1 of 1

ORNL/TM-12285

Engineering Physics and Mathematics Division

Mathematical Sciences Section

**IMPLEMENTATION AND PARALLELIZATION OF FAST MATRIX
MULTIPLICATION FOR A FAST LEGENDRE TRANSFORM**

Wentao Chen
Denison University
Granville, OH 43023

Date Published: September 1993

Research sponsored jointly by the Great Lakes Colleges
Association/Associated Colleges of the Midwest and the
U.S. Department of Energy CHAMMP Program of the Of-
fice of Health and Environmental Research.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

MASTER

ab
DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Contents

1	Introduction	1
2	Mathematical Background	1
2.1	Legendre and Chebyshev Polynomials	1
2.2	Relationship Between Legendre and Chebyshev Expansions	2
2.3	Lagrange Interpolation	3
3	Fast Matrix Multiplication in Sequential Mode	3
3.1	Description of the Fast Matrix Multiplication	4
3.2	Time Complexity Analysis	8
3.3	Numerical Results	9
4	Parallel Implementation	12
4.1	Description of the Parallel Implementation	12
4.2	Numerical Results	13
5	Generalization	17
6	Conclusion	17
7	References	20
A	Derivation of Recursions for $b_{l,j}^r$ and $d_{l,j}^r$	21
B	Pseudo Code For $O(n)$ Method of Matrix-vector Multiplication ($\vec{w} = M\vec{v}$) . .	23

Acknowledgements

Prepared in partial fulfillment of the requirement of the ORNL-GLCA/ACM Science Semester under the direction of Patrick H. Worley and John B. Drake, Research Supervisors, in the Engineering Physics and Mathematics division.

I am very grateful to all the people who have helped with my research project, especially to those in the Engineering Physics and Mathematics Division at the Laboratory. I would like to give special thanks to my advisors John B. Drake and Patrick H. Worley, my collaborator Brad F. Lyon, and Esmond Ng, who helped a great deal putting this report together.

IMPLEMENTATION AND PARALLELIZATION OF FAST MATRIX MULTIPLICATION FOR A FAST LEGENDRE TRANSFORM

Wentao Chen
Denison University
Granville, OH 43023

Abstract

An algorithm was presented by Alpert and Rokhlin for the rapid evaluation of Legendre transforms. The fast algorithm can be expressed as a matrix-vector product followed by a fast cosine transform. Using the Chebyshev expansion to approximate the entries of the matrix and exchanging the order of summations reduces the time complexity of computation from $O(n^2)$ to $O(n \log n)$, where n is the size of the input vector. Our work has been focused on the implementation and the parallelization of the fast algorithm of matrix-vector product. Results have shown the expected performance of the algorithm. Precision problems which arise as n becomes large can be resolved by doubling the precision of the calculation.

1. Introduction

Legendre expansions are very important in applied mathematics. Although they can be useful numerical tools in many applications, the implementation poses difficulty. For an n -term Legendre expansion, normally it takes $O(n^2)$ operations to evaluate the expansion at n nodes. Alpert, et. al., [1] presented an algorithm for the fast evaluation of a Legendre expansion at Chebyshev nodes on the interval $[-1,1]$. Suppose we have a function f that has a finite Legendre expansion:

$$f(t) = \sum_{j=0}^{n-1} \alpha_j \cdot P_j(t) . \quad (1)$$

The algorithm substitutes the Legendre expansion with a Chebyshev expansion of the same length and subsequently evaluates the Chebyshev expansion using a fast cosine transform algorithm. The replacement of the Legendre expansion by a Chebyshev expansion, which turns out to be a matrix-vector multiplication, can be done in $O(n)$ operations, while the fast cosine transform requires $O(n \log n)$ operations. Therefore, the overall time complexity of the algorithm is of order $O(n \log n)$.

We have concentrated our efforts on the implementation and parallelization of the matrix-vector multiplication part of the algorithm since there are existing codes for the fast cosine transform. In the process of generating the code, we discovered that several points needed clarification in [1]. In section 2 of this paper, we present a general mathematical background of the fast algorithm. In sections 3 and 4, we will describe in detail the sequential algorithm of the fast matrix-vector multiplication and parallel implementation respectively. A discussion of generalization of the fast algorithm for the input size can be found in section 5. Section 6 contains the conclusion of the current work and discuss avenue of future research.

2. Mathematical Background

In this section, we summarize the basic mathematical terms and theories used in the fast matrix multiplication for Legendre expansions.

2.1. Legendre and Chebyshev Polynomials

Suppose we have a set of polynomials $\{\phi_0, \phi_1, \dots, \phi_n\}$ of degrees $0, 1, \dots, n$. It is said to be an *orthogonal set of functions* on the interval $[a, b]$ with respect to the nonnegative continuous weight function w on this interval if

$$\int_a^b w(t) \phi_j(t) \phi_k(t) dt = \begin{cases} 0, & \text{whenever } j \neq k, \\ \alpha_k > 0, & \text{whenever } j = k. \end{cases} \quad (2)$$

Both *Legendre* and *Chebyshev* polynomials are orthogonal sets of polynomials on the interval $[-1,1]$ [6]. The weight functions associated with these orthogonal sets are $w(t) \equiv 1$ and $w(t) = 1/\sqrt{1-t^2}$ (on the interval $(-1,1)$), respectively. The *Legendre*

polynomials, $\{P_n\}$, take the form:

$$P_n(t) = \frac{1}{2^n n!} \frac{d^n}{dt^n} (t^2 - 1)^n \quad (n \geq 0), \quad (3)$$

while the *Chebyshev* polynomials, $\{T_n\}$, take the form:

$$T_n(t) = \cos(n \arccos t) \quad (n \geq 0). \quad (4)$$

Chebyshev nodes of order k are the zeros of $T_k(t)$. The i th node of order k on the interval $[-1, 1]$ can be expressed as

$$t_i = \frac{1}{2} \cos\left(\frac{(2i+1)\pi}{2k}\right), \quad (5)$$

for $i = 0, 1, \dots, k-1$.

2.2. Relationship Between Legendre and Chebyshev Expansions

Suppose that a function $f: [-1, 1] \rightarrow \mathbb{R}$ has a finite Legendre expansion of the form

$$f(\cos \theta) = \sum_{i=0}^{n-1} \alpha_i \cdot P_i(\cos \theta), \quad (6)$$

and also has a finite Chebyshev expansion of the form

$$f(\cos \theta) = \sum_{i=0}^{n-1} \beta_i \cdot T_i(\cos \theta) = \sum_{i=0}^{n-1} \beta_i \cdot \cos(i\theta). \quad (7)$$

Notice here that the second term in Eq. (7) is the cosine expansion. The two vectors $\vec{\alpha} = (\alpha_0, \alpha_1, \dots, \alpha_{n-1})^T$ and $\vec{\beta} = (\beta_0, \beta_1, \dots, \beta_{n-1})^T$ have the relationship as follows [1,5]:

$$\vec{\beta} = M^n \vec{\alpha} \quad (8)$$

and

$$\vec{\alpha} = L^n \vec{\beta}. \quad (9)$$

M^n and L^n are $n \times n$ matrices defined by

$$M_{ij}^n = \begin{cases} \frac{1}{\pi} \Lambda\left(\frac{j}{2}\right)^2 & \text{if } 0 = i \leq j < n \text{ and } j \text{ is even,} \\ \frac{2}{\pi} \Lambda\left(\frac{j-i}{2}\right) \Lambda\left(\frac{j+i}{2}\right) & \text{if } 0 < i \leq j < n \text{ and } i+j \text{ is even,} \\ 0 & \text{otherwise,} \end{cases} \quad (10)$$

and

$$L_{ij}^n = \begin{cases} 1 & \text{if } i = j = 0, \\ \frac{\sqrt{\pi}}{2 \Lambda(i)} & \text{if } 0 < i = j < n, \\ \frac{-j(i+\frac{1}{2})}{(j+i+1)(j-i)} \Lambda\left(\frac{j-i-2}{2}\right) \Lambda\left(\frac{j+i-1}{2}\right) & \text{if } 0 \leq i < j < n \text{ and } i+j \text{ is even,} \\ 0 & \text{otherwise,} \end{cases} \quad (11)$$

where function $\Lambda : C \rightarrow C$ is defined by the formula:

$$\Lambda(z) = \frac{\Gamma(z + \frac{1}{2})}{\Gamma(z + 1)}. \quad (12)$$

Here Γ is the gamma function [3].

2.3. Lagrange Interpolation

The following two theorems can be found in [2].

Theorem 2.1. *If $t_0, t_1, \dots, t_n$ are distinct numbers on the interval $[a, b]$ and $f \in C^{n+1}[a, b]$, for each $t \in [a, b]$, there exists a number $\xi(t)$ such that*

$$f(t) = Q(t) + \frac{f^{(n+1)}(\xi(t))}{(n+1)!} (t - t_0)(t - t_1) \cdots (t - t_n), \quad (13)$$

where $Q(t)$ is defined by

$$Q(t) = \sum_{i=0}^n f(t_i) u_i(t), \quad (14)$$

and

$$u_i(t) = \prod_{j=0, j \neq i}^n \frac{t - t_j}{t_i - t_j} \quad (15)$$

is the i th Lagrange interpolating polynomial.

Notice that a function is said to be smooth if it has n continuous derivatives on its domain. The second term of the right hand side in Eq.(13) is the error term of the Lagrange interpolation and it can be minimized using the following theorem.

Theorem 2.2. *If Q is an interpolating polynomial of degree n for a function $f \in C^{n+1}[-1, 1]$, the maximum error of the interpolation is minimized when Chebyshev nodes of order $n + 1$ are used for the interpolation points and*

$$\max_{t \in [-1, 1]} |f(t) - Q(t)| \leq \frac{1}{2^n (n+1)!} \max_{t \in [-1, 1]} |f^{(n+1)}(t)|. \quad (16)$$

We can extend this technique for choosing Chebyshev nodes to minimize the interpolating error to any closed interval $[a, b]$ by using the change of variable $\tilde{t} = \frac{1}{2}[(b-a)t + b+a]$ to transform Chebyshev nodes t_i on the interval $[-1, 1]$ into the corresponding $\tilde{t}_i$ on the interval $[a, b]$.

3. Fast Matrix Multiplication in Sequential Mode

In this section, we discuss the fast matrix multiplication algorithm, clarify the key points, present our implementation, and show the results of our implementation.

3.1. Description of the Fast Matrix Multiplication

First we present an example illustrating the fast matrix-vector multiplication algorithm presented in [1]. Suppose we have an $n \times n$ matrix T^n whose entry T_{ij}^n can be expressed as a smooth function $T(i, j) : R \times R \rightarrow R$. The smoothness of the function T allows us to compute $\vec{w} = T\vec{v}$ efficiently, where $\vec{v} = (v_0, v_1, \dots, v_{n-1})^T$ is an arbitrary vector. To compute

$$w_i = \sum_{j=0}^{n-1} T(i, j) \cdot v_j, \quad (17)$$

we expand $T(i, j)$ at the Chebyshev nodes of order k , $\{t_0, t_1, \dots, t_{k-1}\}$, in j . We have

$$T(i, j) \approx \sum_{r=0}^{k-1} T(i, t_r n) \cdot u_r\left(\frac{j}{n}\right). \quad (18)$$

The error of the approximation depends on the infinite norm $\|T^{(k+1)}\|_\infty$ of the matrix. Substituting Eq.(18) into Eq.(17) and exchanging the order of summation, we get

$$w_i \approx \sum_{r=0}^{k-1} T(i, t_r n) \sum_{j=0}^{n-1} u_r\left(\frac{j}{n}\right) \cdot v_j = \sum_{r=0}^{k-1} T(i, t_r n) \cdot b_r, \quad (19)$$

where

$$b_r = \sum_{j=0}^{n-1} u_r\left(\frac{j}{n}\right) \cdot v_j \quad \text{for } r = 0, 1, \dots, k-1. \quad (20)$$

Evaluating b_r , $r = 0, 1, \dots, k-1$, takes $O(nk)$ operations and, given these values, the evaluation of vector $\vec{w}$ takes an additional $O(nk)$ operations. For bounded $\|T^{k+1}\|_\infty$ and a given error bound ϵ , $k \propto \log(1/\epsilon)$. Therefore, the time complexity of computing $\vec{w} = T\vec{v}$ is reduced from $O(n^2)$ to $O(n \log(1/\epsilon))$. The key to the reduction of the number of operations is the Chebyshev expansion of matrix entries and the exchange of summation.

Examining Eq.(10) and Eq.(11), we notice that the entries for the two matrices M and L cannot be expressed as smooth functions. Both matrices are upper triangular with every other entry at the upper triangular part as 0. We cannot apply the fast method directly. If we define the following two smooth functions $M, L : R \times R \rightarrow R$ by the formulae

$$M(x, y) = \frac{2}{\pi} \Lambda\left(\frac{y-x}{2}\right) \Lambda\left(\frac{y+x}{2}\right), \quad (21)$$

and

$$L(x, y) = \frac{-y(x + \frac{1}{2})}{(y+x+1)(y-x)} \Lambda\left(\frac{y-x-2}{2}\right) \Lambda\left(\frac{y+x-1}{2}\right), \quad (22)$$

we have $M_{ij}^n = M(i, j)$ when $0 < i \leq j < n$ and $i+j$ is even, and $L_{ij}^n = L(i, j)$ when $0 \leq i < j < n$ and $i+j$ even. We can separate the matrices into four parts if we rearrange the matrices in the following way: first group even indexed rows together and put them at the upper halves of the matrices, and odd indexed rows at the lower halves, then rearrange the columns similarly. The bottom-left and the upper-right

submatrices become 0 matrices while the upper-left and the bottom-right ones become upper triangular matrices, whose entries are smooth. In order to keep the correct order of computation, we must rearrange the input vector accordingly, by separating even and odd indexed entries. Thus the matrix-vector multiplication becomes

$$M\vec{v} \Rightarrow \begin{pmatrix} M_{ee} & 0 \\ 0 & M_{oo} \end{pmatrix} \begin{pmatrix} v_e \\ v_o \end{pmatrix} \Rightarrow \begin{pmatrix} M_{ee}\vec{v}_e \\ M_{oo}\vec{v}_o \end{pmatrix}. \quad (23)$$

Finally, we must rearrange the output to obtain the final result with entries in the correct order.¹

By the reordering, the matrix-vector multiplication problem is reduced to the multiplications of upper triangular submatrices and vectors of half size. However, we still cannot apply directly the fast method discussed earlier in this section. Only the upper triangular parts of the submatrices can be expressed as smooth functions. In [1], a method of partitioning the upper triangular matrix was presented. The partitioning is illustrated in Fig. 1.² The upper triangular matrix is partitioned into square submatrices, noted as ${}_lS_{i,j}$, with side sizes of powers of 2. Here l is the level of the submatrix in the partition. The lower the level, the smaller the submatrix is and the more submatrices on this level are. The total number of levels of partition is $h = \log_2(n/s) - 1$. The smallest matrices have level number 1 and are of size s (a parameter which is adjusted later to minimize the execution time). The matrix size of any level $l > 1$ is twice that of the previous level. Indices i and j indicate where the submatrix is located in the original matrix. The position of any submatrix ${}_lS_{i,j}$ in the original matrix can be decided by the formula

$$[i \cdot m, (i + 1) \cdot m - 1] \times [j \cdot m, (j + 1) \cdot m - 1]. \quad (24)$$

where $m = 2^{l-1} \cdot s$ is the side size of a submatrix on level l . We can now apply the fast method to the multiplications of these submatrices and the corresponding portion of the input vector. For the unpartitioned parts, those entries close to or on the main diagonal, we multiply directly with the corresponding portion of the input vector. Summing up corresponding results, we obtain the product of the matrix and the vector.

We now discuss the time complexity of the above method. On level l , there are $3 \cdot (2^{h+1-l} - 1)$ submatrices of order $2^{l-1} \cdot s$. According to the properties of M and L matrices, the error of the interpolation shrinks as 3^{-k} [1]. Therefore for error bound ϵ , $k \propto \log(1/\epsilon)$. We conclude that it takes $O(n \log(1/\epsilon))$ to do a matrix-vector multiplication of order n for a submatrix; if ϵ is a constant, the time complexity is of order $O(n)$. Therefore, the time complexity of multiplications of the partitioned part is

$$\sum_{l=1}^h [3 \cdot (2^{h+1-l} - 1)] \cdot [2^{l-1} \cdot s] = 3 \cdot s \cdot (h - 1) \cdot 2^h + 3 \cdot s. \quad (25)$$

¹These rearrangements are vital to the fast matrix multiplication but were not stated in [1].

²Our work has been focused on input vector of order of power of 2, so in the discussion below, the size of input is assumed to be a power of 2 unless it is explicitly stated otherwise.

Figure 1: Subdivision of the upper triangular matrix to three levels.

If we also consider s constant, the time complexity is of order $O(n \log n)$, since $h = \log_2(n/s) - 1$. Taking into account the unpartitioned part, we conclude the overall time complexity is also of order $O(n \log n)$.

As we have noticed, the key points of the fast matrix multiplication algorithm are the use of the Chebyshev expansion and the exchange of summation. Previously, we expanded the entries of matrix in one direction only. Further improvement can be gained by expanding the entries of matrix in both i and j directions. Let $m = 2^{l-1} \cdot s$ be the side size of a submatrix on level l , the matrix-vector multiplication for submatrix ${}_l S_{i,j}$ becomes

$$\begin{aligned}
 w'_{i+i'} &= \sum_{j'=0}^{m-1} M(im + i', jm + j') \cdot v_{j'+jm} \\
 &= \sum_{j'=0}^{m-1} \sum_{r=0}^{k-1} M[im + i', (j + t_r)m] \cdot u_r\left(\frac{j'}{m}\right) \cdot v_{j'+jm} \\
 &= \sum_{j'=0}^{m-1} \sum_{r=0}^{k-1} \sum_{p=0}^{k-1} M[(i + t_p)m, (j + t_r)m] \cdot u_p\left(\frac{i'}{m}\right) \cdot u_r\left(\frac{j'}{m}\right) \cdot v_{j'+jm} \\
 &= \sum_{p=0}^{k-1} u_p\left(\frac{i'}{m}\right) \sum_{r=0}^{k-1} M[(i + t_p)m, (j + t_r)m] \sum_{j'=0}^{m-1} u_r\left(\frac{j'}{m}\right) \cdot v_{j'+jm}, \quad (26)
 \end{aligned}$$

where $i' = 0, 1, \dots, m - 1$. For clarity, we introduce intermediate variables as in [1] by

the following formulae

$$u_{r,j'}^l = u_r\left(\frac{j'}{m}\right), \quad (27)$$

$$b_{l,j}^r = \sum_{j'=0}^{m-1} u_{r,j'}^l \cdot v_{j'+jm}, \quad (28)$$

$${}_l M_{i,j}^{p,r} = M[(i + t_p)m, (j + t_r)m], \quad (29)$$

$${}_l c_{i,j}^p = \sum_{r=0}^{k-1} {}_l M_{i,j}^{p,r} \cdot b_{l,j}^r. \quad (30)$$

The following notations are helpful since we must sum up the corresponding w_i 's for the partitioned part to obtain the partial result.

$$\left. \begin{aligned} c_{i,2j}^p &= {}_l c_{2j,2j+2}^p + {}_l c_{2j,2j+3}^p \\ c_{i,2j+1}^p &= {}_l c_{2j+1,2j+3}^p \end{aligned} \right\} \quad j = 0, 1, \dots, n/(2^l \cdot s) - 2, \quad (31)$$

$$c_{l,n/m-2}^p = c_{l,n/m-1}^p = 0, \quad (32)$$

and

$$a_{l,j}^{i'} = \sum_{p=0}^{k-1} u_{p,i'}^l \cdot c_{l,j}^p. \quad (33)$$

To obtain the final output vector for $i = 0, 1, \dots, n-1$, we define the following

$$w_i = \sum_{l=1}^{\lceil \log_2((n-i)/(2s)) \rceil} a_{l,[i/m]}^{i \bmod m} + \sum_{r=i}^{\lfloor i/s \rfloor s + 2s - 1} M_{ir}^n \cdot v_r, \quad (34)$$

where the first term on the right hand side represents the computation of the partitioned part of the matrix while the second term represents that of the near diagonal part in the matrix.

If we count the number of operations required to compute $b_{l,j}^r$, we can confirm that the method has the complexity of order $O(n \log n)$. A better algorithm proposed in [1], which is an $O(n)$ method, uses the following recursive expressions³

$$b_{1,j}^r = \sum_{i=0}^{s-1} u_r\left(\frac{i}{s}\right) \cdot v_{i+js}, \quad (35)$$

$$b_{l,j}^r = \sum_{i=0}^{k-1} \left[u_r\left(\frac{t_i}{2}\right) \cdot b_{l-1,2j}^i + u_r\left(\frac{1+t_i}{2}\right) \cdot b_{l-1,2j+1}^i \right] \quad (l = 2, 3, \dots, h), \quad (36)$$

and

$$d_{h,j}^r = c_{h,j}^r, \quad (37)$$

³See Appendix A for detail.

$$\left. \begin{aligned} d_{l,2j}^r &= c_{l,2j}^r + \sum_{i=0}^{k-1} u_i\left(\frac{t_r}{2}\right) \cdot d_{l+1,j}^i \\ d_{l,2j+1}^r &= c_{l,2j+1}^r + \sum_{i=0}^{k-1} u_i\left(\frac{1+t_r}{2}\right) \cdot d_{l+1,j}^i \end{aligned} \right\} \quad (l = 1, 2, \dots, h-1). \quad (38)$$

And the output vector can be calculated by the formula

$$w_{i'+js} = \sum_{i=0}^{k-1} u_i\left(\frac{i'}{s}\right) \cdot d_{1,j}^i + \sum_{i=i'}^{2s-1} M_{i'+js,i+js}^n \cdot v_{i'+js} \quad (39)$$

where $i' = 0, 1, \dots, s-1$, and $j = 0, 1, \dots, n/s-1$.

A detailed pseudo code for the $O(n)$ method can be found in Appendix B.

3.2. Time Complexity Analysis

In this section, we confirm the time complexity for the algorithm we discussed above is actually of order $O(n)$, using operation counts for each step of the algorithm. Table 1 and Table 2 show the time complexity analysis for each step in the initialization phase and the evaluation phase of the fast algorithm.

Table 1: Time complexity of initialization phase.

Step	Complexity	Explanation
1.	$O(k)$	A total of $3k$ Chebyshev nodes are computed.
2.	$O(k^2 + ks)$	$k(k-1)$ operations for calculating the denominators for the Chebyshev interpolation coefficients, $2ks$ operations for $u_r(i/s), i = 0, 1, \dots, s-1$, and $4k^2$ operations for $u_r(t'_i), i = 0, 1, \dots, 2k-1$.
3.	$O(nk^2/s)$	$3(n/s-2-h) \cdot k^2$ operations for two submatrices.
4.	$O(ns)$	$(n(3s+1) - s^2)/2$ operations for two submatrices.

If we consider both k and s constant, each step of the algorithm requires $O(n)$ operations. Therefore, the overall time complexity is of order $O(n)$. From the previous discussion, we know that $k \propto \log(1/\epsilon)$. How about s ? During the initialization phase, step 3 and step 4 dominates the computation time in that they require the evaluation of the M or L function while the previous two steps require constant number of operations regardless of the size of input. If we count the total number of evaluations of the M or L function, we get

$$t_{\text{init}} \approx \frac{3n}{s}k^2 + \frac{3}{2}ns + \frac{n}{2}. \quad (40)$$

Choosing s to minimize the evaluation time t_{init} yields

$$s = \sqrt{2} \cdot k = 1.414 \cdot k. \quad (41)$$

In the evaluation phase, multiplication dominates the computation time. Counting

Table 2: Time complexity of evaluation phase.

Step	Complexity	Explanation
5.	$O(nk)$	Computing $b_{1,j}^r$, for $j = 0, 1, \dots, n/s - 1$ and $r = 0, 1, \dots, k - 1$, at $O(s)$ multiplications each, and total nk multiplications.
6.	$O(nk^2/s)$	Computing $b_{l,j}^r$ for $1 < l \leq h$. Total number of multiplications is $2(n/s - 4) \cdot k^2$.
7.	$O(nk^2/s)$	Computing $c_{l,j}^r$. Total number of multiplications is $3(n/s - 4 - h) \cdot k^2$.
8.	$O(nk^2/s)$	Computing $d_{l,j}^r$. Total number of multiplications is $2(n/s - 4) \cdot k^2$.
9.	$O(nk + ns)$	Final computation of output vector. Total number of multiplications is $nk + (n(3s + 1) - s^2)/2$.

these operations, we obtain

$$t_{eval} \approx \frac{7n}{s}k^2 + \frac{3}{2}ns + 2nk + \frac{n}{2}. \quad (42)$$

Therefore, when

$$s = \sqrt{\frac{14}{3}} \cdot k = 2.160 \cdot k, \quad (43)$$

t_{eval} becomes minimum. This result is different from that in [1]. It agrees with the practical performance of our implementation.

3.3. Numerical Results

We have implemented the $O(n)$ algorithm for the matrix-vector multiplication in the C programming language for both single and double precision arithmetic. The results of the fast algorithm are compared to those produced by the direct multiplication of the matrix and the vector, which is an $O(n^2)$ method, in double precision. The measure of error was chosen to be the $\mathcal{L}^2$ -norm:

$$E = \sqrt{\sum_{i=0}^{n-1} (\gamma_i - \lambda_i)^2 / \sum_{i=0}^{n-1} \lambda_i^2}, \quad (44)$$

where $\tilde{\gamma}$ is the result of the fast algorithm and $\tilde{\lambda}$ is the result of the direct multiplication using double precision arithmetic. The components of the input vector were pseudorandom numbers on the interval $[0, 1]$. The number of Chebyshev nodes for the expansion of submatrix entries decides the precision of the computation. We chose k to be 8 for single precision and 18 for double precision as in [1]. The tests of our C implementation were run on a Sun4/490.

In order to confirm our analysis in § 3.2 for the choice of s , the order of the smallest

submatrix which decides the efficiency, an automatic program was written. Fig. 2 shows the behavior of s for single precision calculation, with input vector size as 16384 and $k = 8$. We can see that $s = 2^4 = 16$, which is $2k$, is the most efficient one. The diagram shows the evaluation times for different s 's, but the behavior of s is similar for the initialization time.⁴ Similar experiments confirm that $s = 32$ is the best choice for double precision calculation.

The comparison of the fast method and the direct method is shown in Table 3 and Table 4. In the forward direction the L matrix was used while in the backward direction the M matrix was used.

When testing the code, we also checked the maximum fractional error of the components of the output vector. The maximum fractional error is of the same order of magnitude as the $\mathcal{L}^2$ error for the backward direction, but it is three orders of magnitude larger than the $\mathcal{L}^2$ error for the forward direction, using the L matrix. This result was obtained by comparing the results of both the fast algorithm and the direct method using single precision arithmetic against that of the direct method using double precision arithmetic. The maximum fractional errors are comparable. Also the position of the occurrence of the maximum fractional error was searched. It varied with different input vectors of the same size. Therefore, we conclude that the fluctuation of the fractional errors of the components of the output vector is due to the characteristics of the matrix L and not to the matrix multiplication algorithm. Using higher precision

⁴Notice that only the size of input vector is needed for the initialization phase, so in practice, if we are doing lots of multiplications with the same input vector size, we can do the initialization once and store all the necessary results for the evaluation phase.

Table 3: Single precision computation.

<i>Input size</i>	$\mathcal{L}^2$ error		<i>Time(seconds)</i>		
	<i>Forward</i>	<i>Backward</i>	<i>Initial.</i>	<i>Eval.</i>	<i>Direct</i>
128	1.304E-07	1.135E-07	0.24	0.01	0.36
256	1.366E-07	1.363E-07	0.58	0.03	1.42
512	1.539E-07	1.607E-07	1.29	0.06	5.94
1024	1.565E-07	1.751E-07	2.81	0.11	22.97
2048	1.590E-07	2.022E-07	5.80	0.27	88.55
4096	1.594E-07	2.409E-07	11.94	0.53	357.19
8192	1.647E-07	2.804E-07	24.36	1.06	1462.59

Table 4: Double precision computation.

<i>Input size</i>	$\mathcal{L}^2$ error		<i>Time(seconds)</i>		
	<i>Forward</i>	<i>Backward</i>	<i>Initial.</i>	<i>Eval.</i>	<i>Direct</i>
256	8.520E-16	5.608E-15	1.03	0.04	1.42
512	1.829E-15	9.936E-15	2.53	0.10	5.94
1024	5.238E-15	2.190E-14	5.71	0.23	22.97
2048	1.220E-14	4.864E-14	12.26	0.46	88.55
4096	3.511E-14	1.129E-13	25.55	0.97	357.19
8192	7.509E-14	2.657E-13	52.09	1.97	1462.59

arithmetic corrects the problem for the input sizes of interest and gives the required precision for the output.

The implementation of the fast algorithm has also been combined with the fast cosine transform (of order $O(n \log n)$). Table 5 shows the maximum fractional difference and the $\mathcal{L}^2$ difference between the input vector and the output vector through a backward and then a forward Legendre transform. Again using a double precision calculation, the maximum fractional difference is within an acceptable range for single precision accuracy.

Table 5: Legendre transform using double precision arithmetic.

<i>Input size</i>	<i>error</i>	
	<i>maximum</i>	$\mathcal{L}^2$
256	2.214E-11	3.075E-13
512	1.500E-10	9.818E-13
1024	1.183E-09	3.232E-13
2048	1.216E-08	9.919E-12
4096	8.842E-07	3.337E-11
8192	6.267E-06	1.141E-10

4. Parallel Implementation

4.1. Description of the Parallel Implementation

In this section, we discuss our initial parallel implementation of the fast matrix multiplication. As mentioned before, our work has been focused on the implementation of a special case, in which the size of input vector is a power of 2. For the parallelization, we keep this assumption for the size of input vector. In addition, we assume that the number of processors working concurrently on the problem will also be a power of 2, within the range of $[1, n/(2s)]$. Computation work is equally divided among the processors except for the first and the last processors.

For the initialization part, the first two steps require a constant number of operations, no matter what the size of input vector is, and the results from these two steps, the Chebyshev interpolation coefficients, will be used multiple times in the evaluation part. Therefore all the processors are assigned to calculate these values and store them in their local memories for later use. The third step evaluates the values ${}_lM_{i,j}^{p,r}$ of the $k \times k$ matrix for each submatrix ${}_lS_{i,j}$ of the partition. We divide these submatrices on each level in columns but consider each set of three submatrices ${}_lS_{2i,2i+2}, {}_lS_{2i,2i+3}, {}_lS_{2i+1,2i+3}$ as indivisible.⁵ If there are more submatrix sets than the number of processors on a level, the sets are evenly distributed among the processors. Otherwise, the processors are evenly distributed for these submatrix sets; several processors work on the same set simultaneously and store the results in their local memories for later use. One should notice here that the top-left set on each level is actually outside the upper-triangular matrix. For these sets, we just assign 0's for the convenience of later calculation. Step 4 evaluates the entries of the unpartitioned part of the original matrix. We evenly distribute the unpartitioned part in rows to the processors and simply put 0's for the entries outside the matrix on the last portion in order to make later computation convenient.

For the evaluation part, we can actually calculate c 's right after we have obtained the results for b 's on the same level. Fig. 3 shows the last three levels of computation of $b_{i,j}^r$ and $c_{i,j}^r$, using 8 processors. All the numbers in the diagram are subindices j . As we can see from the diagram, the computation of b 's and c 's are evenly distributed among the processors and if there are not enough work for all the processors, we assign the same work to relevant processors and this eliminates communication between processors, which is much more expensive than the duplicated computation. But we cannot avoid communication entirely at this stage. We need communication between processors on the last two levels in order to obtain b 's on previous levels for further computation. And we must also send the first two $c_{i,j}^r$'s on each processor to its left neighbor for later calculation of $d_{i,j}^r$ on the latter. For the processor or the group of processors at the right end, we simply append 0's to the end of the c list since the last two c 's on each level are 0. And Fig. 4 shows the first three levels of computation of $d_{i,j}^r$ from the results of $c_{i,j}^r$, using 8 processors. Again if there is not enough work for all the processors, we assign the same work to relevant processors. Notice on level $l = h - 1$, each processor

⁵This is the reason why we have to make the assumption that the number of processors should be less than or equal to $n/(2s)$.

calculates a different d on that level. We must check the processor id in order to decide whether to use $u_i(t'_r)$ or to use $u_i(t'_{k+r})$. On the last step, each processor produces its portion of result for the output vector.

4.2. Numerical Results

A parallel code for the fast matrix-multiplication has been implemented in the C programming language for iPSC/860 parallel computer using the PICL communication library [4]. In order to compare the results of the sequential and the parallel implementations, the sequential code was modified to run on the iPSC/860. The maximum fractional differences between components of output vectors from parallel and sequential implementations are within the machine's precision for both single and double precision arithmetic.

The efficiencies of the parallel implementation over the sequential implementation for the initialization phase and the evaluation phase are shown in Fig.5 and Fig.6 respectively. The efficiency is defined as follows

$$E = \frac{T_{seq}}{T_{par} \cdot P}, \quad (45)$$

where T_{seq} and T_{par} are computation times for sequential and parallel implementations and P is the number of processors used for parallel execution. We can see from the diagrams that for a fixed input size, as number of processors increases, the efficiency decreases. From the discussion of parallel implementation, we know that as the number of processors increases, we introduce more and more redundant work among different processors, which causes the efficiency to drop. But for fixed number of processors, as the input size increases, we have a smaller percentage of redundant work among different processors, causing the efficiency to increase.

Figure 3: Parallelization of the last three levels of computation for b and c , using 8 processors.

-----> Communication Line

Null: Append 0's to the end of list on the last processor.

Figure 4: Parallelization of the first three levels of computation for d , using 8 processors.

5. Generalization

Although the fast matrix-vector multiplication algorithm discussed in the previous sections requires the size of the input vector to be a power of 2, there are ways to generalize the method for any kind of input. Two different generalization methods have been tried.

Suppose the size of input vector is n . The straight forward generalization method extends the size of the input vector to the nearest power of 2, n' , and appends 0's to the end of the input vector.⁶ We can then apply the fast algorithm discussed above to obtain the resulting vector of size n' . Since for either M or L matrix, the top left $n \times n$ portion is the same as long as the size of the matrix is larger than n , the first n components of the resulting vector of size n' produce the desired output vector of size n . The time complexity for this method is simply $O(n') \approx O(n)$.

Another generalization method is to partition the matrix from the top right corner, in the same fashion as the method described in § 3, stopping when we reach submatrices of size s . We then proceed as before, using the fast method to compute the partitioned part and calculate directly for the unpartitioned part close to the major diagonal. Caution must be taken when we partition the matrix. As the error analysis in [1] shows, the partitioned submatrices are required to be separate from the major diagonal. Therefore, for input vector of size between 2^p and $2^{p+1} - 1$, we can only have as many levels of partitioning as we do for input vector of size 2^p . This results in large unpartitioned portions of the original matrix when the size of the matrix is close to but less than a power of 2. Because of this, we expect this generalization method to be of order $O(n^2)$.

Both methods of generalization have been implemented. Table 6 shows the $\mathcal{L}^2$ errors of the results of both generalization methods using single precision calculation. Gene I refers to the straight forward method while gene II the other. The errors are comparable to those of the input sizes of powers of 2. Fig. 7 shows the comparison of the computation time of the direct method and the generalization methods. In the diagram, the platforms for gene I show approximately same amounts of computation time for input sizes between 2^p and $2^{p+1} - 1$. The computation time of gene II has similar tendency as that of the direct method, in that the amount of direct computation increases as the input size increases between 2^p and $2^{p+1} - 1$. Going from one power of 2 range to the next, the computation time drops sharply since the amount of unpartitioned part decreases at this point.

6. Conclusion

The fast matrix-vector multiplication algorithm discussed above is an $O(n)$ method. When the method is combined with a $O(n \log n)$ fast cosine transform, we have an $O(n \log n)$ method for the Legendre expansion. The straight forward generalization method is easy to implement and efficient for the computation.

⁶When implementing the method, programmer should beware that some compilers might not have exact binary representation for floating point number 0. Therefore, this method will introduce extra error.

Table 6: Single precision computation of generalization methods.

Input size	Gene I		Gene II	
	Forward	Backward	Forward	Backward
500	2.567E-07	2.898E-07	1.061E-07	3.031E-07
700	2.542E-07	4.004E-07	1.543E-07	3.564E-07
900	3.114E-07	4.043E-07	1.581E-07	3.983E-07
1100	3.549E-07	5.037E-07	2.846E-07	4.418E-07
1300	3.639E-07	5.215E-07	2.751E-07	5.368E-07
1500	4.023E-07	5.259E-07	2.354E-07	5.191E-07
1700	4.074E-07	5.856E-07	2.174E-07	5.405E-07
1900	4.461E-07	5.857E-07	1.911E-07	5.420E-07
2100	4.769E-07	5.930E-07	4.359E-07	6.453E-07
2300	5.003E-07	6.419E-07	4.094E-07	7.137E-07
2500	5.354E-07	6.471E-07	3.776E-07	7.775E-07

The work we have done leads toward the more general problem of a full spherical harmonic transform. Given α_{nm} for $n = 0, 1, \dots, N-1$ and $m = -n, -n+1, \dots, n$, evaluate the expression

$$f(\theta, \phi) = \sum_{n=0}^{N-1} \sum_{m=-n}^n \alpha_{nm} \cdot P_n^m(\cos \theta) \cdot e^{im\phi}, \quad (46)$$

at the points

$$\begin{aligned} \theta_i &= \frac{2i+1}{2N} \pi, \\ \phi_j &= \frac{j}{N} 2\pi, \end{aligned}$$

where $i, j = 0, 1, \dots, N-1$. Notice that the matrices M and L are functions of m . We have focused our work on the case $m = 0$. Some approximation properties of M_m and L_m need to be established in order to apply the same technique and incorporate multiple M_m and L_m into a single algorithm.

7. References

- [1] B. K. Alpert and V. Rokhlin. A fast algorithm for the evaluation of legendre expansions. *SIAM J. Sci. Stat. Comput.*, 12, No. 1:158–179, 1991.
- [2] R. L. Burden and J. D. Faires. *Numerical Analysis*. PWS-Kent, Boston, MA, 4 edition, 1989.
- [3] P. J. Davis. Gamma function and related functions. In M. Abramowitz and I. A. Stegun, editors, *Handbook of Mathematical Functions*, chapter 6, pages 253–293. Dover Publications, Inc., New York, 1972.
- [4] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. A user's guid to PICL: a portable instrumented communication library. Technical Report ORNL/TM-11616, Oak Ridge National Laboratory, August 1990.
- [5] I. S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, New York, 1980.
- [6] U. W. Hochstrasser. Orthogonal polynomials. In M. Abramowitz and I. A. Stegun, editors, *Handbook of Mathematical Functions*, chapter 22, pages 771–802. Dover Publications, Inc., New York, 1972.

Appendix

A. Derivation of Recursions for $b_{l,j}^r$ and $d_{l,j}^r$

Before we show the recursive relation of $b_{l,j}^r$, we emphasize that the Chebyshev nodes being used in the fast algorithm are on the interval $[0, 1]$, instead of $[-1, 1]$. For any given function $f \in C^{n+1}[a, b]$, we have the Lagrange expansion

$$f(t) \approx \sum_{i=0}^{k-1} u_i\left(\frac{t-a}{b-a}\right) \cdot f[a + (b-a)t_i], \quad (47)$$

where t_i for $i = 0, 1, \dots, k-1$ are the Chebyshev nodes on the interval $[0, 1]$. Therefore we have the following:

$$u_r(t) = \sum_{i=0}^{k-1} u_i(2t) \cdot u_r\left(\frac{t_i}{2}\right) \quad (t \in [0, \frac{1}{2}]), \quad (48)$$

and

$$u_r(t) = \sum_{i=0}^{k-1} u_i(2t-1) \cdot u_r\left(\frac{1+t_i}{2}\right) \quad (t \in [\frac{1}{2}, 1]). \quad (49)$$

If we check the above two equations at the Chebyshev nodes $t_i/2$ and $(t_i + 1)/2$, for $i = 0, 1, \dots, k-1$, respectively, we can confirm that they are not approximations but exact expressions. And we claim the following recursive relations

$$u_{r,j}^l = \sum_{i=0}^{k-1} u_r\left(\frac{t_i}{2}\right) \cdot u_{i,j}^{l-1} \quad (j = 0, 1, \dots, \frac{m}{2} - 1), \quad (50)$$

$$u_{r,j}^l = \sum_{i=0}^{k-1} u_r\left(\frac{1+t_i}{2}\right) \cdot u_{i,j-\frac{m}{2}}^{l-1} \quad (j = \frac{m}{2}, \frac{m}{2} + 1, \dots, m-1), \quad (51)$$

where $m = 2^{l-1} \cdot s$.

Here we show how to derive Eq.(51) as an example.

$$\begin{aligned} u_{r,j}^l &= u_r\left(\frac{j}{m}\right) = \sum_{i=0}^{k-1} u_i\left(\frac{2j}{m} - 1\right) \cdot u_r\left(\frac{1+t_i}{2}\right) \\ &= \sum_{i=0}^{k-1} u_i\left(\frac{j - \frac{m}{2}}{\frac{m}{2}}\right) \cdot u_r\left(\frac{1+t_i}{2}\right) \\ &= \sum_{i=0}^{k-1} u_{i,j-\frac{m}{2}}^{l-1} \cdot u_r\left(\frac{1+t_i}{2}\right). \end{aligned}$$

Check Eq.(27) for the definition of $u_{r,j}^l$ and notice that $m/2 \leq j < m$. Now we prove the recursive expression for $b_{l,j}^r$. The base case is trivial. For the recursive part, using

Eq. (28), we have

$$\begin{aligned}
 & \sum_{i=0}^{k-1} \left[u_r \left(\frac{t_i}{2} \right) \cdot b_{l-1,2j}^i + u_r \left(\frac{1+t_i}{2} \right) \cdot b_{l-1,2j+1}^i \right] \\
 &= \sum_{i=0}^{k-1} u_r \left(\frac{t_i}{2} \right) \cdot \sum_{j'=0}^{m-1} u_{i,j'}^{l-1} \cdot v_{j'+2jm} + \sum_{i=0}^{k-1} u_r \left(\frac{1+t_i}{2} \right) \cdot \sum_{j'=0}^{m-1} u_{i,j'}^{l-1} \cdot v_{j'+(2j+1)m} \\
 &= \sum_{j'=0}^{m-1} \sum_{i=0}^{k-1} u_r \left(\frac{t_i}{2} \right) \cdot u_{i,j'}^{l-1} \cdot v_{j'+2jm} + \sum_{j'=m}^{2m-1} \sum_{i=0}^{k-1} u_r \left(\frac{1+t_i}{2} \right) \cdot u_{i,j'-m}^{l-1} \cdot v_{j'+2jm} \\
 &= \sum_{j'=0}^{m-1} u_{r,j'}^l \cdot v_{j'+2jm} + \sum_{j'=m}^{2m-1} u_{r,j'}^l \cdot v_{j'+2jm} \\
 &= \sum_{j'=0}^{m'-1} u_{r,j'}^l \cdot v_{j'+jm'} \\
 &= b_{l,j}^r.
 \end{aligned}$$

Notice that in the proof above, $m = 2^{l-2} \cdot s$ and $m' = 2m$.

$d_{l,j}^r$ in the recursive definition, Eq. (38), is actually the partial sum of $a_{l,j}^{i'}$ in Eq. (33) from level h down to level l . If there is only one level of partition, i.e. $h = 1$, it is trivial. We just have to substitute Eq. (37) into Eq. (39). When $h > 1$, we consider the summation of $a_{l,j}^{i'}$'s on two adjacent levels l and $l+1$. Suppose the i th row of the matrix crosses the partition level l and $l+1$. In order to compute the component of output vector w_i , we have to take the sum of a 's on these two levels. Suppose $m = 2^{l-1} \cdot s$, $i' = i \bmod m$, $j = \lfloor i/(2m) \rfloor$, we have two cases to consider: $a_{l,2j}^{i'} + a_{l+1,j}^{i'}$ and $a_{l,2j+1}^{i'} + a_{l+1,j}^{i'+m}$. Here we only consider the second case. Using Eq. (33) and Eq. (51), we have

$$\begin{aligned}
 a_{l,2j+1}^{i'} + a_{l+1,j}^{i'+m} &= \sum_{r=0}^{k-1} u_{r,i'}^l \cdot c_{l,2j+1}^r + \sum_{p=0}^{k-1} u_{p,i'+m}^{l+1} \cdot c_{l+1,j}^p \\
 &= \sum_{r=0}^{k-1} u_{r,i'}^l \cdot c_{l,2j+1}^r + \sum_{p=0}^{k-1} \sum_{r=0}^{k-1} u_{r,i'}^l \cdot u_p \left(\frac{1+t_r}{2} \right) \cdot c_{l+1,j}^p \\
 &= \sum_{r=0}^{k-1} u_{r,i'}^l \cdot [c_{l,2j+1}^r + \sum_{p=0}^{k-1} u_p \left(\frac{1+t_r}{2} \right) \cdot c_{l+1,j}^p].
 \end{aligned}$$

Cascade the sum of $u_{r,i'}^l$ with level $l-1$ and the $c_{l+1,j}^p$ with level $l+2$, we can obtain the recursive relation of the summation of a 's. If the i th row of matrix does not cross partition levels $l > l_0$, we can tell that the results of $d_{l,j}^r$, for $l > l_0$, are 0 from the recursive relation since the last two $c_{l,j}^p$ on each level are 0. This is how we obtain the recursive relation for $d_{l,j}^r$.

B. Pseudo Code For $O(n)$ Method of Matrix-vector Multiplication ($\vec{w} = M\vec{v}$)

Comment[Most part of the pseudo code is drawn directly from [1]. Additional comments and modifications are added in order to clear the ambiguity in the original representation.]

Initialization Phase

Comment[Input to this phase is the size of input vector n .]

Set the number of Chebyshev nodes $k \approx \log(1/\epsilon)$, where ϵ is the desired precision. Set the smallest interval size $s \approx 2k$, instead of $s \approx 4k$ as in [1]. (See §3.2 for detail.)

Step 1.

Comment[Construct Chebyshev nodes $t_0, t_1, \dots, t_{k-1}$ on the interval $[0, 1]$, Chebyshev nodes $t'_0, t'_1, \dots, t'_{k-1}$ on the interval $[0, \frac{1}{2}]$, and Chebyshev nodes $t'_k, t'_{k+1}, \dots, t'_{2k-1}$ on the interval $[\frac{1}{2}, 1]$.]

```
do  $r = 0, \dots, k-1$ 
  set  $t_r = [1 - \cos((r + 0.5)/\pi/k)]/2$ 
  set  $t'_r = t_r/2$  and  $t'_{k+r} = (1 + t_r)/2$ 
end do
```

Step 2.

Comment[Evaluate the denominators in the expressions for the Chebyshev interpolation coefficients $u_0, u_1, \dots, u_{k-1}$.]

```
do  $r = 0, \dots, k-1$ 
  set  $den_r = \prod_{l=0, l \neq r}^{k-1} (t_r - t_l)$ 
end do
```

Comment[Evaluate the Chebyshev interpolation coefficients $u_0, u_1, \dots, u_{k-1}$ at the uniformly spaced nodes $0, 1/s, 2/s, \dots, (s-1)/s$. Note: be careful when testing code with odd k 's. The following code needs justification since l/s and t_r can both be $1/2$ for certain cases.]

```
do  $l = 0, \dots, s-1$ 
  set  $x = \prod_{r=0}^{k-1} (l/s - t_r)$ 
  do  $r = 0, \dots, k-1$ 
    set  $u_r(l/s) = [x/(l/s - t_r)]/den_r$ 
  enddo
enddo
```

Comment[Evaluate the Chebyshev interpolation coefficients $u_0, u_1, \dots, u_{k-1}$ at the Chebyshev nodes $t'_0, t'_1, \dots, t'_{2k-1}$.]

```

do  $l = 0, \dots, 2k - 1$ 
  set  $x = \prod_{r=0}^{k-1} (t'_l - t_r)$ 
  do  $r = 0, \dots, k - 1$ 
    set  $u_r(t'_l) = [x / (t'_l - t_r)] / den_r$ 
  enddo
enddo

```

Step 3.

Comment[Evaluate the values ${}_l M_{i,j}^{p,r}$ of the $k \times k$ matrix on each submatrix ${}_l S_{i,j}$ of the partition.]

```

 $h = \log_2(n/s) - 1$ 
do  $l = 1, \dots, h$ 
  do  $i = 0, \dots, n/(2^{l-1}s) - 3$ 
    do  $j = i + 2, \dots, i + 3 - i \bmod 2$ 
      do  $p = 0, \dots, k - 1$ 
        do  $r = 0, \dots, k - 1$ 
          Calculate  ${}_l M_{i,j}^{p,r}$  according to Eq. (29).
        enddo
      enddo
    enddo
  enddo
enddo

```

Step 4.

Comment[Evaluate entries in the unpartitioned part of the matrix, near the diagonal. Note the major difference from that in [1]. Beware that indices might fall out of the boundary of the matrix.]

```

do  $j = 0, \dots, n/s - 2$ 
  do  $m = 0, \dots, s - 1$ 
    do  $i = m, m + 1, \dots, 2s$ 
      Calculate  $M(m + js, i + js)$  using Eq. (21).
    enddo
  enddo
enddo
set  $j = n/s - 1$ 
do  $m = 0, \dots, s - 1$ 
  do  $i = m, m + 1, \dots, s$ 
    Calculate  $M(m + js, i + js)$  using Eq. (21).
  enddo
enddo

```

End of Initialization Phase

Evaluation Phase

Comment[Input to this phase is $\vec{v} = (v_0, v_1, \dots, v_{n-1})^T$.]

Step 5.

Comment[Evaluate the coefficients $b_{1,j}^r$ from the input vector $\vec{v}$ and the interpolation coefficients $u_r(i/s)$.]

```
do  $j = 0, \dots, n/s - 1$ 
  do  $r = 0, \dots, k - 1$ 
    set  $b_{1,j}^r = \sum_{i=0}^{s-1} u_r(i/s) \cdot v_{i+js}$ 
  enddo
enddo
```

Step 6.

Comment[Evaluate the coefficients $b_{l,j}^r$ for $l \geq 2$, which correspond to larger submatrix sizes, from the coefficients for smaller submatrix sizes and the interpolation coefficients $u_r(t'_i)$.]

```
do  $l = 2, \dots, h$ 
  do  $j = 0, \dots, n/(2^{l-1}s) - 1$ 
    do  $r = 0, \dots, k - 1$ 
      set  $b_{l,j}^r = \sum_{i=0}^{k-1} [u_r(t'_i) \cdot b_{l-1,2j}^i + u_r(t'_{k+i}) \cdot b_{l-1,2j+1}^i]$ 
    enddo
  enddo
enddo
```

Step 7.

Comment[Evaluate the coefficients $c_{l,j}^r$ from the values ${}_lM_{i,j}^{p,r}$ and the coefficients $b_{l,j}^r$. Note: do not forget to set the last two elements on each level to zero.]

```
do  $l = 1, \dots, h$ 
  do  $j = 0, \dots, n/(2^l s) - 2$ 
    do  $r = 0, \dots, k - 1$ 
      set  $c_{l,2j}^r = \sum_{i=0}^{k-1} [{}_lM_{2j,2j+2}^{r,i} \cdot b_{l,2j+2}^i + {}_lM_{2j,2j+3}^{r,i} \cdot b_{l,2j+3}^i]$ 
      set  $c_{l,2j+1}^r = \sum_{i=0}^{k-1} {}_lM_{2j+1,2j+3}^{r,i} \cdot b_{l,2j+3}^i$ 
    enddo
  enddo
  set  $j = n/(2^l s) - 1$ 
  set  $c_{l,2j}^r = c_{l,2j+1}^r = 0$  for  $r = 0, \dots, k - 1$ 
enddo
```

Step 8.

Comment[Evaluate the coefficients $d_{l,j}^r$ from the coefficients $c_{l,j}^r$ and the interpolation coefficients $u_r(t'_i)$. Note: the last two elements on each level are always zero, although they are calculated in the same way as others for convenience.]

```
set  $d_{h,j}^r = c_{h,j}^r$  for  $r = 0, \dots, k-1$  and  $j = 0, \dots, 3$ 
do  $l = h-1, \dots, 1$ 
  do  $j = 0, \dots, n/(2^l s) - 1$ 
    do  $r = 0, \dots, k-1$ 
      set  $d_{l,2j}^r = c_{l,2j}^r + \sum_{i=0}^{k-1} u_i(t'_r) \cdot d_{l+1,j}^i$ 
      set  $d_{l,2j+1}^r = c_{l,2j+1}^r + \sum_{i=0}^{k-1} u_i(t'_{k+r}) \cdot d_{l+1,j}^i$ 
    enddo
  enddo
enddo
```

Step 9.

Comment[Evaluate the final result $\vec{w} = (w_0, w_1, \dots, w_{n-1})^T$ from the coefficients $d_{1,j}^r$, the interpolation coefficients $u_r(m/s)$, the values of M near the diagonal, and the input vector $\vec{v}$. Note: check the boundary as in Step 4.]

```
do  $j = 0, \dots, n/s - 2$ 
  do  $m = 0, \dots, s-1$ 
    set  $w_{m+js} = \sum_{i=0}^{k-1} u_i(m/s) \cdot d_{1,j}^i$ 
    do  $i = m, m+1, \dots, 2s$ 
      set  $w_{m+js} = w_{m+js} + M(m+js, i+js) \cdot v_{i+js}$ 
    enddo
  enddo
enddo set  $j = n/s - 1$ 
do  $m = 0, \dots, s-1$ 
  set  $w_{m+js} = \sum_{i=0}^{k-1} u_i(m/s) \cdot d_{1,j}^i$ 
  do  $i = m, m+1, \dots, s$ 
    set  $w_{m+js} = w_{m+js} + M(m+js, i+js) \cdot v_{i+js}$ 
  enddo
enddo
```

End of Evaluation Phase

End of Algorithm

ORNL/TM-12285

INTERNAL DISTRIBUTION

1. B. R. Appleton	22-26. R. F. Sincovec
2-6. W. Chen	27. B. D. Semeraro
7-8. T. S. Darland	28. D. W. Walker
9. E. F. D'Azevedo	29-33. R. C. Ward
10. J. J. Dongarra	34. P. H. Worley
11. T. H. Dunigan	35. Central Research Library
12. J. B. Drake	36. ORNL Patent Office
13. W. R. Emanuel	37. K-25 Applied Technology
Library
14. B. F. Lyon	38. Y-12 Technical Library
15. C. E. Oliver	39. Laboratory Records - RC
16. R. E. Flanery	40-41. Laboratory Records Department
17-21. S. A. Raby	

EXTERNAL DISTRIBUTION

42. David C. Bader, Atmospheric and Climate Research Division, Office of Health and Environmental Research, Office of Energy Research, ER-76, U.S. Department of Energy, Washington, DC 20585
43. Clive Baillie, Physics Department, Campus Box 390, University of Colorado, Boulder, CO 80309
44. David H. Bailey, NASA Ames, Mail Stop 258-5, NASA Ames Research Center, Moffet Field, CA 94035
45. Marsha J. Berger, Courant Institute of Mathematical Sciences, 251 Merscer Street, New York, NY 10012
46. Mike Berry, Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301
47. John H. Bolstad, Lawrence Livermore National Laboratory, L-16, P. O. Box 808, Livermore, CA 94550
48. George Bourianoff, Superconducting Super Collider Laboratory, 2550 Beckleymeade Avenue, Suite 260, Dallas, TX 75237-3946
49. Joe Brandenburg, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
50. Professor Roger W. Brockett, Pierce Hall, 29 Oxford Street, Harvard University, Cambridge, MA 02138
51. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307
52. Thomas A. Callcott, Director, The Science Alliance Program, 53 Turner House, University of Tennessee, Knoxville, TN 37996

53. John Cavallini, Office of Scientific Computing, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585
54. Professor I-Liang Chern, Department of Mathematics, National Taiwan University, Taipei, Taiwan, R.O.C.
55. Ray Cline, Sandia National Laboratories, Livermore, CA 94550
56. Jean Côté, RPN, 2121 Transcanada Highway, Suite 508, Dorval, Quebec H9P 1J3, CANADA
57. William Dannevik, Lawrence Livermore National Laboratory, P. O. Box 808, L-16, Livermore, CA 94550
58. Craig Douglas, IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598-0218
59. Dr. Donald J. Dudziak, Department of Nuclear Engineering, 110B Burlington Engineering Labs, North Carolina State University, Raleigh, NC 27695-7909
60. John Dukowicz, Los Alamos National Laboratory, Group T-3, Los Alamos, NM 87545
61. Ian Foster, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
62. Geoffrey C. Fox, NPAC, 111 College Place, Syracuse University, Syracuse, NY 13244-4100
63. Dr. Rhys Francis, Div. of Information Technology, CSIRO, 723 Swanston Street, Carlton, Vic. 3053, AUSTRALIA
64. Paul O. Frederickson, ACL, MS B287, Los Alamos National Laboratory, Los Alamos, NM 87545
65. Gene Golub, Computer Science Department, Stanford University, Stanford, CA 94305
66. John Gustafson, 236 Wilhelm, Ames Laboratory, Iowa State University, Ames, IA 50011
67. Phil Gresho, Lawrence Livermore National Laboratory, L-262, P. O. Box 808, Livermore, CA 94550
68. William D. Gropp, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
69. James J. Hack, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
70. Michael T. Heath, Center for Supercomputing Research and Development, 305 Talbot Laboratory, University of Illinois, 104 South Wright Street, Urbana, IL 61801-2932
71. Michael Henderson, Los Alamos National Laboratory, Group T-3, Los Alamos, NM 87545
72. Dr. Fred Howes, Office of Scientific Computing, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585

73. Lennart Johnsson, Thinking Machines Inc., 245 First Street, Cambridge, MA 02142-1214
74. Dr. Gary Johnson, Office of Scientific Computing, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585
75. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
76. Kenneth Kennedy, Department of Computer Science, Rice University, P. O. Box 1892, Houston, Texas 77001
77. Tom Kitchens, ER-7, Applied Mathematical Sciences, Scientific Computing Staff, Office of Energy Research, Office G-437 Germantown, Washington, DC 20585
78. Dr. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815
79. Rich Loft, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
80. Frank McCabe, Department of Computing, Imperial college of Science and Technology, 180 Queens Gate, London SW7 2BZ, United Kingdom
81. Michael C. MacCracken, Lawrence Livermore National Laboratory, L-262, P. O. Box 808, Livermore, CA 94550
82. Robert Malone, C-DO/ACL, MS B287, Los Alamos National Laboratory, Los Alamos, NM 87545
83. Len Margolin, Los Alamos National Laboratory, Los Alamos, NM 87545
84. Hal Marshall, Laboratory for Scientific Computation, Rm. 271 Cooley Bld., University of Michigan, Ann Arbor, MI 48109-2104
85. Paul C. Messina, Mail Code 158-79, California Institute of Technology, 1201 E. California Blvd. Pasadena, CA 91125
86. Professor Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801
87. Dr. David Nelson, Director of Scientific Computing, ER-7, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20585
88. V. E. Oberacker, Department of Physics, Vanderbilt University, Box 1807, Station B, Nashville, TN 37235
89. Joseph Oliger, Computer Science Department, Stanford University, Stanford, CA 94305
90. James M. Ortega, Department of Applied Mathematics, Thornton Hall, University of Virginia, Charlottesville, VA 22901
91. Ron Peierls, Applied Mathematical Department, Brookhaven National Laboratory, Upton, NY 11973
92. Richard Pelz, Dept. of Mechanical and Aerospace Engineering, Rutgers University, Piscataway, NJ 08855-0909
93. Jesse Poore, Computer Science Department, University of Tennessee, Knoxville, TN 37996-1300

94. Andrew Priestley, Institute for Computational Fluid Dynamics, Reading University, Reading RG6 2AX, UK
95. Lee Riedinger, Director, The Science Alliance Program, University of Tennessee, Knoxville, TN 37996
96. Ahmed Samèh, University of Illinois at Urbana-Champaign, 469 CSRL, 1308 West Main St., Urbana, IL 61801
97. William C. Skamarock, 3973 Escuela Court, Boulder, CO 80301
98. Richard Smith, Los Alamos National Laboratory, Group T-3, Mail Stop B2316, Los Alamos, NM 87545
99. Peter Smolarkiewicz, National Center for Atmospheric Research, MMM Group, P. O. Box 3000, Boulder, CO 80307
100. Paul N. Swarztrauber, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
101. Jurgen Steppeler, DWD, Frankfurterstr 135, 6050 Offenbach, GERMANY
102. Rick Stevens, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
103. Robert G. Voigt, ICASE, MS 132-C, NASA Langley Research Center, Hampton, VA 23665
104. Mary F. Wheeler, Rice University, Department of Mathematical Sciences, P.O. Box 1892, Houston, TX 77251
105. David L. Williamson, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
106. Samuel Yee, Air Force Geophysics Lab, Department LYP, Hancom AFB, Bedford, MA 01731
107. Dr. John Boyd, Rutgers University, Institute for Marine Coastal Sciences, P. O. Box 231, New Brunswick, NJ 08903-0231
108. Dr. Clive Temperton, European Centre for Medium-Range Weather Forecasts, Shinfield Park, Reading, Berkshire RG2 9AX UK
109. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600
- 110-111. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831

**DATE
FILMED**

12 / 3 / 93

END