NPS-MA-95-007

NAVAL POSTGRADUATE SCHOOL
Monterey, California

PARALLEL SOLUTIONS OF TRIDIAGONAL AND PENTADIAGONAL SYSTEMS

by

Francis X. Giraldo
Beny Neta
C.P. Katti

Technical Report for Period
1 July 1995 - 30 September 1995

Approved for public release; distribution is unlimited.

Prepared for: Naval Postgraduate School
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

Marcia J. Evans, RADM, USN
Superintendent

Richard E. Elster
Provost

This report was prepared in conjunction with research conducted for the Naval Postgraduate School and funded by the Naval Postgraduate School.

This report was prepared by:

Francis X. Giraldo
NRC Research Associate

Beny Neta
Professor of Mathematics

C.P. Katti
J. Nehru University
SC & SS
New Delhi 10067
INDIA

Accesion For	
NTIS CRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

Reviewed by:

RICHARD FRANKE
Chairman

Released by:

PAUL J. MARTO
Dean of Research

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 13, 1995	Technical Report 1 Jul 1995 - 30 Sep 1995

4. TITLE AND SUBTITLE
Parallel Solutions of Tridiagonal and Pentadiagonal Systems

5. FUNDING NUMBERS

6. AUTHOR(S)
Francis X. Giraldo, Beny Neta and C. P. Katti

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-MA-95-007

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this report are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

An algorithm for the parallel solution of tridiagonal and pentadiagonal linear systems having nonzero elements at the top right and bottom left corners. Tridiagonal systems of this kind arise from the solution of two point boundary value problems with periodic boundary conditions. Pentadiagonal systems of this kind arise from e.g the approximation of the shallow water equations by the two-stage Galerkin method combined with a high accuracy compact approximation to the first derivative (Navon, 1983).

14. SUBJECT TERMS
an algorithm for the parallel solution of tridiagonal and pentadiagonal

15. NUMBER OF PAGES
38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

PARALLEL SOLUTIONS OF TRIDIAGONAL AND PENTADIAGONAL SYSTEMS

Francis X. Giraldo
NRC Research Associate
Naval Postgraduate School
Department of Mathematics
Monterey, CA 93943

Beny Neta
Naval Postgraduate School
Department of Mathematics
Code MA/Nd
Monterey, CA 93943

C. P. Katti
J. Nehru University
SC & SS
New Delhi 10067
INDIA

Abstract

We present an algorithm for the parallel solution of tridiagonal and pentadiagonal linear systems having nonzero elements at the top right and bottom left corners. Tridiagonal systems of this kind arise from the solution of two point boundary value problems with periodic boundary conditions. Pentadiagonal systems of this kind arise from e.g the approximation of the shallow water equations by the two-stage Galerkin method combined with a high accuracy compact approximation to the first derivative (Navon, 1983).

1. Introduction

In this paper, we develop an algorithm for the parallel solution of tridiagonal and pentadiagonal linear systems having nonzero elements at the top right and bottom left corners. This is a generalization of an algorithm due to Kowalik et al (1984) for tridiagonal systems. Such tridiagonal systems arise when approximating a class of two-point boundary value problems having periodic boundary conditions:

$$y''(t) = f(t, y(t)), \qquad 0 \leq t \leq 1, \tag{1}$$

$$y(0) = y(1), \tag{2}$$

$$y'(0) = y'(1). \tag{3}$$

It was shown by Katti (1995) that this problem leads to the system

$$Ay = d \tag{4}$$

where the matrix A is of the form

$$\begin{pmatrix} a_1 & c_1 & 0 & 0 & \cdots & p_1 \\ b_2 & a_2 & c_2 & 0 & \cdots & 0 \\ 0 & b_3 & a_3 & c_3 & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & b_{N-1} & a_{N-1} & c_{N-1} \\ q_1 & \cdots & 0 & 0 & b_N & a_N \end{pmatrix} \tag{5}$$

and the right hand side d is

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{N-1} \\ d_N \end{pmatrix}. \tag{6}$$

Pentadiagonal as well as tridiagonal systems appear when the two-stage Galerkin method combined with a high accuracy compact approximation to the first derivative is used for the approximation of the shallow water equations with

periodic boundary conditions (Navon, 1983). In this case, one has to solve a pentadiagonal system of the form

$$Bx = d \tag{7}$$

where the matrix B is given by

$$\begin{pmatrix}
a_1 & c_1 & b_1 & 0 & 0 & 0 & \cdots & \cdots & p_1 & q_1 \\
r_2 & a_2 & c_2 & b_2 & 0 & 0 & \cdots & \cdots & 0 & q_2 \\
s_3 & r_3 & a_3 & c_3 & b_3 & 0 & \cdots & \cdots & 0 & 0 \\
0 & s_4 & r_4 & a_4 & c_4 & b_4 & 0 & \cdots & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\
0 & \cdots & 0 & s_{N-4} & r_{N-4} & a_{N-4} & c_{N-4} & b_{N-4} & 0 & 0 \\
0 & 0 & \cdots & 0 & s_{N-3} & r_{N-3} & a_{N-3} & c_{N-3} & b_{N-3} & 0 \\
0 & 0 & 0 & \cdots & 0 & s_{N-2} & r_{N-2} & a_{N-2} & c_{N-2} & b_{N-2} \\
u_1 & 0 & 0 & 0 & \cdots & 0 & s_{N-1} & r_{N-1} & a_{N-1} & c_{N-1} \\
v_1 & v_2 & 0 & 0 & \cdots & \cdots & 0 & s_N & r_N & a_N
\end{pmatrix} \tag{8}$$

In the next section we describe the parallel algorithm for the direct solution of the tridiagonal system. The algorithm for the pentadiagonal system is described in section 3. Numerical experiments with both algorithms are reported in section 4. The two programs are attached as appendices.

2. Algorithm for Tridiagonal

In this section, we generalize the algorithm developed by Kowalik et al [1] for tridiagonal systems to the case where the matrix A is given by (5). We follow Kowalik in our description. Divide the N equations equally among the π processors (some may have 1 more equation than others). Let's assume for simplicity that each processor gets k equations. The first step is to eliminate b_j which are the elements below the diagonal. Each processor $1 \leq i \leq \pi$ eliminates b_j for $(i-1)k+2 \leq j \leq ik$.

For $j = (i-1)k+2, \ldots, ik$

$f_{(i-1)k+1} \leftarrow b_{(i-1)k+1}$ only for $i \neq 1$

Compute the multiplier $m_j = \frac{b_j}{a_{j-1}}$ and update

$f_j \leftarrow f_j - m_j f_{j-1}$ only for $i \neq 1$

$a_j \leftarrow a_j - m_j c_{j-1}$

$p_j \leftarrow p_j - m_j p_{j-1}$ only for $i = 1$

$d_j \leftarrow d_j - m_j d_{j-1}$

This process creates the new elements f_j and p_j, as can be seen in (9) for the case $\pi = 3$.

$$
\left(\begin{array}{cccccccccccc}
a_1 & c_1 & \cdots & 0 & 0 & \cdots & \cdots & 0 & \cdots & \cdots & p_1 \\
0 & a_2 & c_2 & & 0 & \cdots & \cdots & 0 & \cdots & \cdots & p_2 \\
\vdots & & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & a_k & c_k & 0 & \cdots & 0 & \cdots & \cdots & p_k \\
\hline
0 & 0 & \cdots & f_{k+1} & a_{k+1} & c_{k+1} & \cdots & 0 & \cdots & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \ddots & & \vdots & & & \vdots \\
\vdots & \vdots & \vdots & \vdots & & \ddots & \ddots & \vdots & & & \vdots \\
0 & 0 & \cdots & f_{2k} & 0 & \cdots & a_{2k} & c_{2k} & \cdots & \cdots & 0 \\
\hline
0 & 0 & \cdots & 0 & 0 & \cdots & f_{2k+1} & a_{2k+1} & c_{2k+1} & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & \cdots & f_{3k-1} & 0 & \cdots & a_{3k-1} & c_{3k-1} \\
q_1 & 0 & \cdots & 0 & 0 & \cdots & f_{3k} & 0 & \cdots & \cdots & a_{3k}
\end{array}\right) \tag{9}
$$

The second step is the elimination of c_j which are the elements above the diagonal. Each processor $1 \leq i \leq \pi$ eliminates c_j for $ik-2 \leq j \leq (i-1)k+1$.

For $j = ik-2, \ldots, (i-1)k+1$

$g_{ik-1} \leftarrow c_{ik-1}$

Compute the multiplier $m_j = \frac{c_j}{a_{j+1}}$ and update

$f_j \leftarrow f_j - m_j f_{j+1}$ only for $i \neq 1$

$g_j \leftarrow g_j - m_j g_{j+1}$

$p_j \leftarrow p_j - m_j p_{j+1}$ only for $i = 1$

$d_j \leftarrow d_j - m_j d_{j+1}$

Note that for each processor i, the elements c_{ik} remain and can only be eliminated by passing the first row of processor $i+1$. Upon completing these steps, the system becomes

$$\left(\begin{array}{cccccccccccc}
a_1 & 0 & \cdots & & g_1 & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & p_1 \\
0 & a_2 & 0 & & \vdots & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & p_2 \\
\vdots & \vdots & \ddots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & & a_{k-1} & g_{k-1} & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & & a_k & 0 & 0 & \cdots & g_k & 0 & \cdots & \cdots & p_k \\
\hline
0 & 0 & \cdots & & f_{k+1} & a_{k+1} & 0 & \cdots & g_{k+1} & 0 & \cdots & \cdots & 0 \\
\vdots & \vdots & \vdots & & \vdots & & \ddots & & \vdots & \vdots & & & \vdots \\
\vdots & \vdots & \vdots & & \vdots & 0 & \cdots & a_{2k-1} & g_{2k-1} & \vdots & & & \vdots \\
0 & 0 & \cdots & & f_{2k} & 0 & \cdots & 0 & a_{2k} & 0 & \cdots & \cdots & g_{2k} \\
\hline
0 & 0 & \cdots & & 0 & 0 & \cdots & & f_{2k+1} & a_{2k+1} & 0 & \cdots & g_{2k+1} \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & & \ddots & & \vdots \\
0 & 0 & \cdots & & 0 & 0 & \cdots & & f_{3k-1} & 0 & \cdots & a_{3k-1} & g_{3k-1} \\
q_1 & 0 & \cdots & & 0 & 0 & \cdots & & f_{3k} & 0 & \cdots & \cdots & a_{3k}
\end{array}\right) \tag{10}$$

If we now take the first equation from processor 1 and the last equation from each processor, we end up with a similar tridiagonal system with only $\pi + 1$ equations:

$$\begin{pmatrix} a_1 & g_1 & 0 & p_1 \\ f_k & a_k & g_k & 0 \\ 0 & f_{2k} & a_{2k} & g_{2k} \\ q_1 & 0 & f_{3k} & a_{3k} \end{pmatrix} \tag{11}$$

where f_k is created upon elimination of p_k using the first row. In general, the matrix is

$$\begin{pmatrix} a_1 & g_1 & 0 & \cdots & & p_1 \\ f_k & a_k & g_k & 0 & \cdots & 0 \\ 0 & f_{2k} & a_{2k} & g_{2k} & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \\ 0 & \cdots & \cdots & f_{(\pi-1)k} & a_{(\pi-1)k} & g_{(\pi-1)k} \\ q_1 & 0 & \cdots & 0 & f_{\pi k} & a_{\pi k} \end{pmatrix} \tag{12}$$

This system can be solved either serially or in parallel and is referred henceforth as the reduced system. The solution of this system is then broadcasted to all the processors. Upon receiving the solution of the reduced system, each processor i solves for the unknowns $x_{(i-1)k+1}, \cdots, x_{ik-1}$ in parallel by a straightforward backward substitution. Note that for processor 1, the term x_1 need not be solved for since it is already known by virtue of the solution of the reduced system.

3. Algorithm for Pentadiagonal

In this section, we develop an algorithm for the pentadiagonal system (7). There are a few differences between this algorithm and the previous one. First, the reduced system contains the first two equations from the first processor and the last two equations from each processor. Thus the reduced system is of order $2(\pi + 1)$. Second, each processor i requires two equations from processor $i + 1$ to eliminate certain entries above the diagonal (c_j and b_j).

We now give the details of the algorithm, assuming again $\pi = 3$ processors each containing k equations. The first step is to eliminate the entries below the diagonal, namely s_j and r_j. Each processor $1 \leq i \leq \pi$ performs this task on the equations $(i-1)k + 2 \leq j \leq ik - 1$.

For $j = (i-1)k + 2, \ldots, ik - 1$

$f1_{(i-1)k+1} \leftarrow s_{(i-1)k+1}$ only for $i \neq 1$

$f2_{(i-1)k+1} \leftarrow r_{(i-1)k+1}$ only for $i \neq 1$

$f2_{(i-1)k+2} \leftarrow s_{(i-1)k+2}$ only for $i \neq 1$

Compute the multiplier $m_j = \frac{r_j}{a_{j-1}}$ and update

$f1_j \leftarrow f1_j - m_j f1_{j-1}$ only for $i \neq 1$

$f2_j \leftarrow f2_j - m_j f2_{j-1}$ only for $i \neq 1$

$a_j \leftarrow a_j - m_j c_{j-1}$

$c_j \leftarrow c_j - m_j b_{j-1}$

$p_j \leftarrow p_j - m_j p_{j-1}$ only for $i = 1$

$q_j \leftarrow q_j - m_j q_{j-1}$ only for $i = 1$

$d_j \leftarrow d_j - m_j d_{j-1}$

Now compute the other multiplier $n_j = \frac{s_{j+1}}{a_{j-1}}$ and update

$f1_{j+1} \leftarrow f1_{j+1} - n_j f1_{j-1}$ only for $i \neq 1$

$f2_{j+1} \leftarrow f2_{j+1} - n_j f2_{j-1}$ only for $i \neq 1$

$r_{j+1} \leftarrow r_{j+1} - n_j c_{j-1}$

$a_{j+1} \leftarrow a_{j+1} - n_j b_{j-1}$

$p_{j+1} \leftarrow p_{j+1} - n_j p_{j-1}$ only for $i = 1$

$q_{j+1} \leftarrow q_{j+1} - n_j q_{j-1}$ only for $i = 1$

$d_{j+1} \leftarrow d_{j+1} - n_j d_{j-1}$

Note that for each processor i, the elements r_{ik} remain and are never eliminated. Because the semi-bandwidth of the matrix has been increased by one, then so does the number of equations that need to be passed between processors. The matrix B becomes

$$
\left(\begin{array}{cccccccccccc}
a_1 & c_1 & b_1 & \cdots & 0 & \cdots & \cdots & \cdots & 0 & \cdots & p_1 & q_1 \\
 & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
0 & \cdots & a_{k-1} & c_{k-1} & b_{k-1} & 0 & & \cdots & 0 & \cdots & p_{k-1} & q_{k-1} \\
0 & \cdots & r_k & a_k & c_k & b_k & 0 & \cdots & 0 & \cdots & p_k & q_k \\
\hline
0 & \cdots & f1_{k+1} & f2_{k+1} & a_{k+1} & c_{k+1} & b_{k+1} & \cdots & 0 & \cdots & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots & & & \vdots \\
0 & \cdots & f1_{2k-1} & f2_{2k-1} & 0 & \cdots & a_{2k-1} & c_{2k-1} & b_{2k-1} & 0 & \cdots & 0 \\
0 & \cdots & f1_{2k} & f2_{2k} & 0 & \cdots & r_{2k} & a_{2k} & c_{2k} & b_{2k} & 0 & \cdots \\
\hline
0 & 0 & \cdots & 0 & 0 & \cdots & f1_{2k+1} & f2_{2k+1} & a_{2k+1} & c_{2k+1} & b_{2k+1} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots \\
u_1 & 0 & \cdots & 0 & 0 & \cdots & f1_{3k-1} & f2_{3k-1} & 0 & \cdots & a_{3k-1} & c_{3k-1} \\
v_1 & v_2 & 0 & \cdots & 0 & \cdots & f1_{3k} & f2_{3k} & 0 & \cdots & \cdots & a_{3k}
\end{array}\right) \quad (13)
$$

The second step is to eliminate the entries above the diagonal, namely c_j and b_j. Each processor $1 \leq i \leq \pi$ performs this task on the equations $ik-3 \leq j \leq (i-1)k+1$.

For $j = ik-3, \ldots, (i-1)k+1$

$g1_{ik-3} \leftarrow b_{ik-3}$

$g1_{ik-2} \leftarrow c_{ik-2}$

$g2_{ik-2} \leftarrow b_{ik-2}$

Compute the multiplier $m_j = \frac{c_j}{a_{j+1}}$ and update

$f1_j \leftarrow f1_j - m_j f1_{j+1}$ only for $i \neq 1$

$f2_j \leftarrow f2_j - m_j f2_{j+1}$ only for $i \neq 1$

$g1_j \leftarrow g1_j - m_j g1_{j+1}$

$g2_j \leftarrow g2_j - m_j g2_{j+1}$

$p_j \leftarrow p_j - m_j p_{j+1}$ only for $i = 1$

$q_j \leftarrow q_j - m_j q_{j+1}$ only for $i = 1$

$d_j \leftarrow d_j - m_j d_{j+1}$

Now compute the other multiplier $n_j = \frac{b_{j-1}}{a_{j+1}}$ and update iff $j > (i-1)k+1$

$f1_{j-1} \leftarrow f1_{j-1} - n_j f1_{j+1}$ only for $i \neq 1$

$f2_{j-1} \leftarrow f2_{j-1} - n_j f2_{j+1}$ only for $i \neq 1$

$g1_{j-1} \leftarrow g1_{j-1} - n_j g1_{j+1}$

$g2_{j-1} \leftarrow g2_{j-1} - n_j g2_{j+1}$

$p_{j-1} \leftarrow p_{j-1} - n_j p_{j+1}$ only for $i = 1$

$q_{j-1} \leftarrow q_{j-1} - n_j q_{j+1}$ only for $i = 1$

$d_{j-1} \leftarrow d_{j-1} - n_j d_{j+1}$

Note that this process does not eliminate the elements b_{ik-1}, b_{ik}, and c_{ik} for each processor i. Thus we have the following system

$$
\left(\begin{array}{ccccccccccccccc}
a_1 & 0 & \cdots & g1_1 & g2_1 & 0 & \cdots & \cdots & \cdots & 0 & 0 & \cdots & \cdots & p_1 & q_1 \\
0 & a_2 & & g1_2 & g2_2 & 0 & \cdots & \cdots & \cdots & 0 & 0 & \cdots & \cdots & p_2 & q_2 \\
\vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
0 & 0 & \cdots & a_{k-1} & c_{k-1} & b_{k-1} & 0 & & \cdots & 0 & 0 & \cdots & \cdots & p_{k-1} & q_{k-1} \\
0 & 0 & \cdots & r_k & a_k & c_k & b_k & 0 & \cdots & 0 & 0 & \cdots & \cdots & p_k & q_k \\
\hline
0 & 0 & \cdots & f1_{k+1} & f2_{k+1} & a_{k+1} & 0 & \cdots & g1_{k+1} & g2_{k+1} & 0 & \cdots & \cdots & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \ddots & & \vdots & \vdots & \vdots & & & & \vdots \\
\vdots & \vdots & \cdots & \vdots & \vdots & 0 & \cdots & & a_{2k-1} & c_{2k-1} & b_{2k-1} & 0 & \cdots & \cdots & 0 \\
0 & 0 & \cdots & f1_{2k} & f2_{2k} & 0 & \cdots & & r_{2k} & a_{2k} & c_{2k} & b_{2k} & 0 & \cdots & 0 \\
\hline
0 & 0 & \cdots & \cdots & 0 & 0 & \cdots & & f1_{2k+1} & f2_{2k+1} & a_{2k+1} & 0 & \cdots & g1_{2k+1} & g2_{2k+1} \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & & \ddots & & \vdots & \vdots \\
b_{3k-1} & 0 & \cdots & \cdots & 0 & 0 & \cdots & & f1_{3k-1} & f2_{3k-1} & 0 & \cdots & & a_{3k-1} & c_{3k-1} \\
c_{3k} & b_{3k} & 0 & \cdots & 0 & 0 & \cdots & & f1_{3k} & f2_{3k} & 0 & \cdots & & r_{3k} & a_{3k}
\end{array}\right) . \tag{14}
$$

To eliminate b_{ik-1}, b_{ik} and c_{ik}, the i^{th} processor requires the first two rows of processor $i+1$. The elimination creates four new elements $g1_{ik-1}$, $g2_{ik-1}$, $g1_{ik}$, and $g2_{ik}$. By eliminating p_2 using row 1 and p_{k-1}, q_{k-1}, p_k, and q_k using rows 1 and 2 we then obtain the following reduced system

$$
\left(\begin{array}{cccccccc}
a_1 & c_1 & g1_1 & g2_1 & 0 & 0 & p_1 & q_1 \\
r_2 & a_2 & g1_2 & g2_2 & 0 & 0 & 0 & q_2 \\
f1_{k-1} & f2_{k-1} & a_{k-1} & c_{k-1} & g1_{k-1} & g2_{k-1} & 0 & 0 \\
f1_k & f2_k & r_k & a_k & g1_k & g2_k & 0 & 0 \\
0 & 0 & f1_{2k-1} & f2_{2k-1} & a_{2k-1} & c_{2k-1} & g1_{2k-1} & g2_{2k-1} \\
0 & 0 & f1_{2k} & f2_{2k} & r_{2k} & a_{2k} & g1_{2k} & g2_{2k} \\
b_{3k-1} & 0 & 0 & 0 & f1_{3k-1} & f2_{3k-1} & a_{3k-1} & c_{3k-1} \\
c_{3k} & b_{3k} & 0 & 0 & f1_{3k} & f2_{3k} & r_{3k} & a_{3k}
\end{array}\right) . \tag{15}
$$

Note that the elimination of p_2 only generates one new element r_2, while the elimination of the other p's and q's creates the four new entries $f1_{k-1}$, $f2_{k-1}$, $f1_k$, and $f2_k$. Note that an additional term c_1 has appeared but this term is zero and is only included for illustrating the general structure of the reduced system. This matrix is a block tridiagonal system with nonzero blocks at the top right and bottom left corners, i.e. a block form of (11). This system can be solved serially by the block version of the solver used for (11). The solution is then broadcasted to all the processors. Upon receiving this solution, each processor i solves for the unknowns $x_{(i-1)k+2}, \cdots, x_{ik-2}$ in parallel by a straightforward backward substitution. Note that for processor 1, the terms x_1 and x_2 need not be solved for since they are already known by virtue of the solution of the reduced system.

4. Numerical Experiments

In this section, we present the numerical experiments used to test the efficiency of the tridiagonal and pentadiagonal algorithms. For both algorithms, the number of processors π and equations per processor k are varied but the total number of equations remains fixed at $N = 10,000$. All runs are performed on a cluster of Sun4 workstations running PVM.

For the tridiagonal algorithm, the matrix shown in equation (5) is defined as

$$b_i = -1, \quad a_i = 2, \quad c_i = -1$$

and $d_i = 0$ for $i = 1, \ldots, N$ with the additional conditions that $d_1 = -N$ and $d_N = N$. The corner terms in equation (5) are defined as $p_1 = b_1$ and $q_1 = c_N$. This system then has the exact solution $x_i = i$ for $i = 1, \ldots N$. Such a matrix system arises from a finite difference or finite element centered discretization of the one-dimensional Laplacian operator with periodic boundary conditions.

Table 1 shows the timing results obtained for the tridiagonal algorithm using averages based on five consecutive runs. Results are tabulated for $\pi = 2, 4, 8$ and 10 processors.

Number of Processors	Number of Equations Per Processor	Parallel Time (seconds)	Serial Time (seconds)	Speedup	Efficiency (%)
2	5000	0.10	0.11	1.1	55.0
4	2500	0.22	0.11	0.5	12.5
8	1250	0.22	0.11	0.5	6.3
10	1000	0.12	0.11	0.9	9.0

Table 1: Timings for the parallel and serial versions of the **tridiagonal** algorithm. Speedup is defined as the time ratio between the serial and parallel algorithm. Efficiency is the ratio of speedup to the number of processors.

The serial tridiagonal algorithm used for the purpose of computing speedup rates is a partitioning algorithm. The linear system (3) to be solved can be written as

$$Ax = b$$

which can then be partitioned as such

$$A(x^1 + x_n x^2) = b - A_{i,n} x_n$$

and this system can be decomposed into the two corresponding systems

$$\begin{aligned} Ax^1 &= b \\ Ax^2 &= -A_{i,n}. \end{aligned}$$

These two systems can now be solved via an LU decomposition. This algorithm is the periodic variant of the Thomas algorithm which is a very fast tridiagonal solver. Table 1 shows that the parallel version is competitive with the serial algorithm especially for $\pi = 2$ and $\pi = 10$ processors. For $\pi = 4$ and $\pi = 8$ processors the parallel algorithm is deficient relative to its serial counterpart. Nonetheless, these results demonstrate that no significant gains are made by parallelizing the tridiagonal solver.

For the pentadiagonal algorithm, the matrix shown in equation (8) is defined as

$$s_i = -1, \quad r_i = -1, \quad a_i = 4, \quad c_i = -1, \quad b_i = -1$$

and $d_i = 0$ for $i = 1, \ldots, N$ with the additional conditions that $d_1 = -2N$, $d_2 = -N$, $d_{N-1} = N$ and $d_N = 2N$. The corner terms in equation (5) are defined as $p_1 = s_1$, $q_1 = r_1$, $q_2 = s_2$, $u_1 = b_{N-1}$, $v_1 = c_N$ and $v_2 = b_N$. This system then has the exact solution $x_i = i$ for $i = 1, \ldots N$. Such a matrix system arises from the finite difference centered discretization of the two-dimensional Laplacian operator with periodic boundary conditions.

Table 2 shows the timing results obtained for the pentadiagonal algorithm using averages based on five consecutive runs.

Number of Processors	Number of Equations Per Processor	Parallel Time (seconds)	Serial Time (seconds)	Speedup	Efficiency (%)
2	5000	0.11	0.22	2.0	100.0
4	2500	0.22	0.22	1.0	25.0
8	1250	0.22	0.22	1.0	12.5
10	1000	0.13	0.22	1.7	17.0

Table 2: Timings for the parallel and serial versions of the **pentadiagonal** algorithm. Speedup is defined as the time ratio between the serial and parallel algorithm. Efficiency is the ratio of speedup to the number of processors.

The serial pentadiagonal algorithm used for the purpose of computing speedup rates is the pentadiagonal version of the tridiagonal partitioning algorithm presented above. In this version the linear matrix system (8) can be written as

$$Bx = b$$

and partitioned as

$$B(x^1 + x_{n-1}x^2 + x_n x^3) = b - B_{i,n-1}x_{n-1} - B_{i,n}x_n$$

and this system can be decomposed into the three corresponding systems

$$Bx^1 = b$$

$$
\begin{aligned}
Bx^2 &= -B_{i,n-1} \\
Bx^3 &= -B_{i,n}.
\end{aligned}
$$

These three systems can now be solved via an LU decomposition. Table 2 shows that the parallel version does provide a significant gain over the serial algorithm for all of the number of processors studied. Once again, the major gains are achieved for $\pi = 2$ and $\pi = 10$ processors. In addition it is worth considering that the times for the parallel tridiagonal and pentadiagonal algorithms are almost identical. The communication is extremely detrimental to the overall performance of the tridiagonal algorithm, while for the pentadiagonal it begins to pay dividends. This study hints at the fact that as the bandwidth of the matrix is increased, the better the possible performance of this parallel algorithm versus its serial counterpart. The algorithm described is generalizable to any banded system. For a system with semi-bandwidth β , then β equations need to be passed between processors and the reduced system is of order $\beta(\pi + 1)$. Thus as the semi-bandwidth becomes sufficiently large, the majority of the communication time is spent on the actual transmission of information as opposed to the overhead incurred in communication calls.

5. Conclusions

An algorithm for tridiagonal and pentadiagonal matrices with nonzero elements at the top right and bottom left corners is presented. The algorithm is generalizable to higher banded systems but numerical studies are only performed for the tridiagonal and pentadiagonal cases. The numerical studies show that the communication overhead severely hurts the performance of the tridiagonal case. However, the results also show that gains are made in the pentadiagonal case and this trend points at the possibility of further gains when the bandwidth of the matrix is increased beyond the pentadiagonal case. The algorithm becomes a bit more cumbersome to implement but it does extend rather well to higher bandwidths. The number of equations that are required to be passed and the order of the reduced system increase at the rates β and $\beta(\pi+1)$, respectively, where β is the semi-bandwidth of the matrix.

Acknowledgements

This research was supported in part by the Naval Postgraduate School Research Program and the National Research Council. The third author was supported in part by the DST program under grant number DST/MS(I)-003/93.

References

1. J. S. Kowalik, R. E. Lord, and S. P. Kumar, Design and performance of algorithms for MIMD parallel computers, NATO ASI series, Vol. F7, High Speed Computation (J. S. Kowalik, ed.), Springer Verlag, Berlin, 1984.

2. I. M. Navon, A Numerov-Galerkin technique applied to a finite-element shallow-water equations model with enforced conservation of integral invariants and selective lumping, J. Comp. Phys., **52**, (1983), 313-339.

3. C. P. Katti, Highly Efficient Parallel Algorithms for Finite Difference Solution to Navier-Stokes Equation, Technical Report JNU-02-94, February 1994.

Appendix A

In this appendix we give the tridiagonal solver. The first code is the parent (master) followed by the child (slave). The tridiagonal solver for the reduced system is also given.

```
*---------------------------------------------------------------------*
*This is the PARENT of a program which solves the
*tridiagonal system A x = b with periodic boundary conditions using a
*Tridiagonal Decomposition Method
*as by B. Neta, C.P. Katti and F.X. Giraldo.
*Programmed by F.X. Giraldo on 7/95
*---------------------------------------------------------------------*
      program parent3
      include 'fpvm3.h'
      include 'param.h'
                                          !Global Arrays
      dimension a(imax), b(imax), c(imax), d(imax), x(imax)
      dimension a_p(nk), b_p(nk), c_p(nk), d_p(nk), xx(nk)
      dimension fhat(nprocs+1), ahat(nprocs+1)
      dimension ghat(nprocs+1), dhat(nprocs+1)
      integer tids(16)
      real taray(2)
      character arch*8, nodename*10

      npoin=nprocs*nk
      do i=1,npoin                        !Construct Linear System
         b(i)=-1
         a(i)=4
         c(i)=-1
         d(i)=4 + 2*(i-2)
      end do
      d(1)=-npoin + 2
      d(npoin)=3*npoin
                                          !Start PVM
      call pvmfmytid( mytid )
      call pvmfparent( iptid )
      nodename='child3.exe'
      arch='*'
                                    !Spawn Processors and distribute
                                    !the Linear System among processors
      do i=1,nprocs
         call pvmfspawn(nodename,pvmdefault,arch,1,tids(i),ierr)
```

```
      write(*,'(" Spawning Processor tids ",i2,1x,i10)')i,tids(i)
      if (ierr.ne.1) then
         write(*,'("ierr = ",i3)')ierr
         write(*,'("Error! Could not spawn process # ",i3)')i
         call pvmfexit(ierr)
         stop
      endif
      do j=1,nk
         b_p(j)=b( (i-1)*nk + j )
         a_p(j)=a( (i-1)*nk + j )
         c_p(j)=c( (i-1)*nk + j )
         d_p(j)=d( (i-1)*nk + j )
      end do

      msgtype=1
      call pvmfinitsend(pvmdefault,info)
      call pvmfpack(integer4,i,1,1,info)
      call pvmfpack(real4,b_p,nk,1,info)
      call pvmfpack(real4,a_p,nk,1,info)
      call pvmfpack(real4,c_p,nk,1,info)
      call pvmfpack(real4,d_p,nk,1,info)
      call pvmfsend(tids(i),msgtype,info)
   end do
                                    !Broadcast TIDS to all Processors
   msgtype=2
   call pvmfinitsend(pvmdefault,info)
   call pvmfpack(integer4,tids,16,1,info)
   call pvmfmcast(nprocs,tids,msgtype,info)
                                    !Construct Arrow System
   time1=dtime(taray)
   msgtype=4
   do i=1,nprocs
      call pvmfrecv(-1,msgtype,info)
      call pvmfunpack(integer4,iprocs,1,1,info)
      if (iprocs.eq.1) then                    !For i=1,2
call pvmfunpack(real4,p1,1,1,info)
         call pvmfunpack(real4,a1,1,1,info)
call pvmfunpack(real4,g1,1,1,info)
call pvmfunpack(real4,d1,1,1,info)
call pvmfunpack(real4,f2,1,1,info)
call pvmfunpack(real4,a2,1,1,info)
         call pvmfunpack(real4,g2,1,1,info)
         call pvmfunpack(real4,d2,1,1,info)
         fhat(1)=p1
```

```
      ahat(1)=a1
      ghat(1)=g1
      dhat(1)=d1
      fhat(2)=f2
      ahat(2)=a2
      ghat(2)=g2
      dhat(2)=d2
   else if (iprocs.gt.1.and.iprocs.lt.nprocs) then  !For i=3,NPROCS
      call pvmfunpack(real4,ff,1,1,info)
      call pvmfunpack(real4,aa,1,1,info)
      call pvmfunpack(real4,gg,1,1,info)
      call pvmfunpack(real4,dd,1,1,info)
      fhat(iprocs+1)=ff
      ahat(iprocs+1)=aa
      ghat(iprocs+1)=gg
      dhat(iprocs+1)=dd
   else if (iprocs.eq.nprocs) then                  !For i=NPROCS + 1
      call pvmfunpack(real4,ff,1,1,info)
      call pvmfunpack(real4,aa,1,1,info)
      call pvmfunpack(real4,cc,1,1,info)
      call pvmfunpack(real4,dd,1,1,info)
      fhat(nprocs+1)=ff
      ahat(nprocs+1)=aa
      ghat(nprocs+1)=cc
      dhat(nprocs+1)=dd
   endif
end do
                                  !Solve Tridiagonal Arrow System
call tridiag_periodic(dhat,fhat,ahat,ghat,nprocs+1)
                                 !Store Output in corresponding vector
x(1)=dhat(1)
x(nk)=dhat(2)
do i=3,nprocs+1
   x((i-1)*nk)=dhat(i)
end do
                                  !Back Substitute
                                  !Broadcast TIDS to all Processors
msgtype=5
call pvmfinitsend(pvmdefault,info)
call pvmfpack(real4,x,imax,1,info)
call pvmfmcast(nprocs,tids,msgtype,info)
                                       !Receive Local Sol from Children
msgtype=6
do i=1,nprocs
```

```
         call pvmfrecv(-1,msgtype,info)
         call pvmfunpack(integer4,iprocs,1,1,info)
         call pvmfunpack(real4,xx,nk,1,info)
         print*,' Receiving local solution from Processor= ',iprocs
         if (iprocs.eq.1) then
            do j=2, nk - 1
               x(j)=xx(j)
            end do
         else
            do j=1, nk - 1
               x((iprocs-1)*nk+j)=xx(j)
            end do
         endif
      end do

      time2=dtime(taray)
      write(*,'(" Total time in seconds = ",e12.4)')taray(1)+taray(2)

      write(*,'(" Storing Values " )')
      open(1,file='parent3.out')
      do i=1,npoin
         write(1,'(i7,1x,e16.8)')i,x(i)
      end do
      close(1)

      stop
      end
*---------------------------------------------------------------------*
*This is the CHILD of a program which solves the
*tridiagonal system A x = b with periodic boundary conditions using a
*Tridiagonal Decomposition Method
*as by B. Neta, C.P. Katti and F.X. Giraldo.
*Programmed by F.X. Giraldo on 7/95
*---------------------------------------------------------------------*
      program child3
      include 'fpvm3.h'
      include 'param.h'
                                          !Global Arrays
      dimension a(nk), b(nk), c(nk), d(nk), x(imax), xx(nk)
      dimension f(nk), g(nk), p(nk)
      integer tids(16)
                                          !Start PVM
      call pvmfmytid( mytid )
      call pvmfparent( mtid )
```

```
                                        !Receive Data from Parent
msgtype=1
call pvmfrecv(mtid,msgtype,info)
call pvmfunpack(integer4,iprocs,1,1,info)
call pvmfunpack(real4,b,nk,1,info)
call pvmfunpack(real4,a,nk,1,info)
call pvmfunpack(real4,c,nk,1,info)
call pvmfunpack(real4,d,nk,1,info)
                                        !Receive Broadcast
msgtype=2
call pvmfrecv(mtid,msgtype,info)
call pvmfunpack(integer4,tids,16,1,info)
                                        !Eliminate b's
if (iprocs.eq.1) then
   p(1)=b(1)
   do j=2, nk
      xl=b(j)/a(j-1)
      a(j)=a(j) - xl*c(j-1)
      p(j)=-xl*p(j-1)
      d(j)=d(j) - xl*d(j-1)
   end do
else
   f(1)=b(1)
   do j=2, nk
      xl=b(j)/a(j-1)
      f(j)=-xl*f(j-1)
      a(j)=a(j) - xl*c(j-1)
      d(j)=d(j) - xl*d(j-1)
   end do
endif
                                        !Eliminate c's
if (iprocs.eq.1) then
   g(nk-1)=c(nk-1)
   do j=nk - 2, 1, -1
      xl=c(j)/a(j+1)
      g(j)=-xl*g(j+1)
      p(j)=p(j) - xl*p(j+1)
      d(j)=d(j) - xl*d(j+1)
   end do
else
   g(nk-1)=c(nk-1)
   do j=nk - 2, 1, -1
      xl=c(j)/a(j+1)
      f(j)=f(j) - xl*f(j+1)
```

```
      g(j)=-xl*g(j+1)
      d(j)=d(j) - xl*d(j+1)
   end do
endif
                                   !Send variables from i to i-1
if (iprocs.gt.1) then
   msgtype=3
   call pvmfinitsend(pvmdefault,info)
   call pvmfpack(real4,f(1),1,1,info)
   call pvmfpack(real4,a(1),1,1,info)
   call pvmfpack(real4,g(1),1,1,info)
   call pvmfpack(real4,d(1),1,1,info)
   call pvmfsend(tids(iprocs-1),msgtype,info)
endif
                                   !Receive variables from i+1 to i
if (iprocs.lt.nprocs) then
   msgtype=3
   call pvmfrecv(tids(iprocs+1),msgtype,info)
   call pvmfunpack(real4,ff,1,1,info)
   call pvmfunpack(real4,aa,1,1,info)
   call pvmfunpack(real4,gg,1,1,info)
   call pvmfunpack(real4,dd,1,1,info)
                                   !Eliminate C_nk's
   xl=c(nk)/aa
   g(nk)=-xl*gg
   a(nk)=a(nk) - xl*ff
   d(nk)=d(nk) - xl*dd
endif
                              !For Processor 1 Only -- Multiply 1st
                              !Row by -(p_k/p_1) and add to Kth Row
if (iprocs.eq.1) then
   xl=p(nk)/p(1)
   f(nk)=-xl*a(1)
   a(nk)=a(nk) - xl*g(1)
   d(nk)=d(nk) - xl*d(1)
endif
                        !Construct Arrow System
msgtype=4
call pvmfinitsend(pvmdefault,info)
call pvmfpack(integer4,iprocs,1,1,info)
if (iprocs.eq.1) then                          !For i=1,2
   call pvmfpack(real4,p(1),1,1,info)
   call pvmfpack(real4,a(1),1,1,info)
   call pvmfpack(real4,g(1),1,1,info)
```

```
         call pvmfpack(real4,d(1),1,1,info)
         call pvmfpack(real4,f(nk),1,1,info)
         call pvmfpack(real4,a(nk),1,1,info)
         call pvmfpack(real4,g(nk),1,1,info)
         call pvmfpack(real4,d(nk),1,1,info)
      else if (iprocs.gt.1.and.iprocs.lt.nprocs) then  !For i=3,NPROCS
         call pvmfpack(real4,f(nk),1,1,info)
         call pvmfpack(real4,a(nk),1,1,info)
         call pvmfpack(real4,g(nk),1,1,info)
         call pvmfpack(real4,d(nk),1,1,info)
      else if (iprocs.eq.nprocs) then                  !For i=NPROCS + 1
         call pvmfpack(real4,f(nk),1,1,info)
         call pvmfpack(real4,a(nk),1,1,info)
         call pvmfpack(real4,c(nk),1,1,info)
         call pvmfpack(real4,d(nk),1,1,info)
      endif
      call pvmfsend(mtid,msgtype,info)
                                    !Back Substitute
                                    !Receive Arrow Solution
      msgtype=5
      call pvmfrecv(mtid,msgtype,info)
      call pvmfunpack(real4,x,imax,1,info)
                                    !Obtain Local Solution
      if (iprocs.eq.1) then
         do j=2, nk - 1
            xx(j)=( d(j) - g(j)*x(nk) - p(j)*x(nk*nprocs) )/a(j)
         end do
      else
         do j=1, nk - 1
            xx(j)=( d(j) - f(j)*x((iprocs-1)*nk) - g(j)*x(iprocs*nk) )
     $           /a(j)
         end do
      endif
                                       !Send Local Sol to Parent
      msgtype=6
      call pvmfinitsend(pvmdefault,info)
      call pvmfpack(integer4,iprocs,1,1,info)
      call pvmfpack(real4,xx,nk,1,info)
      call pvmfsend(mtid,msgtype,info)

      stop
      end
      end
*---------------------------------------------------------------------*
```

```
*This is the program which solves the
*reduced tridiagonal system A x = b with periodic boundary conditions
*Programmed by F.X. Giraldo on 7/95
*-------------------------------------------------------------------*
      subroutine tridiag_periodic(d,b,a,c,n)
      include 'param.h'
      dimension b(nprocs+1), a(nprocs+1), c(nprocs+1), d(nprocs+1)
      dimension gam2(nprocs+1)

      a(1)=1.0/a(1)
      gam2(1)=-b(1)*a(1)
      b(1)=d(1)*a(1)
      do i=2,n-1
         c(i-1)=c(i-1)*a(i-1)
         a(i)=1.0/( a(i) - b(i)*c(i-1) )
         gam2(i)=-b(i)*gam2(i-1)*a(i)
         b(i)=( d(i) - b(i)*b(i-1) )*a(i)
      end do
      gam2(n-1)=gam2(n-1) - c(n-1)*a(n-1)

      d(n-1)=b(n-1)
      a(n-1)=gam2(n-1)
      do i1=2,n-1
         i=n-i1
         i2=i+1
         d(i)=b(i) - c(i)*d(i2)
         a(i)=gam2(i) - c(i)*a(i2)
      end do

      i=n
      zaa=d(i) - c(i)*d(1) - b(i)*d(n-1)
      zaa=zaa/( a(i) + b(i)*a(n-1) + c(i)*a(1) )
      d(i)=zaa
      do i=1,n-1
         d(i)=d(i)+a(i)*zaa
      end do

      return
      end
```

Appendix B

In this appendix we give the pentadiagonal solver. The first code is the parent (master) followed by the child (slave). The pentadiagonal solver for the reduced system is also given.

```
*-------------------------------------------------------------------*
*This is the PARENT of a program which solves the
*pentadiagonal system A x = b with periodic boundary conditions using a
*Pentadiagonal Decomposition Method
*as by B. Neta, C.P. Katti and F. X. Giraldo.
*Programmed by F.X. Giraldo on 7/95
*-------------------------------------------------------------------*
      program parent5
      include 'fpvm3.h'
      include 'param.h'
c      external timing_fgettod
                                          !Global Arrays
      dimension s(imax), r(imax), a(imax), b(imax), c(imax)
      dimension d(imax), x(imax)
      dimension s_p(nk), r_p(nk), a_p(nk), b_p(nk), c_p(nk)
      dimension d_p(nk), x_p(nk)
      dimension f1hat(nhat), f2hat(nhat), rhat(nhat)
      dimension ahat(nhat), chat(nhat), g1hat(nhat)
      dimension g2hat(nhat), dhat(nhat), xhat(nhat)
      integer tids(nprocs), itime1(2), itime2(2), itotal
      real taray(2)
      character arch*8, nodename*10

      ichild=0
      npoin=nprocs*nk
      do i=1,npoin                        !Construct Linear System
         s(i)=1
         r(i)=1
         a(i)=4
         c(i)=1
         b(i)=1
         d(i)=8*i
      end do
      d(1)=2*npoin + 8
      d(2)=npoin + 16
      d(npoin-1)=7*npoin - 8
      d(npoin)=6*npoin
```

```
                                        !Start PVM
      call pvmfmytid( mytid )
      call pvmfparent( iptid )
      nodename='child5.exe'
      arch='*'
                                       !Spawn Processors and distribute
                                       !the Linear System among processors
      do i=1,nprocs
         call pvmfspawn(nodename,pvmdefault,arch,1,tids(i),ierr)
         write(*,'(" Spawning Processor tids ",i2,1x,i10)')i,tids(i)
         if (ierr.ne.1) then
            write(*,'("ierr = ",i3)')ierr
            write(*,'("Error! Could not spawn process # ",i3)')i
            call pvmfexit(ierr)
            stop
         endif
         do j=1,nk
            s_p(j)=s( (i-1)*nk + j )
            r_p(j)=r( (i-1)*nk + j )
            a_p(j)=a( (i-1)*nk + j )
            c_p(j)=c( (i-1)*nk + j )
            b_p(j)=b( (i-1)*nk + j )
            d_p(j)=d( (i-1)*nk + j )
         end do

         msgtype=1
         call pvmfinitsend(pvmdefault,info)
         call pvmfpack(integer4,i,1,1,info)
         call pvmfpack(real4,s_p,nk,1,info)
         call pvmfpack(real4,r_p,nk,1,info)
         call pvmfpack(real4,a_p,nk,1,info)
         call pvmfpack(real4,c_p,nk,1,info)
         call pvmfpack(real4,b_p,nk,1,info)
         call pvmfpack(real4,d_p,nk,1,info)
         call pvmfsend(tids(i),msgtype,info)
      end do
                                       !Broadcast TIDS to all Processors
c      call timing_fgettod(%REF(itime1))
      msgtype=2
      call pvmfinitsend(pvmdefault,info)
      call pvmfpack(integer4,tids,nprocs,1,info)
      call pvmfmcast(nprocs,tids,msgtype,info)
                                       !Construct Arrow System
      time1=dtime(taray)
```

```
msgtype=4
do i=1,nprocs
   call pvmfrecv(-1,msgtype,info)
   call pvmfunpack(integer4,iprocs,1,1,info)
   if (iprocs.eq.1) then                              !For i=1,2
      call pvmfunpack(real4,p1_1,1,1,info)
      call pvmfunpack(real4,p2_1,1,1,info)
      call pvmfunpack(real4,a_1,1,1,info)
      call pvmfunpack(real4,c_1,1,1,info)
      call pvmfunpack(real4,g1_1,1,1,info)
      call pvmfunpack(real4,g2_1,1,1,info)
      call pvmfunpack(real4,d_1,1,1,info)
      call pvmfunpack(real4,p2_2,1,1,info)
      call pvmfunpack(real4,r_2,1,1,info)
      call pvmfunpack(real4,a_2,1,1,info)
      call pvmfunpack(real4,g1_2,1,1,info)
      call pvmfunpack(real4,g2_2,1,1,info)
      call pvmfunpack(real4,d_2,1,1,info)
      call pvmfunpack(real4,f1_3,1,1,info)
      call pvmfunpack(real4,f2_3,1,1,info)
      call pvmfunpack(real4,a_3,1,1,info)
      call pvmfunpack(real4,c_3,1,1,info)
      call pvmfunpack(real4,g1_3,1,1,info)
      call pvmfunpack(real4,g2_3,1,1,info)
      call pvmfunpack(real4,d_3,1,1,info)
      call pvmfunpack(real4,f1_4,1,1,info)
      call pvmfunpack(real4,f2_4,1,1,info)
      call pvmfunpack(real4,r_4,1,1,info)
      call pvmfunpack(real4,a_4,1,1,info)
      call pvmfunpack(real4,g1_4,1,1,info)
      call pvmfunpack(real4,g2_4,1,1,info)
      call pvmfunpack(real4,d_4,1,1,info)
      p1hat=p1_1
      p2hat=p2_1
      ahat(1)=a_1
      chat(1)=c_1
      g1hat(1)=g1_1
      g2hat(1)=g2_1
      dhat(1)=d_1
      q2hat=p2_2
      rhat(2)=r_2
      ahat(2)=a_2
      g1hat(2)=g1_2
      g2hat(2)=g2_2
```

```
        dhat(2)=d_2
        f1hat(2*iprocs+1)=f1_3
f2hat(2*iprocs+1)=f2_3
        ahat(2*iprocs+1)=a_3
        chat(2*iprocs+1)=c_3
        g1hat(2*iprocs+1)=g1_3
        g2hat(2*iprocs+1)=g2_3
        dhat(2*iprocs+1)=d_3
        f1hat(2*iprocs+2)=f1_4
        f2hat(2*iprocs+2)=f2_4
        rhat(2*iprocs+2)=r_4
        ahat(2*iprocs+2)=a_4
        g1hat(2*iprocs+2)=g1_4
        g2hat(2*iprocs+2)=g2_4
        dhat(2*iprocs+2)=d_4
    else if (iprocs.gt.1.and.iprocs.lt.nprocs) then  !For i=3,NPROCS
        call pvmfunpack(real4,f1_1,1,1,info)
        call pvmfunpack(real4,f2_1,1,1,info)
        call pvmfunpack(real4,a_1,1,1,info)
        call pvmfunpack(real4,c_1,1,1,info)
        call pvmfunpack(real4,g1_1,1,1,info)
        call pvmfunpack(real4,g2_1,1,1,info)
        call pvmfunpack(real4,d_1,1,1,info)
        call pvmfunpack(real4,f1_2,1,1,info)
        call pvmfunpack(real4,f2_2,1,1,info)
        call pvmfunpack(real4,r_2,1,1,info)
        call pvmfunpack(real4,a_2,1,1,info)
        call pvmfunpack(real4,g1_2,1,1,info)
        call pvmfunpack(real4,g2_2,1,1,info)
        call pvmfunpack(real4,d_2,1,1,info)
        f1hat(2*iprocs+1)=f1_1
f2hat(2*iprocs+1)=f2_1
        ahat(2*iprocs+1)=a_1
        chat(2*iprocs+1)=c_1
        g1hat(2*iprocs+1)=g1_1
        g2hat(2*iprocs+1)=g2_1
        dhat(2*iprocs+1)=d_1
        f1hat(2*iprocs+2)=f1_2
        f2hat(2*iprocs+2)=f2_2
        rhat(2*iprocs+2)=r_2
        ahat(2*iprocs+2)=a_2
        g1hat(2*iprocs+2)=g1_2
        g2hat(2*iprocs+2)=g2_2
        dhat(2*iprocs+2)=d_2
```

```
   else if (iprocs.eq.nprocs) then                !For i=NPROCS + 1
        call pvmfunpack(real4,f1_1,1,1,info)
        call pvmfunpack(real4,f2_1,1,1,info)
        call pvmfunpack(real4,a_1,1,1,info)
        call pvmfunpack(real4,c_1,1,1,info)
        call pvmfunpack(real4,b_1,1,1,info)
        call pvmfunpack(real4,d_1,1,1,info)
        call pvmfunpack(real4,f1_2,1,1,info)
        call pvmfunpack(real4,f2_2,1,1,info)
        call pvmfunpack(real4,r_2,1,1,info)
        call pvmfunpack(real4,a_2,1,1,info)
        call pvmfunpack(real4,c_2,1,1,info)
        call pvmfunpack(real4,b_2,1,1,info)
        call pvmfunpack(real4,d_2,1,1,info)
        f1hat(2*iprocs+1)=f1_1
f2hat(2*iprocs+1)=f2_1
        ahat(2*iprocs+1)=a_1
        chat(2*iprocs+1)=c_1
        v1hat=b_1
        dhat(2*iprocs+1)=d_1
        f1hat(2*iprocs+2)=f1_2
        f2hat(2*iprocs+2)=f2_2
        rhat(2*iprocs+2)=r_2
        ahat(2*iprocs+2)=a_2
        u1hat=c_2
        u2hat=b_2
        dhat(2*iprocs+2)=d_2
     endif
  end do
                                  !Solve Tridiagonal Arrow System
  call pentadiag_periodic(f1hat,f2hat,rhat,ahat,chat,g1hat,
 $     g2hat,p1hat,p2hat,q2hat,v1hat,u1hat,u2hat,dhat,xhat)

                                 !Store Output in corresponding vector
  x(1)=xhat(1)
  x(2)=xhat(2)
  do i=1,nprocs
     x(i*nk-1)=xhat(2*i+1)
     x(i*nk)=xhat(2*i+2)
  end do
                                  !Back Substitute
                                  !Broadcast TIDS to all Processors
  msgtype=5
  call pvmfinitsend(pvmdefault,info)
```

```
      call pvmfpack(real4,x,imax,1,info)
      call pvmfmcast(nprocs,tids,msgtype,info)
                                    !Receive Local Solution from Children
      msgtype=6
      do i=1,nprocs
         call pvmfrecv(-1,msgtype,info)
         call pvmfunpack(integer4,iprocs,1,1,info)
         call pvmfunpack(real4,x_p,nk,1,info)
         call pvmfunpack(integer4,itime,1,1,info)
         ichild=ichild + itime
         print*,' Receiving solution Processor = ',iprocs
         if (iprocs.eq.1) then
            do j=3, nk - 2
               x(j)=x_p(j)
            end do
         else
            do j=1, nk - 2
               x((iprocs-1)*nk+j)=x_p(j)
            end do
         endif
      end do

c      call timing_fgettod(%REF(itime2))
c      iparent=(itime2(1)-itime1(1))*1000000 + itime2(2)-itime1(2)
c      print*,' Children Time in useconds= ',ichild
c      print*,' Parent Time in useconds  = ',iparent
c      print*,' Total Time in useconds   = ',ichild+iparent

      time2=dtime(taray)
      write(*,'(" Total time in seconds = ",e12.4)')taray(1)+taray(2)

      write(*,'(" Storing Values " )')
      open(1,file='parent5.out')
      do i=1,npoin
         write(1,'(i7,1x,e12.6)')i,x(i)
      end do
      close(1)

      stop
      end
*---------------------------------------------------------------------*
*This is the CHILD of a program which solves the
*pentadiagonal system A x = b with periodic boundary conditions using a
*Pentadiagonal Decomposition Method
```

```
*as by B. Neta, C.P. Katti and F. X. Giraldo
*Programmed by F.X. Giraldo on 7/95
*--------------------------------------------------------------------*
      program child5
      include 'fpvm3.h'
      include 'param.h'
c      external timing_fgettod
                                          !Global Arrays
      dimension s(nk), r(nk), a(nk), c(nk), b(nk), d(nk)
      dimension xg(imax), x(nk)
      dimension f1(nk), f2(nk), g1(nk), g2(nk), p1(nk), p2(nk)
      integer tids(nprocs), itime1(2), itime2(2), itotal
                                          !Start PVM
      call pvmfmytid( mytid )
      call pvmfparent( mtid )
                                          !Receive Data from Parent
      msgtype=1
      call pvmfrecv(mtid,msgtype,info)
      call pvmfunpack(integer4,iprocs,1,1,info)
      call pvmfunpack(real4,s,nk,1,info)
      call pvmfunpack(real4,r,nk,1,info)
      call pvmfunpack(real4,a,nk,1,info)
      call pvmfunpack(real4,c,nk,1,info)
      call pvmfunpack(real4,b,nk,1,info)
      call pvmfunpack(real4,d,nk,1,info)
                                          !Receive Broadcast
c      call timing_fgettod(%REF(itime1))
      msgtype=2
      call pvmfrecv(mtid,msgtype,info)
      call pvmfunpack(integer4,tids,nprocs,1,info)
                                          !Eliminate R and S
      if (iprocs.eq.1) then
         p1(1)=s(1)
         p2(1)=r(1)
 p2(2)=s(2)
         do j=2,nk-1
            xl=r(j)/a(j-1)
            a(j)=a(j) - xl*c(j-1)
            c(j)=c(j) - xl*b(j-1)
            p1(j)=p1(j) - xl*p1(j-1)
            p2(j)=p2(j) - xl*p2(j-1)
            d(j)=d(j) - xl*d(j-1)
            xl=s(j+1)/a(j-1)
            r(j+1)=r(j+1) - xl*c(j-1)
```

```
            a(j+1)=a(j+1) - xl*b(j-1)
            p1(j+1)=p1(j+1) - xl*p1(j-1)
            p2(j+1)=p2(j+1) - xl*p2(j-1)
            d(j+1)=d(j+1) - xl*d(j-1)
         end do
      else
f1(1)=s(1)
f2(1)=r(1)
f2(2)=s(2)
         do j=2,nk-1
            xl=r(j)/a(j-1)
            f1(j)=f1(j) - xl*f1(j-1)
            f2(j)=f2(j) - xl*f2(j-1)
            a(j)=a(j) - xl*c(j-1)
            c(j)=c(j) - xl*b(j-1)
            d(j)=d(j) - xl*d(j-1)
            xl=s(j+1)/a(j-1)
            f1(j+1)=f1(j+1) - xl*f1(j-1)
            f2(j+1)=f2(j+1) - xl*f2(j-1)
            r(j+1)=r(j+1) - xl*c(j-1)
            a(j+1)=a(j+1) - xl*b(j-1)
            d(j+1)=d(j+1) - xl*d(j-1)
         end do
      endif
                                                !Eliminate C and B
      g1(nk-3)=b(nk-3)
      g1(nk-2)=c(nk-2)
      g2(nk-2)=b(nk-2)
      if (iprocs.eq.1) then
         do j=nk - 3,1, -1
            xl=c(j)/a(j+1)
            g1(j)=g1(j) - xl*g1(j+1)
            g2(j)=g2(j) - xl*g2(j+1)
            p1(j)=p1(j) - xl*p1(j+1)
            p2(j)=p2(j) - xl*p2(j+1)
            d(j)=d(j) - xl*d(j+1)
   if (j.gt.1) then
               xl=b(j-1)/a(j+1)
               g1(j-1)=g1(j-1) - xl*g1(j+1)
               g2(j-1)=g2(j-1) - xl*g2(j+1)
               p1(j-1)=p1(j-1) - xl*p1(j+1)
               p2(j-1)=p2(j-1) - xl*p2(j+1)
               d(j-1)=d(j-1) - xl*d(j+1)
            endif
```

```
   end do
else
   do j=nk - 3,1, -1
      xl=c(j)/a(j+1)
      f1(j)=f1(j) - xl*f1(j+1)
      f2(j)=f2(j) - xl*f2(j+1)
      g1(j)=g1(j) - xl*g1(j+1)
      g2(j)=g2(j) - xl*g2(j+1)
      d(j)=d(j) - xl*d(j+1)
      if (j.gt.1) then
         xl=b(j-1)/a(j+1)
         f1(j-1)=f1(j-1) - xl*f1(j+1)
         f2(j-1)=f2(j-1) - xl*f2(j+1)
         g1(j-1)=g1(j-1) -xl*g1(j+1)
         g2(j-1)=g2(j-1) -xl*g2(j+1)
         d(j-1)=d(j-1) - xl*d(j+1)
      endif
   end do
endif
                                    !Send variables from i to i-1
if (iprocs.gt.1) then
   msgtype=3
   call pvmfinitsend(pvmdefault,info)
   call pvmfpack(real4,f1(1),1,1,info)
   call pvmfpack(real4,f2(1),1,1,info)
   call pvmfpack(real4,a(1),1,1,info)
   call pvmfpack(real4,g1(1),1,1,info)
   call pvmfpack(real4,g2(1),1,1,info)
   call pvmfpack(real4,d(1),1,1,info)
   call pvmfpack(real4,f1(2),1,1,info)
   call pvmfpack(real4,f2(2),1,1,info)
   call pvmfpack(real4,a(2),1,1,info)
   call pvmfpack(real4,g1(2),1,1,info)
   call pvmfpack(real4,g2(2),1,1,info)
   call pvmfpack(real4,d(2),1,1,info)
   call pvmfsend(tids(iprocs-1),msgtype,info)
endif
                                     !Receive variables from i+1 to i
if (iprocs.lt.nprocs) then
   msgtype=3
   call pvmfrecv(tids(iprocs+1),msgtype,info)
   call pvmfunpack(real4,f1_1,1,1,info)
   call pvmfunpack(real4,f2_1,1,1,info)
   call pvmfunpack(real4,a_1,1,1,info)
```

```
        call pvmfunpack(real4,g1_1,1,1,info)
        call pvmfunpack(real4,g2_1,1,1,info)
        call pvmfunpack(real4,d_1,1,1,info)
        call pvmfunpack(real4,f1_2,1,1,info)
        call pvmfunpack(real4,f2_2,1,1,info)
        call pvmfunpack(real4,a_2,1,1,info)
        call pvmfunpack(real4,g1_2,1,1,info)
        call pvmfunpack(real4,g2_2,1,1,info)
        call pvmfunpack(real4,d_2,1,1,info)
                                        !Eliminate c_(nk), b_(nk) and b_(nk-1)
        xl=c(nk)/a_1
r(nk)=r(nk) - xl*f1_1
a(nk)=a(nk) - xl*f2_1
g1(nk)=-xl*g1_1
g2(nk)=-xl*g2_1
d(nk)=d(nk) - xl*d_1
xl=b(nk)/a_2
r(nk)=r(nk) - xl*f1_2
a(nk)=a(nk) - xl*f2_2
g1(nk)=g1(nk) - xl*g1_2
g2(nk)=g2(nk) - xl*g2_2
d(nk)=d(nk) - xl*d_2
xl=b(nk-1)/a_1
a(nk-1)=a(nk-1) - xl*f1_1
c(nk-1)=c(nk-1) - xl*f2_1
g1(nk-1)=-xl*g1_1
g2(nk-1)=-xl*g2_1
d(nk-1)=d(nk-1) - xl*d_1
     endif
                                         !For Processor 1 Only
     if (iprocs.eq.1) then
        c(1)=0.0
                                        !Eliminate P1_2
        xl=p1(2)/p1(1)
        r(2)=-xl*a(1)
        g1(2)=g1(2) - xl*g1(1)
        g2(2)=g2(2) - xl*g2(1)
        p2(2)=p2(2) - xl*p2(1)
        d(2)=d(2) - xl*d(1)
                                         !Eliminate P1_nk-1
        xl=p1(nk-1)/p1(1)
        f1(nk-1)=-xl*a(1)
        a(nk-1)=a(nk-1) - xl*g1(1)
        c(nk-1)=c(nk-1) - xl*g2(1)
```

```
      p2(nk-1)=p2(nk-1) - xl*p2(1)
      d(nk-1)=d(nk-1) - xl*d(1)
                                        !Eliminate P1_nk
      xl=p1(nk)/p1(1)
      f1(nk)=-xl*a(1)
      r(nk)=r(nk) - xl*g1(1)
      a(nk)=a(nk) - xl*g2(1)
      p2(nk)=p2(nk) - xl*p2(1)
      d(nk)=d(nk) - xl*d(1)
                                        !Eliminate P2_nk-1
      xl=p2(nk-1)/p2(2)
      f1(nk-1)=f1(nk-1) - xl*r(2)
      f2(nk-1)=-xl*a(2)
      a(nk-1)=a(nk-1) - xl*g1(2)
      c(nk-1)=c(nk-1) - xl*g2(2)
      d(nk-1)=d(nk-1) - xl*d(2)
                                        !Eliminate P2_nk
      xl=p2(nk)/p2(2)
      f1(nk)=f1(nk) - xl*r(2)
      f2(nk)=-xl*a(2)
      r(nk)=r(nk) - xl*g1(2)
      a(nk)=a(nk) - xl*g2(2)
      d(nk)=d(nk) - xl*d(2)
   endif
                          !Construct Arrow System
   msgtype=4
   call pvmfinitsend(pvmdefault,info)
   call pvmfpack(integer4,iprocs,1,1,info)
   if (iprocs.eq.1) then                          !For i=1,2
      call pvmfpack(real4,p1(1),1,1,info)
      call pvmfpack(real4,p2(1),1,1,info)
      call pvmfpack(real4,a(1),1,1,info)
      call pvmfpack(real4,c(1),1,1,info)
      call pvmfpack(real4,g1(1),1,1,info)
      call pvmfpack(real4,g2(1),1,1,info)
      call pvmfpack(real4,d(1),1,1,info)
      call pvmfpack(real4,p2(2),1,1,info)
      call pvmfpack(real4,r(2),1,1,info)
      call pvmfpack(real4,a(2),1,1,info)
      call pvmfpack(real4,g1(2),1,1,info)
      call pvmfpack(real4,g2(2),1,1,info)
      call pvmfpack(real4,d(2),1,1,info)
      call pvmfpack(real4,f1(nk-1),1,1,info)
      call pvmfpack(real4,f2(nk-1),1,1,info)
```

```
      call pvmfpack(real4,a(nk-1),1,1,info)
      call pvmfpack(real4,c(nk-1),1,1,info)
      call pvmfpack(real4,g1(nk-1),1,1,info)
      call pvmfpack(real4,g2(nk-1),1,1,info)
      call pvmfpack(real4,d(nk-1),1,1,info)
      call pvmfpack(real4,f1(nk),1,1,info)
      call pvmfpack(real4,f2(nk),1,1,info)
      call pvmfpack(real4,r(nk),1,1,info)
      call pvmfpack(real4,a(nk),1,1,info)
      call pvmfpack(real4,g1(nk),1,1,info)
      call pvmfpack(real4,g2(nk),1,1,info)
      call pvmfpack(real4,d(nk),1,1,info)
   else if (iprocs.gt.1.and.iprocs.lt.nprocs) then  !For i=3,NPROCS
      call pvmfpack(real4,f1(nk-1),1,1,info)
      call pvmfpack(real4,f2(nk-1),1,1,info)
      call pvmfpack(real4,a(nk-1),1,1,info)
      call pvmfpack(real4,c(nk-1),1,1,info)
      call pvmfpack(real4,g1(nk-1),1,1,info)
      call pvmfpack(real4,g2(nk-1),1,1,info)
      call pvmfpack(real4,d(nk-1),1,1,info)
      call pvmfpack(real4,f1(nk),1,1,info)
      call pvmfpack(real4,f2(nk),1,1,info)
      call pvmfpack(real4,r(nk),1,1,info)
      call pvmfpack(real4,a(nk),1,1,info)
      call pvmfpack(real4,g1(nk),1,1,info)
      call pvmfpack(real4,g2(nk),1,1,info)
      call pvmfpack(real4,d(nk),1,1,info)
   else if (iprocs.eq.nprocs) then                  !For i=NPROCS+1
      call pvmfpack(real4,f1(nk-1),1,1,info)
      call pvmfpack(real4,f2(nk-1),1,1,info)
      call pvmfpack(real4,a(nk-1),1,1,info)
      call pvmfpack(real4,c(nk-1),1,1,info)
      call pvmfpack(real4,b(nk-1),1,1,info)
      call pvmfpack(real4,d(nk-1),1,1,info)
      call pvmfpack(real4,f1(nk),1,1,info)
      call pvmfpack(real4,f2(nk),1,1,info)
      call pvmfpack(real4,r(nk),1,1,info)
      call pvmfpack(real4,a(nk),1,1,info)
      call pvmfpack(real4,c(nk),1,1,info)
      call pvmfpack(real4,b(nk),1,1,info)
      call pvmfpack(real4,d(nk),1,1,info)
   endif
   call pvmfsend(mtid,msgtype,info)
                                    !Back Substitute
```

```
                                    !Receive Arrow Solution
      msgtype=5
      call pvmfrecv(mtid,msgtype,info)
      call pvmfunpack(real4,xg,imax,1,info)
                                    !Obtain Local Solution
      if (iprocs.eq.1) then
         do j=3, nk - 2
            x(j)=( d(j) - g1(j)*xg(nk-1) - g2(j)*xg(nk)
     $           - p1(j)*xg(nk*nprocs-1)  - p2(j)*xg(nk*nprocs) )/a(j)
         end do
      else
         do j=1, nk - 2
            x(j)=( d(j) - f1(j)*xg((iprocs-1)*nk-1)
     $          - f2(j)*xg((iprocs-1)*nk) - g1(j)*xg(iprocs*nk-1)
     $          - g2(j)*xg(iprocs*nk) )/a(j)
         end do
      endif
c      call timing_fgettod(%REF(itime2))
c      itotal=(itime2(1)-itime1(1))*1000000 + itime2(2)-itime1(2)
                                    !Send Local Solution to Parent
      msgtype=6
      call pvmfinitsend(pvmdefault,info)
      call pvmfpack(integer4,iprocs,1,1,info)
      call pvmfpack(real4,x,nk,1,info)
      call pvmfpack(integer4,itotal,1,1,info)
      call pvmfsend(mtid,msgtype,info)

      stop
      end
*-----------------------------------------------------------------------*
*This program solves a
*smaller pentadiagonal system A x = b with periodic boundary conditions
*using a matrix partitioning approach.
*Written on 7/95, by F. X. Giraldo
*-----------------------------------------------------------------------*
      subroutine pentadiag_periodic(f1hat,f2hat,rhat,ahat,chat,g1hat,
     $           g2hat,p1hat,p2hat,q2hat,v1hat,u1hat,u2hat,dhat,xhat)
      include 'param.h'
      dimension f1hat(nhat), f2hat(nhat), rhat(nhat), ahat(nhat)
      dimension chat(nhat), g1hat(nhat), g2hat(nhat), dhat(nhat)
      dimension xhat(nhat), d2hat(nhat), d3hat(nhat)
      dimension x1hat(nhat), x2hat(nhat), x3hat(nhat)
                                    !Forward Reduction
                                    !Forward Reduction
```

```
      d2hat(1)=-p1hat
      d2hat(nhat-3)=-g1hat(nhat-3)
      d2hat(nhat-2)=-g1hat(nhat-2)
      d3hat(1)=-p2hat
      d3hat(2)=-q2hat
      d3hat(nhat-3)=-g2hat(nhat-3)
      d3hat(nhat-2)=-g2hat(nhat-2)

      do i=1,nhat-5,2
                                          !Eliminate rhat_i+1
  xl=rhat(i+1)/ahat(i)
ahat(i+1)=ahat(i+1) - xl*chat(i)
g1hat(i+1)=g1hat(i+1) - xl*g1hat(i)
g2hat(i+1)=g2hat(i+1) - xl*g2hat(i)
dhat(i+1)=dhat(i+1) - xl*dhat(i)
d2hat(i+1)=d2hat(i+1) - xl*d2hat(i)
d3hat(i+1)=d3hat(i+1) - xl*d3hat(i)
                                          !Eliminate f1hat_i+2
        xl=f1hat(i+2)/ahat(i)
        f2hat(i+2)=f2hat(i+2) - xl*chat(i)
        ahat(i+2)=ahat(i+2) - xl*g1hat(i)
chat(i+2)=chat(i+2) - xl*g2hat(i)
dhat(i+2)=dhat(i+2) - xl*dhat(i)
d2hat(i+2)=d2hat(i+2) - xl*d2hat(i)
d3hat(i+2)=d3hat(i+2) - xl*d3hat(i)
                                          !Eliminate f1hat_i+3
        xl=f1hat(i+3)/ahat(i)
        f2hat(i+3)=f2hat(i+3) - xl*chat(i)
        rhat(i+3)=rhat(i+3) - xl*g1hat(i)
ahat(i+3)=ahat(i+3) - xl*g2hat(i)
dhat(i+3)=dhat(i+3) - xl*dhat(i)
d2hat(i+3)=d2hat(i+3) - xl*d2hat(i)
d3hat(i+3)=d3hat(i+3) - xl*d3hat(i)
                                          !Eliminate f2hat_i+2
        xl=f2hat(i+2)/ahat(i+1)
ahat(i+2)=ahat(i+2) - xl*g1hat(i+1)
chat(i+2)=chat(i+2) - xl*g2hat(i+1)
dhat(i+2)=dhat(i+2) - xl*dhat(i+1)
d2hat(i+2)=d2hat(i+2) - xl*d2hat(i+1)
d3hat(i+2)=d3hat(i+2) - xl*d3hat(i+1)
                                          !Eliminate f2hat_i+3
        xl=f2hat(i+3)/ahat(i+1)
rhat(i+3)=rhat(i+3) - xl*g1hat(i+1)
ahat(i+3)=ahat(i+3) - xl*g2hat(i+1)
```

```
dhat(i+3)=dhat(i+3) - xl*dhat(i+1)
d2hat(i+3)=d2hat(i+3) - xl*d2hat(i+1)
d3hat(i+3)=d3hat(i+3) - xl*d3hat(i+1)
     end do
                                        !Eliminate rhat_nhat-3
     i=nhat - 3
     xl=rhat(i+1)/ahat(i)
     ahat(i+1)=ahat(i+1) - xl*chat(i)
     g1hat(i+1)=g1hat(i+1) - xl*g1hat(i)
     g2hat(i+1)=g2hat(i+1) - xl*g2hat(i)
     dhat(i+1)=dhat(i+1) - xl*dhat(i)
     d2hat(i+1)=d2hat(i+1) - xl*d2hat(i)
     d3hat(i+1)=d3hat(i+1) - xl*d3hat(i)
                                        !Back Substitution
                                        !Back Substitution
                                        !Solve for NHAT-2 and NHAT-3
     i=nhat-2
     ai=1/ahat(i)
     x1hat(i)=ai*dhat(i)
     x2hat(i)=ai*d2hat(i)
     x3hat(i)=ai*d3hat(i)
     i=nhat-3
     ai=1/ahat(i)
     x1hat(i)=ai*( dhat(i) - chat(i)*x1hat(i+1) )
     x2hat(i)=ai*( d2hat(i) - chat(i)*x2hat(i+1) )
     x3hat(i)=ai*( d3hat(i) - chat(i)*x3hat(i+1) )
                                        !Solve for I=NHAT-4,...,1
     do i=nhat-5,1,-2
        k=i+1
        ai=1/ahat(k)
        x1hat(k)=ai*(dhat(k)-g1hat(k)*x1hat(k+1)-g2hat(k)*x1hat(k+2))
        x2hat(k)=ai*(d2hat(k)-g1hat(k)*x2hat(k+1)-g2hat(k)*x2hat(k+2))
        x3hat(k)=ai*(d3hat(k)-g1hat(k)*x3hat(k+1)-g2hat(k)*x3hat(k+2))
        k=i
        ai=1/ahat(k)
        x1hat(k)=ai*(dhat(k)-chat(k)*x1hat(k+1)-g1hat(k)*x1hat(k+2)-
    $           g2hat(k)*x1hat(k+3))
        x2hat(k)=ai*(d2hat(k)-chat(k)*x2hat(k+1)-g1hat(k)*x2hat(k+2)-
    $           g2hat(k)*x2hat(k+3))
        x3hat(k)=ai*(d3hat(k)-chat(k)*x3hat(k+1)-g1hat(k)*x3hat(k+2)-
    $           g2hat(k)*x3hat(k+3))
     end do
                                        !Solve for NHAT-1 and NHAT
     i=nhat-1
```

```
 aa1=v1hat*x2hat(1) + f1hat(i)*x2hat(i-2) + f2hat(i)*x2hat(i-1) +
$    ahat(i)
 cc1=v1hat*x3hat(1) + f1hat(i)*x3hat(i-2) + f2hat(i)*x3hat(i-1) +
$    chat(i)
 dd1=dhat(i) - v1hat*x1hat(1) - f1hat(i)*x1hat(i-2) -
$    f2hat(i)*x1hat(i-1)
 i=nhat
 bb2=u1hat*x2hat(1) + u2hat*x2hat(2) + f1hat(i)*x2hat(i-3) +
$    f2hat(i)*x2hat(i-2) + rhat(i)
 aa2=u1hat*x3hat(1) + u2hat*x3hat(2) + f1hat(i)*x3hat(i-3) +
$    f2hat(i)*x3hat(i-2) + ahat(i)
 dd2=dhat(i) - u1hat*x1hat(1) - u2hat*x1hat(2) -
$    f1hat(i)*x1hat(i-3) - f2hat(i)*x1hat(i-2)

 xl=bb2/aa1
 aa2=aa2 - xl*cc1
 dd2=dd2 - xl*dd1
 xhat(nhat)=dd2/aa2
 xhat(nhat-1)=( dd1 - cc1*xhat(nhat) )/aa1

 do i=1,nhat-2
    xhat(i)=x1hat(i) + xhat(nhat-1)*x2hat(i) + xhat(nhat)*x3hat(i)
 end do

 return
 end
```

Distribution List

	No. of copies
Director Defense Technology Information Center Cameron Station Alexandria, VA 22314	2
Dean of Research Code 08 Naval Postgraduate School Monterey, CA 93943	1
Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
Department of Mathematics Code MA Naval Postgraduate School Monterey, CA 93943	1
Professor F. X. Giraldo Code MA/Fg Naval Postgraduate School Monterey, CA 93943	10
Professor Beny Neta Code MA/Nd Naval Postgraduate School Monterey, CA 93943	10
Professor C. P. Katti SC&SS Jawaharlal Nehru University New Delhi, 110067 INDIA	10

Lt. Chris Sagovac, USN
819 South Charles St.
Baltimore, MD 21230 1

Professor Zahari Zlatev
Department of Emissions and Air Pollution
National Environmental Res. Inst.
Frederiksborgvej 399
P. O. Box 358
DK-4000 Roskilde
DENMARK 1