A Parallel Symmetric Block-Tridiagonal Divide-and-Conquer Algorithm

Yihua Bai[*§] and Robert C. Ward[*§]

Technical Report UT-CS-05-571[1]
University of Tennessee
December 2005

Abstract

We present a parallel implementation of the block-tridiagonal divide-and-conquer algorithm that computes eigen-solutions of symmetric block-tridiagonal matrices to reduced accuracy. In our implementation, we use mixed data/task parallelism to achieve data distribution and workload balance. Numerical tests show that our implementation is efficient, scalable and computes eigenpairs to prescribed accuracy. We compare the performance of our parallel eigensolver with that of the ScaLAPACK divide-and-conquer eigensolver on block-tridiagonal matrices.

1. Introduction

Real symmetric eigenvalue problems frequently arise from scientific and engineering computations. The structures of the matrices and the requirements for the eigen-solutions vary by application. For instance, many matrices generated from electronic structure calculations in quantum mechanics have strong locality properties. Some of them are block-tridiagonal; some are dense but with larger elements close to the diagonals and a decrease in the magnitudes of elements as they move away from the diagonal. In many situations, the second type of matrices can be approximated by block-tridiagonal matrices with very low computational cost [2].

One of the most popular methods in electronic structure calculations is the Hartree-Fock method [3, 22]. The Hartree-Fock method relies on solving a non-linear generalized symmetric eigenvalue problem. Typically, the non-linear eigenvalue problem is solved iteratively by the self-consistent field (SCF) method. Lower accuracy is usually sufficient in earlier iterations of the SCF with higher accuracy required for the last several iterations as the SCF nears convergence [22]. The SCF method is also widely used in other areas of electronic structure computation, such as density functional theory [16, 18] and configuration interaction [20, 22].

Traditional symmetric dense eigensolvers compute eigen-solutions to full accuracy, which in many situations may exceed users' requirements. These eigensolvers decompose a real symmetric matrix in three steps [13, 17]: 1) Reduce the original matrix into a symmetric tridiagonal form using orthogonal transformations; 2) Compute eigenpairs of the tridiagonal matrix; 3) Back transform eigenvectors of the tridiagonal matrix to the original matrix. Due to the computational complexity and to data access patterns for most large dense matrices, step 1 is considerably more time consuming than the other two steps combined [24].

The block-tridiagonal divide-and-conquer algorithm developed by Gansterer and Ward et al. [10, 11] provides an eigensolver that addresses the above issues. Their algorithm computes all the eigenvalues and eigenvectors of a block-tridiagonal matrix to reduced accuracy with a similar

[*]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.

[1]Available from: http://www.cs.utk.edu/~library/2005.html

reduction in execution time for most applications. In addition, their algorithm does not require the tridiagonalization step, which greatly reduces the problems associated with data access and locality.

One of the challenges in the state-of-art simulation of atoms and molecules in the framework of the SCF method is that the matrix sizes are usually very large and exceed the limitation of single processor architecture. Thus, parallel computation becomes necessary. The block-tridiagonal divide-and-conquer algorithm is considered inherently parallel because the initial problem can be divided into smaller sub-problems and solved independently. We are thus motivated to implement an efficient, scalable parallel block-tridiagonal divide-and-conquer eigensolver with the ability to compute eigen-solutions to a user-specified accuracy. In particular, the parallel eigensolver becomes more efficient as accuracy requirement becomes lower.

In the following we briefly describe in Section 2 the sequential block-tridiagonal divide-and-conquer algorithm. In Section 3, we first list major issues in its parallel implementation and then present the details of our algorithm. Numerical tests are presented in Section 4 with conclusions given in Section 5.

2. Sequential block-tridiagonal divide-and-conquer (BD&C) algorithm

Given a symmetric block-tridiagonal matrix

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

and tolerance τ ($\varepsilon_{mach} \le \tau < 0.1$), where q is the number of diagonal blocks, B_i for $1 \le i \le q$ are the diagonal blocks and C_i for $1 \le i \le q-1$ are the off-diagonal blocks, the BD&C algorithm computes approximate eigenpairs of M to the prescribed accuracy tolerance τ. That is, we compute $\hat{V}$ and $\hat{\Lambda}$ such that

$$M \approx \hat{V}\hat{\Lambda}\hat{V}^T$$

where $\hat{V}$ contains the approximate eigenvectors, the diagonal matrix $\hat{\Lambda}$ contains the approximate eigenvalues, $\hat{V}$ and $\hat{\Lambda}$ satisfy

$$\left\| M - \hat{V}\hat{\Lambda}\hat{V}^T \right\|_2 = O\left(\tau \left\| M \right\|_2\right),$$

and V is numerically orthogonal, i.e.,

$$\max_{i=1,2,\cdots,n} \left\| \left(\hat{V}\hat{V}^T - I\right) e_i \right\|_2 = O\left(\varepsilon_{mach} n\right)$$

for $1 \le i \le n$ where e_i is the *i*-th column of the identity matrix.

There are three major steps in the BD&C algorithm [11]: problem subdivision, sub-problem solution and synthesis of sub-solutions.

2.1. Subdivision

The off-diagonal blocks C_i of sizes $b_{i+1} \times b_i$ are approximated by lower rank matrices using their singular value decompositions:

$$C_i \approx \sum_{j=1}^{\rho_i} \sigma_j^i u_j^i v_j^{iT} = U_i \Sigma_i V_i^T ,$$

where ρ_i is the chosen approximate rank of C_i based on the accuracy requirement, $U_i \in \mathbb{R}^{b_{i+1} \times \rho_i}$ is the orthogonal matrix containing the first ρ_i left singular vectors, $V_i \in \mathbb{R}^{b_i \times \rho_i}$ contains the first ρ_i right singular vectors, Σ_i is the diagonal matrix that contains the largest ρ_i singular values of C_i, and $i = 1, 2, \cdots, q-1$.

The approximate ranks of the off-diagonal blocks in the subdivision step of BD&C typically become smaller as the accuracy requirement becomes lower, which reduces the computational complexity (see Section 3.5 for computational complexity).

Using the above factorizations, the block tridiagonal matrix M can now be represented as an updated block diagonal matrix as follows:

$$M = \tilde{M} + \sum_{i=1}^{q-1} W_i W_i^T , \tag{2.1}$$

where $\tilde{M} = diag\left\{\tilde{B}_1, \quad \tilde{B}_2, \quad \cdots, \quad \tilde{B}_q\right\}$,

$$\tilde{B}_1 = B_1 - V_1 \Sigma_1 V_1^T ,$$

$$\tilde{B}_i = B_i - U_{i-1} \Sigma_{i-1} U_{i-1}^T - V_i \Sigma_i V_i^T , \quad \text{for} \quad 2 \le i \le q-1,$$

$$\tilde{B}_q = B_q - U_{q-1} \Sigma_{q-1} U_{q-1}^T ,$$

$$W_1 = \begin{pmatrix} V_1 \Sigma_1^{1/2} \\ U_1 \Sigma_1^{1/2} \\ 0 \\ 0 \end{pmatrix}, \quad W_i = \begin{pmatrix} 0 \\ V_i \Sigma_i^{1/2} \\ U_i \Sigma_i^{1/2} \\ 0 \end{pmatrix} \text{ for } 2 \le i \le q-2 \text{, and } W_{q-1} = \begin{pmatrix} 0 \\ 0 \\ V_{q-1} \Sigma_{q-1}^{1/2} \\ U_{q-1} \Sigma_{q-1}^{1/2} \end{pmatrix}.$$

2.2. Sub-problems solutions

Each diagonal block $\tilde{B}_i$ is factorized:

$$\tilde{B}_i = Z_i D_i Z_i^T , \quad \text{for} \quad i = 1, 2, \cdots, q , \tag{2.2}$$

from which we obtain

$$\tilde{M} = Z D Z^T , \tag{2.3}$$

where

$Z = diag\{Z_1, Z_2, \cdots, Z_q\}$ is a block diagonal orthogonal matrix, and $D = diag\{D_1, D_2, \cdots, D_q\}$ is a diagonal matrix.

Note that traditional algorithms may be applied to compute the eigen-decomposition of the diagonal blocks. Typically, the number of diagonal blocks q in a block tridiagonal matrix is much greater than 2 and the block sizes b_i are small compared to the matrix size n. Thus, the eigen-decomposition of each sub-problem $\tilde{B}_i$ in Equation 2.2, which involves only the much smaller diagonal block, yields better data access time pattern than traditional decomposition methods on the much larger full matrix.

2.3. Synthesis of sub-solutions

From Equations 2.1 and 2.3 we have:

$$M = Z(D + \sum_{i=1}^{q-1} Y_i Y_i^T) Z^T, \tag{2.4}$$

where $Y_i = Z^T W_i$.

2.3.1. Merging operations

Denoting $S = D + \sum_{i=1}^{q-1} Y_i Y_i^T$ and $\rho = \sum_{i=1}^{q-1} \rho_i$ in the synthesis step, *S* is represented as a sequence of ρ rank-one modifications of *D*. The ρ_i rank-one modifications $D + y_j^i \left(y_j^i \right)^T$ for $j = 1, 2, \cdots, \rho_i$ corresponding to an off-diagonal block C_i are called one merging operation, where $\left\{ y_j^i \right\}$ are the vectors that determine Y_i . Thus, the algorithm performs a total of $q-1$ such merging operations. For each rank-one modification in the *i*-th merging operation, the modified matrix is first decomposed: $D + y_j^i \left(y_j^i \right)^T = V_j^i \Lambda_j^i \left(V_j^i \right)^T$, and the eigenvector matrix from this decomposition is then multiplied onto the accumulated eigenvector matrix starting with the block diagonal eigenvector matrix Z . The accumulation of an intermediate eigenvector matrix for each rank-one modification involves matrix-matrix multiplications.

2.3.2. Deflation

Deflation happens when there is either a zero (or sufficiently small) component in y_j^i or two equal (or close) elements in D [5, 8]. If the *k*-th component in y_j^i is zero, then the *k*-th diagonal d_k of D is an eigenvalue of $D + y_j^i \left(y_j^i \right)^T$ and the corresponding eigenvector is e_k , the *k*-th column of the identity matrix. If there are two equal elements on the diagonal of D, a Givens rotation is used to zero out one of the corresponding elements in y_j^i, and corresponding eigenpairs can be computed as the former case. When deflation occurs, no computation is required to compute and accumulate the corresponding eigenvector. For a rank-one modification of order *m*, the total number of deflated values δ divided by *m* is called the ratio of deflation for that rank-one modification.

The deflation criteria can be relaxed if the accuracy tolerance is greater than full accuracy [11]. Numerical experiments have shown that as accuracy decreases, the higher ratio of deflation and lower ranks for off-diagonal blocks lead to high efficiency of the BD&C algorithm [11].

2.3.3. Balanced merge versus unbalanced merge

A merging operation is a balanced one if the sizes b_1 and b_2 of the two blocks to be merged are approximately the same, i.e., $b_1 \approx b_2$. If $b_1 \gg b_2$ or $b_1 \ll b_2$, then the merging operation is an unbalanced one. It has been shown that the time complexity for the most unbalanced merging operation is less than that for the most balanced one but with a higher rank – even an increase in rank of only one [11]. Therefore, an off-diagonal block with low rank is preferred for the final merging operation, regardless of the relative sizes of the blocks being merged.

3. Parallel block-tridiagonal divide-and-conquer (PBD&C) implementation

Although the BD&C algorithm has the potential of parallelism in that it is divide-and-conquer and recursive in nature, due to the differences in the size of each sub-problem and the amount of work needed for solving each of those sub-problems, PBD&C algorithm may encounter workload imbalance and data storage imbalance.

Suppose we have a block tridiagonal matrix $M \in \mathbb{R}^{n\times n}$ with q diagonal blocks as shown in Figure 1 and p processors. The size of the *i*-th diagonal block B_i is b_i where $\sum_{i=1}^{q} b_i = n$. The *i*-th off-diagonal block C_i has size $b_{i+1} \times b_i$, and let its approximate rank be ρ_i. We design a mixed data/task parallel implementation to maintain both workload and data balance. Some major issues in such an implementation are: 1) assignment of processors to sub-problems and distribution of data; 2) overhead of data redistribution before each merging operation; 3) order of the merging sequence; and 4) handling of deflation. These issues are discussed in this section in the context of our PBD&C implementation, as well as its computational and communication complexities.

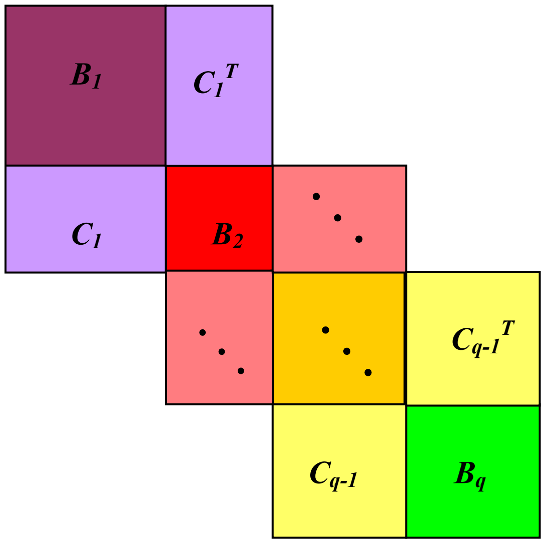

Figure 1. Block-tridiagonal matrix M with q diagonal blocks.

In analyses of computational and communication costs, γ denotes the time for one floating-point operation, α denotes the latency for one communication, β denotes the time to transfer one double precision number, and we define LCM as the least common multiple.

3.1. Processor allocation and data distribution

There are different ways to distribute a matrix on a processor grid. Data parallelism distributes data evenly to all the processors and invokes all relevant processors to work on the same task as the algorithm proceeds. Task parallelism assigns each processor to a different task in the algorithm working simultaneously whenever possible. While these two data distribution models have their advantages and disadvantages [4], neither of them is completely suitable for the distribution of a block-tridiagonal matrix for the PBD&C algorithm.

Our solution is to split the p processors available into q groups. The p_i processors in each group are configured into a 2D processor sub-grid $\mathcal{G}_i = r_i \times c_i$ where r_i and c_i are the number of rows and columns in the processor sub-grid and $1 \le i \le q$. The number of processors p_i in the i-th sub-grid $\mathcal{G}_i$ is determined by

$$p_i = \frac{b_i^3}{\sum_{i=1}^{q} b_i^3} p \tag{3.1}$$

since the computational complexity for solving each sub-problem is $O\left(b_i^3\right)$. Each processor sub-grid is assigned to a pair of diagonal and off-diagonal blocks as shown in Figure 2. On each sub-grid, matrix blocks are distributed in 2D block cyclic pattern as shown in Figure 3. We also use the p_i processors assigned to B_i for $1 \le i \le q-1$ to compute the approximate rank of the off-diagonal blocks C_i using the singular value decomposition $C_i = U_i \Sigma_i V_i^T$ as shown in Figures 1 and 2.

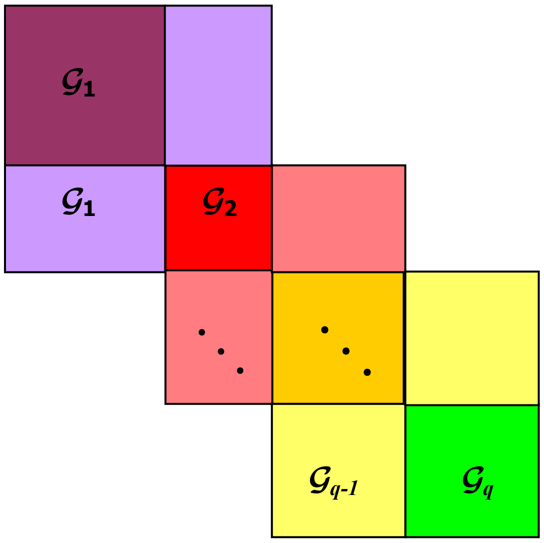

Figure 2. Each diagonal block B_i is assigned processor sub-grid $\mathcal{G}_i$.

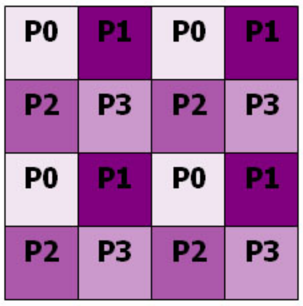

Figure 3. Data distribution of diagonal block B_1 on a 2×2 processor sub-grid $\mathcal{G}_1$.

3.2. Data redistribution

Due to the matrix distribution pattern described in Section 3.1, before the merging operation of two sub-solutions, corresponding sub-grids that hold the sub-matrices of eigenvectors must be combined and re-configured into a super-grid. Consequently, matrix sub-blocks must be redistributed from their original sub-grids to the super-grid. Figures 4 and 5 illustrate the redistribution of two sub-matrices, one from a 2×2 grid and the second from a 2×4 grid, to a 3×4 grid. This type of data redistribution is executed periodically, and the communication cost invoked is non-trivial as shown below.

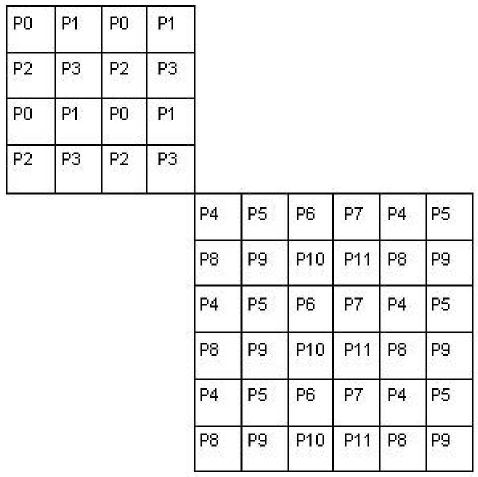

Figure 4. The first submatrix held by a 2×2 grid, the second submatrix held by a 2×4 grid.

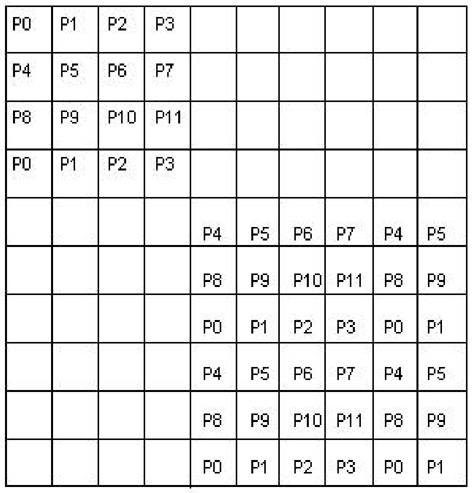

Figure 5. Two submatrices in Figure 4 redistributed to a 3×4 supergrid.

To assess the communication costs, assume k processors numbered from 0 to $k-1$ redistribute their data in a canonical order without pipelining, that is, processor 0 sends its data to processors $1, 2, \ldots, k-1$, then processor 1 sends its data to processors $0, 2, \ldots, k-1$, and so on until finally processor $k-1$ sends its data to processors $0, 1, \ldots, k-2$. Then, the total communication cost for the redistribution of two matrix sub-blocks from two sub-grids with p_i and p_{i+1} processors to its super-grid union with $p_{\cup}$ processors, where $p_{\cup} = p_i + p_{i+1}$ with $r_{\cup}$ rows and $c_{\cup}$ columns in its 2D processor grid, is [1]

$$\begin{aligned} t_{redistr} &= \alpha\left[LCM(r_i, r_{\cup})LCM(c_i, c_{\cup}) + LCM(r_{i+1}, r_{\cup})LCM(c_{i+1}, c_{\cup})\right] \\ &\quad + \beta\left(b_i^2 + b_{i+1}^2\right). \end{aligned} \tag{3.2}$$

As Equation 3.2 shows, when the number of rows and columns of a super-grid and its corresponding sub-grids are mutually prime, there are $p_{\cup}^2$ communications to accomplish the data transfer, and the accumulative startup time for these communications is high. However, this is typically not the case for careful assignment of processors to sub-grids. In the best case, the frequency of communications can be reduced to $2p_{\cup}$ if the two sub-grids have the same number of processors. The PBD&C algorithm is designed for execution with an arbitrary number of processors. However, the communication overhead for data redistribution is close to minimal if each sub-problem is of similar size and assigned the same number of processors. Of course, the communication cost also depends on the computer and network specifications as well as the shapes of sub-grids and super-grid.

3.3. Merging sequence

The synthesis step of PBD&C may be represented as a binary tree of merging operations with the bottom of the tree labeled merge level 0 as illustrated in Figure 6. The leaves are the eigen-solutions of the modified sub-problems $\tilde{B}_1, \tilde{B}_2, \cdots, \tilde{B}_q$ with $\tilde{B}_i$ distributed on sub-grid $\mathcal{G}_i$ with p_i processors. Recall that the order of $\tilde{B}_i$ is given by b_i. Each pair of eigen-solutions on the same level are merged simultaneously. If the off-diagonal block for the final merging operation has index *f*, then the matrix sub-blocks indexed from 1 to *f* construct a left sub-tree, while the matrix sub-blocks indexed from *f*+1 to *q* construct a right sub-tree.

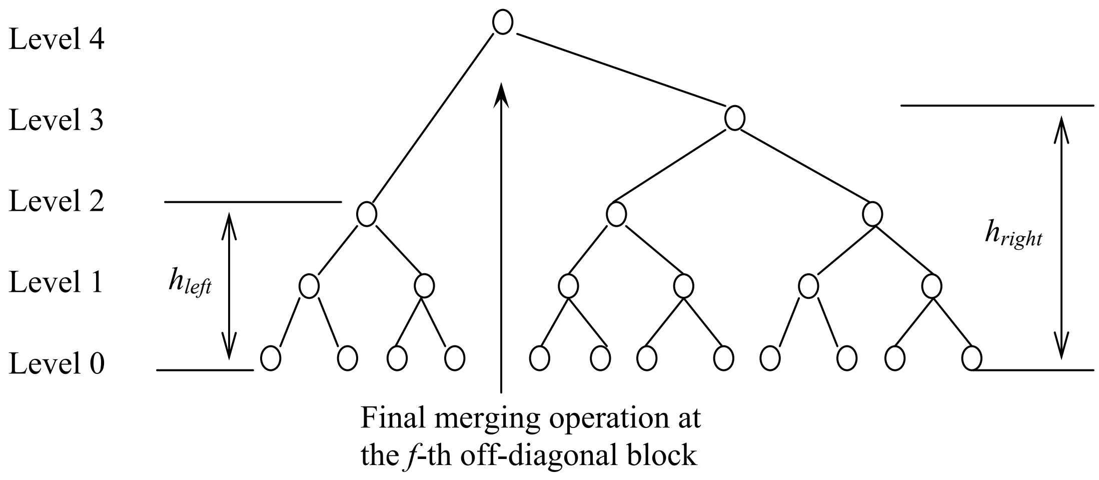

Figure 6. Merging tree and level of merging.

In the sequential implementation of BD&C, merging starts from off-diagonal blocks with the highest rank, leaving the off-diagonal block with the lowest rank for the final merging operation to reduce the computational complexity of BD&C. This merging sequence is sequential in nature. A parallel merging sequence in such an order may lose workload balance and lead to longer execution time [1]. The most time-consuming merging operation is the final one. It counts for up to 75% of the total computational time [1]. In order to choose an optimal final merging operation, we evaluate both the complexity of floating point operations and processor idle time for two different final merges: the off-diagonal block in the middle of the original matrix (a balanced merge) and the off-diagonal block with the lowest rank. We then choose the one that leads to the minimum total execution time.

To be specific, for a block-tridiagonal matrix *M*, we assume the off-diagonal block yielding a balanced final merging operation is located at position m, the rank of C_m is ρ_m, and h_m denotes the height of the balanced merging tree. Due to variation in the ranks for off-diagonal blocks and unknown deflation rate, a higher sub-tree may finish all its merging operations earlier than a lower sub-tree. Thus, we will denote by h'_m the height of the side of the merging tree, left or right, that takes longer to complete its merges. We use corresponding notation with subscript *l* for the final merge based upon the off-diagonal block with the lowest rank. An unbalanced final merging operation with the lowest rank has less total execution time per processor than a balanced final merging operation with higher rank only when [1]

$$\rho_m - \rho_l > \frac{p}{n^3}\left\{ \sum_{i=1}^{h'_l} \max_{j=1,\cdots,K_1^l} \left[\frac{\left(\sum_{k=0}^{2^i-1} b_{K_2^l+K_3^l+k} \right)^3 \rho_{K_4^l}}{\sum_{k=0}^{2^i-1} p_{K_2^l+K_3^l+k}} \right] - \sum_{i=1}^{h'_m} \max_{j=1,\cdots,K_1^m} \left[\frac{\left(\sum_{k=0}^{2^i-1} b_{K_2^m+K_3^m+k} \right)^3 \rho_{K_4^m}}{\sum_{k=0}^{2^i-1} p_{K_2^m+K_3^m+k}} \right] \right\} \quad (3.3)$$

where

$$K_1^l = \begin{cases} \left\lceil \frac{q-l}{2^i} \right\rceil - \operatorname{mod}\left(\left\lceil \frac{q-l}{2^{i-1}} \right\rceil, 2 \right) & \textit{if } h'_l = h_{right}^{low\_rank} \\ \left\lceil \frac{l}{2^i} \right\rceil - \operatorname{mod}\left(\left\lceil \frac{l}{2^{i-1}} \right\rceil, 2 \right) & \textit{if } h'_l = h_{left}^{low\_rank} \end{cases},$$

$$K_2^l = (j-1)2^i + 1 \ ,$$

$$K_3^l = \begin{cases} l & \textit{if } h'_l = h_{right}^{low\_rank} \\ 0 & \textit{if } h'_l = h_{left}^{low\_rank} \end{cases},$$

$$K_4^l = j2^i - 2^{i-1} + K_3^l,$$

$$K_1^m = \begin{cases} \left\lceil \frac{q-m}{2^i} \right\rceil - \operatorname{mod}\left(\left\lceil \frac{q-m}{2^{i-1}} \right\rceil, 2 \right) & \textit{if } h'_m = h_{right}^{balance} \\ \left\lceil \frac{m}{2^i} \right\rceil - \operatorname{mod}\left(\left\lceil \frac{m}{2^{i-1}} \right\rceil, 2 \right) & \textit{if } h'_m = h_{left}^{balance} \end{cases},$$

$$K_2^m = (j-1)2^i + 1 \ ,$$

$$K_3^m = \begin{cases} m & \textit{if } h'_m = h_{right}^{balance} \\ 0 & \textit{if } h'_m = h_{left}^{balance} \end{cases},$$

and $K_4^m = j2^i - 2^{i-1} + K_3^m$.

Therefore, we choose the unbalanced final merging operation with the lowest rank only when Equation 3.3 is satisfied. Otherwise, we choose the most balanced final merging operation.

3.4 Deflation

The efficiency of BD&C algorithm greatly relies on deflation. With lowered accuracy requirement, the frequency of deflation is very high and the amount of computation in the eigenvector accumulation is significantly reduced [10, 11].

When a deflation occurs during the decomposition of $D + yy^T = V\Lambda V^T$ (see Section 2.3.2), a permutation matrix P_{deflat} is used to move the deflated components of y to the bottom of y, and another permutation matrix P_{type} is used to re-group the columns in $\hat{Z} = ZG^T P_{deflat}^T$ according to their structures [14, 19, 23]. Thus,

$$M = ZG^T P_{deflat}^T P_{type}^T P_{type} P_{deflat} G(D + yy^T) G^T P_{deflat}^T P_{type}^T P_{type} P_{deflat} GZ^T ,$$

where Z starts as a block diagonal eigenvector matrix of sub-problems and G is an orthogonal matrix that accumulates all Givens rotations to deflate values in $D + yy^T$.

The structure of $\tilde{Z} = ZG^T P_{deflat}^T P_{type}^T$ for the first rank-one modification of a merging operation is shown in Figure 7 and for subsequent rank-one modifications in Figure 8.

In a sequential implementation, the cost of matrix permutation is trivial compared to the computational cost. In a parallel implementation, the cost of communication between processors cannot be neglected for frequent swaps of matrix columns. The strategy used in the ScaLAPACK subroutine PDSYEVD for symmetric tridiagonal eigenvalue problems is to permute columns of $\hat{Z}$ that reside local on each processor column into four groups shown in Figure 7, instead of a global permutation [23]. In that implementation, the deflation implemented is the minimum of the deflation on each processor. Good speedup is obtained even though the matrix multiplications performed are not of minimal size [23]. We adopt this strategy in our implementation of PBD&C with flexible accuracy tolerance. In our test cases of PBD&C using this strategy for matrix re-grouping, an average of only 5% less number of deflations than in the sequential BD&C algorithm is observed [1], which does not significantly degrade the performance of PBD&C.

3.5. Complexity of PBD&C

Assume that matrix $M \in \mathbb{R}^{n\times n}$ is a real symmetric block tridiagonal matrix with q diagonal blocks, n is divisible by q and each block has the same size $b = n/q$. To simplify the time complexity analysis, we further assume that each off-diagonal block has the same rank ρ and the ratio of deflation is 0%, that is, no deflation is found.

For the sequential BD&C algorithm, the dominant part of the computational time is the matrix multiplications to accumulate eigenvectors during the merging operations. The complexity of all other computations, such as solving secular equations and computing eigenvectors, is $O(n^2)$ or less for reasonable values of q. Therefore, the leading term in the computational complexity of BD&C is [1, 11]

$$t_{BD\&C} \approx \frac{8}{3}\rho n^3 \gamma \ .$$

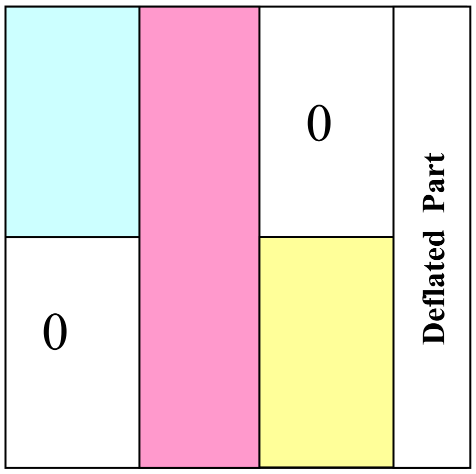

Figure 7. Structure of $\tilde{Z}$ for the first rank-one modification in a merging operation.

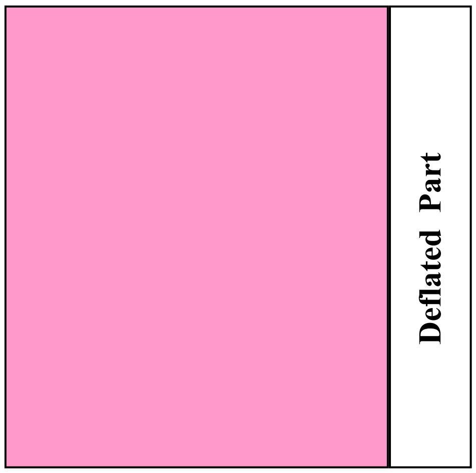

Figure 8. Structure of $\tilde{Z}$ for rank-one modifications after the first one in a merging operation.

For the parallel BD&C algorithm, the communication cost cannot be ignored. The leading terms in the computational and communication costs are [1]

$$t_{PBD\&C} \approx \frac{8n^3\rho}{3p}\gamma + \left(4\kappa + 4n\rho\right)\alpha + \left(n^2 + \frac{3.1n^2\rho}{\sqrt{p}}\right)\beta .$$

Here the term $4\kappa\alpha$ is the initialization costs for data re-distributions at the beginning of merging operations. In the best case where each sub-grid has the same number of processors, $\kappa = p$. In the worst case where the rows and columns of each pair of processor sub-grids and super-grid are mutually prime, $\kappa = p^2/2$.

4. Numerical experiments

Our numerical experiments include performance tests and accuracy tests. We compare PBD&C subroutine PDSBTDC with ScaLAPACK divide-and-conquer subroutine PDSYEVD for computing full eigensystems of symmetric block-tridiagonal matrices. All tests were run on the IBM p690 system nicknamed Cheetah at the Oak Ridge National Laboratory. System specifications and important system parameters are listed in Table 1 [9, 25]. The Fortran compiler on Cheetah is IBM's xlf version 8.1. Codes were compiled in the default 32-bit compile mode and linked to the 32-bit PESSL library [15], which includes the vendor optimized version of BLAS. The compiler options used are:

```
            -O4 -qarch=auto -qcache=auto –qtune=auto -bmaxdata:0x70000000.
```

Table 1. Cheetah system specifications and benchmarks.

Number of nodes		27
Memory per node		32 GB for most of the nodes
Processors per node		32
CPU frequency		1.3 GHz
L1 cache	Data	32 KB
	Instruction	64 KB
L2 cache		1.5 MB shared between 2 processors
L3 cache		32 MB off chip
Interconnect switch		Federation
Message passing latency		7 μs
Message passing bandwidth		1400 MBs
DGEMM GFLOPS per processor		3.174 GFLOPS

For a symmetric block-tridiagonal matrix $M = \hat{V}\hat{\Lambda}\hat{V}^T$, where $\hat{V}$ is the computed approximate eigenvector matrix and $\hat{\Lambda}$ is the diagonal matrix that contains the computed approximate eigenvalues, we use the scaled residual error

$$\mathcal{R} = \max_{i=1,\cdots,n} \frac{\left\| M\hat{v}_i - \hat{\lambda}_i \hat{v}_i \right\|_2}{\left\| M \right\|_2}$$

and the scaled departure from orthogonality

$$\mathcal{O} = \frac{\max\limits_{i=1,\cdots,n} \left\| \left(\hat{V}^T \hat{V} - I \right) e_i \right\|_2}{n}$$

to evaluate the accuracy of results.

For all the numerical tests, the number of processors used is a power of 2. We start from the smallest number of processors that provides sufficient memory to solve the problems and verify computational results, and then increment the number of processors by powers of 2 up to 512.

4.1. Test matrices

We use three types of matrices to test PDSBTDC: 1) block-tridiagonal matrices with specified eigenvalue distributions generated by LAPACK/ScaLAPACK test matrix generator [6]; 2) block-tridiagonal matrices with random number entries; 3) matrices generated from electronic structure calculations. Matrix sizes and block sizes are listed in Table 2. In the following, we briefly describe characteristics of all test matrices and their eigenvalue distributions.

Table 2. Test matrices

Matrix Name	Matrix Size	Block Size
M-rand	4000, 8000, 12000, 16000, 20000	20
M-geom		
M-arith		
M-ala	5027	104 for the first and last block, 79 for all other blocks
M-tPA	8000, 16000	500

M-geom: Matrix eigenvalues are distributed in a geometric sequence ranging from 1 to ε_{mach} with random signs attached to eigenvalues, $\lambda_i = \pm\left(\varepsilon_{mach}\right)^{i-1/n-1}$.

M-arith: Matrix eigenvalues distributed in an arithmetic sequence ranging from 1 to ε_{mach} with random signs attached to eigenvalues, $\lambda_i = \pm\left[1-\left(1-\varepsilon_{mach}\right)\left(i-1\right)/\left(n-1\right)\right]$.

M-rand: Matrix entries are random numbers uniformly distributed in the interval $\left(0,1\right)$.

M-ala: Matrices are generated from simulating polypeptide molecules made from alanine using the MNDO method [7]. Figures 9 and 10 show the magnitudes of the elements in the Fock matrix from a linear polyalanine chain of length 200 and its eigenvalue distribution.

M-tPA: Matrices are generated from simulating Trans-Polyacetylene (PA). Trans-PA consists of a chain of CH units. It has the general molecular formula trans-$(CH)_n$. The SSH Hamiltonian [21] is combined with the Hartree-Fock approximation to produce test matrices in this family. Figures 11 and 12 show the magnitudes of matrix elements of trans-$(CH)_{8000}$ and its eigenvalue distribution. Matrices used in our tests are generated from trans-$(CH)_{8000}$ and trans-$(CH)_{16000}$. Although the original matrices are dense, our blocking does not affect the eigenvalue error more than the accuracy tolerance in the tests since the matrix elements become very small away from the diagonals.

4.2. Test Results

First we set the accuracy tolerance to 10^{-6}. Figures 13 – 15 show the execution times of PDSBTDC scaled by that of PDSYEVD for different matrices. Figure 13 shows that PDSBTDC is very efficient on **M-geom** matrices. With the prescribed tolerance, all the block tridiagonal matrices in **M-geom** have rank of 0 for the off-diagonal block in the final merging operation, which decouples the problem into two smaller ones. In addition, matrices with clustered eigenvalues tend to have very high ratio of deflation [11]. Those two factors lead to the high efficiency of PDSBTDC.

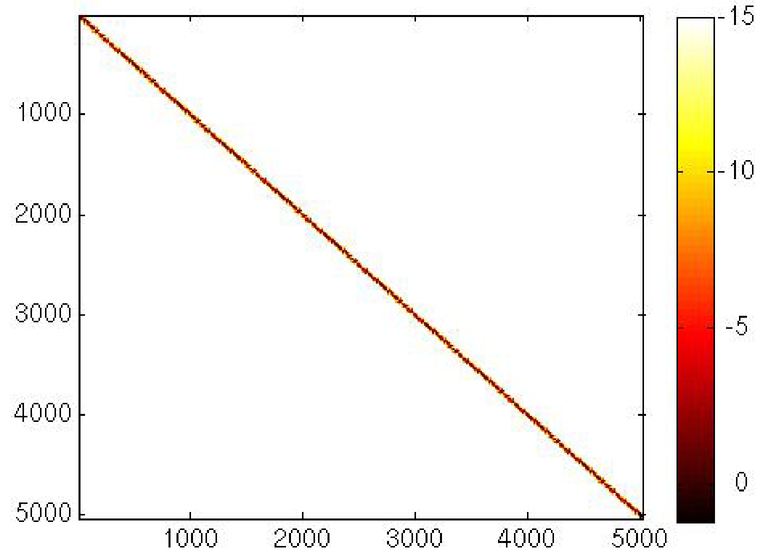

Figure 9. $\log_{10}$ **of absolute value of matrix elements for linear polyalanine chain of length 200,** $n = 5,027$ **.**

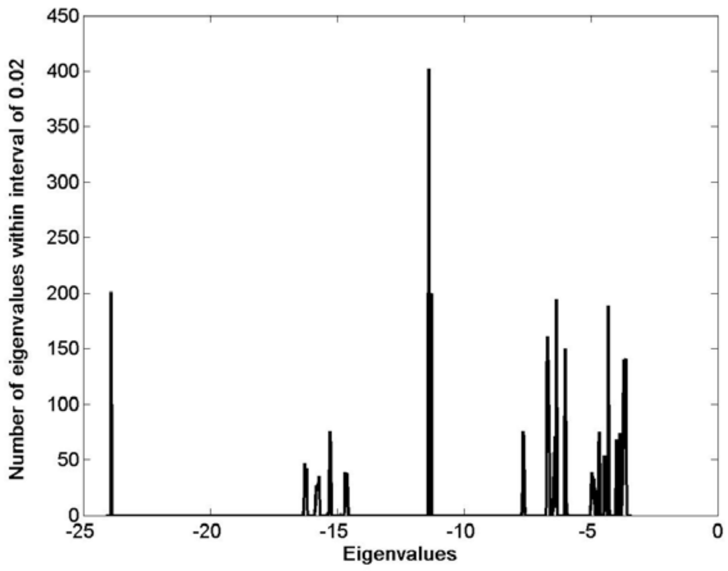

Figure 10. Eigenvalue distribution of matrix in Fig. 9.

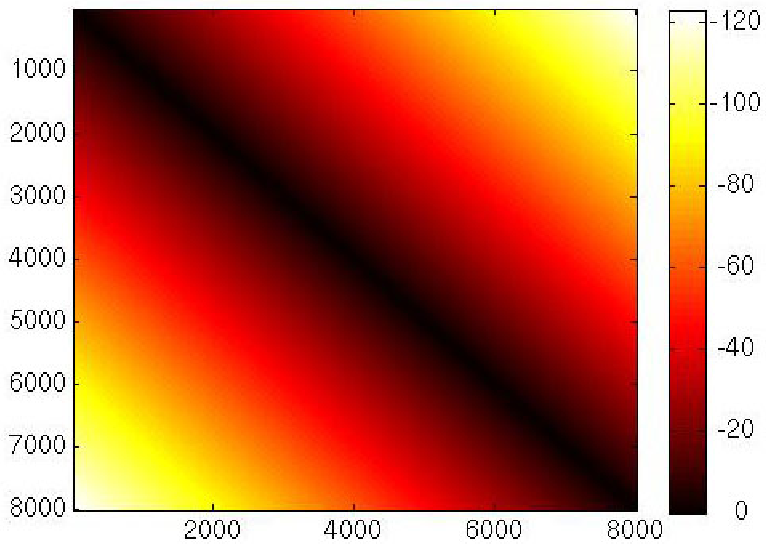

Figure 11. $\log_{10}$ **of absolute value of matrix elements for trans-PA molecule,** $n = 8,000$**.**

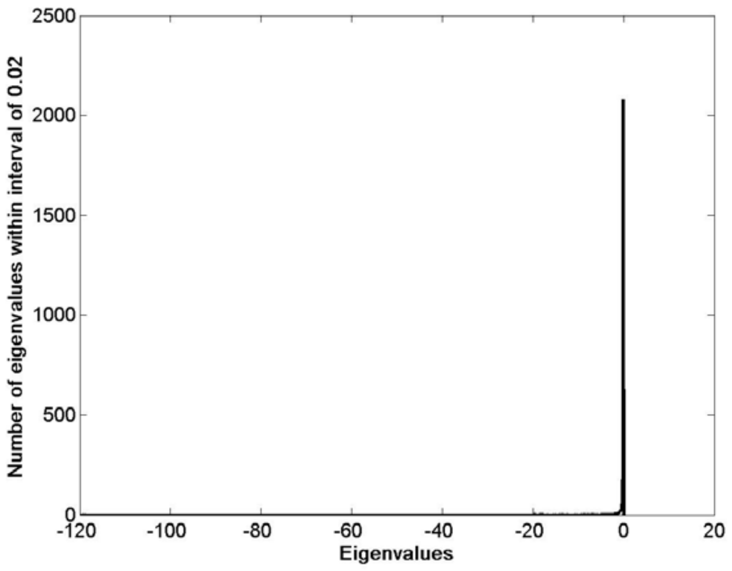

Figure 12. Eigenvalue distribution of matrix in Fig. 11.

Figure 14 shows the performance of PDSBTDC on **M-arith** matrices. Since the eigenvalues are evenly distributed, deflation rates are relatively low. In addition, the ranks of the off-diagonal blocks are much higher than those in the **M-geom** matrices. Although PDSBTDC still computes its eigensystem faster than PDSYEVD, those two factors contribute to a slower performance than observed on **M-geom**. The performance of PDSBTDC on matrices **M-rand** is better than that on **M-arith** but still slower than that on **M-geom** as Figure 15 shows.

In Figure 16 we display the relative execution time of PDSBTDC on application matrices **M-ala** and **M-tPA**. With a tolerance $\tau = 10^{-6}$, Figure 16 shows that PDSBTDC is very efficient for both matrices **M-ala** and **M-tPA**. The good performance is due to a very high ratio of deflation for **M-ala** and very low ranks for the off-diagonal blocks in **M-tPA**.

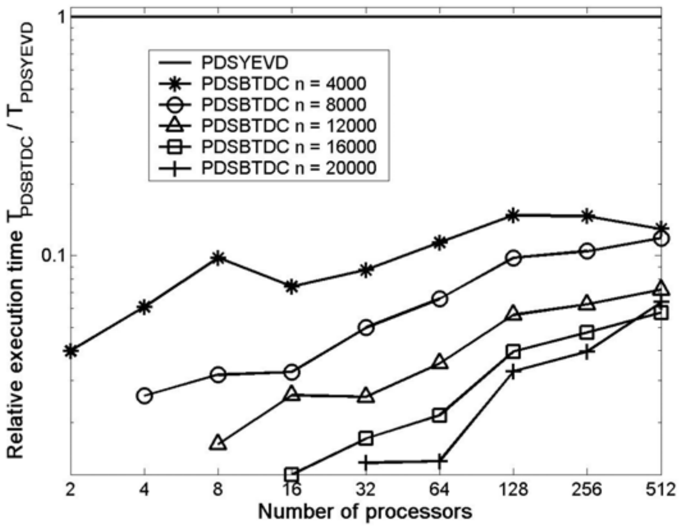

Figure 13. Execution time of PDSBTDC relative to PDSYEVD in log scale for M-geom matrices.

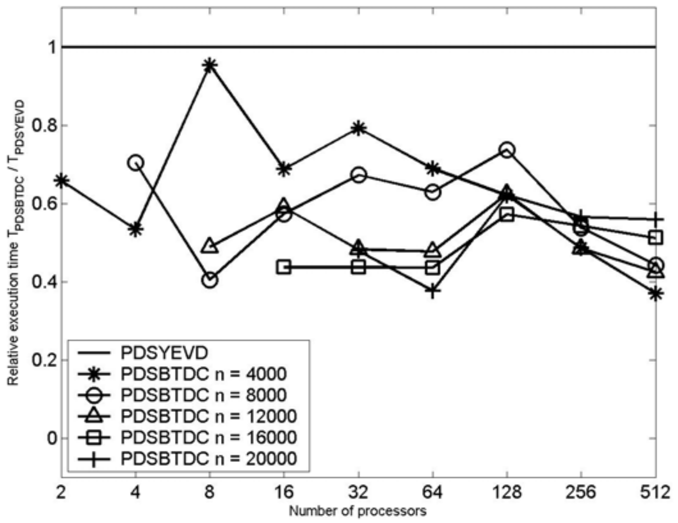

Figure 14. Execution time of PDSBTDC relative to PDSYEVD for M-arith matrices.

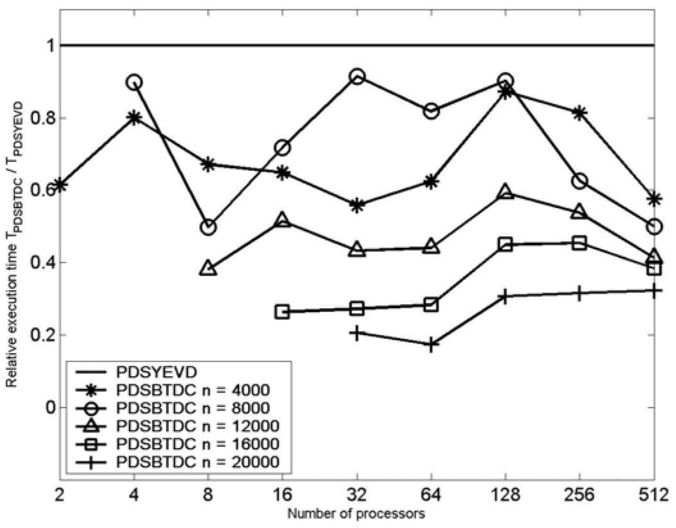

Figure 15. Execution time of PDSBTDC relative to PDSYEVD for M-rand matrices.

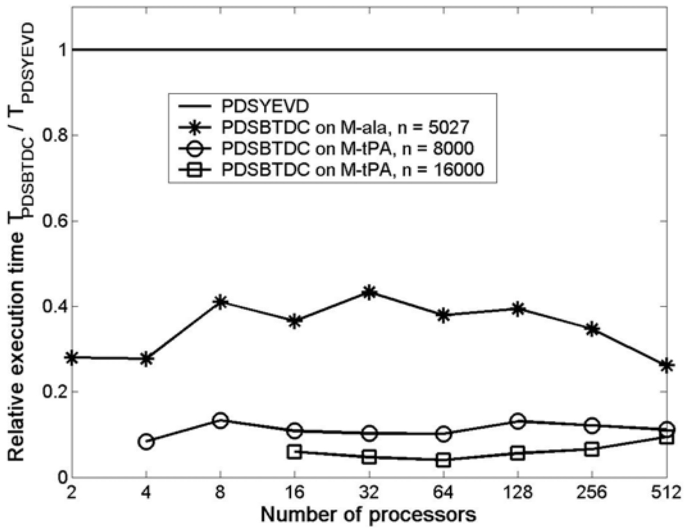

Figure 16. Execution time of PDSBTDC relative to PDSYEVD for application matrices M-ala and M-tPA.

Maximum residual $\mathcal{R}$ and orthogonality errors $\mathcal{O}$ for all test matrices are plotted in Figure 17. All maximum residuals are of order 10^{-6}, and all orthogonality errors are of order 10^{-18}.

To test performance of PDSBTDC with different accuracy requirements, we use matrix **M-arith** with size 12,000 and **M-ala** and set the accuracy tolerance to different values: 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10} and 10^{-12}. Figures 18 and 19 reveal two pieces of important information. Firstly, as the tolerance decreases, execution time increases due to less deflation and higher ranks for the off-diagonal blocks. When the tolerance reaches 10^{-10} and less, PDSBTDC is no longer

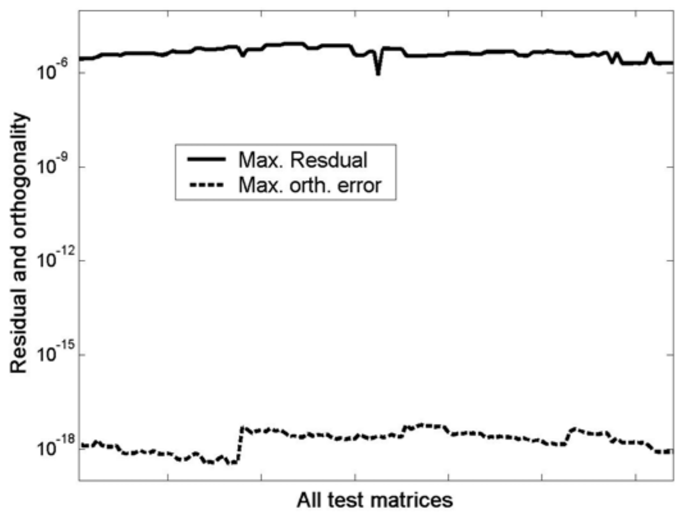

Figure 17. Residual $\mathcal{R} = \max_{i=1,\cdots,n} \frac{\left\| M\hat{v}_i - \hat{\lambda}_i \hat{v}_i \right\|_2}{\left\| M \right\|_2}$ **and departure from orthogonality** $\mathcal{O} = \frac{\max_{i=1,\cdots,n} \left\| \left(\hat{V}^T \hat{V} - I \right) e_i \right\|_2}{n}$ **for all test matrices,** $\tau = 10^{-6}$

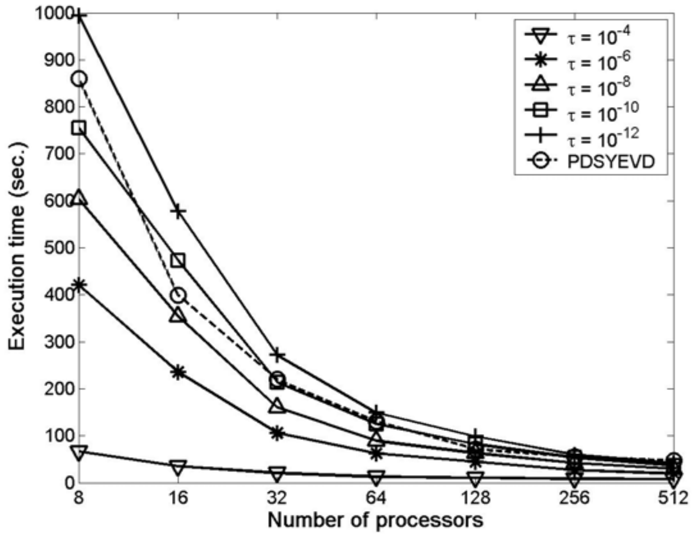

Figure 18. Execution times of PDSBTDC and PDSYEVD on matrix M-arith, $n = 12,000$ **.**

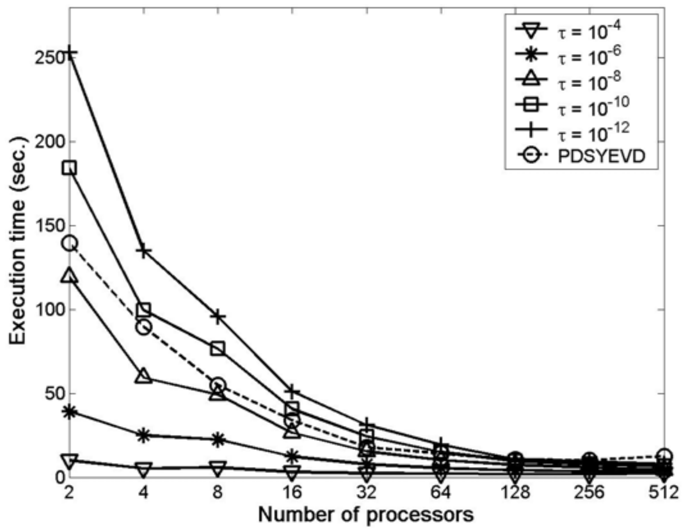

Figure 19. Execution times of PDSBTDC and PDSYEVD on matrix M-ala, $n = 5,027$ **.**

competitive in comparison to PDSYEVD. Secondly, the execution time of PBD&C decreases as the number of processors increases. This illustrates the scalability of the PBD&C implementation.

5. Conclusion

A mixed data/task parallel implementation of the block-tridiagonal divide-and-conquer algorithm is presented. In this implementation, processors are divided into sub-grids and each sub-grid is assigned to a sub-problem. Because of the data distribution pattern, at the beginning of each merging operation, matrix sub-blocks must be re-distributed. The communication overhead of data re-distribution not only depends on interconnection of processors, but also on the topology of processor sub-grids. Both floating-point operation count and workload balance are evaluated for the final merging operation to determine a merging sequence with the shortest execution time. Deflations are handled as in the ScaLAPACK tridiagonal divide-and-conquer subroutine PDSYEVD to minimize communication cost.

Accuracy tests show that PDSBTDC computes eigen-solutions of block-tridiagonal matrices to required accuracy. Complexity analyses and performance tests show that for a low accuracy such as $\tau = 10^{-6}$, PDSBTDC is very efficient on block tridiagonal matrices with either relatively low ranks for off-diagonal blocks or high ratio of deflation during the merging operations or both.

The mixed data/task parallel implementation of the BD&C algorithm has the potential of losing workload balance when the ratio of deflation for pairs of sub-problems on the same level of the merging tree vary drastically. Although the workload imbalance problem caused by unfavorable deflation distribution is not unique to our method [1, 12, 23], such cases are exceptionally rare in real applications.

Acknowledgments. We thank Dr. Richard Muller and Dr. Guoping Zhang for providing application matrices. We also thank Justin Pratt for helping with some of the tests, and Dr. Mark Fahey and staff of the National Center for Computational Sciences, ORNL for their assistance in running the test cases.

References

[1] Y. Bai, *High Performance Parallel Approximate Eigensolver for Real Symmetric Matrices*, PhD. dissertation, University of Tennessee, Knoxville, 2005.

[2] Y. Bai, W. N. Gansterer and R. C. Ward, *Block-Tridiagonalization of "Effectively" Sparse Symmetric Matrices*, ACM Trans. Math. Softw., 30 (2004), pp. 326 -- 352.

[3] J. Callaway, *Quantum Theory of the Solid State*, Academic Press, Boston, 1991.

[4] S. Chakrabarti, J. Demmel and D. Yelick, *Modeling the Benefits of Mixed Data and Task Parallelism*, Proceeding of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (1995), pp. 74 -- 83.

[5] J. J. M. Cuppen, *A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem*, Numer. Math., 36 (1981), pp. 177 -- 195.

[6] J. Demmel and A. McKenney, *A Test Matrix Generation Suite*, Courant Institute of Mathematical Sciences, New York (1989).

[7] M. J. S. Dewar and W. Thiel, *Ground States of Molecules, 38. The MNDO Method. Approximations and Parameters*, J. Amer. Chem. Soc., 99 (1977), pp. 4899-4907.

[8] J. Dongarra and D. Sorensen, *A Fully Parallel Algorithm for the Symmetric Eigenvalue Problem*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 139 -- 154.

[9] T. H. Dunigan, Jr., *ORNL IBM Power4 (p690) Evaluation*, www.cs.ornl.gov/~dunigan/sp4.

[10] W. N. Gansterer, R. C. Ward and R. P. Muller, *An Extension of the Divide-and-Conquer Method for a Class of Symmetric Block-tridiagonal Eigenproblems*, ACM Trans. Math. Softw., 28 (2002), pp. 45 -- 58.

[11] W. N. Gansterer, R. C. Ward, R. P. Muller and W. A. Goddard, III, *Computing Approximate Eigenpairs of Symmetric Block Tridiagonal Matrices*, SIAM J. Sci. Comput., 25 (2003), pp. 65 -- 85.

[12] K. Gates and P. Arbenz, *Parallel Divide and Conquer Algorithms for the Symmetric Tridiagonal Eigenproblem*, Technical Report, Institute for Scientific Computing, ETH Zurich, 1994.

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

[14] M. Gu, *Studies in Numerical Linear Algebra*, PhD. Thesis, Yale University, New Haven, Connecticut, 1993.

[15] IBM, *Parallel ESSL for AIX, Guide and Reference, V2.3*, 2003.

[16] W. Kohn and L. J. Sham, *Self-Consistent Equations Including Exchange and Correlation Effects*, Phys. Rev., 140 (1965), pp. A1133 -- 1138.

[17] B. N. Parlett, *The Symmetric Eigenvalue Problem*, SIAM Press, Philadelphia, PA, 1997.

[18] R. G. Parr and W. Yang, *Density-Functional Theory of Atoms and Molecules*, Oxford University Press, 1994.

[19] J. Rutter, *A Serial Implementation of Cuppen's Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem*, Technical Report CS-94-225, Department of Computer Science, University of Tennessee, Knoxville, TN (1994).

[20] I. Shavitt, *The method of configuration interaction*, Methods of Electronic Structure Theory, Plenum Press, New York (1977), pp. 189-275.

[21] W. P. Su, J. R. Schrieffer and A. J. Heeger, *Soliton Excitations in Polyacetylene*, Phys. Rev. B, 22 (1980), pp. 2099 -- 2111.

[22] A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry*, Dover Publications, Mineola, NY, 1996.

[23] F. Tisseur and J. Dongarra, *Parallelizing the Divide and Conquer Algorithm for the Symmetric Tridiagonal Eigenvalue Problem on Distributed Memory Architectures*, SIAM J. Sci. Comput., 20 (1999), pp. 2223 -- 2236.

[24] R. C. Ward, Y. Bai and J. Pratt, *Performance of Parallel Eigensolvers on Electronic Structure Calculations*, Technical Report, UT-CS-05-560, University of Tennessee, Knoxville, TN, 2005.

[25] P. H. Worley, T. H. Dunigan, Jr., M. R. Fahey, J. B. White, III and A. S. Bland, *Early Evaluation of the IBM p690*, Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, Baltimore, Maryland (2002), pp. 1 -- 21.