Improved Parallel Polynomial Division and Its Extensions

Dario Bini* Victor Pan†

TR-92-051

September 1992

Abstract

We compute the first N coefficients of the reciprocal $r(x)$ of a given polynomial $p(x)$, $(r(x)p(x) = 1 \bmod x^N,\ p(0) \neq 0)$, by using, under the PRAM arithmetic models, $O(h \log N)$ time-steps and $O(\,(N/h)(1 + 2^{-h} \log^{(h)} N)\,)$ processors, for any h, $h = 1, 2, \ldots, \log^* N$, provided that $O(\log m)$ steps and m processors suffice to perform DFT on m points and that $\log^{(0)} N = N$, $\log^{(h)} N = \log_2 \log^{(h-1)} N$, $h = 1, \ldots, \log^* N$, $\log^* N = \max\{h : \log^{(h)} N > 0\}$. The same complexity estimates apply to some other computations, such as the division with a remainder of two polynomials of degrees $O(N)$ and the inversion of an $N \times N$ triangular Toeplitz matrix. This improves the known estimates of Reif-Tate and Georgiev. We also show how to extend our techniques to parallel implementation of other recursive processes, such as the evaluation modulo x^N of the m-th root, $p(x)^{1/m}$, of $p(x)$ (for any fixed natural m), for which we need $O(\log N \log\log N)$ time-steps and $O(N/\log\log N)$ processors. The paper demonstrates some new techniques of supereffective slowdown of parallel algebraic computations, which we combine with a technique of stream contraction.

*Department of Mathematics, University of Pisa. Dario Bini was supported by NSF Grants CCR 8805782 and CCR 9020690, and by MPI (40% funds).

†Lehman College, CUNY, and ICSI. Victor Pan was supported by NSF grants CCR 8805782, CCR 9020690 and by PSC CUNY Awards #661340, #668541, #669210 and #662478.

1 Introduction

In this paper we will improve the known upper estimates for the parallel arithmetic complexity of computing the reciprocal of a polynomial and, consequently, of the equivalent computations, such as polynomial and power series division and triangular Toeplitz matrix inversion (compare [2], [3] and the appendix). This improvement relies on the new techniques that, in particular, decrease by q times the processor bound for a large class of parallel algebraic computations in the result of their slowdown by $s = o(q)$ times. We call such a slowdown *supereffective* since a variant of Brent's scheduling principle only implies an effective decrease of the processor bound by $O(s)$ times in this case (see [6], [8]). We demonstrate these techniques for polynomial division (sections 2–5) and for computing (modulo x^N) the $(\frac{1}{m})$-th power (or the m-th root), $p(x)^{1/m}$, of a polynomial $p(x)$ (section 6), but they can be immediately extended to some other polynomial computations based on recursive processes, such as Newton's iteration. An outline of this general approach is given in the beginning of section 6.

We will state our estimates in the form $O_A(t, P)$, which means that our computations can be performed by using $O(st)$ time-steps and $O(P/s)$ processors (for any fixed s, $1 \le s \le P$), under the PRAM arithmetic models of computing (see [6]). We will assume the bound $O_A(\log m, m)$ on the cost of performing (forward and inverse) discrete Fourier transforms on the m-th roots of 1 [hereafter referred to as $DFT(m)$]. This bound holds, in particular, over the complex field of constants. Over any ring (with unity), the bound $O_A(\log m, m \log\log m)$ holds [4], so that all our estimates apply if their processor bounds are increased by the factor of $\log\log N$ provided that the ring supports Bluestein's generalized FFT ([7]). In section 5 we will comment on the latter assumption. Hereafter, log stands for $\log_2$ (all logarithms are binary).

Given a natural N and the coefficients of a polynomial $p(x)$, $p(0) \neq 0$, we will compute the first N coefficients of the reciprocal power series,

$$r(x) = \sum_{i=0}^{\infty} r_i x^i \ , \ r(x)\, p(x) = 1 \ , \ r_0 = \frac{1}{p(0)} \ , \tag{1.1}$$

yielding the complexity estimates

$$\begin{array}{l} c_N = O_A(h \log N, \frac{N}{h}(1 + 2^{-h} \log^{(h)} N)) \ , \\ \text{for} \quad h = 1, 2, \ldots, \log^* N \ . \end{array} \tag{1.2}$$

Here,

$$\log^{(0)} N = N \ , \ \log^{(h)} N = \log_2 \log^{(h-1)} N \ , \ h = 1, \ldots, \log^* N \ ,$$

$$\log^* N = \max\{h : \log^{(h)} N > 0\} \ .$$

In particular, for $h = \log^* N$, (1.2) turns into

$$c_N = O_A(\log^* N \log N \ , \ N/\log^* N) \ . \tag{1.3}$$

This improves the previous record bounds of [5],

$$c_N = O_A(\log N \ , \ N \log^2 N) \ ,$$

and of [11],

$$c_N = O_A(\log N \log\log N \ , \ N/\log\log N) \ .$$

We refer the reader to [2] and [3] on other previous works, in particular, on Reif's results of [10], including the bound $c_N = O_A(\log N, N^2)$, and Bini's results of [1], where $c_N = O_A(\log N, N)$ is proven for any precision approximation to the reciprocal of a polynomial modulo x^N.

2 Stream of Newton's Steps and Its Contraction

It is well known that, given a polynomial $p(x)$, $p(0) \neq 0$, the first m coefficients of the reciprocal power series $r(x)$ of (1.1) can be computed by performing $\lceil \log m \rceil$ steps of Newton's iteration,

$$\begin{array}{rl} z_0(x) & = 1/p(0) \ , \quad z_{i+1}(x) = z_i(x)\,(2 - p(x)z_i(x)\,) \ , \\ i & = 1, \ldots, \lceil \log m \rceil \ . \end{array} \tag{2.1}$$

In fact, since

$$r(x) - z_{i+1}(x) = (r(x) - z_i(x)\,)^2\, p(x) \ , \tag{2.2}$$

the number of coefficients shared by $r(x)$ and $z_i(x)$ is doubled in each iteration step i, so that, for all i,

$$z_i(x) = r(x) \bmod x^{2^i} \ . \tag{2.3}$$

The algorithm of Sieveking-Kung is based on this scheme, where, however, both polynomials $p(x)$ and $z_i(x)$ of (2.1) are reduced modulo x^{2^i}. In this case step i of Newton's process outputs a polynomial of degree at most $3(2^i - 1)$, whose coefficients can be computed at the cost $O_A(i, 2^i)$, by means of forward and inverse DFT's on $m(i)$ points, $m(i) > 3(2^i - 1)$. This implies the bound $O_A(\log^2 n, n/\log n)$ on the overall cost of computing the coefficients of $r(x) \bmod x^n$, which turns into the bound

$$c_{n,k} = O_A(\log n \log(\frac{n}{k}) \ , \ n/\log(\frac{n}{k})\,) \tag{2.4}$$

if we initially know the k coefficients of $r(x) \bmod x^k$.

In [5] it is pointed out that the relation (2.3) still holds even if $p(x)$ in (2.1) is replaced by $p(x) \bmod x^{2^{i+1}}$. Then the degrees of the polynomials $z_i(x)$ are bounded by $i2^i$, $i = 0, 1, \ldots$, and thus, the coefficients of $z_d(x)$, for $d = \lceil \log n \rceil$, can be computed by means of a single inverse DFT on $O(n \log n)$ points (roots of 1) once the values of $z_d(x)$ on these points have been computed. To evaluate $z_d(x)$ on a set of points, we compute the values of $p_i(x) = p(x) \bmod x^{2^{i+1}}$ on this set and then, for each point, recursively apply the equation (2.1) for $i = 0, 1, \ldots, d-1$, thus avoiding the application of DFT until we compute the values of $z_d(x)$. This way, we *contract the stream* of recursive steps, by cutting-off their slowest parts (that is, DFT's) and thus accelerating the transition to each new step (compare another example of stream contraction in [9]). As a result, the n coefficients of $r(x) \bmod x^n$ are computed at the cost bounded by $O_A(\log n, n \log^2 n)$, ([5]).

3 A Basis Algorithm for New Improvements and Its Recursive Restarting

In this and in the next two sections, we will combine the two ways of applying Newton's iteration (2.1), that is, the coefficientwise way (Sieveking-Kung) and the pointwise way ([5]), in order to decrease the overall parallel computational cost. Moreover, we will *recursively stop and restart* the pointwise iteration process so as to bound the number of processors involved. It turns out that relatively few restarts suffice to compute a considerable part of all the output coefficients. Indeed, we will consider Newton's iteration that starts with

$$y_0(x) = r(x) \bmod x^k, \tag{3.1}$$

for any fixed natural k (not necessarily for $k = 1$), and that continues as follows:

$$p_i(x) = p(x) \bmod x^{k2^{i+1}} , \tag{3.2}$$

$$y_{i+1}(x) = y_i(x)\,(2 - p_i(x)\,y_i(x)\,) , \tag{3.3}$$

$i = 0, 1, \ldots$. Then (2.2) immediately implies that

$$y_i(x) = r(x) \bmod x^{k2^i} , \tag{3.4}$$

and we verify by induction on i that

$$\deg\, y_i(x) \le (i+1)k\,2^i - 2^{i+1} + 1 . \tag{3.5}$$

(3.4) and (3.5) immediately imply the correctness of our basis algorithm below.

With no loss of generality, we assume that n is large enough, say, $n \ge 16$, so that $\log^* n \ge 3$. Furthermore, for convenience, let s denote an integer, $4 \le s \le n$, and let $\log\log s$ and $\log n$ be integers.

Basis algorithm.

Input: integers n and s and the coefficients of $p(x)$ and $r(x) \bmod x^{n/s}$.

Output: the coefficients of $r(x) \bmod x^{n/\log s}$.

Initialize: set $k = n/s$, $j = \log s - 2\log\log s$, $q = n/\log s$, $y_0(x) = r(x) \bmod x^k$.

Computation:

1. $j + 1$ times apply DFT on q points, to evaluate the polynomials $y_0(x)$ of (3.1) and $p_i(x)$ of (3.2), for $i = 0, \ldots, j-1$, on the q-th roots of 1.

2. Use the equations (3.3), for $i = 0, \ldots, j-1$, in order to evaluate $y_{i+1}(x)$ on the q-th roots of 1.

3. Apply the inverse DFT on q points in order to evaluate the coefficients of $y_j(x)$. [$\deg\, y_j(x) \le (j+1)k\,2^j - 2^{j+1} + 1$; $y_j(x) = r(x) \bmod x^{k2^j}$, due to (3.4), and $k2^j = n/\log^2 s$.]

4. Update the initialized parameters by first setting $y_0(x) = y_j(x) \bmod x^{k2^j}$, $k_1 = k2^j$, $q = n(1 + \log\log s)/\log s$, and then $k = k_1 \ge n/\log^2 s$, $j = \log\log s$; then redefine $p_{i-1}(x)$ and $y_i(x)$, for $i = 1, \ldots, j$, by using the equations (3.2) and (3.3); then again apply steps 1-3 to compute and to output the coefficients of the polynomial $y_j(x)$ (which equals $r(x) \bmod x^{k2^j}$, where $k2^j = n/\log s$ for the updated k and j).

The cost of the computation by the basis algorithm is clearly bounded by $O_A(\log n, n)$, and we will recursively apply this algorithm for a fixed n and for s successively taking the values

$$\begin{aligned} &s_0 = n\ ,\ s_1 = \log n\ ,\ s_2 = \log\log n\ ,\ \ldots, \\ &s_{h-2} = \log^{(h-2)} n\ , \end{aligned} \tag{3.6}$$

for $2 \le h \le \log^* n$ (for simplicity, let $\log^{(g)} n$ be integers for $g = 1, \ldots, h$). This way, we will compute the first $m(h,n) = n/\log^{(h-1)} n$ coefficients of $r(x)$ at the cost bounded by $O_A(h \log n, n)$. For $h \le \log^* n - 2$, we will extend this process to $s_{h-1} = \log^{(h-1)} n$.

Note that already the basis algorithm uses two successive recursive loops, and we are now taking advantage of recursive restarting of this algorithm. Indeed, if we applied the basis algorithm only once, by setting $s = n$, $N = n/\log s$, then the resulting cost bound $O_A(\log n, n) = O_A(\log N, N \log N)$ would have assumed too many processors.

4 Final Refinement of the Algorithm

Let $N = n2^h$. In particular, this choice of N ensures that the above cost bound $O_A(h \log n, n)$ is not greater than the desired bound (1.2). We still need, however, to compute the coefficients of $r(x) \bmod x^N$ that still remain unknown after the recursive application of the basis algorithm. We will first extend the computation to the evaluation of $r(x) \bmod x^n$ [note that $m(h,n) < n$]. Set

$$\begin{aligned} y_0(x) &= r(x) \bmod x^{m(h,n)}\ , \\ k &= m(h,n)\ ,\ j = \log^{(h-1)} n\ , \end{aligned} \tag{4.1}$$

$$q = (j+1)k2^j - 2^{j+1} + 1\ , \tag{4.2}$$

[so that $k2^j = n$, $q \le (1 + \log^{(h)} n)n$], and again apply steps 1-3 of the basis algorithm. The complexity of these steps is bounded from above by $O_A(\log n, n(\log^{(h)} n)^2)$, which we may replace by the weaker bound $O_A(h \log n, \frac{n}{h}(\log^{(h)} n)^2)$. The overall cost of computing the coefficients of $r(x) \bmod x^n$ is thus bounded by $O_A(h \log n\ ,\ n(1 + \frac{(\log^{(h)} n)^2}{h}))$. For $N = n2^h$ and for $h = \log^* n - O(1)$, in particular, for $h > \log^* n - 2$, this turns into the bound

$$c^*_{N,h} = O_A(h \log N\ ,\ N\, 2^{-h}(1 + \frac{\log^{(h)} N}{h}))\ , \tag{4.3}$$

not exceeding the desired bound (1.2).

If $h \le \log^* n - 2$, we proceed similarly, but extend the computation of $r(x) \bmod x^{n/s_i}$ for $s_0, s_1, \ldots$ of (3.6) to $s_{h-1} = \log^{(h-1)} n$. This way we will compute all the coefficients of $r(x) \bmod x^n$, at the cost bounded by $O_A(h \log n, n(1 + (\log^{(h+1)} n)^2)/h)$. We replace this bound by the weaker bound $O_A(h \log n, n(1 + \frac{\log^{(h)} n}{h}))$ on the overall cost of computing all the coefficients of $r(x) \bmod x^n$. Now substitute $N = n2^h$ and arrive at (4.3) also for $h \le \log^* n - 2$.

Finally, we will compute the remaining coefficients of $r(x) \bmod x^N$, by applying h steps of the Sieveking-Kung algorithm; the cost of this stage is bounded by $O_A(h \log N, N/h)$ [see

(2.4)]. This cost bound implies the bound (1.2), for $h \le \log^* n$. Surely, $\log^* N \le 1 + \log^* n$ as $N \to \infty$, so that we extend (1.2) to all $h \le \log^* N$ and then also deduce (1.3).

To relax the assumption about the integrality of $\log\log s$ and $\log^{(g)} n$ for $g = 1, \dots, h$, we just need to replace the values of k in the basis algorithm (chosen at the initialization step and at step 4) and in (4.1), as well as the values of n/s_i [for $i = 1, \dots, h$ and for $s_1, s_2, \dots, s_h$ of (3.6)] and of $m(h, n)$, by their ceilings, $\lceil\ \rceil$, that is, by the minimum integers not exceeded by these values; to define q by (4.2) and to let j denote the minimum integer such that $k2^j$ is not less than the number of the coefficients or $r(x)$ that should be computed in the current stage, that is, $\lceil n/\log^2 n \rceil$ in step 3 of the basis algorithm, $\lceil n/\log s \rceil$ in its step 4, and n in the final refinement stage. It is easy to verify that such a replacement preserves (4.3).

5 Application over Abstract Rings

Our algorithms can be applied over any ring $\mathbf{R}$ of constants that contains unity and an element g such that $g^{-1} \in \mathbf{R}$ and $1, g, g^2, \dots, g^{N-1}$ are distinct in $\mathbf{R}$. In this case we may replace FFT by its Bluestein's extension (where g replaces the principal root of 1), [7]. Such an extended FFT is performed at the cost $O_A(\log N, N \log\log N)$ over any ring $\mathbf{R}$ ([4]).

Furthermore, there exists a desired element g in $\mathbf{R}$ if the number $|\mathbf{R}|$ of distinct elements that belong to $\mathbf{R}$ together with their reciprocals exceeds $N-1$. Otherwise, there must exist such an element g in the extension $\mathbf{E}$ formed by all polynomials over $\mathbf{R}$ modulo an irreducible polynomial of degree d if $d \ge \log N/\log|\mathbf{R}|$. In the latter case, each univariate polynomial of degree at most K in $\mathbf{E}$ turns into a bivariate polynomial in $\mathbf{R}$ of degrees at most K and $d-1$ in its two variables, respectively. Multiplication of two such polynomials has computational cost $O_A(\log(Kd), Kd\log\log(Kd))$, ([4]). This implies that the transition from application of our algorithms in $\mathbf{E}$ (where they are supported by Bluestein's extension of FFT) to their application in the original ring $\mathbf{R}$ will only require increasing the resulting processor bound by the factor of d, and we may assume that $d = O(\log N/\log|\mathbf{R}|)$. As an alternative, we may compute $p^{-1}(x) \bmod x^{\lceil N/d \rceil}$ by using our algorithms and then make transition to $p^{-1}(x) \bmod x^N$ by means of the Sieveking-Kung algorithm, at the overall cost $O_A(\log N \log\log N, N)$ over any ring of constants (compare [11]).

The same comments apply to our algorithm for the evaluation modulo x^N of $p^{1/2}(x)$, to be presented next.

6 Extensions

We will demonstrate two ways of extension of the results of this paper.

1) At first we will consider application of the techniques of the previous sections to the recursive computations of the form

$$v_{i+1}(x) = h(v_i(x), x) \bmod x^N \tag{6.1}$$

where $h(v, x)$ is a polynomial in v and x; $\deg_v h$ is small, $v(x) \bmod x^N$ is the output polynomial, and $v_i(x) = v(x) \bmod x^{m(i)}$, for a rapidly growing sequence $m(i), i = 0, 1, \dots$. Assume that such a recursive computation is represented by a parallel algorithm which is

processor efficient but relatively slow (as in the case of the Sieveking–Kung algorithm). We first contract the stream of its recursive steps (as in section 2), to arrive at a faster (although processor inefficient) algorithm. We may, however, proceed as in section 3 and apply this algorithm in a limited way, computing only a small fraction of the desired output coefficients of $v(x)$, to keep the number of processors within the desired bounds. Following the patterns of section 3, we will then apply this algorithm again, to compute more coefficients of $v(x)$, and will recursively repeat this process until we compute a considerable fraction of all the output coefficients. Then we will shift to the original algorithm, which will easily complete the evaluation of the remain

The next example once again demonstrates the efficacy of such an approach for parallelization of a large class of recursive computations. Indeed, suppose that we need to compute modulo x^N the square root, $v(x) = p(x)^{1/2} \bmod x^N$, of a polynomial $p(x)$, where $p(0) = 1$. Since $f(v,x) = v^{-2}(x)\,p(x) - 1 = 0$, we apply Newton's iteration with $v_0(x) = 1$ and recursively compute

$$\begin{aligned} v_{i+1}(x) &= v_i(x) - f(v_i,x) \,/\, f'_v(v_i,x) \\ &= 0.5\, v_i(x)(3 - p^{-1}(x)v_i^2(x)\,)\ , \\ & \qquad i = 0,1,\ldots \end{aligned} \tag{6.2}$$

We observe that

$$\begin{aligned} & v_{i+1}(x) - p^{1/2}(x) \\ &= 0.5\, v_i(x)\,(3 - p^{-1}(x)v_i^2(x)\,) - p^{1/2}(x) \\ &= v_i(x) - p^{1/2}(x) + 0.5 v_i(x) p^{-1}(x)(p(x) - v_i^2(x)) \\ &= (v_i(x) - p^{1/2}(x))\,(1 - 0.5 v_i(x)\,p^{-1}(x) \\ & \ (p^{1/2}(x) + v_i(x))\,) \\ &= (v_i(x) - p^{1/2}(x))\,[(1 - v_i(x)\,p^{-1/2}(x)\,) \\ & \ -0.5 v_i(x)\,p^{-1}(x)(v_i(x) - p^{1/2}(x))] \\ &= -(v_i(x) - p^{1/2}(x))^2\,(p^{-1/2}(x) + 0.5 v_i(x)\,p^{-1}(x)) \\ &= O(v_i(x) - p^{1/2}(x))^2\ , \end{aligned}$$

and therefore,

$$v_j(x) - p^{1/2}(x) = 0 \bmod x^J\ , \quad J = 2^j\ , \quad j = 0,1,\ldots$$

As in the Sieveking-Kung algorithm, substitute $v_j(x) \bmod x^J$, $J = 2^j$, for $v_j(x)$, $j = i, i+1$, in (6.2), substitute $p^{-1}(x) \bmod x^{2^i}$ for $p^{-1}(x)$ in (6.2) and arrive at the complexity bound $O_A(\log^2 N\ ,\ N/\log N)$ for this computational problem.

Similarly to the algorithm of [5], contract the stream of Newton's steps by replacing $p^{-1}(x)$ by $p_i^{-1}(x) = p^{-1}(x) \bmod x^{2^{i+1}}$ in (6.2). Observe that in this case the degrees of the polynomials $v_i(x)$ are bounded by $\sum_{g=1}^{i}(2/3)^g 3^i < 3^{i+1}$ and deduce the bound $O_A(\log k\ ,\ k^d \log k)$, $d = \log 3 = 1.5849\ldots$, $1/d = 0.6309\ldots$, on the complexity of the evaluation of $p^{1/2}(x) \bmod x^k$.

We apply the latter result for $k = n^{0.63}$, where we will specify $n \le N$ later on [see (6.6)]. Then $k^d \log k = o(n^{0.999})$, and we obtain the k first coefficients of $p^{1/2}(x)$ at the cost bounded by $O_A(\log n, n^{0.999})$.

Next, extend the basis algorithm of section 3, by using the equations

$$p_i^{-1}(x) = p^{-1}(x) \bmod x^{k\,2^{i+1}}\ ,$$

$$\begin{aligned} y_0(x) &= v(x) \bmod x^k \ , \\ y_{i+1}(x) &= 0.5\, y_i(x)(3 - p_i^{-1}(x)\, y_i^2(x)) \ , \\ & \quad i = 0, 1, \ldots, j \ , \end{aligned}$$

instead of (3.1)–(3.3). Observe that the degrees of the polynomials $y_i(x)$ are bounded by $k3^i \sum_{g=0}^{i} (2/3)^g < k3^{i+1}$, so that the transition from $y_0(x) = p^{1/2}(x) \bmod x^k$ to $y_i(x) = p^{1/2}(x) \bmod x^{k2^i}$ costs

$$\begin{aligned} & O_A(\log(k\,2^i), k2^{id}(i + \log k)\,) \\ & = O_A(\log n,\ k(m/k)^d \log m) \ , \end{aligned} \tag{6.3}$$

for $d = \log 3$, $m = m(i,k) = k2^i \le n$.

Let us now define recursive restarting of the algorithm. First choose an integer g and an increasing sequence of integers $k_0 = 1$, $k_1 = \lceil n^{0.63} \rceil$, $k_2, \ldots, k_g$, $k_j = \lceil n^{1-0.37^j} \rceil$, $j = 0, 1, \ldots, g$, and then recursively apply the above transition from $y_0(x)$ to $y_i(x)$, for $k = k_j \ge n^{1-0.37^j}$, $m = k_{j+1}$, $j = 0, 1, \ldots, g-1$. Denote $b_j = \log k_j / \log n \ge 1 - 0.37^j$, so that $1 - b_g \ge 1/\log n$ for

$$g = \lceil \log\log n / \log(1/0.37) \rceil \ . \tag{6.4}$$

In g recursive transitions we will arrive at $p^{1/2}(x) \bmod x^{k_g}$ where

$$k_g \ge n^{1-1/\log n} = n/2 \ .$$

Due to (6.3) and (6.4), the overall cost of this computation is bounded by

$$O_A(\log n \log\log n, n \log n) \ . \tag{6.5}$$

Now, as the final refinement, let

$$n = \lceil N/(\log N \log\log N) \rceil \ , \tag{6.6}$$

apply $1 + \lceil \log(\log n \log\log n) \rceil$ Newton's steps of the Sieveking-Kung type and arrive at $p^{1/2}(x) \bmod x^N$ at the additional cost

$$O_A(\log N \log\log N \ , N/\log\log N) \tag{6.7}$$

where we may assume that $p^{1/2}(x) \bmod x^{k_g}$ is known for $k_g \ge n/2$. (6.7) also bounds the overall cost of computing $p^{1/2}(x) \bmod x^N$ since (6.5) turns into (6.7) for n of (6.6).

The complexity estimates (6.7) for computing the square root of a polynomial modulo x^N can be immediately extended to computing its m-th root modulo N for any fixed positive integer m. Moreover, the techniques applied above enable us to extend (6.7) as follows:

Theorem 6.1. *Suppose that for $c > 1$ and for a pair of integers k and m, $1 \le k < m$, we are given the first k components of a vector and we need to compute the next $m - k$ components. Suppose that two alternate "black box" parallel algorithms, 6.1 and 6.2, are available, which solve this problem at the cost bounded by $O_A(\log m \log(m/k),\ m/\log(m/k))$, and $O_A(\log m,\ k(m/k)^c)$, respectively. Then recursive application of Algorithm 6.2, for an appropriate recursive choice of the pairs $k = k_i, m = m_i$, for $i = 0, 1, \ldots$, such that $k_{i+1} = m_i$, for all i, followed by a single application of Algorithm 6.1, supports the solution of the problem of size N, for $k = 1$ and $m = N$, at the cost bounded by (6.7).*

For a more narrow class of computational problems, exemplified by polynomial division, we may improve (6.7) to (1.2). Somewhat similar techniques yield supereffective slowdown of some fundamental computations in linear algebra and in path algebras, where we recursively apply fast but processor inefficient algorithms to decrease the size of the problem (see [8]).

2) The complexity estimates (1.2), (1.3) and (6.7) can be extended to the computational problems readily reducible to computing the reciprocal of a polynomial $p(x)$ and its m-th root, respectively. For demonstration, let us recall the well-known reduction of polynomial division with a remainder to computing the reciprocal of a polynomial (and to polynomial multiplication).

Indeed, given the coefficients of two polynomials $s(x) = \sum_{i=0}^{m} s_i x^i$ and $t(x) = \sum_{i=0}^{n} t_i x^i$, $t_n \neq 0$,we seek the quotient $q(x) = \sum_{i=0}^{m-n} q_i x^i$ and the remainder $r(x) = \sum_{i=0}^{n-1} r_i x^i$ such that

$$s(x) = t(x)\, q(x) + r(x) \ . \tag{6.8}$$

Having computed $q(x)$ we immediately obtain $r(x)$ from (6.8). To compute $q(x)$, we define

$$\begin{aligned} S(z) &= z^m s(1/z) = \textstyle\sum_{i=0}^{m} s_{m-i} z^i \ , \\ T(z) &= z^n t(1/z) = \textstyle\sum_{i=0}^{n} t_{n-i} z^i \ , \\ Q(z) &= z^{m-n}\, q(1/z) = \textstyle\sum_{i=0}^{m-n} q_{m-n-i} z^i \ , \end{aligned}$$

observe that

$$S(z) = T(z)\, Q(z) \bmod z^{m-n+1} \ , \tag{6.9}$$

and apply the algorithms of this paper to compute the reciprocal,

$$V(z) = \sum_{i=0}^{K-1} v_i z^i = T^{-1}(z) \bmod z^K \ , \quad K = m - n + 1 \ .$$

It immediately follows from (6.9) that $Q(z) = V(z)\, S(z) \bmod z^K$, so that the coefficients of $q(x)$ are the leading coefficients of the polynomial product $V(z) \sum_{i=n}^{m} s_i z^{i-n}$.

References

[1] D. Bini, "Parallel Solution of Certain Toeplitz Linear Systems," *SIAM J. on Computing,* 13(2), (1984) 268–276.

[2] D. Bini and V. Pan, "Polynomial Division and its Computational Complexity," *Jour. Complexity*, 2 (1986) 179–203.

[3] D. Bini and V. Pan, *Numerical and Algebraic Computations with Matrices and Polynomials,* Vols. 1 and 2, Birkhauser, Boston (1992, to appear).

[4] G. Cantor and Kaltofen, E., "On Fast Multiplication of Polynomials over Arbitrary Algebras," *Acta Inf.*, 28 (1991) 693–701.

[5] R. E. Georgiev, "Inversion of Triangular Toeplitz Matrices by Using the Fast Fourier Transform," *J. New Gener. Comput. Sys.* 2(3) (1989) 247–256.

[6] R. M. Karp and V. Ramachandran, "A Survey of Parallel Algorithms for Shared Memory Machines," in *Handbook of Theor. Comp. Science*, North-Holland (1990) 869–941.

[7] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, v. 2, Addison-Wesley, Mass., 1981.

[8] V. Y. Pan and F. P. Preparata, "Supereffective Slowdown of Parallel computations," *Proc. 4th Annual ACM Symp. on Parallel Algrithms and Architectures* (1992), 402–409.

[9] V. Y. Pan and J. H. Reif, "The Parallel Computation of the Minimum Cost Paths in Graph by Stream Contraction," *Information Processing Letters*, 40 (1991) 79–83.

[10] J. H. Reif, "Logarithmic Depth Circuits for the Algebraic Problems," *SIAM J. on Computing,* 15 (1986) 231–242.

[11] J. H. Reif and S. H. Tate, "Optimal Size Division Circuits," *Proc. 21-th Ann. ACM Symp. on Theory of Computing*, (1989) 264–270 and *SIAM J. on Computing*, 19(5) (1990) 264–270.