POLYNOMIAL ROOT-FINDING: ANALYSIS AND COMPUTATIONAL INVESTIGATION OF A PARALLEL ALGORITHM

by

B. Narendran
P. Tiwari

Computer Sciences Technical Report #1061

December 1991

Polynomial Root-Finding : Analysis and Computational Investigation of a Parallel Algorithm

B. Narendran *

P. Tiwari

Computer Sciences Department
University of Wisconsin-Madison
Madison, Wisconsin-53706.

Abstract

A practical version of a parallel algorithm that approximates the roots of a polynomial whose roots are all real is developed using the ideas of an existing NC algorithm. An new elementary proof of correctness is provided and the complexity of the algorithm is analyzed. A particular implementation of the algorithm that performs well in practice is described and its run-time behaviour is compared with the analytical predictions.

1 Introduction

In this paper we describe and analyze the behaviour of an implementation of a parallel algorithm that approximates the roots of a polynomial which has only real roots. The polynomial root approximation problem we consider can be defined as follows. Given a positive integer μ, and a polynomial $p_0(x)$ of degree n, whose coefficients are m-bit integers and whose roots $x_1, x_2, \ldots, x_n$ are all real, we wish to compute *μ-approximations* $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n$ respectively to these roots, where the μ-approximation $\tilde{x}_i$ to the root x_i is defined to be the rational number $2^{-\mu}\lceil 2^{\mu} x_i \rceil$.

The primary goals of this implementation (the first such for this algorithm) were to study its performance in practice, particularly with regard

*Partially supported by NSF under grant CCR-9024516.

to speedups attained by the parallel implementation. We also compare its running times with those of another existing (sequential) root-finding routine. In addition, we do a careful analytical study of our implementation with the intention of predicting its run-time behaviour.

In Section 2, we describe the algorithm and provide an elementary proof of correctness. Section 3 discusses some of the implementation details. Section 4 analyzes the time complexity of the implementation and Section 5 presents the actual running times obtained from the implementation.

2 The Algorithm

Ben-Or and Tiwari [BT90] describe an NC algorithm for the root approximation problem being considered. Our implementation is based on the ideas used in this algorithm. We have not, however, implemented the NC version, which, although theoretically efficient, is impractical due to the overheads associated with its fine-grained parallelism.

In the following description of the algorithm, we will assume that the given polynomial of degree n has n distinct real roots, for the sake of keeping the algorithm description simple. Multiple roots are easily handled by a simple modification of the algorithm that is described in subsection 2.3.

The algorithm can be conceptually divided into two stages. In the first stage, we solve a *Root Isolation* problem, wherein a set of n intervals on the real line are identified such that each of these intervals contains exactly one root of the polynomial $p_0(x)$. In the second stage, we solve the *Interval* problem where we compute the μ-approximations to the roots, given the isolating intervals.

I) The Root Isolation Problem

To isolate the roots $x_1 < x_2 < \ldots < x_n$ of $p_0(x)$, we recursively compute the roots of two other polynomials $p_1(x)$ and $p_2(x)$, whose degrees are roughly half that of p_0 and sum to $n-1$. The polynomials $p_1(x)$ and $p_2(x)$ are chosen such that their roots $y_1, y_2, \ldots, y_{n-1}$ *interleave* the roots of $p_0(x)$, i.e.

$$x_1 \leq y_1 \leq x_2 \leq y_2 \leq \ldots \leq y_{n-1} \leq x_n$$

The roots of $p_1(x)$ and $p_2(x)$ thus provide the desired intervals that isolate the roots of $p_0(x)$. We will call any pair $(p_1(x), p_2(x))$ that

has this property an *interleaving pair* for $p_0(x)$. If $p_2(x)$ is of degree zero, then $p_1(x)$ is said to interleave $p_0(x)$. Ben-Or and Tiwari [BT90] define and describe the computation of a class of possible choices of $p_1(x)$ and $p_2(x)$ that satisfy these properties.

II) The Interval Problem

Given an interval $[a, b]$ with the property that there is exactly one root λ of the polynomial $p_0(x)$ in this interval, there are several ways to estimate the root. The method of *bisection* converges to the root by performing a binary search on the interval. Newton's method can be guaranteed to converge quadratically if we start close enough to the root. Other methods are described in [BT90].

The two steps described above can be applied recursively to compute the roots of the two new polynomials $p_1(x)$ and $p_2(x)$. This recursive process gives rise to a binary tree whose nodes correspond to polynomials with the root corresponding to $p_0(x)$. A node corresponding to a polynomial $p_i(x)$ has two children $p_{2i+1}(x)$ and $p_{2i+2}(x)$ whose degrees are roughly half that of $p_i(x)$ and whose roots interleave those of $p_i(x)$. The leaves of the tree correspond to linear polynomials, whose roots are easy to estimate.

It can be shown that computing *all* the polynomials in this tree is easy once we have precomputed a certain sequence of polynomials related to the given polynomial $p_0(x)$. The computation of the tree polynomials is described in Section 2.1. The algorithm used to solve the interval problems is described in Section 2.2.

2.1 Computing the tree polynomials

The *standard remainder sequence* corresponding to the polynomial $p_0(x)$ is the sequence $F_0(x), F_1(x), \ldots, F_s(x)$ defined as follows (see, for instance [Col67]). [1] In the following definition, c_i is the leading coefficient of the polynomial $F_i(x)$.

- $F_0(x) = p_0(x), \;\; F_1(x) = p_0'(x)$.
- $F_2(x) = Q_1(x)F_1(x) - c_1^2 F_0(x)$.

[1] This is similar to the remainder sequence computed by the Euclidean algorithm to compute polynomial gcds (see, for instance [AHU74].

- $F_{i+1}(x)$ is a constant multiple of the negative of the remainder polynomial when $c_i^2 F_{i-1}(x)$ is divided by $F_i(x)$ i.e.,

$$F_{i+1}(x) = \frac{Q_i(x)F_i(x) - c_i^2 F_{i-1}(x)}{c_{i-1}^2}, \text{ where } \deg(F_{i+1}) < \deg(F_i)$$

- $F_s(x)$ is a constant.

The sequence $Q_1(x), Q_2(x), \ldots, Q_{s-1}(x)$ of quotients in the above definition is called the *quotient sequence* corresponding to $p_0(x)$. Ben-Or and Tiwari show that the polynomials in the tree can be computed as functions of subsequences of the quotient sequence.

If the original polynomial $p_0(x)$ had distinct roots, then it is easily verified that each $Q_i(x)$ is a linear polynomial, $s = n$, and that the degree of $F_i(x)$ is $n - i$ (see Theorem 1, Appendix A). Further, [Col67] shows that all the $F_i(x)$'s and $Q_i(x)$'s have integer coefficients.

From the recurrence defining the $F_i(x)$'s it is clear that each $F_i(x)$ in the sequence can be expressed as a linear combination of $F_0(x)$ and $F_1(x)$:

$$F_i(x) = A_i(x)F_0(x) + B_i(x)F_1(x),$$

where the "coefficient" polynomials $A_i(x)$ and $B_i(x)$ can be expressed as follows. If we define the matrices

$$S_1 = \begin{pmatrix} 0 & 1 \\ -c_1^2 & Q_1(x) \end{pmatrix}, \tag{1}$$

$$S_i = \begin{pmatrix} 0 & 1 \\ \frac{-c_i^2}{c_{i-1}^2} & \frac{Q_i(x)}{c_{i-1}^2} \end{pmatrix} \quad 2 \le i \le n-1. \tag{2}$$

then the recurrences defining the $F_i(x)$'s can be expressed in matrix form as

$$\begin{pmatrix} F_i(x) \\ F_{i+1}(x) \end{pmatrix} = S_i \begin{pmatrix} F_{i-1}(x) \\ F_i(x) \end{pmatrix}$$

and

$$\begin{pmatrix} F_i(x) \\ F_{i+1}(x) \end{pmatrix} = S_i S_{i-1} \ldots S_1 \begin{pmatrix} F_0(x) \\ F_1(x) \end{pmatrix} \quad (3)$$

$$= \begin{pmatrix} A_i(x) & B_i(x) \\ A_{i+1}(x) & B_{i+1}(x) \end{pmatrix} \begin{pmatrix} F_0(x) \\ F_1(x) \end{pmatrix} \quad (4)$$

The motivation for defining the sequences $\{F_i(x)\}, \{Q_i(x)\}, \{A_i(x)\}$, and $\{B_i(x)\}$ is the following. Let

$$P_{i,j}(x) = \begin{cases} A_{i-1}(x)B_{j+1}(x) - A_{j+1}(x)B_{i-1}(x), & 1 \leq i \leq j < n. \\ F_{i-1}(x), & 1 \leq i \leq n, \quad j = n \\ 1 & i > j. \end{cases} \quad (5)$$

Then the $P_{i,j}(x)$'s satisfy the following property :

Theorem 1 *The polynomial $P_{i,j}(x)$ has integer coefficients, is of degree $j - i + 1$ and has distinct real roots. In addition, if $i < j$, then, for any $1 \leq i \leq k \leq j \leq n$, the polynomials $P_{i,k-1}(x)$ and $P_{k+1,j}(x)$ are interleaving polynomials for the polynomial $P_{i,j}(x)$.*

Proof: See Appendix A ∎

The above theorem combined with the fact that $P_{1,n}(x) = F_0(x)$ gives us the desired interleaving polynomials in the entire tree. In order to keep the tree balanced, the two children of $P_{i,j}(x)$, $i < j$ are chosen to be $P_{i,k-1}(x)$ and $P_{k+1,j}(x)$ where $k = \lfloor (j - i + 1)/2 \rfloor$.

The definition of the $P_{i,j}(x)$'s gives rise to an alternate notation that we will sometimes use to refer to the nodes of the tree. The tree node corresponding to the polynomial $P_{i,j}(x)$ will be labeled $[i, j]$. The root node is thus labeled $[1, n]$, and each leaf is labeled $[i, i]$, for some i.

The $P_{i,j}(x)$'s can clearly be computed from the sequences $\{A_i(x)\}$ and $\{B_i(x)\}$, which, in turn are computed using Eq. (4). However, in keeping with the bottom-up traversal of the tree by the algorithm, we compute the $P_{i,j}$'s in a slightly different manner. Define the matrices $T_{i,j}$ as follows :

$$T_{i,j} = c_{i-1}^2 S_j S_{j-1} \ldots S_i, \quad n > j \geq i > 1 \quad (6)$$

and

$$T_{1,j} = S_j S_{j-1} \ldots S_1, \quad n > j \geq 1 \tag{7}$$

Then, it can be verified (see Appendix A) that for $1 \leq i \leq j < n$,

$$P_{i,j}(x) = T_{i,j}(2,2). \tag{8}$$

Thus, the polynomial in the tree corresponding to node $[i,j]$ is an element of the matrix $T_{i,j}$. Further, given a particular node $[i,j]$ in the tree and its two children $[i,k-1]$ and $[k+1,j]$, the relevant T matrix for the parent node can be computed in terms of the T matrices of its children as follows :

$$T_{i,j} = \frac{1}{c_k^2 c_{k-1}^2} T_{k+1,j} S_k T_{i,k-1}. \tag{9}$$

2.2 The Interval Problems

In this section, we describe the algorithm used to compute μ-approximations to the roots of a polynomial, given the μ-approximations to the roots of its interleaving polynomials. If the coefficients of the given polynomial $F_0(x)$ are at most m bit integers, it is well known that all its roots (and consequently the roots of all interleaving polynomials in the tree) lie in the interval $[2^{-m}, 2^m]$ (see, for instance [Hou70]). In the rest of this section, we restrict ourselves to the following situation. $P_0(x)$ is a polynomial of degree n and $P_1(x)$ and $P_2(x)$ are a pair of interleaving polynomials for $P_0(x)$. Let $\{x_0, x_1, \ldots, x_{n-1}\}$ be the set of distinct roots of $P_0(x)$ sorted in ascending order and let $\{y_1, y_2, \ldots, y_{n-1}\}$ be the multiset of roots of $P_1(x)$ and $P_2(x)$ sorted in ascending order. Also, let $y_0 = 2^{-m}$ and $y_n = 2^m$. By the interleaving property, we know that the interval $[y_i, y_{i+1}]$ contains precisely one root of $P_0(x)$, namely x_i. However, we do not know the actual roots y_i, but only their rational μ-approximations $\tilde{y}_i$ and hence the intervals we have do not necessarily isolate single roots of $P_0(x)$.

We have the following cases for the value of a μ-approximation $\tilde{x}_i$ to x_i :

Case 1) If $\tilde{y}_i = \tilde{y}_{i+1}$, then clearly, $\tilde{x}_i = \tilde{y}_i$, and we are done.

Case 2) In this case, $\tilde{y}_{i+1} - \tilde{y}_i \geq 2^{-\mu}$. Root x_i then lies in one of three disjoint intervals : $(\tilde{y}_i - 2^{-\mu}, \tilde{y}_i]$, $(\tilde{y}_i, \tilde{y}_{i+1} - 2^{-\mu}]$, or $(\tilde{y}_{i+1} - 2^{-\mu}, \tilde{y}_{i+1}]$. These

cases can be distinguished as follows. Let $sgn(x)$ denote the sign of the real number x. Let r_i denote the number of roots of $P_0(x)$ in the interval $(-\infty, \tilde{y}_i)$. Then,

$$r_i = \begin{cases} i & \text{if } sgn(P_0(-\infty)) = (-1)^i sgn(P_0(\tilde{y}_i)) \\ i+1 & \text{otherwise} \end{cases}$$

Now, we have the following sub-cases :

Case 2a) If $r_i = i + 1$, then $x_i \in (\tilde{y}_i - 2^{-\mu}, \tilde{y}_i]$ and $\tilde{x}_i = \tilde{y}_i$.

Case 2b) If $r_i = i$ and $sgn(P_0(\tilde{y}_i)) = sgn(P_0(\tilde{y}_{i+1})$, then, $x_i \in (\tilde{y}_{i+1} - 2^{-\mu}, \tilde{y}_{i+1}]$ and $\tilde{x}_i = \tilde{y}_{i+1}$.

Case 2c) If $r_i = i$ and $sgn(P_0(\tilde{y}_i)) \neq sgn(P_0(\tilde{y}_{i+1})$, then, x_i is the only root of $P_0(x)$ in $(\tilde{y}_i, \tilde{y}_{i+1} - 2^{-\mu}]$ and we can compute the μ-approximation $\tilde{x}_i$ to the isolated root as described below.

Once we have a true isolating interval (as in case 2c above), we can use several different ways to compute an approximation to the root. Our implementation uses a hybrid method based on Newton's method. Let (a, b) denote the isolating interval containing the single root ξ. Newton's method guarantees quadratic convergence if the iterations are started from a point that is "sufficiently close" to the root ξ. We will call any such starting point a *good* one. Our algorithm has two phases; the first phase serves to narrow the given interval appropriately, so that any point in the resulting interval is a *good* starting point.

The following Lemma due to Renegar [Ren87] makes precise the notion of being "sufficiently close" to the root ξ.

Lemma 2.1 *Let $p(x)$ be a polynomial of degree $\leq n$, $p(\xi) = 0$, and let ρ be the smallest of the distances from ξ to the other roots of $p(x)$. If α satisfies $|\xi - \alpha| \leq \frac{\rho}{5n^2}$, then Newton's iteration, starting at α, converges quadratically from the start.*

In view of Lemma 2.1, we follow the strategy outlined in [BT90] to determine a subinterval $(\hat{a}, \hat{b})$ of (a, b) that contains the desired root ξ, and such that there is no other root of $p(x)$ within $10(\xi - \hat{a})n^2$ of $\hat{a}$ or within $10(\xi - \hat{b})n^2$ of $\hat{b}$. Then, Newton's iteration, starting from any point in $[\hat{a}, \hat{b}]$ converges quadratically to the root ξ.

A "double-exponential sieve" is used to locate the interval $(\hat{a}, \hat{b})$. Let $I_0 = (a, b)$ and let $l_0 = b - a$ denote the length of the interval I_0. By

evaluating $p(x)$ at $a + l_0/2$, we can determine if $\xi \leq a + l_0/2$. Assume, without loss of generality, that $\xi \in (a, a + l_0/2)$. Evaluate $p(x)$ at points $a + l_0/2^i$, $i = 0, 1, 2, \ldots$, to the maximum i_0 such that $\xi \in (a, a + l_0/2^{i_0})$. Let $I_1 = (a, a + l_0/2^{i_0})$ and let l_1 be the length of I_1. If $\xi \geq a + l_1/2$, then, we can isolate the desired interval $(\hat{a}, \hat{b})$ by $\log_2(10n^2)$ bisections of the interval I_1. On the other hand, if $\xi < a + l_1/2$, we repeat the same procedure on I_1, to construct another interval I_2.

2.3 Handling repeated roots

In this section, we briefly sketch a modification to the algorithm that allows it to handle the case where the original polynomial $p_0(x)$ may have repeated roots. For the sake of brevity, we do not provide proofs for the results stated in this section.

Suppose $p_0(x)$ has $n^* < n$ distinct real roots $x_1, x_2, \ldots, x_{n^*}$, with multiplicities $m_1, m_2, \ldots, m_{n^*}$ respectively. Then, if we proceeded to compute the remainder sequence as defined in Section 2.1, we would find that $F_{n^*}(x) | F_{n^*-1}(x)$ and consequently $F_{n^*+1}(x) = 0$.[2]

We extend the remainder sequence (and the corresponding quotient sequence) in the following manner:

$$F_i(x) = 1, \quad n^* \leq i < n, \tag{10}$$

$$F_n(x) = 0, \tag{11}$$

$$Q_i(x) = 1, \quad n^* \leq i < n. \tag{12}$$

We define the matrices S_i and $T_{i,j}$ exactly as before in terms of this extended remainder sequence. If we now let $P_{i,j}(x) = T_{i,j}(x)(2, 2)$, the following version of Theorem 1 can be shown to hold:

Theorem 2 *$P_{i,j}(x)$ has integer coefficients, and is of degree* $\min\{0, n^* - i + 1, j - i + 1\}$. *If degree$(P_{i,j}(x)) > 0$, then $P_{i,j}(x)$ has distinct real roots. In addition, if degree$(P_{i,j}) > 1$, then, for any $1 \leq i \leq k \leq j \leq n$, either the polynomials $P_{i,k-1}(x)$ and $P_{k+1,j}(x)$ form an interleaving pair of polynomials for the polynomial $P_{i,j}(x)$, or degree$(P_{k+1,j}(x)) = 0$ and $P_{i,k-1}(x) = P_{i,j}(x)$.*

Furthermore, it can be shown that $P_{1,n}(x)$ (which has degree n^*) has roots $x_1, x_2, \ldots, x_{n^*}$. Hence, we can now proceed as described in the previous

[2] In fact, $F_{n^*}(x)$ is the polynomial gcd of $F_0(x)$ and $F_1(x)$ and has roots $x_0, x_1, \ldots, x_{n^*}$ with respective multiplicities $m_1 - 1, m_2 - 1, \ldots, m_{n^*} - 1$.

section, and by computing the roots of the polynomial at the root of the tree, we would have determined all the distinct roots of $p_0(x)$.

3 The Parallel Implementation

The parallel root approximation algorithm described above was implemented in the C language on a Sequent Symmetry shared memory machine with 20 processors. In this section, we present an overview of the implementation.

The parallel implementation uses a dynamic scheduling paradigm, where the computations to be performed by the algorithm are divided into a set of tasks that are maintained in a task queue [3]. Whenever a processor becomes free, it picks the first task from the queue to execute. Completion of a task usually causes other tasks to be added to the queue. The "grain" of the tasks was chosen small enough so as to keep all processors busy for as much of the time as possible, and yet not so small as to make the overheads large. The number of processors used by the algorithm was a parameter that could be varied from 1 to the maximum value of 19.

The entire algorithm can be divided into two stages. The first stage is the "precomputation" stage that computes the standard remainder and quotient sequences for the given polynomial. The parallel implementation of this stage is described in Section 3.1. As a run-time option, the implementation allows this stage to be executed sequentially, if so desired. The second stage of the algorithm involves computing and finding the roots of the polynomials in the tree. This stage is described in Section 3.2.

3.1 Computing the Remainder Sequence

Computing the remainder sequence involves $n-1$ iterations, where the i^{th} iteration, $1 \leq i \leq n-1$ involves computing Q_i and F_{i+1} from F_{i-1} and F_i. Each iteration is parallelized as follows. If we let

$$Q_i = q_{i,1}x + q_{i,0} \quad \text{and} \tag{13}$$

$$F_i = f_{i,n-i}x^{n-i} + f_{i,n-i-1}x^{n-i-1} + \ldots + f_{i,0} \tag{14}$$

and recall that $c_i = f_{i,n-i}$, then, from the recurrence relations defining the $F_i(x)$, (see Section 2.1), it is seen that

[3] An earlier implementation used a static scheduling policy

$$q_{i,1} = f_{i-1,n-i+1}\, f_{i,n-i} \quad (15)$$

$$= c_{i-1} c_i, \quad (16)$$

$$q_{i,0} = f_{i,n-i}\, f_{i-1,n-i} - f_{i,n-i-1}\, f_{i-1,n-i+1}, \quad (17)$$

$$f_{i+1,j} = \frac{f_{i,j}\, q_{i,0} + f_{i,j-1}\, q_{i,1} - c_i^2 f_{i-1,j}}{c_{i-1}^2}, \quad 0 \le j \le n-i-1 \quad (18)$$

Thus, once $q_{i,1}$ and $q_{i,0}$ have been computed, computing the coefficients of F_{i+1} involves $3(n-i)$ multiplications, $2(n-i)$ additions and $n-i$ divisions. Each of these $5(n-i)$ operations forms a distinct task of this phase. [4] A process that performs one of the subtractions or divisions makes sure that the corresponding multiplications required have been completed.

3.2 The Tree Computations

This stage of the algorithm involves computing the polynomials in the tree and approximating their roots. Most of the work in this stage is done in a bottom-up traversal of the tree, where tasks corresponding to a node in the tree require the completion of other tasks in its children. Figure 3.2 shows the dependencies of the tasks both within a node of the tree and across nodes.These dependencies require that the status of the tasks completed at each node be maintained for synchronization purposes. The actual construction of the tree and the initialization of the data structures that record the status are done in a top-down phase. The various tasks are described in greater detail below.

The top-down phase consists solely of RECURSE tasks. Initially, the task queue contains a single RECURSE task corresponding to the root node of the tree. When a RECURSE task corresponding to a particular node is executed, it initializes a data-structure that is used later to record the status of tasks completed at that node, and generates RECURSE tasks corresponding to its two children. A RECURSE task corresponding to a leaf node initiates tasks that begin the bottom-up phase at that leaf.

The bottom-up phase consists of several different kinds of tasks as shown in Figure 3.2. Recall that the polynomial corresponding to a particular non-leaf node $[i,j]$ is computed as a particular entry of the corresponding matrix

[4] While this may seem to be too fine a grain of parallelism at first glance, it should be noted that the coefficients of the Q_i and the F_i grow rapidly as i increases.

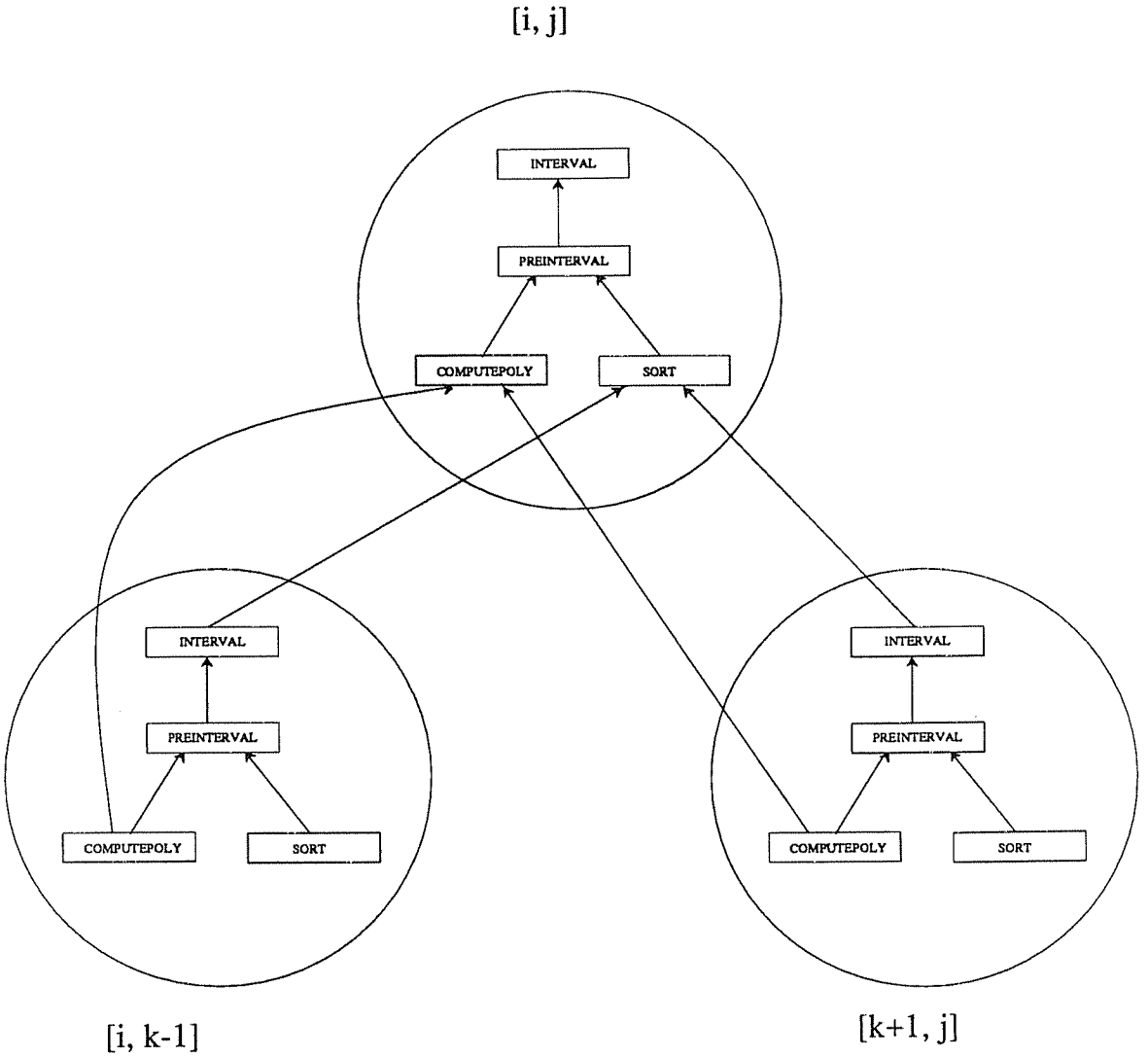

Fig. 3.2 Dependencies among Tasks.

$T_{i,j}$. The task COMPUTEPOLY(i,j) computes this matrix $T_{i,j}$. However, the computation is actually split into several tasks. Recall that the matrix at $[i,j]$ is computed from the matrices corresponding to its children:

$$T_{i,j} = \frac{1}{c_k^2 c_{k-1}^2} T_{k+1,j}\, S_k\, T_{i,k-1}$$

COMPUTEPOLY is not executed as a single task. The two matrix multiplications involved are performed one after the other, with each multiplication being split into four distinct tasks, one for each entry in the result matrix.

The SORT task at node $[i,j]$ merges the two sorted sequences of roots from the two child nodes, to get a sorted sequence of roots that serve as intervals for the parent node's roots. This is performed sequentially as a single task.

The PREINTERVAL tasks perform the preliminary computations described in Section 2.2 that are needed before we can solve the real interval problems. This involves evaluating the polynomial $P_{i,j}$ at each of the interleaving points, and each such evaluation is performed as a distinct task.

The INTERVAL tasks each solve the interval problem for one particular interval as described in Section 2.2.

As described earlier, the status data structures corresponding to the nodes of the tree are used to schedule the tasks. Typically, the completion of a certain task at a node would cause an update of that node's status. Further, if the new status of the current node (and perhaps of its sibling) enables the execution of another task (as determined by the dependencies of Fig. 3.2), that task is added to the queue.

We note that the grain of the tasks in the tree computations phase is, in general, larger than the grain in the phase that computes the remainder sequence. This is justified by the fact that the tree computations generate several tasks that tend to keep all the processors busy, and hence a larger grain can be used without losing any parallelism.

3.3 Implementation of Multi-Precision Arithmetic

All the multi-precision computations required by our algorithm were performed using the UNIX "mp" package that handles integer arithmetic for arbitrarily long integers. This required some changes in the computations, as the algorithm as described above involves arithmetic over the rationals. However, since we only deal with μ-approximations, every rational number x

that we encounter can be identified with the integer $2^{\mu}x$. All computations of the algorithm are performed in this scaled fashion.

The "mp" package uses the straightforward algorithms to perform the basic arithmetic. Thus, addition and subtraction of two n bit numbers takes linear time, and multiplication and divisions take quadratic time.

4 Analysis of Running Time

In this section we analyze the time complexity of the algorithm that was implemented. Our analysis assumes that the running time of the algorithm is dominated by the cost of multiplications performed. This is justified by the fact that of the two arithmetic operations that take quadratic time in the size of their arguments, the number of multiplications is far greater than the number of divisions. [5] We also assume in the following analysis that the roots of the original polynomial are all distinct.

We first estimate bounds on the sizes of all intermediate quantities computed by the algorithm. We will use the following notation in the following analysis. For an integer x, let $\|x\|$ denote the size of x in bits. For a polynomial p, $\|p\|$ will denote the size of the largest coefficient of p. If $A = (a_{ij})$ is a matrix of polynomials, $\|A\|$ denotes the matrix $\max_{i,j}\{\|a_{ij}\|\}$. In a similar fashion, $d(\cdot)$ is used to denote the degrees of polynomials and matrices of polynomials.

Recall that the input is a polynomial $F_0(x)$ of degree n and m-bit coefficients and that μ is the desired precision in the roots. Further, recall that the algorithm induces a tree of polynomials that were referred to in Section 2.1 using the notation $P_{i,j}(x)$. For the purposes of this section, it is useful to introduce an alternate notation for these tree polynomials. The j^{th} polynomial (counting from left to right) at the i^{th} level of the tree (the root being level 0) will also be denoted by $P^{(i,j)}$.

We first look at the degrees of the polynomials. From Lemma 1, we know that

$$d(F_i) = n - i.$$

and

$$d(Q_i) = 1.$$

[5] We also have empirical justification of this assumption that indicates that 75 to 90 percent of the actual running time is spent in multiplications.

The following are easily verified from the definitions of these matrices (see Equations (1), (6) and (7)).

$$d(S_i) = 1, \tag{19}$$

$$\begin{aligned} d(T_{i,i+k-1}) &= d(S_{i+k-1}S_{i+k-2}\ldots S_i) \\ &= k, \quad k \geq 2. \end{aligned} \tag{20}$$

For the coefficient sizes, since $\|c_i\| \leq \|F_i(x)\|$, we will use $\|F_i(x)\|$ as an estimate for $\|c_i\|$. For $i \leq 1$, we have the following bounds.

$$\|F_0(x)\| = m,$$
$$\|F_1(x)\| = \|F_0'(x)\| \leq m + \log n,$$

and

$$\|Q_1(x)\| \leq 2m + \log n.$$

For $i \geq 2$, we use a result of Collins [Col67] that bounds the sizes of the coefficients of $F_i(x)$ by the determinants of certain $(2i-1) \times (2i-1)$ matrices, whose first $i-1$ rows contain as entries the coefficients of $F_0(x)$ and whose last i rows contain as entries the coefficients of $F_1(x)$. Using this fact, we have the following for $i \geq 2$:

$$\begin{aligned} \|F_i(x)\| &\leq (i-1)m + i(m + \log n) + (2i-1)\log(2i-1) \\ &\leq (2i-1)m + (3i-1)\log n + (2i-1). \end{aligned} \tag{21}$$

$$\begin{aligned} \|Q_i(x)\| &\leq \|F_i(x)\| + \|F_{i-1}(x)\| + 1 \\ &\leq 4(i-1)m + (6i-5)\log n + (4i-3). \end{aligned} \tag{22}$$

It can further be shown (see [Col67], [BT90]) that the coefficients of $A_i(x)$ and $B_i(x)$ are similarly bounded by determinants of $(2i-2) \times (2i-2)$ matrices whose first $i-2$ rows contain as entries the coefficients of $F_0(x)$ and whose last i rows contain as entries the coefficients of $F_1(x)$. Using these bounds, we have,

$$\begin{aligned} \|A_i(x)\| &\leq (i-2)m + i(m + \log n) + (2i-2)\log(2i-2) \\ &\leq (2i-2)m + (3i-2)\log n + (2i-2). \end{aligned} \tag{23}$$

$$\begin{aligned} \|B_i(x)\| &\leq (i-1)m + (i-1)(m + \log n) + (2i-2)\log(2i-2) \\ &\leq (2i-2)m + (3i-3)\log n + (2i-2). \end{aligned} \tag{24}$$

It is convenient to let $\beta = 2m + 3\log n + 2$, so that

$$\begin{aligned} \|F_i(x)\| &\leq i\beta. && (25)\\ \|Q_i(x)\| &\leq 2i\beta. && (26)\\ \|A_i(x)\| &\leq (i-1)\beta + \log n. && (27)\\ \|B_i(x)\| &\leq (i-1)\beta. && (28) \end{aligned}$$

Consequently, we have the following bounds on the sizes of the $P_{i,j}(x)$:

$$\begin{aligned} \|P_{i,i+k-1}\| &= \|A_{i-1}(x)B_{i+k}(x) - A_{i+k}(x)B_{i-1}(x)\| \\ &\leq (2i+k-3)\beta + 2\log n + 1 \\ &\leq (2i+k-2)\beta. && (29)\\ \|P_{i,n}\| &= \|F_{i-1}(x)\| \\ &\leq (i-1)\beta, \quad i > 1. && (30) \end{aligned}$$

$$\begin{aligned} \|T_{i,i+k-1}\| &= \left\| \begin{pmatrix} -P_{i+1,i+k-2} & P_{i,i+k-2} \\ -P_{i+1,i+k-1} & P_{i,i+k-1} \end{pmatrix} \right\| \\ &\leq \|P_{i+1,i+k-1}(x)\| \\ &\leq (2i+k-1)\beta. && (31) \end{aligned}$$

In the following analysis, we ignore the costs incurred in scaling the polynomials, the pre-interval problem and sorting the roots at each node of the tree. These phases of the algorithm either perform no multiplications at all or are dominated by other phases of the algorithm. The three computationally intensive phases of the algorithm are computing the remainder sequence, computing the tree polynomials and solving the interval problems. These phases are respectively analyzed in the next three subsections. Table 1 summarizes the asymptotic results. In addition to the bit complexity, Table 1 also gives the arithmetic complexity (number of multiplications) of the different phases. The arithmetic complexities are derived in essentially the same manner as the analysis in the following sections, and we omit the details.

Phase of Algorithm	Arithmetic Complexity	Bit Complexity
Computing Remainder Sequence	$O(n^2)$	$O(n^4(m+\log n)^2)$
Computing Tree Polynomials	$O(n^2)$	$O(n^4(m+\log n)^2)$
Interval Problems (Worst Case)	$O(n^2(\log n + \log^2 X))$	$O(n^3X(X+\beta)(\log n + \log^2 X)$
Interval Problems (Average Case)	$O(n^2(\log n + \log X))$	$O(n^3X(X+\beta)(\log n + \log X)$

Table 1: Asymptotic Complexity of Phases of the Algorithm

4.1 Computing the Remainder Sequence

In this section, we analyze the time taken to compute the remainder sequence $\{F_i(x)\}$ and the corresponding quotient sequence $\{Q_i(x)\}$. The reader is referred to Section 3.1 for the description of the computations performed. The i^{th} iteration ($i \geq 2$) of the loop computes $Q_i(x)$ and $F_{i+1}(x)$ from $F_i(x)$ and $F_{i-1}(x)$, using equations (15). Since we assume that the cost of the multiplications dominate, we are interested only in the $3(n-i)$ multiplications[6] performed in Eq. (15). Computing the products $f_{i,j}q_{i,0}$ and $f_{i,j-1}q_{i,1}$ involve multiplying two numbers whose sizes are $\|F_i(x)\|$ and $\|Q_i(x)\|$. The last product, $c_i^2 f_{i-1,j}$ involves multiplying two numbers whose sizes are $2\|F_i(x)\|$ and $\|F_{i-1}\|$ respectively. Using the size estimates derived in the previous section, we find that the cost of the i^{th} iteration is proportional to the following quantity :

$$\begin{aligned}(n-i)\left[2\|F_i(x)\|\|Q_i(x)\| + 2\|F_i(x)\|\|F_{i-1}(x)\|\right] &\leq (n-i)\left[4i^2\beta^2 + 2i(i-1)\beta^2\right] \\ &\leq 6i^2\beta^2(n-i).\end{aligned}$$

Ignoring lower order terms and summing up for $i = 2, \ldots, n-1$ gives

$$\begin{aligned}6\beta^2\left[n\sum_{i=2}^{n-1} i^2 - \sum_{i=2}^{n-1} i^3\right] &\sim \frac{n^4\beta^2}{2} \\ &= O(n^4(m+\log n)^2)\end{aligned}$$

4.2 Computing the polynomials

In this and subsequent sections of the analysis, we will assume that the degree n is of the form $2^K - 1$, so that the tree induced by the algorithm

[6] We are ignoring here the multiplications performed in computing $F_1(x)$ and $F_2(x)$.

has K levels $0, 1, \ldots, K-1$. Level l has 2^l nodes, each of which corresponds to a polynomial of degree $2^{K-l} - 1$.

Under the above assumption, the following matrix multiplication is performed at the j^{th} node on level l :

$$\left(T_{(2j+1)2^{K-l-1}+1,(2j+2)2^{K-l-1}-1}\right)\left(S_{(2j+1)2^{K-l-1}}\right)\left(T_{2j2^{K-l-1}+1,(2j+1)2^{K-l-1}-1}\right)$$

If the matrices are multiplied in the left-to-right order, the second multiplication's cost dominates and hence we will analyze only that multiplication, namely :

$$\left(T_{(2j+1)2^{K-l-1}+1,(2j+2)2^{K-l-1}}\right)\left(T_{2j2^{K-l-1}+1,(2j+1)2^{K-l-1}-1}\right) = T_R T_L$$

The degrees and coefficient sizes of the polynomials in T_R and T_L are as follows. Let $\alpha = 2^{K-l-1} - 1$. Then,

$$d(T_R) = \alpha + 1, \quad d(T_L) = \alpha.$$

and, recalling from Eq. (31) that $\|T_{i,i+k-1}\| \leq (2i + k - 1)\beta$,

$$\begin{aligned} \|T_R\| &\leq ((4j+3)2^{K-l-1} + 1)\beta \\ &\leq (4j+4)(\alpha+1)\beta. \end{aligned} \tag{32}$$

$$\begin{aligned} \|T_L\| &\leq (2(2j2^{K-l-1} + 1) + 2^{K-l-1} - 2)\beta \\ &\leq (4j+1)(\alpha+1)\beta. \end{aligned} \tag{33}$$

Now, for any matrix of polynomials $A = (a_{ij})$, define

$$md(A) = \max_{i,j}\left(deg(a_{ij})\|a_{ij}\|\right),$$

so that the cost of computing the product AB of two such matrices A and B is bounded by the $8md(A)md(B)$.
Now,

$$md(T_R) \leq ((4j+4)(\alpha+1)^2\beta),$$
$$md(T_L) \leq ((4j+1)\alpha(\alpha+1)\beta).$$

and,

$$md(T_R)md(T_L) \leq (16j^2 + 20j + 4)\alpha(\alpha+1)^3\beta^2. \tag{34}$$

and hence,

$$\begin{aligned} \text{Total work done at level } l &= \sum_{j=0}^{2^l-2} 8(16j^2 + 20j + 4)\alpha(\alpha+1)^3\beta^2 \\ &\sim \frac{4}{3}n^3\alpha\beta^2 + 20n^2\alpha^2\beta^2 + 16n\alpha^3\beta^2 \end{aligned} \tag{35}$$

Summing up over levels $l = 1, 2, \ldots, K-2$ (i.e. $\alpha = \frac{n}{4}, \frac{n}{8}, \ldots, 2$),

$$\begin{aligned} \text{Cost for all levels} &\sim \frac{55}{21}n^4\beta^2 \\ &= O(n^4(m + \log n)^2) \end{aligned} \tag{36}$$

4.3 The Interval Problems

The predominant computation performed in solving the interval problems is the evaluation of various polynomials with integer coefficients at rational points. In our implementation however, we were constrained to use only integer arithmetic. In order to overcome this, the polynomial coefficients had to be scaled appropriately before evaluation. We first estimate the computational cost of evaluating the polynomials in this fashion. Let x be a rational point, and $\mu \geq 0$ an integer such that $2^\mu x$ is an integer. Let $p(x) = p_0 + p_1x + p_2x^2 + \ldots + p_dx^d$ be the polynomial that we wish to evaluate at x and let $m = \|p(x)\|$. Let $X = \|2^\mu x\|$. Clearly, $X \geq \mu$. The scaled version of the polynomial that we are interested in is

$$p_\mu(x) = 2^{d\mu}p_0 + 2^{(d-1)\mu}p_1x + \ldots + p_dx^d.$$

Observe that

$$p_\mu(2^\mu x) = 2^{d\mu}p(x).$$

We use Horner's rule to evaluate the polynomials. After the i^{th} iteration of Horner's rule, the partial evaluation obtained is

$$E_i(x) = 2^{i\mu}p_{d-i} + 2^{(i-1)\mu}p_{d-i+1}x + \ldots + 2^{0\mu}p_dx^i.$$

Now,

$$\begin{aligned} \|E_i(x)\| &\leq m + i\max\{X,\mu\} + \log(i+1) \\ &\leq m + iX + \log(i+1). \end{aligned}$$

and hence,

$$\begin{aligned} \text{Cost of the } i^{\text{th}} \text{ step} &\leq X(m + (i-1)X + \log i) \\ &\leq mX + X^2(i-1) + X\log i. \end{aligned}$$

The cost of the entire evaluation is thus bounded by

$$\begin{aligned} \sum_{i=1}^{d}\left(mX + X^2(i-1) + X\log i\right) &\leq mXd + \frac{X^2d(d-1)}{2} + \\ &\quad Xd\log d. \\ &\sim mXd + \frac{X^2d^2}{2}. \end{aligned} \tag{37}$$

To compute the roots of the above polynomial, we need to solve d interval problems. For each of these interval problems, the hybrid algorithm we use performs $I(X,d)$ evaluations of the polynomial, where

$$I(X,d) = \frac{1}{2}\log^2 X + \log(10d^2) + O(\log X) \tag{38}$$

$$\sim \frac{1}{2}\log^2 X + 2\log d. \tag{39}$$

where the three terms in Eq. (38) correspond to the number of evaluations performed in each of the three phases of the algorithm : "double exponential sieve", binary search and Newton's method respectively (see Section 2.2 and [BT90]).

Thus the overall complexity of solving all the interval problems for a polynomial of degree d, with m-bit coefficients is asymptotically

$$dI(X,d)(mXd + \frac{X^2d^2}{2}) \sim (\frac{1}{2}\log^2 X + 2\log n)(mXd^2 + \frac{X^2d^3}{2}) \tag{40}$$

However, $I(X,d)$ in Eq. (38) is a worst-case estimate on the number of iterations performed, and results in a poor estimate in practice. The main

reason for the overestimate is that the double-exponential sieve typically executes far fewer than $\frac{1}{2}\log^2 X$ iterations before it identifies a suitable interval for the start of the bisection phase. In fact, if we make the assumption that the desired root is distributed uniformly in the original interval, then it is easy to see that the double-exponential sieve takes only a constant number of iterations. In this case, the bisection and newton phases dominate, and we can estimate the *average* number of iterations as

$$I_{avg}(X,d) \sim \log(10d^2) + \log\left(\lceil\frac{X}{\log(10d^2)}\rceil\right). \tag{41}$$

$$\tag{42}$$

where the second term is the number of iterations performed by Newton's method given the $\log(10d^2)$ bits of accuracy already attained by the bisection phase. We will use this average case estimate rather than the worst-case estimate of Eq. (38) in fitting the analysis to the observed data in Section 5.

Having established the cost of all the interval problems corresponding to a generic polynomial in our model of computation, we now turn to the analysis of the specific interval problems that our algorithm solves. Recall that level l in the tree ($0 \leq l \leq K-1$), has 2^l polynomials, each of degree $d(l) = 2^{K-l} - 1$.

Let μ be the precision required in the computation of the roots. Further, suppose all roots of the original polynomial ($F_0(x)$) lie in the interval $[-2^R, 2^R]$. Then, throughout the algorithm, all evaluations of polynomials are performed at rational points x that are μ-approximations and lie in $[-2^R, 2^R]$. Thus, we can use $R + \mu$ as an upper bound for $X = \|x\|$ for all possible points x at which our algorithm might evaluate a polynomial [7].

Now consider the j^{th} polynomial at level l of the tree. Observing that

$$P^{(l,j)}(x) = P_{j2^{K-l}+1,(j+1)2^{K-l}-1}(x)$$

and using the size estimates derived earlier, we have

$$d(P^{(l,j)}(x)) = d(l) = 2^{K-l} - 1. \tag{43}$$

[7] Recall from Section 2.2 that if $\|F_0(x)\| = m$, then $R \leq m$

$$\begin{aligned} \|P^{(l,j)}(x)\| &\leq (2j2^{K-l} + 2^{K-l} - 1)\beta & (44)\\ &\leq 2^{K-l}(2j+1)\beta, \quad 1 \leq l \leq K-1; \ 0 \leq j < 2^l - 1. & (45) \end{aligned}$$

and

$$\begin{aligned} \|P^{(l,2^l-1)}(x)\| &\sim \|F_{(2^l-1)2^{K-l}}(x)\| & (46)\\ &\leq (2^K - 2^{K-l})\beta, \quad 0 \leq l \leq K-1. & (47) \end{aligned}$$

Equations (40), (43), (44) and (46) give us the following cost estimates for the polynomial evaluations:

For the rightmost nodes in the tree, by summing up over all levels $0 \leq l \leq K-2$, we obtain an asymptotic cost of:

$$\left(\frac{\log^2 X}{2} + 2\log n\right)\left(\frac{4}{21}\beta X n^3 + \frac{4}{7}X^2 n^3\right). \tag{48}$$

And for the non–rightmost nodes, we obtain the asymptotic cost to be

$$\left(\frac{\log^2 X}{2} + 2\log n\right)\left(\frac{4}{3}\beta X n^3 + \frac{1}{6}X^2 n^3\right).$$

The overall complexity of the interval problems is thus

$$O(n^3 X(X+\beta)(\log n + \log^2 X)),$$

where $X = R + \mu$ and $\beta = 2m + 2 + 3\log n$.

5 Empirical Results

In this section, we report the actual running times observed from our implementation of the algorithm. We ran the algorithm on polynomials of degrees $10, 15, 20, \ldots, 70$, and for each degree 3 different polynomials were generated. The results of this section were obtained by running the algorithm on these inputs several times. The input polynomials we used were the characteristic equations of randomly generated symmetric matrices over the integers. Thus, for each degree n, the size m, of the coefficients of the polynomial we generate depended both on n and the sizes of the entries of the random matrix chosen to generate the polynomial. For the data in

n	m(n)	μ				
		4	8	16	24	32
10	2	2.7	3.2	5.7	8.0	11.8
15	4	5.1	8.0	15.5	26.7	41.0
20	7	12.6	19.3	38.7	66.8	102.6
25	9	31.5	45.4	84.2	143.8	217.1
30	12	78.7	107.2	177.1	288.5	423.8
35	14	174.7	222.5	342.2	521.2	744.8
40	17	385.5	458.5	644.5	911.5	1264.2
45	20	799.8	919.3	1210.0	1613.6	2120.2
50	23	1517.0	1690.4	2108.0	2692.1	3412.2
55	26	2860.3	3076.5	3659.0	4446.3	5455.2
60	29	4877.4	5228.0	6019.3	7122.2	8476.1
65	32	7785.8	8248.6	9305.2	10746.5	12506.9
70	36	12930.5	13557.8	14963.7	17270.8	19243.2

Table 2: Single processor Running Times

this section, the matrices generated were random 0–1 matrices, and hence the tables use $m(n)$ to denote the coefficient sizes. Note that it is fairly easy to obtain analytic bounds on $m(n)$, but in what follows we have used the empirical values observed from the actual inputs we generated. We also note that, not unexpectedly, the polynomials we generate all had distinct roots, and hence we can apply the analysis of the previous section to our experimental data.

5.1 Sequential Running Times

Table 2 shows the running times for the algorithm on a single processor for different values of n and μ.

The primary motivation behind the analysis presented in Section 4 was to see how closely our theoretical estimates matched the actual run-time behaviour of the algorithm. As a first step in this direction, we attempted to validate the analytical expressions for the arithmetic complexity of the algorithm. To this end, a algorithm was run (on a single processor) and the actual number of multiplications performed in the different phases were traced. Of course, for the purposes of this section, the analytical estimates we used were much more precise versions of the asymptotic expressions presented in Section 4. Furthermore, we considered all phases of the algorithm

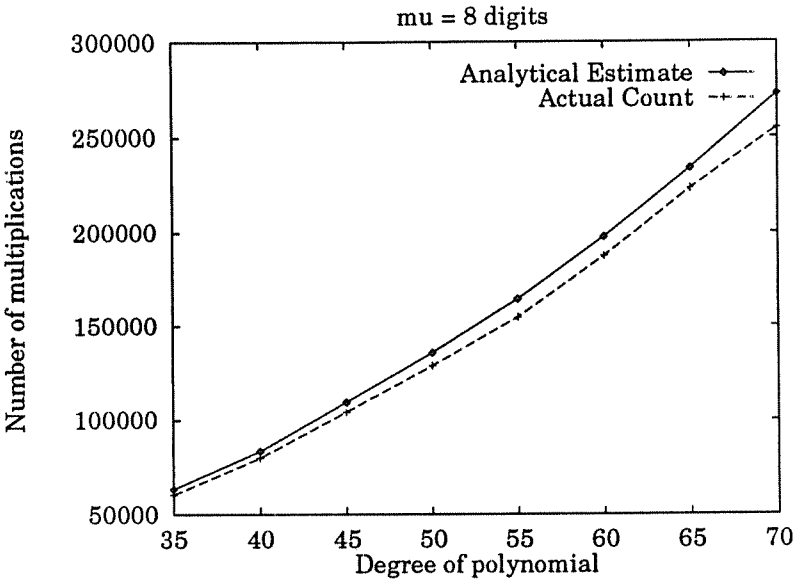

Figure 2: Predicted and Observed Multiplication Counts ($\mu = 8$ digits)

instead of just the dominant phases as was done in Section 4. Figures 2 through 5 plot the predicted and observed number of multiplications for a subset of the inputs that were considered. Note that the predicted counts match the observed counts quite well, especially for larger input parameters.

In predicting the bit-complexity, however, our analytical expressions did not provide as good bounds as the ones above. A typical case is illustrated by comparing Figures 6 and 7. Fig. 6 plots the predicted and observed number of multiplications for a particular phase of the algorithm (in this case, the bisection sub-phase of the Interval Problems.) The excellent fit exhibited here translates to a rather weak upper bound in Fig. 6 when we incorporate the size bounds on the polynomial coefficients and compare the resultant bit-complexity estimate with the actual bit multiplication costs. Thus, in order to be able to predict run-times with accuracy, we would need much tighter bounds on the sizes of polynomial coefficients than those provided by the results of Collins that we used in Section 4. However, these estimates may still be used to provide weak upper bounds on the run-times.

We also compared the one-processor run-times of our implementation with the performance of a sequential root-finding algorithm in the PARI multi-precision package [BBCO91]. Unfortunately, we were unable to run

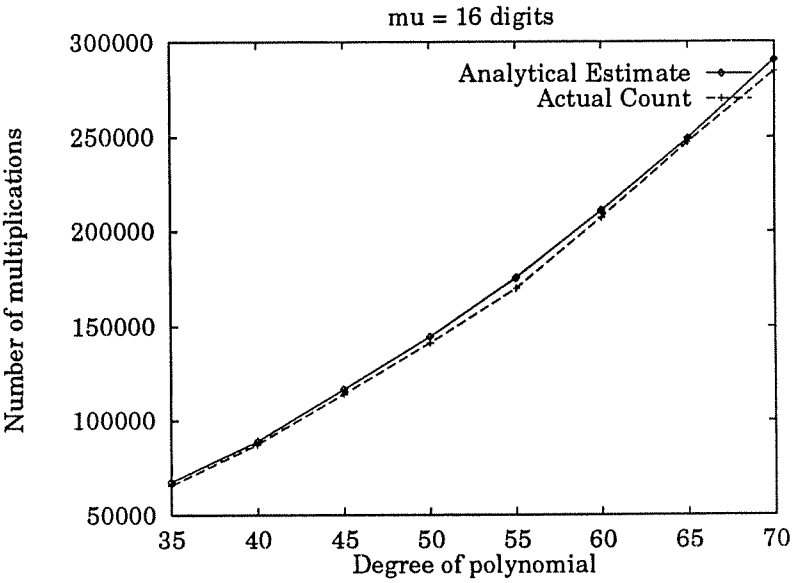

Figure 3: Predicted and Observed Multiplication Counts (μ = 16 digits)

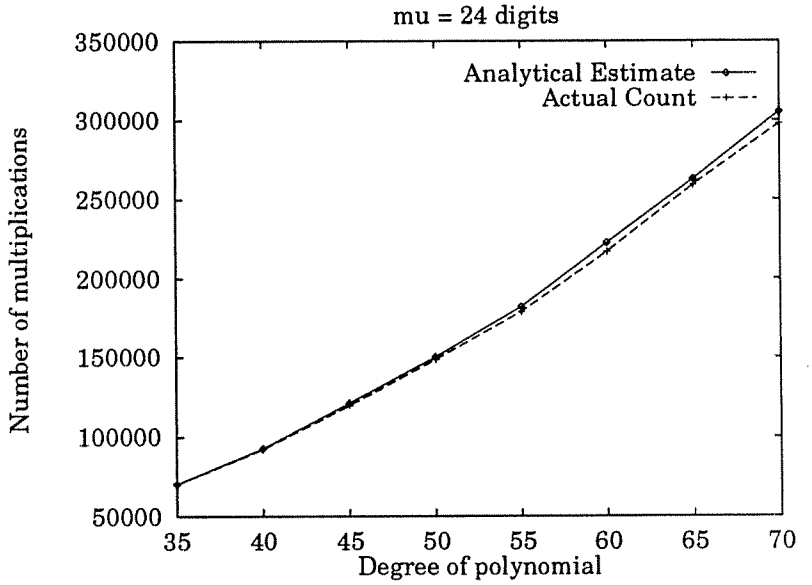

Figure 4: Predicted and Observed Multiplication Counts (μ = 24 digits)

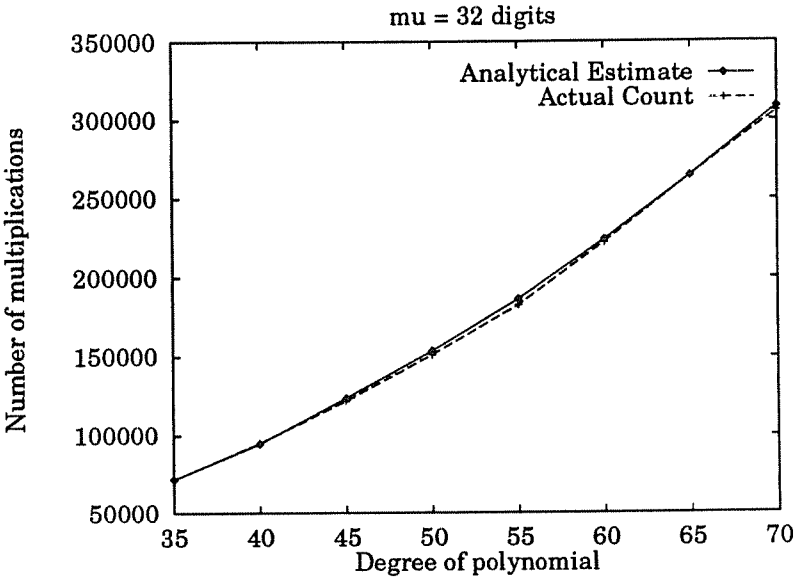

Figure 5: Predicted and Observed Multiplication Counts (μ = 32 digits)

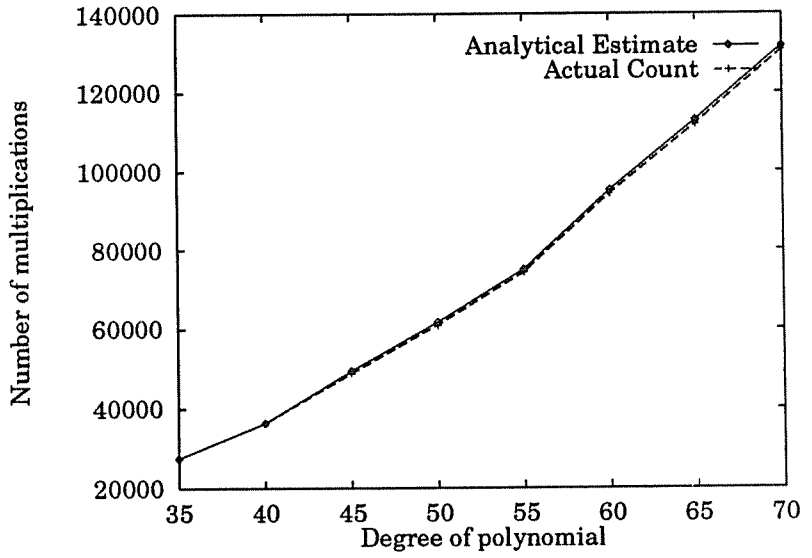

Figure 6: Multiplication Counts for Bisection phase (μ = 32 digits)

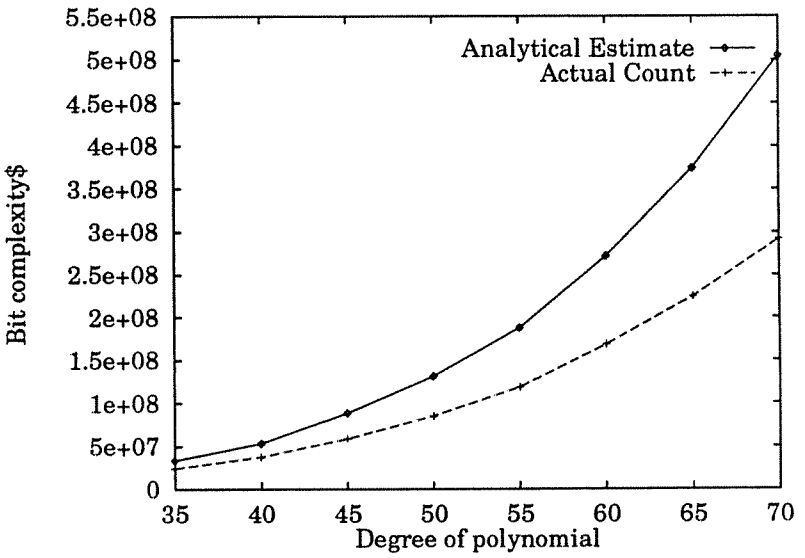

Figure 7: Bit Complexity of Multiplications in Bisection Phase (μ = 32 digits)

the PARI algorithm on polynomials of degree larger than 30. The comparison for degrees less than 30 and μ = 30 digits is shown in Figure 8. For degrees larger than 15, our implementation takes less time to compute the roots. For smaller values of the precision parameter μ, while our algorithm's cost decreased significantly, the PARI algorithm seemed insensitive to this parameter.

5.2 Speedups

Figures 9 through 13 plot the execution times of the algorithm with increasing number of processors for different values of n and μ. Tables 3 through 7 present the same information in the form of speedup figures with respect to the parallel program with one processor. Appendix B presents the complete set of run-times for all input combinations that were considered in our experiments. We observe that the algorithm exhibits good speedups for small numbers of processors. The speedups begin to drop at 16 processors, since in the range of input parameters that we considered, the granularity of the tasks used was not fine enough to keep all the processors busy at all times. Another somewhat anomalous situation is the fact that in going from one

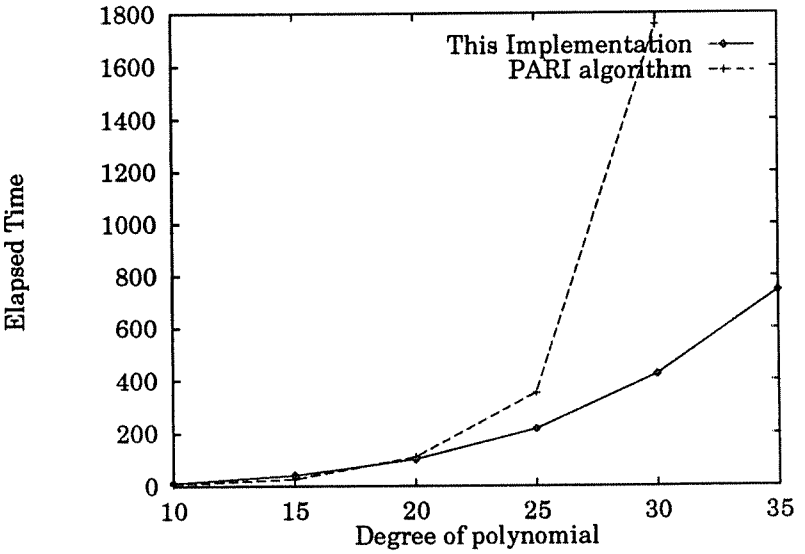

Figure 8: Comparison with PARI algorithm (μ = 30 digits)

to two processors, the speedups attained are often more than two. This phenomenon is probably due to the effect of decreased cache misses when more processors are used.

6 Conclusions

The implementation described in this paper has demonstrated that a practical version of the NC algorithm of Ben-Or and Tiwari is capable of attaining good performance and realizes good speedups on a shared memory multiprocessor. Furthermore, this version of the algorithm seems to quite competitive with other existing root-finding algorithms and does not suffer from problems of stability that characterize many other implementations.

A careful analysis of the algorithm and a comparison of the analytical estimates with the actual run-time characteristics shows that while the behaviour of the algorithm is well understood and predictable with good accuracy, the main bottleneck in attempting to predict the actual execution times is the lack of good analytical estimates on the *sizes* of intermediate quantities computed by the algorithm, expressed in terms of the size of the original input. It would be interesting to see if improved estimates on these

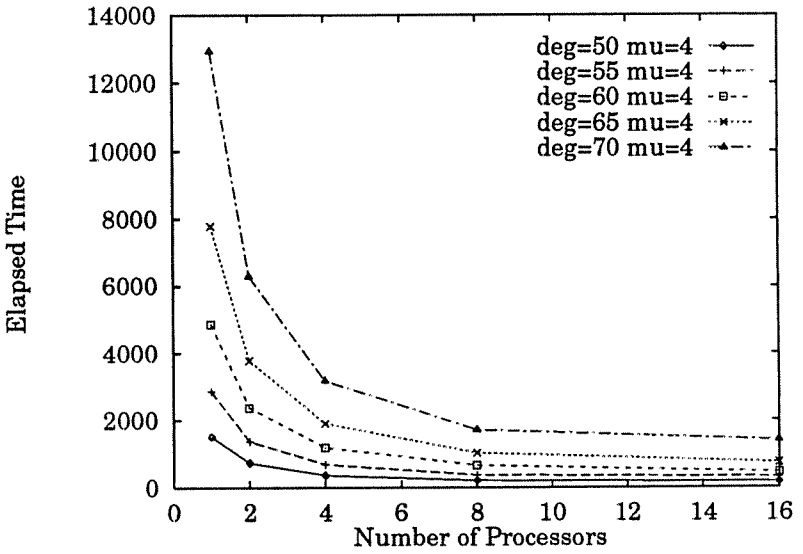

Figure 9: Effect of Number of Processors on Execution Times (μ = 4 digits)

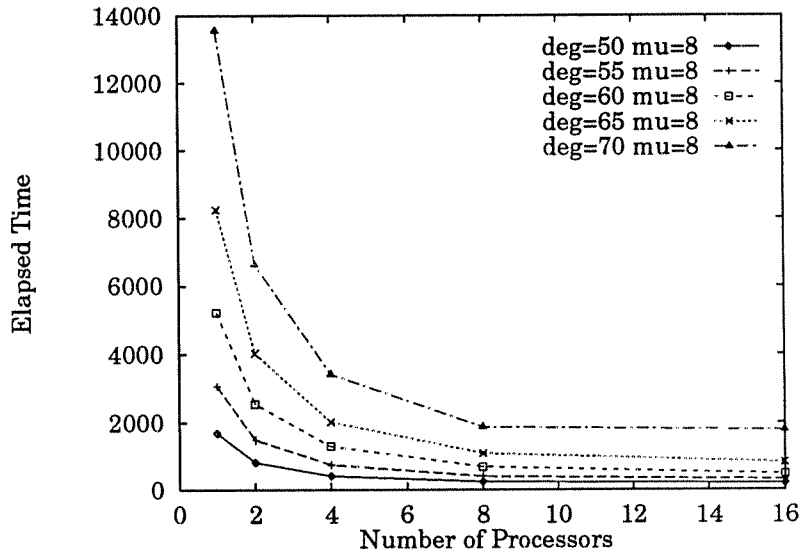

Figure 10: Effect of Number of Processors on Execution Times (μ = 8 digits)

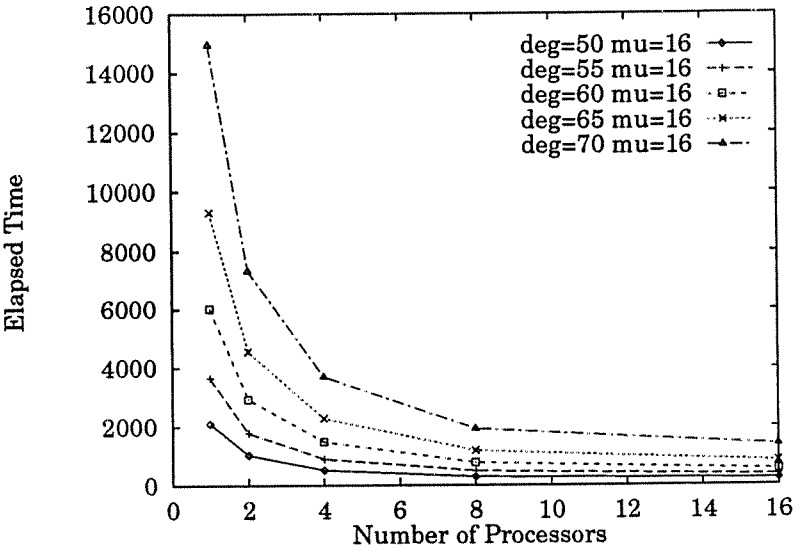

Figure 11: Effect of Number of Processors on Execution Times (μ = 16 digits)

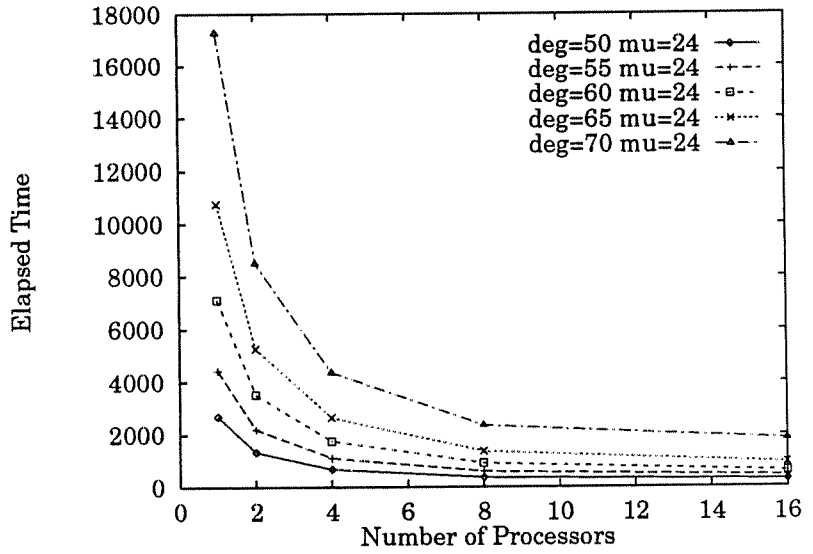

Figure 12: Effect of Number of Processors on Execution Times (μ = 24 digits)

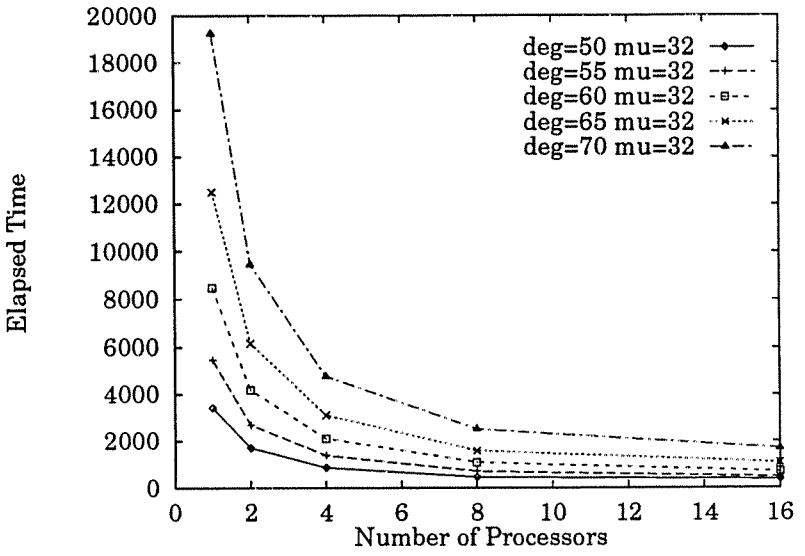

Figure 13: Effect of Number of Processors on Execution Times (μ = 32 digits)

degree	Processors				
	1	2	4	8	16
35	1.0	2.03	3.86	6.15	5.90
40	1.0	2.06	3.98	6.95	7.65
45	1.0	2.06	4.03	7.27	8.94
50	1.0	2.05	4.06	7.08	8.54
55	1.0	2.08	4.12	7.61	8.94
60	1.0	2.06	4.09	7.29	10.61
65	1.0	2.06	4.10	7.55	10.50
70	1.0	2.05	4.08	7.56	9.22

Table 3: Speedups with respect to single processor execution of algorithm ($\mu = 4$ digits)

degree	Processors				
	1	2	4	8	16
35	1.0	2.02	3.81	6.34	6.83
40	1.0	2.04	3.94	7.22	8.77
45	1.0	2.05	4.03	7.28	9.60
50	1.0	2.06	4.06	6.92	8.47
55	1.0	2.06	4.07	7.55	9.77
60	1.0	2.05	4.01	7.55	10.91
65	1.0	2.05	4.08	7.54	10.07
70	1.0	2.04	3.96	7.25	7.63

Table 4: Speedups with respect to single processor execution of algorithm ($\mu = 8$ digits)

degree	Processors				
	1	2	4	8	16
35	1.0	1.99	3.74	6.29	7.92
40	1.0	2.02	3.93	7.15	9.58
45	1.0	2.04	3.99	7.32	10.39
50	1.0	2.03	4.00	7.20	9.25
55	1.0	2.05	4.04	7.44	10.40
60	1.0	2.05	4.05	7.70	11.24
65	1.0	2.04	4.07	7.86	11.23
70	1.0	2.04	4.05	7.74	10.80

Table 5: Speedups with respect to single processor execution of algorithm ($\mu = 16$ digits)

degree	Processors				
	1	2	4	8	16
35	1.0	1.98	3.77	6.55	9.06
40	1.0	2.00	3.92	7.17	10.33
45	1.0	2.02	3.98	7.35	11.10
50	1.0	2.02	3.93	7.16	9.34
55	1.0	2.02	3.99	7.43	10.19
60	1.0	2.02	4.04	7.76	11.79
65	1.0	2.04	4.05	7.84	11.47
70	1.0	2.03	3.96	7.32	9.41

Table 6: Speedups with respect to single processor execution of algorithm ($\mu = 24$ digits)

degree	Processors				
	1	2	4	8	16
35	1.0	1.96	3.77	6.58	9.40
40	1.0	1.99	3.92	7.15	10.43
45	1.0	2.01	3.96	7.37	11.78
50	1.0	1.99	3.93	7.35	9.13
55	1.0	2.03	3.95	7.64	11.49
60	1.0	2.03	4.01	7.74	12.09
65	1.0	2.03	4.03	7.85	11.46
70	1.0	2.04	4.05	7.66	11.35

Table 7: Speedups with respect to single processor execution of algorithm ($\mu = 32$ digits)

quantities can be obtained.

7 Acknowledgements

We would like to thank Anne Condon, who, during the Spring of '91, taught the Parallel Algorithms course at the University of Wisconsin-Madison that served as a starting point for this project. We also thank Alain Kagi, Afroditi Michailidi and T. N. Vijaykumar for their participation in an early version of the implementation.

References

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA., 1974.

[BBCO91] C. Batut, D. Bernardi, H. Cohen, and M. Olivier. User's Guide to PARI-GP, February 1991.

[BT90] M. Ben-Or and P. Tiwari. Simple Algorithms for Approximating All Roots of a Polynomial with Real Roots. *Journal of Complexity*, 6:417–442, 1990.

[Col67] G. E. Collins. Subresultants and Reduced Remainder Sequences. *Journal of the Association for Computing Machinery*, 14:128–142, 1967.

[Hou70] A. S. Householder. *The Numerical Treatment of Single Nonlinear Equation.* McGraw-Hill, New York, 1970.

[Ren87] J. Renegar. On the Worst-Case Arithmetic Complexity of Approximating Zeros of Polynomials. Manuscript, 1987.

A Appendix A : Proof of Theorem 1

Theorem 1 *Let $F_0(x)$ be a polynomial of degree n with integer coefficients such that all its roots are distinct and real. Let the polynomials $P_{i,j}(x)$ be defined as in Eq. (5). Then,*

i) $P_{i,j}(x)$ has integer coefficients, and is of degree $j - i + 1$. For any $j \leq n$, the leading coefficients of all the $P_{i,j}(x)$, $1 \leq i \leq j$, have the same sign.

ii) $P_{i,j}$ has distinct real roots. In addition, if $i < j$, then, for any $1 \leq i \leq k \leq j \leq n$, the polynomials $P_{i,k-1}(x)$ and $P_{k+1,j}(x)$ form an interleaving pair of polynomials for the polynomial $P_{i,j}(x)$.

Proof: Let $\{F_i(x)\}, \{Q_i(x)\}, \{A_i(x)\}$, and $\{B_i(x)\}$ be as defined in Section 2.1. Define the function $sgn(x)$ defined on the reals as follows :

$$sgn(x) = \begin{cases} 0, & if x = 0, \\ \frac{x}{|x|}, & \text{otherwise.} \end{cases}$$

We first prove statement (i) of the Theorem. Collins [Col67] shows that the $F_i(x)$, $A_i(x)$ and the $B_i(x)$ have integer coefficients. The integrality of the coefficients of the $P_{i,j}(x)$ thus follows from their definition (Eq. 5). It is clear that the leading coefficients of all the $F_i(x)$ have the same sign, and that the $Q_i(x)$ are linear polynomials with positive leading coefficients. Let c_i be the leading coefficient of the polynomial F_i for $i \geq 1$ and let c_0 be the the *sign* of the leading coefficient of $F_0(x)$. [8] Let

$$S_i = \begin{pmatrix} 0 & 1 \\ -\frac{c_i^2}{c_{i-1}^2} & \frac{Q_i(x)}{c_{i-1}^2} \end{pmatrix}, \quad 1 \leq i \leq n-1. \tag{49}$$

$$T_{i,j} = c_{i-1}^2 S_j S_{j-1} \ldots S_i, \quad 1 \leq i \leq j \leq n-1. \tag{50}$$

Then, for $1 \leq i \leq j < n$,

$$T_{i,j} = c_{i-1}^2 T_{1,j} T_{1,i-1}^{-1} \tag{51}$$

$$= c_{i-1}^2 \begin{pmatrix} A_j & B_j \\ A_{j+1} & B_{j+1} \end{pmatrix} \begin{pmatrix} A_{i-1} & B_{i-1} \\ A_i & B_i \end{pmatrix}^{-1} \tag{52}$$

[8] Note that this definition of c_0 is a somewhat different from that described in Section 2.1. Defining c_0 in this manner avoids having to treat certain frequently arising boundary cases differently in the proof.

$$= \frac{c_{i-1}^2}{det(T_{1,i-1})} \begin{pmatrix} A_j & B_j \\ A_{j+1} & B_{j+1} \end{pmatrix} \begin{pmatrix} B_i & -B_{i-1} \\ -A_i & A_{i-1} \end{pmatrix} \quad (53)$$

$$= \begin{pmatrix} -P_{i+1,j-1} & P_{i,j-1} \\ -P_{i+1,j} & P_{i,j} \end{pmatrix} \quad (54)$$

From the definition of the $F_i(x)$, it is clear that $P_{i+1,n} = F_i(x)$ is of degree $n - i$.

From the above matrix equations, we see that

$$P_{i,i}(x) = T_{i,i}(2,2) \quad (55)$$

$$= Q_i(x) \quad (56)$$

and, for $1 \leq i < j < n$,

$$P_{i,j}(x) = T_{i,j}(2,2) \quad (57)$$

$$= \frac{c_{i-1}^2}{c_i^2} \left(T_{i+1,j} S_i\right)(2,2) \quad (58)$$

$$= \frac{c_{i-1}^2}{c_i^2} \left[-P_{i+2,j}(x) + P_{i+1,j}(x) \frac{Q_i(x)}{c_{i-1}^2}\right] \quad (59)$$

Given that $Q_i(x)$ is a linear polynomial, it is clear from the above that $P_{i,j}(x)$ is a polynomial of degree $j - i + 1$. Further, since $Q_i(x)$ has a positive leading coefficient, the leading coefficient of $P_{i,j}(x)$ has the same sign as $P_{i+1,j}(x)$, and hence by induction the leading coefficients of all $P_{i,j}(x)$, $1 \leq i \leq j$, are all positive.

We will denote by $\mathcal{P}(i,j,k)$ the predicate that asserts property (ii) in the statement of the Theorem for $P_{i,j}(x)$. We first prove $\mathcal{P}(i,j,k)$ for the case $k = i$.

We will consider the cases $j = n$ and $j < n$ separately. For the former case, we need to show that the polynomial $F_i(x)$ interleaves the polynomial $F_{i-1}(x)$, $1 < i \leq n$. We show this by induction on i. $F_1(x) = F_0'(x)$ clearly interleaves $F_0(x)$ by Rolle's Theorem. For $i > 1$, we have

$$F_i(x) = \frac{Q_{i-1}(x) F_{i-1}(x) - c_{i-1}^2 F_{i-2}(x)}{c_{i-2}} \quad (60)$$

Consider two consecutive real roots x_1 and x_2 of $F_{i-1}(x)$. By the inductive hypothesis, $F_{i-2}(x)$ interleaves $F_{i-1}(x)$ and hence

$$sgn(F_i(x_1)) sgn(F_i(x_2)) = sgn(F_{i-2}(x_1)) sgn(F_{i-2}(x_2)) < 0.$$

Thus, $F_i(x)$ has an odd number of roots in (x_1, x_2). Since the degree of $F_i(x)$ is $n-i$, $F_i(x)$ has *exactly* one root in each interval formed by successive roots of $F_{i-1}(x)$. The interleaving property just proved also guarantees that the polynomial $F_i(x)$ has $n-i$ distinct real roots.

For the case $j < n$, we need to show that $P_{i+1,j}(x)$ interleaves $P_{i,j}(x)$, for $1 \leq i < j$. We show this by downward induction on i. For $i = j-1$, Equation (54) gives us the following relation between $P_{j,j}(x) = Q_j(x)$ and $P_{j-1,j}(x)$.

$$P_{j-1,j}(x) = \frac{Q_j(x)Q_{j-1}(x) - c_j^2 c_{j-2}^2}{c_{j-1}^2} \tag{61}$$

Let $x_1 \leq x_2$ be the two real zeros of the linear polynomials $Q_j(x)$ and $Q_{j-1}(x)$. Since both $Q_j(x)$ and $Q_{j-1}(x)$ have positive leading coefficients and since $(c_j c_{j-2})^2 > 0$, $P_{j-1,j}$ must have two distinct real zeros a and b such that $a < x_1 \leq x_2 < b$.

For $i \leq j-2$, we have

$$P_{i,j}(x) = \frac{c_{i-1}^2}{c_i^2}\left[-P_{i+2,j}(x) + P_{i+1,j}(x)\frac{Q_i(x)}{c_{i-1}^2}\right] \tag{62}$$

Since $P_{i+2,j}(x)$ interleaves $P_{i+1,j}(x)$ by the induction hypothesis, for two successive roots x_1 and x_2 of $P_{i+1,j}(x)$, we have

$$sgn(P_{i,j}(x_1))sgn(P_{i,j}(x_2)) = sgn(P_{i+2,j}(x_1))sgn(P_{i+2,j}(x_2)) < 0$$

and hence $P_{i,j}(x)$ has at least one root in (x_1, x_2).

Let a and b be the leftmost and rightmost roots, respectively, of $P_{i+1,j}(x)$. To complete the proof of the interleaving property, we need to show that $P_{i,j}(x)$ has precisely one root in each of the intervals $(-\infty, a)$ and (b, ∞). From (i), we know that the polynomials $P_{i,j}(x)$ and $P_{i+2,j}(x)$ have degrees of the same parity and their leading coefficients are of the same sign. Thus,

$$sgn(P_{i,j}(-\infty))sgn(P_{i+2,j}(-\infty)) = sgn(P_{i,j}(\infty))sgn(P_{i+2,j}(\infty)) > 0$$

However, from Eq.(62),

$$sgn(P_{i,j}(a))sgn(P_{i+2,j}(a)) = sgn(P_{i,j}(b))sgn(P_{i+2,j}(b)) < 0$$

Since $P_{i+2,j}(x)$ has no roots in the two intervals of interest, $P_{i,j}(x)$ must necessarily have unique real roots in those intervals. By counting arguments,

it follows that $P_{i,j}(x)$ has a unique root in each interval formed by the roots of $P_{i+1,j}(x)$.

That completes the proof of the case $k = i$. Now consider the case where $j \geq k > i$. We have,

$$\begin{aligned} T_{k+1,j} &= c_k^2 S_j S_{j-1} \ldots S_{k+1} && (63) \\ &= c_k^2 S_j S_{j-1} \ldots S_i (S_k S_{k-1} \ldots S_i) && (64) \\ &= c_k^2 T_{i,j} T_{i,k}^{-1} && (65) \end{aligned}$$

Using Equation (54), this expands to

$$\begin{pmatrix} -P_{k+2,j-1} & P_{k+1,j-1} \\ -P_{k+2,j} & P_{k+1,j} \end{pmatrix} = c_i^2 \begin{pmatrix} -P_{i+1,j-1} & P_{i,j-1} \\ -P_{i+1,j} & P_{i,j} \end{pmatrix} \begin{pmatrix} P_{i,k} & -P_{i,k-1} \\ P_{i+1,k} & -P_{i+1,k-1} \end{pmatrix} \quad (66)$$

From the above matrix equation, we get

$$P_{k+1,j}(x) = c_k^2 \left[P_{i+1,j}(x) P_{i,k-1}(x) - P_{i,j}(x) P_{i+1,k-1}(x) \right] \quad (67)$$

Consider two consecutive roots x_1 and x_2 of $P_{i,j}(x)$. By the case already proved, we know that $P_{i+1,j}(x)$ interleaves $P_{i,j}(x)$ and thus changes sign in the interval $[x_1, x_2]$. If x_1 (x_2) is a root of $P_{i,k-1}(x)$, it is also a root of $P_{k+1,j}(x)$, and we can use it as the interleaving root for the interval $[x_1, x_2]$. Otherwise, at least one of $P_{i,k-1}(x)$ and $P_{k+1,j}(x)$ changes sign in $[x_1, x_2]$ and we again have an interleaving root for the interval.

∎

B Appendix B : Empirical Data

n	Number of processors				
	1	2	4	8	16
10	2.7	2.0	2.4	3.3	6.2
15	5.1	3.4	2.9	3.5	6.2
20	12.7	8.3	5.4	4.5	8.7
25	31.5	16.4	9.9	8.0	12.4
30	78.8	40.4	21.6	14.0	18.1
35	174.7	86.2	45.3	28.4	29.6
40	385.5	187.5	96.8	55.5	50.4
45	799.8	388.3	198.3	110.0	89.5
50	1517.1	738.5	373.7	214.3	177.6
55	2860.4	1372.7	694.7	375.9	320.1
60	4877.5	2368.9	1193.1	668.7	459.7
65	7785.8	3782.0	1901.2	1031.2	741.3
70	12930.5	6297.1	3171.9	1711.0	1402.2

Table 8: Running Times for $\mu = 4$ digits

n	Number of processors				
	1	2	4	8	16
10	3.3	2.4	2.3	4.5	4.9
15	8.1	6.0	4.0	3.8	7.9
20	19.4	10.8	6.9	6.2	10.0
25	45.5	23.5	13.9	9.6	10.2
30	107.3	54.7	30.0	20.6	17.7
35	222.5	110.2	58.4	35.1	32.6
40	458.6	225.0	116.4	63.5	52.3
45	919.4	448.5	228.4	126.3	95.8
50	1690.4	822.5	416.8	244.2	199.5
55	3076.6	1496.8	755.1	407.6	315.0
60	5228.0	2554.8	1303.5	692.9	479.4
65	8248.7	4019.1	2020.7	1093.5	819.3
70	13557.8	6644.6	3425.4	1871.3	1776.3

Table 9: Running Times for $\mu = 8$ digits

n	Number of processors				
	1	2	4	8	16
10	5.7	3.4	3.3	3.9	5.0
15	15.6	9.2	6.0	5.0	6.4
20	38.8	20.8	14.2	10.5	11.6
25	84.3	43.4	23.8	15.8	15.8
30	177.2	91.3	50.0	30.1	28.3
35	342.2	172.3	91.4	54.4	43.2
40	644.5	319.7	163.9	90.2	67.3
45	1210.0	594.0	302.9	165.2	116.5
50	2108.1	1036.7	526.6	292.9	227.9
55	3659.1	1786.7	904.8	491.9	351.9
60	6019.4	2942.6	1487.6	781.4	535.5
65	9305.2	4552.5	2284.2	1184.3	828.4
70	14963.7	7328.1	3692.5	1933.9	1385.2

Table 10: Running Times for $\mu = 16$ digits

n	Number of processors				
	1	2	4	8	16
10	8.0	4.9	3.9	5.0	6.7
15	26.7	14.8	8.9	6.6	7.3
20	66.9	35.1	19.9	14.5	14.7
25	143.8	74.9	41.2	25.1	20.8
30	288.5	146.5	79.0	43.8	32.8
35	521.2	263.4	138.4	79.6	57.5
40	911.5	456.8	232.8	127.1	88.2
45	1613.6	797.6	405.2	219.6	145.4
50	2692.1	1335.2	684.6	376.2	288.1
55	4446.4	2198.0	1113.5	598.1	436.4
60	7122.2	3517.2	1762.5	918.3	604.0
65	10746.5	5273.3	2653.1	1370.4	936.6
70	17270.8	8505.1	4364.1	2357.9	1835.9

Table 11: Running Times for $\mu = 24$ digits

n	Number of processors				
	1	2	4	8	16
10	11.8	7.2	5.2	4.8	7.8
15	41.0	22.3	12.9	9.3	9.0
20	102.6	53.6	30.1	22.0	17.2
25	217.2	112.4	61.3	38.1	26.1
30	423.9	215.0	115.3	62.9	44.1
35	744.8	380.1	197.4	113.2	79.2
40	1264.2	634.0	322.4	176.7	121.2
45	2120.3	1056.4	536.0	287.6	180.0
50	3412.3	1711.9	867.4	464.3	373.8
55	5455.2	2691.9	1382.5	714.4	474.6
60	8476.1	4181.9	2111.2	1095.3	700.8
65	12506.9	6161.1	3100.3	1592.7	1091.6
70	19243.2	9455.9	4754.5	2511.8	1695.5

Table 12: Running Times for $\mu = 32$ digits