References

[BP] Bini, D., and V. Pan, Polynomial division and its computational complexity, *Jour. Complexity*, 2 (1986) 179-203.

[BPa] Bini, D., and V. Pan, On polynomial and matrix computation, to appear.

[G] Georgiev, R. E., Inversion of triangular Toeplitz matrices by using the fast Fourier transform, *J. New Gener. Comput. Sys.* 2(3) (1989) 247-256.

[KR] Karp, R. M. and V. Ramachandran, A survey of parallel algorithms for shared memory machines, Report No. UCB/CSD 88/408, Comp. Sci. Div. (EECS), U. California, Berkeley, CA (1988), to appear in *Handbook of Theor. Comp. Science*, North-Holland, 1990.

[RT] Reif, J. H., and S. H. Tate, Optimal size division circuits, *Proc. 21-th Ann. ACM Symp. on Theory of Computing* (1989) 264-270.

$k(g)$, respectively, $q(g) = (j(g)+1)\,k(g)\,2^{(j(g)} - 2^{j(g)+1} + 2$, $j(g) = \lceil \log \dfrac{k(g)}{k(g-1)} \rceil$, for all g; $k(G) = n$, and $k(g) = n/\log^{(2,g)} n$, for $j = 0, 1, \ldots, G-1$.

set

$$q = (j+1)k2^j - 2^{j+1} + 1 \tag{3.8}$$

and apply steps 1-3 of the computation of the basis algorithm. The complexity of these steps is upper bounded by $O_A(\log n, n(\log^{(h+1)} n)^2)$, which we may replace by the weaker bound $O_A(h \log n, \frac{n}{h} \log^{(h)} n)$. The overall cost of computing the coefficients of $r(x) \bmod x^n$ is thus bounded by $O_A(h \log n \ , \ n(1 + \frac{\log^{(h)} n}{h}))$, which turns into the bound

$$c^*_{N,h} = O_A(h \log N \ , \ N\, 2^{-h}(1 + \frac{\log^{(h)} N}{h})) \tag{3.9}$$

if we denote $N = n\, 2^h$.

Finally, we apply h steps of the Sieveking-Kung algorithm to compute the coefficients of $r(x) \bmod x^N$, that have not yet been computed; the cost of this step, $O_A(h \log N \ , \ N/h)$ [see (2.4)], is smaller than the bound (3.9), and thus we arrive at (1.2), for $h \leq \log^* n$, and surely, $\log^* N = O(\log^* n)$ as $N \to \infty$, so that we extend (1.2) to all $h \leq \log^* N$ and then deduce (1.3).

To relax the assumption about the integrality of $\log \log s$ and $\log^{(g)} n$ for $g = 1, \ldots, h$, we just need to replace the values of k and j in the basis algorithm (chosen at the initialization step and at step 4) and in (3.7), as well as the values of $s_1, s_2, \ldots, s_h$ in (3.6) and $m(h,n)$, by the minimum integers exceeding these values and to define q by (3.8). It is easy to verify that such a replacement preserves (3.9).

Let us finally show a simple extension of our computation that also leads to (3.9) [and therefore, to (1.2) and (1.3)]. Denote

$$\log^{(2,0)} n = n \ , \log^{(2,a)} n = \log \left(\log^{(2,a-1)} n \right)^2 \ , \quad s = 1, 2, \ldots, \log^{(2,*)} n \ ,$$

$$\log^{(2,*)} n = \min\{a : \log^{(2,a)} n \leq 1\} \ .$$

Recursively apply steps 1-3 of the basis algorithm, for $g = 1, \ldots, G$, $G \leq \log^{(2,*)} n$, $G = O(h)$, where in the g-th call to these steps, q, j and k are replaced by $q(g)$, $j(g)$, and

Now, let s denote an integer, $4 \le s \le n$; 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 times apply DFT at q points, to evaluate the polynomials $y_0(x)$ of (3.1) and $p_i(x)$ of (3.2), for $i = 0, \ldots, j-1$, at the q-th roots of 1.

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

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

4. Update the initialized parameters by first setting $y_0(x) = y_j(x) \bmod x^{k2^j}$, $q = n\log\log s/\log s$, and then $k = n/\log^2 s$, $j = \log\log s$; 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

$$s_0 = n \ , \ s_1 = \log n \ , \ s_2 = \log\log n \ , \ \ldots \ s_{h-1} = \log^{(h-1)} n \ , \tag{3.6}$$

for $1 \le h \le \log^* n$ (for simplicity, let $\log^{(g)} n$ be integers for $g = 1, \ldots, h$). This way, we will compute the first $m(h,n) = n/\log^{(h)} n$ coefficients of $r(x)$ for the cost $O_A(h\log n, n)$. Note that $m(h,n) = n$ if $h = \log^* n$; otherwise $m(h,n) < n$, and then, to compute the remaining coefficients of $r(x) \bmod x^n$, we set

$$y_0(x) = r(x) \bmod x^{m(h,n)} \ , \quad k = m(h,n) \ , \ j = \lceil \log^{(h+1)} n \rceil \ , \tag{3.7}$$

In [G] it is pointed out that the relation (2.3) still holds even if $p(x)$ in (2.1) is replaced by $p(x) \bmod x^{2^i}$. 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 the single inverse DFT at $O(n \log n)$ points (roots of 1) once the values of $z_d(x)$ at these points have been computed. To compute the values of $z_d(x)$ at a set of points, we may compute the values of $p_i(x) = p(x) \bmod x^{2^i}$ at 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. This way, the n coefficients of $r(x)$ and x^n can be computed for the cost $O_A(\log n, n \log^2 n)$ ([G]).

3. Improved algorithms. In this section, we will combine the two ways of applying Newton's iteration (2.1), that is, the coefficientwise way (Sieveking-Kung) and the pointwise way ([G]), in order to decrease the overall parallel computational cost. In particular, 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:

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

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

$i = 0, 1, \ldots$. 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) will immediately imply the correctness of our basis algorithm below.

This improves the previous record bounds of [G],

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

and of [RT],

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

(See [BP] and [BPa] for other previous works.)

2. Some known algorithms. 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,

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

In fact, since

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

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}; then step i of Newton's process outputs a polynomial of degree at most $3(2^i - 1)$, whose coefficients can be computed by means of forward and inverse DFTs at $m(i)$ points, $m(i) > 3(2^i - 1)$, for the cost $O_A(i, 2^i)$. This implies the bound $O_A(\log^2 n, n/\log n)$ on the overall cost of computing the coefficients of $r(x) \bmod x^n$, whicn 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 $r(x) \bmod x^k$.

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 [BP]). The techniques and the results can be extended to other polynomial computations based on recursive processes, such as Newton's iteration, in particular, to computing (modulo x^N) the $(\frac{1}{m})$-th power (or the m-th root) $p(x)^{1/m}$ of a polynomial $p(x)$.

We will state our estimates in the form $O_A(t,P)$, which means that our computations can be performed by using $O(st)$ arithmetic steps and $O(P/s)$ processors (for any fixed s, $1 \le s \le P$), under the PRAM arithmetic models of computing (see [KR]). We will assume the bounds $O_A(\log m, m)$ on the cost of performing (forward and inverse) discrete Fourier transforms at the m-th roots of 1 [hereafter referred to as $DFT(m)$]. 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,

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

yielding the complexity estimates

$$c_N = O_A(h \log N, \frac{N}{h}(1 + 2^{-h} \log^{(h)} N)) \ , \quad \text{for all } h \ , \quad h = 1, 2, \dots, \log^* N \ . \tag{1.2}$$

Here,

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

$$\log^* N = \min\{h : \log^{(h)} N \le 1\} \ .$$

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

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

Improved parallel polynomial division

Dario Bini*
Department of Mathematics
University of Pisa

Victor Pan**
Dept. of Comp. Sci.
Columbia University;
Department of Math. & Comp. Sci.
Lehman College, CUNY

CUS-029-90

Summary. We compute the first N coefficients of the reciprocal $r(x)$, $r(x)p(x) = 1 \bmod x^N$ (given a natural N and a polynomial $p(x)$), $(p(0) \neq 0)$, by using $O(h \log N)$ arithmetic steps and $O\big((N/h)(1 + 2^{-h} \log^{(h)} N)\big)$ processors, for any h, $h = 1, 2, \ldots, \log^* N$, under the PRAM arithmetic models, provided that $O(\log N)$ steps and N processors suffice to perform DFT at N points and that $\log^{(0)} N = N$, $\log^{(h)} N = \log_2 \log^{(h-1)} N$, $h = 1, \ldots, \log^* N$, $\log^* N = \min\{h : \log^{(h)} N \leq 1\}$. The same estimates apply to several other computations such as the division with a remainder of two polynomials of degrees $O(N)$, the inversion of an $N \times N$ triangular Toeplitz matrix, and the evaluation modulo x^N of the m-root $p(x)^{1/m}$ of $p(x)$ for any fixed natural m. This improves the known estimates of Reif-Tate and Georgiev.

Keywords: polynomial division, parallel algorithms, computational complexity, triangular Toeplitz matrices.

---

* Supported by NSF Grant CCR 8805782 and by MPI (40% funds).

** Supported by NSF grant CCR 8805782 and by PSC CUNY Awards #661340, #668541 and #669210.