ELSEVIER

Information Processing Letters 74 (2000) 41–46

Information Processing Letters

www.elsevier.com/locate/ipl

Variations on computing reciprocals of power series

Arnold Schönhage

Institut für Informatik, Universität Bonn, D-53117 Bonn, Germany

Received 25 January 2000
Communicated by T. Lengauer

Abstract

Over infinite fields, inverting $b_0 + b_1x + \cdots + b_nx^n + \cdots \bmod x^{n+1}$ has nonscalar complexity $< 3.75 \cdot n$, versus $2n + 1$ for multiplication mod x^{n+1}. For fields like $\mathbb{C}$ supporting FFTs, and counting all arithmetic operations, we can bound the total cost for inversion mod x^{n+1} by $(13.5 + o(1)) \cdot n \cdot \log n$, versus $\approx 9n \cdot \log n$ (presently the best known) for multiplication. Furthermore we describe a new method how to compute reciprocals by Graeffe root squaring steps and demonstrate this novel approach with the inversion of (monic) power series over $\mathbb{Z}$.

Keywords: Algorithms; Computational complexity; Reciprocals of power series

1. Introduction

In algebraic complexity theory, coefficientwise manipulation of formal power series (or polynomials) $A(x) = a_0 + a_1x + a_2x^2 + \cdots$, $B(x) = b_0 + b_1x + \cdots$ over some ground field F is usually formalized by regarding inputs a_i, b_j as indeterminates over F. So computations actually are viewed to take place in domains $\mathcal{D} = F(a_0, a_1, \ldots, b_0, b_1, \ldots)$ endowed with operations $+$, $-$, $*$, $/$ and scalar multiplications $\lambda\cdot$ for any $\lambda \in F$, while looking at $\mathcal{D}[\![x]\!]$ with dummy variable x supports our overall understanding of corresponding algorithms. Quite formally, *computations* are then regarded as finite result sequences σ in $\mathcal{D}$ reflecting the step by step performance of straight-line programs. A computation σ is said to compute a set $\{f_0, f_1, \ldots\} \subset \mathcal{D}$ iff all f_i are occurring in σ. Initial elements in such computations are the inputs and any constants from F, always taken to be free of cost, all other elements are obtained from previous elements by application of one of the operations. The *total* cost measure charges one unit for each of these steps, while *nonscalar* complexity $C(f_0, f_1, \ldots)$ means minimum cost in computations σ of the f's when counting "essential" multiplications and divisions only, so forming linear combinations from previous elements by scalar multiplications, additions or subtractions is then free of cost—see [4, p. 108] for further background.

In this setting, (asymptotically) fast algorithms for multiplying polynomials admit to construct fast algorithms for division $A(x)/B(x)$ of polynomials or of formal power series up to precision x^n (computing mod x^{n+1}), here in particular to be discussed for large n, where computing reciprocals $R(x) = r_0 + r_1x + r_2x^2 + \cdots \equiv 1/B(x) \bmod x^{n+1}$ is at the very heart of all such methods. One of the most important applications is polynomial division with remainder: for given polynomials

$$a(z) = a_0z^{m+n} + a_1z^{m+n-1} + \cdots$$

and

$$b(z) = b_0z^m + b_1z^{m-1} + \cdots,$$

0020-0190/00/$ – see front matter
PII: S0020-0190(00)00044-2

one finds the unique quotient $q(z)$ in $a(z) = b(z) \cdot q(z) + r(z)$ (with $\deg(r) < \deg(b)$) by switching to the reversed polynomials

$$A(x) = x^{m+n} \cdot a(1/x),$$

$$B(x) = x^{m} \cdot b(1/x).$$

After computation of the reciprocal

$$R(x) \equiv 1/B(x) \bmod x^{n+1},$$

one obtains q in its reversed form by subsequent multiplication

$$Q(x) \equiv A(x) \cdot R(x) \bmod x^{n+1}.$$

Let us restrict our further discussions to infinite ground fields F. Then multiplication mod x^{n+1} has nonscalar complexity $2n + 1$, and Kung [6] has shown $C(r_0, \ldots, r_n) < 4n$ for reciprocals ($n > 0$), but actually his third order iteration [6, Section 3] yields the better bound $< 3.75 \cdot n$, since squaring power series mod x^{k+1} requires only $\lfloor 3k/2 \rfloor + 1$ multiplications (for $\mathrm{char}(F) \neq 2$), see Kalorkoti [5] and [4, Section 2.4] for more details. It is hard to believe, though, that this factor of $15/8$ for reciprocals versus multiplication should be the ultimate answer. In Section 2 we will discuss the same topic for total cost, specializing to fields like $\mathbb{C}$ supporting fast Fourier transforms (FFTs). Then we are able to obtain a better (and more convincing) factor of about $3/2$—so far, of course, only relative to our limited knowledge about the total complexity of discrete Fourier transforms (DFTs) and of polynomial multiplication.

Implementing algorithms for (approximate) *numerical* division of polynomials (over the complex numbers, say) is yet another story. In [7] we have proposed Laurent series combined with FFT methods for this task. Later Bini and Pan [3] have discussed similar issues, they even claim that Bini's approach [2] be algorithmically equivalent to ours. Some of their corresponding comments are, however, rather misleading and deserve a clarifying remark from our side: those algorithms are actually different with respect to essential points. Just note that one of the decisive input parameters of our method (meanwhile successfully implemented, cf. Section 9.1.4 of [8]) is a (good) upper bound for the root radius of the denominator polynomial. Unfortunately, however, all such DFTs over $\mathbb{C}$ share a common drawback. Their favorable asymptotical speed relies on Bluestein's reduction of DFTs to convolutions, cf. [7, Section 3], so that fast polynomial multiplication over $\mathbb{Z}[i]$ can be used. But in discussing improvements by factors close to one, extra factors for such detours lead to unpractically high cross-over points. This has been one of our motivations for the completely new method of Section 3 to compute reciprocals by means of Graeffe root squaring steps, admitting direct reduction to polynomial multiplication. Finally (Section 4) we will describe a numerical version of this method for the simple case of computing reciprocals of (monic) power series over $\mathbb{Z}$.

2. Total cost bounds

Here we assume $F = \mathbb{C}$, likewise any field of characteristic zero containing all roots of unity would do. To recover the coefficients of a polynomial f of degree less than m from its values $f(\omega^j)$ at all Nth roots of unity, one needs $N \geqslant m$, and with the standard choice of minimal $N = 2^\nu \geqslant m$, $\nu = \lceil \log m \rceil$ (here and in the sequel, *log* shall denote the binary logarithm $\log_2$), the total cost complexities of that N-point FFT and of its inverse are bounded by

$$\begin{aligned} L(N) &\leqslant 1.5 \cdot N \cdot \log N - N + \log N + 2 \\ &< 3 \cdot m \cdot \log(2m). \end{aligned} \tag{1}$$

For the FFT itself, the smaller bound $\Lambda(N) = 1.5 \cdot N \cdot \log N - N + 1$ is well known, cf. [4, pp. 8, 32], whereas our bound in (1) for computing the λ-fold multiple of an N-point DFT, i.e., $\hat{a}_l = \lambda \cdot \sum_j a_j \omega^{lj}$ for $0 \leqslant l \leqslant N-1$ (with $\lambda = 1/N$ and $1/\omega$ instead of ω to get the inverse) seems to be new, so let us give a brief outline of a proof: with $H = N/2$, $\tau = 0, 1$, the identity

$$\hat{a}_{l+\tau H} = \lambda \cdot \sum_{j=0}^{H-1} a_{2j} (\omega^2)^{lj} \pm (\lambda \cdot \omega^l) \cdot \sum_{j=0}^{H-1} a_{2j+1} (\omega^2)^{lj} \tag{2}$$

for $0 \leqslant l < H$ shows how to reduce such a scaled DFT to two half size problems, now without scaling in the second sum, since λ has been combined with those twiddle factors. Accounting for these H extra scalar multiplications, plus H additions (for $\tau = 0$) and H subtractions (for $\tau = 1$) we thus have $L(N) \leqslant L(N/2) + \Lambda(N/2) + 3/2 \cdot N$, which yields (1) by induction.

Restriction of N to powers of two in combination with values of m just a bit greater than $N/2$ leads to the amplified estimation factor of $3 \cdot m$ in (1), but for our present discussion it is good to know that one can avoid such overshooting (at least asymptotically) up to lower order terms.

Proposition 1. *Let $m \mapsto N = N(m) \geqslant m$ be defined as $N(m) = k \cdot 2^\nu$, where*

$$\nu = \lceil \log m \rceil - \lfloor \log\log(m+1) \rfloor$$

and $k = \lceil m/2^\nu \rceil$. Then N-point DFT and its inverse have total complexities

$$L(N) \leqslant m \cdot \big(1.5 \cdot \log m + 6.5 \cdot \log\log(m+1) + \mathrm{O}(1)\big). \qquad (3)$$

Proof. First we show $L(k \cdot 2^h) \leqslant k \cdot 2^h(1.5 \cdot h + 8 \cdot \log k + 1)$ for all $h \in \mathbb{N}$. The case $h = 0$ is covered by a uniform total cost bound of $8k \cdot \log k$ from [1] for any k-point DFT, adding k scalings for the inverse, then use the recursion (2) for induction in h. Next we set

$$h = \nu < \log m - \log\log(m+1) + 2$$

and observe that $\nu \leqslant \log m$ implies

$$k = \left\lceil \frac{m}{2^\nu} \right\rceil < \frac{(m+2^\nu)}{m} \leqslant \frac{2m}{2^\nu} \leqslant 2 \cdot \log(m+1),$$

hence $8 \cdot \log k < 8 + 8 \cdot \log\log(m+1)$. Finally, replacing the outer factor N by m is estimated by $N/m = k \cdot 2^\nu/m < (m+2^\nu)/m \leqslant 1 + 4/\log(m+1)$, the effect of this change is thus absorbed by the O(1) term in (3). □

With this result, we are prepared to derive a smooth total cost bound for the multiplication of two power series $A(x) \cdot B(x) \bmod x^{n+1}$, which can simply be done by forming the product $P(x) = A_n(x) \cdot B_n(x)$ of the nth degree segments and discarding the upper n coefficients in the $\mathrm{O}(x^{n+1})$ part of P. To compute P, we choose $N = N(m)$ for $m = 2n+1$ according to Proposition 1, set $\omega = \exp(2\pi \mathrm{i}/N)$, evaluate A_n, B_n at the N points ω^j $(0 \leqslant j < N)$ by two DFTs, perform pointwise multiplication $P(\omega^j) := A_n(\omega^j) \cdot B_n(\omega^j)$, and then a final backward DFT yields the coefficients of P. In virtue of (3), all this has total cost $\leqslant 3L(N) + N \approx 9n \cdot \log n$ for large n.

Let us now turn to the case of reciprocals $R_n(x) = 1/B(x) \bmod x^{n+1}$. Simply carrying out all steps of Kung's third order iteration in this manner, adding up such total cost bounds for all (smaller) multiplications involved would apparently result in the preceding bound amplified by the factor 15/8, thus amount to $\approx 135/8 \cdot n \cdot \log n$. Surprisingly, however, here Kung's *second* order iteration (economically implemented) yields the following better bound.

Theorem 2. *Over ground fields like $\mathbb{C}$ supporting FFTs, computing reciprocals $1/B(x) \bmod x^{n+1}$ has total complexity $\leqslant (13.5 + \mathrm{o}(1)) \cdot n \cdot \log n$.*

Proof. The underlying algorithm works recursively, with $k = \lfloor n/2 \rfloor$ first computing

$$R_k(x) = 1/B(x) \bmod x^{k+1}$$

at total cost of $tc(k)$—initially starting from $tc(1) = 3$ for $R_1(x) = 1/b_0 - b_1/b_0^2 \cdot x$. As

$$R_k(x) \cdot B(x) \equiv 1 \bmod x^{k+1}$$

implies

$$R_k(x) \cdot B_n(x) = 1 + x^{k+1} D(x)$$

with a polynomial D of degree $n-1$, we have

$$\frac{1}{B(x)} \equiv \frac{R_k(x)}{1 + x^{k+1} D(x)} \equiv R_k(x) - x^{k+1} R_k(x) \cdot D(x) \bmod x^{n+1},$$

so it suffices for extension to R_n to compute D and the $n-k$ lower coefficients of $E = R_k \cdot D$. Here we choose $N = N(m)$ for $m = n + k + 1 \leqslant 3n/2 + 1$, $\omega = \exp(2\pi \mathrm{i}/N)$, evaluate $R_k(\omega^j)$ and $B_n(\omega^j)$ by two N-point DFTs, and compute the values

$$E(\omega^j) = R_k(\omega^j) \cdot \big(R_k(\omega^j) \cdot B_n(\omega^j) - 1\big) \cdot \omega^{-j(k+1)},$$

from which the coefficients of E are obtained by a backward DFT. Accordingly this computation of R_n has total cost bound

$$tc(n) \leqslant tc(k) + 3L(N) + 4N \leqslant tc(k) + \big(27/4 + \mathrm{o}(1)\big) \cdot n \cdot \log n$$

by means of (3) with $m \leqslant 3n/2 + 1$. A routine argument applied to this recursive estimate finally leads to the complexity bound of the theorem. □

3. Reciprocals by repeated root squaring

Here F may be any infinite field. Graeffe's root squaring method proceeds from any $f(x) = b_0 + b_1x + \cdots + b_nx^n$ to the even polynomial $g(x^2) = f(x) \cdot f(-x)$, the coefficients in $g(y) = \sum_{m=0}^{n} c_m y^m$ being determined as

$$c_m = (-1)^m(b_m^2 - 2b_{m-1}b_{m+1} + 2b_{m-2}b_{m+2} - \cdots). \tag{4}$$

Looking at $1/B(x) = B(-x)/(B(x) \cdot B(-x))$, we see immediately how to exploit this for computing reciprocals of power series mod x^{n+1}. Application with $f = B_n$ reduces that task to a smaller problem with only half the number of significant terms in the denominator.

Algorithm 3.
Inputs are $n \in \mathbb{N}$ and coefficients $b_0, \ldots, b_n$ of $B(x) = \sum b_j x^j$, where $b_0 \neq 0$.
Returns the coefficients of $R_n(x) = 1/B(x) \bmod x^{n+1}$.
If $n = 0$ **then return** $R_0 = 1/b_0$;
else set $k = \lfloor n/2 \rfloor$;
perform Graeffe step to compute $G_k(y) = c_0 + \cdots + c_k y^k$ with $G_k(x^2) \equiv B_{2k}(x) \cdot B_{2k}(-x) \bmod x^{2k+1}$,
determine $H_k(y) = 1/G_k(y) \bmod y^{k+1}$ by calling Algorithm 3 recursively;
return $R_n(x) = B_n(-x) \cdot H_k(x^2) \bmod x^{n+1}$.

The recursive call is justified by $c_0 = b_0^2 \neq 0$; note that $H_k = 1/b_0^2 + \cdots$ will usually be different from $R_k = 1/b_0 + \cdots$. Let us analyze the number $s(n)$ of *nonscalar* steps performed by this algorithm (formally regarded as a straight-line computation after unrolling the recursive calls, so there is just one division step $1/b_0^{2^l}$ for the innermost call). The $2k+1$ coefficients of the even product $G(x^2) = B_{2k}(x) \cdot B_{2k}(-x)$ require at most $2k+1$ essential multiplications, e.g., by evaluation of $B_{2k}(\pm\xi_j)$ for $2k+1$ distinct values ξ_j^2 in the ground field F (such linear combinations are free of cost), $2k+1$ pointwise multiplications and subsequent interpolation (again free of cost). After the recursive call further $n + 2k + 1$ essential multiplications suffice to compute $B_n(-x) \cdot H_k(x^2)$. Therefore $s(n) \leqslant n + 4k + 2 + s(k)$, with $s(0) = 1$, which yields $s(n) \leqslant 6n + 2 \cdot \log(n/2)$, with equality for powers of two.

This estimate is weaker than Kung's bound of $4n$, and similarly total cost bounds for this algorithm (over $F = \mathbb{C}$, say) are weaker than in Theorem 2. On the other hand, one advantage of this approach is its extreme simplicity, ready for an immediate implementation by standard routines. When dealing with numerical approximations, for instance with input of an upper bound γ for the root radius ρ of the denominator polynomial $B_n^*(x) = b_0x^n + \cdots + b_n$, the recursion in Algorithm 3 poses the additional problem to estimate the root radius ρ' of the shorter polynomial G_k^*, opposed to the root radius of G_n^* (which is $\rho^2 \leqslant \gamma^2$). We have a proof that then always $\rho' \leqslant \rho^2(3 + \sqrt{8})$, but that is beyond the scope of this paper, so let us now turn to the simpler case of integer coefficients.

4. Inverting power series over the integers

Division by monic integer polynomials $z^n + b_1z^{n-1} + \cdots + b_n$ of large size motivates us to consider inversion of units in the ring $\mathbb{Z}[\![x]\!]$, so we will now be concerned with an asymptotically efficient implementation of Algorithm 3 for the case $B(x) = 1 + b_1x + b_2x^2 + \cdots$ with coefficients $b_j \in \mathbb{Z}$ given as inputs in binary code. In order to obtain reasonable explicit bit complexity bounds, some assumption about the size of the b_j is required, including proper a priori bounds for the size of the output. Simple examples like

$$B(x) = 1 - 2x - 4x^2 - \cdots - 2^nx^n$$

and

$$\widehat{B}(x) = 1 - 2^nx$$

with $R_n(x) = 1 + 2x + 8x^2 + 32x^3 + \cdots + 2^{2n-1}x^n$ of total bit size $\sim n^2$ versus $\widehat{R}_n(x) = 1 + 2^nx + 2^{2n}x^2 + \cdots + 2^{n^2}x^n$ of size $\sim n^3/2$ illustrate that a measure like $\max_{j \leqslant n} |b_j|$ (e.g., used in [3]) is not so appropriate. As mentioned before, good root radius bounds are very convenient. Here we take a compromise and assume that $|b_j| \leqslant \beta^j$ for all j, where such β may often be quite large.

Lemma 4. *For real power series* $B(x) = 1 + \sum_{j>0} b_j x^j$ *with* $|b_j| \leqslant \beta^j$, *and* β *independent of* n, *the coefficients of* $1/B(x) = \sum_m r_m x^m$ *and the* c_m *of*

$B(x)B(-x)$ in (4) *satisfy* $|r_m| \leqslant \frac{1}{2}\cdot(2\beta)^m$ (*for* $m>0$), *and* $|c_m| \leqslant (3\beta^2)^m$.

Proof. Expanding

$$\left(1+\sum_{j>0} b_j x^j\right)^{-1} = 1-\left(\sum\cdots\right)+\left(\sum\cdots\right)^2-\left(\sum\cdots\right)^3+\cdots$$

as a huge sum, collecting terms with common factor x^m and estimating all b's by $|b_j| \leqslant \beta^j$ shows that the maximum values $|r_m| = 2^{m-i}\beta^m$ are attained with

$$(1-\beta x-\beta^2x^2-\beta^3x^3-\cdots)^{-1} = \tfrac{1}{2}+\tfrac{1}{2}/(1-2\beta x),$$

and (4) yields $|c_m| \leqslant (2m+1)\beta^{2m} \leqslant \delta^m$ with $\delta = 3\beta^2$. □

In the context of repeated root squaring, we need the latter estimate in iterated form also suited for bit length estimations. For that reason we introduce the length parameter $\tau = \log(3\beta)$ and rewrite the previous bounds as

$$|b_j| \leqslant 2^{\tau j}/3^j \quad \text{and} \quad |c_m| \leqslant 2^{2\tau m}/3^m, \tag{5}$$

so root squaring is simply covered by replacing τ with 2τ.

Let us now describe how to implement the polynomial multiplications of Algorithm 3 (in its asymptotical form, for large input) by means of long integer multiplications, more precisely by fast multiplications mod (2^N+1) with "suitable" numbers N. For details about our corresponding routine *SML* we refer to [8, pp. 32–34, 209]. For small n, some "school method" can be used, so let $n \geqslant n_0 \geqslant 8$, say, and $k = \lfloor n/2 \rfloor$. For the first multiplication $B_{2k}(x)\cdot B_{2k}(-x)$, we choose $l := \lceil \tau k \rceil$ and a suitable $N \geqslant 2l(2k+1)$, form the long numbers $B_{2k}(2^l)$ and $B_{2k}(-2^l)$ mod (2^N+1), compute their product

$$P \equiv G_{2k}(2^{2l}) \bmod (2^N+1),$$

and extract from this residue P the binary representations of the coefficients $c_1,\ldots,c_k$ of G_k. This last step is without problems, because (5) and our choice of l imply $|c_k| < \frac{1}{2}\cdot 2^{2l}$. Any overlapping of the potentially longer coefficients c_m for $m>k$ may be ignored, and even c_{2k} at the top, perhaps of length close to $4l$ and possibly causing (negative) wrap around, cannot spoil the result, since $c_0 = 1$ at the bottom end is known!

After recursive computation of

$$H_k(y) = 1/G_k(y) \bmod y^{k+1},$$

the second multiplication $B_n(-x)\cdot H_k(x^2)$ requires a greater *SML* length $N' \approx 2N$. Here we choose $l' := \lceil \tau n \rceil$ and a suitable $N' \geqslant l'(n+2k+1)$, compute the product

$$P' \equiv B_n(-2^{l'})\cdot H_k(2^{2l'}) \bmod (2^{N'}+1),$$

and extract from this P' the coefficients $r_1,\ldots,r_n$ of R_n each one contained in a bit string of length l'. The correctness of these choices relies on the bound $|r_n| \leqslant \frac{1}{2}\cdot 2^{l'}(2/3)^n$ resulting from (5) and the a priori bound in Lemma 4.

The time analysis of this asymptotical version of Algorithm 3 is very simple. We have $N \approx \tau n^2$, $N' \approx 2\tau n^2$, so we estimate the time for the top level of the recursion by adding *SML* time $t(N) \approx M(\tau n^2)$ and $t(N') \approx M(2\tau n^2)$, where

$$M(y) = \gamma_1 \cdot y \cdot \log y \cdot \log\log y \tag{6}$$

with some constant γ_1, plus linear overhead $O(\tau n^2)$ for encoding and decoding of those long numbers. One recursion level below, we replace n by $k \leqslant n/2$, and τ by 2τ, see (5), which leads to $2\tau k^2 \leqslant \tau n^2/2$. Thus we can estimate all levels of the recursion with a geometric series (ignoring irregularities at the low end), which means that we have to multiply the top level time bound by two. Altogether this amounts to an overall bit complexity time bound $\approx 6\cdot M(\tau n^2)$ for computing reciprocals mod x^{n+1} of monic integer power series with coefficients bounded as in (5).

Such asymptotical bounds are usually read as purely theoretical statements, but here we can refer to our very practical experience with *SML*. Counting *time units* (tu) as in [8], roughly equivalent to the cycle time (about 2–3 ns) of present computer hardware, the constant in (6) becomes a $\gamma_1 < 1.6$. For an example with $n = 1000$ and $\tau = 10$ (for $\beta = 341$) we thus obtain a bound of $6\cdot M(10^7) \approx 10^{10}$ time units, nowadays equivalent to about 20–30 seconds.

Acknowledgement

Thanks to an anonymous referee for valuable suggestions to improve the presentation of our results.

References

[1] U. Baum, M. Clausen, B. Tietz, Improved upper complexity bounds for the discrete Fourier transform, Appl. Algebra in Engrg., Commun. and Comput. 2 (1991) 35–43.

[2] D. Bini, Parallel solution of certain Toeplitz linear systems, SIAM J. Comput. 13 (1984) 268–276.

[3] D. Bini, V. Pan, Polynomial division and its computational complexity, J. Complexity 2 (1986) 179–203.

[4] P. Bürgisser, M. Clausen, M.A. Shokrollahi, Algebraic Complexity Theory, Springer, Berlin, 1997.

[5] K. Kalorkoti, Inverting polynomials and formal power series, SIAM J. Comput. 22 (1993) 552–559.

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

[7] A. Schönhage, Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients, in: J. Calmet (Ed.), Proc. EUROCAM '82 (Marseille 1982), European Computer Algebra Conference, Lecture Notes in Comput. Sci., Vol. 144, Springer, Berlin, 1982, pp. 3–15.

[8] A. Schönhage, A.F.W. Grotefeld, E. Vetter, Fast Algorithms—A Multitape Turing Machine Implementation, Mannheim, 1994.