ASYMPTOTICALLY FAST ALGORITHMS FOR THE NUMERICAL MULTIPLICATION AND DIVISION OF POLYNOMIALS WITH COMPLEX COEFFICIENTS

Arnold Schönhage

Mathematisches Institut
der Universität Tübingen
Auf der Morgenstelle 10
D 74 Tübingen, W-Germany

Abstract. Multiplication of univariate n-th degree polynomials over $\mathbb{C}$ by straight application of FFT's carried out numerically in ℓ-bit precision will require time $O(n \log n \, \psi(\ell))$, where $\psi(m)$ bounds the time for multiplication of m-bit integers, e.g. $\psi(m) = cm$ for pointer machines or $\psi(m) = cm \cdot \log(m+1) \cdot \log\log(m+2)$ for multitape Turing machines. Here a new method is presented, based upon long integer multiplication, by which even faster algorithms can be obtained. Under reasonable assumptions (like $\ell \geq \log(n+1)$, and on the coefficient size) polynomial multiplication and discrete Fourier transforms of length n and in ℓ-bit precision are possible in time $O(\psi(n\ell))$, and division of polynomials in $O(\psi(n(\ell+n)))$. Included is also a new version of integer multiplication $\bmod(2^N+1)$.

Introduction

A central topic of my lecture is how to multiply polynomials numerically. I am talking about univariate polynomials with complex coefficients, say of modulus less than one and with ℓ-bit precision. This type of problem leads us to the very heart of computer algebra. On the one hand the underlying algebraic structure suggests an analysis in the framework of algebraic complexity theory, i.e., we have to study polynomial multiplication over $\mathbb{C}$, where all arithmetic steps are to be counted. For this model of straight line program complexity it is well known since the early seventies that (dealing with n-th degree polynomials) the basic algorithm of order $O(n^2)$ can be replaced by an asymptotically fast method with time bound $O(n \log n)$, namely by means of fast Fourier transforms - FFT (cf. [1]). Many investigators have conjectured that this bound (with respect to its

order) is optimal, but until now nobody was able to establish any non-linear lower bound for polynomial multiplication or for discrete Fourier transforms in the algebraic model.

On the other hand, in order to cope (at least theoretically) with all those additional problems that arise when we try to implement such fancy FFT algorithms in a real computer environment, we also have to investigate complexity issues related to some machine model which adequately reflects the main features of our computer in use. In that respect a crucial point is how to encode and how (or where) to store ℓ-bit numbers. Provided a sufficiently large central high-speed storage device is available such that linked list structures can be used, then a RAM model can furnish a good approximation to reality. As an especially clean model of that kind I like to mention the pointer machines described in [4]. Otherwise, if frequent data transfers between main and auxiliary storage are necessary (e.g. in case of a micro processor), our theoretical model should perhaps be a multitape Turing machine. With respect to large scale applications the latter one seems to be more realistic.

A natural subproblem of polynomial multiplication is the zero degree case, i.e. the multiplication of two complex numbers (given in ℓ-bit precision), which can be reduced in the usual way to 4 integer multiplications (or to 3 mults at the cost of some extra additions). At this point the choice of the machine model becomes decisive. According to the state of the art, for multitape TMs the asymptotically best complexity bound for m-bit integer multiplication is

$$\psi(m) = cm \cdot \log(m+1) \cdot \log \log(m+2) \ ,$$

where the anonymous constant c will depend on the implementation (see [5], or [2, Sec. 4.3.3]). In the case of pointer machines, however, integer multiplication is possible in linear time [4], thus $\psi(m) = cm$ could be used. In any case it lies beyond the scope of this lecture to cover the details of an implementation, and I must confess not to have spent much effort on that so far. It should be stressed, however, that it is mainly via the multiplication of rather long integers that the machine model will influence the effectiveness of all the other algorithms I am going to talk about.

There is this strange phenomenon that we know asymptotically fast algorithms now for more than ten years while usually the multiple precision software still uses the standard procedures. In their monograph [1, p. 88] Borodin and Munro comment on the Schönhage-Strassen integer multiplication algorithm in this way: "However, it is not at

all clear that the method could ever be practically implemented." - Of course, I never could share such an extreme pessimism. Today's computer software and hardware contains several biases against our method. Thus a shift of a binary word by 25 places, for instance, will roughly take as much time as a full 32×32 bit multiplication, etc. Admittedly, in its original version our method was somewhat clumsy. Therefore I want to give a brief outline of a slightly modified version in § 1, which seems to be more feasible especially for numbers of moderate length.

The main new idea will be presented in § 2. Simply combining the FFT approach of the algebraic model with fast integer multiplication will yield 'only' the bound $O(n \log n\, \psi(\ell))$. Instead, we can reduce polynomial multiplication to integer multiplication of size $O(n(\ell+\log n))$, and that will lead to the better bound $O(\psi(n\ell))$, provided $\ell \geq \log(n+1)$. The saving is by a factor of order $\min(\log n, \log \ell)$. The same improvement is obtained for discrete Fourier transforms in § 3.

In the algebraic model the <u>division of polynomials</u> is reduced to multiplication via computing reciprocals of formal power series by a Newton type iterative procedure (Sieveking [6], Kung [3]). Principally it is possible to mimic this approach numerically. One has, however, to master additional problems of numerical stability. After a careful reformulation of the problem I found it much easier to work with discrete Fourier transforms in a direct manner. The reciprocal of a polynomial is a Laurent series whose coefficients are obtained by approximately evaluating Cauchy's integral formula. In this way division of polynomials is possible in time $O(\psi(n(\ell+n)))$. The details are described in § 4. With regard to the shape of this bound I would like to state as a general rule of thumb that the coefficients of an n-th degree polynomial usually should be given with a precision ℓ which is of order n at least. Then our complexity bound for division is the same again as for multiplication.

1. Integer multiplication $\bmod(2^N+1)$

In [5] the original algorithm was restricted to the Fermat numbers 2^N+1 with N being a power of 2 . Here the <u>suitable numbers</u> N will have the form

(1.1) $\qquad N = \nu 2^n$ with $n-1 \leq \nu \leq 2n-1 \qquad (n \geq 4)$.

The representation is unique; in particular all numbers 2^r can be

written in this way $(r \geq 6)$.

Let us consider now such an $N \geq 5 \cdot 2^6 = 320$, hence $n \geq 6$. I want to describe how to reduce the multiplication $A*B \bmod(2^N+1)$ to a collection of smaller multiplications $\bmod(2^K+1)$, where $K = \kappa 2^k$ will be suitable again. The idea is to use FFT's $\bmod(2^K+1)$. We split A and B into 2^m pieces each,

$$(1.2) \qquad A = \sum_{i=0}^{2^m-1} a_i 2^{i\nu 2^{n-m}} , \qquad B = \sum_{j=0}^{2^m-1} b_j 2^{j\nu 2^{n-m}}$$

$$\text{with } 0 \leq a_i, b_j \leq 2^{\nu 2^{n-m}} .$$

In view of $2^N \equiv -1 \bmod(2^N+1)$ the product $A*B \bmod(2^N+1)$ takes the form

$$(1.3) \qquad \sum_{\mu=0}^{2^m-1} C_\mu 2^{\mu\nu 2^{n-m}} \quad \text{with} \quad C_\mu = \sum_{\substack{i,j \\ i+j=\mu}} a_i b_j - \sum_{\substack{i,j \\ i+j=2^m+\mu}} a_i b_j .$$

How large must K be chosen such that these integers C_μ are uniquely representable by residues $\bmod(2^K+1)$? - An easy argument shows that

$$(1.4) \qquad K \geq \nu 2^{n-m+1} + m + 1$$

is sufficient. Moreover, let K be a multiple of 2^m (which is certainly true if $k \geq m$), $K = d \cdot 2^m$. Then $w = 2^d$ is a primitive 2^m-th root of $-1 \bmod(2^K+1)$, therefore

$$(1.5) \qquad C_\mu \equiv w^{-\mu} \sum_{\substack{i,j \\ i+j \equiv \mu \bmod 2^m}} (w^i a_i)(w^j b_j) \qquad \bmod(2^K+1) .$$

This wrapped convolution formula shows how to compute the desired residues by means of 3 FFT's of order 2^m and 2^m multiplications $\bmod(2^K+1)$ in the usual way, where $w^2 = 2^{2d}$ serves as a primitive 2^m-th root of unity.

Finally we choose (compatible with (1.4) and with $m \leq k$)

$$(1.6) \qquad \kappa = \left\lceil \frac{\nu+1}{2} \right\rceil , \quad k = \lceil n/2 \rceil + 1 , \quad m = \lfloor n/2 \rfloor + 1 ,$$

and then also $k-1 \leq \kappa \leq 2k-2$ holds. Let $t(N)$, $t(K)$ denote the running time of one multiplication $\bmod(2^N+1)$, or $\bmod(2^K+1)$, respectively. Since multiplications $\bmod(2^K+1)$ by powers of $w^2 = 2^{2d}$ can be achieved by proper shifts, the FFT's will require not more than $O(m2^mK)$ bit operations. Thus we have the recursive estimate

$$(1.7) \qquad t(N) \leq 2^m t(K) + O(m2^mK) ,$$

which, by means of (1.1) and (1.6), easily yields

$$\frac{t(\nu 2^n)}{2^n(n-3)(\nu-2)} \le \frac{t(\varkappa 2^k)}{2^k(k-3)(\varkappa-2)} + O(1) .$$

After about log n steps the recursion comes down to earth, therefore $t(N) = O(N \cdot \log N \cdot \operatorname{loglog} N)$.

An Example. $N = 13.2^{12} = 53248$ reduces to $K = 7.2^7 = 896$ with $m = 7$, then the next reduction is to $K' = 4 \cdot 2^5 = 128$ with $m' = 4$. In this way one multiplication $\bmod(2^N+1)$ requires 2048 multiplications $\bmod(2^{128}+1)$ and the connecting FFT's.

Ordinary integer multiplication of ℓ-bit numbers is achievable by a multiplication $\bmod(2^N+1)$ with any suitable $N \ge 2\ell$. Since the ratios of successive suitable N's tend to 1 in the limit, we can choose $N \sim 2\ell$ and need not waste another factor of nearly 2 as could be the case when only powers of 2 were used.

It goes without saying that, on the other hand, integer multiplication $\bmod(2^N+1)$ is at least as fast as ordinary integer multiplication, for instance in time $O(N)$ in the case of pointer machines.

Furthermore it should be mentioned that there exists also an elegant method for multiplying complex (Gaussian) integers via multiplication $\bmod(2^N+1)$, due to the fact that $2^{N/2}$ is a square root of $-1 \bmod (2^N+1)$. We will come back to that in the next paragraph.

2. How to multiply polynomials numerically

Let us consider the task of performing the polynomial multiplication

$$(2.1) \qquad (a_m z^m + \ldots + a_1 z + a_0) \cdot (b_n z^n + \ldots + b_1 z + b_0) = c_{m+n} z^{n+m} + \ldots + c_0 ,$$

where the given coefficients a_μ and b_ν are ℓ-bit precision real or complex numbers, more precisely

$$(2.2) \qquad a_\mu, b_\nu \in 2^{-\ell}(\mathbb{Z}+i\mathbb{Z}),\ |\operatorname{Re} a_\mu|<1,\ |\operatorname{Im} a_\mu|<1,\ |\operatorname{Re} b_\nu|<1,\ |\operatorname{Im} b_\nu|<1 .$$

Equivalently, we can multiply both sides of (2.1) by $2^{2\ell}$. Then the inputs are ℓ-bit integers, and the outputs are integers of modulus less than $2(m+1)2^{2\ell}$ (w.r. let $m \le n$).

The basic idea of my method is displayed best by an example. For simplicity, the coefficients are real and positive, with 2 decimal digits each. The polynomial multiplication

$$(87z^2+45z+73)\cdot(91z^2+29z+46) = c_4z^4+c_3z^3+c_2z^2+c_1z+c_0$$

translates into the integer multiplication

$$870004500073 \cdot 910002900046 = 791706618|11950|04187|03358$$

from which the resulting coefficients c_j can be read off simply as 5-digit segments, e.g. $c_2 = 11950$. Here the leading 'one' shows that one extra digit for overflow protection was indeed necessary.

In general we have to deal also with negative or complex coefficients. Since integer multiplication $\mathrm{mod}(2^N+1)$ with some suitable N shall be used, the following lemma is fundamental.

Lemma 2.1. Let k, L, N be positive integers with $(k+1)L \le N$. Then the congruence

$$(2.3) \qquad \sum_{j=0}^{k} u_j \, 2^{jL} \equiv \sum_{j=0}^{k} v_j \, 2^{jL} \qquad \mathrm{mod}(2^N+1)$$

with integers u_j, v_j bounded by $|u_j|, |v_j| < 2^{L-1}$ implies $u_j = v_j$ for all j.

Proof. Add $\sum_{j=0}^{k} 2^{L-1} 2^{jL}$ on both sides of (2.3) and use $0 < u_j + 2^{L-1} < 2^L$, etc.

In the case of real (integer) coefficients of length $\le \ell$ the polynomial multiplication (2.1) can now be done in the following way: Choose L and N according to

$$(2.4) \qquad L = 2\ell + \lceil \log(m+1) \rceil + 1 \quad \text{and} \quad (m+n+1)L \le N$$

with a minimal suitable N. Then form the $\mathrm{mod}(2^N+1)$ residues

$$A \equiv \sum_{\mu=0}^{m} a_\mu 2^{\mu L}, \qquad B \equiv \sum_{\nu=0}^{n} b_\nu 2^{\nu L}$$

and compute their product $AB \equiv C$ by multiplication $\mathrm{mod}(2^N+1)$. Finally 'extract' from C the results c_j which, in virtue of (2.4) and Lemma 2.1, are uniquely determined. This latter step deserves some extra explanation, as well as the encoding process from the a's to A and from the b's to B. Let us assume that negative numbers are represented by complements, thus any ℓ-bit integer reads u or $u-2^\ell$ with an ℓ-bit sequence u. Then

$$A \equiv \sum_{\mu=0}^{m} (a_\mu + 2^\ell) 2^{\mu L} - \sum_{\mu=0}^{m} 2^\ell 2^{\mu L}$$

shows how to obtain A by a subtraction $\mathrm{mod}(2^N+1)$ which, of course, can be done in one pass together with forming the first sum. The same applies to B, and the final decoding simply follows from

$$C + \sum_{j=0}^{m+n} 2^{L-1} 2^{jL} \equiv \sum_{j=0}^{m+n} (c_j + 2^{L-1}) 2^{jL} \qquad \mathrm{mod}(2^N+1).$$

In the case of complex coefficients $a_\mu = a_\mu^0 + i a_\mu'$, $b_\nu = b_\nu^0 + i b_\nu'$ with ℓ-bit integer components we choose L and a suitable N according to

$$(2.5) \qquad L = 2\ell + \lceil \log(m+1) \rceil + 2 \quad \text{and} \quad N \ge 2(m+n+1)L.$$

Now $2^{N/2}$ will serve as the imaginary unit $\bmod(2^N+1)$. Correspondingly we have to form the $\bmod(2^N+1)$ residues

$$A \equiv \sum_{\mu=0}^{m} a'_\mu 2^{\mu L} + 2^{N/2} \sum_{\mu=0}^{m} a''_\mu 2^{\mu L}, \quad B \equiv \sum_{\nu=0}^{n} b'_\nu 2^{\nu L} + 2^{N/2} \sum_{\nu=0}^{n} b''_\nu 2^{\nu L} .$$

Their product $C \equiv AB \bmod(2^N+1)$ then reads

$$C \equiv \sum_{j=0}^{m+n} c'_j 2^{jL} + 2^{N/2} \sum_{j=0}^{m+n} c''_j 2^{jL} \qquad \bmod(2^N+1) ,$$

with the components of the coefficients c_j being uniquely determined in this way (due to a slightly more general version of Lemma 2.1).

This method of multiplying two polynomials is most effective, if the degrees m and n are of the same order. Otherwise the longer polynomial should be subdivided into pieces of appropriate size. Let us spezialize now to the case $m = n$. The main theoretical result can be summarized as follows.

Theorem 2.2. Multiplication of n-th degree polynomials with complex coefficients of modulus less than one in ℓ-bit precision, $\ell \geq \log(n+1)$, is possible on a multitape Turing machine in time $O(n\ell \cdot \log(n\ell) \cdot \log\log(n\ell))$, on a pointer machine in linear time $O(n\ell)$.

Next let us consider an example. Assume $m = n = 30$, and $\ell = 200$. Such a precision may seem to be chosen artificially high, but just think of two factors of a 60th degree polynomial whose complex roots are to be determined numerically! - By (2.5) we get $L = 407$ and $N \geq 49654 \approx 12.13 \cdot 2^{12}$, hence the next suitable number is $N = 13 \cdot 2^{12}$, $N = 53248$; it occurred already in the example at the end of § 1. There we also mentioned the multiplication of complex integers which is contained in the preceding analysis as the zero degree case $m = n = 0$, where $N \geq 4\ell + 4$ is sufficient.

3. Discrete Fourier transforms

Let us now consider the task of computing the discrete Fourier transform $(u_0,\ldots,u_{p-1}) \mapsto (\hat{u}_0,\ldots,\hat{u}_{p-1})$, where

$$(3.1) \qquad \hat{u}_k = \sum_{j=0}^{p-1} u_j \omega^{kj} , \quad \text{with} \quad \omega = \exp\left(\frac{2\pi i}{p}\right) .$$

The inputs u_j are assumed to be given as complex numbers of modulus less than one in ℓ-bit precision, at least $\ell \geq \log(p+1)$. The outputs $\hat{u}_k$ shall be computed numerically up to an error bounded by some reasonable ε, say $\varepsilon = 9p2^{-\ell}$. There is a nice trick by which this task can be reduced to a convolution, i.e. to polynomial multiplication

(cf. [2], Ch. 4.3.3 - Ex. 8). Express $\omega = v^2$, with $v = \exp(2\pi i/2p)$, and use $\omega^{kj} = v^{2kj}$, $2kj = k^2+j^2-(k-j)^2$. Then $\hat{u}_k$ becomes

$$(3.2) \qquad \hat{u}_k = v^{k^2} \sum_{j=0}^{p-1} \left(u_j v^{j^2}\right) v^{-(k-j)^2} .$$

Correspondingly consider the polynomials $\alpha_{p-1}z^{p-1}+\ldots+\alpha_0$, $\beta_{2p-2}z^{2p-2}+\ldots+\beta_0$ with

$$(3.3) \qquad \alpha_\mu = u_\mu v^{\mu^2}, \quad \beta_{p-1+\nu} = v^{-\nu^2} \qquad (-(p-1)\le\nu\le p-1),$$

and the coefficients of their product, $\gamma_t = \sum_{\mu+\nu=t} \alpha_\mu\beta_\nu$. With regard to (3.2), i.e. $\hat{u}_k = v^{k^2}\gamma_{p-1+k}$, only the γ_t for $p-1 \le t \le 2p-2$ need to be computed.

In an initial phase ℓ-bit precision approximations to the powers of v with errors less than $2\cdot 2^{-\ell}$ are computed. For that purpose one can use $\lambda = \ell + \lceil \log p \rceil + 3$ bits of accuracy for first computing v and then successively multiplying by v, before rounding to ℓ places. All this will require not more than time $O(p\,\psi(\lambda)) = O(p\,\psi(\ell))$. Observe that a λ-bit approximation to v can be obtained by Newton iteration (v is a zero of $f(z) = 1 - 1/z^{2p}$), which is $v_{r+1} = v_r + \frac{1}{2p}(v_r - v_r^{2p+1})$, within time $O(\log(2p+1)\,\psi(\lambda))$, by successively doubling the precision. A suitable starting value v_0 can be found by a few terms of the Taylor expansion of $\exp(\pi i/p)$.

Next the multiplications by u_μ of (3.3) are carried out. Thus we get ℓ-bit approximations a_μ, b_ν to the α's and β's with

$$(3.4) \qquad |a_\mu-\alpha_\mu| < 3\cdot 2^{-\ell}, \quad |b_\nu-\beta_\nu| < 2\cdot 2^{-\ell}.$$

By the method of § 2, where now $m = p-1$, $n = 2p-2$, the polynomial multiplication yields the exact 2ℓ-bit results

$$(3.5) \qquad c_t = \sum_{\mu+\nu=t} a_\mu\, b_\nu \quad \text{with} \quad |c_t-\gamma_t| < 6p2^{-\ell}$$

in time $O(\psi(p\ell))$. Finally another p multiplications (by the k^2-th powers of v) give the desired approximations to the $\hat{u}_k$. On the whole the time bound is $O(\psi(p\ell)) + O(p\,\psi(\ell)) = O(\psi(p\ell))$.

The auxiliary multiplication AB mod(2^N+1) will, according to (2.5), require the parameter values $L = 2\ell + \lceil \log p \rceil + 2$ and a suitable $N \ge 2(3p-2)L$. For practical applications it will be nice, however, to know that we can afford to use a smaller value of N, subject to $N \ge 2(2p-1)L$, thereby saving about one third. The reason is that we are interested only in $c_{p-1},\ldots,c_{2p-2}$. With such a smaller N these coefficients are still properly extractable from the mod(2^N+1) product AB, while the coefficients $c_0,\ldots,c_{p-2}$ and $c_{2p-1},\ldots,c_{3p-2}$ will

overlap in a chaotic way producing some garbage.

4. Numerical division of polynomials

The division of complex numbers up to a relative error less than $2^{-\ell}$ is possible within a time bound of the same order as that for integer multiplication, $O(\psi(\ell))$, by reduction to reciprocals of positive real numbers, which are obtained via Newton iteration (see [2, pp. 295/96], Alg. R.). Let us now consider the division of polynomials with complex coefficients. Algebraically, given F and G,

(4.1) $\quad F(z) = a_0 z^{n+m}+\ldots+a_{n+m}$, $G(z) = b_0 z^n+\ldots+b_n$ with $b_0 \neq 0$, $n \geq 1$,

there exist a quotient $Q(z) = q_0 z^m+\ldots+q_m$ and a remainder H such that $F = QG + H$, unique by the condition $\deg H < n$. Numerically, however, additional quantitative specifications are required. As a norm for polynomials let us use $|F|_1 = |a_0|+\ldots+|a_{n+m}|$, $|G|_1 = |b_0|+\ldots+|b_n|$, etc. Then numerical division of F by G up to an error less than $2^{-\ell}$ shall be defined as the task of computing (some) Q_1 and H_1 such that

(4.2) $\quad |F-(Q_1G+H_1)|_1 < 2^{-\ell}$, $\deg Q_1 \leq m$, $\quad \deg H_1 \leq n-1$.

Reasonable assumptions are $|F|_1 \leq 1$ and $1 \leq |G|_1 \leq 2$. The coefficients of F and G (possibly available with very high accuracy) will be needed as inputs with a certain precision $2^{-\lambda}$ to be specified later. Moreover, we must guarantee that G really does have degree n, for instance by telling how much the leading coefficient b_0 is bounded away from zero. In combination with the condition $1 \leq |G|_1$ I find it preferable instead to put a bound on the root radius of G, which means to specify some positive number r such that

(4.3) $\quad G(z) = b_0 \prod_{j=1}^{n}(z-v_j)$ implies $|v_j| \leq r$ for all j.

This yields

$$1 \leq |G|_1 = |b_0| \sum_k \left|\frac{b_k}{b_0}\right| \leq |b_0| \prod_{j=1}^{n}(1+|v_j|) \leq |b_0|(1+r)^n,$$

whence

(4.4) $\quad |b_0| \geq (1+r)^{-n}$.

The example $F(z) = z^{2n}$, $G(z) = (z-r)^n/(1+r)^n$ with

$$Q(z) = (1+r)^n \sum_{j=0}^{n}\binom{n-1+j}{n-1} r^j z^{n-j}$$

shows the need for controlling this extra parameter r, since the output will have a length of order $n(\ell + n + n\cdot\log(1+r))$. We have, for instance,

(4.5) $\log q_n = n \log(1+r) + n \log(4r) - O(\log n)$.

Thus large values of r may better be handled by suitable scaling before the division is performed; we are mainly interested in the case of uniformly bounded r . As with multiplication the fast algorithms are most effective if the degrees n, m, and $n+m$ are of the same order, say $m \le n \le 2m$. Otherwise reductions to this more special situation can be applied first.

After these preliminary discussions I am ready now to present the main result of this paragraph.

Theorem 4.1. Numerical division of a polynomial F of degree $\le 2n$ and norm $|F|_1 \le 1$ by an n-th degree polynomial G with norm $1 \le |G|_1 \le 2$ and a given bound r on its root radius up to an error less than $2^{-\ell}$ is possible in time

(4.6) $O(\psi(n(\ell+n+n\cdot\log(1+r))))$.

For bounded root radius, $r = O(1)$, this yields the complexity bound mentioned in the Introduction. The proof of Theorem 4.1 will be given subsequently by description of a corresponding algorithm. Its main novel feature is the direct application of discrete Fourier transforms. Thus the asymptotically best result is achieved by the technique of § 3, while with respect to practical applications it is quite conceivable that, for moderate values of n and ℓ , the 'old' FFT is still the best we have.

The main part of the algorithm is the computation of a sufficiently good approximation to the quotient Q . Its coefficients appear in an initial segment of the Laurent series belonging to the auxiliary function f defined for $|z| = R > r$ by

(4.7) $$f(z) = \frac{F(z)/z^m}{G(z)} = \frac{a_0+a_1/z+a_2/z^2+\dots}{b_0+b_1/z+b_2/z^2+\dots} = q_0 + \frac{q_1}{z} + \frac{q_2}{z^2} + \dots$$

Hence the q's are represented by Cauchy's integral formula,

(4.8) $$q_k = \frac{1}{2\pi i} \int_{|z|=R} \frac{F(z)/z^m}{G(z)} z^{k-1}\, dz .$$

From $|F|_1 \le 1$ and (4.3), (4.4) we get the estimates

$$|F(z)\, z^{k-m-1}| \le R^{n+k-1} \qquad (R \ge 1)$$

$$|G(z)| \ge |b_0|(R-r)^n \ge \left(\frac{R-r}{1+r}\right)^n .$$

Therefore (4.8) with $R = 1+r$ yields the bound

(4.9) $$|q_k| \le (1+r)^{2n+k} \qquad (k=0,1,2,\dots) .$$

With regard to the time bound (4.6) we may assume in our further analy-

sis that $m = n$, and $r \geq 1$. Then (4.9) immediately gives

$$(4.10) \qquad |Q|_1 \leq 2(1+r)^{3n} .$$

For $r = 1$ these estimates are rather sharp as can be seen by comparison with (4.5).

Next I have to explain how to use Fourier transforms. The root of unity will be $w = \exp(2\pi i/p)$, where $p = n+1$ for the theoretical proof, while practically p should perhaps be the power of 2 with $m+1 \leq p \leq 2m$. Two Fourier transforms serve to evaluate the numerator and the denominator of the quotient representing f in (4.7) at the points $z_j = Rw^{-j}$, $0 \leq j < p$. Then p divisions with complex numbers yield the values of f at these points,

$$(4.11) \qquad \varphi_j = f(Rw^{-j}) = \frac{\alpha_j}{\beta_j} = \sum_{\mu=0}^{\infty} q_\mu R^{-\mu} w^{\mu j} \qquad (0 \leq j < p) .$$

Now a third Fourier transform (backward this time) produces numbers

$$(4.12) \qquad \gamma_k = \frac{1}{p} \sum_{j=0}^{p-1} \varphi_j w^{-kj} , \qquad (0 \leq k < p),$$

from which we get (as can be checked easily)

$$(4.13) \qquad R^k \gamma_k = q_k + \sum_{\nu=1}^{\infty} q_{k+\nu p} R^{-\nu p} \qquad (0 \leq k < p) .$$

The necessary scaling by powers of R is simplified to pure shifting by choosing $R = 2^s$ with some integer s. Moreover, R must be large such that the right-hand sums in (4.13) become sufficiently small. From $R \geq 2(1+r)$ and (4.9) we obtain

$$|R^k \gamma_k - q_k| \leq \sum_{\nu=1}^{\infty} (1+r)^{2n+k+\nu p} R^{-\nu p} \leq (1+r)^{2n+k} \cdot 2 \cdot \left(\frac{1+r}{R}\right)^p ,$$

and if Q_o denotes the approximation to Q with the coefficients $R^k \gamma_k$, then summation yields

$$|Q_o - Q|_1 \leq 4(1+r)^{3n} \left(\frac{1+r}{R}\right)^p \leq 2(1+r)^{4n} 2^{-sn} .$$

With respect to the final bound (4.2) let us make this less than $2^{-\ell-4}$ by imposing the condition

$$(4.14) \qquad sn > \ell + 5 + 4n \cdot \log(1+r) ,$$

which, by the way, implies also $R = 2^s > (1+r)^4 \geq 2(1+r)$.

Furthermore, assume that the three Fourier transforms are carried out numerically in λ-bit precision. Division of such approximate results, $\tilde{\alpha}_j / \tilde{\beta}_j$, will produce approximations $\tilde{\varphi}_j$, then $\tilde{\gamma}_k$, and the polynomial Q_1 with the coefficients $R^k \tilde{\gamma}_k$ will deviate from Q_o by not more than

$$|Q_1 - Q_o|_1 \leq 2R^n \cdot \max_k |\tilde{\gamma}_k - \gamma_k| .$$

The estimates $|\alpha_j| \le |F|_1 \le 1$ and

$$|\beta_j| = |G(z_j)|R^{-n} \ge |b_0|\left(1 - \frac{r}{R}\right)^n \ge (1+r)^{-n}\cdot 2^{-n}$$

show that all these computations will stay within an error bound of $2^{-\tau}$, where $\tau \ge \lambda - O(n) - n\log(1+r)$ holds. Therefore, $|Q_1 - Q_0|_1 \le 2^{ns-\tau+1}$ will be less than $2^{-\ell-4}$, too, if we impose the further condition

(4.15) $\qquad \tau \ge ns + \ell + 5\ ,\ \lambda \ge ns + \ell + 5 + O(n) + n\cdot\log(1+r)\ .$

This Q_1 is all-right for (4.2). A corresponding H_1 can now be determined as the residue mod z^n of $F_1 - Q_1G_1$, where F_1 and G_1 are suitable finite precision approximations to F and to G, respectively. In view of (4.10), for instance, something like $|G_1 - G|_1 \le 2^{-\ell-5}/(1+r)^{3n}$ must hold. The further details of deriving a final estimate are left to the reader.

The global time analysis starts from specifying the parameters $R = 2^s$ and the precision λ. At first the integer s is chosen minimal under the constraint (4.14), thus $sn = O(\ell + n + n\cdot\log(1+r))$. Then λ can be chosen equal to the right-hand side of (4.15). By the results of § 3 the three Fourier transforms are possible in time $O(\psi(p\lambda))$. The divisions α_j/β_j are done in $O(p\,\psi(\lambda))$. And certainly also the back multiplication for finding the remainder can be accomplished in a similar amount of time. That completes the proof of Theorem 4.1.

5. Conclusion

We can learn from the preceding results that the standard model of algebraic complexity theory, where all arithmetic instructions are counted, does not give adequate bounds for several basic algorithmic problems, when numerical implementation is concerned. Even if somebody can prove non-linear lower bounds for the algebraic model, they do not exclude the existence of faster machine algorithms. This may be one of the deeper reasons for the difficulties in obtaining such lower bounds.

'Fast' integer multiplication is not fast for numbers of moderate length ℓ; multiplication of polynomials by means of 'fast' Fourier transforms is not fast for moderate degree n. In § 2 numbers of length $n\ell$ were multiplied. In this way integer multiplication of numbers with 10 000 to 100 000 bits becomes practically important.

Much can be done to make the computers of tomorrow more adaptable to algorithms of the type presented in this lecture. I hope I could con-

vince you to some extent that something should be done in this respect.

REFERENCES

[1] A. Borodin, I. Munro: The computational complexity of algebraic and numeric problems. New York, American Elsevier, 1975.

[2] D. E. Knuth: The art of computer programming, vol. 2: Seminumerical algorithms. 2nd edition; Reading, Mass. 1981.

[3] H. T. Kung: On computing reciprocals of power series. Numer. Math. 22, 341-348 (1974).

[4] A. Schönhage: Storage modification machines. SIAM J. Comp. 9, 490-508 (1980).

[5] A. Schönhage, V. Strassen: Schnelle Multiplikation großer Zahlen. Computing 7, 281-292 (1971).

[6] M. Sieveking: An algorithm for division of power series. Computing 10, 153-156 (1972).