

Parallel Solution of Pentadiagonal Systems Using Generalized Odd-Even Elimination

CREON LEVIT
 NASA Ames Research Center
 Mail Stop T045-1
 Moffett Field, CA. 94035

Abstract

A new method for the solution of pentadiagonal systems of linear equations is presented. The method is a generalization of ordinary odd-even elimination used for tridiagonal systems. Using n processors, an $n \times n$ pentadiagonal system can be solved using the new method (generalized odd-even elimination) in time proportional to $\log_2 n$.

1 Introduction

While much has been written on the parallel solution of tridiagonal systems of linear equations (see [1, 2, 3, 4, 5] for surveys), comparatively little work [6, 7] has been done on the parallel solution of pentadiagonal systems. Nevertheless, pentadiagonal matrices are an important form of banded matrix, second only perhaps to the tridiagonal and block-tridiagonal forms. Pentadiagonal matrices arise, for example, in the numerical solution of partial differential equations when implicit methods are used in conjunction with spatial differencing operators that have 5-point stencils. Thus they are important in computational fluid dynamics[8].

We begin with a review of odd-even elimination as it is normally applied in solving tridiagonal linear systems. Then, we present a generalization of odd-even elimination for solving pentadiagonal linear systems. We report on an initial implementation of the new method on a parallel processor.

2 Tridiagonal Odd-Even Elimination

Odd-even elimination (also known as parallel cyclic reduction) is a method for solving tridiagonal systems of linear

equations [1, 2, 3]. It is conceptually simple and naturally parallel. Briefly, we wish to solve the system

$$Ax = y \tag{1}$$

where

$$A = \begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & 0 \\ & & \ddots & & \\ 0 & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix}. \tag{2}$$

Consider row i of A :

$$A(i) = (0, \dots, a_i, b_i, c_i, 0, \dots). \tag{3}$$

We add to row $A(i)$ a multiple of row $A(i-1)$ chosen such as to annihilate a_i in $A(i)$. Doing so creates one element of fill-in directly to the left of a_i . We also add to $A(i)$ a multiple of $A(i+1)$ chosen such as to annihilate c_i in $A(i)$. This generates one element of fill-in to the immediate right of c_i .

Thus we form

$$A'(i) = \frac{-a_i}{b_{i-1}}A(i-1) + A(i) + \frac{-c_i}{b_{i+1}}A(i+1) \tag{4}$$

$$= (0, \dots, a'_i, 0, b'_i, 0, c'_i, 0, \dots). \tag{5}$$

If we perform the above operation in parallel to all the rows of A , we produce a new matrix

$$A' = \begin{pmatrix} b'_1 & 0 & c'_1 & & & \\ 0 & b'_2 & 0 & c'_2 & & 0 \\ a'_3 & 0 & b'_3 & 0 & c'_3 & \\ & & & \ddots & & \\ & & & & a'_{n-2} & 0 & b'_{n-2} & 0 & c'_{n-2} \\ & 0 & & & a'_{n-1} & 0 & b'_{n-1} & 0 & \\ & & & & & a'_n & 0 & b'_n & \end{pmatrix}. \tag{6}$$

We have thus reduced the original tridiagonal system into two independent tridiagonal systems of half the original size¹.

¹This can be seen by permuting the rows and columns of A'

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

This process is then iterated for $k = 1, 2, \dots, \lceil \log_2 n \rceil - 1$, with $i - 2^k$ and $i + 2^k$ replacing $i - 1$ and $i + 1$ respectively in equation (4). After the last iteration A has been transformed into a strictly diagonal matrix A^D .

For example (after [1]), starting with an 8×8 tridiagonal matrix and denoting any nonzero elements with a "■", we have:

$$A = \begin{pmatrix} \blacksquare & \blacksquare & & & & & & \\ & \blacksquare & \blacksquare & & & & & \\ & & \blacksquare & \blacksquare & & & & \\ & & & \blacksquare & \blacksquare & & & \\ & & & & \blacksquare & \blacksquare & & \\ & & & & & \blacksquare & \blacksquare & \\ & & & & & & \blacksquare & \blacksquare \\ & & & & & & & \blacksquare \end{pmatrix} \quad A' = \begin{pmatrix} \blacksquare & & & & & & & \\ & \blacksquare & & & & & & \\ & & \blacksquare & & & & & \\ & & & \blacksquare & & & & \\ & & & & \blacksquare & & & \\ & & & & & \blacksquare & & \\ & & & & & & \blacksquare & \\ & & & & & & & \blacksquare \end{pmatrix}$$

$$A'' = \begin{pmatrix} \blacksquare & & & & & & & \\ & \blacksquare & & & & & & \\ & & \blacksquare & & & & & \\ & & & \blacksquare & & & & \\ & & & & \blacksquare & & & \\ & & & & & \blacksquare & & \\ & & & & & & \blacksquare & \\ & & & & & & & \blacksquare \end{pmatrix} \quad A^D = \begin{pmatrix} \blacksquare & & & & & & & \\ & \blacksquare & & & & & & \\ & & \blacksquare & & & & & \\ & & & \blacksquare & & & & \\ & & & & \blacksquare & & & \\ & & & & & \blacksquare & & \\ & & & & & & \blacksquare & \\ & & & & & & & \blacksquare \end{pmatrix}$$

By including, in the standard manner, the right hand side y in all row operations, we produce the solution vector x after a final, fully parallelizable division of the transformed y by the respective diagonal elements of A^D .

To consistently handle references to elements "off the edges" of the matrix (i.e., where $i - 2^k < 1$ or $i + 2^k > n$) we can use:

$$\left. \begin{array}{l} a_j = 0 \\ b_j = 1 \\ c_j = 0 \\ y_j = 0 \end{array} \right\} j < 1 \text{ or } j > n \quad (7)$$

Stage k ($0 \leq k \leq \lceil \log_2 n \rceil - 1$) of the odd-even elimination can be performed in the following manner, where \leftarrow denotes the replacement operator. Let $m = 2^k$.

$$\begin{aligned} a_i &\leftarrow a_i/b_i \\ c_i &\leftarrow c_i/b_i \\ y_i &\leftarrow y_i/b_i \\ b_i &\leftarrow 1 - a_i c_{i-m} - c_i a_{i+m} \\ y_i &\leftarrow y_i - a_i y_{i-m} - c_i y_{i+m} \\ a_i &\leftarrow -a_i a_{i-m} \\ c_i &\leftarrow -c_i c_{i+m} \end{aligned} \quad (8)$$

This algorithm solves the linear system $Ax = y$ in $\log_2 n$ stages with $13n$ floating-point operations per stage, plus one floating-point divide in each processor for the final solve of $A^D = y$, giving a total of $(13 \log_2 n + 1)n$ floating-point operations in $\log_2 n + 1$ stages. Standard Gaussian elimination (without pivoting) would require $8n - 7$ floating-point operations in $2n - 1$ stages.

Since all interprocessor communication is between rows that are an exact power of two apart, on hypercube connected multiprocessor architectures (for example, on the Connection Machine [9]), the interprocessor communication time can be kept relatively low by using a binary reflected grey code to map row numbers to processor numbers [10].

For each row, we need to store its a_i, b_i, c_i, y_i , and two "stack" temporaries for intermediate expressions. Thus, the total storage required to (destructively) solve the $n \times n$ tridiagonal system $Ax = y$ is $6n$ words.

Unfortunately, if at any time one of the b_i 's becomes zero, even though the system may not be singular, the algorithm will attempt to divide by zero. However, in practice, for random tridiagonal matrices and especially for the types of tridiagonal matrices generated by finite differencing in computational fluid dynamics problems, the algorithm is reasonably well behaved.

3 Generalized Odd-Even Elimination for Pentadiagonal Systems

Assume that we have a pentadiagonal system

$$P = \begin{pmatrix} c_1 & d_1 & e_1 & & & & & \\ b_2 & c_2 & d_2 & e_2 & & & & 0 \\ a_3 & b_3 & c_3 & d_3 & e_3 & & & \\ & & & \ddots & & & & \\ & & & & a_{n-2} & b_{n-2} & c_{n-2} & d_{n-2} & e_{n-2} \\ & 0 & & & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} & \\ & & & & & a_n & b_n & c_n & \end{pmatrix} \quad (9)$$

In the spirit of the transformation $A \rightarrow A'$ above, we wish to transform P , via elementary row operations that can be performed in parallel, into a new matrix P' of the form:

$$P' = \begin{pmatrix} c'_1 & 0 & d'_1 & 0 & e'_1 & & & & \\ 0 & c'_2 & 0 & d'_2 & 0 & e'_2 & & & \\ b'_3 & 0 & c'_3 & 0 & d'_3 & 0 & e'_3 & & 0 \\ 0 & b'_4 & 0 & c'_4 & 0 & d'_4 & 0 & e'_4 & \\ a'_5 & 0 & b'_5 & 0 & c'_5 & 0 & d'_5 & 0 & e'_5 \\ & & & \ddots & & & & & \\ & & & & a'_{n-4} & 0 & b'_{n-4} & 0 & c'_{n-4} & 0 & d'_{n-4} & 0 & e'_{n-4} \\ & & & & a'_{n-3} & 0 & b'_{n-3} & 0 & c'_{n-3} & 0 & d'_{n-3} & 0 & \\ & 0 & & & a'_{n-2} & 0 & b'_{n-2} & 0 & c'_{n-2} & 0 & d'_{n-2} & \\ & & & & a'_{n-1} & 0 & b'_{n-1} & 0 & c'_{n-1} & 0 & \\ & & & & & a'_n & 0 & b'_n & 0 & c'_n & \end{pmatrix} \quad (10)$$

This will effectively decouple the original system into two independent pentadiagonal systems of half the original size².

To accomplish the transformation $P \rightarrow P'$, consider row i of P .

$$P(i) = (0, \dots, a_i, b_i, c_i, d_i, e_i, 0, \dots). \quad (11)$$

Using a linear combination of other rows, we need to transform $P(i)$ into

$$P'(i) = (0, \dots, a'_i, 0, b'_i, 0, c'_i, 0, d'_i, 0, e'_i, 0, \dots). \quad (12)$$

To transform $P(i)$ into $P'(i)$ requires using rows $P(i-2)$, $P(i-1)$, $P(i+1)$ and $P(i+2)$. The correct linear combination of these rows can be obtained as follows: Listing five adjacent rows of P , we denote with a boxed entry in row $P(i)$ those positions where we require a zero in the new row $P'(i)$.

²As in the tridiagonal case, this can be seen by permuting the rows and columns of P' appropriately.

- [3] R. N. Kapur and J. C. Browne. "Techniques for Solving Block Tridiagonal Systems on Reconfigurable Array Computers". *SIAM J. Sci. Stat. Comput.* 5, 3 (September 1984) 701-719.
- [4] H. S. Stone. "Parallel Tridiagonal Equation Solvers". *ACM Trans. Math. Soft.* 1, 4 (December 1975) 289-307.
- [5] H. A. van der Horst. Large Tridiagonal and Block Tridiagonal Linear Systems on Vector and Parallel Computers". *Parallel Comput.* 5 (1987) 45-54.
- [6] N. K. Madsen and G. H. Rodrigue. "Odd-Even Reduction for Pentadiagonal Matrices". in *Parallel Computers - Parallel Mathematics*. M. Feilmeier (ed.). International Association for Mathematics and Computers in Simulation (1977) 103-106.
- [7] S. Cie, M. Galli, M. Soccio, P. Zellini. "On Some Parallel Algorithms for Inverting Tridiagonal and Pentadiagonal Matrices". *Calcolo* 18, 4 (October-December 1981) 303-319.
- [8] T. H. Pulliam. "Implicit Solution Methods in Computational Fluid Dynamics". *Applied Num. Math.* 2, 6 (December 1986), 441-474.
- [9] "The Connection Machine System Model CM-2 Technical Summary". Technical Report HA87-4. Thinking Machines Corp., Cambridge, MA. (April 1987).
- [10] S. L. Johnsson. "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures". *J. Parallel and Distributed Comput.* 4, (1987) 133-172.
- [11] G. L. Steele Jr. *Common Lisp the Language*. Digital Press. Burlington, MA. (1984).