A Completely Parallelizable Algorithm for the Determinant of a Tridiagonal Matrix

A. Mahmood, D. J. Lynch and L. D. Philipp

School of Electrical Engineering and Computer Science, Washington State University at Tri-Cities, Richland, U.S.A.

A new parallel algorithm (MIMD-PRAM class) having parallel time complexity of $\log_2 n$ for computing the determinant of a tridiagonal matrix is developed. The algorithm is based on coupling the determinants of two neighboring submatrix blocks. With each coupling, the block size is increased by a factor of two until the entire determinant of an $n \times n$ matrix is found by the final coupling of two $\frac{n}{2}$ sized blocks. It is shown that the determinant of an $n \times n$ tridiagonal matrix can be computed in $(3\log_2 n - 2)$ parallel steps with a maximum parallel requirement of $7(\frac{n}{4}) + 3$ processors. The algorithm achieves linear speedup as the number of processors is increased.

Keywords: Determinant, Tridiagonal Matrix, Parallel Algorithm, MIMD, PRAM

1. Introduction

Tridiagonal matrices appear in many practical problems such as interpolation by cubic splines, the solution of two-point boundary value problems by finite differences, and the solution of certain partial differential equations [1]. Another application where tridiagonal matrices are encountered is in computing the eigenvalues of a banded matrix. The eigenvalues are often determined by first reducing a banded matrix to tridiagonal form, and then computing the ei genvalues [2,3]. Recently, the use of tridiagonal matrices was demonstrated in the design of special digital-filter structures [5] which also involves computation of the determinant.

The most efficient way of computing the determinant of a matrix on a single processor computer is by using LU factorization, and then multiplying the diagonal elements of the upper triangular matrix [4,6], i.e.,

$$A = LU$$

$$\det(A) = \prod_{i=1}^{n} U_{ii}$$

The value of a determinant can vary a great deal. For instance, multiplying A by a scalar α yields $\alpha^n \det(A)$. Therefore, the above method requires greater accuracy in the computation of the U_{ii}. In order to compute the determinant of an $n \times n$ tridiagonal matrix, this method requires $4(n-1)$ floating point operations (flops), where $3(n-1)$ flops are for the LU decomposition and $(n-1)$ flops are for multiplication of the n diagonals.

On a parallel machine, the drawback in using LU decomposition for a tridiagonal matrix is that the decomposition is inherently sequential and so it still requires $3(n-1)$ steps. That is, the determinant of an $n \times n$ tridiagonal matrix will be computed in $3(n-1)+1$ steps on a parallel machine, since the product of U_{ii} terms can be carried out concurrently with the computation of LU decomposition.

The amount of parallelism in the computation of the determinant of a tridiagonal matrix can be improved if an expansion is carried out in terms of the minors of the determinant. Unfortunately, this results in a very large number of terms, and hence requires enormous parallel computing resources in order to optimize the number of steps. In the expansion by minors, the exact number of terms that appear in the determinant of a tridiagonal $n \times n$ matrix follows

a Fibonacci sequence and is given by

$$T_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right]$$

where each term has n factors. Hence, the determinant can be computed in $\log_2[nT_n]$ steps if $\frac{n}{2}T_n$ processors are available. For example, if $n = 100$, then the determinant can be computed in 77 steps requiring $\frac{n}{2}T_n \approx 6.4 \times 10^{22}$ processors. Obviously, this approach is impractical. A practical and highly efficient approach is developed in the next section.

2. Decomposing the Determinant of a Tridiagonal Matrix

A parallel algorithm for evaluation of the determinant of a matrix can be developed by using a divide and conquer method based on partitioning. Consider the block partitioning of a matrix into four submatrices:

$$A = \left[\begin{array}{c|c} A_{11} & A12 \\ \hline A_{21} & A_{22} \end{array}\right] \tag{1}$$

The determinant of A is given by [7]

$$\det(A) = \det(A_{11}) \times \det(B) \tag{2}$$

where

$$B = A_{22} - A_{21} \times A_{11}^{-1} \times A_{12}. \tag{3}$$

The special case where either of the antidiagonal submatrices A_{12} or A_{21} is zero is next used to define decoupled submatrices:

Two submatrices P and Q of size $p \times p$ and $q \times q$, respectively, belonging to a matrix A of size $n \times n$, where $n = p + q$, are *decoupled* if A can be partitioned such that

$$A_{11} = P,\ A_{22} = Q$$

and either

$$A_{12} = O \text{ or } A_{21} = O$$

Theorem 1. *The determinant of a matrix comprised of two decoupled matrices is given by the product of the determinant of the decoupled submatrices.*

PROOF. Since $A_{12} = O$ or $A_{21} = O$, $B = A_{22} = Q$ in equation (3). Thus, from equation (2)

$$\det(A) = \det(P) \times \det(Q) \tag{4}$$

□

A more interesting situation arises when considering the coupling of submatrices P and Q by a small overlapping submatrix C. An example of coupled submatrices is shown in *Fig. 1.*

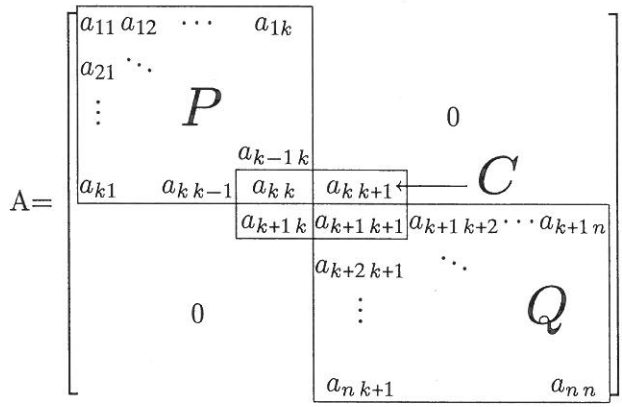


Fig. 1. A matrix with coupled submatrices

A tridiagonal matrix can be considered a coupled system where the submatrices P and Q are themselves tridiagonal. The matrix shown in *Fig. 2.* is an example of a tridiagonal matrix decomposable into smaller coupled submatrices.

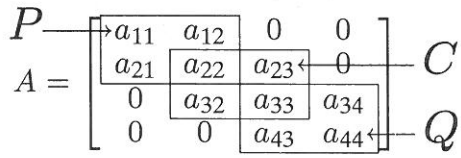


Fig. 2. A 4 × 4 tridiagonal matrix

The determinant of such a coupled system can be expressed in terms of the determinants of the decoupled blocks and a coupling product term. In this example

$$\begin{aligned} \det(A) &= a_{11}(a_{22}(a_{33}a_{44} - a_{34}a_{43}) \\ &\quad - a_{23}(a_{32}a_{44} - 0)) \\ &- a_{12}(a_{21}(a_{33}a_{44} - a_{34}a_{43}) - a_{23}(0)) \end{aligned} \tag{5}$$

$$\begin{aligned} &= (a_{11}a_{22} - a_{12}a_{21})(a_{33}a_{44} - a_{34}a_{43}) \\ &\quad - (a_{11}a_{44}a_{23}a_{32}) \\ &= \det(P)\det(Q) \\ &- \text{Cofactor } P_{22} \text{ Cofactor } Q_{11}a_{23}a_{32} \end{aligned} \tag{6}$$

$$\begin{aligned} &= \det(P)\det(Q) \\ &\quad - \text{CouplingProduct}(P, Q) \end{aligned} \tag{7}$$

Thus the determinant of matrix A is given by the product of the determinants of the decoupled blocks P and Q less a coupling product term.

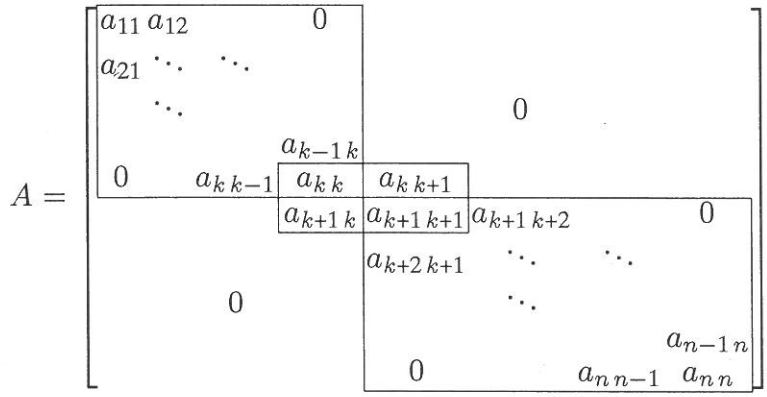


Fig. 3. An $n \times n$ tridiagonal matrix with coupled submatrices

The coupling product involves the anti-diagonal terms in the coupling block C and cofactors of the coupled elements of P and Q. Theorem 2 states the general case for tridiagonal matrices (*Fig. 3*).

Theorem 2. *Let A be an $n \times n$ tridiagonal matrix, consisting of two decoupled blocks, P, a $k \times k$ tridiagonal matrix, and Q, an $(n-k) \times (n-k)$ tridiagonal matrix, and a coupling block C; $n \geq 4; 2 \leq k \leq n-1$. The determinant of A is then*

$$\begin{aligned}\det(A) &= (\det(P))(\det(Q)) \\ &\quad - (\text{Cofactor } P_{kk})(\text{Cofactor } Q_{11}) \times \\ &\qquad \times (a_{k\,k+1})(a_{k+1\,k}) \end{aligned} \quad (8)$$

where Cofactor $A_{ij} = (-1)^{i+j} \det(M_{ij}(A))$. *$M_{ij}(A)$ is the minor matrix of a_{ij}, that is, the matrix formed by elimination of the i^{th} row and j^{th} column of A.*

PROOF.. Using a cofactor expansion for column $k+1$ in A (*Fig. 3.*), the determinant of A is

$$\begin{aligned}\det(A) &= a_{k\,k+1}(\text{Cofactor } A_{k\,k+1}) \\ &\quad + a_{k+1\,k+1}(\text{Cofactor } A_{k+1\,k+1}) \\ &\quad + a_{k+2\,k+1}(\text{Cofactor } A_{k+2\,k+1}) \end{aligned} \quad (9)$$

Each of the cofactors is determined separately and then substituted into (9).

Cofactor $A_{k\,k+1} =$

$$= (-1) \times \left| \begin{array}{ccc|c|ccc} & & & 0 & & & \\ & & & \vdots & & & \\ & M_{k\,k}(P) & & 0 & & 0 & \\ & & & a_{k-1\,k} & & & \\ \hline 0 & \cdots & 0 & a_{k+1\,k} & a_{k+1\,k+2} & 0 & \cdots 0 \\ \hline & & & 0 & & & \\ & 0 & & \vdots & & M_{11}(Q) & \\ & & & 0 & & & \end{array} \right| \quad (10)$$

where $|\ | = \det(\ )$

Applying Theorem 1 to relation (10) with $A_{11} = M_{kk}(P)$ yields

$$\begin{aligned}&\text{Cofactor } A_{k\,k+1} = (-1)(a_{k+1\,k}) \times \\ &\quad \times (\text{Cofactor } P_{k\,k})(\text{Cofactor } Q_{11}) \end{aligned} \quad (11)$$

Similar application of Theorem 1 yields

$$\begin{aligned}&\text{Cofactor } A_{k+1\,k+1} \\ &\qquad = (\det(P))(\text{Cofactor } Q_{11}) \end{aligned} \quad (12)$$

and

$$\begin{aligned}&\text{Cofactor } A_{k+2\,k+1} \\ &\qquad = (\det(P))(\text{Cofactor } Q_{12}) \end{aligned} \quad (13)$$

Using relations (11), (12) and (13) in (9) yields

$$\begin{aligned}\det(A) &= [(a_{k+1\,k+1})(\text{Cofactor } Q_{11}) \\ &\quad + (a_{k+2\,k+1})(\text{Cofactor } Q_{12})] \det(P) \\ &\quad - (a_{k+1k})(a_{kk+1})(\text{Cofactor } P_{kk})(\text{Cofactor } Q_{11}) \\ &= \det(Q) \det(P) - \text{CouplingProduct}(P, Q) \end{aligned}$$

□

3. Parallel Algorithm for the Determinant of a Tridiagonal Matrix

Theorem 2 relates the determinant of a tridiagonal matrix to determinants of submatrices within it. This provides a natural mechanism for a divide and conquer approach to the calculation of a determinant by parallel evaluation of determinants of submatrices. Equivalently, the evaluation of a determinant can be viewed as a bottom-up process, constructing the determinant of a matrix from determinants of smaller submatrices that are computed in parallel.

The algorithm presented in this section describes how to boot-strap from the $\frac{n}{2}$ determinants of 2×2 matrices to the resultant determinant of an $n \times n$ matrix in $\log_2 n$ steps. For sake of simplicity in describing the algorithm, it is assumed that n is a power of 2. A general program is provided in the appendix which can compute the determinant of a tridiagonal matrix for arbitrary size n.

In the algorithm that follows, $\det[i][j]$ denotes the determinant of the $j \times j$ submatrix whose upper, leftmost element is a_{ii} (i.e., the first bracketed index indicates the starting element and the second indicates the size of the submatrix). For example, the determinant of block Q in matrix A of *Fig. 2.* will be denoted as $\det[3][2]$. Similarly, $\mathrm{CP}[i][j]$ denotes the coupling product required in the computation of $\det[i][j]$. The algorithm is described in terms of C language constructs and thus the row and column numbers start at 0. The tridiagonal matrix is assumed to be stored in array $A[n][3]$ with the lower diagonal in column 0, the center diagonal in column 1, and upper diagonal in column 2 of A.

For a given block (submatrix) of size B, $\det[i][B]$ is computed by the application of Theorem 2 as,

$$\det[i][B] = \det[i]\left[\frac{B}{2}\right] \times \det\left[i+\frac{B}{2}\right]\left[\frac{B}{2}\right] - \mathrm{CP}[i][B]$$

where B is a power of 2 and i is a positive multiple of B. The coupling product $\mathrm{CP}[i][B]$ in above expression is computed as,

$$\mathrm{CP}[i][B] = A\left[i+\frac{B}{2}-1\right][0] \times A\left[i+\frac{B}{2}-1\right][2] \times \det[i]\left[\frac{B}{2}-1\right] \times \det\left[i+\frac{B}{2}+1\right]\left[\frac{B}{2}-1\right]$$

Now if $\det[i][B]$ is to be coupled to a succeeding submatrix of size B to form a larger submatrix of size $2 * B$, then the $\det[i][B-1]$ is required in the coupling product term for computing $\det[i][2*B]$. Note that in the computation of $\det[i][B-1]$, $\mathrm{CP}[i][B-1]$ requires $det[i][B/2-1]$ and $\det[i+B/2+1][B/2-2]$. Similarly, for coupling of $\det[i][B]$ to a preceding submatrix of size B, $\det[i+1][B-1]$ is required. The coupling product computation for $\det[i+1][B-1]$ will in turn need $det[i+1][B/2-2]$. In short, for a given block size B, four different determinants are computed, i.e.:

$$\begin{array}{l} \det[i+1][B-2] \\ \det[i][B-1] \\ \det[i+1][B-1] \\ \det[i][B] \end{array}$$

where $\det[i+1][B-1]$ and $\det[i+1][B-2]$ will be used to couple to a preceding block; $\det[i][B-1]$ and $\det[i+1][B-2]$ will be used in coupling to a succeeding block, and $\det[i][B]$ will be used in $\det[i][2*B]$.

```
Algorithm:

 for (i = 0; i < n; i = i + 1)                    /* initialization */
  { det[i][0] = 1.0; det[i][1] = A[i][1];  }

/*------parallel STEP 1-------n/2 2x2 determinants-----------*/
 for (i = 0; i < n; i = i + 2)      /* forall */
   det[i][2] = A[i][1] * A[i+1][1] - A[i][0] * A[i][2] ;
/*  endsync */
 B = 4 ;  hB = 2 ;  /* B = current block size, hB= half block */
 while(B < n)
```

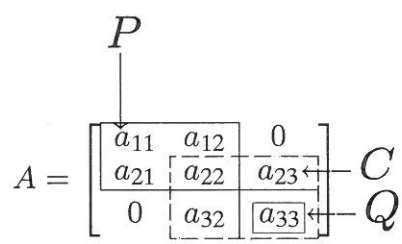
```
{
 /*---parallel step 2, 4,...-coupling products for next step*/
 for (i = 0; i < n; i = i + B)     /* forall */
  {  /*  parbegin */
   CP[i+1][B-2] = A[i+hB-1][0] * A[i+hB-1][2] *
                          det[i+1][hB-2] * det[i+hB+1][hB-2];
   CP[i][B-1] = A[i+hB-1][0] * A[i+hB-1][2] *
                            det[i][hB-1] * det[i+hB+1][hB-2];
   CP[i+1][B-1] = A[i+hB-1][0] * A[i+hB-1][2] *
                          det[i+1][hB-2] * det[i+hb+1][hB-1];
   CP[i][B] = A[i+hB-1][2] * A[i+hB-1][2] *
                            det[i][hB-1] * det[i+hb+1][hB-1];
   /*  parend */
  }
 /* endsync */
 /*-------------------------------------------------------*/

 /*---parallel STEP 3, 5, ....determinants-----------------*/
 for (i = 0; i < n; i = i + B)     /* forall */
  {
   /* parbegin */
   det[i+1][B-2]= det[i+1][hB-1] *det[i+hB][hB-1] - CP[i+1][B-2];
   det[i][B-1] = det[i][hB] * det[i+hB][hB-1] - CP[i][B-1] ;
   det[i+1][B-1] = det[i+1][hB-1] * det[i+hB][hB] - CP[i+1][B-1];
   det[i][B] = det[i][hB] * det[i+hB][hB] - CP[i][B] ;
   /* parend */
  }
 /* endsync */
 /*-------------------------------------------------------*/
 hB = B ;  B = B * 2 ;
} /* end while */

CP[0][n]= A[n/2-1][0] * A[n/2-1][2]*
                      det[0][n/2-1] * det[n/2+1][n/2-1];
det[0][n]= det[0][n/2]*det[n/2][n/2] - CP[0][n]; /* STEP log2n */
```

The above algorithm can be run on an MIMD-shared memory (PRAM) machine. The constructs needed for a parallel processor implementation are indicated in boldface within the comment fields. These parallel constructs, i.e., **forall**, **parbegin**, **parend** and **endsync**, are general [8] and can be changed to the exact syntax required by a given parallel language. The forall construct indicates parallel execution of the different iterations of the for loop; **parbegin** and **parend** indicate a parallel block of statements; **endsync** provides the barrier synchronization [9] to synchronize the sequencing of parallel steps.

Although the algorithm above is presented for matrices which are a positive power of 2 in size, Theorem 2 is general, and thus the algorithm can be easily modified to compute the determinant of any size matrix. As an example, the determinant of a 3×3 tridiagonal matrix is computed as,



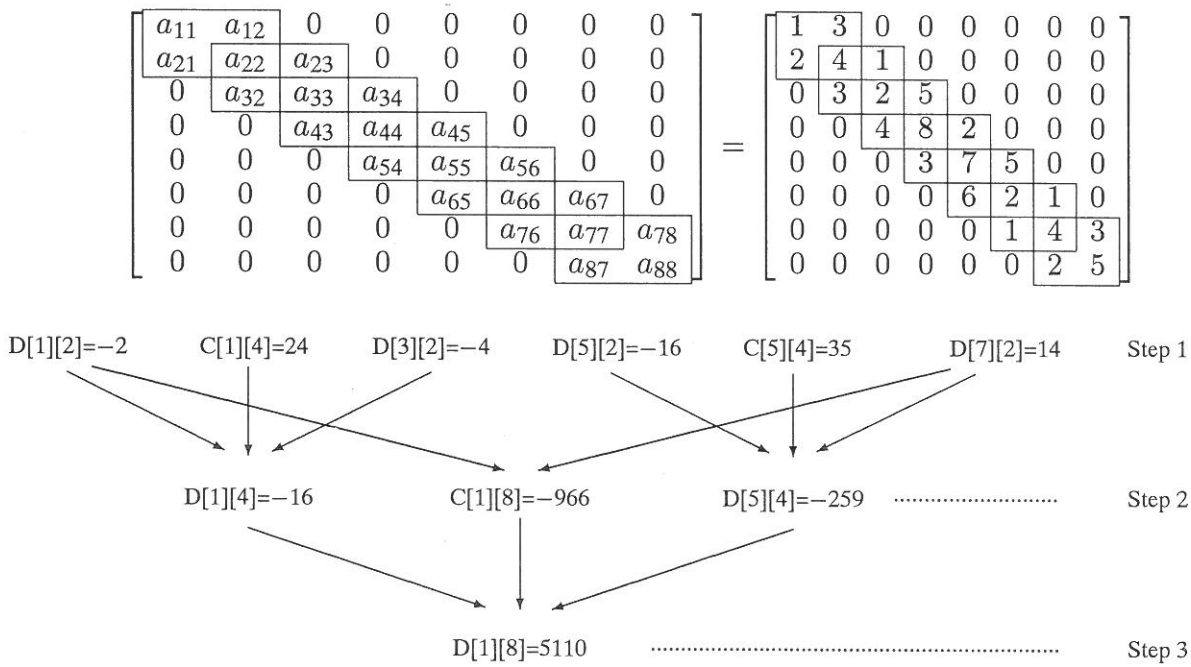


Fig. 4. An example showing the application of the algorithm from Section 3 to an 8×8 tridiagonal matrix

$$\begin{aligned}\det(A) &= \det(P) \times \det(Q) \\ &\quad - \text{CouplingProduct}(P, Q) \\ &= (a_{11}a_{22} - a_{12}a_{21}) \times (a_{33}) \\ &\qquad - a_{23} \times a_{32} \times a_{11} \times 1\end{aligned}$$

Note that in order to determine the coupling product, the cofactor P_{22} of P is a_{11} and cofactor Q_{11} is 1.

The appendix lists a modified version of the above algorithm such that parallelism is improved in terms of floating point steps. Analysis of this program indicates that $3\log_2(n) - 2$ floating point steps are required to compute the determinant with a maximum parallel requirement of $7(\frac{n}{4}) + 3$ processors in parallel step 3.

4. Results

An example is presented in *Fig. 4.* to show the application of the algorithm of Section 3 to computing the determinant of an 8×8 tridiagonal matrix. In the figure, D refers to det and C refers to CouplingProduct. *Fig. 5.* shows a graph relating the execution time, in terms of floating point steps for computing the determinant of a $262,144 \times 262,144$ tridiagonal matrix, to the available number of processors. When using one processor, the execution time is 2,359,270 floating point steps. As the number of processors is increased (up to $\sim 10^4$), the computation time decreases by a factor equal to the number of processors used. Theoretically, the minimum time is achieved when $7(\frac{n}{4}) + 3 = 458755$ processors are employed yielding a total execution time of $3\log_2 n - 2 = 52$ steps. If the number of processors is increased beyond $7(\frac{n}{4}) + 3$, there is no improvement in execution time.

Since the dataflow graph for the computation of the determinant by the algorithm of Section 3 follows the pattern of a binary tree, the speedup in execution time is linear (within 5%) as the number of processors, p, is increased up to $p\log_2 p \leq \frac{n}{5}$. As indicated by Figure 5, the algorithm achieves ideal performance in terms of speedup for a wide range of number of processors (i.e., when the number of processors is $< 10^4$ in this example). Such properties of a parallel algorithm are highly desirable and an algorithm having the linear speedup property is referred to as maximally parallelizable [9].

It should be noted that the number of floating point operations for sequential processing of this algorithm is $O(n)$ and is roughly twice the number required by the LU decomposition approach for computing the determinant. The number of sequential floating point operations

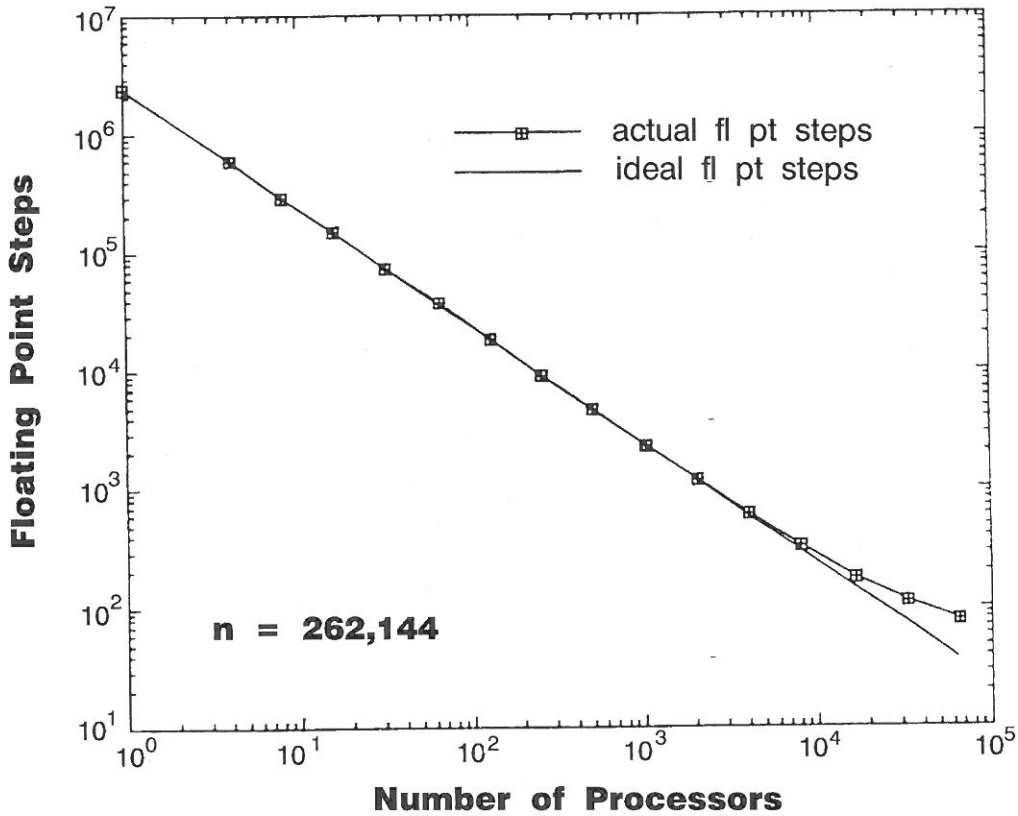


Fig. 5. Plot of execution time vs. number of processors for a tridiagonal matrix

is $9n - 26$ for the program in the appendix, as compared to $4(n-1)$ for the LU decomposition approach.

5. Conclusions

An algorithm for efficient parallel computation of a tridiagonal matrix has been developed in Section 3. This algorithm is based on a new formulation of the determinant in terms of the determinants of two coupled submatrices. In terms of floating point operations, the determinant of an $n \times n$ tridiagonal matrix can be computed in $3 \log_2 n - 2$ steps with $7(\frac{n}{4}) + 3$ number of processors. The algorithm yields a linear speedup as the number of processors, p, is increased (for $p < n$).

Since this algorithm does not involve any division operations, it can be applied to symbolic evaluation of determinants involving polynomials as matrix elements. Also, the roundoff errors are slightly improved due to the absence of division.

The idea of coupling submatrices is new and may have potential in parallelizing other matrix problems. In particular, the algorithm of Section 3 can be extended to computing the determinant of general banded matrices provided the coupling theory is established for the case when the coupling block is larger than 2×2. Work is continuing on the determinant of a general banded matrix and other matrix problems using this approach.

Acknowledgements

This work was supported in part from a grant from Electric Power Research Institute (EPRI) under contract number RP 1570–26. Constructive comments from an anonymous reviewer were helpful in refining discussions in Sections 1 and 2.

Appendix

```
/*-----------------compute_determinant()-------------------------*/
/* A parallel algorithm for the determinant of a Tridiagonal matrix.
   The matrix is stored as:  lower diagonal in 0th, diagonal in 1st,
   upper diagonal in 3rd column */

float compute_determinant(n,A)
int n; float A[][3];
{
 float pD2[max_size] , pD4[max_size], pD[max_size][4], pC[max_size] ;
 /* these arrays store partial determinants and coupling products */

 float D[max_size][4], C[max_size][4] ;
 /* D and C are for storing the intermediate determinants and the
    coupling products respectively. The second dimension is for
    storing:  [i][0] =  det[i+1][B-2] or CP[i+1][B-2]
              [i][1] =  det[i][B-1]   or CP[i][B-1]
              [i][2] =  det[i+1][B-1] or CP[i+1][B-1]
              [i][3] =  det[i][B]     or CP[i][B]                */

 int i, B, hB ;  /* B  = current block size, hB = half block size */

/*------parallel floating point STEP 1----------------------*/
/* parbegin */
 for (i = 0; i < n; i = i + 1)     /* forall */
    pC[i] = A[i][0] * A[i][2] ;    /*  antidiagonal products  */

 for (i = 0; i < n; i = i + 2)     /* forall */
    pD2[i] = A[i][1] * A[i+1][1]; /* partial 2x2 determinant */
/* parend */
/* endsync */

/*------parallel STEP 2---------------------------------------*/
/* parbegin */
 for (i = 0; i < (n-1); i = i + 2) /* forall */
   D[i][3] = pD2[i] - pC[i] ;      /* ----2x2 determinants    */
   for (i = 0; i < (n-3) ; i = i + 4) /* forall */
   pD4[i] = A[i][1] * A[i+3][1];  /* partial 4x4 determinant  */
/* parend */
/* endsync */

/*------parallel STEP 3---------------------------------------*/
/* coupling products and partial determinants for next step   */
/* parbegin */
 for (i = 0; i < (n-3); i = i + 4) /* forall */
  { /* parbegin */
   C[i][0] = pC[i+1] ;            pD[i][0]= A[i+1][1] * A[i+2][1];
   C[i][1] = pC[i+1] * A[i][1];   pD[i][1]= D[i][3] * A[i+2][1] ;
   C[i][2] = pC[i+1] * A[i+3][1]; pD[i][2]= A[i+1][1] * D[i+2][3];
   C[i][3] = pC[i+1] * pD4[i] ;   pD[i][3]= D[i][3] * D[i+2][3];
    /* parend */
  }
```

```
 switch(n-i) {
  case 1: /* parbegin */
          C[i][2]= 0; C[i][3]= 0;
     pD[i][2]= 1; pD[i][3]= A[i][1];
          /* parend */ break ;
  case 2: /* parbegin */
          C[i][2]= 0 ; C[i][3]= pC[i+1] ;
     pD[i][2]= A[i+1][1]; pD[i][3]= A[i][1]* A[i+1][1];
          /* parend */ break ;
  case 3: /* parbegin */
          C[i][2]= pC[i+1]; C[i][3]= pC[i+1]* A[i][1];
     pD[i][2]=A[i+1][1]*A[i+2][1]; pD[i][3]=D[i][3]*A[i+2][1];
          /* parend */  break ;
  }
/* parend */
/* endsync */
/*-------------------------------------------------------------*/

 B = 4 ;  hB = 2 ; /* B = current block size, hB= half block  */
 while(B < n) {
/*---parallel STEP 4, 7, 10, ....(determinants)--------------*/
   for (i = 0; i < n; i = i + B) /* forall */
    { /* parbegin */
     D[i][0] = pD[i][0] - C[i][0] ;  D[i][1] = pD[i][1] - C[i][1] ;
     D[i][2] = pD[i][2] - C[i][2] ;  D[i][3] = pD[i][3] - C[i][3] ;
      /* parend */
   }  /* endsync */
/*-------------------------------------------------------------*/

   hB = B ; B = B * 2 ;
   if (B < n) {
      /*----parallel steps 5,6,  8,9, ....-------------------*/
      /* coupling products and partial determinants for next step */
      for (i = 0; i < (n-hB); i = i + B) /* forall */
       { /* parbegin */
        C[i][0]=pC[i+hB-1]*D[i][0]*D[i+hB][0];
        pD[i][0]=D[i][2]*D[i+hB][1];
        C[i][1]=pC[i+hB-1]*D[i][1]*D[i+hB][0];
        pD[i][1]=D[i][3]*D[i+hB][1];
       C[i][2]=pC[i+hB-1]*D[i][0]*D[i+hB][2];
        pD[i][2]=D[i][2]*D[i+hB][3];
   C[i][3]=pC[i+hB-1]*D[i][1]*D[i+hB][2];
        pD[i][3]=D[i][3]*D[i+hB][3];
         /* parend */
       } /* endsync */
    /*-----------------------------------------------------*/
    }
  }
  /* parbegin */
  C[0][3] = pC[B/2-1]* D[0][1]* D[B/2][2]; pD[0][3]= D[0][3]* D[B/2][3];
  /* parend */
  return(ApD[0][3] - C[0][3]) ; /* STEP 3log2n-2 */
}
```

References

[1] GERALD, C.F. , and P. O. WHEATLEY, *Applied Numerical Analysis*, Addison-Wesley Publishing Company, 1985.

[2] PISSANETZKY, SERGIO, *Sparse Matrix Technology*, Academic Press, Inc., 1984, pp. 207–214.

[3] SMITH, B. T., et. al., "Matrix Eigensystem Routines — EISPACK Guide", *Lecture Notes in Computer Science*, Vol. 6, 2nd Ed. Springer-Verlag, Berlin, 1976, p. 532.

[4] DUFF, I. S., A. M. ERISMAN, and J. K. REID, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986, pp. 81–82.

[5] SARCINELLI, M. D. and P. S. R. DINIZ, *Tridiagonal State-Space Digital-Filter Structures*, IEEE Transactions on Circuits and Systems, Vol. 37 no. 6, June 1990, pp. 818–824.

[6] GOLUB, G. H., and C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, 1983, p. 56.

[7] KAILATH, T., *Linear Systems*, Prentice-Hall, 1980, p. 650.

[8] JORDAN, H. F., *A Special Purpose Architecture for Finite Element Analysis*, Proceedings of the 1978 International Conference on Parallel Processing, pp. 263–266, 1978.

[9] ALMASI, G. S. , and A. GOTTLEIB, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, pp. 115–167.

Received: Semptember, 1993
Accepted: January, 1994

Contact address:
School of Electrical Engineering
and Computer Science,
Washington State University
at Tri-Cities,
Richland, WA 99352, U.S.A.

AUSIF MAHMOOD is currently an assistant professor in the School of Electrical Engineering and Computer Science at Washington State University Tri-Cities. His research interests are in parallel algorithms and architectures, and CAD for VLSI.

DONALD J. LYNCH is an associate professor in the School of Electrical Engineering and Computer Science at Washington State University Tri-Cities. His research interests are in numerical methods, parallel algorithms and software testing.

LEE D. PHILIPP is a professor in the School of Electrical Engineering and Computer Science at Washington State University Tri-Cities. His research interests are in non-destructive evaluation by Eddy currents and numerical methods.