Acta Informatica 28, 693–701 (1991)

On Fast Multiplication of Polynomials Over Arbitrary Algebras

David G. Cantor and Erich Kaltofen*

1. Introduction. In this paper we generalize the well-known Schönhage-Strassen algorithm for multiplying large integers to an algorithm for multiplying polynomials with coefficients from an arbitrary, not necessarily commutative, not necessarily associative, algebra $\mathcal{A}$. Our main result is an algorithm to multiply polynomials of degree $< n$ in $O(n \log n)$ algebra multiplications and $O(n \log n \text{ loglog } n)$ algebra additions/subtractions (we count a subtraction as an addition). The constant implied by the "O" does not depend upon the algebra $\mathcal{A}$. The parallel complexity of our algorithm, i.e., the depth of the corresponding arithmetic circuit, is $O(\log n)$.

When division by 2 is possible, then the Schönhage-Strassen [13] integer multiplication algorithm can be easily reformulated as a polynomial multiplication procedure (c.f. [11]). Schönhage [12] investigated the polynomial multiplication problem for arbitrary fields of characteristic 2, in which the standard 2^k-point Discrete Fast Fourier Transform algorithm (DFT) cannot be used because it requires division by 2.

The fields over which the DFT is used do not necessarily contain the primitive roots of unity necessary for the computation of the Discrete Fast Fourier Transform and, to use it, such roots must be adjoined to the ground field. It is this which increases the complexity from $O(n \log n)$ to $O(n \log n \text{ loglog } n)$. Schönhage's algorithm for fields of characteristic 2 uses a 3^k-point Fourier transform. When division by 3 is possible, he obtains again an algorithm of complexity $O(n \log n \text{ loglog } n)$. His approach does not appear to generalize to s^k transforms, even when $s = 5$.

Here, we exhibit an alternate method that works for order s^k for any integer $s \geq 2$. By applying this method for two relatively prime values of s, we obtain a method not requiring division. As a result our method is valid for any algebra $\mathcal{A}$: Specifically the algebra $\mathcal{A}$ must be an Abelian group under "+" and have a binary operation "$\cdot$" satisfying the distributive law

$$(u + v) \cdot (x + y) = u \cdot x + u \cdot y + v \cdot x + v \cdot y$$

for all u, v, x, y in $\mathcal{A}$. In this generality, multiplication of two polynomials $\sum_{i=0}^{m} a_i x^i$ and $\sum_{j=0}^{n} b_j x^j$ means the computation of all of the terms of the product, i.e., computation of all terms of the form $c_k = \sum_i a_i b_{k-i}$.

Thus our method may be used for multiplying "string polynomials" [8], 4.6.1, exercises 17 and 18, or for multiplying matrix polynomials.

Over special rings the complexity may be smaller: For finite fields see [5] and [10]. If one allows the total operation count to be asymptotically worse, then an $O(n)$ non-scalar multiplicative complexity can be achieved differently, e.g., by using the method of [3] recursively; see also [4] and [14]. The algorithm in [3] also enables polynomial evaluation.

*The authors would like to acknowledge the partial support of NSA Grant MDA-904-88-H-2031 and NSF Grant Nr. CCR-87-05363. To appear in *Acta Informatica*.

We also refer to [7] for achieving $O(n \log n)$ multiplicative complexity over a ring, but asymptotically worse additive complexity.

Our model of computation is that of a straight line program for obtaining the coefficients of the product from the coefficients of the input [1], although the asymptotic complexity remains the same if the model is an algebraic random access machine [6]. Non-scalar multiplications are those in which both factors depend upon the coefficients of the input.

Our method is a special case of a *bilinear algorithm* [14]. Suppose first that we wish to multiply two polynomials of degree $n-1$, say, $\sum_{i=0}^{n-1} a_i x^i$ and $\sum_{j=0}^{n-1} b_j x^j$, each with integral coefficients, to obtain their product $\sum_{i=0}^{2n-2} c_i x^i$. We shall, in effect, describe two sequences of matrices $A_0, A_1, \ldots A_r$, and $B_0, B_1, \ldots, B_s$. Let $\mathbf{a}$ denote the vector $(a_0, a_1, \ldots, a_{n-1})$, let $\mathbf{b}$ denote the vector $(b_0, b_1, \ldots, b_{n-1})$ and let $\mathbf{c}$ denote the vector $(c_0, c_1, \ldots, c_{2n-2})$. We will compute the vectors $A_0 A_1 \cdots A_r \mathbf{a}$ and $A_0 A_1 \cdots A_r \mathbf{b}$, and take the term-by-term product of these two vectors to obtain a vector $\mathbf{d}$. Then the product $B_0 B_1 \cdots B_s \mathbf{d}$ will be a certain integer multiple $N_1 \mathbf{c}$ of $\mathbf{c}$. We will do this twice, the second time computing $N_2 \mathbf{c}$, where N_1 and N_2 are relatively prime integers. The Euclidean algorithm shows that there exist two integers M_1 and M_2 such that $M_1 N_1 + M_2 N_2 = 1$. Thus we may obtain $\mathbf{c} = M_1(N_1 \mathbf{c}) + M_2(N_2 \mathbf{c})$. The matrices A_i and B_i will be sparse, consisting entirely of 0's, 1's and -1's, and multiplying by them can be done entirely using additions and subtractions. The multiplications by M_1 and M_2 may be treated as repeated additions. The only multiplications absolutely necessary are those to compute $\mathbf{d}$. Combining all of the above shows that there exist matrices U and V with integral coefficients such that

$$\mathbf{c} = V((U\mathbf{a}) \cdot (U\mathbf{c})),$$

where "$\cdot$" denotes term-by-term multiplication. It is easy to verify that such a bilinear algorithm which is valid over the ring of integers is a formal identity relating the coefficients a_i, b_i, and c_i and that, as such, it is valid over any algebra $\mathcal{A}$ as described above.

2. The Discrete Fast Fourier Transform. Throughout this paper s will denote a positive integer ≥ 2. We will use the Discrete Fast Fourier Transform (DFT) of order $n = s^r$. Suppose that $\mathcal{D}$ is an integral domain containing a primitive n^{th} root of unity ω_n. The DFT of order n takes as input a sequence $\{a_0, a_1, \ldots, a_{n-1}\}$ of n elements from $\mathcal{D}$ and and a primitive n^{th} root of unity ω_n, also from $\mathcal{D}$. Its output is the sequence $\{A_0, A_1, \ldots, A_{n-1}\}$ where $A_j = f(\omega_n^j)$, with $f(x) = \sum_{i=0}^{n-1} a_i x^i$.

Recall the algorithm by noting that if $r \geq 2$ then

$$\begin{aligned}
f(x) &= \sum_{i=0}^{n-1} a_i x^i \\
&= \sum_{i=0}^{s-1} \sum_{j=0}^{s^{r-1}-1} a_{sj+i} x^{sj+i} \\
&= \sum_{i=0}^{s-1} x^i \sum_{j=0}^{s^{r-1}-1} a_{sj+i} (x^s)^j \\
&= \sum_{i=0}^{s-1} x^i f_i(x^s),
\end{aligned}$$

where

$$f_i(x) = \sum_{j=0}^{s^{r-1}-1} a_{sj+i} x^j.$$

Thus, to evaluate $f(x)$ at all of the roots of unity of order s^r, one first evaluates each of the $f_i(x)$ at all of the roots of unity of order s^{r-1}. Then one evaluates the $f(\omega)$, where ω is an n^{th} root of unity, as a polynomial of degree $< s$, with coefficients the (already evaluated) $f_i(\omega^s)$, using Horner's method. This yields

LEMMA 2.1. *The DFT of order $n = s^r$ can be performed as a straight-line algorithm using $\leq r(s-1)n$ additions/subtractions, and $\leq r(s-1)n$ multiplications. All multiplications are by powers of ω_n.*

PROOF: When $r = 1$, simply use Horner's method to evaluate $f(x)$ at each of the s^{th} roots of unity. This amounts to s evaluations of a polynomial of degree $\leq s-1$ and requires $\leq s(s-1)$ multiplications and additions/subtractions, with each multiplication being by a power of ω_s. When $r \geq 2$, one uses the above method, recursively. The evaluation of each of the s polynomials $f_i(x)$ at all roots of unity of order s^{r-1} can be done using s DFT's, each of order s^{r-1}, hence, inductively, with $\leq s(r-1)(s-1)n/s = (r-1)(s-1)n$ additions and multiplications, with the latter being by powers of $\omega_{n/s}$. Then n evaluations of $f(x)$, as a polynomial of degree $\leq s-1$, using Horner's method, requires $\leq (s-1)n$ additions and multiplications, with the latter being by powers of ω_n. Combining these numbers yields the Lemma. ∎

We now consider the problem of finding the product $\sum_{i=0}^{2n-2} c_i x^i$ of the polynomials $\sum_{i=0}^{n-1} a_i x^i$ and $\sum_{i=0}^{n-1} b_i x^i$ with coefficients in $\mathcal{D}$, and where $n = s^r$. Suppose that $\{A_0, A_1, \ldots, A_{n-1}\}$, and, respectively, $\{B_0, B_1, \ldots, B_{n-1}\}$ are the DFT's of the sequences $\{a_0, a_1, \ldots, a_{n-1}\}$ and, respectively, $\{b_0, b_1, \ldots, b_{n-1}\}$, both with respect to the same root of unity ω_n. Put $D_i = A_i B_i$ for $0 \leq i < n$ and let $\{d_0, d_1, \ldots, d_{n-1}\}$ be the DFT of the sequence $\{D_0, D_1, \ldots, D_{n-1}\}$ with respect to the root of unity $1/\omega_n$. Then, if $0 \leq h < n$, we have

$$\begin{aligned}
d_h &= \sum_{i=0}^{n-1} D_i \omega_n^{-hi} \\
&= \sum_{i=0}^{n-1} A_i B_i \omega_n^{-hi} \\
&= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_j \omega_n^{ij} \sum_{k=0}^{n-1} b_k \omega_n^{ik} \omega_n^{-hi} \\
&= \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} a_j b_k \sum_{i=0}^{n-1} \omega_n^{i(j+k-h)} \\
&= n \sum_{j+k \equiv h \pmod{n}} a_j b_k \\
&= n(c_h + c_{n+h}),
\end{aligned}$$

(where c_{2n-1} is defined to be 0) since, in the next to last sum, either $j+k=h$ or $j+k=h+n$.

Suppose that $\omega = \omega_{ns}$ is a primitive $(ns)^{\text{th}}$ root of unity in the integral domain $\mathcal{D}$. Then $\omega_n = \omega^s$ is a primitive n^{th} root of unity and $\omega_s = \omega^n$ is a primitive s^{th} root of unity. Let $\{A'_0, A'_1, \ldots, A'_{n-1}\}$ and, respectively, $\{B'_0, B'_1, \ldots, B'_{n-1}\}$ be the DFT's of the sequences $\{a_0, a_1\omega, a_2\omega^2, \ldots, a_{n-1}\omega^{n-1}\}$ and, respectively, $\{b_0, b_1\omega, b_1\omega^2, \ldots, b_{n-1}\omega^{n-1}\}$ both with respect to the root of unity ω_n. Put $E_i = A'_i B'_i$ for $0 \le i < n$ and, similarly to the above, let $\{e_0, e_1\omega, e_2\omega^2, \ldots, e_{n-1}\omega^{n-1}\}$ be the DFT of the sequence $\{E_0, E_1, \ldots, E_{n-1}\}$ with respect to the root of unity $1/\omega_n$. Then computing as before we obtain

$$e_h \omega^h = n \sum_{j+k \equiv h \pmod{n}} a_j \omega^j b_k \omega^k.$$

Since in the above sum, as before, either $j+k=h$ or $j+k=n+h$, we obtain

$$e_h = n(c_h + \omega_s c_{n+h}).$$

Combining the latter two equations, we find that

$$(1-\omega_s) n c_h = e_h - \omega_s d_h, \qquad (1-\omega_s) n c_{n+h} = d_h - e_h.$$

Define

$$\tau_s = \begin{cases} p & \text{if } s \text{ is a power of a prime } p, \\ 1 & \text{if } s \text{ is not a prime-power.} \end{cases}$$

In [9], page 73 (see also [2]), it is shown that

$$\prod_{\substack{1 \le i < s \\ (i,s)=1}} (1-\omega_s^i) = \tau_s,$$

where the product is restricted to those i which are relatively prime to s. It follows that

$$\tau_s/(1-\omega_s) = \prod_{\substack{2\le i<s \\ (i,s)=1}} (1-\omega_s^i),$$

and that, if $d \in \mathcal{D}$, then $d\tau_s/(1-\omega_s)$ can be computed using $\phi(s)-1$ additions/subtractions and $\phi(s)-1$ multiplications, the latter all being by powers of ω_s.

Summarizing, we have proved

LEMMA 2.2. *If*

$$\left(\sum_{i=0}^{n-1} a_i x^i\right)\left(\sum_{i=0}^{n-1} b_i x^i\right) = \sum_{i=0}^{2n-2} c_i x^i,$$

we can compute the elements $\tau_s n c_i$ by (1) performing the DFT of order n 6 times, (2) $2n\phi(s)$ additions/subtractions, (3) $n(2\phi(s)+1)$ multiplications by roots of unity which are powers of ω_{ns}, and (4) $2n$ other multiplications (of polynomials in ω_{ns}).

Let q be a positive integer $\ge r+1$. We now estimate the complexity of calculating the DFT of polynomials in the integral domain $\mathcal{D} = \mathbb{Z}[\omega_{s^q}]$, of cyclotomic integers, where $\omega = \omega_{s^q}$ is a primitive $(s^q)^{\text{th}}$ root of unity in an algebraic closure of $\mathbb{Z}$. The roots of unity used in the above multiplication method all lie in $\mathcal{D}$. In what follows, we put $\omega_{ns} = \omega^{s^{q-r-1}}$, $\omega_n = \omega_{ns}^s$, and $\omega_s = \omega_{ns}^n$; then ω_{ns} is a primitive ns^{th} root of unity, ω_n is a primitive n^{th} root of unity, and ω_s is a primitive s^{th} root of unity. Recall that the cyclotomic polynomial $\Phi_{s^q}(z) = \Phi_s(z^{s^{q-1}})$. We use as a basis for the ring of cyclotomic integers the powers $1, \omega, \omega^2, \ldots, \omega^{\phi(s^q)-1}$ and suppose that the input coefficients a_i and b_i are polynomials with integral coefficients in ω of degree $< \phi(s^q) = s^{q-1}\phi(s)$. We may replace ω by an indeterminate X and calculate the DFT in the ring $\mathbb{Z}[X]/\Phi_{s^q}(X)$. Since $\Phi_{s^q}(X)$ divides $X^{s^q}-1$, we may first perform the calculations in the ring $\mathbb{Z}[X]/(X^{s^q}-1)$ and then reduce the answer (mod $\Phi_{s^q}(X)$). Computing in the ring $\mathbb{Z}[X]/(X^{s^q}-1)$ amounts to computing with polynomials in X of degree $< s^q$. Addition of two such polynomials requires $\le s^q$ integer additions/subtractions. Multiplication by a power of ω_{s^q}, or equivalently, in this ring, by a power of X, requires no computation, for multiplying a polynomial by X and reducing it (mod $X^{s^q}-1$) amounts to a cyclic shift of its coefficients. Finally, after the DFT is completed, the $n = s^r$ polynomials in the result must be reduced (mod $\Phi_{s^q}(X)$). That is, we must divide each such polynomial by $\Phi_{s^q}(X)$ and keep the remainder. Now, $\Phi_{s^q}(X)$ is monic and has integral coefficients. If μ_s denotes the sum of the absolute values of the coefficients of $\Phi_s(X)$, then the reduction of a polynomial of degree $< s^q$ mod $\Phi_{s^q}(X)$ requires $\le (s^q - \phi(s^q))\mu_s$ additions/subtractions and **no** multiplications. Note that the division with remainder algorithm performs $s^q - \phi(s^q)$ subtractions of scalar multiples of $\Phi_{s^q}(X)X^l$, $0 \le l \le s^q - \phi(s^q) - 1$, each of which can be carried out in μ_s additions/subtractions. E.g., for an integral coefficient t of $\Phi_{s^q}(X)$, we compute $d_1 - td_2$, $d_1, d_2 \in \mathcal{D}$, by subtracting d_2 t times from d_1. Estimating crudely, we obtain

LEMMA 2.3. *Performing the DFT of order $n = s^r$ in the ring $\mathcal{D} = \mathbb{Z}[\omega_n]$,where $q \ge r+1$, requires no more than $(r(s-1)\mu_s)s^{q+r}$ additions/subtractions and no multiplications.*

We now consider the complexity of multiplying two polynomials of degree $< n$ with coefficients restricted to the ring $\mathbb{Z}[\omega_{s^q}]$, calculating as above. This can be obtained by combining Lemmas 2.2 and 2.3. Note that we need not count multiplications by powers of ω_{s^q}. Thus we obtain:

LEMMA 2.4. *Suppose that* $\sum_{i=0}^{n-1} a_i x^i \sum_{i=0}^{n-1} b_i x^i = \sum_{i=0}^{2n-2} c_i x^i$ *where the* a_i *are elements of the ring* $\mathcal{D}$ *of cyclotomic integers* $\mathbb{Z}[\omega_{s^q}]$*, represented in the standard basis consisting of powers of* ω_{s^q}*, and that* $n = s^r$ *with* $r < q$*. Then, we can compute the elements* $\tau_s n c_i$ *in* $\leq 6rs\mu_s s^{q+r}$ *integer additions/subtractions and* $\leq 2n$ *multiplications of elements of* $\mathcal{D}$*.*

3. Polynomial multiplication. We first consider the problem of multiplying cyclotomic integers. Suppose $M = s^Q$, that ω_M is a primitive M^{th} root of unity and that we wish to multiply two cyclotomic integers, i.e., elements of the ring $\mathbb{Z}[\omega_M]$, say $A = \sum_{i=0}^{\phi(M)-1} a_i\omega_M^i$ and $B = \sum_{i=0}^{\phi(M)-1} b_i\omega_M^i$. If $Q \leq 2$ we shall simply multiply them, in the ordinary way, as polynomials in ω_M, reducing $(\bmod\ \Phi_M(\omega_M))$. Otherwise let $H \geq 0$ be the unique integer such that $2^H + 2 \leq Q \leq 2^{H+1} + 1$. Put $q = \lfloor Q/2 \rfloor + 1$, and $r = \lceil Q/2 \rceil - 1$. Then $q + r = Q$, $q - r$ is 1 or 2, r is ≥ 1 and $2^{H-1} + 2 \leq q \leq 2^H + 1$. Put $m = s^q$ and $n = s^r$. Note that if Q is even, we have $m = s^2 n$, while if Q is odd, we have $m = sn$. In both cases $mn = M$. We shall reduce the desired multiplication to $2n$ multiplications in the smaller cyclotomic ring $\mathbb{Z}[\omega_m]$.

We can write

$$
\begin{aligned}
A &= \sum_{i=0}^{\phi(M)-1} a_i\omega_M^i \\
&= \sum_{j=0}^{n-1} \sum_{i=0}^{\phi(M)/n-1} a_{in+j}\omega_M^{in+j} \\
&= \sum_{j=0}^{n-1} \left(\sum_{i=0}^{\phi(m)-1} a_{in+j}\omega_m^i \right) \omega_M^j,
\end{aligned}
$$

where $\omega_m = \omega_M^n$ is a primitive m^{th} root of unity. Similarly, we can write

$$
B = \sum_{j=0}^{n-1} \left(\sum_{i=0}^{\phi(m)-1} b_{in+j}\omega_m^i \right) \omega_M^j.
$$

Now put

$$
a(x) = \sum_{j=0}^{n-1} \left(\sum_{i=0}^{\phi(m)-1} a_{in+j}\omega_m^i \right) x^j
$$

and

$$
b(x) = \sum_{j=0}^{n-1} \left(\sum_{i=0}^{\phi(m)-1} b_{in+j}\omega_m^i \right) x^j.
$$

so that $a(\omega_M) = A$ and $b(\omega_M) = B$.

If we form the product

$$\begin{aligned} c(x) &= a(x)b(x) \\ &= \sum_{j=0}^{2n-2} \left(\sum_{i=0}^{\phi(m)-1} c_{ij}\omega_m^i \right) x^j, \end{aligned}$$

then $C = c(\omega_M)$ is the desired product; i.e., $C = AB$. However, we shall need to express C in the same form as A and B, i.e., as a polynomial in ω_M of degree $< \phi(M)$. Now,

$$\begin{aligned} C &= \sum_{j=0}^{2n-2} \left(\sum_{i=0}^{\phi(m)-1} c_{ij}\omega_m^i \right) \omega_M^j \\ &= \sum_{j=0}^{n-1} \sum_{i=0}^{\phi(m)-1} \left(c_{ij}\omega_M^{ni+j} + c_{i,j+n}\omega_M^{ni+j+n} \right) \\ &= \sum_{j=0}^{n-1} \sum_{i=0}^{\phi(m)} (c_{ij} + c_{i-1,j+n})\omega_M^{ni+j}, \end{aligned}$$

where we define c_{ij} to be 0 if $i < 0$, $i \geq \phi(m)$ or $j \geq 2n-1$. If we carry out the indicated additions, then we will express C as a polynomial in ω_M of degree $< \phi(M) + n$. Thus n reductions modulo the polynomial $\Phi_M(\omega_M)$ will leave C in the required form.

All of this works equally well if we compute K_Q times $a(x)b(x)$ instead of $a(x)b(x)$, for any $K_Q \in \mathbb{Z}$. Of course, we will obtain $K_Q C$ instead of C, but all else remains the same. Summarizing, we have

LEMMA 3.1. *We can compute K_Q times the product of two cyclotomic integers in $\mathbb{Z}[\omega_M]$ by computing K_Q times the product of two polynomials of degree $< n$ over $\mathbb{Z}[\omega_m]$ and $\leq M + n\mu_s$ additional additions/subtractions.*

We now show, inductively, that the method of the previous section, used recursively, enables us to choose $K_Q = \tau_s^{H+1}s^{Q-2}$ when $Q \geq 3$.

Indeed, if $Q = 3$, then $H = 0$, $q = 2$, $n = s$, and the $2n$ products necessary for the above computation are computed directly (using ordinary polynomial multiplication) and we obtain $K_Q = \tau_s s = \tau_s^{H+1}s^{Q-2}$. When $Q > 3$, then, recursively, using Lemma 2.4, we obtain $K_Q = \tau_s s^r K_q = \tau_s s^r \tau_s^{h+1} s^{q-2}$, where $h = H - 1$ is the unique integer satisfying $2^h + 2 \leq q \leq 2^{h+1} + 1$. This shows that $K_Q = \tau_s^{H+1}s^{Q-2}$.

Now denote by α_Q the number of addition/subtractions and by β_Q the number of multiplications required for the above computation.

LEMMA 3.2. *We have*

$$\alpha_Q \leq s^Q 2^H((6s+2)\mu_s(H+1) + 2\alpha_2/s^2) \text{ and } \beta_Q \leq s^Q 2^H \beta_2.$$

PROOF: From Lemmas 2.4 and 3.1 we find that

$$\begin{aligned}\alpha_Q &\le 6rs\mu_s s^Q + 2s^r\alpha_q + s^Q + s^r\mu_s\\ &\le (r(6s+2)\mu_s + 2\alpha_q/s^q)s^Q\\ &\le (2^H(6s+2)\mu_s + 2\alpha_q/s^q)s^Q.\end{aligned}$$

When $Q = 3$, then $H = 0$, $q = 2$, $r = 1$, $n = s$, and the desired result for α_3 follows from Lemmas 2.4 and 3.1, namely,

$$\alpha_3 \le 6s\mu_s s^3 + 2s\alpha_2 + s^3 + s\mu_s \le s^3((6s+2)\mu_s + 2\alpha_2/s^2).$$

If $Q > 3$ then $H > 1$ and we have, inductively,

$$\begin{aligned}\alpha_Q/s^Q &\le 2^H(6s+2)\mu_s + 2\alpha_q/s^q\\ &\le 2^H(6s+2)\mu_s + 2\cdot 2^h((6s+2)\mu_s(h+1) + 2\alpha_2/s^2)\\ &\le 2^H((6s+2)\mu_s(H+1) + 2\alpha_2/s^2).\end{aligned}$$

Similarly, from Lemmas 2.4 and 3.1, we find that $\beta_Q \le 2s^r\beta_q$ and by induction we obtain the desired result. ∎

Now, to multiply two polynomials with integral coefficients, $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) = \sum_{i=0}^{n-1} b_i x^i$, each of degree $< n$, we simply choose Q so that that $\phi(s^Q) \ge 2n$ and multiply the cyclotomic integers $a(\omega_m)$ and $b(\omega_m)$. The coefficients of the product c expressed as a polynomial of degree $< \phi(s^Q)$ will be the coefficients of the desired product polynomial.

From what we have proven above, we can compute an integer multiple $Na(x)b(x)$ using $O(n \log n \text{ loglog } n)$ addition/subtraction steps and $O(n \log n)$ multiplications. The multiplier N may be chosen to be a positive power of s, or a power of the unique prime p dividing s if s is a prime power, and may be chosen to be $O(n \log n)$ (indeed, if s is not a prime, then it may be chosen $O(n)$).

Now, as described at the beginning of this paper, this method remains valid when the polynomials have coefficients in $\mathcal{A}$. We can therefore, as described earlier, choose two different, relatively prime, integers s_1 and s_2, and compute $N_1 a(x)b(x)$ using s_1 and $N_2 a(x)b(x)$ using s_2. Then N_1 and N_2 will be relatively prime and there will be integers M_1 and M_2 such that $M_1N_1 + M_2N_2 = 1$. Using the Euclidean algorithm, or otherwise, we can choose $|M_1| \le N_2$ and $|M_2| \le N_1$. We must compute the sum $M_1(N_1a(x)b(x)) + M_2(N_2a(x)b(x))$. By repeated doubling any coefficient of $M_1(N_1a(x)b(x))$ may be computed in $O(\log M_1) = O(\log n)$ additions/subtractions. Thus, the entire sum may be computed in $O(n \log n)$ additions/subtractions. This completes the proof of our main theorem:

THEOREM. *There exists a bilinear algorithm which computes the product of two polynomials of degree $< n$ with coefficients in $\mathcal{A}$ using $O(n \log n$ loglog $n)$ addition/subtractions and $O(n \log n)$ multiplications. The algorithm is bilinear and the constant implied by the O does not depend upon $\mathcal{A}$.*

Acknowledgement: We like to thank the referees for several comments that have improved this paper; one referee brought to our attention the references [4], [5], and [7].

References

1. Aho, A., Hopcroft, J., Ullman, J.: The design and analysis of computer algorithms. Reading (Mass.): Addison-Wesley 1974.
2. Apostol, T. M.: Resultants of cyclotomic polynomials. Proc. Amer. Math. Soc. **24**, 457–462 (1970).
3. Cantor, D. G.: On arithmetical algorithms over finite fields, J. Combinatorial Theory, Series A **50**, 285–300 (1989).
4. Chudnovsky, D. V., Chudnovsky, G. V.: Algebraic complexities and algebraic curves over finite fields. J. Complexity **4**, 285–316 (1988).
5. Grigoriev, D. Yu.: Multiplicative complexity of a pair of bilinear forms and of the polynomial multiplication. Proc. 7th MFCS, Springer Lect. Notes Comput. Sci. **64**, 250–256 (1978).
6. Kaltofen, E.: Greatest common divisors of polynomials given by straight-line programs. J. ACM **35**, 231–264 (1988).
7. Kaminski, M.: An algorithm for polynomial multiplication that does not depend on the ring of constants. J. Algorithms **9**, 137–147 (1988).
8. Knuth, D. E.: The art of computer programming, vol. 2, ed. 2. Reading (Mass.): Addison-Wesley 1981.
9. Lang, S.: Algebraic number theory. Reading (Mass.): Addison-Wesley 1970.
10. Lempel, A., Seroussi, G., Winograd, S.: On the complexity of multiplication in finite fields. Theoret. Comput. Sci. **22**, 285–296 (1983).
11. Nussbaumer, H. J.: Fast polynomial transform algorithms for digital convolutions. IEEE Trans. ASSP **28**, 205–215 (1980).
12. Schönhage, A.: Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. Acta Inf. **7**, 395–398 (1977).
13. Schönhage, A, Strassen, V.: Schnelle Multiplikation grosser Zahlen. Computing **7**, 281–292 (1971).
14. Winograd, S.: Arithmetic complexity of computations. CBMS-NSF Regional Conference Series in Applied Math. 33, Philadelphia, PA: SIAM 1980.

Keywords. multiplication, fast, polynomials, algorithm

David G. Cantor
Department of Mathematics
University of California
Los Angeles, CA 90024-1555

Erich Kaltofen
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180-3590