A Parallel Jacobi Method for the Takagi Factorization

Xiaohong Wang
Department of Computing and Software
McMaster University
Hamilton, ON, Canada L8S 4L7

Sanzheng Qiao
Department of Computing and Software
McMaster University
Hamilton, ON, Canada L8S 4L7

Abstract ***In this paper, we discuss a special form of the singular value decomposition (SVD), called Takagi factorization, of a complex-symmetric matrix. We present a multthreading parallel Jacobi algorithm for computing the Takagi factorization.***

Keywords: Complex-symmetric matrix, Singular Value Decomposition (SVD), Takagi factorization, Parallel Jacobi method, Multithreading.

1 Introduction

A matrix A of order n is symmetric if $A = A^{\mathrm{T}}$. When the entries of A are complex, A is complex-symmetric. Note that a complex-symmetric matrix is different from a complex Hermitian matrix. For the entries a_{ij} in a complex-symmetric matrix A, $a_{ij} = a_{ji}$; whereas in a Hermitian matrix A, $a_{ij} = \bar{a}_{ji}$, where $\bar{a}$ denotes the complex conjugate of a.

Complex-symmetric matrices do not occur in applications as frequently as Hermitian matrices, but there are circumstances in which they do occur. For example, they arise in complex coordinate rotation method [3] and optical potential method [5], which are applied to chemical problems.

Also, a special complex-symmetric matrix is square Hankel matrix

$$H = \begin{bmatrix} h_1 & h_2 & \cdots & h_{n-1} & h_n \\ h_2 & h_3 & \cdots & h_n & h_{n+1} \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ h_{n-1} & h_n & \cdots & h_{2n-3} & h_{2n-2} \\ h_n & h_{n+1} & \cdots & h_{2n-2} & h_{2n-1} \end{bmatrix}$$

where all entries on the same anti-diagonal are equal. Hankel matrix occurs commonly in signal processing.

In general, for any m-by-n matrix A, there exist unitary matrices

$$U \in C^{m\times m} \quad \text{and} \quad V \in C^{n\times n}$$

such that

$$U^{\mathrm{H}} A V = \mathrm{diag}(\sigma_1, ..., \sigma_p) \in R^{m\times n} \qquad (1)$$

where $p = \min\{m, n\}$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ [2, page 70]. The decomposition (1) is called the singular value decomposition (SVD) of A. The SVD of a matrix reveals many properties of the matrix and has wide applications.

Exploiting the symmetry, a complex-symmetric matrix has a special SVD where $\bar{V} = U$. The formal definition of this special SVD, called Takagi factorization, is given in the following section. This paper presents a parallel Jacobi method for computing the Takagi factorization. After introducing the Takagi factorization in Section 2, we discuss this factorization in the two dimensional case, which is the building block of the Jacobi method for SVD. In Section 4, we show a sequential Jacobi method for the Takagi factorization. This algorithm is efficient in both computation and storage by exploiting the symmetry. The multithreading Jacobi method is presented in Section 5. In particular, we discuss the load balancing and synchronization issues. Finally, we conclude our paper in Section 6.

2 Takagi Factorization

In the special case when A is complex-symmetric, the following theorem [4, page 204] shows that A has a special form of the SVD, namely Takagi factorization.

Theorem 1 (Takagi factorization) *If A is an order n complex-symmetric matrix, then there exists a unitary $Q \in C^{n\times n}$ and an order n nonnegative diagonal $\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_n)$ where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$, such that*

$$Q^{\mathrm{H}} A Q^* = \Sigma \quad or \quad A = Q\Sigma Q^{\mathrm{T}}. \tag{2}$$

Note that if Q is unitary, then $\bar{Q}$ and Q^{T} are also unitary, since $\bar{Q}^{\mathrm{H}}\bar{Q} = Q^{\mathrm{T}}\bar{Q} = (Q^{\mathrm{H}}Q)^{\mathrm{T}} = I$.

The SVD of a complex-symmetric matrix occurs in mathematical applications such as uniform polynomial or rational approximation of a high degree polynomial on a disk (see [1] for references).

From the Takagi factorization (2), we can see that we need to compute only one set of singular vectors the columns of Q, instead of two sets of singular vectors the columns of U and V in the SVD (1) of a general matrix. Moreover, since A is symmetric, we can assume that A is stored in its upper or lower triangular part. Thus both computation and storage can be saved by computing the Takagi factorization instead of the SVD of a complex-symmetric matrix.

One might think that the Takagi factorization can be obtained by simply scaling an SVD. For example,

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}.$$

Its SVD is

$$U\Sigma V^{\mathrm{T}} = \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}^{\mathrm{T}}.$$

Its Takagi factorization is

$$Q\Sigma Q^{\mathrm{T}} = \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}i \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}i \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}i \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}i \end{bmatrix}^{\mathrm{T}},$$

which can be obtained by scaling the second columns of U and V in the SVD by factors of i and $-i$ respectively. However, consider

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Its SVD is

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^{\mathrm{T}}.$$

Whereas, its Takagi factorization is

$$\begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}i \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}i \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}i \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}i \end{bmatrix}^{\mathrm{T}}.$$

In this example, it is not trivial to obtain the Takagi factorization from the SVD.

3 Two Dimensional Case

In this section, we present a Takagi factorization algorithm for a 2-by-2 complex-symmetric matrix [6]. Let

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}.$$

We want to find its Takagi decomposition (2), which is equivalent to

$$A\bar{Q} = Q\Sigma.$$

We first find a unit norm vector $\mathbf{q}$ such that

$$A\bar{\mathbf{q}} = \sigma\mathbf{q}$$

where σ is a nonnegative scalar. Similar to the relation between the eigenvalues of AA^{H} and the singular values of A, let $(\lambda, \mathbf{v})$ be an eigenpair of $A\bar{A}$:

$$A\bar{A}\mathbf{v} = \lambda\mathbf{v}.$$

Since A is symmetric, $A\bar{A} = AA^{\mathrm{H}}$. Thus the eigenvalue $\lambda \geq 0$. Now, we use $\mathbf{v}$ as a candidate for $\mathbf{q}$ in two cases depending on whether $A\bar{\mathbf{v}}$ and $\mathbf{v}$ are linearly dependent. In the first case, if $A\bar{\mathbf{v}} = \alpha\mathbf{v}$ for some scalar α, then

$$\lambda\mathbf{v} = A\bar{A}\mathbf{v} = A(\bar{\alpha}\bar{\mathbf{v}}) = |\alpha|^2\mathbf{v},$$

which implies that one of the singular values $\sigma = \sqrt{\lambda} = |\alpha|$. Thus we have found

$$\mathbf{q} = \mathrm{sign}(\sqrt{\alpha})\,\mathbf{v},$$

where the sign function is defined by

$$\mathrm{sign}(c) = \begin{cases} c/|c| & c \neq 0 \\ 1 & c = 0. \end{cases}$$

We can verify that

$$A\bar{\mathbf{q}} = \mathrm{sign}(\sqrt{\bar{\alpha}})\,A\bar{\mathbf{v}} = \mathrm{sign}(\sqrt{\bar{\alpha}})\,\alpha\mathbf{v} = |\alpha|\,\mathbf{q}.$$

In the second case, if there is no scalar α so that $A\bar{\mathbf{v}} \neq \alpha\mathbf{v}$, we consider

$$\mathbf{u} = A\bar{\mathbf{v}} + \sqrt{\lambda}\,\mathbf{v}.$$

This vector $\mathbf{u}$ is nonzero because in this case we assume $A\bar{\mathbf{v}} \neq \alpha\mathbf{v}$ for any α, which means that $A\bar{\mathbf{v}}$ and $\mathbf{v}$ are linearly independent. Now that

$$A\bar{\mathbf{u}} = A\bar{A}\mathbf{v} + \sqrt{\lambda}A\bar{\mathbf{v}} = \lambda\mathbf{v} + \sqrt{\lambda}A\bar{\mathbf{v}} = \sqrt{\lambda}\mathbf{u},$$

we can choose

$$\mathbf{q} = \mathbf{u}/\|\mathbf{u}\|.$$

Then we construct a unitary Q such that its first column is the unit norm vector $\mathbf{q}$. For example, suppose $\mathbf{q} = [q_1\ q_2]^{\mathrm{T}}$, we can choose

$$Q = \begin{bmatrix} q_1 & -\bar{q}_2 \\ q_2 & \bar{q}_1 \end{bmatrix}.$$

Thus

$$Q^{\mathrm{H}}A\bar{Q} = \begin{bmatrix} \sigma & \times \\ 0 & \times \end{bmatrix}$$

is upper triangular. Obviously, $Q^{\mathrm{H}}A\bar{Q}$ is symmetric, so $Q^{\mathrm{H}}A\bar{Q}$ is diagonal. Now, we have the following algorithm for two dimensional Takagi decomposition.

Algorithm 1 (2D Takagi Decomposition) *Given a 2-by-2 complex symmetric A, this algorithm computes a unitary Q and nonnegative diagonal $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2)$ such that $A = Q\Sigma Q^{\mathrm{T}}$.*

Find an eigenpair $(\lambda, \mathbf{v})$ of $A\bar{A}$:
    $A\bar{A}\mathbf{v} = \lambda\mathbf{v}$;
$\sigma_1 = \sqrt{\lambda}$;
If $A\bar{\mathbf{v}} = \alpha\mathbf{v}$ for some α
    $\mathbf{u} = \mathrm{sign}(\sqrt{\alpha})\,\mathbf{v}$;
else
    $\mathbf{u} = A\bar{\mathbf{v}} + \sqrt{\lambda}\mathbf{v}$;
end
Determine a unitary Q so that its first column is the normalized $\mathbf{u}$;
$\sigma_2 = Q^{\mathrm{H}}_{:,2}A\bar{Q}_{:,2}$;
Make σ_2 nonnegative by scaling the second column $Q_{:,2}$ of Q.

4 Jacobi Method

The two dimensional Takagi factorization can be used in Jacobi method for the singular value decomposition. For each index pair (i, j) $(i < j)$, we apply Algorithm 1 to the block

$$\begin{bmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{bmatrix}$$

to obtain the Takagi factorization

$$\begin{bmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{bmatrix} = V(i,j) \begin{bmatrix} \sigma_i & 0 \\ 0 & \sigma_j \end{bmatrix} V^{\mathrm{T}}(i,j). \tag{3}$$

Then we apply the unitary transform $V(i, j)$ to update the ith and jth rows of A

$$\begin{bmatrix} \mathbf{a}_{i,1:n} \\ \mathbf{a}_{j,1:n} \end{bmatrix} \leftarrow V^{\mathrm{H}}(i,j) \begin{bmatrix} \mathbf{a}_{i,1:n} \\ \mathbf{a}_{j,1:n} \end{bmatrix} \tag{4}$$

followed by updating the ith and jth columns

$$[\mathbf{a}_{1:n,i}\ \mathbf{a}_{1:n,j}] \leftarrow [\mathbf{a}_{1:n,i}\ \mathbf{a}_{1:n,j}]\,\bar{V}(i,j). \tag{5}$$

For efficiency, a symmetric A is usually stored in its upper or lower triangular part. Assuming A is stored in its upper part, we can combine (4) and (5) to update the upper triangular part. The subvectors $\mathbf{a}_{1:i-1,i}$ and $\mathbf{a}_{1:i-1,j}$ are updated by postmultiplying $\bar{V}(i,j)$ as in (5):

$$[\mathbf{a}_{1:i-1,i}\ \mathbf{a}_{1:i-1,j}] \leftarrow [\mathbf{a}_{1:i-1,i}\ \mathbf{a}_{1:i-1,j}]\bar{V}(i,j).$$

These subvectors are not touched by the row update (4). Similarly, the subvectors $\mathbf{a}_{i,j+1:n}$ and $\mathbf{a}_{j,j+1:n}$ are updated by premultiplying $V^{\mathrm{H}}(i,j)$ as in (4). The row update (4) computes the new subvector $\mathbf{a}_{i,i+1:j-1}$ in the upper triangular part using $\mathbf{a}_{i,i+1:j-1}$ and $\mathbf{a}_{j,i+1:j-1}$. The column update (5) computes the new subvector $\mathbf{a}_{i+1:j-1,j}$ in the upper triangular part using $\mathbf{a}_{i+1:j-1,i}$ and $\mathbf{a}_{i+1:j-1,j}$. Since A is symmetric, $\mathbf{a}^{\mathrm{T}}_{i,i+1:j-1} = \mathbf{a}_{i+1:j-1,i}$ and $\mathbf{a}_{i+1:j-1,j} = \mathbf{a}^{\mathrm{T}}_{j,i+1:j-1}$. Thus we can simultaneously update both subvectors $\mathbf{a}_{i,i+1:j-1}$ and $\mathbf{a}_{i+1:j-1,j}$:

$$[\mathbf{a}^{\mathrm{T}}_{i,i+1:j-1}\mathbf{a}_{i+1:j-1,j}] \leftarrow [\mathbf{a}^{\mathrm{T}}_{i,i+1:j-1}\ \mathbf{a}_{i+1:j-1,j}]\bar{V}(i,j).$$

In summary, we have the following updating algorithm.

Algorithm 2 (Update) *Suppose a symmetric A is stored in its upper triangular part, this algorithm updates the upper triangular part by applying (5) and (4).*

$a_{ii} = \sigma_i$; $a_{jj} = \sigma_j$; $a_{ij} = 0$;
if $i > 1$
  $[\mathbf{a}_{1:i-1,i}\ \mathbf{a}_{1:i-1,j}] = [\mathbf{a}_{1:i-1,i}\ \mathbf{a}_{1:i-1,j}]\bar{V}(i,j)$;
end
if $j > i+1$
  $[\mathbf{a}^{\mathrm{T}}_{i,i+1:j-1}\ \mathbf{a}_{i+1:j-1,j}] = [\mathbf{a}^{\mathrm{T}}_{i,i+1:j-1}\ \mathbf{a}_{i+1:j-1,j}]\bar{V}(i,j)$;
end
if $j < n$
  $\begin{bmatrix} \mathbf{a}_{i,j+1:n} \\ \mathbf{a}_{j,j+1:n} \end{bmatrix} = V^{\mathrm{H}}(i,j) \begin{bmatrix} \mathbf{a}_{i,j+1:n} \\ \mathbf{a}_{j,j+1:n} \end{bmatrix}$
end

There are total of $N = n(n-1)/2$ possible pairs. We refer to N Jacobi updates as a sweep. There are various ordering schemes for the index pairs. For example, if $n = 4$, the cyclic-by-row ordering [2, Page 430] is:

$$(i,j) = (1,2),\ (1,3),\ (1,4),\ (2,3),\ (2,4),\ (3,4),\ (1,2), \ldots$$

The following is the cyclic-by-row Jacobi algorithm for the Takagi factorization, where

$$\mathrm{off}(A) \equiv \sqrt{2\sum_{i=1}^{n}\sum_{j=i+1}^{n}|a_{ij}|^2}.$$

Algorithm 3 (Cyclic-by-row Jacobi) *Given a complex-symmetric $A \in C^{n\times n}$ stored in its upper triangular part and a tolerance $tol > 0$, this algorithm overwrites the upper triangular part of A with the upper triangular part of $Q^{\mathrm{H}}AQ^*$, where Q is unitary and* $\mathrm{off}(Q^{\mathrm{H}}AQ^*) \le tol\,\|A\|_{\mathrm{F}}$.

$Q = I_n$;
$eps = tol\,\|A\|_{\mathrm{F}}$;
while $\mathrm{off}(A) > eps$
  for $i = 1 : n-1$
    for $j = i+1 : n$
      Apply Algorithm 1 to obtain the Takagi factorization (3);
      Apply Algorithm 2 to update the upper triangular part of A;
      $[\mathbf{q}_{1:n,i}\ \mathbf{q}_{1:n,j}] \leftarrow [\mathbf{q}_{1:n,i}\ \mathbf{q}_{1:n,j}]V(i,j)$;
    end
  end
end.

This algorithm requires $O(3n^3)$ floating-point operations without accumulating Q and $O(6n^3)$ floating-point operations with accumulating Q.

5 Parallel Jacobi Method

The cyclic-by-row ordering is sequential. For example, the two subsequent pairs $(1,2)$ and $(1,3)$ share index 1. The Jacobi update $(1,3)$ cannot start until the $(1,2)$ update finishes. Various ordering schemes suitable for parallel implementations have been proposed [2, Page 431]. In this section, we discuss the chess tournament ordering as an illustrative example.

Assuming n is even, say $n = 8$, we start with the first $n/2$ index pairs:

$$(1,2),\ (3,4),\ (5,6),\ (7,8) \qquad \begin{matrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{matrix}$$

Note that the above $n/2$ pairs share no common indices. Thus the $n/2$ Jacobi updates can be carried out on $n/2$ processors in parallel. The next $n/2$ pairs $(1,4)$, $(2,6)$, $(3,8)$, $(5,7)$ are generated as follows:

$$\begin{array}{ccccccc} 1 & & 3 & \to & 5 & \to & 7 \\ & \nearrow & & & & & \downarrow \\ 2 & \leftarrow & 4 & \leftarrow & 6 & \leftarrow & 8 \end{array} \quad \Rightarrow \quad \begin{array}{cccc} 1 & 2 & 3 & 5 \\ 4 & 6 & 8 & 7 \end{array}$$

A sweep is completed by the following pairs:

$$(1,6),\ (4,8),\ (2,7),\ (3,5) \qquad \begin{array}{cccc} 1 & 4 & 2 & 3 \\ 6 & 8 & 7 & 5 \end{array}$$

$$(1,8),\ (6,7),\ (4,5),\ (2,3) \qquad \begin{array}{cccc} 1 & 6 & 4 & 2 \\ 8 & 7 & 5 & 3 \end{array}$$

$$(1,7),\ (5,8),\ (3,6),\ (2,4) \qquad \begin{array}{cccc} 1 & 8 & 6 & 4 \\ 7 & 5 & 3 & 2 \end{array}$$

$$(1,5),\ (3,7),\ (2,8),\ (4,6) \qquad \begin{array}{cccc} 1 & 7 & 8 & 6 \\ 5 & 3 & 2 & 4 \end{array}$$

$$(1,3),\ (2,5),\ (4,7),\ (6,8) \qquad \begin{array}{cccc} 1 & 5 & 7 & 8 \\ 3 & 2 & 4 & 6 \end{array}$$

A straightforward implementation of the chess tournament algorithm would be analogous to Algorithm 3, in which only the upper triangular part of A is accessed. A close look, however, reveals some obstacles in parallelization in a shared address space. For example, both $(1,2)$ and $(3,4)$ updates modify the entries a_{13}, a_{14}, a_{23}, and a_{24}. The $(1,2)$ update writes to these four entries during its row update (4), whereas the $(3,4)$ update writes them during its column update (5). Thus, synchronization is necessary if $(1,2)$ update and $(3,4)$ update are carried out in parallel. The synchronization would reduce the parallelism. To alleviate this problem, we apply a group of $n/2$ independent row updates followed by a group of $n/2$ independent column updates. This requires whole matrix A. Now we assume that the whole A is stored.

Consider multithreading in a shared address space. Let p be the number of child threads, in addition to the main thread, used in the program. The main thread is responsible for initializing and updating the global index vectors and $\mathrm{off}(A)$, which determines the termination. The child threads carry out the Jacobi updates. The synchronization mechanism used is barrier. A barrier is associated with a group of threads. The function `barrier()` does not return until all the threads associated with this barrier have called `barrier()`.
Barrier implementation:

```
acquire lock;
if the last thread to reach barrier
    reset counter
    broadcast all waiting threads;
else
    update counter
    wait until woken up;
end;
release lock
```

In our method, only one barrier is used for synchronization all the threads at the following points:

- After the main thread initializes the index vector and $\mathrm{off}(A)$, so the child threads get the correct values of the index vector and $\mathrm{off}(A)$;
- After the child threads completes the row updates, so the child threads have correct values to start column updates;
- After the column updates. While the child threads are updating the columns, the main thread updates the index vector in parallel;
- After the main thread updates $\mathrm{off}(A)$, so all thread has the correct value of $\mathrm{off}(A)$ to check convergence.

To balance the work load, the $n/2$ pairs in a group are distributed among the p child threads cyclically.

Algorithm 4 (Parallel Jacobi) *Given a complex-symmetric* $A \in C^{n\times n}$ *(n even) and a tolerance tol* > 0*, this algorithm overwrites* A *with* $Q^{\mathrm{H}}A\bar{Q}$*, where* Q *is unitary and*

$\mathrm{off}(Q^{\mathrm{H}} A\bar{Q}) \leq tol\,\|A\|_{\mathrm{F}}$. *This algorithm uses the chess tournament ordering in the Jacobi method running on p threads in addition to the main thread.*

Main thread program

$m = n/2$;
% Initialize the index vectors
$indxi = [1, 3, ..., n-1]$; $indxj = [2, 4, ..., n]$;
Create p threads running thread program;
$eps = tol\,\|A\|_{\mathrm{F}}$;
$offA = \mathrm{off}(A)$;
% end of initialization
barrier(b);
while $offA > eps$
    % one sweep
    for $k = 1 : (n-1)$
        % wait threads for row updates
        barrier(b);
        % update index vector
        $tmp = indx1(m)$;
        $indxi[2:m]$
            $= [indxj(1), indxi(2:m-1)]$;
        $indxj = [indxj(2:m), tmp]$;
        barrier(b);
    end % a sweep
    % update $offA$
    $offA = \mathrm{off}(A)$;
    barrier(b);
end % while
join the threads.

Child thread program:

p = myid;
% wait for the initialization in main
barrier(b);
while $offA > eps$
    % one sweep
    for $k = 1 : (n-1)$
        % row updates
        $pt = p$;
        while $pt \leq m$
            $i = \min(indxi(pt), indxj(pt))$;
            $j = \max(indxi(pt), indxj(pt))$;
            Apply Algorithm 1 to obtain the
                Takagi factorization (3);
            Row update as shown in (4);
            $[\mathbf{q}_{1:n,i}\ \mathbf{q}_{1:n,j}] \leftarrow [\mathbf{q}_{1:n,i}\ \mathbf{q}_{1:n,j}]V(i,j)$;
            Save $V(i,j)$;
            $pt = pt + p$;
        end
        barrier(b);
        % column updates
        $pt = p$;
        while $pt \leq m$
            $i = \min(indxi(pt), indxj(pt))$;
            $j = \max(indxi(pt), indxj(pt))$;
            Retrieve $V(i,j)$;
            Column update as shown in (5);
            $pt = pt + p$;
        end
        % wait main to update the index pairs
        barrier(b);
    end % a sweep
    % wait main to update $offA$
    barrier(b);
end % while

We implemented our algorithms in C++ and Solaris threads and ran the program on a SUN SPARC station with four processors. We used UNIX command timing. The preliminary experiments showed that using two child threads the typical performance for a matrix of order 260 was:

```
66.26u 0.41s 0:39.15 170.2%
```

The total user time, sum of all threads, was 66.26 seconds, the system time 0.41 second, and the elapsed time 39.15 seconds.

6 Conclusion

In this paper, we present a parallel Jacobi method for computing the Takagi factorization. The chess tournament ordering is used to illustrate the synchronization and load balancing issues. Other ordering schemes can be used in a similar way.

References

[1] A. Bunse-Gerstner and W. B. Gragg, "Singular value decompositions of complex symmetric matrices," *Journal of Computational and Applied Mathematics*, **21** (1988) 41–54.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd Ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[3] Y. K. Ho, "The method of complex coordinate rotation and its applications to atomic collision processes," *Physics Reports (Review Section of Physics Letters)*, **99**, No. 1 (1983) 1–68.

[4] Roger A. Horn and Charles R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[5] G. Jolicard and E. J. Austin, "Optical potential stabilization method for predicting resonance levels," *Chemical Physics Letters*, **121** (1985) 106–110.

[6] F. T. Luk and S. Qiao, "A Fast Singular Value Algorithm for Hankel Matrices," In *Advanced Signal Processing Algorithms, Architectures, and Implementations XI*, Franklin T. Luk, Editor, Proceedings of SPIE Vol. 4474 (2001) 254–261.