The Power of Combining the Techniques of Algebraic and Numerical Computing: Improved Approximate Multipoint Polynomial Evaluation and Improved Multipole Algorithms

V. Y. Pan

Dept. of Mathematics & Computer Sci.
Lehman College, CUNY
Bronx, NY 10468

J. H. Reif and S. R. Tate

Computer Science Dept.
Duke University
Durham, NC 27706

Abstract

We demonstrate the power of combining the techniques of algebraic computation with ones of numerical computation. We do this by improving the known methods for polynomial evaluation on a set of real points and for simulation of n charged particles on the plane (the latter problem is equivalent to Trummer's problem). In both cases we approximate (rather than exactly compute) the solutions and do this by exploiting algebraic techniques of the algorithm design.

The previous best algorithm for approximate evaluation of a polynomial on a real set was due to Rokhlin and required the order of $mu + nu^3$ *infinite precision arithmetic operations to approximate [on a fixed bounded set* $X(m)$ *of* $m + 1$ *real points] a degree n polynomial* $p(x) = \sum_{i=0}^{n} p_i x^i$ *within the error bound* $2^{-u} \sum_{i=0}^{n} |p_i|$. *We develop an approximation algorithm, which decreases Rokhlin's record estimate to* $O(m \log^2 u + n \min \{u, \log n\})$. *This result may also be favorably compared with the record bound* $O((m + n) \log^2 n)$ *on the complexity of the exact multipoint polynomial evaluation. Furthermore, if some values independent of the input coefficients and of the set* $X(m)$ *and depending only on n and u can be precomputed cost-free, then we decrease the computational cost estimate to* $O(m \log^2 u + u \log u)$. *All our algorithms allow NC and processor efficient parallel implementation. Because of the fundamental nature of the multipoint polynomial evaluation, our results have further applications to numerical and algebraic computational problems.*

For Trummer's problem, our algorithm approximates the output values to p bits each, by using $O(n \log^2 p)$ *time, thus improving the previous record time bound of* $O(np \log p)$.

1 Introduction

The techniques for algebraic computing and for numerical computing have been historically developing quite independently of each other, although the power of combining their advantages has been increasingly appreciated in recent years and has been advocated, for instance, in [31] and [3]. The present paper gives some new evidence of the advantages of such a combination: we improve the known results on the multipoint polynomial evaluation (on a real set) and on the solution of Trummer's problem, which is equivalent to the two-dimensional n-body problem of the multipole simulation of n charged particles in the electrostatic or gravitational potential fields on the plane (such a simulation is extensively used in molecular chemistry, chemistry, biochemistry, astronomy and, to a little lesser extent, in physics). In both cases we compute approximations, rather than the exact solutions (which is a typical feature of numerical computing), but we obtain such approximations by applying (some new and some old) purely algebraic techniques.

There is no apparent correlation between our two topics, and our section 9, devoted to an algorithm of [35] for Trummer's problem, can be read independently of sections 2-8, devoted to the algorithms of [32] for polynomial approximation. The technical overlaps of the solutions of both these computational problems are thus quite symptomatic.

Let us now turn to the details of our first subject, of approximate multipoint polynomial evaluation, postponing our further comments on Trummer's problem to the end of this section. The literature on efficient evaluation of a polynomial dates back to a linear time algorithm, using n additions and n multiplications, for

0-8186-2900-2/92 $3.00 © 1992 IEEE

the exact evaluation of a polynomial at a single point, usually called Horner's rule, due to the work of 1819 by Horner [17], but actually discussed by Isaac Newton [24], already in 1669. For the evaluation at a single point, this algorithm has been proved to be optimum ([26]).

We are concerned with the fundamental problem of evaluation of a polynomial of degree n, simultaneously at m points, in less than nm time (measured by the number of arithmetic operations involved). In 1971 Borodin and Munro [6] published an $O(n^{1.91})$ time algorithm for multipoint polynomial evaluation, for $m = n$, thus giving the first algorithm with the complexity below nm. The asymptotically fastest algorithm known for the multipoint exact polynomial evaluation problem is due to Moenck and Borodin [22], who reduced the problem to repeated polynomial divisions. Including here the Sieveking-Kung fast algorithm for polynomial division made this an $O((m+n)\log^2 n)$ algorithm. Strassen [39] proved the lower bound $\lceil m \log n \rceil$.

Our goal is to present efficient algorithms for multipoint *approximate* polynomial evaluation. Furthermore, in most applications, the computation needs to be done up to some limited accuracy of the machine, say, 32 or 64 bits, whereas the number of points and the degree of the polynomial may grow arbitrarily large. Thus, we desire an algorithm that achieves a fixed accuracy of the output and supports computational complexity estimates that grow linearly (with a small constant multiple) with $m + n$, or at least noticeably slower than $(m+n)\log^2 n$.

A customary requirement for numerical algorithms is numerical stability, and until 1988, the known numerically stable algorithms for the approximate evaluation of a polynomial

$$p(x) = \sum_{i=0}^{n} p_i x^i \qquad (1.1)$$

on a fixed set $X(m)$ of $m + 1$ points,

$$X(m) = \{x_0, x_1, \ldots, x_m\} \,, \qquad (1.2)$$

required the order of mn arithmetic operations (hereafter referred to as to ops).

In 1988 major progress in this area was made by Rokhlin [37], who proposed the innovative idea of approximating the given polynomial only up to the amount of accuracy required, and devised an algorithm that both preserved numerical stability and decreased the computational complexity. Rokhlin's algorithm uses $O(mu + nu^3)$ (infinite precision) ops in order to compute (within the error bound $2^{-u}p$, $p \leq \sum_{i=0}^{n} |p_i|$) the set of the values

$$V(m) = \{p(x_j)\,,\, j = 0, 1, \ldots, m\}, \qquad (1.3)$$

for a positive u and for a set $X(m)$ of (1.2) lying on a real interval $\{x,\ 0 \leq x \leq 1\}$. The number of ops is linear in $m + n$ if u is a fixed constant. The transition to larger linear intervals can be made by means of linear transformations of the variable x and/or of applying the same algorithm on two or more than two adjacent real intervals. The algorithm of [37] is numerically stable and performs well with a finite precision. Although its computational complexity is linear in n and m, the multiplicative factor of u^3 is large even for a relatively small u, and this limits both practical application and theoritical value of the algorithm.

In the present paper we substantially modify the approach of [37] (see our Remark 3.1), making it conceptually simpler. Technically, we rely on the approximation to a given polynomial by means of its interpolation on the Chebyshev set of points by a lower degree polynomial. As in [37], we estimate the arithmetic running time assuming infinite precision computation (with no errors), even though the output values are sought within a certain fixed tolerance to the errors, and we are going to take advantage of such a combination of the powers of numerical and symbolic techniques.

Indeed, whereas the algorithm of [6] exactly computes the values $p(x_j)$, for $j = 0, 1, \ldots, m$, in $O((m + n)\log^2 n)$ ops, already our first Algorithm 3.1 of section 3 decreases this cost estimate, to $O((m + u)\log^2 u + n \min\{u, \log n\})$ ops because, instead of computing exactly, we approximate $p(x_j)$ within the error bound $2^{-u}p$, $p \leq \sum_{i=0}^{n} |p_i|$. The complexity bound of [6] (for the exact evaluation) exceeds this our bound (for approximation), by more than a constant factor, whenever $\log u = o(\log n)$. Our upper bound (on the complexity of the approximation) turns out to be strictly less than the lower bound of [39] (for the exact multipoint evaluation) already for u of the order of $\log n$. Our bound is also superior to the cited bound of [37] for the approximate evaluation problem.

Our algorithms allow NC and processor efficient parallel implementation. To specify the resulting complexity estimates, let $O(t,p)$ denote the bounds $O(t)$ on the arithmetic parallel time and, simultaneously, $O(p)$ on the number of processors used. [Note that according to a variant of Brent's scheduling principle, the bounds $O(t, ps)$ imply the bounds $O(ts, p)$ for any positive integers t, p and s; in particular, $O(t,p)$ implies the sequential bound of $O(tp)$ ops.]

Now, we may formally state our estimates, supported by Algorithm 3.1. Hereafter, $\log^{(0)} N = N$, $\log^{(h)} N = \log_2 \log^{(h-1)} N$, $h = 1, \ldots, \log^* N$, $\log^* N = \max\{h : \log^{(h)} N > 0\}$, for N standing for d, n, and u.

Theorem 1.1. *Given natural m and n, positive $g < 1$ and u, $m+1$ points $x_0, \ldots, x_m$ on a real interval $\{x: -g \le x \le g\}$, and the coefficients of a polynomial $p(x) = \sum_{i=0}^{n} p_i x^i$, it is possible, by using infinite precision arithmetic, to approximate the values of $p(x)$ at $x_0, \ldots, x_m$ within the error bound $E^* = 2^{-u}p, p \le \sum_{i=0}^{n} |p_i|$, at the cost of $T = O((\log^2 u)(m+u) + n$ min $\{u, \log n\})$ arithmetic operations or at the parallel cost $O(t, T/t)$ where T is defined above, and $t = (\log^2 u)\log^* u +$ min $\{u, (\log n)\log^* n\}$.*

Remark 1.1. The factors of $\log^* N$, for $N = d, n, u$, are transfered to our parallel time estimates from the complexity bounds of [3] for polynomial division. These factors can be eliminated by using the alternate estimates of [3] if we agree to increase the processor bounds so that their products with the time bounds grow by the factor of $1 + 2^{-h}\log^{(h)} N$, for any choice of h such that $1 \le h \le \log^* N$.

Algorithm 3.1 first evaluates the coefficients of an auxiliary polynomial $v(x)$ (of a lower degree) approximating $p(x)$ on the set $X(m)$ and then evaluates $v(x)$ on $X(m)$. The first stage, of the evaluation of the coefficients of $v(x)$, involves irrational values of Chebyshev's real points, which deprives us of a chance of using the advantages of modular (residue) arithmetic. (Recall, in particular, that we may implement a rational algorithm applying the Chinese remainder theorem and other customary techniques of symbolic and algebraic computing [3], so as to reduce the precision of computations, by performing them modulo several fixed primes $p_1, \ldots, p_s$.)

In sections 4-7 we modify the first stage of this algorithm to ensure its *rationality*, so as to perform the computations of $v(x)$ over the field generated by the input variables $p_0, \ldots, p_n$, and we may use modular arithmetic to our advantage whenever $p_0, \ldots, p_n$ are rational. [At the last stage of the evaluation of $v(x)$ on the set $X(m)$, we have a choice between using an infinite precision rational algorithm ([22], [3]), at the cost $O((\log^2 d)\log^* d, m/\log^* d)$, or a numerically stable algorithm ([23]), at the cost $O(d, m)$.]

We achieve the rationalization of the algorithm with no increase and, surprisingly, with some decrease of its estimated computational cost [see the estimates (6.1), (6.2), (7.1) and Remark 6.1, for $d = O(u)$]. Moreover, this decrease becomes substantial [so that we reach the overall complexity bound $O(\log u, u)$, at the first stage, and $O((\log^2 u)\log^* u, ((u/\log u) + m)/\log^* u)$ for the entire algorithm], provided that we allow cost-free precomputation of some functions in n and u, which are independent of $x_0, \ldots, x_m, p_0, \ldots, p_n$ (see Theorem 6.1 in section 6).

In section 9, we study Trummer's problem. We combine numerical and algebraic approaches to improve the known solutions to this problem, which has well known applications to the solution of integral equations [36], to fluid mechanics [13], and to numerical evaluation of the Riemann zeta-function [25]. Furthermore, as we have already mentioned, Trummer's problem is equivalent to the two-dimensional n-body problem of simulation of n charged particles in the electrostatic or gravitational potential fields.

Technically, *Trummer's problem* is the problem of multiplying a structured matrix $S = (s_{i,j})$ by a vector $\vec{y}$ where

$$s_{i,j} = \begin{cases} \frac{1}{c_i - c_j} & \text{if } i \ne j, \\ 0 & \text{if } i = j. \end{cases}$$

The problem was originally posed by Golub in SIGACT News [11], as a challenge to use the structure of the problem to obtain an algorithm with running time $o(n^2)$. Indeed, Gerasoulis [10] gave an $O(n \log^2 n)$ time algebraic (and numerically unstable, for large n) algorithm that exactly solved Trummer's problem, whereas the approximation algorithms of Greengard and Rokhlin (known as the *multipole algorithms*) decreased the runnning time bound, first to $O(np^2)$ [36,14] and then to $O(np\log p)$ [15] where the output values were accurate to p bits. (We also refer the reader to [19, 5, 20, 21] for various implementations of multipole algorithms.) The factor of $p\log p$, however, is still quite large in some applications (for instance, n-body simulation typically requires about 16-bit precision in applications to molecular dynamics, and often much larger values of p are needed for the high accuracy planetary simulations).

In section 9, we will give an improved approximation algorithm that runs in time $O(n\log^2 p)$, thus saving a factor of $\Theta(p/\log p)$. In addition, our algorithm relaxes some restrictive assumptions required for the Greengard-Rokhlin algorithm.

2 Auxiliary Results for Polynomial Evaluation

For a fixed positive a, we identify the set $X(a, d)$ of the d Chebyshev points (nodes) on the real interval

$\{x\colon -a \le x \le a\}$ as follows:

$$X(a,d) = \{t_{2h+1} = a\cos\left(\tfrac{2h+1}{d}\pi\right), \quad h = 0,1,\ldots,d-1\} . \tag{2.1}$$

Evaluation of a polynomial of degree k on the set $X(a,d)$ for any $k \ge d$ and interpolation (for $k = d$) can be performed by using $T^* = O(k \min\{d, \log k\} + d\log^2 d)$ ops ([22]) or in parallel implementation, at the cost $O(d,k)$ or, alternatively, $O(\log^2 d + (\log k)\log^* k, (k/\log^* k)+d)$ (see Appendix A or [28]).

We will extend the latter algorithms to any real set $X(m)$ of (1.2), based in particular on the following

Theorem 2.1 ([9], p. 120). *For real a and b, a natural d, and a real function $f(x)$, having the d-th order bounded derivative $f^{(d)}(x)$ on the interval $\{x\colon b \le x \le a\}$, such that $a \ge b$,*

$$M = \max_{b\le x\le a} |f^{(d)}(x)| ,$$

let a polynomial $v(x) = \sum_{i=0}^{d-1} v_i x^i$ interpolate to $f(x)$ on the set of the Chebyshev points of (2.1), that is, let

$$v(t_{2h+1}) = f(t_{2h+1}) , \qquad h = 0,1,\ldots,d-1 . \tag{2.2}$$

Denote that

$$E = E(f,a,b,d) = \max_{b\le x\le a} |f(x) - s(x)|.$$

Then

$$E \le \frac{2M}{d!}\left(\frac{a-b}{4}\right)^d .$$

Since we may shift from $f(x)$ to $f(-x)$, we will hereafter assume that $a \ge |b|$, so that

$$0 \le a - b \le 2a . \tag{2.3}$$

We will apply Theorem 2.1 in the case where $a < 1$, and we will further assume that $d/(1-a)$ is an integer, so that

$$r = 1/a > 1 , \left\lceil\frac{d}{1-a}\right\rceil = \frac{d}{1-a} = \frac{dr}{r-1}, \tag{2.4}$$

and furthermore, $f(x)$ is the polynomial $p(x) = \sum_{i=0}^n p_i x^i$ of (1.1). In this case we will deduce that

$$E = E(f,a,b,d) \le E^* = \left(\frac{r}{de}\right)^{1/2} \sum_{i=d}^{n} |p_i| \left(\frac{1-br}{4r-4}\right)^d \tag{2.5}$$

where $e = 2.71828\ldots, \frac{1-br}{4r-4} \le \frac{1}{2r-2}$, due to (2.3), (2.4).

Remark 2.1. Observe that $1 - br \le 2$ for $|b| \le a$ and $1 - br = 2$ for $b = -a$. Denote that $p = \sum_{i=d}^n |p_i|$ and deduce from (2.5) that

$$E^* = (r/(de))^{1/2} p/(2r-2)^d ,$$

for $b = -a$. In particular,

$$E^* = p(2/(de))^{1/2} 2^{-d}, \text{ for } r = 2,\ a = -b = 1/2 ,$$
$$E^* = p(3/(de))^{1/2} 4^{-d}, \text{ for } r = 3,\ a = -b = 1/3 ,$$
$$E^* = 2p(de)^{-1/2} 6^{-d}, \text{ for } r = 4,\ a = -b = 1/4 .$$

Remark 2.2. Since the error bound E^* rapidly decreases as a and/or $a - b$ decrease, we may greatly decrease the approximation error if we partition the original interval $\{x\colon b \le x \le a\}$ into two or several smaller subintervals and approximate $p(x)$ separately on each of them.

To prove (2.5), apply Theorem 2.1, take into account (2.3), and deduce that

$$\begin{aligned} E &\le 2\left(\tfrac{a-b}{4a}\right)^d \left|\textstyle\sum_{i=d}^n p_i a^i \binom{i}{d}\right| \\ &\le 2m(a,d)\left(\tfrac{a-b}{4a}\right)^d \textstyle\sum_{i=d}^n |p_i| , \end{aligned} \tag{2.6}$$

where $\dfrac{a-b}{4a} = \dfrac{1-br}{4}$ [see (2.4)],

$$m(a,d) = \max_{d\le h\le n} \left(a^h \binom{h}{d}\right) .$$

Since $a^{h+1}\binom{h+1}{d} \le a^h\binom{h}{d}$ for $h \ge d$ if and only if $(h+1)a \le h - d$, it follows that

$$m(a,d) = a^H \binom{H}{d} = (1/r^H)\, H!/(d!(H-d)!)$$

where $r = (1/a) > 1$ [see (2.4)] and

$$H = \left\lfloor\frac{d}{1-a}\right\rfloor = \frac{dr}{r-1}.$$

Applying Stirling's formula, we obtain that

$$m(a,d) < \frac{(H/r)^H}{d^d (H-d)^{H-d}} \left(\frac{H}{d(H-d)e}\right)^{1/2}.$$

We have that

$$H - d = \frac{H}{r} = \frac{d}{r-1},$$

so that

$$m(a,d) < \frac{1}{(r-1)^d}\left(\frac{r}{de}\right)^{1/2}.$$

Combine this estimate with (2.6) and obtain (2.5).

3 The Evaluation – Interpolation – Evaluation Algorithm

We will combine the results of section 2 with those of [22] (or [28]) and [3] to arrive at a fast algorithm for real multipoint approximate polynomial evaluation.

Algorithm 3.1.
Input: a positive rational $a < 1$, natural m, n and d, real $p_0, p_1, \ldots, p_n$ and $x_0, x_1, \ldots, x_m$ [compare (1.1) and (1.2)], such that (2.4) holds and

$$|x_j| \leq a \ , \qquad j = 0, 1, \ldots, m \ .$$

Output: real values $v_0, v_1, \ldots, v_m$ such that

$$|v_j - p(x_j)| \leq E^* \ , \qquad j = 0, 1, \ldots, m \ ,$$

for $p(x)$ of (1.1) and E^* of (2.5).
Computations.

Stage 1, *evaluation on the set of the Chebyshev nodes.* Compute

$$p(t_{2h+1}) = \sum_{i=0}^{n} p_i t_{2h+1}^i \ , \qquad h = 0, 1, \ldots, d-1 \ ,$$

for $t_1, t_3, \ldots, t_{2d-1}$ of (2.1).

Stage 2, *interpolation.* Compute the coefficients of the polynomial $v(x) = \sum_{i=0}^{d-1} v_i x^i$ such that $v(t_{2h+1}) = p(t_{2h+1})$, $h = 0, 1, \ldots, d-1$ [compare (2.2)].

Stage 3, *multipoint evaluation.* Compute and output the values

$$v_j = v(x_j) \ , \qquad j = 0, 1, \ldots, m \ .$$

The correctness of the algorithm immediately follows from (2.5).

At stages 1 and 2, application of the algorithms of [22], [28] yields the complexity estimates $O(d, n)$ or, alternatively, with the incorporation of a parallel polynomial division algorithm of [3], yields the estimates $O(\log^2 d + (\log n) \log^* n, (n/\log^* n) + d)$. At stage 3, we may apply the algorithm of [22], incorporating an algorithm of [3], for polynomial division, to reach the cost bound

$$O(\log^2 d \log^* d, (m + d)/\log^* d).$$

Alternatively, we may apply a slower but numerically stable algorithm having the cost bounded by $O(d, m + n + d)$. The estimates of Theorem 1.1 follow since $d = O(\log(p/E^*))$, according to (2.5) and Remark 2.1, and since we may approximate any positive $g < 1$ by a rational $a, g \leq a \leq 1$, and choose $d = d(a) \to \infty$ satisfying (2.4).

Remark 3.1. The algorithm of [37] also uses Chebyshev points, but only in order to approximate each monomial x^i as an exponential function in i. This leads to a distinct algorithm and distinct complexity estimates.

4 The Evaluation-Division Algorithm

Algorithm 3.1 still works if, instead of the coefficients $p_0, \ldots, p_n$, a black box subroutine for the evaluation of $p(x)$ on the set (1.2) of the Chebyshev nodes is available. In particular, the algorithm can be applied to the evaluation of the characteristic polynomial of a matrix without computing its coefficients. This is required, for instance, in some algorithms for matrix eigenvalues [4].

On the other hand, Algorithm 3.1 has a deficiency: generally, the values t_{2h+1} of (2.1) are not rational, and we cannot use rational arithmetic at stages 1 and 2. Thus, we should apply numerical computation with roundoff errors. To bound the resulting output errors, we may apply the known numerically stable algorithm for polynomial evaluation [23] and interpolation ([40], [12]), increasing the computational cost of stages 1, 2 and 3 to $O(d, n)$, $O(d, d)$ and $O(d, m)$, respectively. The resulting algorithm as a whole will not be numerically stable, however. Indeed, usually, the errors introduced in stage 1 are greatly magnified in stage 2 (of interpolation) because interpolation is an ill-conditioned computational problem (see Appendix B).

If, however, the coefficients of $p(x)$ are available and if they are rational, we may avoid the above difficulties by modifying Algorithm 3.1 as follows:

Algorithm 4.1.
Input, output are as in Algorithm 3.1.
Computations.

Stage 1. Evaluate the coefficients of the polynomial

$$L(x) = \prod_{h=0}^{d-1} (x - t_{2h+1}) \tag{4.1}$$

where t_{2h+1} are defined by (2.1).

Stage 2. Evaluate the coefficients of the polynomial $w(x) = p(x) \bmod L(x)$ and set $v(x) = w(x)$.

Stage 3, as in Algorithm 4.1.

Correctness of Algorithm 4.1 follows since, clearly, $p(t_{2h+1}) = w(t_{2h+1})$ for all h.

Stage 1 can be performed at the cost $O(\log^2 d, d)$ ([1] or [3]), stage 2 at the cost $O(d, n)$ or, alternatively, $O((\log n) log^* n, n/\log^* n)$ ([3]). This only leads to improving the cost bounds of stages 1 and 2 of Algorithm 3.1 by a constant factor, but we will next show how to make the computations at stages 1 and 2 of Algorithm 4.1 rational and also how to decrease their complexity further, by exploiting the following expressions (already used in [28]):

$$t_{2h+1} = (a/2)(\omega_{2d}^{2h+1} + \omega_{2d}^{-(2h+1)}) \ , \ h = 0, 1, \ldots, d-1 \ , \tag{4.2}$$

where $\omega_{2d} = \exp(\pi\sqrt{-1}/d)$ is a primitive $(2d)$-th root of 1.

5 Rationalization of Stage 1 of Algorithm 4.1

Let us set

$$t_h = \frac{a}{2}(\omega_{2g}^h + \omega_{2g}^{-h}),\ h = 0, 1, \ldots, 2g-1,$$

so that

$$L(x) = \prod_{h=0}^{d-1}(x - t_{2h+1}) = L_{2d}^*(x)/L_d^*(x)$$

where $L_g^*(x) = \prod_{h=0}^{g-1}(x - t_h)$, for $g = d$, $g = 2d$.

We will now compute the coefficients of $L_g^*(x)$, for $g = d$, $g = 2d$, and then will divide $L_{2d}^*(x)$ by $L_d^*(x)$, to obtain $L(x)$.

To evaluate the coefficients of the polynomial $L_g^*(x)$, for $g = d$, $g = 2d$, we will first observe the following simple expressions for the power sums of its zeros:

$$\begin{aligned}\textstyle\sum_{h=0}^{g-1} t_h^k &= \textstyle\sum_{i=0}^{g-1}(a/2)^k(\omega_g^i + \omega_g^{-i})^k \\ &= \begin{cases} 0\ , & \text{for odd } k\ , \\ g\binom{k}{k/2}(\frac{a}{2})^k\ , & \text{for even } k\ , \end{cases}\end{aligned}$$

and then will apply the rational algorithm of [38] (see also [30], appendix) to make the transition from the power sums to the coefficients, at the cost $O(\log^2 d,\ d/\log d)$, which dominates the cost of the subsequent division of $L_{2d}^*(x)$ by $L_d^*(x)$.

The latter algorithm, as well as any algorithm for the transition from the power sums to the coefficients, requires division by $2, 3, \ldots, 2d$, which cannot be performed in some fields. To compute $L(x)$ over any field (containing $a/2$), we may multiply together, modulo ω_{2d}^d+1, all the d linear factors $x - t_{2h+1}$ of (4.1), substituting the expressions (4.2) for t_{2h+1} and treating ω_{2d} as the second variable. The cost of this computation increases to $O(\log^2 d, d^2)$.

Remark 5.1. Another direction (which we leave as a chalenge to the reader) is to apply Newton's identities in order to extend the above explicit expressions for the power sums to the similar expressions for the coefficients of $L_g^*(x)$.

Remark 5.2. All our computations are ultimately reduced to polynomial multiplication for which, throughout this paper, we assume the cost bound $O(\log D, D)$, where D bounds the degree of the output polynomial. In fact, this bound turns into $O(\log D,\ D\log\log D)$ for the computation over any field of constants (see [3]), and all our estimates should be corrected respectively (we leave this slight correction to the reader).

6 Computation of the Coefficients of the Reciprocal and Stage 2 of Algorithm 4.1

In this section we will assume $L(x)$ available, and our next task will be the computation of the d trailing coefficients of the reciprocal polynomial $R(x) = (1/L(x)) \bmod x^{n-d+1}$; when this is done, the desired coefficients of $v(x)$ can be obtained as the d trailing coefficients of the polynomial $p(x) - R(x)L(x)$, at the additional cost $O(\log d,\ d)$ of a single multiplication and a single subtraction of degree d polynomials. We first observe that the polynomials $L(x)$ and $R(x)$ do not depend on the set $X(m)$ of (1.2) and on the coefficients of the polynomial $p(x)$ of (1.1) and thus, in principle, can be preprocessed.

Theorem 6.1. *If precomputation of the values depending only on degree n and the precision value u and independent of the input set $X(m)$ of the nodes of (1.2) and of the coefficients of the polynomial $p(x)$ of (1.1) is allowed cost-free, then the complexity estimates of Theorem 1.1 can be decreased to $O(u,\ m + \log u)$ (using numerically stable implementation of stage 3 of Algorithms 3.1 and 4.1) or to $O((\log^2 u)\log^* u, ((u/\log u) + m)/\log^* u)$ (using infinite precision arithmetic).*

Without cost-free preprocessing, we may rationally compute the coefficients of $R(x)$ (by using the known algorithms for polynomial reciprocal) at the cost $O(d, n)$ or, alternatively, $O((\log n)\log^* n,\ n/\log^* n)$ [3]. The overall cost of performing Algorithm 4.1 over the fields of constants allowing divisions by $2, 3, \ldots, 2d$ is thus bounded by

$$\begin{aligned}&O(t', p'),\quad t' = (\log n)\log^* n + (\log^2 d)\log^* d\ , \\ &p't' = m\log^2 d + n\min(d,\ \log n),\end{aligned} \tag{6.1}$$

whereas over any field we reach the cost bounds

$$O(t', p'')\ ;\quad p''t' = (m + d^2)\log^2 d + n\log n. \tag{6.2}$$

In all these cases, with and without preprocessing, the evaluation of the coefficients of $v(x)$ is by a rational algorithm.

Remark 6.1. Besides the reduction of the precision of the computations in the latter case, the cost of the evaluation of $p(x) \bmod L(x)$ can be reduced to

$O(n)$ ops over the field of integers modulo p_i, for each i, provided that $d = O(\log n)$ [18].

Remark 6.2. Numerical division of $p(x)$ by $L(x)$ (with an arbitrarily high output precision) can be performed at the cost $O(\log n,\, n)$ (see [3], [33]).

7 Solution via Reduction modulo z^d+1

In this section, we will develop some ideas of [28] in order to compute the coefficients of $v(x)$ at the cost $O(n, d)$ over any field of constants. This alternative to performing both stages 1 and 2 of Algorithms 3.1 and 4.1 leads to the alternate overall cost bound of

$$O(n,\, d + (m/n)\log^2 d)\ . \tag{7.1}$$

For simplicity, assume (with no loss of generality) that d divides n and that the variable x has been scaled so that $a = 1$. Substitute $x = (z + z^{-1})/2$ into (1.1) and obtain that

$$p(\,(z + z^{-1})/2) = Q(z) + Q(1/z)$$

for some unknown polynomial $Q(z) = \sum_{i=0}^{n} Q_i z^i$. It remains to compute the coefficients of the polynomial $w(x)$ (of degree at most $d-1$) such that

$$w(\,(z+z^{-1})/2) = p(\,(z+z^{-1})/2) \bmod (z^d+1) \tag{7.2}$$

and to set $v(x) = w(x)$. Indeed, due to (7.2), we have that $w(x) = p(x)$ on the set $X(1,d)$ of (2.1), because $x = (z + z^{-1})/2$ for $z = \omega_{2d}^{2h+1}$, $h = 0, 1, \ldots, d-1$, defines precisely the set (2.1).

To compute the coefficients of $w(x)$:

a) first represent $p((z + z^{-1})/2) \bmod (z^d + 1)$ as $q(z) + q(1/z)$ where $q(z)$ is a polynomial of degree at most $d-1$ and

b) then rewrite $q(z) + q(1/z)$ as $w((z + z^{-1})/2)$.

To perform stage a), recursively represent $(z + z^{-1})^i \bmod (z^d + 1)$ (for $i = 1, 2, \ldots, n$) as $q_i(z) + q_i(1/z)$ where $q_i(z)$ are polynomials of degrees at most $d-1$. Specifically, suppose that we know the coefficients of the polynomial $q_i(z) = \sum_{h=0}^{d-1} q_{i,h} z^h$. [We surely know $q_0(x)$ and $q_1(x)$.] Then let

$$q_{i+1,d-1} = q_{i,d-2}\ , \quad q_{i+1,0} = q_{i,1} - q_{i,d-1}\ ,$$

$$q_{i+1,h} = q_{i,h-1} + q_{i,h+1}\ , \quad \text{for} \quad 0 < h < d-1\ ,$$

and thus define $q_{i+1}(z)$. Knowing $q_i(z)$ for all i, we then immediately compute the coefficients of the polynomial $q(z) = \sum_{i=0}^{n} 2^{-i} p_i q_i(z) = \sum_{i=0}^{n} q_i z^i$. The overall cost of this stage is $O(n, d)$.

Remark 7.1. The reader may try to deduce some explicit expressions for $q_{i,h}$ and for all i and h in order to decrease the parallel time of performing stage a).

To perform stage b), proceed recursively, for $h = d-1, d-2, \ldots, 0$. For every h, compute $w_{h,h}$ and then $v_h = 2^h w_{h,h}$ where $w_{h,h}$ is the leading coefficient of the polynomial $w_h(z)$ of degree at most h such that

$$\begin{aligned} w(z) + w(1/z) \; &= q(z) + q(1/z) \\ &\quad - \textstyle\sum_{h=k+1}^{d-1} (q_h(z) + q_h(z^{-1})\,) v_h\ . \end{aligned}$$

8 Discussion

It would be interesting and desirable to improve our results on the evaluation of the coefficients of $L(x)$ (section 5), $R(x)$ and $v(x)$ (section 6) and $w(x)$ (section 7), (compare, in particular, Remarks 5.1 and 7.1).

Our algorithms can be applied recursively, to simplify the multipoint evaluation of $v(x)$ at stage 3 of Algorithms 3.1 and 4.1 if $p = \sum_{i=0}^{n} |p_i|$ substantially exceeds $v = \sum_{h=0}^{d-1} |v_h|$.

A major challenge is an extension of our results to interpolation. As a heuristic approach, we suggest replacing the input set $X(n)$ of $n+1$ nodes of interpolation (6.2) by the set $X(a,d)$ of d Chebyshev nodes (2.1). The input values of $p(x)$ at $x = x_i$, $i = 0, 1, \ldots, m$, are then used in order to approximate the values $p(t_{2h+1})$, for $h = 0, 1, \ldots, d-1$. Then interpolation to $p(x)$ is replaced by the simpler interpolation to a polynomial of degree at most $d-1$ that approximates $p(x)$ at $x = t_{2h+1}$, for $h = 0, 1, \ldots, d-1$. The approach is promising if the points of $X(m)$ closely approximate all the nodes t_{2h+1} of $X(a,d)$. Some difficulties with this heuristic may stem from the fact of ill-conditioning of the interpolation problem (see Appendix B).

Of course, it is interesting to consider approximate interpolation and multipoint evaluation with the complex input nodes, although this case is much harder to treat than the real case (compare [34]).

9 Fast Approximation Algorithm for Trummer's Problem

In this section we present our improved solution of Trummer's problem.

9.1 Quad–Tree Construction

All efficient algorithms for the n-body problem use some form of hierarchical space decomposition. The algorithms of [2] and [8] use variants of quad-trees with $O(n)$ nodes and with particles associated with the leaf nodes. (Quad-trees are also applied, in Weyl's construction, for approximating polynomial zeros [27].) We create a non-trivial modification of these trees, in which particles may be associated with *internal nodes*, as well as the leaf nodes. In addition, particles are "clustered" at nodes in sets of size s, which is a parameter of our decomposition algorithm. In doing so we guarantee that the number of nodes in our decomposition tree T is small [35].

Lemma 9.1. *The number of nodes in T is $O(\frac{n}{s})$. Moreover, building the tree T from the $O(n)$ node trees of [2, 8] can be done optimally, in sequential time $O(n)$ or by using $O(\frac{n}{\log s})$ processors and $O(\log s)$ time on a PRAM.*

9.2 A Fast Algorithm

Fast approximation algorithms for Trummer's problem operate by computing power series approximations to the desired potential function. In two dimensions this is particularly easy — since the potential function is a harmonic function, there is an analytic function in one complex variable that completely describes the potential function over the plane. It is this analytic function that we approximate with a power series. The two types of power series expansions required are a standard Taylor series (for representing the potential within a local disk, due to particles outside the disk) and a Taylor series in $1/z$, which is known as the multipole expansion (for representing the potential *outside* a disk, due to particles within the disk).

The five major components of such an approximation algorithm are

- Creating the initial multipole expansions
- Translating multipole expansions
- Converting multipole expansions to Taylor series expansions
- Translating Taylor series expansions
- Evaluating local forces

As proved by Greengard and Rokhlin [14], to perform computations with p bits of accuracy, only $O(p)$ terms of the power series expansions need to be kept. It is fairly clear from the form of the equations that translation of multipole expansions and Taylor series, as well as the conversion between these two expansions, can be done quickly via a reduction to convolution requiring $O(p \log p)$ time for each power series operation.

Let us next create the initial multipole expansions. For s particles at locations $z_1, z_2, \cdots, z_s$ with charges $q_1, q_2, \cdots, q_s$, the multipole expansion about the origin can be written (see, for example, [14]) as

$$\Lambda(z) = Q \ln z + \sum_{k=1}^{p} \frac{a_k}{z^k},$$

where

$$Q = \sum_{i=1}^{s} q_i \,, \quad a_k = \sum_{i=1}^{s} \frac{-q_i z_i^k}{k}.$$

We now consider computing only the a_k coefficients, as Q can be computed in additional $O(s)$ time.

We just need to compute the row vector $(-a_1, -2a_2, \cdots, -ka_k, \cdots, -pa_p)$, and then $O(p)$ divisions will produce exactly the multipole coefficients. Since $-ka_k = \sum_{k=1}^{s} q_i z_i^k$, it will remain to multiply the row vector $(q_1, q_2, \cdots, q_s)$ by the $s \times p$ matrix $Z = (z_{i,k} = z_i^k, i = 1, 2, \cdots, s; i = 1, 2, \cdots, p)$. We choose $s = p$, so that Z is a $p \times p$ Vandermonde matrix. Due to its special structure, we can premultiply it by a vector in time $O(p \log^2 p)$ [7,29], so that calculating all the coefficients of the multipole expansion can be done in time $O(p \log^2 p + p)$.

Finally, we need to consider the complexity of the last step of the multipole algorithm, which is the stage of calculating the local forces. This step involves two types of operations: (1) computing the force on every particle within a region of the space decomposition due to particles in another region, and (2) computing the force on every particle in a region due to the other particles in the *same* region. Clearly, each of these problem is just a smaller version of the original problem, which we solve by using Gerasoulis' algebraic algorithm [10] in time $O(s \log^2 s)$. It is shown in the full paper [35] that the total number of local calculations that need to be done is proportional to the size of the decomposition tree, which is $O(n/s)$ by Lemma 9.1, and which results in the overall $O(n \log^2 s)$ time for *all* local force calculations. We now arrive at the main result of this section.

Theorem 9.1. *Given the decomposition tree of [2,8], having $O(n)$ nodes, associated with the n-body problem in two dimensions, we can approximate (to p bits) the values of the forces acting on all the n particles in total $O(n \log^2 p)$ time.*

Proof: Each major operation in the multipole algorithm can be charged to a particular node in the tree, and no node will ever be charged by more than a constant number of these operations. In other words, there are at most $O(n/s)$ evaluations, translations, and conversions (see [35] for the details). We estimated the complexity of each of these operations above: putting all of these complexity estimates together, we can see that the entire algorithm takes time

$$O\left(\frac{n}{s}\left(p\log p + s\log^2 p + s\log^2 s\right)\right),$$

which for $s = p$, turns into $O(n\log^2 p)$.

[35] also contains a substantial improvement of the previous multipole algorithms for the *three-dimensional* n-body problem (which is the most customary means of particle simulation), but in this case the resulting time bound is still far larger than the known lower bound.

Appendix A. Special Techniques for Chebyshev's Nodes.

To simplify the evaluation of a polynomial $p_n(x)$ of (1.1) on the set $X(1,d)$ of Chebyshev's nodes and the interpolation by $p(x)$ on the set $X(1,d)$ (provided in the latter case that $n = d-1$), represent $p(x)$ as $p_0(x^2) + xp_1(x^2) = q_0(y) + xq_1(y)$ where $y = 1-2x^2$. The substitution of $y = 1-2x^2$ for x maps the Chebyshev set $X(1,d)$ of (1.2) into the half-size Chebyshev set $X(1,d/2)$. We may assume for simplicity that d and n are integer powers of 2 and recursively apply the above relations to obtain the divide-and-conquer algorithms for both evaluation and interpolation (see more details in [28]). The resulting algorithms slightly imporve the parallel complexity estimates based on the algorithm of [22], incorporating the polynomial division algorithm of [3] (compare the beginning of section 2).

Appendix B. Error Magnification in the Interpolation.

Fix a set $X(m)$ of (1.2), denote

$$L(x) = \prod_{k=0}^{m}(x - x_k)\ ,$$

$$\ell_k(x) = L(x)\,/\,(x - x_k)\ ,\quad k = 0,1,\ldots,m\ ,$$

assume that $m = n$, and recall Lagrange's interpolation formula,

$$p(x) = \sum_{k=0}^{m} p(x_k)\,\ell_k(x)\,/\,\ell_k(x_k)\ .$$

Observe that an error ϵ in the value of $p(x_k)$ causes the output perturbation $\epsilon\,\ell_k(x)/\ell_k(x_k)$. This perturbation is large if $|\ell_k(x_k)|$ is small, which is frequently the case. For instance, if $x_k = -a + 2ak/m$, $k = 0,1,\ldots,m$, then

$$\ell_m(x_m) = \max_k |\ell_k(x)| = (2a/m)^m m!\ ,$$

so that

$$(2a/e)^m m^{1/2} e^{3/4} \leq \ell_m(x_m) \leq (2a/e)^m e m^{1/2}\ . \quad (B.1)$$

If $m = d-1$ and if $X(m) = X(a,d)$ is the Chebyshev set (2.1), then the magnificaion of the errors at $x_{d-1} = a$ (and similarly at $x_0 = -a$) is greater than (B.1) indicates, since the nodes t_{2h+1} are accumulated near $x_{d-1} = a$ and $x_0 = -a$, so that the distances from x_k to a are of the order of $(2ak/d)^2$ if $d - k = o(d)$ (and similarly for x_k near $-a$).

Acknowledgements

Pan is supported by NSF Grant CCR 9020690 and PSC CUNY Award #662478. Reif and Tate are supported by DARPA/ISTO Contracts N00014-88-K-0458, DARPA N00014-91-J-1985, N00014-91-C-0114, NASA subcontract 550-63 of prime contract NAS5-30428, US-Israel Binational NSF Grant 88-00282/2, and NSF Grant NSF-IRI-91-00681. The authors are grateful to C. Engle for her superb assistance with typing this paper.

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Mass., 1976.

[2] J. Barnes, P. Hut, A Hierarchical $O(N\log N)$ Force-calculation Algorithm, *Nature*, 324, 446-449, 1986.

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

[4] D. Bini, V. Pan, Practical Improvement of the Divide-and-Conquer Eigenvalue Algorithms, *Computing*, 48, 109-123, 1992.

[5] J.A. Board Jr., R.R. Batchelor, J.F. Leathrum Jr., High Performance Implementations of the

Fast Multipole Algorithm, *Symposium on Parallel and Vector Computation in Heat Transfer, Proc. 1990 AIAA/ASME Thermophysics and Heat Transfer Conference*, 1990.

[6] A. B. Borodin, I. Munro, Evaluating Polynomials at Many Points, *Information Processing Letters*, 1:2, 66–68, 1971.

[7] J. F. Canny, E. Kaltofen, Y. Lakshman, Solving System of Nonlinear Polynomial Equations Faster, *Proc. ACM-SIGSAM Intern. Symp. on Symbolic and Algebraic Comp.*, 121-128, 1989.

[8] J. Carrier, L. Greengard, V. Rokhlin, A Fast Adaptive Multipole Algorithm for Particle Simulations, *SIAM Journal of Scientific and Statistical Computing*, 9, 4, 669-686, 1988.

[9] G. Dahlquist, A. Björck, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

[10] A. Gerasoulis, A Fast Algorithm for the Multiplication of Generalized Hilbert Matrices with Vectors, *Mathematics of Computation*, 50, 181, 179-188, 1988.

[11] G. Golub, Trummer's Problem, *SIGACT News*, 17, 2, 12, 1985.

[12] G. H. Golub, C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1989.

[13] L. Greengard, Potential Flow in Channels, *SIAM Journal on Scientific and Statistical Computing*, 11, 4, 603–620, 1990.

[14] L. Greengard, V. Rokhlin, A Fast Algorithm for Particle Simulations, *Journal of Computational Physics*, 73, 325-348, 1987.

[15] L. Greengard, V. Rokhlin, On the Efficient Implementation of the Fast Multipole Algorithm, Technical Report RR-602, *Yale University, Department of Computer Science*, 1988.

[16] R. A. Horn, C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, 1985.

[17] W. G. Horner, *Philosophical Transactions, Royal Society of London*, 109, 308-335, 1819.

[18] M. Kaminski, Linear Time Algorithm for Residue Computation and a Fast Algorithm for Division with a Sparse Divisor, *J. of ACM*, 34, 4, 968-984, 1987.

[19] J. Katzenelson, Computational Structure of the N-body Problem, *SIAM Journal of Scientific and Statistical Computing*, 10, 787-815, 1989.

[20] J.F. Leathrum Jr., J.A. Board Jr., Parallelization of the Fast Multipole Algorithm Using the b012 Transputer Network, *Transputing T91*, IOS Press, Washington, D.C., 1991.

[21] J. F. Leathrum Jr., J. A. Board Jr., Mapping the Adaptive Multipole Algorithm onto Mind Systems, *NASA/ICASE Workshop: Unstructured Parallel Scientific Computing*, MIT Press, Cambridge, MA (in press), 1992.

[22] R. Moenck, A. B. Borodin, Fast Modular Transforms via Division, *Conference Record, IEEE 13th Annual Symposium on Switching and Automata Theory*, 90–96, 1972.

[23] A. C. R. Newbery, Error Analysis for Polynomial Evaluation, *Math. Comp.*, 28, 127, 789–793, 1979.

[24] I. Newton, *Analysis per Quantitatem Series*, **10** London, 1711 (also see D. T. Whiteside, ed., *The Mathematical Papers of Isaac Newton*, 2, 222, Cambridge Univ. Press, 1968).

[25] A. M. Odlyzko, A. Schönhage, Fast Algorithm for Multiple Evaluations of the Riemann Zeta Function, *Transactions of the American Math. Soc.*, 309, 2, 1988.

[26] V. Y. Pan, Methods of Computing Values of Polynomials, *Russian Mathematical Surveys*, 21, 1, 105–136, 1966.

[27] V. Y. Pan, Sequential and Parallel Complexity of Approximate Evaluation of Polynomial Zeros, *Computers and Math. (with Applications)*, 14, 591–622, 1987.

[28] V. Y. Pan, Fast Evalulation and Interpolation at the Chebyshev Set of Points, *Applied Math. Letters*, 2, 3, 255–258, 1989.

[29] V. Y. Pan, On Computations with Dense Structured Matrices, *Mathematics of Computation*, 55, 191, 179-190, 1990.

[30] V. Y. Pan, Parallel Least-Squares Solution of General and Toeplitz-like Linear Systems, *Proc. 2nd Ann.ACM Symp. on Parallel Algorithms and Architecture*, 244-253, 1990.

[31] V. Y. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Rev.*, 34, 2, 225–262, 1992.

[32] V. Y. Pan, Approximate Evaluation of a Polynomial on a Set of Real Points, Techn. Report, *International Computer Science Institute, Berkeley, CA*, 1992.

[33] V. Pan, E. Landowne, A. Sadikou, Polynomial Division with a Remainder by Means of Evaluation and Interpolation, *Proc. of the 3rd Ann. IEEE Symp. on Parallel and Distributed Computing*, 212–217, 1991, and to appear in *Information Processing Letters.*

[34] V. Pan, A. Sadikou, E. Landowne, O. Tiga, A New Approach to Fast Polynomial Interpolation and Multipoint Evaluation, submitted to *Computers and Math. (with Applications).*

[35] J. H. Reif, S. R. Tate, N-body Simulation I: Potential Field Evaluation, Technical Report, *Duke University, Department of Computer Science*, 1992.

[36] V. Rokhlin, Rapid Solution of Integral Equations of Classical Potential Theory, *J.of Comput. Physics*, 60, 187–207, 1985.

[37] V. Rokhlin, A Fast Algorithm for the Discrete Laplace transformation, *J. of Complexity*, 4, 12–32, 1988.

[38] A. Schönhage, The Fundamental Theorem of Algebra in Terms of Computational Complexity, unpublished manuscript, 1982.

[39] V. Strassen, Die Berechnungskomplexität von elementarysymmetrischen Funktionen und von Interpolationskoeffizienten, *Numerische Math.*, 20, 3, 238–257, 1973.

[40] W. Werner, Polynomial Interpolation: Lagrange versus Newton, *Math. of Computation*, 43, 167, 205–217, 1984.