Combinatorial preconditioning for accelerating the convergence of the parallel block Jacobi method for the symmetric eigenvalue problem

Masaki Kugaya¹, Shuhei Kudo¹ and Yusaku Yamamoto¹

¹The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo, 182-8585, Japan

*Corresponding author: yusaku.yamamoto@uec.ac.jp

Received July 29, 2021, Accepted August 30, 2021

Abstract

In this paper, we propose combinatorial preconditioning to accelerate the convergence of the parallel block Jacobi method for the symmetric eigenvalue problem. The idea is to gather matrix elements of large modulus near the diagonal prior to each annihilation by permutation of rows and columns and annihilate them at once, thereby leading to large reduction of the off-diagonal norm. Numerical experiments show that the resulting method can actually speedup the convergence and reduce the execution time of the parallel block Jacobi method.

Keywords symmetric eigenvalue problem, block Jacobi method, preconditioning, combinatorial optimization, parallel computing

1. Introduction

Let \( A = (a_{ij}) \in \mathbb{R}^{n \times n} \) be a symmetric matrix and consider computing its eigendecomposition \( A = VDV^\top \), where \( V \in \mathbb{R}^{n \times n} \) is an orthogonal matrix and \( D \in \mathbb{R}^{n \times n} \) is a diagonal matrix. The standard procedure for this problem is the tridiagonalization-based method, which consists of tridiagonalization of \( A \), eigendecomposition of the resulting tridiagonal matrix and back-transformation of the eigenvectors. In the second step, various algorithms such as the QR algorithm, divide-and-conquer algorithm, or the MR² algorithm, can be used. This type of methods are widely used due to its computational efficiency and are implemented in standard libraries like LAPACK and ScaLAPACK.

In recent years, there has been a revived interest in the Jacobi and block Jacobi methods as an alternative to the tridiagonalization-based methods. The reason is twofold. First, these methods can compute the smallest eigenvalues of a certain class of matrices to high relative accuracy [1]. This feature is not shared by the tridiagonalization-based methods and is useful in applications such as vibration analysis and quantum chemistry [2]. Second, when implemented on parallel computers, the block Jacobi method requires fewer synchronization points than tridiagonalization-based methods. Since inter-processor synchronization causes large overhead on modern parallel machines, this is a great advantage of the block Jacobi method. In fact, performance evaluation on the K computer shows that the block Jacobi method outperforms ScaLAPACK when the number of nodes exceeds 100 [3]. Similar results are reported also for the block Jacobi method for the singular value decomposition [4]. However, the block Jacobi method is generally slower on fewer nodes, where the influence of synchronization overhead is less significant, because its computational work is several times larger than that of the tridiagonalization-based methods.

In this paper, we propose to accelerate the convergence of the block Jacobi method by introducing combinatorial preconditioning. We focus on the parallel block Jacobi method with dynamic ordering [5, 6]. In this variant, the matrix is partitioned into square blocks of appropriate size. At each step, the off-diagonal blocks with the largest Frobenius norm are chosen under the constraint that they can be annihilated simultaneously, and they are annihilated in parallel. While this strategy generally works well, its efficiency degrades when most of the off-diagonal blocks have comparable Frobenius norms. To resolve the problem, we propose to permute the rows and columns of the matrix prior to each annihilation and generate off-diagonal blocks with a larger Frobenius norm. This can be achieved by solving a combinatorial optimization problem. We call this combinatorial preconditioning. It is designed mainly for shared-memory parallel computers, on which permutation does not cause too much cost. In this paper, we formulate the idea of this preconditioning, give an approximate algorithm to solve the resulting combinatorial optimization problem and present some preliminary numerical results.

The rest of this paper is structured as follows. In Section 2, we present an outline of the parallel block Jacobi method with dynamic ordering, along with its possible difficulties. Section 3 introduces the combinatorial preconditioning and an approximate algorithm to solve the associated optimization problem. Numerical results that illustrate the effectiveness of combinatorial preconditioning are given in Section 4. Finally, Section 5 provides some discussion on the numerical results.
2. Parallel block Jacobi method with dynamic ordering

2.1 The algorithm

In the parallel block Jacobi method, we partition the matrix $A$ into a $q \times q$ block structure. We usually set $q = 2p$, where $p$ is the number of processors. Let us denote the matrix at the $k$th iteration step by $A^{(k)}$ and its $(I, J)$ block by $A^{(k)}_{IJ}$.

Now, we focus on the so-called parallel block Jacobi method with dynamic ordering, which has proved effective in terms of convergence [5, 6]. At the $k$th step of this method, we choose an off-diagonal block $A_{X_1,Y_1}^{(k)}$ (where $X_1 \neq Y_1$) with the largest Frobenius norm and construct an orthogonal transformation matrix that annihilates $A^{(k)}_{X_1,Y_1}$ and $A^{(k)}_{Y_1,X_1}$. To be concrete, let

$$A^{(k)}_{X_1,Y_1,X_1} = \left( \begin{array}{cc} A^{(k)}_{X_1,Y_1} & A^{(k)}_{X_1,X_1} \\ A^{(k)}_{Y_1,X_1} & A^{(k)}_{Y_1,Y_1} \end{array} \right)$$

and $P^{(k)}_{X_1,Y_1,X_1}$ be an orthogonal eigenvector matrix of $A^{(k)}_{X_1,Y_1,X_1}$. Then, the desired orthogonal matrix is obtained by embedding $P^{(k)}_{X_1,Y_1,X_1}$ into the $n \times n$ identity matrix $I_n$ in the same way as $A^{(k)}_{X_1,Y_1,X_1}$ is embedded in $A^{(k)}$. Next, we choose the off-diagonal block $A^{(k)}_{X_2,Y_2}$ ($X_2 \neq Y_2$) with the largest Frobenius norm under the condition that $X_2 \neq X_1, Y_1$ and $Y_2 \neq X_1, Y_1$ and construct $P^{(k)}_{X_2,Y_2}$ in the same way as we constructed $P^{(k)}_{X_1,Y_1}$.

We repeat this procedure: assuming that $X_r$ and $Y_r$ ($X_r \neq Y_r$) have been determined, we choose $X_{r-1}$ and $Y_{r-1}$ ($X_{r-1} \neq Y_{r-1}$) such that $A^{(k)}_{X_{r-1},Y_{r-1}}$ is the off-diagonal block with the largest Frobenius norm under the condition that $X_{r-1} \neq X_1, Y_1, \ldots, X_{r-2}, Y_{r-2}$ and $Y_{r-1} \neq X_1, Y_1, \ldots, X_{r-2}, Y_{r-2}$ and construct $P^{(k)}_{X_{r-1},Y_{r-1}}$. In this way, $2p$ off-diagonal blocks, $A^{(k)}_{X_1,Y_1}, A^{(k)}_{Y_1,X_1}, \ldots, A^{(k)}_{X_{p-1},Y_{p-1}}, A^{(k)}_{Y_{p-1},X_{p-1}}$, are selected. They can be annihilated in parallel using $p$ processors because the row blocks (and also column blocks) they belong to are disjoint. After annihilating all of them, we move to step $k+1$.

The algorithm of the parallel block Jacobi method with dynamic ordering is shown as Algorithm 1. Here, $w_{IJ}$ is the Frobenius norm of the $(I, J)$ block and $\text{MWM}(\{ w_{IJ} \})$ is a procedure to determine $(X_1, Y_1), \ldots, (X_p, Y_p)$ as described above. It is based on an algorithm to find the maximal weight matching (MWM) of a perfect graph. See [6] for details. In line 10, we apply $(P^{(k)}_{X_{\ell-1},Y_{\ell-1}})^{\top}$ to the $X_{\ell}$-th and $Y_{\ell}$-th row blocks for $\ell = 1, 2, \ldots, p$. In line 13, we apply $P^{(k)}_{X_{\ell+1},Y_{\ell+1}}$ to the $X_{\ell}$-th and $Y_{\ell}$-th column blocks for $\ell = 1, 2, \ldots, p$. This completes the orthogonal transformations and annihilates the $2p$ off-diagonal blocks. Also, in line 14, these orthogonal transformations are accumulated into an $n \times n$ orthogonal matrix $V^{(k)}$, which converges to an orthogonal eigenvector matrix of $A$ as $A^{(k)}$ converges to a diagonal matrix. The readers are referred to [6] for more detailed explanation of the algorithm.

**Algorithm 1** Parallel block Jacobi method with dynamic ordering

1: $A^{(0)} = A$
2: $V^{(0)} = I_n$
3: $k = 0$
4: while $w_{\max} > \epsilon$ do
5: Compute $w_{IJ} = \| A^{(k)}_{IJ} \|_F$ for $1 \leq I < J \leq q$.
6: $w_{\max} := \max_{1 \leq I < J \leq q} w_{IJ}$.
7: $(X_1, Y_1), \ldots, (X_p, Y_p) = \text{MWM}(\{ w_{IJ} \})$.
8: for $\ell = 1, \ldots, p$ do
9: Compute the eigenvector matrix $P^{(k)}_{X_{\ell-1},Y_{\ell-1}}$.
10: $A^{(k+1)}_{X_{\ell-1},Y_{\ell-1}} = (P^{(k)}_{X_{\ell-1},Y_{\ell-1}})^{\top} A^{(k)}_{X_{\ell-1},Y_{\ell-1}}$
11: end for
12: $A^{(k+1)}_{X_{\ell-1},Y_{\ell-1}} = A^{(k)}_{X_{\ell-1},Y_{\ell-1}} P^{(k)}_{X_{\ell-1},Y_{\ell-1}}$
13: $V^{(k+1)} = V^{(k)} P^{(k)}_{X_{\ell-1},Y_{\ell-1}}$
14: end for
15: $k := k + 1$
16: end while

2.2 The difficulty

It is expected that the parallel block Jacobi method with dynamic ordering can reduce the off-diagonal norm $\text{off}(A^{(k)}) = \sqrt{\sum_{i \neq j} a_{ij}^2}$ efficiently, because it chooses an off-diagonal block with the largest Frobenius norm for annihilation under the constraint that it can be annihilated in parallel with the already chosen blocks. In fact, numerical experiments show that it converges faster than the cyclic block Jacobi method, which is also a popular variant of the block Jacobi method [6].

However, when the off-diagonal blocks have comparable Frobenius norms, the strategy of choosing the off-diagonal blocks with the largest norm loses its advantage and the convergence acceleration effect of the method is diminished. Such a situation is apt to occur when the matrix order $n$, and therefore the block size $L \equiv n/(2p)$ becomes large, due to the statistical averaging effect. Hence, it is desirable to introduce a new mechanism to resolve this problem and accelerate the convergence even in such cases.

3. Combinatorial preconditioning for the parallel block Jacobi method

3.1 Main idea and mathematical formulation

In this paper, we propose to accelerate the convergence of the parallel block Jacobi method by introducing combinatorial preconditioning. The idea is to permute the rows and columns of $A^{(k)}$ prior to annihilation (before line 8 of Algorithm 1) so that elements with large absolute values are concentrated in $p \times 2L$ blocks on the diagonal. Then, we diagonalize these $p$ diagonal blocks by setting $X_1 = 2\ell - 1$ and $Y_1 = 2\ell$ ($\ell = 1, 2, \ldots, p$) and executing lines 8 through 15 of Algorithm 1. Then, it is expected that the elements with large modulus are annihilated at once and the off-diagonal norm is greatly reduced in one step. We repeat this combination of permutation and annihilation for $k = 1, 2, \ldots$. We call this
Algorithm 2 A greedy algorithm to solve (1)

1: \( J_0 = \{1, 2, \ldots, n\} \quad \triangleright \text{set of unused rows} \\
2: \text{for } \ell = 1, 2, \ldots, p \text{ do} \\
3: \quad J_\ell = \emptyset \\
4: \quad (i, j) = \arg \max_{i,j \in J_\ell, i \neq j} |a_{ij}^{(k)}|^2 \\
5: \quad J_0 := J_0 - \{i\} - \{j\} \\
6: \quad J_\ell := J_\ell \cup \{i\} \cup \{j\} \\
7: \text{for } r = 3, 4, \ldots, 2L \text{ do} \\
8: \quad s = \arg \max_{s \in J_\ell} \sum_{i \in J_\ell} |a_{is}^{(k)}|^2 \\
9: \quad J_0 := J_0 - \{s\} \\
10: \quad J_\ell := J_\ell \cup \{s\} \\
11: \text{end for} \\
12: \text{end for}

3.2 A greedy algorithm for the optimization problem

The problem (1) is a combinatorial optimization problem with 0-1 variables. While it is not shown to be an NP-hard problem, we have not been able to find an efficient polynomial time algorithm for it yet. We therefore propose to solve it approximately using a greedy algorithm. The strategy is to determine the rows (and columns) that constitute the first 2\(L \times 2L\) diagonal blocks of \(QA^{(k)}Q^\top\) that maximize this. This can be expressed as the following maximization problem

\[
\max_{Q \in \Pi_n} \sum_{\ell=1}^p \sum_{i,j=1}^{2L} \left| \left(Q^\top A^{(k)}Q\right)_{2L(\ell-1)+i,2L(\ell-1)+j} \right|^2,
\]

where \(\Pi_n\) is the set of permutation matrices of order \(n\).

4. Numerical experiments

4.1 Experimental conditions

We performed numerical experiments to evaluate the effectiveness of the combinatorial preconditioning. In our experiments, we used a workstation with the Intel Xeon E2660V2 processor, which has 10 cores running at 2.2GHz. Our program was written in C, parallelized using OpenMP and compiled with icc version 16.0.0. We used all the 10 cores. To compute the eigenvector matrix in line 9 of Algorithm 1 and to perform the matrix multiplications in lines 10, 13 and 14, we used Intel Math Kernel Library. As test matrices, we used random matrices whose elements obey uniform distribution in [0, 10). The matrix order \(n\) was set to 1600, 3200 and 6400.

4.2 Numerical results

Effect of preconditioning First, we study how well the combinatorial preconditioning using Algorithm 2 can gather matrix elements with large modulus into 2\(L \times 2L\) diagonal blocks. To see this, we picked up intermediate matrices of the parallel block Jacobi method before and after the preconditioning and compared the Frobenius norms (F-norms) of the \(L \times L\) blocks. The result is shown as heat maps in Fig. 1. Here, \(n = 3200\) and \(L = 160\), and we used the generalized (blockwise) version of combinatorial preconditioning described in subsection 3.3 with permutation block size \(M = 8\). As an intermediate matrix, we used a matrix at a step when the Frobenius norm of each \(L \times L\) block is \(O(1)\). In the figure, only the strictly block upper triangular part is shown.

As can be seen from the figure, a lot of \(L \times L\) blocks of comparable Frobenius norms that were present before preconditioning were restructured as a result of permutation and blocks of larger Frobenius norms (yellow ones) were generated in the 2\(L \times 2L\) diagonal blocks. Thus, we conclude that the combinatorial preconditioning works as expected in this case. Note that the upper triangular part of the diagonal 2\(L \times 2L\) blocks is zero before preconditioning, because they were annihilated in iterations. We therefore chose to use the approximate solution obtained by Algorithm 2 directly in this paper.

3.3 Generalization

So far, we have considered permuting rows and columns of \(A^{(k)}\). Instead, we can consider row blocks and column blocks of width \(M\), say, and permute them. Here, we assume that \(M\) is a divisor of the block size \(L\). This amounts to replacing \(\Pi_n\) in (1) with its subset corresponding to block-wise permutations. In that case, we use the Frobenius norm of \(M \times M\) submatrices of \(A^{(k)}\) instead of \(|a_{ij}^{(k)}|\) in Algorithm 2. While this modification lowers the maximum of the optimization problem (1) due to the restriction of the search space, the computational work for Algorithm 1 is reduced from \(O(n^2)\) to \(O(n^2/M^2)\). Moreover, the decrease of the maximum of (1) does not necessarily lead to slower convergence of the block Jacobi method, as we will discuss in Section 5. Thus, we will try several values of \(M\) in the numerical experiments in the next section.
the previous step. We also checked the effect of preconditioning using matrices from other parts of iteration and confirmed that it works well throughout the iteration.

**Convergence acceleration effect** Now, we evaluate the convergence acceleration effect of the combinatorial preconditioning. To this end, we plotted the off-diagonal norm of $A^{(k)}$ against the number of sweeps. Here, one sweep is defined as $(q-1)/2$ iterations, which is required to annihilate each off-diagonal block once in the cyclic block Jacobi method. The results of $n = 6400$ and $L = 320$ are shown in Fig. 2. There are six curves, namely, the conventional parallel block Jacobi method with dynamic ordering (PDO) and the proposed method with $M = 10, 20, 40, 80$ and $160$. From the graph, it can be seen that the proposed method succeeds in reducing the number of sweeps until convergence. What is interesting is that, contrary to our expectation, decreasing the permutation block size does not lead to faster convergence; rather the proposed method with $M = L/2 = 160$ was the fastest. This tendency was observed for other combinations of $n$ and $L$ as well. We will discuss a possible reason for this in Section 5.

**Execution time** Finally, we show the execution time of the existing and proposed methods in Fig. 3. The parameters are the same as those for Fig. 2. The fastest one is the proposed method with $M = 160$, which is 10% faster than the existing method. The time for combinatorial preconditioning was about 12% of the total execution time in this case. The proposed method with $M = 10$ is slower than the existing method for two reasons: slower convergence and the time of preconditioning. The same tendency was seen in other cases as well and the case with $M = L/2$ was the fastest also for $n = 1600$ and $n = 3200$.

5. Discussion

The preliminary numerical results given in the previous section suggests that the combinatorial preconditioning is a promising new idea for accelerating the convergence of the parallel block Jacobi method. We are planning to do extensive numerical experiments using various types of test matrices and architectures.

In the numerical results, the case with $M = 10$ was slow not only in terms of execution time but also in terms of convergence of the off-diagonal norm. This seems strange considering that the search space in the optimization problem (1) is the largest when $M = 10$. A possible explanation for this is that permutation with small block size mixes annihilated (zero) blocks and nonzero blocks completely, thereby destroying the quadratic convergence property that guarantees fast convergence of the block Jacobi method. We plan to study this topic further in the future.

Acknowledgment

We thank the anonymous referee for insightful comments. We are also grateful to Prof. Shao-Liang Zhang of Nagoya University for valuable discussion, which led to the idea of combinatorial preconditioning. This study is supported by JSPS KAKENHI Grant Numbers 17H02828, 17K19966 and 19KK0255.

References

[1] J. Demmel and K. Veselić, Jacobi’s method is more accurate than QR, SIAM J. Matrix Anal. Appl., 13 (1992), 1204–1245.

[2] J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić and Z. Drmač, Computing the singular value decomposition with high relative accuracy, Linear Algebra Appl., 299 (1999), 21–80.

[3] S. Kudo, Implementation and performance evaluation of the block Jacobi method for the symmetric eigenvalue problem on K computer (in Japanese), Master’s Thesis, Kobe University, 2015.

[4] S. Kudo, Y. Yamamoto, M. Bećka and M. Vajteršic, Performance analysis and optimization of the parallel one-sided block Jacobi SVD algorithm with dynamic ordering and variable blocking, Concurr. Comput., 29 (2017), e4059.

[5] M. Bećka, G. Okša and M. Vajteršic, Dynamic ordering for a parallel block-Jacobi SVD algorithm, Parallel Comput., 28 (2002), 243–262.

[6] S. Kudo, K. Yasuda and Y. Yamamoto, Performance of the parallel block Jacobi method with dynamic ordering for the symmetric eigenvalue problem, JSIAM Lett., 10 (2018), 41–44.