

Research Article

A Novel Parallel Algorithm Based on the Gram-Schmidt Method for Tridiagonal Linear Systems of Equations

Seyed Roholah Ghodsi and Mohammad Taeibi-Rahni

*Mechanical and Aerospace Engineering Department, Science and Research Branch,
Islamic Azad University (IAU), Tehran 1477-893855, Iran*

Correspondence should be addressed to Seyed Roholah Ghodsi, roholahghodsi@gmail.com

Received 8 June 2010; Revised 19 September 2010; Accepted 6 December 2010

Academic Editor: David Chelidze

Copyright © 2010 S. R. Ghodsi and M. Taeibi-Rahni. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper introduces a new parallel algorithm based on the Gram-Schmidt orthogonalization method. This parallel algorithm can find almost exact solutions of tridiagonal linear systems of equations in an efficient way. The system of equations is partitioned proportional to number of processors, and each partition is solved by a processor with a minimum request from the other partitions' data. The considerable reduction in data communication between processors causes interesting speedup. The relationships between partitions approximately disappear if some columns are switched. Hence, the speed of computation increases, and the computational cost decreases. Consequently, obtained results show that the suggested algorithm is considerably scalable. In addition, this method of partitioning can significantly decrease the computational cost on a single processor and make it possible to solve greater systems of equations. To evaluate the performance of the parallel algorithm, speedup and efficiency are presented. The results reveal that the proposed algorithm is practical and efficient.

1. Introduction

Linear systems of equations occur frequently in many fields of engineering and mathematics. Different approaches have been used to solve the linear systems of equations [1], for example, QR factorization. There are different QR factorization methods, such as Givens transformations, Householder transformations, and Gram-Schmidt orthogonalisation.

A considerable amount of study has been done on parallel QR factorization algorithms [2, 3]. In addition, some numerical libraries and softwares have supplied the QR factorization, such as ScaLAPACK [4], and Parallel Linear Algebra Software for Multicore Architectures (PLASMA) [5]. PLASMA Tile QR factorization is provided for multicore system architectures using the block algorithms [6].

Tridiagonal system is a well known form that arises in areas such as linear algebra, partial differential equations, orthogonal polynomials, and signal processing. Problem of solving tridiagonal systems on parallel computers has been studied widely. It seems that first parallel tridiagonal system solver, referred to as cyclic reduction, was introduced by Hockney in 1965 [7]. Throughout more than forty years since then, different parallel algorithms have been proposed and developed. The recursive doubling algorithm was presented by Stone for fine grained parallel systems in 1973 [8]. Wang suggested a partitioning algorithm for coarse grained systems in 1981 [9]. The classical method is the inverse of the matrix containing unknown elements; recently, an explicit formula was given by Kilic [10].

In this paper, we consider a novel parallel algorithm based on the modified Gram-Schmidt orthogonalization method [11]. This method is a kind of QR factorization, in which firstly orthogonal and upper-triangular matrices are computed and then the solution of the system of equations is found using these matrices. Amodio et al. proposed a parallel factorization for tridiagonal matrices [12, 13]. On the other hand, parallel implementation of modified Gram-Schmidt algorithm has been studied in different literatures [14–17]. Although the proposed method in this paper is currently developed and verified for tridiagonal systems, the authors are working on extension of the method to other systems.

The scalability of parallel algorithms is a vital problem, so different techniques have been developed to overcome it such as processors load balancing [18] and reducing the number of synchronizations [19]. The obtained results show that the suggested parallel algorithm is practically scalable.

Therefore, the objective of this paper is to propose an efficient parallel algorithm. In Section 2, the QR factorization is introduced as a base of this algorithm. In Section 3, the procedure of the suggested parallel algorithm is completely described. A numerical example is given in Section 4. In Section 5, accuracy, speedup, and efficiency of the algorithm are evaluated. Finally, conclusions are given in Section 6.

2. QR Factorization

A QR factorization of a nonsingular matrix $A \in \mathbb{R}_{n \times n}$ is a decomposition of the matrix into a unitary orthogonal matrix $Q \in \mathbb{R}_{n \times n}$ (i.e., $Q^T Q = I$) and an upper-triangular matrix $R \in \mathbb{R}^{n \times n}$ as

$$A = QR, \quad (2.1)$$

where the decomposition is unique. One of the most well-known QR methods is the Classical Gram-Schmidt (CGS) process. To overcome numerical instability, the modified Gram-Schmidt algorithm (MGS) is preferred after a simple change in the classical algorithm [20]. Each column of orthogonal matrix can be normalized as follows:

$$q_k = \frac{u^{(k)}}{\|u^{(k)}\|}, \quad (2.2)$$

where $q^{(k)}$ is a column of orthogonal matrix and $1 \leq k \leq n$. Column $u^{(k)}$ is determined from previously computed orthogonal columns $q^{(i)}$, where $1 \leq i \leq k - 1$, as

$$u^{(k)} = a^{(k)} - \sum_{i=1}^{k-1} (a^{(k)}, q^{(i)}) q^{(i)}, \quad (2.3)$$

where $(a^{(k)}, q^{(i)}) q^{(i)}$ denotes the projection operator of the column $a^{(k)}$ onto the column $q^{(i)}$ and

$$(a^{(k)}, q^{(i)}) = \frac{\langle a^{(k)}, q^{(i)} \rangle}{\langle q^{(i)}, q^{(i)} \rangle}, \quad (2.4)$$

where, $\langle a^{(k)}, q^{(i)} \rangle$ presents the inner product of the columns $a^{(k)}$ and $q^{(i)}$. Loss of orthogonality depends on matrix A , and the amount of deviation from orthogonality is minimized when A is a well-conditioned matrix [21].

Additionally, an upper-triangular matrix is constructed column by column as

$$r_{ik} = (u^{(k)}, q^{(i)}), \quad (2.5)$$

where r_{ik} is a typical element of upper-triangular matrix and $1 \leq i \leq n$.

Furthermore, the QR factorization can be used as a part of solution of a linear system of equations. To obtain the solution of a linear system

$$AX = B, \quad [7] \quad X \in \mathbb{R}^n, \quad B \in \mathbb{R}^n, \quad (2.6)$$

we consider the QR factorization

$$QR X = B. \quad (2.7)$$

Then we multiply the transpose of Q by both sides of (2.7), where $Q^T Q = I$ and $Q^T B = B'$. Finally, a linear system with an upper-triangular coefficient matrix is achieved as

$$R' X = B', \quad (2.8)$$

where $R' = R$. The final step is computing the matrix X by usual back substitution.

3. Parallel Algorithm

The concept of novel parallel algorithm based on the Gram-Schmidt method will be described here. The parallel implementation of the Gram-Schmidt method is not efficient compared with other methods, particularly when it is employed to solve a tridiagonal system of equations. However, some modifications allow appropriate conditions needed to parallelize the Gram-Schmidt method. In the Gram-Schmidt method, computation of each column of the

$$\begin{aligned}
q^{(1)} &= a^{(1)} + \\
q^{(2)} &= a^{(2)} + \frac{\overbrace{|a^{(2)} \cdot q^{(1)}|}^{r_{1,2}}}{|q^{(1)} \cdot q^{(1)}|} q^{(1)} \\
q^{(3)} &= a^{(3)} + \frac{\overbrace{|a^{(3)} \cdot q^{(1)}|}^{r_{1,3}}}{|q^{(1)} \cdot q^{(1)}|} q^{(1)} + \frac{\overbrace{|a^{(3)} \cdot q^{(2)}|}^{r_{2,3}}}{|q^{(2)} \cdot q^{(2)}|} q^{(2)} \\
q^{(4)} &= a^{(4)} + \frac{\overbrace{|a^{(4)} \cdot q^{(1)}|}^{r_{1,4}}}{|q^{(1)} \cdot q^{(1)}|} q^{(1)} + \frac{\overbrace{|a^{(4)} \cdot q^{(2)}|}^{r_{2,4}}}{|q^{(2)} \cdot q^{(2)}|} q^{(2)} + \frac{\overbrace{|a^{(4)} \cdot q^{(3)}|}^{r_{3,4}}}{|q^{(3)} \cdot q^{(3)}|} q^{(3)} \\
q^{(5)} &= a^{(5)} + \frac{\overbrace{|a^{(5)} \cdot q^{(1)}|}^{r_{1,5}}}{|q^{(1)} \cdot q^{(1)}|} q^{(1)} + \frac{\overbrace{|a^{(5)} \cdot q^{(2)}|}^{r_{2,5}}}{|q^{(2)} \cdot q^{(2)}|} q^{(2)} + \frac{\overbrace{|a^{(5)} \cdot q^{(3)}|}^{r_{3,5}}}{|q^{(3)} \cdot q^{(3)}|} q^{(3)} + \frac{\overbrace{|a^{(5)} \cdot q^{(4)}|}^{r_{4,5}}}{|q^{(4)} \cdot q^{(4)}|} q^{(4)} \\
q^{(6)} &= a^{(6)} + \frac{\overbrace{|a^{(6)} \cdot q^{(1)}|}^{r_{1,6}}}{|q^{(1)} \cdot q^{(1)}|} q^{(1)} + \frac{\overbrace{|a^{(6)} \cdot q^{(2)}|}^{r_{2,6}}}{|q^{(2)} \cdot q^{(2)}|} q^{(2)} + \frac{\overbrace{|a^{(6)} \cdot q^{(3)}|}^{r_{3,6}}}{|q^{(3)} \cdot q^{(3)}|} q^{(3)} + \frac{\overbrace{|a^{(6)} \cdot q^{(4)}|}^{r_{4,6}}}{|q^{(4)} \cdot q^{(4)}|} q^{(4)} + \frac{\overbrace{|a^{(6)} \cdot q^{(5)}|}^{r_{5,6}}}{|q^{(5)} \cdot q^{(5)}|} q^{(5)} \\
&\dots
\end{aligned}$$

Figure 1: The zero and nonzero terms in the equation of each orthogonal column, where $q^{(m)}$ is the m th column of orthogonal matrix and $a^{(m)}$ is m th column of matrix A . Also, the zero terms are crossed out.

orthogonal and upper-triangular matrices depends on the previously computed columns of the orthogonal matrix.

Most of the elements of a tridiagonal matrix are zero, except for those on three diagonals. Each term of projection operator in (2.3) consists of inner product of one column of this tridiagonal matrix with a computed column of orthogonal matrix. When most of the elements of one of these columns are zero, then a few elements of projection operator can be nonzero. As a result, only the last two terms of projection operator in the computation of each column of orthogonal matrix are nonzero, as

$$r_{ik} \neq 0, \quad i = k - 2, k - 1. \quad (3.1)$$

Figure 1 shows the terms of (2.3) for the first six columns of orthogonal matrix Q .

As a result, computation of each column of the orthogonal and upper-triangular matrices depends only on the two previous adjacent columns of the orthogonal matrix Q . It means that, if the column (k) of orthogonal matrix is computing, only two computed columns ($k - 1$) and ($k - 2$) of matrix Q and the column (k) of matrix A are requested, as

$$u^{(k)} = a^{(k)} - \left(a^{(k)}, q^{(k-2)} \right) q^{(k-2)} - \left(a^{(k)}, q^{(k-1)} \right) q^{(k-1)}. \quad (3.2)$$

This property is a base of our parallel algorithm. In this situation, if we omit two adjacent columns of matrix A , then all terms of projection operator for the next columns are zero.

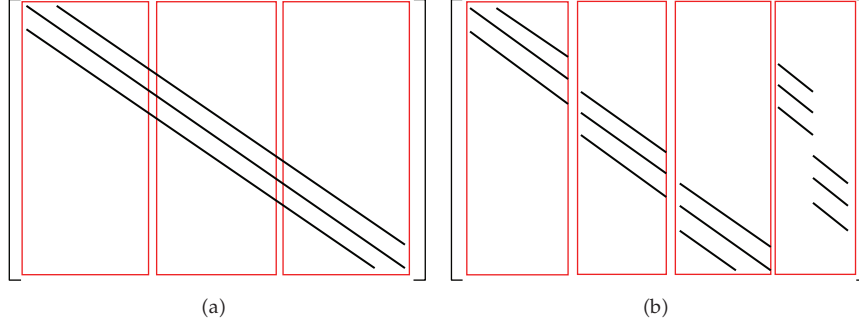


Figure 2: The partitioning (a) and transformation (b) of matrix's columns. The amount of partition can be related to the number of processor. The last partition is made by transferred columns from between other blocks.

It means that the relation of columns of orthogonal matrix in (2.3) is disrupted. In order to use this feature, if two columns of matrix $A_{n \times n}$, for instance, $a^{(k-2)}$ and $a^{(k-1)}$, are transferred to the end of matrix, the next corresponding columns in the orthogonal matrix are disconnected from the previous ones. It means that $(a^{(k)}, q^{(k-2)}) = 0$ and $(a^{(k)}, q^{(k-1)}) = 0$ and as a result,

$$u^{(k)} = a^{(k)}. \quad (3.3)$$

This transformation decreases the amount of computation significantly.

The parallel algorithm is achieved by partitioning the matrix $A_{n \times n}$ into $m - 1$ (m is the number of processors) blocks of fairly equal size, that is, each block contains approximately $N_e = n / (m - 1)$ columns, and each processor is concerned with computing only its block. The computation can be done mostly independently by each processor, but at some points, communication is required. The first column of the matrix owned by processor e is denoted as j_e , where $j_e < j_{e+1}$ for $e = 1, \dots, m - 1$ and $N_e = j_{e+1} - j_e$.

To disconnect the relationship between adjacent blocks, two columns between them, that is, $j_{e+1} - 1$ and j_{e+1} for $e = 1, \dots, m - 2$, are transferred to the end of matrix (3.4). These transferred columns make another partition, as shown in Figure 2. Therefore, the number of columns of the new block is $H = 2 \times (m - 2)$:

$$j_m + (e - 1) \leftarrow j_{e+1} - 1, \quad j_m + e \leftarrow j_{e+1} \quad \text{for } e = 1, \dots, m - 2. \quad (3.4)$$

It should be noticed that the place of each element of matrix X corresponding to the transferred columns of matrix A is also changed.

Each processor, except for the last one, can start computing some columns of the orthogonal and upper-triangular matrices that are related to the columns of its own block without receiving any data from other processors. This feature plays a significant role in improving the speedup of the algorithm. If processor P_e is responsible for a block consisting of columns from $a^{(j_e)}$ to $a^{(j_{e+1}-1)}$, it computes the columns from $q^{(j_e)}$ to $q^{(j_{e+1}-1)}$ of the orthogonal matrix with a minimum request for data of other partition, and the same holds for $r^{(j_e)}$ to $r^{(j_{e+1}-1)}$ of the upper-triangular matrix. All the blocks, except the last one, only need data of the last block, because the first two and last two columns of each block have some relation to some columns of the last block.

To achieve more speedup, another switching is used in each block, except for the last block. The last two columns in each block are transferred between the second and third columns of block. This switching causes the first four columns to be computed firstly and then be sent to the last processor before other columns of block are determined. As a result, all processors are in use simultaneously, and the efficiency of the parallel algorithm increases.

Lastly, all columns of the orthogonal and upper-triangular matrices are computed with the least network transaction. To show the final result, each processor sends its block to the last processor.

The next step is to find the solution of the linear system of equations. According to (2.7), the values of $Q^T Q$ and $Q^T B$ should be determined. Because data of each block of matrix Q and also matrix B exist in each processor, the computation of these matrices can be done without any request to the other processors' data.

Finally, the unknown matrix is obtained by a back substitution procedure. In this step, each processor just needs to communicate with the last processor. The back substitution is started from the last processor which consists of H columns. The last H elements of the unknown matrix are computed using just the last processor and are sent to other ones. Because the H elements of matrix X are computed, the last processor can compute independently, the last H terms of each linear equation. As a result, instead of H columns, just one column is needed to broadcast to the other processors. Then the other processors compute the related unknown elements with the received data. The complete flowchart of described parallel algorithm is shown in Figure 3.

It may seem that the amount of computation in the last processor is more than the others. However, it should be mentioned that the number of columns in the last block is less than the number of columns in the others. If the number of unknowns increases, the difference between size of the last block and other blocks becomes greater.

4. Example

Let A be a 15×15 nonsingular tridiagonal matrix. The results of three cases are presented here and compared with each other; we use the label case (a) for one processor, case (b) for two, and case (c) for three. In Figure 4, the transformations of two columns in case (b) (two processors) and four columns in case (c) (three processors) are shown.

The computed orthogonal matrices in the three cases are presented in Figure 5. The number of nonzero elements is 135 in case (a), 97 in case (b), and "93" in case (c). Hence, the computational cost decreases as the computational power (i.e., the number of processors) increases. This situation is an ideal condition for a parallel algorithm.

Finally, the upper-triangular matrices are shown in Figure 6. Notice that, in cases (b) and (c), the elements in the last row of each block are zero, except for elements that belong to the last processors, in addition to an element which is equal to 1. Therefore, the back substitution procedure in each block does not require the data of the other processors, only the last processor. This feature is also significant for solving a huge system of equations.

5. Results

The testbed specification is shown in Table 1. The results of using up to 16 processors are presented here. The numerical experiment is done on a parallel system with MIMD (Multiple

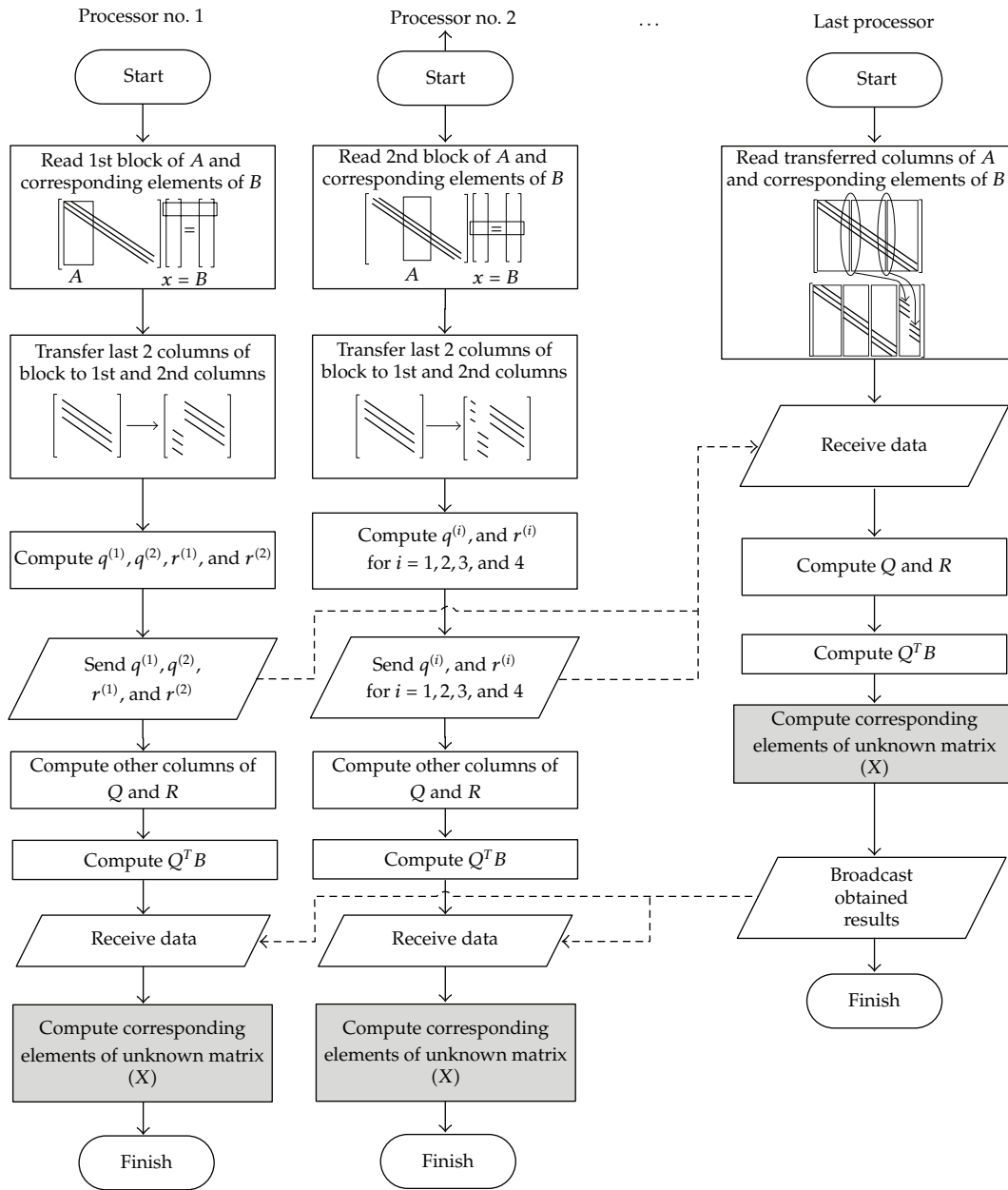
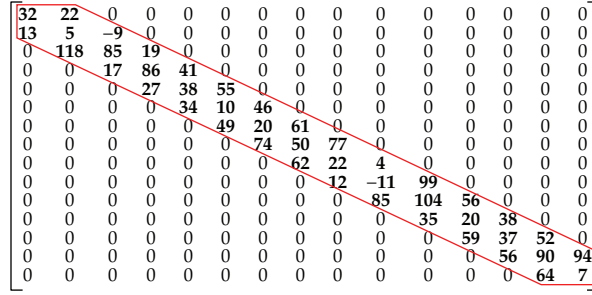


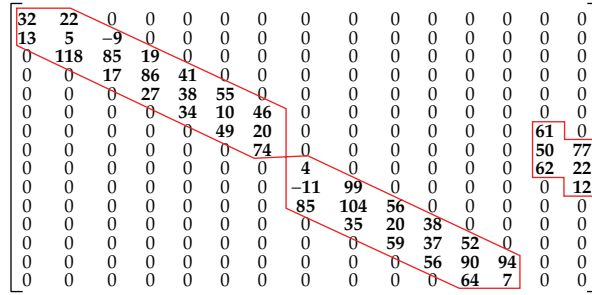
Figure 3: The flowchart of proposed parallel algorithm.

Data, Multiple Instruction) structure and shared memory [22]. The language of parallel processing that is used in this paper is standard MPI [23].

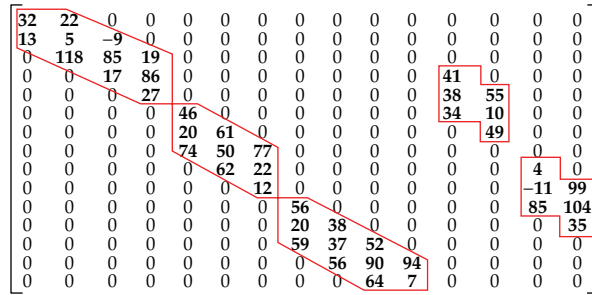
The results of the procedure described above consist of different aspects. Its main purpose was to find the solution of a tridiagonal system of equations in a fast and efficient manner. The other achievements were computing the orthogonal and upper-triangular matrices which are useful computations in different fields. This entire procedure in



(a)



(b)



(c)

Figure 4: The matrix $A_{15 \times 15}$; (a) one processor, (b) two processors, and (c) three processors.

Table 1: The Specification of test bed.

CPU	RAM	Cache	Network	OS	Language
up to 16× Intel Core 2 Due 3.0GHz	4 GB	4 MB	100 Mb/s	Win XP	MPI

a sequential form requires $O(n^2)$ number of operations, where n is the number of equations. The complexity of the parallel algorithm is

$$9H^2 + 26H + 32n + 130, \quad (5.1)$$

where $H = n/p$ is the ratio of the number of equations to the number of processors. Furthermore, as mentioned in Section 3, the number of zero elements raises as the number of blocks increases. This feature can be thought of as being on route to linear complexity.

32	1.37	2.15	-9.96	4.52	6.58	-5.40	-1.99	5.39	0.64	-6.27	0.32	0.45	-0.82	-0.34
13	-3.38	-5.28	24.5	-11.1	-16.2	13.3	4.90	-13.3	-1.57	15.4	-0.87	-1.10	2.03	0.84
0	118	-0.18	0.82	-0.37	-0.54	0.44	0.16	-0.44	-5.23	0.51	-2.62	-3.69	6.76	2.80
0	0	17	8.89	-4.04	-0.87	4.82	1.78	-4.81	-0.57	5.60	-0.28	-0.40	0.73	0.30
0	0	0	27	13.1	19.1	-15.6	-5.77	15.6	1.85	-18.2	0.92	1.30	-2.39	-0.99
0	0	0	0	34	-14.2	11.69	4.31	-11.7	-1.38	13.6	-0.69	-0.97	1.78	0.74
0	0	0	0	0	49	15.19	5.60	-15.2	-1.79	17.6	-0.90	-1.26	2.32	0.96
0	0	0	0	0	0	74	-4.19	11.3	1.34	-13.2	0.67	0.94	-1.73	-0.72
0	0	0	0	0	0	0	62	5.76	0.68	-6.71	0.34	0.48	-0.88	-0.36
0	0	0	0	0	0	0	0	12	-9.86	97.1	-4.94	-6.95	12.7	5.27
0	0	0	0	0	0	0	0	0	85	12.9	-0.65	-0.92	1.69	0.70
0	0	0	0	0	0	0	0	0	0	35	15.9	22.3	-41.1	-17.0
0	0	0	0	0	0	0	0	0	0	0	59	-6.71	12.3	5.10
0	0	0	0	0	0	0	0	0	0	0	0	56	19.7	8.17
0	0	0	0	0	0	0	0	0	0	0	0	0	64	-15.6

(a)

32	1.37	2.15	-9.96	4.52	6.58	-5.40	0	0	0	0	0	0	-1.99	5.41
13	-3.38	-5.28	24.5	-11.1	-16.2	13.3	0	0	0	0	0	0	4.90	-13.3
0	118	-0.18	0.82	-0.37	-0.54	0.44	0	0	0	0	0	0	0.16	-0.44
0	0	17	8.89	-4.04	-0.87	4.82	0	0	0	0	0	0	1.76	-4.83
0	0	0	27	13.1	19.1	-15.6	0	0	0	0	0	0	-5.77	15.7
0	0	0	0	34	-14.2	11.69	0	0	0	0	0	0	4.31	-11.7
0	0	0	0	0	49	15.19	0	0	0	0	0	0	5.60	-15.2
0	0	0	0	0	0	74	0	0	0	0	0	0	-4.19	11.4
0	0	0	0	0	0	0	4	-4.21	-2.12	1.98	-0.75	-2.51	61.1	5.78
0	0	0	0	0	0	0	-11	110.6	-5.11	-6.96	12.8	5.21	3.84	0.36
0	0	0	0	0	0	0	85	14.5	-0.56	-0.99	1.70	0.79	-2.37	-0.22
0	0	0	0	0	0	0	0	35	16.1	22.6	-41.4	-17.1	-3.80	-0.36
0	0	0	0	0	0	0	0	0	59	-6.73	12.4	5.04	3.54	0.33
0	0	0	0	0	0	0	0	0	0	56	19.9	8.26	0.24	0.02
0	0	0	0	0	0	0	0	0	0	0	64	-15.7	-3.22	-0.30

(b)

32	1.37	2.15	-9.96	0	0	0	0	0	0	0	4.52	7.02	0.64	-6.08
13	-3.38	-5.28	24.5	0	0	0	0	0	0	0	-11.1	-17.3	-1.57	14.9
0	118	-0.18	0.82	0	0	0	0	0	0	0	-0.37	-0.58	-0.05	0.50
0	0	17	8.89	0	0	0	0	0	0	0	-4.04	-6.3	-0.57	5.42
0	0	0	27	0	0	0	0	0	0	0	13.1	20.3	1.85	-17.6
0	0	0	0	46	-28.3	-25.9	0	0	0	0	10.1	-15.2	-1.38	13.1
0	0	0	0	20	48.7	-26.1	0	0	0	0	-8.29	16.7	-1.79	17.1
0	0	0	0	74	4.44	23.2	0	0	0	0	-4.06	4.94	1.34	-12.8
0	0	0	0	0	62	6.96	0	0	0	0	11.4	-20.4	0.68	-6.50
0	0	0	0	0	0	12	0	0	0	0	5.08	5.73	-9.85	94.1
0	0	0	0	0	0	0	56	-23.5	24.8	-0.99	0	0	11.1	12.5
0	0	0	0	0	0	0	20	29.6	-16.1	22.6	0	0	17.8	20.0
0	0	0	0	0	0	0	59	12.2	-18.1	-6.73	0	0	-16.6	-18.6
0	0	0	0	0	0	0	0	56	22.8	56	0	0	-1.12	-1.26
0	0	0	0	0	0	0	0	0	-17.5	56	0	0	15.1	16.9

(c)

Figure 5: The orthogonal matrices; (a) one processor, (b) two processors, and (c) three processors.

Choosing a proper method to solve tridiagonal system of equations depends on different criteria, such as the order of complexity, accuracy and variety of achievements. For instance, the order of complexity of Gaussian elimination without pivoting is $O(n)$, but it is not well suited for parallel processing [24]. On the other hand, in spite of favorable order of complexity of parallel iterative algorithms, their main focus is just on solution of the system with good enough accuracy but without any further outcomes.

1	0.64	-0.1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0.72	0.16	0	0	0	0	0	0	0	0	0	0	0
0	0	1	4.53	2.17	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0.92	0.98	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0.71	1.05	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0.1	0.90	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0.73	0.90	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0.26	0.06	0	0	0	0	0
0	0	0	0	0	0	0	0	1	-0.09	1.04	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1.07	0.65	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0.12	0.11	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0.74	0.81	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1.25	1.41
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.35
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(a)

1	0.64	-0.1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0.72	0.16	0	0	0	0	0	0	0	0	0	0	0
0	0	1	4.53	2.17	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0.92	0.98	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0.71	1.05	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0.1	0	0	0	0	0	0	0.90	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0.73	0.91
0	0	0	0	0	0	0	1	1.05	0.65	0	0	0	0.03	0
0	0	0	0	0	0	0	0	1	0.11	0.1	0	0	-0.02	0.09
0	0	0	0	0	0	0	0	0	1	0.74	0.81	0	-0.03	0.03
0	0	0	0	0	0	0	0	0	0	1	1.25	1.40	0.03	-0.01
0	0	0	0	0	0	0	0	0	0	0	1	0.35	0	0.02
0	0	0	0	0	0	0	0	0	0	0	0	1	-0.23	0.01
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.27
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(b)

1	0.64	-0.1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0.72	0.16	0	0	0	0	0	0	0	0	0	0	0
0	0	1	4.53	0	0	0	0	0	0	2.17	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0.92	0.98	0	0	0
0	0	0	0	1	0.61	0.71	0	0	0	0.19	0.18	0	0	0
0	0	0	0	0	1	0.24	0	0	0	-0.14	0.30	0.03	0	0
0	0	0	0	0	0	1	0	0	0	-0.42	-0.74	-0.05	0.57	0
0	0	0	0	0	0	0	1	0.42	0.44	0	0	0	0.68	0.93
0	0	0	0	0	0	0	0	1	1.20	1.11	0	0	-0.42	-0.29
0	0	0	0	0	0	0	0	0	1	0.38	0	0	0.05	-0.17
0	0	0	0	0	0	0	0	0	0	1	0	0	1.04	0.99
0	0	0	0	0	0	0	0	0	0	0	1	0.61	-0.01	0.74
0	0	0	0	0	0	0	0	0	0	0	0	1	-0.08	0.31
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.76
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(c)

Figure 6: The Upper-triangular matrices; (a) one processor, (b) two processors, and (c) three processors.

The complexities of different QR factorization methods in sequential form are brought here as follows [25]:

- (1) Gram-Schmidt: $(2/3)n^3 + O(n^2)$,
- (2) Householder: $(4/3)n^3 + O(n^2)$,
- (3) Given: $(8/3)n^3 + O(n^2)$,
- (4) Fast Given: $(4/3)n^3 + O(n^2)$.

Table 2: A norm of $\|X\|$ to evaluate error over all computed unknowns and a norm of $\|Q^T Q - I\|$ to determine the accuracy of the orthogonal matrix.

order	$\ X\ $	$\ Q^T Q - I\ $
$n = 200\,000$	$6.91e-7$	$9.39e-12$
$n = 300\,000$	$4.48e-8$	$8.62e-12$
$n = 400\,000$	$8.78e-8$	$9.32e-12$
$n = 500\,000$	$3.01e-9$	$8.52e-12$

There are some important points to justify our algorithm. The first is the accuracy of obtained results. Obviously, the accuracy of direct solvers, that is, mostly exact, is better than that of the iterative ones. The second item is the variety of results, such as computing the orthogonal and upper-triangular matrices. The QR decomposition is usually used to solve the linear least squares problem and find the eigenvalues of system. The third is the scalability of the algorithm, which is a crucial parameter for success of a parallel algorithm, that is, the proposed algorithm can employ more processors to solve larger problems efficiently. Consequently, development of this efficient algorithm based on the QR factorization is valuable and practical. Some sequential and parallel QR factorization methods are investigated by Demmel et al. [26].

The obtained results are mostly exact. The accuracy of the algorithm was investigated using two methods. The first method was a norm of $\|X\|$ to evaluate error over all computed unknowns, and the second one was a norm of $\|Q^T Q - I\|$ to determine the accuracy of the orthogonal matrix. The level of error for systems of order $n = 10\,000$, $20\,000$, $30\,000$, and $40\,000$ is shown in Table 2. The order of error in these cases is acceptable. The final results, that is, the computed unknown, became more accurate in larger matrices. Moreover, the level of orthogonality is high in all cases.

Furthermore, this algorithm includes a new and interesting parallel feature. The parallel algorithm can decrease the computational cost and the requested memory on a single processor. The MPI can be used on a single processor to divide it into different parts. In other words, the parallel algorithm based on the MPI library can be executed on different parts of a processor in the same way as it is executed on a network of parallel CPUs. The enormous number of equations causes the processor to return the “insufficient virtual memory” error. Increasing the number of blocks in the matrix results in increasing the amount of zero elements and decreasing the size of the requested memory (physical and virtual). In Figure 7, the size of the requested memory for a matrix of order $n = 20\,000$ and $50\,000$ is shown. For instance, computation of a matrix of order $n = 50\,000$ was not possible on a specified processor with a single block. However, the computation was done on this processor with 4 blocks using 4.75 GB of memory. When the number of blocks was 16, the used memory decreased to 1.48 GB. The consumed time to find the solution of a system of equations of order $n = 20\,000$ and $n = 50\,000$ is presented in Figure 8. It is shown that when the number of blocks increased, time consumption was reduced. The rate of reduction nearly vanished when the number of blocks reached 16.

The key issue in the parallel processing of a single application is speedup, especially its dependence on the number of processors used. Speedup is defined as a factor by which the execution time for the application changes:

$$S_p = \frac{T_s}{T_p}, \quad (5.2)$$

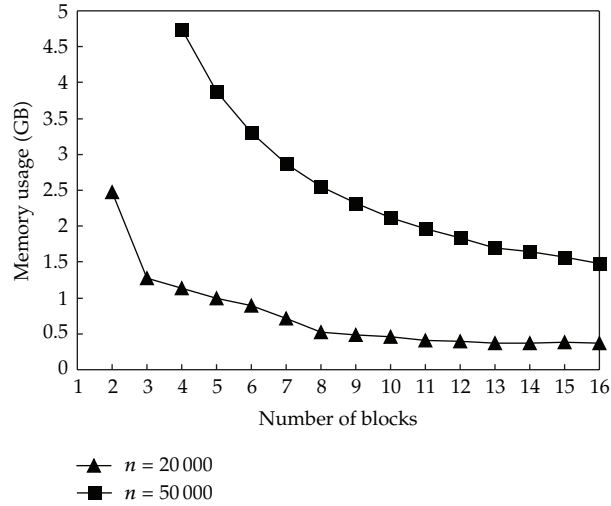


Figure 7: The size of requested memory in computation by a single CPU.

where, p , T_s , and T_p are the number of processor, the executing time of the sequential algorithm, and the executing time of the parallel algorithm with p processors, respectively.

On the other hand, the efficiency of parallel processing is a performance metric which is defined as

$$E_p = \frac{S}{p}. \quad (5.3)$$

It is a value, typically between zero and one, estimating how well utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization.

To investigate the ability of the parallel algorithm, the CPU time (T), the speedup (S_p), and the efficiency (E_p) were considered in different cases (Table 3). The mentioned orders are 20 000, 40 000, 60 000, 80 000, and 100 000. The first column shows the computational time in a sequential case. A critical problem in solving a large system of equations using one processor, as mentioned before (e.g., order more than 50 000 in this study), is insufficient memory. To conquer this obstacle, the virtual memory was increased to 10 GB. Although using virtual memory can remove the “insufficient memory” error, it causes the speed of the computation to decrease, leading to inefficiency. The memory problem is completely solved in the parallel algorithm because each processor just needs to have the data of its block. For instance, if the order of matrix is 100 000, then in two processors the order of the data is 50 000, and in three processors it will reach to about 33 000. Therefore, the size of a huge system of equations does not create any computational limitation on the parallel cases.

The speedup and efficiency curves for the computation of a tridiagonal system of equations from orders 20 000 to 100 000 using different numbers of processors are presented in Figures 9–11. The comparison between computation times of different sizes of systems of equations using up to 16 processors is shown in Figure 9. The rate of time reduction was greater in larger systems. For instance, the time was reduced from 267 milliseconds to 22 milliseconds, that is, 11 times less than sequential form.

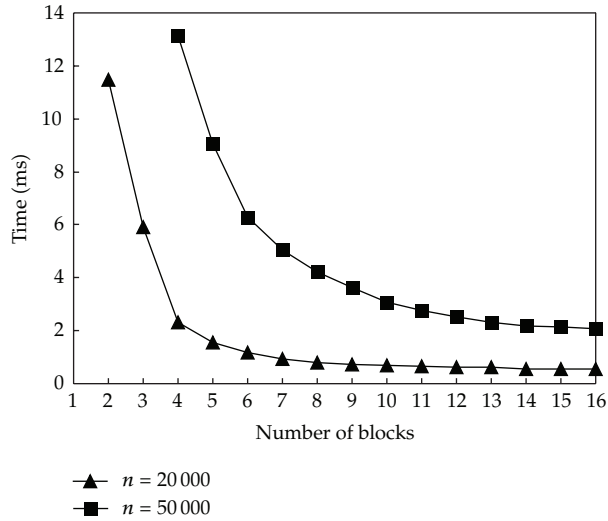


Figure 8: Time of computation by a single CPU.

Table 3: The computation time, speed up, and efficiency of parallel solution of the systems of equations with different order.

	Processors no.									
	1	2	4	6	10	12	14	16	18	
<i>n</i> = 100 000	<i>T</i>	0.00338	0.00175	0.00092	0.00067	0.00056	0.00051	0.00051	0.00052	0.00054
	<i>S_p</i>	—	1.927	3.651	5.027	5.954	6.528	6.566	6.466	6.171
	<i>E_p</i>	—	0.964	0.913	0.838	0.744	0.653	0.547	0.462	0.386
<i>n</i> = 200 000	<i>T</i>	0.02495	0.01289	0.00678	0.00487	0.00406	0.00365	0.00347	0.00336	0.00336
	<i>S_p</i>	—	1.935	3.679	5.121	6.139	6.822	7.180	7.425	7.423
	<i>E_p</i>	—	0.968	0.920	0.854	0.767	0.682	0.598	0.530	0.464
<i>n</i> = 300 000	<i>T</i>	0.05973	0.03025	0.01601	0.01140	0.00931	0.00821	0.00761	0.00700	0.00662
	<i>S_p</i>	—	1.974	3.731	5.239	6.413	7.272	7.845	8.523	9.017
	<i>E_p</i>	—	0.987	0.933	0.873	0.802	0.727	0.654	0.609	0.564
<i>n</i> = 400 000	<i>T</i>	0.12345	0.06221	0.03225	0.02252	0.01787	0.015441	0.01427	0.01320	0.01236
	<i>S_p</i>	—	1.984	3.828	5.482	6.907	7.995	8.635	9.350	9.983
	<i>E_p</i>	—	0.992	0.957	0.914	0.863	0.800	0.720	0.668	0.624
<i>n</i> = 500 000	<i>T</i>	0.26724	0.12916	0.06842	0.04718	0.03648	0.03094	0.02694	0.02397	0.02248
	<i>S_p</i>	—	2.069	3.905	5.664	7.325	8.635	9.918	11.147	11.887
	<i>E_p</i>	—	1.034	0.976	0.944	0.916	0.864	0.827	0.796	0.743

The amount of speedup in different problem sizes is shown in Figure 10. When the order of the system was 20 000 and the number of processors was greater than 12, the speedup decreased. The reason for this inefficiency is the effect of network delay on computation time. On the other hand, in the larger problem sizes, this problem does not exist.

It is encouraging to note that, with an increase in problem size, the efficiency of the parallel algorithm increases. In Figure 11, the efficiency of different executions is presented. It is obvious that the rate of reduction in the efficiency of the parallel algorithm decreased

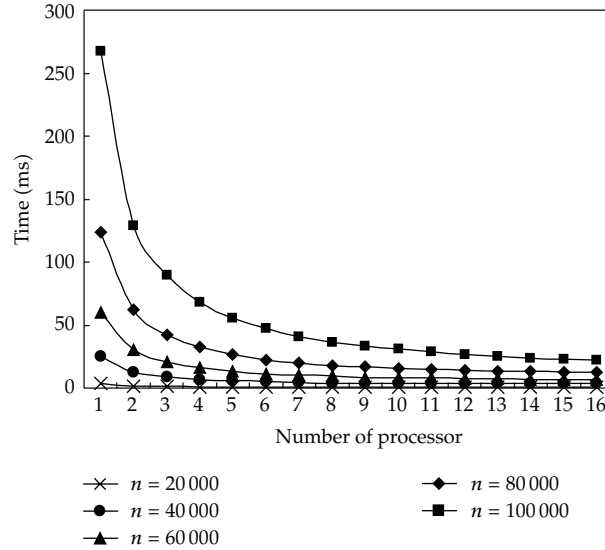


Figure 9: The computation time of system of equation in different orders using up to 16 processors.

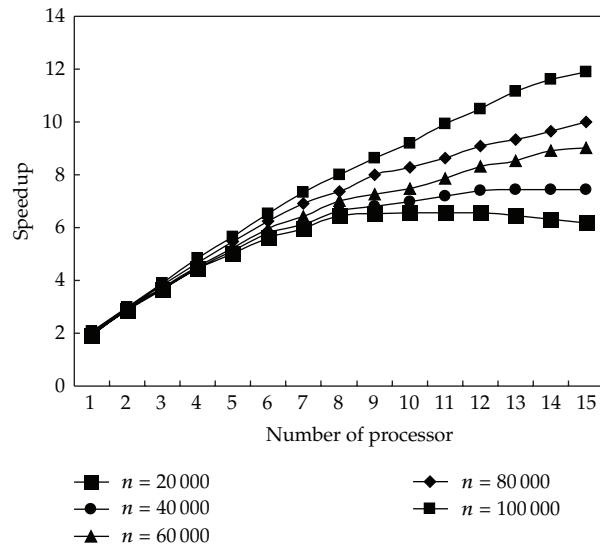


Figure 10: The speedup of parallel algorithm.

by raising the problem size. This characteristic makes this parallel algorithm a practical subroutine in numerical computations.

In addition, the speedup of proposed algorithm is compared with parallel Cycle Reduction algorithm (Figure 12). Cycle Reduction is a well-known algorithm for solving system of equations [27]. The speedup of both algorithms is shown in two orders of system of equations, that is, $n = 200\,000$ and $400\,000$. Obviously, the speedup of proposed algorithm is more than that of Cycle Reduction and also would be greater by increasing number of equations and processors.

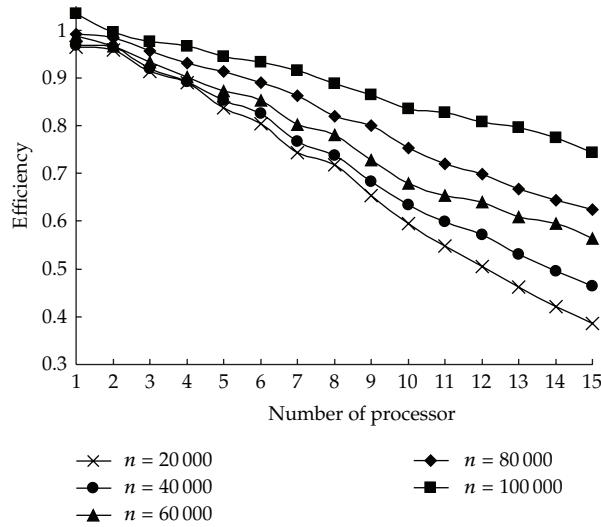


Figure 11: The efficiency of parallel algorithm.

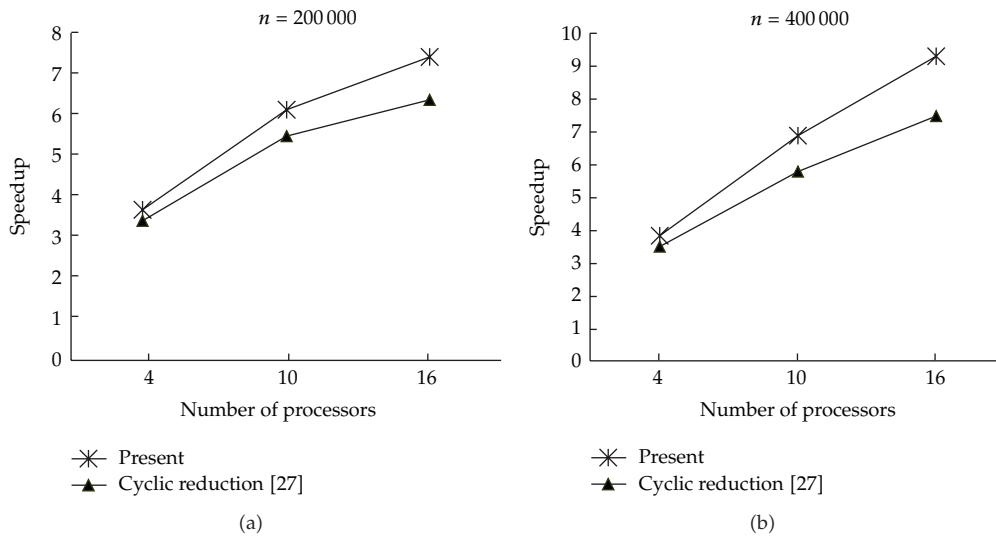


Figure 12: The speedup comparison between proposed algorithm and Cycle Reduction in two different orders of system of equations: (a) $n = 200\,000$ and (b) $n = 400\,000$.

6. Conclusion

In this paper, a novel parallel algorithm based on the Gram-Schmidt method was presented to solve a tridiagonal linear system of equations in an efficient way. Solving linear systems of equations is a famous problem in different scientific fields, especially systems in the tridiagonal form. Therefore, the efficiency and accuracy of the algorithm can play a vital role in the entire solution procedure.

In this parallel algorithm, the coefficient matrix is partitioned according to the number of processors. After that, some columns in each block are switched. Therefore,

the corresponding orthogonal and upper-triangular matrices are computed by each processor with a minimum request from the other partitions' data. Finally, the system of equations is solved almost exactly.

The proposed algorithm has some remarkable advantages, which include the following:

- (1) the ability to find almost exact solutions to tridiagonal linear systems of equations of different sizes, as well as the corresponding orthogonal and upper-triangular matrices;
- (2) considerable reduction of computation cost proportional to the number of processors and blocks;
- (3) increasing the efficiency of the algorithm by raising the size of the system of equations;
- (4) the significant ability of a single processor using the MPI library; the requested memory decreases by partitioning the matrix, and hence the computational speed increases.

The orders of the systems of equations that are presented in this paper range from 10 000 to 100 000. The results show that the suggested algorithm is scalable and completely practical.

References

- [1] B. N. Datta, *Numerical Linear Algebra and Applications*, SIAM, Philadelphia, Pa, USA, 2nd edition, 2010.
- [2] D. P. O'Leary and P. Whitman, "Parallel QR factorization by householder and modified Gram-Schmidt algorithms," *Parallel Computing*, vol. 16, no. 1, pp. 99–112, 1990.
- [3] A. Pothen and P. Raghavan, "Distributed orthogonal factorization: givens and Householder algorithms," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 6, pp. 1113–1134, 1989.
- [4] J. Blackford, J. Choi, A. Cleary et al., *ScaLAPACK Users' Guide*, SIAM, Philadelphia, Pa, USA, 1997.
- [5] E. Agullo, J. Dongarra, B. Hadri et al., "PLASMA 2.0 users' guide," Tech. Rep., ICL, UTK, 2009.
- [6] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, "Parallel tiled QR factorization for multicore architectures," *Concurrency Computation Practice and Experience*, vol. 20, no. 13, pp. 1573–1590, 2008.
- [7] R. W. Hockney, "A fast direct solution of Poisson's equation using Fourier analysis," *Journal of the Association for Computing Machinery*, vol. 12, pp. 95–113, 1965.
- [8] H. S. Stone, "An efficient parallel algorithm for the solution of a tridiagonal linear system of equations," *Journal of the Association for Computing Machinery*, vol. 20, pp. 27–38, 1973.
- [9] H. H. Wang, "A parallel method for tridiagonal equations," *ACM Transactions on Mathematical Software*, vol. 7, no. 2, pp. 170–183, 1981.
- [10] E. Kilic, "Explicit formula for the inverse of a tridiagonal matrix by backward continued fractions," *Applied Mathematics and Computation*, vol. 197, no. 1, pp. 345–357, 2008.
- [11] G. H. Golub and C. F. van Loan, *Matrix Computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, Md, USA, 3rd edition, 1996.
- [12] P. Amodio, L. Brugnano, and T. Politi, "Parallel factorizations for tridiagonal matrices," *SIAM Journal on Numerical Analysis*, vol. 30, no. 3, pp. 813–823, 1993.
- [13] P. Amodio and L. Brugnano, "The parallel QR factorization algorithm for tridiagonal linear systems," *Parallel Computing*, vol. 21, no. 7, pp. 1097–1110, 1995.
- [14] S. R. Ghodsi, B. Mehri, and M. Taeibi-Rahni, "A parallel implementation of Gram-Schmidt algorithm for dense linear system of equations," *International Journal of Computer Applications*, vol. 5, no. 7, pp. 16–20, 2010.
- [15] R. Doallo, B. B. Fraguera, J. Touriño, and E. L. Zapata, "Parallel sparse modified Gram-Schmidt QR decomposition," in *Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, vol. 1067 of *Lecture Notes In Computer Science*, pp. 646–653, 1996.

- [16] E. L. Zapata, J. A. Lamas, F. F. Rivera, and O. G. Plata, "Modified Gram-Schmidt QR factorization on hypercube SIMD computers," *Journal of Parallel and Distributed Computing*, vol. 12, no. 1, pp. 60–69, 1991.
- [17] L. C. Waring and M. Clint, "Parallel Gram-Schmidt orthogonalisation on a network of transputers," *Parallel Computing*, vol. 17, no. 9, pp. 1043–1050, 1991.
- [18] B. Kruatrachue and T. Lewis, "Grain size determination for parallel processing," *IEEE Software*, vol. 5, no. 1, pp. 23–32, 1988.
- [19] A. W. Lim and M. S. Lam, "Maximizing parallelism and minimizing synchronization with affine partitions," *Parallel Computing*, vol. 24, no. 3-4, pp. 445–475, 1998.
- [20] L. Giraud, J. Langou, and M. Rozloznic, "The loss of orthogonality in the Gram-Schmidt orthogonalization process," *Computers & Mathematics with Applications*, vol. 50, no. 7, pp. 1069–1075, 2005.
- [21] A. Björck and C. C. Paige, "Loss and recapture of orthogonality in the modified Gram-Schmidt algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 13, no. 1, pp. 176–190, 1992.
- [22] T. Wittwer, *An Introduction to Parallel Programming*, VSSD uitgeverij, 2006.
- [23] M. Snir and W. Gropp, *MPI: The Complete Reference*, MIT Press, Cambridge, UK, 2nd edition, 1998.
- [24] T. M. Austin, M. Berndt, and J. D. Moulton, "A memory efficient parallel tridiagonal solver," Tech. Rep. LA-UR 03-4149, Mathematical Modeling and Analysis Group, Los Alamos National Laboratory, Los Alamos, NM, USA, 2004.
- [25] W. M. Lioen and D. T. Winter, "Solving large dense systems of linear equations on systems with virtual memory and with cache," *Applied Numerical Mathematics*, vol. 10, no. 1, pp. 73–85, 1992.
- [26] J. Demmel, L. Grigori, M. F. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," Tech. Rep. UCB/EECS-2008-89, Electrical Engineering and Computer Sciences University of California at Berkeley, Berkeley, Calif, USA, 2008.
- [27] P. Amodio, "Optimized cyclic reduction for the solution of linear tridiagonal systems on parallel computers," *Computers & Mathematics with Applications*, vol. 26, no. 3, pp. 45–53, 1993.