Intern. J. Computer Math., Vol. 32, pp. 217–231
Reprints available directly from the publisher
Photocopying permitted by license only

PARALLEL RATIONAL INTERPOLATION

ÖMER EĞECİOĞLU*

Department of Computer Science, University of California Santa Barbara, Santa Barbara, CA 93106, USA

ÇETIN K. KOÇ†

Department of Electrical Engineering, University of Houston, Houston, TX 77204, USA

(*Received 3 April 1989*)

A fast parallel algorithm for rational interpolation based on orthogonal polynomials, which is suitable for both shared-memory and message-passing multiprocessor systems is proposed. In the shared-memory case with $N+1$ identical processors, the algorithm requires $O(N \log N)$ parallel arithmetic steps to construct all rational interpolants at once, where $N+1$ is the number of data points. Extensions to message-passing multiprocessor systems such as the hypercube are also discussed. The hypercube version of the algorithm requires $O(N \log N)$ inter-processor communication overhead. Thus in effect, the algorithm constructs each rational interpolant using $O(\log N)$ parallel arithmetic and $O(\log N)$ communication steps.

KEY WORDS: Rational interpolation, orthogonal polynomials, parallel algorithms.

C.R. CATEGORIES: G.1.0, G.1.2, F.2.1.

1. INTRODUCTION

Given a set of $N+1=m+n+1$ distinct points x_i and function values $f_i=f(x_i)$ for $0 \leqq i \leqq N$, the *rational interpolation* problem is the task of determining polynomials $p_m(x)$ and $q_n(x)$ of degrees m and n respectively, such that the rational function $r_{m,n}(x)=p_m(x)/q_n(x)$ takes on the value f_i at the point x_i for $0 \leqq i \leqq N$. It is well known that unlike polynomial interpolation, for a given m and n, the rational interpolant $r_{m,n}(x)$ may fail to satisfy these requirements for some pairs (x_i, f_j). For such values of m and n, these pairs are the *unattainable* points. In this case the given data set is called *degenerate*. Roughly speaking, the existence of an unattainable point (x_j, f_j) means that x_j is a common zero of both $p_m(x)$ and $q_n(x)$.

A direct sequential algorithm for rational interpolation solves a set of linear equations

$$p_m(x_i) - f_i q_n(x_i) = 0 \qquad 0 \leqq i \leqq N \tag{1.1}$$

*Supported in part by NSF Grant No. DCR-8603722.

†Supported in part by Lawrence Livermore National Laboratory, Contract No. LLNL-7526225 and the National Science Foundation (and AFOSR) under Grant No. ECS84-06152.

to determine the coefficients of $p_m(x)$ and $q_n(x)$, the oldest such method being an elegant elimination process due to Jacobi [9, 12]. Jacobi's method yields exact determinantal formulae for the coefficients by making use of elementary properties of divided differences and solving a related Hankel system. The sequential time required for this can be shown to be $O(N^2)$ [3]. A recent algorithm by Schneider and Werner [15] utilizes the barycentric representation of the rational interpolant. The solution of the resulting linear system by Gaussian elimination is an $O(N^3)$ sequential algorithm in the general case.

Alongside the direct algorithms there exist an array of iterative algorithms for rational interpolation as well, the reader is referred to [6] for an exposition of these.

In this paper we propose a parallel algorithm for rational interpolation, suitable both for shared-memory and message-passing multiprocessors. Our approach is based on fast direct algorithms for rational interpolation via orthogonal polynomials. The reader is referred to [3] for further discussion and the proofs for the sequential version of rational interpolation via orthogonal polynomial algorithm that we describe next.

2. A SUMMARY OF SEQUENTIAL INTERPOLATION VIA ORTHOGONAL POLYNOMIALS

Assuming $f_i \neq 0$ for $i=0, 1, \ldots, N$, we set

$$h_s = \sum_{i=0}^{N} \frac{f_i}{w_i} x_i^s, \qquad 0 \leqq s \leqq 2n-1,$$

$$h'_s = \sum_{i=0}^{N} \frac{f_i^{-1}}{w_i} x_i^s, \qquad 0 \leqq s \leqq 2m-1,$$

where

$$w_i = \prod_{\substack{j=0 \\ j \neq i}}^{N} (x_i - x_j), \qquad 0 \leqq i \leqq N.$$

Next, define the Hankel matrices

$$\mathbf{H}_j = \begin{bmatrix} h_0 & h_1 & \ldots & h_j \\ h_1 & h_2 & \ldots & h_{j+1} \\ \vdots & \vdots & \ldots & \vdots \\ h_j & h_{j+1} & \ldots & h_{2j} \end{bmatrix} \qquad \mathbf{H}'_j = \begin{bmatrix} h'_0 & h'_1 & \ldots & h'_j \\ h'_1 & h'_2 & \ldots & h'_{j+1} \\ \vdots & \vdots & \ldots & \vdots \\ h'_j & h'_{j+1} & \ldots & h'_{2j} \end{bmatrix}$$

for $0 \leqq j \leqq n$, and $0 \leqq j \leqq m$, respectively.

Finally, define two symmetric discrete bilinear forms $\langle \cdot, \cdot \rangle_q$ and $\langle \cdot, \cdot \rangle_p$ on polynomials of degree $\leqq n$ and $\leqq m$, respectively, by setting

$$\langle u(x), v(x) \rangle_q := \sum_{i=0}^{N} \frac{f_i}{w_i} u(x_i) v(x_i), \tag{2.1}$$

$$\langle u(x), v(x) \rangle_p := \sum_{i=0}^{N} \frac{f_i^{-1}}{w_i} u(x_i) v(x_i). \tag{2.2}$$

It can be proved [1, 2, 3] that the denominator polynomials $q_0(x), q_1(x), \ldots, q_n(x)$ form an orthogonal family with respect to the bilinear form $\langle \cdot, \cdot \rangle_q$ provided $\det(\mathbf{H}_j) \neq 0$ for $j = 0, 1, \ldots, n$. In this case $\langle \cdot, \cdot \rangle_q$ is non-degenerate, and the polynomials $q_j(x)$ can be generated by the classical three-term recursion formula [4, 10, 17]. Note that since we have one degree of freedom in the choice of the coefficients of $q_n(x)$, in any rational interpolation problem $q_n(x)$ can be taken to be monic. Then

$$\begin{aligned} q_0(x) &= 1, \quad q_{-1}(x) = 0 \\ q_{j+1}(x) &= (x - \alpha_j) q_j(x) - \beta_j q_{j-1}(x), \qquad j = 0, 1, \ldots \end{aligned} \tag{2.3}$$

where α_j and β_j are constants determined as

$$\alpha_j = \frac{\langle x q_j(x), q_j(x) \rangle_q}{\langle q_j(x), q_j(x) \rangle_q}, \qquad \beta_j = \frac{\langle q_j(x), q_j(x) \rangle_q}{\langle q_{j-1}(x), q_{j-1}(x) \rangle_q}. \tag{2.4}$$

Note that the relation (2.3) generates monic orthogonal polynomials as required. Furthermore, the numerator polynomials $p_0(x), p_1(x), \ldots, p_m(x)$ form an orthogonal family with respect to the bilinear form $\langle \cdot, \cdot \rangle_p$ provided $\det(\mathbf{H}'_j) \neq 0$ for $j = 0, 1, \ldots, m$. Since $p_m(x)$ and $q_n(x)$ cannot be forced to be monic at the same time, the monic polynomial $t_m(x)$ produced by the corresponding three-term recursion for $\langle \cdot, \cdot \rangle_p$ can be scaled to construct the numerator polynomial $p_m(x)$ by setting

$$p_m(x) = A_{mm} t_m(x), \tag{2.5}$$

where

$$A_{mm} = \frac{f_i q_n(x_i)}{t_m(x_i)}$$

for any attainable point (x_i, f_i).

For a fixed m and n, if $\det(\mathbf{H}_j) = 0$ for some j, $0 \leqq j \leqq n$, then the bilinear form $\langle \cdot, \cdot \rangle_q$ is degenerate and it is not possible to generate $q_n(x)$ via the recursion (2.3). A similar condition holds for the numerator polynomials. However, these cases can be treated [3] by employing a variant of Jacobi's algorithm together with a

fast algorithm to solve the resulting Hankel systems due to Rissanen [11], and the Newton interpolation algorithm [7]. Thus here we shall assume the complete non-degeneracy of both of the bilinear forms defined in (2.1) and (2.2), or equivalently $\det(\mathbf{H}_j)$, $\det(\mathbf{H}'_j) \neq 0$ for $j=0,1,\ldots,N$.

For a family of polynomials $t_j(x)$ of degree j, $j=0,1,\ldots,N$, orthogonal with respect to the bilinear form $\langle \cdot,\cdot \rangle_q$, let $T=[T_{ji}]$ denote the $(N+1)\times(N+1)$ matrix where

$$T_{ji}=t_j(x_i), \qquad 0\leqq i,j\leqq N.$$

In other words, the jth row of T consists of the vector of values of the jth orthogonal polynomial $t_j(x)$ at the nodes $x_0, x_1, \ldots, x_N$. The following procedure generates this matrix T together with the vectors of coefficients $\alpha=(\alpha_0,\alpha_1,\ldots,\alpha_N)$, $\beta=(\beta_0,\beta_1,\ldots,\beta_N)$ defined by (2.4) via the three-term recursion (TTR) in (2.3).

PROCEDURE TTR

INPUT: (x_i, f_i) for $0\leqq i\leqq N$

OUTPUT: T_{ji} for $0\leqq i,j\leqq N$, the vectors α and β

Step 1. Compute w_i and c_i for $0\leqq i\leqq N$ using

$$w_i=\prod_{\substack{j=0\\ j\neq i}}^{N}(x_i-x_j) \quad \text{and} \quad c_i=\frac{f_i}{w_i}.$$

Step 2. Set $T_{0i}=1$ for $0\leqq i\leqq N$, $\beta_0=0$, and compute

$$\gamma_0=\sum_{i=0}^{N} c_i x_i, \quad \theta_0=\sum_{i=0}^{N} c_i, \quad \text{and} \quad \alpha_0=\frac{\gamma_0}{\theta_0}.$$

Step 3. Set $T_{1i}=x_i-\alpha_0$ for $0\leqq i\leqq N$, and compute

$$\gamma_1=\sum_{i=0}^{N} c_i x_i T_{1i}^2, \quad \theta_1=\sum_{i=0}^{N} c_i T_{1i}^2, \quad \text{and} \quad \alpha_1=\frac{\gamma_1}{\theta_1}, \quad \beta_1=\frac{\theta_1}{\theta_0}.$$

Step 4. For $1\leqq j\leqq N-1$ compute

$$T_{j+1,i}=(x_i-\alpha_j)T_{ji}-\beta_j T_{j-1,i} \qquad 0\leqq i\leqq N$$

$$\gamma_{j+1}=\sum_{i=0}^{N} c_i x_i T_{j+1,i}^2, \quad \theta_{j+1}=\sum_{i=0}^{N} c_i T_{j+1,i}^2$$

$$\alpha_{j+1}=\frac{\gamma_{j+1}}{\theta_{j+1}}, \quad \beta_{j+1}=\frac{\theta_{j+1}}{\theta_j}.$$

END TTR

We will denote by

$$[T, \alpha, \beta] := \mathrm{TTR}(x_0, \ldots, x_N, f_0, \ldots, f_N)$$

the $(N+1)\times(N+1)$ matrix of values T and the vectors α and β produced by TTR from the input data (x_i, f_i), $0 \leqq i \leqq N$. Thus, from the definition of the bilinear form $\langle \cdot, \cdot \rangle_q$ it follows that

$$[Q, \alpha, \beta] := \mathrm{TTR}(x_0, \ldots, x_N, f_0, \ldots, f_N)$$

where α and β are defined by (2.4), and $Q = [Q_{ji}] = [q_j(x_i)]$. Similarly,

$$[P, \alpha', \beta'] := \mathrm{TTR}(x_0, \ldots, x_N, f_0^{-1}, \ldots, f_N^{-1})$$

where

$$\alpha'_j = \frac{\langle x t_j(x), t_j(x) \rangle_p}{\langle t_j(x), t_j(x) \rangle_p} \quad \beta'_j = \frac{\langle t_j(x), t_j(x) \rangle_p}{\langle t_{j-1}(x), t_{j-1}(x) \rangle_p}, \tag{2.6}$$

and $P = [P_{ji}] = [A_{jj}^{-1} p_j(x_i)]$ is the matrix of values of the numerator polynomials, up to the scaling factor A_{jj} given in (2.5) for the jth row.

Note that along with the values of the numerator and the denominator polynomials, it is possible to compute their coefficients recursively using the three-term recurrence formula in (2.3). This can be done by applying the three-term recursion directly to the coefficients of the polynomials produced. More precisely, let $B = [B_{jk}]$ be the $(N+1)\times(N+1)$ matrix in which the jth row consists of the coefficients of the polynomial $q_j(x)$, i.e.

$$q_j(x) = \sum_{k=0}^{j} B_{jk} x^k. \tag{2.7}$$

Then B is a lower triangular matrix with unit diagonal whose elements satisfy the recursion

$$B_{j+1,k} = B_{j,k-1} - \alpha_j B_{jk} - \beta_j B_{j-1,k} \qquad 0 \leqq k \leqq j \leqq N \tag{2.8}$$

induced by (2.3). In (2.8) we take

$$\begin{aligned} B_{j,-1} &= 0 \quad \text{for} \quad 0 \leqq j \leqq N, \\ B_{-1,k} &= 0 \quad \text{for} \quad 0 \leqq k \leqq N. \end{aligned} \tag{2.9}$$

Using (2.8) we can generate the coefficients of the polynomials $q_j(x)$ for $0 \leqq j \leqq N$.

The values of the polynomials $q_j(x)$ at the node points x_i (i.e. Q_{ji}) are computed at each step to calculate α_j and β_j, but this also helps to locate the unattainable points if the point set happens to be degenerate for the particular values of m and n.

Furthermore, for $0 \leqq j \leqq N$ the coefficients of the polynomials $p_j(x)$ can be computed in a similar fashion. We define $A=[A_{jk}]$ to be the $(N+1)\times(N+1)$ matrix where the jth row of A contains the coefficients of the polynomial $p_j(x)$

$$p_j(x)=\sum_{k=0}^{j} A_{jk}x^k, \tag{2.10}$$

similar to the definition of the matrix B.

The following algorithm ALGORITHM RIVOP (Rational Interpolation Via Orthogonal Polynomials) first generates the coefficients of the polynomials $q_j(x)$ for all $0 \leqq j \leqq N$ by using PROCEDURE TTR. The algorithm then proceeds to generate the coefficients of the polynomials $p_j(x)$ for $0 \leqq j \leqq N$ by applying the same technique to the data (x_i, f_i^{-1}) for $0 \leqq i \leqq N$.

ALGORITHM RIVOP

INPUT: (x_i, f_i) for $0 \leqq i \leqq N$

OUTPUT: Coefficients of $p_m(x)$ and $q_n(x)$ for $0 \leqq n \leqq N$ and $m=N-n$

Step 1. $[Q, \alpha, \beta]:=\mathrm{TTR}(x_0, \ldots, x_N, f_0, \ldots, f_N)$

Step 2. Set $B_{jj}=1$ for $0 \leqq j \leqq N$ and $B_{jk}=0$ for $0 \leqq j<k \leqq N$. Set $B_{10}=-\alpha_0$.

For all $0 \leqq k<j \leqq N-1$ compute

$$B_{j+1,k}=B_{j,k-1}-\alpha_j B_{jk}-\beta_j B_{j-1,k}.$$

Step 3. Compute f_i^{-1} for $0 \leqq i \leqq N$ and

$$[P, \alpha', \beta']:=\mathrm{TTR}(x_0, \ldots, x_N, f_0^{-1}, \ldots, f_N^{-1}).$$

Step 4. Set $A_{jj}=1$ for $0 \leqq j \leqq N$ and $A_{jk}=0$ for $0 \leqq j<k \leqq N$. Set $A_{10}=-\alpha'_0$.

For all $0 \leqq k<j \leqq N-1$ compute

$$A_{j+1,k}=A_{j,k-1}-\alpha'_j A_{jk}-\beta'_{j-1,k}.$$

Step 5. Update the coefficients of $p_j(x)$ according to (2.5)

$$A_{jk}=f_0\frac{Q_{N-j,0}}{P_{j0}}A_{jk} \qquad 0 \leqq k \leqq j \leqq N.$$

END RIVOP

At the end of ALGORITHM RIVOP, the coefficients of the polynomials $p_m(x)$ and $q_n(x)$ are A_{mk} and B_{nk} respectively for $0 \leqq k \leqq N$. Note that $A_{mk}=0$ for $k>m$ and $B_{nk}=0$ for $k>n$.

It can be shown [3] that the number of arithmetic steps required by ALGORITHM RIVOP to compute all of the rational interpolants $r_{m,n}(x)$ for $0 \leqq n \leqq N$ and $m=N-n$ is $O(N^2)$.

3. PARALLEL COMPUTATION OF RATIONAL INTERPOLANTS ON A SHARED-MEMORY MULTIPROCESSOR

In this section we assume that we have a shared-memory multiprocessor with $N+1$ identical processors. Each processor reads its operands from the memory, performs floating point operations and writes the result back to the prescribed memory location. We also assume that read, write and wait times are negligible with respect to floating point operations and a floating point operation takes 1 unit time. The algorithms we now describe are suitable for a shared-memory PRAM model [5].

First we observe that PROCEDURE TTR can be parallelized.

LEMMA 1 *Given $N+1$ identical processors, PROCEDURE TTR can be parallelized to compute the matrix T and the vectors α and β using $O(N \log N)$ parallel arithmetic operations.*

Proof In Step 1 of PROCEDURE TTR we assign the ith processor to the computation of w_i and then to the computation of c_i. The computation of w_i takes $2N-1$ steps and c_i is computed in a single step. Since $N+1$ processors operate simultaneously, Step 1 can be completed in $2N$ parallel steps.

For the computation of α_0 in Step 2, we first let the ith processor compute $c_i x_i$ in a single step. The computation of γ_0 involves the summation of these $N+1$ scalars that are already available. This can be done in $\log(N+1)$ parallel using $N+1$ processors by implementing a binary tree algorithm. After this, the computation of θ_0 can be done in $\log(N+1)$ parallel steps. Then we perform a single division to compute α_0. Thus Step 2 can be performed by using $2\log(N+1)+2$ parallel arithmetic operations.

In Step 3, processor i computes the quantities T_{1i}, $c_i T_{1i}^2$ and $c_i x_i T_{1i}^2$ for $0 \leqq i \leqq N$, using a total of four parallel steps. Next, γ_1 is computed by performing a binary tree addition algorithm using $\log(N+1)$ parallel operations. Similarly, the computation of θ_1 can be completed in $\log(N+1)$ parallel arithmetic steps. Then two of the processors compute α_1 and β_1 in a single step. Thus Step 3 requires a total of $2\log(N+1)+5$ arithmetic steps.

In Step 4, $T_{j+1,i}$ is computed by the ith processor using four steps for a fixed j. Then α_{j+1} and β_{j+1} are computed in $2\log(N+1)+4$ steps, again by resorting to a binary tree parallel addition algorithm. Since this needs to be done for $j=1,2,\ldots,N-1$, the total number of parallel arithmetic steps required by Step 4 of PROCEDURE TTR is $(N-1)[2\log(N+1)+8]$.

Thus by summing the number of parallel arithmetic operations performed at each step, we find that the total number of parallel arithmetic operations required

by the parallel version of PROCEDURE TTR to compute the matrix T and the vectors α and β is

$$2N\log(N+1)+2N+8N+2\log(N+1)-1=O(N\log N). \quad \square \tag{3.1}$$

Now using the parallel version of PROCEDURE TTR, we show that ALGORITHM RIVOP can be parallelized to compute all rational interpolants in $O(N\log N)$ parallel arithmetic steps.

THEOREM 1 *Given $N+1$ identical processors, ALGORITHM RIVOP can be parallelized to compute all rational interpolants $r_{m,n}(x)$ for $0\leqq n\leqq N$ and $m=N-n$, using $O(N\log N)$ parallel arithmetic operations.*

Proof Step 1 of ALGORITHM RIVOP is a call to the parallel version of PROCEDURE TTR with the data set (x_i, f_i) for $0\leqq i\leqq N$. By Lemma 1, the resulting triple $[Q,\alpha,\beta]:=\mathrm{TTR}(x_0,\ldots,x_N,f_0,\ldots,f_N)$ is computed using $N+1$ processors in

$$2N\log(N+1)+10N+2\log(N+1)-1$$

parallel steps.

In Step 2, for a fixed $j\geqq 1$ the kth processor computes $B_{j+1,k}$ for $k=0,1,\ldots,j$ in four parallel steps. For all $j=1,2,\ldots,N-1$, Step 2 takes $4(N-1)$ parallel arithmetic operations. Note that the same implementation and arithmetic operation count applies also to the execution of Step 4.

In Step 3, the ith processor computes f_i^{-1}, and then a call to PROCEDURE TTR is made with the data set (x_i, f_i^{-1}) for $0\leqq i\leqq N$.

In Step 5, the jth processor computes $f_0Q_{N-j,0}/P_{j0}$ in two steps. The lower triangular half of the matrix A is then updated in $N+1$ steps using all processors. Thus Step 5 takes exactly $N+3$ parallel arithmetic steps. Hence the number of parallel arithmetic operations for each step of the parallel version of ALGORITHM RIVOP can be summarized as follows:

Step 1: $2N\log(N+1)+10N+2\log(N+1)-1$;

Step 2: $4(N-1)$;

Step 3: $1+2N\log(N+1)+10N+2\log(N+1)-1$;

Step 4: $4(N-1)$;

Step 5: $N+3$.

Thus we conclude that the parallel version of ALGORITHM RIVOP computes all rational interpolants using

$$4N\log(N+1)+29N+4\log(N+1)-6=O(N\log N) \tag{3.2}$$

parallel arithmetic steps with $N+1$ identical processors. $\square$

4. PARALLEL COMPUTATION OF RATIONAL INTERPOLANTS ON A HYPERCUBE MULTIPROCESSOR

Although the proposed parallel rational interpolation algorithm is suitable for implementation on a shared-memory multiprocessor system, it is interesting to examine its implementation on a message-passing architecture. We choose the hypercube multiprocessor since this architecture and interconnection network have been studied extensively and there are commercially available versions. This architecture is based on a binary d-cube consisting of 2^d identical nodes labeled $0, 1, \ldots, 2^d-1$. The *node_id* of the node i is the binary expansion of i. Two nodes in the hypercube topology are connected if and only if their node_id's differ by one bit [13, 16].

It is well known that in a message-passing architecture, communication requirements of the algorithm should be made minimal for an efficient implementation [8, 14]. We will show that by making use of the properties of hypercube spanning trees, binary addition trees, and Gray code, we can efficiently implement PROCEDURE TTR, and thus ALGORITHM RIVOP on the hypercube architecture.

Following the widely accepted nomenclature, we assume that in the hypercube multiprocessor a node can send a data item to one of its d neighboring nodes by issuing a command

SEND (X, *target_node*)

where X is the data and *target_node* is the node to which the data is to be sent. The target node receives the data by executing

RECEIVE (X).

It is assumed here that a short message (e.g. a floating-point number) takes one unit time to arrive. If the target node is not a neighboring node then the message will be forwarded by the nodes along a path between the sender and the receiver. In a hypercube with 2^d nodes the longest path length is d.

To implement PROCEDURE TTR on a hypercube with $2^d=N+1$ nodes we use the following two procedures: First, PROCEDURE BROADCAST (X, i) broadcasts data item X, which is initially located in node i, to all nodes in the system. This algorithm makes use of a hypercube spanning tree rooted at node i, which is illustrated in Figure 1 for $d=3$ and $i=7$. In this procedure, $\oplus$ denotes modulo 2 bitwise addition.

The second routine, PROCEDURE ACCUMULATE (X_i), computes the sum

$$S=\sum_{i=0}^{2^d-1} X_i$$

where X_i is a data item initially located in node i for $i=0, 1, \ldots, 2^d-1$. After the execution of PROCEDURE ACCUMULATE (X_i), the sum S can be found in node 2^d-1. The routine $bit(k, j)$ in this procedure returns the kth bit of the binary number j. For example $bit(1,(1101)_2)=1$ and $bit(2,(1101)_2)=0$. Figure 2 illustrates the communication links utilized in PROCEDURE ACCUMULATE for $d=3$.

PROCEDURE BROADCAST (X, q)

INPUT: X at node q

OUTPUT: X at all nodes

$j = node\_id$

FOR $k = 0$ TO $d-1$ DO

BEGIN

IF $j \oplus q \leqq 2^k - 1$ THEN

SEND $(X, j \oplus 2^k)$

IF $2^k \leqq j \oplus q \leqq 2^{k+1} - 1$ THEN

END FOR

END BROADCAST

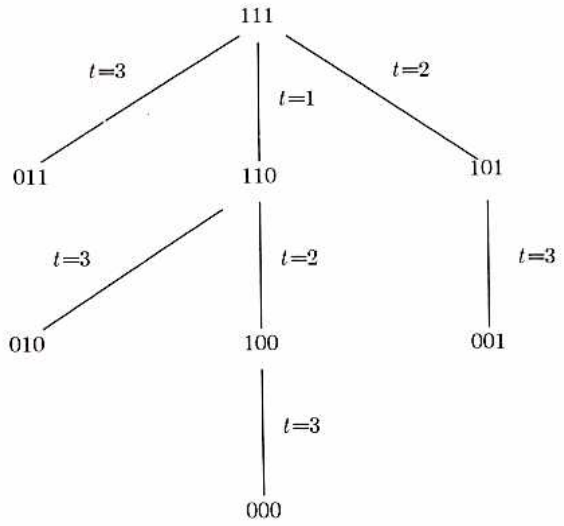

Figure 1 A hypercube spanning tree for broadcasting a data item from node $(111)_2$.

An inspection shows that PROCEDURE BROADCAST used d routing steps to complete. Similarly PROCEDURE ACCUMULATE takes d routing and d floating-point addition steps. These routines allow us to implement PROCEDURE TTR on a hypercube with $O(N \log N)$ routing and $O(N \log N)$ floating-point operation steps. In the following implementation, we assume that $N+1 = 2^d$ and for $0 \leqq i \leqq N$, processor i initially contains $x_0, x_1, \ldots, x_N$ together with f_i.

PROCEDURE ACCUMULATE (X_i)

INPUT: X_i at node i for $0 \leqq i \leqq 2^d - 1$

OUTPUT: $\sum_{i=0}^{2^d-1} X_i$ at node $2^d - 1$

FOR $k = 1$ TO d DO

BEGIN

IF $bit(k, node\_id) = 0$ AND

$bit(1, node\_id) = \cdots = bit(k-1, node\_id) = 1$ THEN

SEND $(X_i, node\_id + 2^{k-1})$

IF $bit(1, node\_id) = \cdots = bit(k, node\_id) = 1$ THEN

RECEIVE $(temp\_X_i)$

$X_i = X_i + temp\_X_i$

END FOR

END ACCUMULATE

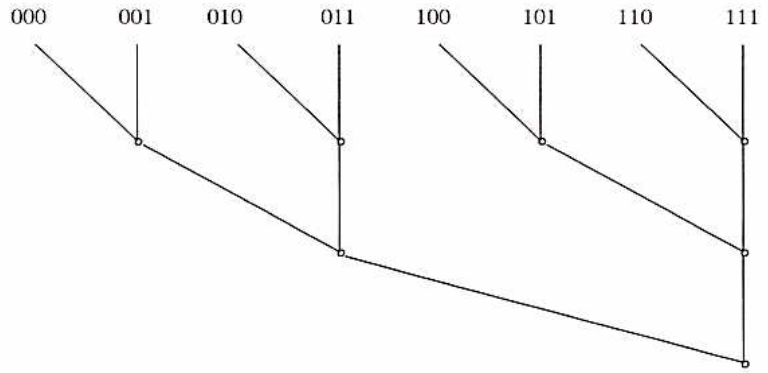

Figure 2 Hypercube communication links for PROCEDURE ACCUMULATE.

PROCEDURE TTR_CUBE

INPUT: $x_0, \ldots, x_N$ and f_i at node i

OUTPUT: T_{ji} for $0 \leqq j \leqq N$ at node i, and α, β at every node

$i = node\_id$

$$w_i = \prod_{\substack{j=0 \\ j \neq i}}^{N} (x_i - x_j); \quad c_i = \frac{f_i}{w_i}; \quad T_{0i} = 1; \quad \beta_0 = 0$$

$\gamma_0 = \text{ACCUMULATE } (c_i x_i);\ \theta_0 = \text{ACCUMULATE } (c_i)$

IF $i = 2^d - 1$ THEN $\alpha_0 = \dfrac{\gamma_0}{\theta_0}$

BROADCAST $(\alpha_0, 2^d - 1)$

$T_{1i} = x_i - \alpha_0$

$\gamma_1 = \text{ACCUMULATE } (c_i x_i T_{1i}^2)$: $\theta_1 = \text{ACCUMULATE } (c_i T_{1i}^2)$

IF $i = 2^d - 1$ THEN $\alpha_1 = \dfrac{\gamma_1}{\theta_1}$; $\beta_1 = \dfrac{\theta_1}{\theta_0}$

BROADCAST $(\alpha_1, 2^d - 1)$; BROADCAST $(\beta_1, 2^d - 1)$

FOR $j = 1$ TO $N - 1$ DO

BEGIN

$T_{j+1,i} = (x_i - \alpha_j) T_{ji} - \beta_j T_{j-1,i}$

$\gamma_{j+1} = \text{ACCUMULATE } (c_i x_i T_{j+1,i}^2);\ \theta_{j+1} = \text{ACCUMULATE } (c_i T_{j+1,i}^2)$

IF $i = 2^d - 1$ THEN $\alpha_{j+1} = \dfrac{\gamma_{j+1}}{\theta_{j+1}}$; $\beta_{j+1} = \dfrac{\theta_{j+1}}{\theta_j}$

BROADCAST $(\alpha_{j+1}, 2^d - 1)$; BROADCAST $(\beta_{j+1}, 2^d - 1)$

END FOR

END TTR_CUBE

LEMMA 2 *PROCEDURE TTR_CUBE computes the matrix T and the vectors α and β defined by the three-term recursion* (2.4) *using $O(N \log N)$ parallel arithmetic operations and $O(N \log N)$ routing steps.*

Proof In PROCEDURE TTR_CUBE, the subprocedures ACCUMULATE and BROADCAST are called $4 + 2(N-1)$ and $3 + 2(N-1)$ times, respectively. Since each such call incurs $d = \log(N+1)$ routing steps, the total overhead in interprocessor communication is

$$(7 + 4(N-1)) \log(N+1) = O(N \log N). \tag{4.1}$$

In addition to the number of arithmetic steps required by the shared-memory case, each execution of ACCUMULATE and BROADCAST contributes $O(\log N)$extra arithmetic operations. Thus the total number of parallel arithmetic operations required for PROCEDURE TTR_CUBE does not exceed the count in (3.2) by more than $O(N \log N)$. Hence the total number of parallel arithmetic steps remains $O(N \log N)$. □

The hypercube implementation of the recursions in Steps 2 and 4 of ALGORITHM RIVOP is a little more subtle and requires a *Gray code G*, of the node_id's of the processors. For our purposes, it suffices to G as a permutation of the set of node_id's $\{0, 1, \ldots, 2^d-1\}$ such that the Hamming distance between $G(k)$ and $G(k+1)$ is 1 for $0 \leqq k \leqq 2^d-2$ [11].

In the implementation of ALGORITHM RIVOP on the hypercube, we perform the computation of the kth column of matrix B in the processor whose node_id is $G(k)$. Note that the computation of $B_{j+1,k}$ requires the quantities $B_{j,k+1}$, B_{jk}, and $B_{j-1,k}$ as well as α_j and β_j. After PROCEDURE TTR_CUBE is called, the vectors α and β are readily available at every node. Thus by making use of the Gray code, we make sure that adjacent columns of the B matrix are computed by neighboring nodes in the hypercube. The computation of the entries of the matrix A is carried out in the same fashion. For the scaling phase of the computation of the matrix A, we make use of the fact that after the second call to PROCEDURE TTR_CUBE, the ith column of the matrices Q and P are available in processor i. At the end of the algorithm, the kth column of the coefficient matrices B and A can be found in node $G(k)$ for $0 \leqq k \leqq N$.

ALGORITHM RIVOP_CUBE

INPUT: $x_0, \ldots, x_N$ and f_i at node i

OUTPUT: A_{jk} and B_{jk} for $0 \leqq j \leqq N$ at node $G(k)$

$[Q, \alpha, \beta]$: =TTR_CUBE$(x_0, \ldots, x_N, f_0, \ldots, f_N)$

$[P, \alpha', \beta']$: =TTR_CUBE$(x_0, \ldots, x_N, f_0^{-1}, \ldots, f_N^{-1})$

$k = G^{-1}$ (*node_id*)

IF $k=0$ THEN $B_{00}=1$; $B_{10}=-\alpha_0$; $A_{00}=1$; $A_{10}=-\alpha_0'$

IF $k=1$ THEN $B_{01}=0$; $B_{11}=1$; $A_{01}=0$; $A_{11}=1$

FOR $j=1$ TO $N-1$ DO

BEGIN

IF $k=j$ THEN $B_{jk}=1$; $A_{jk}=1$

IF $k>j$ THEN $B_{jk}=0$; $A_{jk}=0$

IF $k<j$ THEN SEND $(B_{jk}, G(k+1))$; SEND $(A_{jk}, G(k+1))$

IF $1 \leqq k<j$ THEN

BEGIN

RECEIVE $(temp\_B_{j,k-1})$; RECEIVE $(temp\_A_{j,k-1})$

$B_{j+1,k} = temp\_B_{j,k-1} - \alpha_j B_{jk} - \beta_j B_{j-1,k}$

$A_{j+1,k} = temp\_A_{j,k-1} - \alpha_j A_{jk} - \beta_j A_{j-1,k}$

END

END FOR
FOR $j=0$ to N

BEGIN

IF *node_id* $=0$ THEN $scale = f_0 \dfrac{Q_{N-j,0}}{P_{j0}}$

BROADCAST (*scale*, 0)

$A_{jk} = A_{jk} * scale$

END FOR

END RIVOP_CUBE

THEOREM 2 *On a hypercube multiprocessor with* $2^d = N+1$ *nodes, ALGORITHM RIVOP_CUBE computes all rational interpolants* $r_{m,n}(x)$ *for* $0 \leqq n \leqq N$ *and* $m = N-n$, *using* $O(N \log N)$ *parallel arithmetic operations and* $O(N \log N)$ *routing steps.*

Proof Two calls to PROCEDURE TTR_CUBE are completed with a total of $O(N \log N)$ parallel arithmetic operations and $O(N \log N)$ routing steps by Lemma 2. The first FOR loop contributes $2(N-2)$ additional routing steps., The second FOR loop requires $(N+1)\log(N+1)$ routing steps for broadcasting the scaling factors. Thus the total number of inter-processor communication steps is $O(N \log N)$.

In the first FOR loop, $8(N-1)$ parallel arithmetic operations are performed. The second FOR loop requires $N+1$. Thus the total number of arithmetic steps executed by the algorithm is dominated by the number of arithmetic steps required by PROCEDURE TTR_CUBE, which is $O(N \log N)$ by Lemma 2. □

5. CONCLUSION

Under suitable non-degeneracy conditions on the initial data, ALGORITHM RIVOP makes use of properties of orthogonal polynomials to compute all rational interpolants $r_{m,n}(x)$, $m+n=N$, with $O(N^2)$ arithmetic operations where $N+1$ is the number of data points. This algorithm is suitable for implementation on shared-memory multiprocessor systems. In the shared-memory model, $O(N \log N)$ parallel arithmetic steps suffice construct all rational interpolants using $N+1$ identical processors.

The hypercube version of the algorithm makes use of the properties of hypercube spanning trees, binary addition trees and Gray code for efficient implementation. On a hypercube with $N+1$ processors, ALGORITHM RIVOP_CUBE requires $O(N \log N)$ parallel arithmetic steps and at most $O(N \log N)$ inter-processor communication overhead to compute all rational interpolants. Thus in effect, both parallel versions of ALGORITHM RIVOP require $O(\log N)$ parallel arithmetic

operations to compute each rational interpolant. In the hypercube case, the communication overhead is $O(\log N)$ per interpolant constructed.

References

[1] T. S. Chihara, *An Introduction to Orthogonal Polynomials*, Gordon and Breach, 1978.

[2] P. J. Davis, *Interpolation and Approximation*, Dover Publications, 1975.

[3] Ö. Eğecioğlu and Ç. K. Koç, A fast algorithm for rational interpolation via orthogonal polynomials, *Mathematics of Computation* **53,** No. 187 (1989), 249–264.

[4] G. E. Forsythe, Generation and use of orthogonal polynomials for data fitting with a digital computer, *Journal of SIAM* **5,** No. 2., June (1957), 74–78.

[5] S. Fortune and J. Wylie, Parallelism in random access machines, Proceedings of the 10th ACM Annual Symposium on Theory of Computing, San Diego, CA, May 1978, pp. 114–118.

[6] P. R. Graves-Morris and T. R. Hopkins, Reliable rational interpolation, *Numerische Mathematik* **36** (1981), 111–128.

[7] F. B. Hildebrand, *An Introduction to Numerical Analysis*, McGraw-Hill, 1956.

[8] O. A. McBryan and E. F. Van de Velde, Hypercube algorithms and implementations, *SIAM Journal on Scientific and Statistical Computing* **8,** No. 2, March (1987), s227–s287.

[9] T. Muir, *The Theory of Determinants*, Vol. 2, Dover Publications, 1960, pp. 326–330.

[10] A. Ralston and P. Rabinowitz, *A First Course in Numerical Analysis*, McGraw-Hill, 1978.

[11] J. Rissanen, Solution of linear equations with Hankel and Toeplitz matrices, *Numerische Mathematik* **22** (1974), 361–366.

[12] T. J. Rivlin, *An Introduction to the Approximation of Functions*, Dover Publications, 1969.

[13] Y. Saad and M. H. Schultz, Topological properties of hypercubes, Research Report, Yale University, YALEU/DCS/RR-389, June 1985.

[14] Y. Saad and M. H. Schultz, Data communication in hypercubes, Research Report, Yale University, YALEU/DCS/RR-428, October 1985.

[15] C. Schneider and W. Werner, Some new aspects of rational interpolation, *Mathematics of Computation* **47,** No. 175, July (1986), 285–299.

[16] C. L. Seitz, The cosmic cube, *Communications of the ACM* **28,** No. 1, January (1985), 22–23.

[17] G. Szego, *Orthogonal Polynomials*, American Mathematical Society, 1959.