} + r) + (x_0 - 1) \cdot r + \sum_{i\in[2,\ell-1]} x_{i-1} r^i. \end{aligned}$$

We see that in contrast to Fürer's choice, operations in $\mathbf{R}$ must take into account the propagation of carries (possibly further than in the example above, as x_0 may be zero). However the induced cost remains linear.

We have to give an estimate for the number of bits of p. We need to go beyond the aforementioned lower bound on $\log p$, and provide an interval within which an appropriate prime p may be found.

Conjecture 1. *Let P be a such that $P^\ell \leq 2^{2\log n \log^{(2)} n}$ and $P \geq \log n$, ρ a constant verifying $\rho \geq 6$, and $\epsilon \in]0,1[$. There exists $r \in [P, P + \rho \log n \cdot (\log^{(2)} n)^{1+\epsilon}]$ such that $p = r^\ell + 1$ is prime and r a multiple of 4. For n large enough, we have $\log p \in [\ell \log P, \ell \log P + 2\ell \log^{(3)} n]$, which implies $\log p \leq 2 \log n \cdot (\log^{(2)} n + \log^{(3)} n)$.*

Argument. We begin by establishing the second statement. For n large enough, we have:

$$\begin{aligned} \log p &\leq \ell \log(P + \rho \log n (\log^{(2)} n)^{1+\epsilon}), \\ &= \ell \log P + \ell \log(1 + \frac{1}{P} \cdot \rho \log n (\log^{(2)} n)^{1+\epsilon}), \\ &\leq \ell \cdot \log P + \ell \log(1 + \rho \cdot (\log^{(2)} n)^{1+\epsilon}), \\ &\leq \ell \cdot \log P + (\log \rho + 1)\ell + (1+\epsilon) \cdot \ell \log^{(3)} n, \\ &\leq \ell \cdot \log P + \ell((1+\epsilon) \cdot \log^{(3)} n + 2 \log \rho). \end{aligned}$$

We use the bound on P^ℓ to further bound the above value by $2 \log n \cdot (\log^{(2)} n + \log^{(3)} n)$, which we denote by $s(n)$. Now given the bit size of p, we assume heuristically that prime numbers are encountered with probability at least $\frac{1}{\log_e 2} \frac{1}{s(n)}$. For n large enough, $s(n)$ is below $3 \log n \cdot \log^{(2)} n$, whence we expect that a prime of the desired form can be found. □

We now discuss the advantages of the ring $\mathbf{R} = \mathbb{Z}/p\mathbb{Z}$ in our context. The notation $s(p)$ denoting a bound on the bit size of p used for multiplying two n-bit integers will be retained throughout this section. We remind that $\ell = O(\log n)$ so that, if $p = r^\ell + 1$, there exists an element of order r^ℓ in $\mathbf{R}$. Since r is even, we have a 2^ℓ-th root of unity and there exists, for any power two N such that $N < 2^\ell$, a N-th root of unity. This allows large transform lengths.

The ring $\mathbf{R}$ also spares a factor 2 due to the zero-padding involved in the original Fürer's algorithm. Indeed, the embedding of the coefficients a_i of the polynomial A described in §2.1 leads to elements in $\mathbb{C}[x]/(x^\ell + 1)$ whose representation $\sum_{i=0}^{\ell-1} a_{ij} x^j$ has $a_{ij} = 0$ for $j \in [\ell/2, \ell[$ and the number of bits required to store the a_{ij} is approximatively twice smaller than the number of bits required to store the coefficients of the product. In other terms, the coefficients of A occupy $\frac{1}{4}$ of the bitsize of the ring $\mathbb{C}[x]/(x^\ell + 1)$, whereas in $\mathbf{R}$, they occupy $\frac{1}{2}$ of the bitsize.

Moreover a speed-up is obtained through the number of recursive calls. Using our ring $\mathbf{R}$, we recurse on data of size $\log p = s(n) = 2 \log n \cdot (\log^{(2)} n + \log^{(3)} n)$ instead of $\log^2 n$, which roughly halves the number of recursion levels. There a few more operations to take into account in the formula representing the complexity estimate of the algorithm:

- The decomposition of the elements of $\mathbf{R}$ in base r. According to [1, §1.7], this can be done in $c \cdot M(s(n)) \log \ell$ for some constant c.
- The modulo r operation after a multiplication in radix-r representation. This requires ℓ successive reductions modulo r, hence a total cost $c'\ell M(s(n)/\ell)$ for some constant c'.
- The linear operations: additions, subtractions, multiplication by powers of r and propagation of carries.

The computation of the roots of unity used in $\mathbf{R}$ can be done with a probabilistic algorithm finding a generator g such that $g^{N/(2\ell)} = r$.

The previous remarks lead to the following recursive formula for the complexity of the algorithm:

$$\begin{aligned} M(n) \leq & N(3\lceil \log_{2\ell} N \rceil + 1 + c \log \ell) M(s(n)) \\ & + c' N \ell (3\lceil \log_{2\ell} N \rceil + 1) M(s(n)/\ell) \\ & + c''(3N \log N) \cdot s(n) \end{aligned}$$

Proposition 2. *There exists a constant L such that the time complexity of this algorithm is $O(n \cdot \log n \cdot L^{\log^* n})$.*

Proposition 2 follows from the observation that the second term is bounded by $N(3\lceil \log_\ell N \rceil + 1) M(s(n))$ and that the term $c \log \ell$ can be merged with $3\lceil \log_{2\ell} N \rceil$ since

$$\log \ell = \log^{(2)} n = o(\frac{\log n}{\log^{(2)} n}) = o(\log_\ell N).$$

Thus, we get:

$$\begin{aligned} M(n) \leq & N(O(\lceil \log_{2\ell} N \rceil)) M(s(n)) \\ & + O((3N \log N) \cdot s(n) \end{aligned} \tag{2}$$

Using a proof similar to the one proposed by Fürer, we can deduce from Equation (2) the existence of $L > 1$ such that $M(n) \leq n \log n L^{\log^* n}$.

The result announced in the introduction needs however a few additional tricks to compete with [7]:

- It suffices to compute the FFT of the roots of unity once per recursion level considered (that is, per prime p). This cost is amortized over the numerous multiplications of elements of $\mathbf{R}$ by the same root of unity.
- The naive way to multiply elements of $\mathbf{R}$ uses the Kronecker substitution, which leads to zero-padding and increases the complexity. The next part introduces another strategy, which borrows from the Schönhage-Strassen algorithm.

3.2 Avoiding the Kronecker Substitution

Instead of embedding an element of $\mathbf{R}$ in radix-r representation into an integer, it might be profitable to stay in radix-r representation and to consider the same element in radix-r^β representation, for some β dividing ℓ. In other terms, some coefficients may be grouped together. We may then perform the multiplication in $\mathbf{R}$ via the multiplication of two polynomials modulo $X^{N/\beta}+1$, as is done in the Schönhage-Strassen algorithm.

Before giving the technical analysis, we provide a justification by handwaving of how large the chunks can be. We assume for this fuzzy analysis that $\ell = \log n$. As per the choices in Conjecture 1, we then have $\log r \leq 2\log^{(2)} n$, while we recall that the bit size of p is bounded by $s(n) = 2\log n \cdot (\log^{(2)} n + \log^{(3)} n)$. It follows that the second recursion will use a prime p' of bit size $s(\log p) = 2\log^{(2)} n \cdot (\log^{(3)} n + \log^{(4)} n)$. We thus expect that as many as $\frac{1}{2}\log^{(3)} n$ coefficients of the radix-r representation can be grouped together in a single element modulo p'.

We now examine more precisely the bounds involved in the discussion. Let us fix some notations. The notations $n, N, \mathbf{R}, p, r, \ell$ keep their meaning from the previous sections. We denote by $n', N', \mathbf{R}', p', r', \ell'$ the corresponding parameters for the next recursive call. We thus have $n' = \log p \leq s(n)$.

Let β be the largest power of two such that $2\beta\log r + \log\frac{\ell}{\beta} \leq 2\log^{(2)} p \log^{(3)} p$. We will compute the product of two elements of $\mathbf{R}$ by considering the inputs as polynomials of degree $N' = \ell/\beta$, whose coefficients correspond to the radix-r^β expansion.

We choose the prime p' to recurse on as follows. Let $\ell' = 2^{\lfloor \log^{(3)} p \rfloor}$ (the largest power of 2 below $\log n' = \log^{(2)} p$). Let $P' = 2^{\frac{1}{\ell'}(2\beta\log r + \log\frac{\ell}{\beta})}$. We have $P'^{\ell'} \leq 2^{2\log^{(2)} p \log^{(3)} p}$ by construction, and by definition of ℓ' and β we also have $P' \geq \log^{(2)} p$.

We may thus use Conjecture 1, and find a prime $p' = (r')^{\ell'} + 1$, with $r' \in [P', P' + \rho\log^{(2)} p(\log^{(3)} p)^{1+\epsilon}]$.

The size of p' can be estimated thanks to the upper bound in Conjecture 1: $\log p' \leq \ell' \cdot P' + 2 \cdot \log^{(2)} p \cdot \log^{(4)} p = 2 \cdot \beta \cdot \log r + \log N' + 2 \cdot \log^{(2)} p \cdot \log^{(4)} p$.

Thus, we transform an element x with ℓ coefficients of $\log r$ bits each into a polynomial of degree $N' = \ell/\beta$. This polynomial is first written over the integers, then interpreted over $\mathbf{R}' = \mathbb{Z}/p'\mathbb{Z}$. In order to multiply such polynomials modulo $X^{\ell'}+1$, we need a slightly modified FFT, involving one extra multiplication per polynomial coefficient.

We can now write a recursive formula for the complexity U of multiplication in $\mathbf{R}$.

$$\begin{aligned} U(\log p) \leq & N'(3\lceil \log_{2\ell'} N' \rceil) U(\log p') \\ & + N'(3 + \log\beta + c\log\ell') \cdot M(\log p') \\ & + c' N'(3\lceil \log_{2\ell'} N' \rceil + 3)\ell' \cdot M(\log r') \\ & + c'' N' \log N' \cdot \log p' \end{aligned} \tag{3}$$

The second term and the third terms of the previous sum correspond respectively to the component-wise multiplication with the merging operation of β elements, the decomposition in radix r', and the modulo operations. The careful reader notices that instead of calling recursively the function U for these terms, the function M is used. The reason is that the operations associated to them are negligible, and do not require a sharp analysis, allowing one to reuse the rough complexity given in §3.1.

It may conceivably happen that the requirement on β cannot be met, if for example $2\log r + \log\ell$ already exceeds the required upper bound. Should such a construction fail, this would mean that the recursive call for the multiplication of elements of $\mathbf{R}$ cannot be done in the "smart" way which avoids the factor of two incurred by the Kronecker substitution. While we do acknowledge that such a failure may happen, the following result shows that this is indeed possible asymptotically —albeit for truly gigantic inputs.

Proposition 3. *For n such that $\log^{(3)} n \geq 5.1$, the multiplication in the underlying ring $\mathbf{R}$ can be performed as discussed above.*

Proof. We simply show that under these assumptions, $\beta = 1$ works. We have that $\log r \leq \frac{1}{\ell}s(n) \leq 4(\log^{(2)} n + \log^{(3)} n)$. This implies that $2\log r + \log\ell \leq 9\log^{(2)} n + 8\log^{(3)} n$. We can further derive from $\log p \geq \ell \log P$ that whenever $\log^{(2)} n \geq 1$, we have $\log^{(2)} p \geq \log^{(2)} n$. These two results imply the announced claim. □

3.3 Running example

Let us suppose that we are multiplying two 2^{42}-bit integers a and b modulo $2^{42}-1$. The corresponding prime according to Table 3.1 is $p = 96^{32}+1$. The integers a and b can be transformed into polynomials A and B of degree N such that $A(\eta) = a$ and $B(\eta) = b$, with η such that $2\log\eta + \log N < \log p$. One can check that $\eta = 2^{64}$ works since $2^{42}/64 = 2^{36}$ and $2^{36} \cdot (2^{64})^2 \leq 96^{32}+1$. The degree N is equal to 2^{36}. There is indeed a 2^{36}-th root of unity in $\mathbb{Z}/p\mathbb{Z}$ since 2^{36} divides 4^{32}.

It is now possible to estimate the number of multiplications $\mathbb{Z}/p\mathbb{Z}$ involved in the FFT: $3N\lceil \log_{64} N \rceil = 3 \cdot 2^{36} \cdot 6$ (componentwise product contribute only linearly).

4. A SHARPER COMPLEXITY

This section establishes the bound announced in the introduction, following a sharp analysis of the algorithm described in Section 3.

We let **MulR** be the algorithm performing the multiplication in $\mathbf{R}$ and using the strategy of §3.2, and **MulZ** the algorithm which multiplies integers, and calls **MulR** recursively (we include in **MulZ** the conversion to radix-r representation, as well as the preparation of data which needs be computed only once per recursion level, namely transforms of roots of unity). The complexity of the former algorithm will be denoted by $U(\log p)$, the complexity of the latter by $D(n)$.

Propositions 4 to 6 are dedicated to the proof that the main contribution to $D(n)$ is due to **MulR**.

Theorem 7 establishes how many recursive levels are involved in **MulR**.

Theorem 8 proves the main bound for $U(\log p)$. Lemmata 9 to 14 establish the negligible part in the equation describing U, and the follow-up statements conclude the proof.

Let $q(n)$ be the recursion depth of **MulR** and p_1 to $p_{q(n)}$ the sequence of primes such that for $i \in [1, q(n)]$, the field considered at the depth i is $\mathbb{Z}/p_i\mathbb{Z}$. Let $R(n)$ be the complexity of computing the Fourier Transforms of the primitive

roots within fields $\mathbb{Z}/p_i\mathbb{Z}$. Let $R'(n)$ denote the complexity of computing the primes p_i themselves. If the toplevel algorithm is **MulZ**, then the computation of the first prime p_1 is not negligible compared to $n \log n 4^{\log^* n}$, this is why we need to assume that at the toplevel, a fast algorithm is executed such as Fürer's algorithm (e.g. over $\mathbb{C}$), as suggested in [7, §8.2], calling **MulZ** for its inner multiplications and computing p_1. Given these notations, and following estimates already given in previous sections, we may write $D(n)$ as:

$$\begin{aligned} D(n) \leq & N(3\lceil \log_{2\ell} N \rceil + 1 + c \log \ell) U(s(n)) \\ & + c' N \ell (3\lceil \log_{2\ell} N \rceil + 1) \cdot M(\frac{s(n)}{\ell}) \\ & + c''(3N \log N) \cdot s(n) + R(n) + R'(n) \end{aligned} \tag{4}$$

Since one third of the transforms in the expensive multiplications is already accounted for in $R(n)$, this allows us to rewrite $U(\log p)$ with the same notations as in §3.2:

$$\begin{aligned} U(\log p) \leq & N'(2\lceil \log_{2\ell'} N' \rceil) U(\log p') \\ & + N'(3 + \log \beta + c \log \ell') \cdot M(\log p') \\ & + c' N' (2\lceil \log_{2\ell'} N' \rceil + 3) \ell' M(\log(r')) \\ & + c'' N' \log N' \cdot \log p' \end{aligned} \tag{5}$$

Let us remind that $s(n)$ denotes the quantity $(2 \log^{(2)} n + 2 \log^{(3)} n) \cdot \log n$.

Proposition 4. *If $q(n) \leq \log^* n$, $R(n)$ is negligible compared to $n \log n \cdot 4^{\log^* n}$.*

Proof. There are at first $\frac{N}{2\ell}$ different roots of unity modulo a power of r in the first call of **MulR**. So, the complexity estimate for the computation of the transforms of the roots of unity is smaller than $\frac{N}{2\ell} M(s(n))$.

We have shown in §3.1 that $M(n) \leq n \log n \cdot L^{\log^* n}$ for some $L > 1$. Consequently,

$$\frac{N}{2\ell} M(s(n)) \leq \frac{N}{2\ell} \cdot s(n) \cdot \log s(n) \cdot L^{\log^* n}$$

It is possible to bound roughly $s(n)$ by $4 \cdot \log n \cdot \log^{(2)} n$, which, asymptotically, implies that $\log s(n) \leq 2 \log^{(2)} n$.

This leads to:

$$\frac{N}{2\ell} M(s(n)) \leq \frac{N}{\log n} \cdot 16 \cdot \log n \cdot (\log^{(2)} n)^2 \cdot L^{\log^* n}$$

Since $N = \frac{n}{s(n)}$, it is clear that the previous quantity is negligible compared to $n \log n$. Moreover, if $q(n) \leq \log^* n$, we can roughly bound the computation of the transforms of the roots of unity, for all recursion levels, by $O(n \log n \log^* n)$, which brings the announced proposition. □

Proposition 5. *If $q(n) \leq \log^* n$, $R'(n)$ is negligible compared to $n \log n$.*

Proof. In order to prove the statement, it is enough to prove that finding the prime in the first recursive call gives a complexity of the form $o(n)$. Since for deeper sublevels, the size of the prime is smaller, we get a complexity of the type $o(n \cdot q(n)) = o(n \log^* n)$.

In the first recursive call, we are computing a multiplication modulo p where $\log p \leq 3 \log n \log^{(2)} n$. The next prime p' verifies $2 \log^{(2)} p \cdot \log^{(3)} p \leq \log p' \leq 3 \log^{(2)} p \cdot \log^{(3)} p$ according to Conjecture 1. Also, there are potentially $\rho \log^{(2)} p \cdot (\log^{(3)} p)^{1+\epsilon}$ tests. Using the sieve of Eratosthenes, one gets a time complexity estimate of the form:

$$2^{3 \log^{(2)} p \cdot \log^{(3)} p} (3 \log^{(2)} p \log^{(3)} p)^2 = (\log n)^{O(\log^{(3)} n)}$$

Combined with the number of tests, one gets a complexity negligible when compared to n. □

Proposition 6. *In Equation (4), $c' N \ell (3\lceil \log_{2\ell} N \rceil + 1) \cdot M(\frac{s(n)}{\ell})$ is negligible compared to $n \log n$.*

Proof. Since $\frac{s(n)}{\ell} \leq 2 \log^{(2)} n + 2 \log^{(3)} n \leq 4 \cdot \log^{(2)} n$, $M(\frac{s(n)}{\ell}) \leq 4 \log^{(2)} n \cdot \log^{(3)} n \cdot L^{\log^* n}$ for n large enough. It is possible to bound $3\lceil \log_{2\ell} N \rceil + 1$ by $4 \log_{2\ell} N$ and N by $\frac{n}{\log n \log^{(2)} n}$.

Combining all these remarks leads to:

$$c' N \ell (3\lceil \log_{2\ell} N \rceil + 1) \cdot M(\frac{s(n)}{\ell}) \leq 8n \log n \frac{\log^{(3)} n}{\log^{(2)} n} \cdot L^{\log^* n}$$

and allows one to conclude. □

The most expensive part of Equation (4) corresponds to the call to the function U. Thus, it should be enough to prove that $U(\log p)$ corresponds to the bound announced in the introduction.

Theorem 7. *For n large enough, $q(n) \leq \log^* n$.*

Proof. In order to compute the recursion depth of our algorithm, we look at the sequence of the sizes s_m. We have $s_0 = n$, and $s_{m+1} \leq 3 \log s_m \log^{(2)} s_m$.

There exists M and m_0 such that for $m \leq m_0$, s_m is a decreasing sequence verifying $s_m \geq M$ and for $m \geq m_0$, $s_m \leq M$. Then $q(n) = m_0$.

We have $\log s_{m+1} \leq \log^{(2)} s_m + 2 + \log^{(3)} s_m$. Thus, $\log^{(2)} s_{m+1} \leq \log^{(3)} s_m + \frac{2 + \log^{(3)} s_m}{\log^{(2)} s_m}$. For M large enough, we have $\log^{(2)} s_{m+1} \leq \log^{(3)} s_m + 1$ for $m \leq m_0$. Then, $\log^{(2)} s_2 \leq \log^{(3)} s_1 + 1 \leq \log^{(4)} s_0 + \frac{1}{\log^{(3)} s_0} + 1$.

Similarly, one deduces:

$$\log^{(2)} s_3 \leq \log^{(3)} s_2 + 1 \leq \log(\log^{(4)} s_0 + \frac{1}{\log^{(3)} s_0} + 1) + 1$$

and $\log^{(2)} s_3 \leq \log^{(5)} s_0 + \frac{1}{\log^{(3)} s_0 \log^{(4)} s_0} + \frac{1}{\log^{(4)} s_0} + 1 \leq \log^{(5)} s_0 + \frac{1}{(\log^{(4)} s_0)^2} + \frac{1}{\log^{(4)} s_0} + 1$.

By developing the formula for s_4, we get:

$$\begin{aligned} \log^{(2)} s_4 \quad \leq \quad & \log^{(6)} s_0 + \frac{1}{(\log^{(5)} s_0)^3} + \frac{1}{(\log^{(5)} s_0)^2} + \\ & \frac{1}{\log^{(5)} s_0} + 1 \end{aligned}$$

For $m = \log^* s_0 - 2$, $\log^{(m+1)} \leq 2$ and $\log^{(m)} \geq 2$, which leads to:

$$\log^{(2)} s_m \leq \log^{(m+1)} s_0 + \frac{1}{(\log^{(m)} s_0)^{m-2}} + \cdots + 1$$

and:

$$\log^{(2)} s_m \leq 2 + \sum_{i \in [0, m-2]} \frac{1}{2^i} \leq 4$$

By choosing M such that $\log^{(2)} M \geq 4$, one concludes that $m_0 \leq \log^* s_0 - 2 = \log^* n - 2$. □

Let us announce the main result of this subsection.

Theorem 8. *There exist $d \in]0,1[$ and $C > 1$ such that $U(\log p) \leq C \log p \log^{(2)} p \cdot (4^{q(\log p)} - d)$, for any p, where $q(\log p)$ denotes the number of levels of recursion of* **MulR**.

In order to prove this theorem, let us rewrite Equation (5) and remind the definition of the different quantities.

- ℓ is the degree of $p = r^\ell + 1$
- $\log r$ is the size of the coefficients: $\log r = \frac{\log p}{\ell}$
- β corresponds to the biggest power of two verifying $2\beta \log r + \log \frac{\ell}{\beta} \leq 2 \cdot \log^{(2)} p \cdot \log^{(3)} p$
- N' is equal to ℓ/β
- ℓ' is the degree of the polynomial of the next recursive call, so $\ell' \approx \log^{(2)} p$
- p' is the prime that we use on the next level, which means that $\log p' \leq \log N' + 2\beta \frac{\log p}{\ell} + 2 \log^{(2)} p \cdot \log^{(4)} p$ and $\log p' \geq \log^{(2)} p \log^{(3)} p$

Thus, it is possible to rewrite Equation (5):

$$\begin{aligned} U(\log p) \leq & N'(2\lceil \log_{2^{\ell'}} N' \rceil) U(\log p') \\ & + N'(3 + \log \beta + c \log \ell') M(\log p') \\ & + c' N' \ell' (2\lceil \log_{2^{\ell'}} N' \rceil + 3) \cdot M(\log(r')) \\ & + c'' N' \log N' \cdot \log p' \end{aligned} \quad (6)$$

This equation leads to the following partition: $U(\log p) \leq A_1(\log p) + A_2(\log p) + A_3(\log p) + A_4(\log p)$ where each A_i corresponds in (6) to the i-th term of the sum.

In a first time, it has to be proven that there exists C' such that $A_2 + A_3 + A_4 \leq C' \log p \log^{(2)} p$ for $\log p$ large enough.

Lemma 9. *On any level of recursion of the algorithm* **MulR**, *the input p verifies the hypothesis $\log \ell \leq \frac{\log p}{\ell}$.*

Proof. This follows directly from $\log p \geq \ell \log r$ and from $\log r \geq \log P \geq \log^{(2)} p \geq \log \ell$. □

Lemma 10. $N' \leq 4 \frac{\log p}{\log^{(2)} p \log^{(3)} p}$.

Proof. Let us remind that $N' = \frac{\ell}{\beta}$, and by corollary, $N' = \ell \log r / (\beta \log r) = \log p / (\beta \log r)$.

$4\beta \log r \geq 2\beta \log r + 2 \log r$ since β is an integer, and $2\beta \log r + 2 \log r = 2\beta \log r + 2 \frac{\log p}{\ell}$.

By the lemma 9, $2\beta \log r + 2\frac{\log p}{\ell} \geq 2\beta \log r + 2 \log \ell \geq 2\beta \log r + \log N$. By definition of β, we conclude that $4\beta \log r \geq \log^{(2)} p \log^{(3)} p$. □

Lemma 11. $\beta \leq \log^{(3)} p$.

Proof. By definition, $2\beta \log r \leq 2 \log^{(2)} p \log^{(3)} p$. Moreover, according to the lemma 9, $\log r = \frac{\log p}{\ell} \geq \log \ell$. This means that $\beta \leq \log^{(3)} p$. □

Lemma 12. $A_2(\log p) = o(\log p \log^{(2)} p)$.

Proof. M verifies $M(\log p') \leq \log p' \log^{(2)} p' L^{\log^* p' - 1}$ as it has been proven in §3.1. $\beta \leq \log^{(3)} p \leq \ell' + 1$.

$$A_2(\log p) \leq N' \cdot 2c \log \ell \log p' \log^{(2)} p' L^{\log^* p' - 1}$$

Reusing the bound on $\log p'$, $\log p' \leq 2\beta \log r + \log N' + 2 \log^{(2)} p \cdot \log^{(4)} p \leq 3 \log^{(2)} p \log^{(3)} p$ and the lemma 10:

$$A_2(\log p) \leq 4 \log p 2c \log \ell \cdot 3 \log^{(2)} p' L^{\log^* p' - 1}$$

Since $\log p' \leq 3 \log^{(2)} p \log^{(3)} p$ and $\ell \leq \log^{(2)} p$, one can conclude that $A_2(\log p) = o(\log p \log^{(2)} p)$. □

Lemma 13. $A_3(\log p) = o(\log p \log^{(2)} p)$.

Proof. According to the bound on N' and on M, it is possible to rewrite A_3 like this: $A_3(\log p) \leq c' 4 \frac{\log p}{\log^{(2)} p \log^{(3)} p} \cdot \ell' \cdot (4\lceil \log_{2^{\ell'}} N' \rceil) \cdot \log r' \cdot \log^{(2)} r' \cdot L^{\log^*(r') - 1}$.

We know that $\ell' \cdot \log r' = \log p' \leq 3 \log^{(2)} p \log^{(3)} p$ and that for $\log p$ large enough, $4\lceil \log_{2^{\ell'}} N' \rceil \leq 5 \frac{\log n}{\log^{(2)} n}$. Moreover,

$$\log r' \leq \frac{1}{\ell'} \cdot 3 \log^{(2)} p \log^{(3)} p \leq 6 \log^{(3)} p$$

which involves that:

$$A_3(n) \leq 12 \cdot 20 \cdot \log p \log^{(2)} p \frac{\log^{(4)} p}{\log^{(3)} p} L^{\log^*(r') - 1}$$

and this proves that $A_3(\log p) = o(\log p \log^{(2)} p)$. □

Lemma 14. *There exists C' such that $A_2 + A_3 + A_4 \leq C' \log p \log^{(2)} p$ for $\log p$ large enough.*

Proof. Using the bound given in the lemma 10 and the bound on $\log p'$, A_4 is bounded by $4c'' \log p \log^{(2)} p$.

Thus, since A_2 and A_3 are negligible when compared with $\log p \log^{(2)} p$, there exists $C' > 4c''$ such that $A_2 + A_3 + A_4 \leq C' \log p \log^{(2)} p$. □

It remains to prove Theorem 8.

Proof. Let us suppose that the theorem holds for any input of size smaller than $\log p$. According to the inductive hypothesis:

$$A_1(\log p) \leq N'(2\lceil \log_{2^{\ell'}} N' \rceil) C \log p' \cdot \log^{(2)} p' \cdot (4^{q(\log p) - 1} - d)$$

Rewriting the bounds on some quantities, we get:

- $\log p' \leq 2\beta \log r + \log N' + 2 \log^{(2)} p \log^{(4)} p \leq 2\beta \cdot \log r + 3 \log^{(2)} p \log^{(4)} p$
- $\log^{(2)} p' \leq \log(3 \log^{(2)} p \log^{(3)} p)$ and this quantity is smaller than $\log^{(3)} p + 2 \cdot \log^{(4)} p$ for $\log p$ large enough
- $2\lceil \log_{2^{\ell'}} N' \rceil \leq 2 \frac{\log N'}{\log(\frac{\ell}{2} \log^{(2)} p)} + 2 \leq 2 \frac{\log^{(2)} p}{\log^{(3)} p} + 2$

In the last inequality, the term 2 can be neglected, using a similar argument as in lemma 12: thus, it contributes to C'.

One needs to decompose $\log p' \log^{(2)} p'$ using the previous bounds:

$$\begin{aligned} \log p' \log^{(2)} p' & \leq (2\beta \log r + 3 \log^{(2)} p \log^{(4)} p) \cdot \\ & \quad (\log^{(3)} p + 2 \log^{(4)} p) \\ & \leq 2\beta \log r \log^{(3)} p + 10 \log^{(2)} p \log^{(3)} p \cdot \\ & \quad \log^{(4)} p \end{aligned}$$

Thus, it is possible to rewrite A_1 like this:

$$\begin{aligned} A_1(\log p) & \leq 2 \frac{\log p \log^{(2)} p}{\beta \log r \log^{(3)} p} C \cdot (2\beta \log r \log^{(3)} p + \\ & \quad 10 \log^{(2)} p \log^{(3)} p \log^{(4)} p) \cdot \\ & \quad (4^{q(\log p) - 1} - d) \\ A_1(\log p) & \leq 4C \log p \log^{(2)} p \cdot (4^{q(\log p) - 1} - d) + 20 \cdot \\ & \quad \frac{\log p \log^{(2)} p}{\beta \log r \log^{(3)} p} \cdot C \log^{(2)} p \log^{(3)} p \log^{(4)} p \cdot \\ & \quad (4^{q(\log p) - 1} - d) \end{aligned}$$

and since $\beta \log r \geq \frac{1}{4} \log^{(2)} p \cdot \log^{(3)} p$, combined with Theorem 7:

$$\begin{aligned} A_1(\log p) &\leq 4C \log p \log^{(2)} p \cdot (4^{q(\log p)-1} - d) + \\ &\quad o(C \log p \log^{(2)} p) \end{aligned}$$

For $\log p$ large enough, there exists $d' \leq \frac{1}{2}$ such that:

$$\begin{aligned} A_1(\log p) &\leq 4C \log p \log^{(2)} p \cdot (4^{q(\log p)-1} - d) + \\ &\quad d'C \log p \log^{(2)} p \end{aligned}$$

One needs to choose d and C such that $3dC \geq d'C + C'$. Given the cost Γ of the algorithm **MulR** for $q(\log p) = 0$, the parameter C has also to be large enough such that $C(1-d) \geq \Gamma$. Thus, $U(\log p) \leq \log p \log^{(2)} p \cdot C(4^{q(\log p)} - d)$. □

By combining the two previous theorems, we get the complexity we are looking for: $O(\log p \log^{(2)} p 4^{\log^* \log p})$ for U.

Corollary 15. $D(n) = O(n \log n 4^{\log^* n})$.

Proof. Coming back to the equation 4:

$$\begin{aligned} D(n) \leq & N(3\lceil \log_{2\ell} N \rceil + 1 + c \log \ell) U(s(n)) \\ & + c' N \ell (3\lceil \log_{2\ell} N \rceil + 1) \cdot M(\frac{s(n)}{\ell}) \\ & + c''(3N \log N) \cdot s(n) + R(n) + R'(n) \end{aligned} \quad (7)$$

It has been proven that $D(n) \leq 4N(\log_{2\ell} N) \cdot U(s(n)) + o(n \log n \cdot 4^{\log^* n})$.

By Theorem 8, $D(n) = O(\frac{n}{\log n \log^{(2)} n} \cdot \frac{\log n}{\log^{(2)} n} \cdot 2 \log n \cdot \log^{(2)} n \cdot \log(2 \log n \log^{(2)} n) 4^{\log^*(2 \log n \log^{(2)} n)}) = O(n \log n \cdot 4^{\log^* n})$. □

5. CONCLUSIONS

Our algorithm follows Fürer's perspective, and improves on the cost of the multiplications in the underlying ring. Although of similar asymptotic efficiency, it therefore differs from the algorithm in [7], which is based on Bluestein's chirp transform and balances the costs of the "expensive" and "cheap" multiplications.

It is interesting to note that both algorithms rely on conjectures related to the repartition of two different kinds of primes. It is not clear which version is the most practical, but our algorithm avoids the use of bivariate polynomials and seems easier to plug in a classical radix-ℓ FFT by modifying the arithmetic involved. The only additional cost we have to deal with is the question of the decomposition in radix r, and the computation of the modulo, which can be improved using particular primes.

A natural question arises: can we do better? The factor $4^{\log^* n}$ comes from the direct and the inverse FFT we have to compute at each level of recursion, the fact that we have to use some zero-padding each time, and of course the recursion depth, which is $\log^* n + O(1)$.

Following the same approach, it seems hard to improve any of the previous points. Indeed, the evaluation-interpolation paradigm suggests a direct and an inverse FFT, and getting a recursion depth of $\frac{1}{2} \log^* n + O(1)$ would require a reduction from n to $\sqrt{\log n}$ at each step.

We can also question the practicality of our approach, like for all existing Fürer-like algorithms. Is it possible to make a competitive implementation of those algorithms which would beat the current implementations of Schönhage-Strassen's algorithm ?

6. REFERENCES

[1] R. Brent and P. Zimmerman. *Modern computer algebra*, pages 38–39. Cambridge University Press, The Edinburgh Building, Cambridge CB2 8RU, UK, 2011.

[2] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comput.*, 19:297–301, 1965.

[3] A. De, P. P. Kurur, and C. S. and Ramprasad Saptharishi. Fast integer multiplication using modular arithmetic. In *40th annual ACM symposium on Theory of computing*, STOC '08, pages 499–506, New York, NY, USA, 2008. ACM.

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

[5] M. Fürer. Faster integer multiplication, 2007.

[6] M. Fürer. Faster integer multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009.

[7] D. Harvey, J. van der Hoeven, and G. Lecerf. Even faster integer multiplication. *CoRR*, abs/1407.3360, 2014.

[8] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics-Doklady*, 7:595–596, 1963. (English translation).

[9] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[10] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3-4):281–292, 1971.

[11] A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963. (English translation).