

Parallel Block Tridiagonalization of Real Symmetric Matrices

Yihua Bai^{*§} and Robert C. Ward^{*§}
Technical Report UT-CS-06-578¹
June 2006

Abstract. Two parallel block tridiagonalization algorithms and implementations for dense real symmetric matrices are presented. Block tridiagonalization is a critical pre-processing step for the block-tridiagonal divide-and-conquer algorithm for computing eigensystems and is useful for many algorithms desiring the efficiencies of block structure in matrices. For an “effectively” sparse matrix, which frequently results from applications with strong locality properties, a heuristic parallel algorithm is used to transform it into a block tridiagonal matrix such that the eigenvalue errors remain bounded by some prescribed accuracy tolerance. For a dense matrix without any usable structure, orthogonal transformations are used to reduce it to block tridiagonal form using mostly level 3 BLAS operations. Numerical experiments show that block-tridiagonal structure obtained from this algorithm directly affects the computational complexity of the parallel block-tridiagonal divide-and-conquer eigensolver.

Categories and Subject Descriptors: G.1.3 [Numerical Analysis]: Numerical Linear Algebra - *Eigenvalues and eigenvectors (direct and iterative methods)*; G4 [Mathematical Software]: *Algorithm design and analysis; Parallel and vector implementations*; F.2.1 [Analysis of Algorithms and Problem Complexity]: Numerical Algorithms and Problems – *Computations on matrices*
General Terms: Algorithms, Performance
Additional Key Words and Phrases: Block-tridiagonal matrix, approximate eigenvalues, block divide-and-conquer, orthogonal block reduction

1. Introduction

Efficient and stable eigen-decomposition algorithms like QR iteration [Francis 1961, Wilkinson 1968], divide-and-conquer [Cuppen 1981, Dongarra and Sorensen 1987, Gu and Eisenstat 1994, 1995], and most recently, MRRR [Dhillon and Parlett 2004, Dhillon et al. 2004] compute eigen-solutions of a real symmetric matrix to full accuracy. The recently developed block-tridiagonal divide-and-conquer (BD&C) algorithm [Gansterer et al. 2003] represents a new development in this field. It computes the eigensystem of a symmetric block-tridiagonal matrix to reduced accuracy very efficiently without tridiagonalization. The BD&C algorithm benefits from the reduced accuracy requirement of the eigen-solutions as well as localized data-access pattern. The BD&C algorithm is particularly effective on nonlinear eigenproblems with strong locality properties, such as in modern electronic structure calculations. To meet the demand of solving large eigenvalue problems in electronic structure calculation, an efficient and scalable parallel implementation of the BD&C algorithm has been developed [Bai and Ward 2005].

^{*}Department of Computer Science, University of Tennessee, 203 Claxton Complex, 1122 Volunteer Blvd., Knoxville, TN 37996-3450.

[§]This work was partially supported by Oak Ridge National Laboratory’s Computing and Computational Sciences Directorate and the University of Tennessee’s Science Alliance Program.

¹Available from: <http://www.cs.utk.edu/~library/2006.html>

Electronic structure determines the properties of materials, such as the hardness, elasticity, conductivity, superconductivity, photoelectric efficiency, optical reflectivity and absorption, heat transfer, material structure, and so on. Therefore, the computation of electronic structure plays a critical role in simulation and discovery of new materials. One popular approach to these computations is based on the Hartree-Fock method. The Hartree-Fock equation is a non-linear integro-differential equation. This equation is further converted into a non-linear symmetric eigenvalue problem so that it can be solved numerically using an iterative procedure called the self-consistent field (SCF) method [Szabo and Ostlund 1996]. In each iteration of the SCF procedure, a linear real symmetric eigenvalue problem is solved. In early and intermediate iterations, it may be advantageous to require lower accuracy, especially if this significantly saves execution time. As SCF converges, higher accuracy may be used if necessary to obtain final results [Szabo and Ostlund 1996].

The BD&C algorithm is applicable only to block tridiagonal matrices. Although every matrix is trivially in block tridiagonal form using just two diagonal blocks, the real benefits of the BD&C algorithm occur when the number of diagonal blocks is much larger. So, block tridiagonal matrices in this paper refer to such matrices with more than 2 blocks.

Few matrices generated in electronic structure calculations are block-tridiagonal; most of them are dense with any structure determined by the chemical and physical properties of materials. For example, some materials have strong locality properties resulting in a matrix with its larger elements close to the diagonal and the elements away from the diagonal much smaller. Such matrices can be approximated with relatively high accuracy by block-tridiagonal matrices. Unfortunately, some matrices do not possess any usable structure and must be handled as general dense matrices.

In this paper, we present two different algorithms and their parallel implementations to transform symmetric dense matrices into block-tridiagonal form based on their structure. These algorithms are the essential pre-processing steps for the BD&C algorithm. If the input matrix is “effectively” sparse (see Section 2.1), we construct with very low computational cost a block-tridiagonal matrix whose eigensystem approximates that of the original input matrix. Otherwise, we use orthogonal transformations with very high ratio of level 3 BLAS operations to reduce efficiently a dense matrix to block-tridiagonal form.

The rest of this paper is arranged as follows. In Section 2, we present algorithms for the block-tridiagonalization of “effectively” sparse matrices and of dense matrices without specific structure. Parallel implementations are presented in Section 3 with the results and analyses of the numerical tests discussed in Section 4. We conclude our paper in Section 5.

2. Block-tridiagonalization algorithms

The BD&C algorithm computes approximate eigen-solutions of a symmetric block-tridiagonal matrix

$$M = \begin{bmatrix} B_1 & C_1^T & & & & \\ C_1 & B_2 & C_2^T & & & \\ & C_2 & \ddots & \ddots & & \\ & & \ddots & B_{q-1} & C_{q-1}^T & \\ & & & C_{q-1} & B_q & \end{bmatrix} \in R^{n \times n}$$

in three steps: problem subdivision, sub-problem solution and synthesis of sub-solutions [Gansterer et al. 2003]. Among these three steps, the synthesis step dominates the computational complexity of BD&C. It executes a sequence of matrix multiplications to accumulate eigenvectors. The calculation of eigen-solutions for each pair of adjacent sub-solutions is called a merging operation. For each merging operation, the number of matrix multiplications executed is determined by the approximate rank of the off-diagonal block that connects these two matrix sub-blocks. As the merging operations proceed, problem sizes become larger and larger. The last merging operation involves matrix multiplications of order n . One major goal of our block-tridiagonalization algorithms is to produce at least a few small block sizes for the last merging operation, in order to reduce computational complexity of BD&C.

Most matrices generated in real applications do not have a block-tridiagonal structure. We divide input matrices into two groups: 1) “Effectively” sparse with strong locality properties; 2) dense without usable structure. For the first type of matrix, in the context of approximate eigen-solutions with accuracy tolerance τ , we want to construct a block-tridiagonal matrix M from the original input matrix A such that the maximum eigenvalue error between those of A and M is bounded by $\tau \|A\|$. For the second type of matrices, we use blocked Householder transformations to reduce the original matrix A to block-tridiagonal form.

2.1. Block-tridiagonalization of “effectively” sparse matrices (BTS)

We call a dense matrix A “effectively” sparse with respect to a given accuracy tolerance τ , if it has the property that a large portion of its nonzero entries may be set to zero without influencing its eigenvalues by more than $\tau \|A\|$. The 6-step heuristic block-tridiagonalization algorithm [Bai et al. 2004] has been developed to transform a full matrix that is “effectively” sparse into a sparse matrix, and then find a block tridiagonal structure for the sparse matrix as shown in Fig. 1.

$$\begin{aligned}
A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{pmatrix} &\Rightarrow & \begin{pmatrix} a_{11} & 0 & a_{13} & 0 & \cdots & 0 \\ 0 & a_{22} & a_{23} & a_{24} & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & 0 & \cdots & a_{3n} \\ 0 & a_{42} & 0 & a_{44} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3} & 0 & \cdots & a_{nn} \end{pmatrix} \\
&\Rightarrow & \begin{pmatrix} B_1 & C_1^T & & & & \\ C_1 & B_2 & C_2^T & & & \\ & C_2 & B_3 & \ddots & & \\ & & \ddots & \ddots & C_{q-1}^T & \\ & & & C_{q-1} & B_q & \end{pmatrix} = M
\end{aligned}$$

Figure 1. Transform a full symmetric matrix into a block-tridiagonal matrix [Bai et al. 2004].

The BTS algorithm partitions τ into two parts, $\tau = \tau_1 + \tau_2$, allowing a portion of the acceptable error to be used for different steps in the algorithm. The 6 steps of the algorithm are briefly described below.

Step 1. Global threshold A with $\sqrt{\tau}\|A\|$

We start with a threshold $\tau' = \sqrt{\tau}$, larger than permitted by the accuracy requirement, and obtain matrix A' by eliminating all elements in A less than $\sqrt{\tau}\|A\|$. For many matrices resulting from modeling physical phenomena with strong locality properties, most of the elements will be eliminated. The resultant matrix A' will contain only the largest elements of A and would hopefully be sparse.

Step 2. Reorder A'

In this step, A' is reordered to reduce its bandwidth using the Gibbs-Poole-Stockmeyer (GPS) algorithm [Gibbs et al. 1976, Lewis 1982]. The bandwidth of A' may be significantly reduced by GPS reorder. Thus, the elements of A' are moved closer to the diagonal. The permutation matrix P accomplishing this task will be used in **Step 3**.

Step 3. Permute A with permutation matrix P from **Step 2**

The permutation matrix P computed in **Step 2** is applied to A , resulting in matrix $A'' = P^T A P$. The larger elements of A are expected to be closer to the diagonal in A'' .

Step 4. Target threshold A'' with $\tau_1\|A\|$.

In this step, we eliminate those elements far away from the diagonal in matrix A'' whose influence on the error of any eigenvalue is negligible compared to $\tau_1 \|A\|$. This step produces matrix A''' such that $A'' = A''' + E$, with $\|E\|_1 < \tau_1 \|A\|$.

It can be shown that the absolute difference between the eigenvalues λ_i of A'' and the eigenvalues λ'_i of A''' for $1 \leq i \leq n$ is bounded by [Demmel 1997, Golub and van Loan 1996]

$$|\lambda_i - \lambda'_i| \leq \|E\|_2 \leq \|E\|_1. \quad (2.1)$$

Since the eigenvalue errors are bounded by the 1-norm of the error matrix E , we may eliminate elements of A'' before each column-wise sum of absolute values of the dropped elements exceeds $\tau_1 \|A\|$. Note that due to symmetry, for each element a_{ij} ($1 \leq i, j \leq n$ and $i > j$) to be checked, its symmetric counterpart a_{ji} must be checked as well.

Step 5. Covering A''' .

The sizes of the diagonal blocks are determined such that the resulting block-tridiagonal matrix contains all the matrix elements that are effectively nonzero. These are the matrix elements whose effect on the accuracy of the eigenpair approximation may be non-negligible.

Step 6. Target block reduction (TBR).

As an option, the last step of the BTS algorithm attempts to produce a few small blocks for a lower computational complexity in the merging operation of the BD&C algorithm.

In **Step 4**, none of the matrix elements dropped are greater than the given error bound $\tau_1 \|A\|$. It may be possible to eliminate some of the matrix elements whose absolute values are larger than the given error bound without causing the accumulative error in the eigenvalues to exceed this error. For real symmetric matrix A''' , the sensitivity analysis on the eigenvalue error as a result of zeroing matrix element a'''_{ij} can be estimated by [Wilkinson 1965]

$$\Delta\lambda = 2a'''_{ij}x_i x_j + O\left((a'''_{ij})^2\right), \quad (2.2)$$

where x_i, x_j are the i -th and j -th entries, respectively, of the eigenvector x corresponding to λ . Several elements may be eliminated as long as the maximum of the sum of the eigenvalue errors is less than the error bound $\tau_2 \|A\|$.

Step 6 is only possible if an approximation for the eigenvectors is available. For an iterative method solving a non-linear eigenvalue problem, like the SCF method, we may use the eigenvectors from the previous iterations as an approximation. There may be other similar applications with eigenvector approximations permitting this last step in the algorithm. Using

TBR, we may reduce the size of a few diagonal blocks in the hope that the corresponding off-diagonal blocks have a lower rank.

2.2. Orthogonal block-tridiagonal reduction of dense matrices (OBR)

If a dense symmetric matrix cannot be transformed into a block-tridiagonal matrix for use by the BD&C algorithm with little computational effort as described in Section 2.1, we use a sequence of blocked Householder transformations to reduce the dense matrix to a block-tridiagonal one with reasonably small block sizes. Similar algorithms have been investigated for the reduction of a matrix from dense form to banded, then to tridiagonal [Bishof et al. 2000, 2000, Bishof and Van Loan 1987]. The algorithm reducing the matrix from dense to banded form has been shown to have a good data access pattern and large portion of level 3 BLAS operations [Bishof et al. 2000, Dongarra et al. 1989, Gansterer et al. 1998].

Given a dense symmetric matrix $A \in \mathbb{R}^{n \times n}$, we desire to apply a sequence of orthogonal transformations to reduce A to a block-tridiagonal matrix M . The orthogonal transformations zero out elements below the block subdiagonal panel by panel as shown in Figures 2 and 3.

In a general case, let p_b be the number of matrix columns in a blocked orthogonal transformation, then $n_i = p_b$ is the width and $m_i = n - ip_b$ is the length of the i -th panel G_i , b is the block size of the reduced block-tridiagonal matrix. Figures 2 and 3 illustrate the case when $p_b = b$. For blocked orthogonal reduction scheme using Householder transformations, p_b must be no greater than b in order to achieve high ratio of level 3 BLAS operations [Bai 2005].

Let $A_i \in \mathbb{R}^{(m_i+n_i) \times (m_i+n_i)}$ as illustrated by Figures 2 and 3 be the lower right principal submatrix of A at the i -th stage of orthogonal reduction. For each matrix panel $G_i \in \mathbb{R}^{m_i \times n_i}$ in Figure 2, its QR factorization $G_i = Q_i R_i$ where $Q_i \in \mathbb{R}^{m_i \times m_i}$ and $R_i \in \mathbb{R}^{m_i \times n_i}$ is used to reduce A to a block tridiagonal matrix. Partitioning $R_i = \begin{bmatrix} \widehat{R}_i \\ 0 \end{bmatrix}$, where $\widehat{R}_i = (R_i)_{(1:n_i, 1:m_i)}$ is upper triangular, and A_i as

$$A_i = \begin{bmatrix} n_i & m_i \\ A_{11}^i & A_{12}^i \\ A_{21}^i & A_{22}^i \end{bmatrix} \begin{matrix} n_i \\ m_i \end{matrix}, \quad (2.3)$$

then applying Q_i , we have

$$\begin{bmatrix} I & O \\ O & Q_i^T \end{bmatrix} A_i \begin{bmatrix} I & O \\ O & Q_i \end{bmatrix} = \begin{bmatrix} A_{11}^i & R_i^T \\ R_i & \widetilde{A}_{22}^i \end{bmatrix}. \quad (2.4)$$

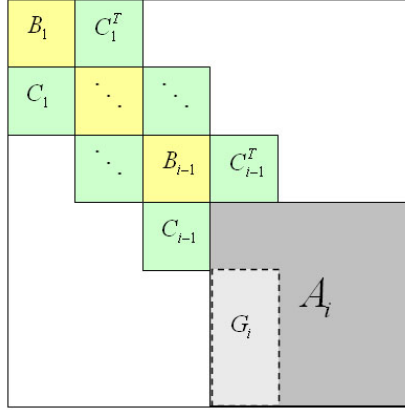


Figure 2. Matrix A at the i -th stage of orthogonal reduction.

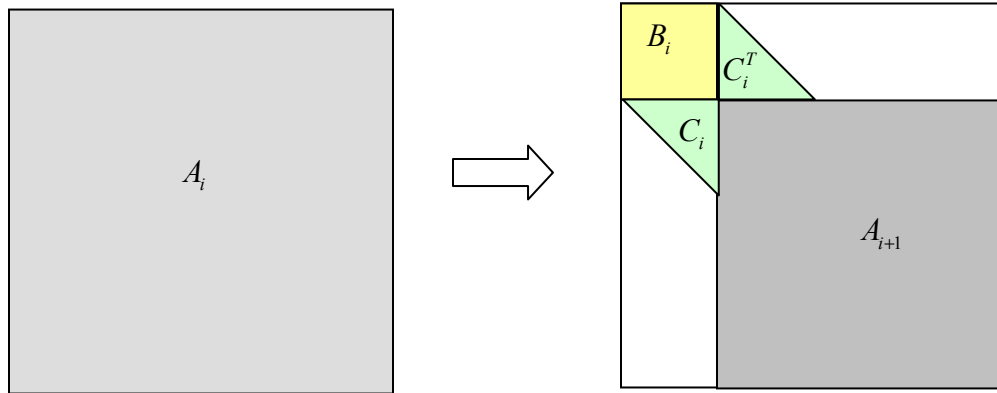


Figure 3. One step of block reduction for submatrix A_i

The diagonal block A_{11}^i and off-diagonal block \widehat{R}_i can be obtained directly, and $A_{i+1} = \widetilde{A}_{22}^i = Q_i^T A_{22}^i Q_i$. We continue this procedure until the whole matrix A is reduced to a block tridiagonal matrix M . All the subdiagonal blocks of M except the last one are upper triangular. The panel width p_b needs to be chosen carefully. It should be small enough to keep the cache miss rate low yet large enough to benefit from data-reuse in level 3 BLAS operations.

Computational complexity analyses show that if $p_b \leq b$ where p_b is the algorithmic panel width and b is the block size in the orthogonal block tridiagonal reduction algorithm, the ratio of level 3 BLAS operations increases with matrix size n for fixed b [Bai 2005]. As shown in Section 3.2.1, the relationship between p_b and b will become important in any parallel

implementation of this algorithm. Figure 4 shows that the ratio of level 3 BLAS operations exceeds 90% quickly as the matrix size increases.

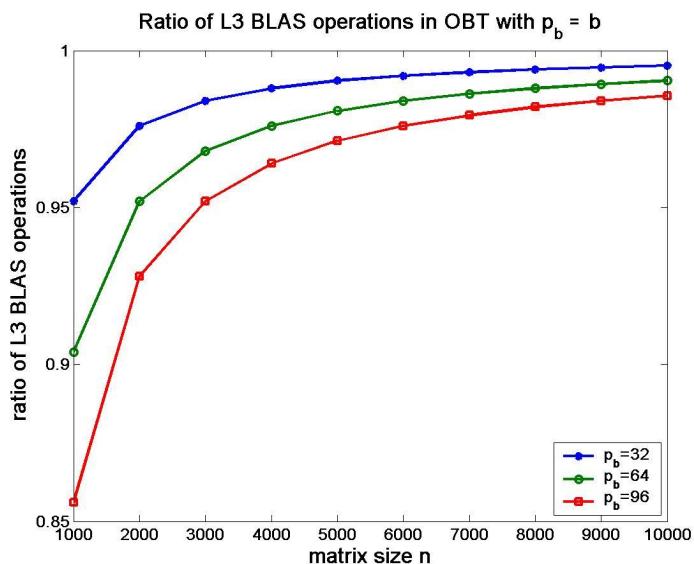


Figure 4. Ratio of level 3 BLAS operations in OBR with $p_b = b$.

3. Parallel implementation details

The BTS algorithm typically has very low computational complexity. For an effective parallel implementation, the major issue is to choose a proper parallel matrix distribution pattern and limit the complexity of data communication. The OBR algorithm is comprised mainly of level 3 BLAS operations. The key toward an efficient implementation of OBR is the proper choice of sizes b for tridiagonal blocks, p_b for algorithmic panel width, and n_b for 2D block cyclic matrix distribution. In this section, we use α , β and γ to denote the time to start up a data transfer, time to transfer one data item, and time to perform one floating point computation, respectively.

3.1 Parallel BTS implementation

The BTS algorithm is heuristic and inherently sequential. The floating-point operations in the algorithm are mainly comparisons and additions, and typically its operation count is $O(n^2)$ [Bai et al. 2004]. Since the original matrix A is symmetric, an operation on any entry a_{ij} inevitably involves its symmetric counterpart a_{ji} . If the matrix is not distributed properly, the performance of parallel BTS could degrade severely as the matrix size n and the number of processors p increases.

The 2D block cyclic matrix distribution, which is frequently used in scalable parallel dense matrix algorithms, is not the most suitable data distribution pattern for the BTS algorithm since the matrix must be traversed column-wise numerous times. Intuitively, a 1D column block distribution with n/p matrix columns assigned to each processor, as shown in Figure 5, for the matrix is most desirable.

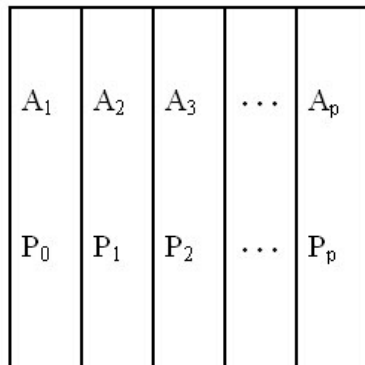


Figure 5. Matrix A distributed in column blocks.

If the original input matrix A is distributed in a 2D block cyclic pattern, then it must be redistributed from 2D to 1D for the parallel implementation of the BTS algorithm. If we assume that the system buffer is large enough so that each message can be sent and received without being partitioned into several smaller packages and point-to-point communication (i.e., send and receive) cannot be overlapped, then the total communication cost in the worst case for matrix redistribution from 2D block cyclic pattern to 1D column block pattern is

$$t_{2D \rightarrow 1D} = p^2 \alpha + n^2 \beta.$$

After the matrix has been redistributed, each processor holds n/p columns, and the parallel block tridiagonalization is then applied.

In the following sections, we describe in detail the parallel implementation of the 6-step BTS algorithm with the matrix distributed in 1D column blocks. The accuracy tolerance τ is partitioned as $\tau = \tau_1 + \tau_2$ for target threshold and optional target block reduction, respectively.

3.1.1 6-steps of parallel BTS

Step 1. Parallel global threshold with $\sqrt{\tau} \|A\|$.

This step is an embarrassingly parallel process. Every processor simultaneously drops elements $a_{ij} < \sqrt{\tau} \|A\|$, and stores indices of all elements $a_{ij} \geq \sqrt{\tau} \|A\|$ in compressed sparse row (CSR) format. The resultant matrix A' is expected to be very sparse and all its nonzero entries can be

stored on one processor. After thresholding, each processor sends its vectors of indices of nonzeros to a master processor. The master processor stores indices of all the nonzeros of A' . The collection of indices of nonzeros takes $2p\alpha + (n + nnz_1)\beta$ communication time where nnz_1 is the number of nonzeros in A' . No floating-point operations are involved in this step.

Step 2. Sequential matrix reorder for parallel BTS.

The most thoroughly studied and parallelized sparse matrix ordering algorithms are nested dissection and minimum degree algorithms, which are used to minimize the fill-in during LU factorization of matrices in sparse linear systems [Heath et al. 1990]. Scalable and efficient parallel implementations of those algorithms such as ParMetis [Karypis and Kumar 1998] are available. However, the purpose of matrix reordering in the BTS algorithm is to minimize the bandwidth of a sparse matrix, and the nested dissection and minimum degree methods do not directly attack this objective. We use the Gibbs-Poole-Stockmeyer (GPS) algorithm [Gibbs et al. 1976, Lewis 1982] for this step since it has also undergone thorough studies and considerable testing and bandwidth minimization is its goal.

The matrix after global thresholding is expected to be very sparse and can be stored on the local memory of one processor. Since the reordering consumes a small fraction of computational time of the BTS, we do not parallelize the reordering step. Instead, only the master processor that contains the indices of all nonzeros of A' performs matrix reordering using the GPS algorithm, while all other processors stay idle. After the permutations are determined, the master processor broadcasts the permutation matrix P to all other processors.

Step 3. Parallel symmetric permutation of A .

The permutation matrix P from *step 2* is used to permute the matrix A to produce the matrix $A'' = P^T A P$. Parallel symmetric matrix permutation can be an expensive step. As shown by the blue arrow in Figure 6, if two matrix columns are on different processors, the swap of those two columns invokes communication. In such a case, the communication cost of each swap is $2\alpha + 2n\beta$. Permutation of matrix rows does not involve any communication. If two rows of a matrix are to be swapped, local data on each processor are exchanged as shown by the red arrow in Figure 6. Thus, the worst-case communication cost is bounded by $n\alpha + n^2\beta$.

Because of the potentially heavy communication, matrix permutation is executed only when it can significantly reduce the bandwidth of A' . For the parallel implementation of BTS algorithm, we permute A when the bandwidth of A' can be reduced by at least 20%.

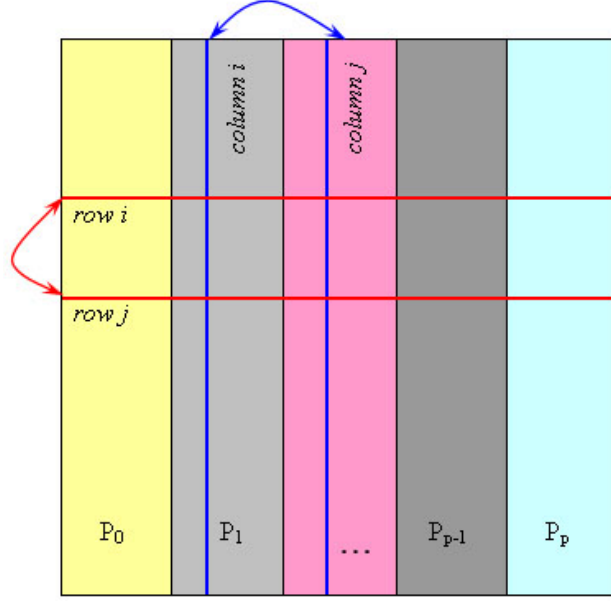


Figure 6. Swaps of rows and columns in parallel matrix permutation.

Step 4. Parallel target threshold with $\tau_1 \|A\|$.

In the BTS algorithm, all elements far away from the diagonal of matrix A'' are eliminated if their influence on the error of any eigenvalue is less than $\tau_1 \|A\|$. The resultant matrix is $A''' = A'' + E$, with $\|E\|_1 \leq \tau_1 \|A\|$. Since A'' is symmetric, for each element a''_{ij} in the lower triangular part of A'' that is checked for elimination, its symmetric counterpart a''_{ji} in the upper triangular of A'' must also be checked. Elements a''_{ij} and a''_{ji} can be dropped only when the sum of the absolute values of the dropped elements in both the i -th and the j -th column of A'' is less than $\tau_1 \|A\|$.

In a parallel matrix distribution, chances are that entries a''_{ij} and a''_{ji} are often on two different processors requiring communications between those two processors, in order to inform each other whether a''_{ij} and a''_{ji} can be dropped simultaneously or not. On the average, this leads to $O(n^2)$ communications.

The communication overhead can be reduced drastically if elements on each processor can be checked independently without communication. For this purpose, the error bound τ_1 is further split into two equal parts of $\frac{1}{2} \tau_1$ for the parallel target threshold. Consequently, the error matrix

E is also split into two parts: $E = E_1 + E_2$, where E_1 is an upper triangular matrix and E_2 is a lower triangular matrix.

The lower triangular part of A'' is first checked column by column. An element in the lower triangular part of A'' can be eliminated if the sum of the absolute values of the dropped elements in that column is less than $\frac{1}{2} \tau_1 \|A\|$. This guarantees that the error matrix E_1 satisfies

$\|E_1\|_1 \leq \frac{1}{2} \tau_1 \|A\|$. After that, the sum of the absolute values of all dropped elements in each column of A'' is broadcasted so that each processor contains a copy of the accumulated error for each matrix column. Then the upper triangular part of A'' is checked in a similar way. This guarantees that the error matrix E_2 which contains all the dropped elements in the upper triangular part of A'' satisfies $\|E_1 + E_2\|_1 \leq \tau_1 \|A\|$.

For each eliminated element a''_{ij} , its symmetric counter part a''_{ji} is not necessarily eligible for elimination, and vice versa. In general, from the above procedure, E_1 does not equal E_2^T . Therefore, the sum of those two matrices, $E_1 + E_2$, is not symmetric. Since matrices E and A''' must be symmetric, we need to symmetrize $E_1 + E_2$. For the i -th column and row of A''' , $1 \leq i \leq n$, the row index of the last nonzero of column i and the column index of the last nonzero of row i is compared. The larger index is chosen as the index of the last nonzero for the i -th row and column as illustrated in Figures 7 and 8. The communication cost in this modified parallel target threshold algorithm is only $(2\alpha + 2n\beta) \log p + 2\alpha p + 2n\beta$.

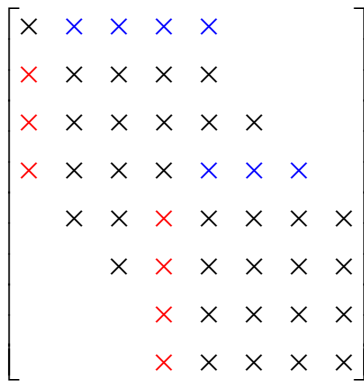


Figure 7. A''' after separate lower and upper triangular eliminations.

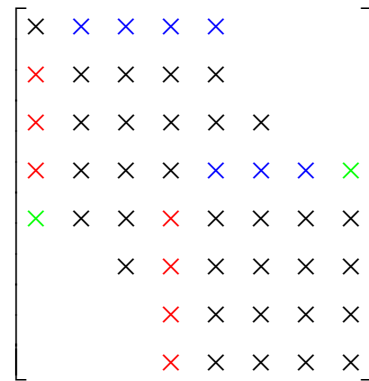


Figure 8. Symmetrize A''' by adding back nonzeros.

By using the above approach in our parallel target threshold algorithm, we may not be able to drop as many elements as we mathematically could and as in the sequential BTS algorithm [Bai et al. 2004]. However, the difference in the bandwidths produced by sequential BTS and parallel BTS is typically small as our test results of application matrices show (see Section 4.1.1).

Step 5. Covering A''' .

After the parallel target threshold step, all processors obtain the row indices of the last nonzero entries in each column of matrix A''' . Each processor redundantly determines the sizes of the diagonal blocks independently as in the sequential BTS algorithm, so that the resulting block tridiagonal matrix contains all the matrix elements that are effectively nonzero (i.e., nonzeros in A''').

Step 6. Parallel target block reduction using $\tau_2 \|A\|$.

The BTS algorithm provides the option of Target Block Reduction (TBR) to produce a few small blocks in matrix A''' for a lower computational complexity in the merging operations of the BD&C algorithm [Bai et al. 2004]. In a merging operation of BD&C, a lower rank of the off-diagonal block leads to a lower computational complexity. Since the ranks of off-diagonal blocks are not available during block tridiagonalization, we use the smaller dimension of an off-diagonal block as an approximation to its rank. TBR uses sensitivity analysis to check elements in each column/row of an off-diagonal block from outside toward inside for elimination. For the sensitivity analysis, approximations to the eigenvectors are required (see Section 2.1 *step 6*). If approximate eigenvectors are not available, we may set $\tau_1 = \tau$ and $\tau_2 = 0$ so that this optional step is not applied.

For the parallel implementation of sensitivity analysis, we assume that the approximate eigenvector matrix Z is distributed in 2D block cyclic pattern on a processor grid with r processor rows and c processor columns as would typically be the case. When rows of the approximate eigenvector matrix are required, they are sent from several processors to one processor. That is, for each entry a'''_{ij} to be checked, the i -th and j -th rows of the eigenvector matrix Z need to be sent to the processor that possesses a'''_{ij} (see Equation 2.2), which costs $2c\alpha + 2n\beta$ communication time. When several matrix entries in the same column/row are checked for elimination, the strategy used in parallel TBR is to send all the relevant rows in the eigenvector matrix to the processor that is applying the sensitivity analysis. For example, as shown in Figures 9 and 10, if we want to check elements b_{25} , b_{52} , b_{35} and b_{53} of a matrix B for elimination, rows 2, 3, and 5 of the eigenvector matrix Z (red shade in Figure 10) are sent to

P_2 , since b_{25} and b_{35} are both on processor P_2 . The updated block size will then be broadcast to all other processors.

To find diagonal blocks eligible for sensitivity analysis, parallel TBR starts with the smallest diagonal block. If there are several diagonal blocks with the same size, then the diagonal block closest to the middle of A'' will be selected. The reduction of the size of one diagonal block leads to the expansion of its neighboring diagonal block(s). To avoid oscillation in block sizes, after a diagonal block has been compressed, it should not be expanded. Next, the second smallest diagonal block is selected in a similar manner for sensitivity analysis, and so on.

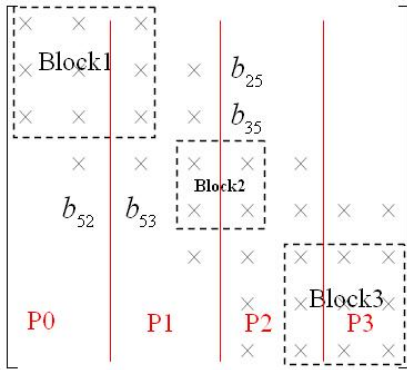


Figure 9. Check matrix entries (2,5), (3,5), (5,2) and (5,3).

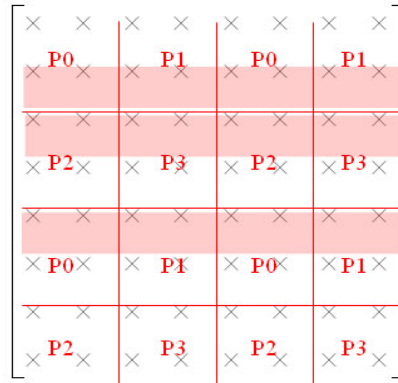


Figure 10. Rows in the eigenvector matrix Z for sensitivity analysis.

In the sequential BTS implementation, all eligible diagonal blocks are checked for block size reduction [Bai et al. 2004]. However, this may be too costly for parallel BTS implementation since communication overhead for collecting and distributing rows of the eigenvector matrix can be prohibitive. Since our goal is to find small blocks in an attempt to reduce the complexity of the last few merging operations of BD&C, we restrict the number of diagonal blocks to be checked to three.

3.1.2 Complexity of parallel BTS

In the sequential BTS algorithm, the computational complexity and the number of data accessed are both $O(n^2)$ [Bai et al. 2004]. For the parallel implementation of BTS algorithm,

computational complexity per processor is $O(n^2/p)$. The extra communication cost becomes the dominant part of the execution time since the time to transfer one floating-point number is typically much longer than to execute a floating-point operation.

The total computational and communication cost for parallel BTS as well as the complexity of matrix redistribution between 2D and 1D at the beginning and end of parallel BTS is approximately [Bai 2005]

$$T_{Parallel\ BTS} = \gamma \left(\frac{n^2}{p} + 2kn \right) + \alpha (2p^2 + n + 2kc) + \beta (3n^2 + 2kn)$$

where k denotes the number of matrix elements that are checked for elimination in step 6 (typically $k \ll n$), and c is the number of processor columns in the 2D distribution of the eigenvector matrix.

3.2. Parallel OBR implementation

The OBR algorithm reduces a dense matrix A to block tridiagonal form using a sequence of QR factorizations on column blocks of A , as shown in Figures 2 and 3. The resultant block tridiagonal matrix is similar to a banded matrix except that the last off-diagonal block is not a triangular. With minor modification, the parallel implementation of the OBR algorithm can also be used to reduce a symmetric dense matrix to banded form.

There are parallel implementations of the orthogonal bandwidth reduction algorithm, which reduce a dense matrix to banded matrix using blocked Householder transformations. One implementation is based on the Chamelon parallel programming tools [Bishof et al. 1994, Bishof et al. 1993], and another one uses PLAPACK [Wu et al. 1996]. Our parallel orthogonal block-tridiagonal reduction is implemented under ScaLAPACK environment. Input matrix $A \in \mathbb{R}^{m \times n}$ is distributed in 2D block cyclic pattern adopted by ScaLAPACK. A critical issue in the orthogonal reduction from a dense matrix to a block tridiagonal one is how to choose the algorithmic panel width p_b of each QR factorization, the block size b of the resultant block-tridiagonal matrix, and the block size n_b for 2D block cyclic matrix distribution.

3.2.1 Determination of b , p_b and n_b

First, we consider the relationship between b and p_b . Block size b directly affects the computational complexity of the BD&C merging operations. When b is large, the rank of the off-diagonal blocks tends to be large as well, which increases the computational complexity of the BD&C merging operations. Therefore we wish to obtain a block tridiagonal matrix with small block size b . However, as explained in Section 2.2, b should not be smaller than p_b .

Therefore, we set $b = p_b$. If we choose a small panel width p_b , the resultant block size b is also small, but we may not be able to obtain full performance of level 3 BLAS operations. If p_b is large, we may obtain slightly better performance during the reduction; but then b will be large and the rank of the off-diagonal blocks will likely be large as well. The reduction of execution time in OBR is not likely to compensate the increased execution time from BD&C. See Section 4.2 for some experimental results on p_b .

Second, we consider the relationship between n_b and p_b . Since matrices are distributed using ScaLAPACK 2D block cyclic pattern, to reduce data transfer between processor columns and the complexity of local index calculation, p_b should equal n_b , as the ScaLAPACK reduction subroutine PDSYTRD does. This guarantees that QR factorization of each matrix column block is performed on only one processor column, and does not involve row-wise communication in the processor grid.

From the above two restrictions $b = p_b$ and $p_b = n_b$, we fix the sizes of panel width and diagonal blocks of the reduced block tridiagonal matrix to be the block size of the parallel 2D block cyclic matrix distribution, i.e., $p_b = b = n_b$. ScaLAPACK recommends a block size of 64 for parallel matrix distribution [Blackford et al. 1997].

3.2.2 Complexity of parallel OBR

In parallel OBR, there are four steps to compute a sequence of n_b Householder transformations and reduce column block G_i of matrix A as shown in Figure 2. These four steps are:

- 1) Compute QR factorization of each column block G_i . The computed Householder vectors overwrite corresponding columns of G_i .
- 2) Construct blocked Householder transformation in the form of $I - Y_i W_i^T = H_{n_b} \cdots H_1$. Here Y_i holds n_b columns of Householder vectors $y_j^i, 1 \leq j \leq n_b$, and W_i holds vectors $2y_j^i / (y_j^i)^T y_j^i$.
- 3) Compute submatrix $Z_i = A_{22}^i W_i - \frac{1}{2} Y_i W_i^T A_{22}^i W_i$ where A_{22}^i is as shown in Equation 2.3.
- 4) Apply symmetric rank- $2n_b$ update $A_{i+1} = A_{22}^i - Y_i Z_i^T - Z_i Y_i^T$. A symmetric rank- $2n_b$ update requires only half of the computation as that of a non-symmetric update, but the communication cost cannot be reduced.

With p processors that form a $r \times c$ processor grid, where r and c are the number of rows and columns of the grid, the computational and communication cost for each processor is approximately [Bai 2005]

$$T_{Parallel\ OBR} = \frac{4n^3}{3p} \gamma + \alpha \frac{2rn}{n_b} + \beta n^2 \left(\frac{2 \log r}{c} + \frac{\log c}{r} \right).$$

4. Numerical tests

In this section, we present results of accuracy and performance tests. Our tests were run on the IBM p690 system nicknamed Cheetah in Oak Ridge National Laboratory. The system is comprised of twenty-seven 32-processor nodes. Each compute node has 1.3 GHz Power4 processors and at least 32MB SMP memory. The CPUs have 32 KB of Level-1 data cache and 64 KB of Level-1 instruction cache, 1.5 MB of Level-2 cache shared between two processors, and 32 MB of off-chip Level-3 cache. The system employs the Federation interconnection switch that supports about 1400 MBps MPI bandwidth and about 7 μ s MPI latency.

4.1 Test of parallel BTS implementation

For “effectively” sparse matrices with larger elements closer to the diagonals, parallel BTS subroutine PDSBTRI (URL: <http://www.cs.utk.edu/~cape/parallel/PBTri.tar.gz>) as described in section 3.1 is used to construct block-tridiagonal matrices. In Sections 4.1.1, we compare the resultant bandwidths of matrices using parallel and sequential target thresholding implementations. In Section 4.1.2, the ranks of off-diagonal blocks from parallel BTS with and without the optional TBR are compared. In Section 4.1.3, the accuracy of the eigenpairs for the block tridiagonal matrix is compared with that of the original full matrix. Performance test results are presented in Section 4.1.4.

Test matrices are generated from simulating alkane molecules using the CNDO method [Pople and Beveridge 1970, Pople et al. 1967, Pople et al. 1965, Pople and Segal 1965, 1966] and trans-Polyacetylene (PA) molecules using the SSH Hamiltonian [Su et al. 1980] that includes only the nearest neighboring atoms. The general molecular formula of an alkane is C_nH_{2n+2} . Figures 11 and 12 show the magnitudes of the elements of a matrix generated from $C_{502}H_{1006}$ and its eigenvalue distribution. Trans-PA consists of a chain of CH units. It has the general molecular formula $trans-(CH)_n$. Figures 13 and 14 show the magnitudes of the matrix elements of $trans-(CH)_{8000}$ and its eigenvalue distribution.

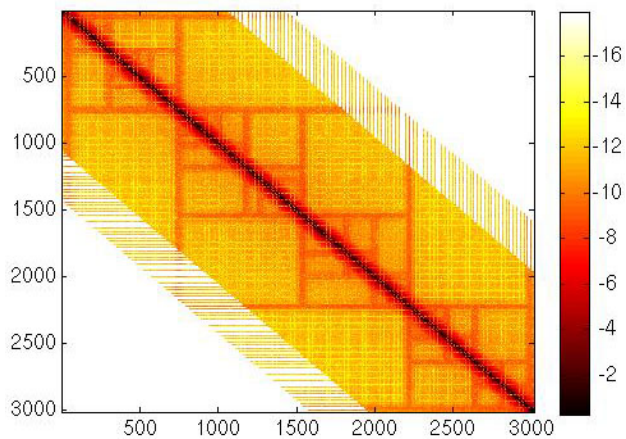


Figure 11. \log_{10} of absolute value of matrix elements for alkane $C_{502}H_{1006}$ molecule, $n = 3,014$.

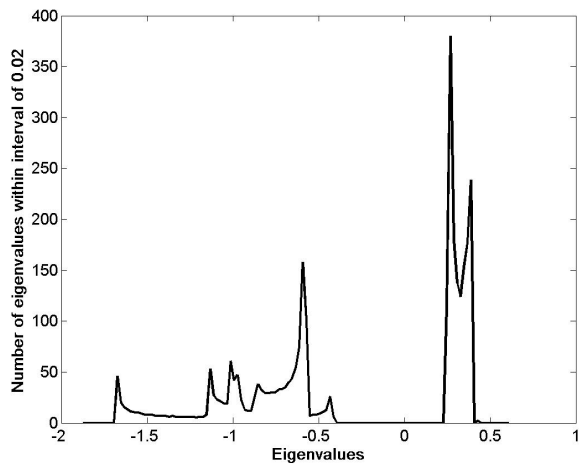


Figure 12. Eigenvalue distribution of matrix in Fig. 11.

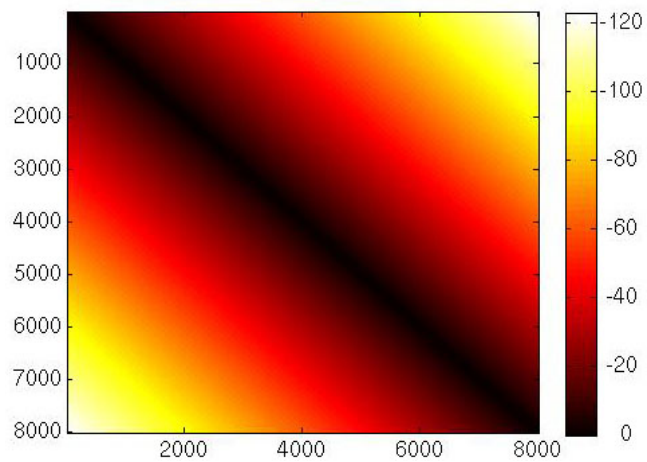


Figure 13. \log_{10} of absolute value of matrix elements for $\text{trans}-(\text{CH})_{8000}$ molecule, $n = 8,000$.

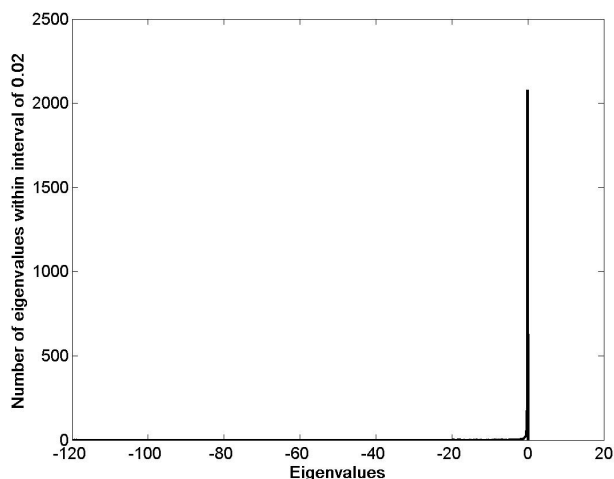


Figure 14. Eigenvalue distribution of matrix in Fig. 13.

4.1.1 Bandwidths of modified parallel target threshold

As described in Section 3.1.1, *step 4*, parallel target threshold checks lower and upper triangular part of matrix A'' independently without communication, while sequential target threshold checks each pair of symmetric matrix entries along off-diagonals. The bandwidths after parallel and sequential target threshold are compared using application matrices generated from alkane and trans-PA molecules. Table 1 shows that the bandwidth after parallel target threshold is very close to that after sequential threshold on the test matrices.

Table 1. Bandwidth after parallel and sequential target threshold, $\tau = 10^{-6}$

Trans-PA molecules			Alkane molecules		
Matrix and Size	Bandwidth after target threshold		Matrix and Size	Bandwidth after target threshold	
	Parallel	Sequential		Parallel	Sequential
Trans-(CH) ₂₀₀₀ $n = 2000$	390	390	C ₁₆₂ H ₃₂₆ $n = 974$	81	80
Trans-(CH) ₃₀₀₀ $n = 3000$	390	390	C ₃₂₂ H ₆₄₆ $n = 1934$	83	81
Trans-(CH) ₄₀₀₀ $n = 4000$	390	390	C ₅₀₂ H ₁₀₀₆ $n = 3014$	85	85

4.1.2 Effect of target block reduction (TBR) on block sizes and ranks

In an iterative method like the SCF, a nonlinear eigenvalue problem is solved by solving a linear eigensystem iteratively until convergence [Szabo and Ostlund 1996]. For the alkane molecules, we completed all the iterations using a sequential SCF subroutine and stored the Fock matrices and eigenvector matrices from each iteration. Thus, in the test of parallel BTS using matrices from $C_{502}H_{1006}$, we are able to test the optional target block reduction (TBR) step using the eigenvector matrix from the previous iteration as approximate eigenvectors. The TBR step tries to produce a few small blocks in matrix A''' for a lower computational complexity in the merging operations of the BD&C algorithm. Table 2 shows the minimum size of diagonal blocks and minimum rank of off-diagonal blocks with and with out TBR using alkane matrices from the second iteration of SCF. The TBR step significantly reduces the size of some off-diagonal blocks when work space is sufficient. When tolerance is 10^{-8} , the work space required for TBR is too large due to large block sizes. In this case, TBR step is replaced by a second round of target threshold. The execution times in the table show that the overhead of TBR is typically small compared to the total execution time of parallel BTS. Although the reduction in the minimum rank of off-diagonal blocks is not as significant as that in the minimum block size, even a reduction of one is a reduction of $O(n^3)$ in total floating-point operations for the BD&C algorithm.

Table 2. Minimum size of diagonal blocks and minimum rank of off-diagonal blocks from block-tridiagonalization of alkane matrixes using 4 processors.

Tolerance	Molecule	Matrix size	Minimum size of diagonal blocks		Minimum rank of off-diag. blocks		Execution time in seconds	
			without TBR	with TBR	without TBR	with TBR	without TBR	with TBR
10^{-4}	$C_{162}H_{326}$	974	40	21	14	14	0.28	0.30
	$C_{322}H_{646}$	1974	40	21	14	14	0.21	0.30
	$C_{502}H_{1006}$	3014	40	26	14	14	0.16	0.26
10^{-6}	$C_{162}H_{326}$	974	76	38	21	20	1.03	1.12
	$C_{322}H_{646}$	1974	76	39	20	20	0.88	1.19
	$C_{502}H_{1006}$	3014	74	39	21	21	0.65	1.04
10^{-8}	$C_{162}H_{326}$	974	90	90	47	47	2.38	2.59
	$C_{322}H_{646}$	1974	130	130	46	46	2.18	2.96
	$C_{502}H_{1006}$	3014	319	319	34	34	1.53	2.57

4.1.3 Accuracy of eigenpairs

To examine the actual errors in eigen-solutions incurred by the parallel BTS algorithm, LAPACK routine DSYEVD is used to compare the eigenpairs of A , the original matrix, to those of M , the approximating block tridiagonal matrix. The results are presented in Table 3, where λ_i and $\hat{\lambda}_i$ are the i -th eigenvalues of A and M , respectively, and v_i and \hat{v}_i are the corresponding eigenvectors. Accuracy tolerance is set to 10^{-4} , 10^{-6} and 10^{-8} , respectively. Table 3 shows that the scaled maximum error between the eigenvalues of A and M for each test matrix

$|\lambda(A) - \lambda(M)| / \|A\|$ is bounded by τ , and the scaled 2-norm of the maximum residual

$\|(A - \hat{\lambda}_i)\hat{v}_i\|_2 / \|A\|$ is of order $O(\tau)$.

Table 3. Maximum eigenvalue error and residual with $\tau = 10^{-4}, 10^{-6}$ and 10^{-8}

Matrix	Size	With TBR	Tolerance	$\max_{i=1, \dots, n} \frac{ \lambda_i - \hat{\lambda}_i }{\ A\ }$	$\max_{i=1, \dots, n} \frac{\ (A - \hat{\lambda}_i)\hat{v}_i\ _2}{\ A\ }$
C ₅₀₂ H ₁₀₀₆	3,014	No	10^{-4}	6.02×10^{-6}	1.07×10^{-5}
		Yes	10^{-4}	5.36×10^{-6}	2.46×10^{-5}
Trans-(CH) ₄₀₀₀	4,000	No	10^{-4}	1.07×10^{-5}	1.67×10^{-5}
C ₅₀₂ H ₁₀₀₆	3,014	No	10^{-6}	3.96×10^{-8}	8.85×10^{-8}
		Yes	10^{-6}	4.73×10^{-7}	4.24×10^{-6}
Trans-(CH) ₄₀₀₀	4,000	No	10^{-6}	7.56×10^{-8}	1.43×10^{-7}
C ₅₀₂ H ₁₀₀₆	3,014	No	10^{-8}	7.61×10^{-12}	2.35×10^{-10}
		Yes	10^{-8}	4.70×10^{-12}	2.11×10^{-10}
Trans-(CH) ₄₀₀₀	4,000	No	10^{-8}	5.35×10^{-10}	1.15×10^{-9}

4.1.4 Performance of PDSBTRI

The execution times of subroutine PDSBTRI with tolerance $\tau = 10^{-6}$ is illustrated in Figure 15. The parallel BTS algorithm contains steps that are sequential in nature as well as steps that parallelize well. For example, the matrix reorder (*step 2*) is completely sequential; while the global thresholding (*step 1*) and the modified target thresholding (*step 4*) are embarrassing (or

pleasantly) parallel, in that each processor checks elements to be eliminated independently. As the number of processors increases, the execution times for steps 2 and 4 decrease. However, the overhead of redistributing the matrix from 2D block cyclic distribution to 1D column block distribution increases with the number of processors. Overall, the execution time of PDSBTRI demonstrates a complexity of $O(n^2)$.

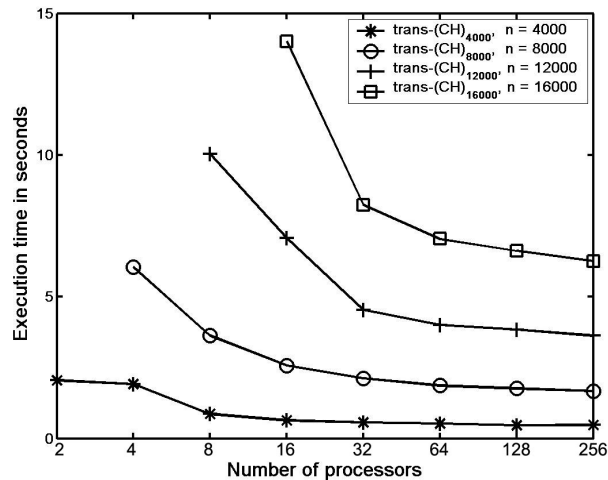


Figure 15. Execution times of PDSBTRI (without TBR) using matrices from trans-(PA).

4.2 Performance test of parallel OBR implementation

Random matrices generated by C build-in random number generator are used to test performance of the parallel OBR subroutine PDSBTRD (URL: <http://www.cs.utk.edu/~cape/parallel/PBTrd.tar.gz>). Figure 16 shows the execution times of PDSBTRD scaled by that of the ScaLAPACK subroutine PDSYTRD. The block size n_b for 2D block-cyclic matrix distribution is 32. It should be noted that the floating-point operation count for the two subroutines are not exactly the same since PDSBTRD only reduces a matrix to block-tridiagonal form while PDSYTRD reduces a matrix to tridiagonal form. The improved performance as a result of using level 3 BLAS operations can be seen from the relative execution time of PDSBTRD to PDSYTRD. In particular, PDSBTRD performs better when the problem size per processor n^2/p becomes larger.

The off-diagonal blocks of the block-tridiagonal matrix after parallel OBR tend to have full rank. As a pre-processing step for the BD&C algorithm, the ranks of the off-diagonal blocks directly affect the computational complexity of BD&C. Off-diagonal blocks with smaller sizes are preferred for lower computational complexity in BD&C. On the other hand, high performance of level 3 BLAS operations cannot be completely achieved when block sizes are too small. Figure

17 shows that PDSBTRD scales up with the number of processors in use. With block size 32, PDSBTRD achieves optimal performance. When block size is reduced to 16, there is a slight loss in performance; however, a significant reduction in the computational complexity of BD&C (by half) should compensate for the slower performance of the parallel OBR step. If we further reduce the block size to 8, the performance of PDSBTRD can be more than 50% slower than that with block size 32. In addition, with such a small block size, the performance of back transformation of the eigenvectors is going to be degraded drastically. Therefore, when combined with the BD&C eigensolver, we should choose block size 16 for the orthogonal block reduction subroutine PDSBTRD.

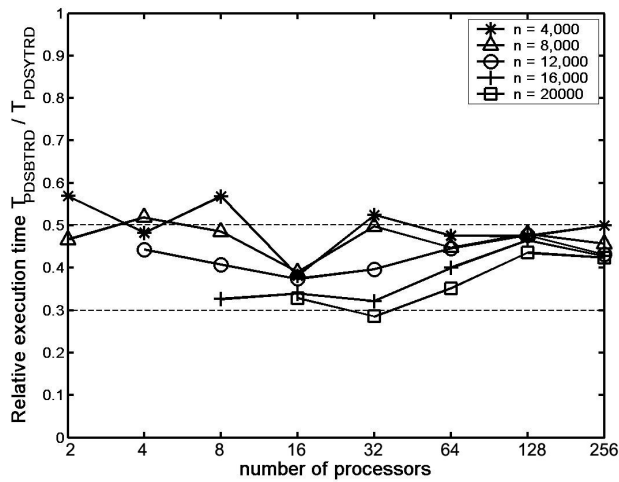


Figure 16. Relative execution times of PDSBTRD to PDSYTRD.

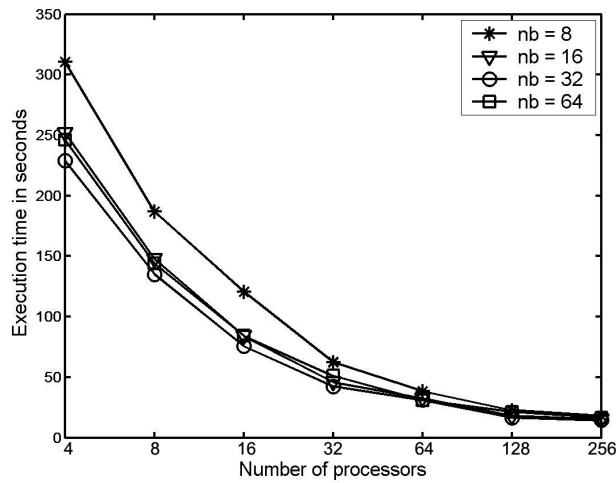


Figure 17. Execution time of PDSBTRD with different block sizes.

5. Conclusion

Two parallel algorithms are presented for block-tridiagonalization of symmetric dense matrices. They can be used according to the structure of input matrices and transform the original dense matrix into block-tridiagonal form.

When the input matrix is “effectively” sparse with strong locality properties, the parallel BTS algorithm is used to construct a block-tridiagonal matrix that approximates the original input matrix. In the context of approximate eigensolutions, given an accuracy tolerance τ and input matrix A , the eigenvalue errors between the resultant block-tridiagonal matrix and the original matrix is bounded by $\tau\|A\|$. In an iterative procedure like the self-consistent field (SCF) method, the eigenvector matrix from previous iteration can be used for the optional TBR step in an attempt to further reduce the block size of a few off-diagonal blocks. The computational complexity of parallel BTS is typically trivial compared to the complexity of computing all eigenpairs of a dense symmetric matrix.

When the input matrix is dense without usable structure, the parallel OBR algorithm is used to reduce the original matrix into block-tridiagonal form using blocked orthogonal transformations. The parallel OBR algorithm is very efficient by itself due to its high ratio of Level 3 BLAS operations; however, the off-diagonal blocks after orthogonal block-tridiagonal reduction tend to have full ranks. Since the block size of the block tridiagonal matrix equals the block size of the parallel 2D matrix distribution, one may try to reduce the ranks of off-diagonal blocks using smaller block size for parallel matrix distribution. For our tests using a block size of 16, the effect of level 3 BLAS operations in parallel OBR is reduced slightly, but the reduction of complexity in BD&C eigensolver is going to be great enough to compensate the small amount of performance loss in the orthogonal reduction step.

References

- Bai, Y. 2005. High performance parallel approximate eigensolver for real symmetric matrices. *PhD. dissertation, University of Tennessee, Knoxville.*
- Bai, Y., Gansterer, W. N. and Ward, R. C. 2004. Block-tridiagonalization of "effectively" sparse symmetric matrices. *ACM Trans. Math. Softw.* 30, 326 -- 352.
- Bai, Y. and Ward, R. C. 2005. A parallel symmetric block-tridiagonal divide-and-conquer algorithm. *Technical Report CS-05-571, Department of Computer Science, University of Tennessee, Knoxville, TN.*
- Bishof, C., Lang, B. and Sun, X. 1994. Parallel tridiagonalization through two-step band reduction. *Proceedings of Scalable High Performance Computing Conference '94. Knoxville, Tennessee, 23 -- 27.*
- Bishof, C., Lang, B. and Sun, X. 2000. A framework for symmetric band reduction. *ACM Trans. Math. Softw.* 26, 581 -- 601.
- Bishof, C., Lang, B. and Sun, X. 2000. The sbr toolbox -- software for successive band reduction. *ACM Trans. Math. Softw.* 26, 602-616.

Bishof, C., Marques, M. and Sun, X. 1993. Parallel bandreduction and tridiagonalization. *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM Press, Philadelphia, PA. 383 -- 390.

Bishof, C. and Van Loan, C. F. 1987. The wy representation for products of householder matrices. *SIAM J. Sci. Comput.* 8, 2 -- 13.

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D. and Whaley, R. C. 1997. Scalapack user's guide. *SIAM, Philadelphia, PA.*

Cuppen, J. J. M. 1981. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.* 36, 177 -- 195.

Demmel, J. 1997. *Applied numerical linear algebra*. SIAM, Philadelphia, PA.

Dhillon, I. S. and Parlett, B. N. 2004. Multiple representation to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra and Appl.* 387, 1 -- 28.

Dhillon, I. S., Parlett, B. N. and Vömel, C. 2004. Lapack working note: The design and implementation of the mrrr algorithm. *Technical Report UCB/CSD-04-1346, Computer Science Division, University of California at Berkeley, Berkeley, CA.*

Dongarra, J., Hammarling, S. and Sorensen, D. 1989. Block reduction of matrices to condensed forms for eigenvalue computation. *J. Comp. Appl. Math.* 27, 215 -- 227.

Dongarra, J. and Sorensen, D. 1987. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Stat. Comput.* 8, 139 -- 154.

Francis, G. J. F. 1961. The qr transformation: A unitary analogue to the lr transformation, parts i and ii. *Comput. J.* 4, 265 -- 271, 332 -- 345.

Gansterer, W. N., Kvasnicka, D. F. and Ueberhuber, C. W. 1998. Multi-sweep algorithms for the symmetric eigenproblem. *Lecture Notes in Computer Science. Vol. 1573*, 20 -- 28.

Gansterer, W. N., Ward, R. C., Muller, R. P. and Goddard, W. A., III. 2003. Computing approximate eigenpairs of symmetric block tridiagonal matrices. *SIAM J. Sci. Comput.* 25, 65 -- 85.

Gibbs, N. E., Poole, W. G. J. and Stockmeyer, P. K. 1976. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM J. Numer. Anal.* 13, 236 -- 250.

Golub, G. H. and van Loan, C. F. 1996. *Matrix computations*. John Hopkins University Press, Baltimore /London,

Gu, M. and Eisenstat, S. C. 1994. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM J. Matrix Anal. Appl.* 15(4), 1266 -- 1276.

Gu, M. and Eisenstat, S. C. 1995. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.* 16, 172 -- 191.

Heath, M. T., NG, E. and Peyton, B. W. 1990. Parallel algorithms for sparse linear systems. *Parallel Algorithms for Matrix Computations*, SIAM Press, Philadelphia, PA. 83 -- 124.

Karypis, G. and Kumar, V. 1998. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *J. Parallel Distrib. Comput.* 48, 71 -- 95.

Lewis, J. G. 1982. The gibbs-poole-stockmeyer and gibbs-king algorithms for reordering sparse matrices. *ACM Trans. Math. Softw.* 8, 190 -- 194.

Pople, J. A. and Beveridge, D. L. 1970. *Approximate molecular orbital theory*. 1st ed., McGraw-Hill, New York.

Pople, J. A., Beveridge, D. L. and Dobosh, P. A. 1967. Approximate self-consistent molecular orbital theory. V. Intermediate neglect of differential overlap. *J. Chem. Physics.* 47, 2026.

Pople, J. A., Santry, D. P. and Segal, G. A. 1965. Approximate self-consistent molecular orbital theory. I. Invariant procedures. *J. Chem. Physics.* 43, 129.

Pople, J. A. and Segal, G. A. 1965. Approximate self-consistent molecular orbital theory. Ii. Calculations with complete neglect of differential overlap. 43, 136.

Pople, J. A. and Segal, G. A. 1966. Approximate self-consistent molecular orbital theory. Iii. Cndo results for ab2 and ab3 systems. *J. Chem. Physics.* 44, 3289.

Su, W. P., Schrieffer, J. R. and Heeger, A. J. 1980. Soliton excitations in polyacetylene. *Phys. Rev. B.* 22, 2099 -- 2111.

- Szabo, A. and Ostlund, N. S. 1996. *Modern quantum chemistry*. Dover Publications, Mineola, NY.
- Wilkinson, J. H. 1965. *The algebraic eigenvalue problem*. Oxford University Press, Oxford.
- Wilkinson, J. H. 1968. Global convergence of the tridiagonal qr algorithm with origin shifts. *Linear Algebra and Appl.* 1, 409 -- 420.
- Wu, Y. J., Alpatov, P. A., Bishof, C. H. and van de Geijn, R. A. 1996. A parallel implementation of symmetric band reduction using lapack. *Proceedings of Scalable Parallel Library Conference, Starkville, Mississippi*.