Orthogonal Polynomials and Least-Squares Approximation on the Hypercube Multiprocessor

Ömer Eğecioğlu
Department of Computer Science
University of California
Santa Barbara, CA 93106

Çetin K. Koç
Department of Electrical Engineering
University of Houston
Houston, TX 77204

ABSTRACT

A parallel algorithm to construct the weighted least-squares approximating polynomials $y_n(x)$ to a given function f on a discrete set $x_0, \ldots, x_N$ is presented. The algorithm first parallelises the classical three-term recursion formula to construct an orthogonal family of polynomials with respect to the associated inner product. The least-squares approximating polynomials can then be obtained from this data by computation of prefix sums. On a hypercube with p processors, the algorithm requires $O(\frac{N^2}{p} + N \log p)$ arithmetic and $O(N \log p)$ routing steps to construct *all* of the polynomials $y_n(x)$, $0 \leq n \leq N$. The hypercube implementation is quite simple, requiring only scalar broadcast and accumulation procedures together with a Gray code mapping. We also present some experimental results obtained on an Intel cube with 8 nodes.

1 Introduction

Given a set of $N+1$ node points $\{x_0, x_1 \ldots, x_N\}$ and observed or computed function values $f_j = f(x_j)$ for $j = 0, 1, \ldots, N$, the least-squares polynomial approximation $y_n(x)$ of degree $n \leq N$ to f on the node set minimises the sum

$$\sum_{j=0}^{N} w_j (f_j - y_n(x_j))^2 \ . \tag{1}$$

Here the weight function $w(x)$ takes on the value w_j on x_j and is assumed to be such that $w_j > 0$ for $0 \leq j \leq N$. For $n = N$, $y_n(x)$ is simply the interpolating polynomial [4, 6].

It is well known that for all but very small values of n, the normal equations arising from (1) result in an ill-conditioned system. For example, when the points x_j are uniformly distributed in the interval $(0, 1)$, essentially a principal minor of the infinite *Hilbert matrix* needs to be inverted [4]. These are classical examples of ill-conditioned matrices and thus, even for reasonable values of n, it becomes impossible to solve the required linear system of equations for the coefficients of the $y_n(x)$'s reliably. For this reason, it is not advisable to compute a solution to (1) via the normal equations [1, 4, 6].

An alternate approach is to use orthogonal polynomials. If the polynomials $\{p_0(x), p_1(x), \ldots, p_N(x)\}$ are orthogonal with respect to the inner-product

$$< u(x), v(x) > = \sum_{j=0}^{N} w_j u(x_j) v(x_j) \tag{2}$$

defined on polynomials of degree $\leq N$ on the nodes, then the least-square polynomial approximant $y_n(x)$ to $f(x)$ has an expansion of the form

$$y_n(x) = \sum_{i=0}^{n} b_i p_i(x) \tag{3}$$

where the coefficients b_i are independent of n [4] and are given by

$$b_i = \frac{\sum_{j=0}^{N} w_j f_j p_i(x_j)}{< p_i(x), p_i(x) >} \ . \tag{4}$$

The polynomials $p_i(x)$ for $0 \leq i \leq N$ can be generated using the classical *three-term recursion* formula [8]

$$p_{i+1}(x) = (x - \alpha_i) p_i(x) - \beta_i p_{i-1}(x) \tag{5}$$

for $0 \leq i \leq N-1$ with

$$p_{-1}(x) = 0 \quad \text{and} \quad p_0(x) = 1 \ , \tag{6}$$

where α_i and β_i are constants determined as

$$\alpha_i = \frac{< x p_i(x), p_i(x) >}{< p_i(x), p_i(x) >} \ , \quad \beta_i = \frac{< p_i(x), p_i(x) >}{< p_{i-1}(x), p_{i-1}(x) >}$$

Consider the $(N+1) \times (N+1)$ matrix $P = [P_{ij}]$ of the values of $p_i(x)$ at the node points x_j for $0 \leq i, j \leq N$, i.e.,

$$P_{ij} = p_i(x_j) \text{ for } \quad 0 \leq i, j \leq N \ .$$

The entries of P can be computed by specializing the three-term recursion (5) by putting $x = x_j$ for $j = 0, 1, \ldots, N$. Furthermore, (5) induces a doubly-indexed recursion on the coefficients of the polynomials $p_i(x)$ for $0 \leq i \leq N$ directly. More precisely, let $A = [A_{ik}]$ be the $(N+1) \times (N+1)$ matrix in which the i*th* row consists of the coefficients of the polynomial $p_i(x)$, i.e.,

$$p_i(x) = \sum_{k=0}^{i} A_{ik} x^k \ .$$

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

$$A_{i+1,k} = A_{i,k-1} - \alpha_i A_{ik} - \beta_i A_{i-1,k} \qquad (7)$$

for $0 \le k \le i \le N$, induced by (5). In (7) we take

$$A_{i,-1} = A_{-1,k} = 0 \text{ for } \quad 0 \le i, k \le N \ .$$

In the generation of the coefficients of the polynomials $p_i(x)$ for $0 \le i \le N$ using the recursion (7), the values of $p_i(x)$ at the node points x_j (i.e. P_{ij}) are also required at each step to compute the quantities α_i and β_i. Thus it is necessary to generate the values and the coefficients simultaneously. This can be done by iterating first the recursion (5) for the values and then the recursion (7) for the coefficients.

Once $\{p_0(x), p_1(x), \ldots, p_N(x)\}$ have been generated with respect to the inner-product (2), we can to compute $y_n(x)$ via (3) and (4) to approximate $f(x)$ in the weighted least-squares sense (1).

Note from (4) that the coefficients of the orthogonal polynomials $p_i(x)$ in the expansion of $y_n(x)$ can be written as

$$b_i = \frac{\sum_{j=0}^{N} w_j f_j P_{ij}}{\sum_{j=0}^{N} w_j P_{ij}^2} \ .$$

Therefore the coefficients b_i can be generated along with α_i and β_i. To find the coefficients of $y_n(x)$ in the power basis in terms of the quantities already available to us at this point, we observe that

$$y_n(x) = \sum_{i=0}^{n} b_i p_i(x) = \sum_{i=0}^{n} b_i \left(\sum_{k=0}^{i} A_{ik} x^k \right) \ .$$

Since $A_{ik} = 0$ for $k > i$, the above equation can also be written as

$$y_n(x) = \sum_{k=0}^{n} \left(\sum_{i=0}^{n} b_i A_{ik} \right) x^k \ .$$

Thus

$$y_n(x) = \sum_{k=0}^{n} C_{nk} x^k \ ,$$

where

$$C_{nk} = \sum_{i=0}^{n} b_i A_{ik} \ . \qquad (8)$$

If we explicitly write equation (8) for $n = 0, 1, \ldots, N$

$$\begin{aligned}
C_{0k} &= b_0 A_{0k} \\
C_{1k} &= b_0 A_{0k} + b_1 A_{1k} \\
C_{2k} &= b_0 A_{0k} + b_1 A_{1k} + b_2 A_{2k} \\
&\cdots \\
C_{Nk} &= b_0 A_{0k} + b_1 A_{1k} + b_2 A_{2k} + \cdots + b_N A_{Nk}
\end{aligned}$$

we immediately observe that C_{nk} for $0 \le n \le N$ are the prefix sums of the quantities

$$b_0 A_{0k} \ , \ b_1 A_{1k} \ , \ \cdots \ , \ b_N A_{Nk} \ .$$

PROCEDURE LSPA (Least-Squares Polynomial Approximation) given below computes all of the polynomials $y_n(x)$ for $0 \le n \le N$ which approximate $f(x)$ on the node points in least-squares sense.

PROCEDURE LSPA

Input: x_j, w_j, f_j for $0 \le j \le N$

Output: C_{nk} for $0 \le k, n \le N$ such that $y_n(x) = \sum_{i=0}^{n} C_{nk} x^k$ minimizes (1)

Step 1. Set $P_{0j} = 1$ for $0 \le j \le N$ and $\beta_0 = 0$, and compute

$$\gamma_0 = \sum_{j=0}^{N} w_j x_j \ , \quad \theta_0 = \sum_{j=0}^{N} w_j \ , \quad \eta_0 = \sum_{j=0}^{N} w_j f_j \ ,$$

$$\alpha_0 = \frac{\gamma_0}{\theta_0} \ , \quad b_0 = \frac{\eta_0}{\theta_0} \ .$$

Step 2. Set $P_{1j} = x_j - \alpha_0$ for $0 \le j \le N$, and compute

$$\gamma_1 = \sum_{j=0}^{N} w_j x_j P_{1j}^2 \ , \quad \theta_1 = \sum_{j=0}^{N} w_j P_{1j}^2 \ ,$$

$$\alpha_1 = \frac{\gamma_1}{\theta_1} \ , \quad \beta_1 = \frac{\theta_1}{\theta_0} \ ,$$

$$\eta_1 = \sum_{j=0}^{N} w_j f_j P_{1j} \ , \quad b_1 = \frac{\eta_1}{\theta_1} \ .$$

Step 3. For $1 \le i \le N-1$ compute

$$P_{i+1,j} = (x_j - \alpha_i) P_{ij} - \beta_i P_{i-1,j} \text{ for } \quad 0 \le j \le N$$

$$\gamma_{i+1} = \sum_{j=0}^{N} w_j x_j P_{i+1,j}^2 \ , \quad \theta_{i+1} = \sum_{j=0}^{N} w_j P_{i+1,j}^2 \ ,$$

$$\alpha_{i+1} = \frac{\gamma_{i+1}}{\theta_{i+1}} \ , \quad \beta_{i+1} = \frac{\theta_{i+1}}{\theta_i} \ ,$$

$$\eta_{i+1} = \sum_{j=0}^{N} w_j f_j P_{i+1,j} \ , \quad b_{i+1} = \frac{\eta_{i+1}}{\theta_{i+1}} \ .$$

Step 4. Set $A_{ii} = 1$ for $0 \le i \le N$, and $A_{ik} = 0$ for $0 \le i < k \le N$. Set $A_{10} = -\alpha_0$. For all $1 \le k < i \le N-1$ compute

$$A_{i+1,k} = A_{i,k-1} - \alpha_i A_{ik} - \beta_i A_{i-1,k}$$

Step 5. For all $0 \le k \le n \le N$ compute $C_{nk} = b_n A_{nk}$. Set $C_{0k} = 0$ for $0 \le k \le N$, and for all $0 \le k \le n \le N-1$ compute

$$C_{n+1,k} = C_{n+1,k} + C_{nk}$$

Theorem 1 *PROCEDURE LSPA computes C_{nk} for $0 \le k \le n \le N$ using $O(N^2)$ sequential arithmetic steps.*

Proof In step 1, the computation of γ_0, θ_0, η_0, α_0, and b_0 requires $2N+1$, N, $2N+1$, 1, and 1 arithmetic steps, respectively. In step 2, first we compute the P_{1j}'s using $N+1$ steps. Then γ_1, θ_1, η_1, α_1, β_1, and b_1 are computed using a total of $11N+11$ arithmetic operations. Similarly, step 3 requires $(N-1)(14N+14)$ arithmetic steps.

The computation of A_{ik} in step 4 requires

$$4(1 + 2 + \cdots + N - 1 + N - 1) = 2N^2 + 2N - 4$$

arithmetic steps. In step 5, a total of $N^2 + 2N + 1$ arithmetic operations need to be performed. Thus a total of

$$17N^2 + 10N + 8 = O(N^2) \qquad (9)$$

arithmetic steps are required by PROCEDURE LSPA. □

2 Hypercube Implementation

We consider the implementation of PROCEDURE LSPA on a hypercube with $2^d = p \leq N+1$ nodes. For simplicity we assume that p divides $N+1$, i.e., $pm = N+1$ for some $m > 1$. We partition the matrix P such that the first m columns are computed at node 0, the second m columns at node 1, etc. Let $G(i)$ represent the binary-reflected Gray code of integer i [5]. We partition A such that first m columns are computed at processor $G(0)$, the second m columns at processor $G(1)$, etc.

This partitioning allows us to compute the values and the coefficients of the orthogonal polynomials, and thus all least-square polynomial approximants in an efficient manner. To implement PROCEDURE LSPA on a hypercube with $2^d = N+1$ nodes we use the following two procedures:

broadcast A data item X, which is initially located in node q, is being sent to all nodes. This algorithm makes use of a hypercube spanning tree [2, 7] rooted at node q. The number of parallel routing steps required by this procedure is d, the dimension of the cube.

accumulate Computation of the sum $S = \sum_{k=0}^{N} X_k$ where $N+1 = 2^d \times m$ for some $m \geq 1$. The data items X_k for $jm \leq k \leq jm+m-1$ are initially located in node j for $0 \leq j \leq 2^d-1$, i.e., $N+1$ elements are partitioned such that each node contains m elements where $pm = N+1$. First we perform sequential summation at each node simultaneously which takes $m-1$ arithmetic steps to complete. Then we use the binary tree addition algorithm to sum these block sums at each node to find the total sum. This step takes d arithmetic and d routing steps. Thus, this procedure executes $m-1+d$ arithmetic operations and d routing operations to find the sum of $N+1$ elements distributed on 2^d nodes. After the execution of procedure accumulate, the sum is found in node 2^d-1 [7].

Theorem 2 *PROCEDURE LSPA computes the coefficient matrix $C = [C_{nk}]$ of the least-squares approximating polynomials $y_n(x)$ to f using $O(\frac{N^2}{p} + N \log p)$ parallel arithmetic and $O(N \log p)$ routing steps on a hypercube with $p \leq N+1$ processors.*

Proof Initially we assume that each processor has copies of x_i, w_i for $0 \leq i \leq N$. In steps 1–4 we compute the entries of matrix P and coefficients b_i for $0 \leq i \leq N$. The operations performed are a) accumulation which computes a sum, e.g. the computation of γ_0 in step 1, and b) broadcast which sends a value, e.g. α_0, to all processors since it will be used by all processors in the following step. As it follows from the discussions above, the accumulation step requires $m-1+d$ arithmetic steps and d routing steps, while the broadcast step requires d routing steps.

In step 1, the number of arithmetic and routing operations for the computation of γ_0 are $2m-1+d$ and d, respectively. Similarly, the computation of η_0 requires $2m-1+d$ arithmetic and d routing steps. For θ_0, we need to perform $m-1+d$ arithmetic and d routing steps. After these accumulation steps, γ_0, θ_0, and η_0 are found in node 2^d-1. We then perform 2 division operations in this node to compute α_0 and b_0. Since these values are needed by all nodes, we use the broadcast routine explained above to send these values to all nodes. The broadcast of a scalar takes d steps. Thus, step 1, all together, requires $5m+3d-1$ arithmetic and $5d$ routing steps.

Similarly, the number of arithmetic and routing steps in step 2 of PROCEDURE LSPA are $11m+3d$ and $6d$, respectively. Step 3 takes $(N-1)(14m+3d)$ arithmetic and $(N-1)6d$ routing steps.

For the computation of the entries of the matrix A, recall that we have partitioned the matrix A such that the first m columns are computed at the processor $G(0)$, the second group is computed at the node $G(1)$, and so on. Thus the element $A_{i,qm+k}$ is computed at the node $G(q)$ for $0 \leq k \leq m-1$ and for $i = 0, 1, \ldots, N$. The analysis can be greatly simplified by considering only those operations which processor $G(0)$ has to perform. Since the elements $A_{i,0}, A_{i,1}, \ldots, A_{i,m-1}$ are computed at the node $G(0)$, there is no need to send operands for $i = 0, 1, \ldots, m-1$. For $i = m, m+1, \ldots, N$, the processor $G(q)$ sends $A_{i,qm+m-1}$ to processor $G(q+1)$ for the computation of $A_{i+1,(q+1)m}$ which is the first element to be computed at this processor. Since the Hamming distance between $G(q)$ and $G(q+1)$ is just 1 bit, this takes a single routing step. It follows from this observation that the computation of the entries of A in step 4 takes $N-m$ parallel routing steps. Also at the first step processor $G(0)$ computes the quantities $A_{i,0}, A_{i,1}, \ldots, A_{i,m-1}$ for $i = 0, 1, \ldots, m-1$ except for A_{10} and A_{ii}, $0 \leq i \leq m-1$. Each of these operations takes 4 arithmetic steps in the light of (7). The total number of arithmetic operations for this step becomes $4(1+2+\cdots+m)-4-4m = 2m^2-2m+4$. The elements that follow, $A_{i,k}$ for $m \leq i \leq N$ and $0 \leq k \leq m-1$, are computed using locally available data. Therefore this step takes $(N+1-m)4m$ arithmetic steps.

Thus we observe that the total number of arithmetic and routing operations for the computation of the entries A in step 4 becomes $4Nm-2m^2+2m-4$ and $N-m$, respectively.

Step 5 of the cube implementation of PROCEDURE LSPA is completely sequential. Again, we count the number of operations which processor $G(0)$ has to perform. First $C_{nk} = b_n A_{nk}$ is computed using

$$(N+1)+(N)+\cdots+(N+1-(m-1)) = (N+1)m-\frac{1}{2}m(m-1)$$

arithmetic operations with the locally available data. Then the sequential prefix algorithm is used to compute the final values of C_{nk}. This step requires

$$(N)+(N-1)+\cdots+(N-(m-1)) = Nm-\frac{1}{2}m(m-1)$$

arithmetic operations. Thus a total of $2Nm+2m-m^2$ arithmetic steps are required in step 5.

Thus for the hypercube implementation of PROCEDURE LSPA, a total of

$$18Nm+3Nd+3d+6m-3m^2-5 = O(\frac{N^2}{p}+N\log p) \quad (10)$$

arithmetic operations and

$$6Nd + N + 5d - m = O(N \log p) \qquad (11)$$

routing operations are required. □

3 Experimental Results

We have performed experiments, similar to those mentioned in [3], on an Intel iPSC/d3 hypercube running XENIX 286 R3.4 and iPSC Software R3.1 to measure the time required to perform a floating-point operation (τ_{comp}), and the time required to transfer a floating-point number to an adjacent node (τ_{comm}). The experiments indicated that $\tau_{comm} \approx 1.48$ milliseconds, and if the floating-point operation is taken to be multiplication, addition, or subtraction then $\tau_{comp} \approx 0.058$ milliseconds. Division takes a little longer (≈ 0.072 milliseconds). Using these values and equations (9),(10), and (11) we find the sequential and the parallel time required by PROCEDURE LSPA as

$$T_{seq} = \left(17N^2 + 10N + 8\right) \tau_{comp} \ ,$$

$$T_{par} = \left(18Nm + 3N \log p + 3 \log p + 6m - 3m^2 - 5\right) \tau_{comp} + \left(6N \log p + N + 5 \log p - m\right) \tau_{comm} \ ,$$

where $m = (N+1)/p$.

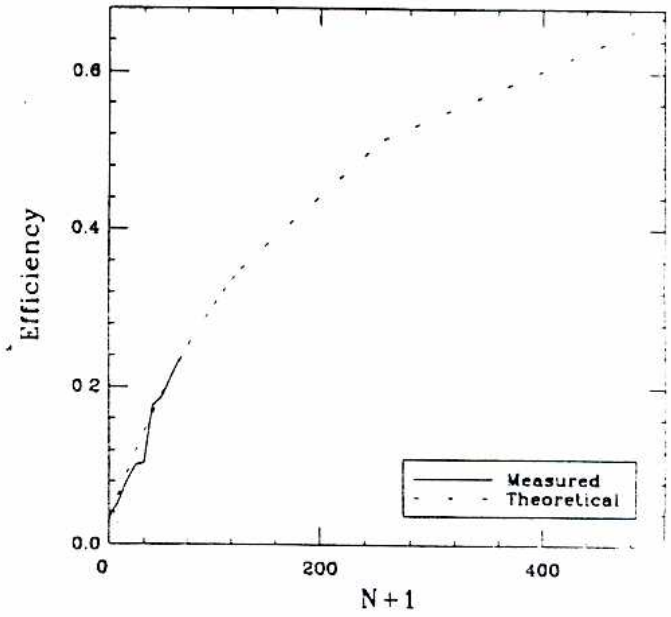
We have implemented PROCEDURE LSPA on a first generation Intel cube with 8 nodes (iPSC/d3, a.k.a. SugarCube). The timing results are shown in Table 1 for the values of $8 \leq N+1 \leq 72$.

Table 1. The measured time on the cube (in milliseconds).

N+1	Time ($p=1$)	Time ($p=8$)
8	45	160
16	185	455
24	415	655
32	740	920
40	1160	1400
48	1675	1180
56	2285	1535
64	2995	1775
72	3815	2035

We also computed the theoretical efficiency using the values of $\tau_{comp} = 0.058$ and $\tau_{comm} = 1.48$. Figure 1 illustrates the theoretical and the measured efficiency of PROCEDURE LSPA. Thus, we see that on a 3-cube more than 50 % efficiency is achieved for $N + 1 = 256$.

Figure 1. Efficiency of PROCEDURE LSPA



References

[1] G. E. Forsythe, "Generation and Use of Orthogonal Polynomials for Data-Fitting with a Digital Computer," *Journal of SIAM*, Vol. 5, No. 2, pp. 74–88, 1957.

[2] S. L. Johnsson, "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures," *Journal of Parallel and Distributed Computing*, No. 4, pp. 133–172, 1987.

[3] O. A. McBryan and E. F. Van de Velde, "Hypercube Algorithms and Implementations," *SIAM Journal on Scientific and Statistical Computing*, Vol. 8, No. 2, pp. s227–s287, March 1987.

[4] A. Ralston and P. Rabinowitz, *A First Course in Numerical Analysis*, McGraw-Hill, 1985.

[5] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms*, Prentice-Hall, 1977.

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

[7] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *IEEE Transactions on Computers*, Vol. 37, No. 7, pp. 867–872, July 1988.

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