Scientia Iranica A (2017) 24(6), 2762–2775

Sharif University of Technology
Scientia Iranica
Transactions A: Civil Engineering
www.scientiairanica.com

Modal analysis of two-dimensional beams using parallel finite-element method

S. Heydari and S. Asil Gharebaghi*

Civil Engineering Faculty, K. N. Toosi University of Technology, Tehran, Iran.

Received 14 December 2015; received in revised form 31 May 2016; accepted 8 August 2016

KEYWORDS
Eigenvalue problem;
Parallel processing;
FEM;
CSR matrix compression;
Davidson algorithm;
Modified checkered method.

Abstract. Modal analysis is the process of determining the natural frequencies and mode shapes of structures. In practical problems, modal analysis may be repeated many times, resulting in a huge quantity of computations. Although parallel processing technique can reduce the analysis time, civil engineers rarely implement it because it requires high programming skills as well as designing parallel algorithms. In the present paper, the Davidson algorithm is adopted for parallel modal analysis of two-dimensional beams. More precisely, the parallel version of the Davidson algorithm is implemented from scratch. A new proposed method, which is called "Modified Checkered Method" (MCM), is introduced, and four versions of the algorithm are implemented. Two out of four versions use row-wise and MCM in combination with Compressed Sparse row Algorithm, while the others utilize the previously mentioned methods without matrix compression. It is shown that the speedup increases when the main matrix of the standard form of eigenvalue problem is not compressed. Moreover, the speedup will increase in comparison to the row-wise division method when MCM is used. It is notable that the implemented Parallel Finite-Element source code is capable of being used in companion with a wide variety of finite elements.

1. Introduction

Modal analysis has turned into an important tool in determination, improvement, and optimization of the intrinsic dynamic characteristics of various types of structures. Intrinsic dynamic characteristics of a structure are its natural frequencies and modal shapes. Natural frequencies and modal shapes are calculated using eigenvalues and eigenvectors, respectively. The set of an eigenvalue and its associated eigenvector is called an eigenpair. The dynamical behavior of the structure can be mathematically expressed using these characteristics. Modal analysis results are also useful in structure design. The dynamical response of the structure under any imposed load could be calculated using the results of modal analysis. Thus, modal analysis is an important step in understanding the behavior of a structure. For instance, Garinei [1] studied the vibrations of a simple beam under the impact of harmonically moving loads. In addition, the analysis of vibrations of a bridge using modal analysis is another example of the studies done in this field [2].

Conventionally, computer programs are written in a serial manner. The commands of an algorithm in this manner are executed sequentially and on one single processor, only. More precisely, in serial mode, only one command can be executed at a specific time, and the next command has to be executed after the previous command. Therefore, a complex algorithm turns into many commands, which wait to be executed in an ordered queue. On the other hand, in parallel processing, several processors are exploited

*. *Corresponding author. Tel.: +98 21 88779473*
E-mail addresses: soroush16@gmail.com (S. Heydari); asil@kntu.ac.ir (S. Asil Gharebaghi)

doi: 10.24200/sci.2017.4529

simultaneously. In this manner, the main algorithm is divided into as many independent sub-algorithms as possible. After division into independent parts, each part is assigned to a separate processor. Eventually, by simultaneous execution of these independent parts, the parallel program will be executed in a short time. The amount of shortening of the time is a function of the ratio of the independent parts of the algorithm to all of it and the architecture used in parallel processing. Considering that every processor needs to communicate with at least one other processor, if the amount of the information transferred between the processors increases excessively, a notable portion of time will be consumed for handling the traffic and decreasing the performance of analysis using parallel processing.

Calculating the eigenvalues of the matrix, which is formed using stiffness and mass matrices of the structure, is one of the most important and costly steps in performing modal analysis. Practically, there exists a variety of solving methods based on the type of the algorithm used for calculating the eigenpairs and the parallelization method of this algorithm. A study of this sort is conducted in [3], where modal analysis is performed by dividing the problem into some subdomains using Component Mode Synthesis (CMS) technique in finite-element theory; subsequently, each element of the finite-element mesh is assigned to a separate processor. In the aforementioned study, the Jacobi algorithm is used for modal analysis. According to the numerical results of this study, using the CMS method increases the speedup of the parallelization, but the duration of the overall processing also increases significantly. In fact, in this method, there will be a need for calculating the inverse of some parts of the main matrix, which in turn highly increases the computational complexity. For instance, in numerical results section of the previously mentioned work, modal analysis of a cantilever beam with 50306 degrees of freedom was studied. The analysis has taken 4419 seconds, with 3113 seconds (about 70%) taken to execute the commands of the CMS algorithm. In [4], Davidson algorithm and parallel power method algorithm are used for solving a critical calculation problem, i.e. a problem related to the calculation of eigenvalues and eigenvectors in atomic reactors. Unlike what is usual in civil engineering, there are some of the greatest eigenvalues that are demanded in eigenvalue problems like the one in [4]. In the above study, a comparison between parallel power method and Davidson algorithms has led to this conclusion that Davidson algorithm gives better results in terms of execution time and precision.

A two-dimensional beam has infinite Degrees Of Freedom (DOF). The number of DOFs can become limited using discretization and finite-element method, and thus the problem can be solved numerically. The more degrees of freedom are considered, the more accurate the solution will be. However, using more degrees of freedom results in a longer computation time. Using parallel processing together with modal analysis suggests many advantages, e.g. speeding up the computations and the possibility of using decentralized systematic and hardware resources. Therefore, parallel processing helps to analyze problems that are more complex in a short time. Performing parallel analysis needs the serial algorithm to have parallelization potential. Fortunately, the costly steps of Davidson algorithm have this advantage. Additionally, this algorithm is able to compute a limited number of the smallest eigenpairs. In this study, CMS technique is not used to reduce the overall processing time, but instead, various techniques of sharing the elements of a matrix between processors are investigated to measure their performance together with Davidson algorithm.

2. Parallel programs speedup and its limits

The speedup of a parallel program is expressed using the ratio between processing time in serial mode and processing time in the case of using np processors. Generally, the speedup of a parallel program can also be a function of the number of the processors. Speedup can be expressed as Eq. (1):

$$S_{np} = \frac{t_0}{t_{np}}. \tag{1}$$

In this formula, S_{np} denotes the speedup of the parallel program that uses np processors compared to the program that uses one. t_0 and t_{np} denote processing times using one processor and np processors, respectively.

In an ideal case, the speedup caused by parallelization of an algorithm must be linear. The "ideal case" means a case in which all parts of the algorithm have the ability of parallelization and no data would wait in queue to reach a specific node or processing unit. In other words, no time should be wasted because of the data traffic between the processors. In the case of ideality, processing time could be halved by doubling the number of the processors. Although this case would never happen, comparing the real speeds with those of the ideal case could be helpful in determining the efficiency of the algorithms. As mentioned before, parallelization ability in the main algorithm and high traffic between the processors are two main factors which affect the limitation of the speedup caused by parallelization.

On the one hand, although Eq. (1) suggests that the speedup is linearly related to the number of the processors, in practice and for most of the parallel programs, it is approximately linear only for small number of processors. In fact, when the number of processors grows up, the speedup approaches a constant asymptote. In problems that do not require much computational effort, parallel processing will not increase solution speed, and even will lead to an increase in process time. In such cases, the time consumed for communication among the processors will dominate and reduce the speedup caused by parallelization.

On the other hand, the speedup in the execution of a parallel program, using a specific number of processors, sometimes goes beyond the expected values. This kind of speedup is called "super linear speedup". The main causes of this condition are related to Central Processing Unit (CPU) and its architecture as well as parallelization algorithm. Based on a hardware point of view, each CPU has a multiple layers of cache memory called L1, L2, etc. having very high access speed in comparison to Random Access Memory (RAM). Therefore, if all the required data for each processor could be totally placed in the cache memory of that processor, each processor would not need to communicate with other processors via RAM. As a result, the speed of the computations would dramatically grow. Moreover, one of the main methods for parallelization of an algorithm, which can cause super linear speedup, is "parallelism over data". The emergence of super linear speedup in parallelism over data is often called "caching effect". In this case, which usually takes place in solving problems with rather low complexity and amount of data, the amount of allocated data to each processor is lessened by increasing the number of the processors, which participate in the process. Therefore, the data portion of each processor can be completely placed in the cache memory portion of that processor satisfying the above-mentioned condition. As prescribed above, super linear speedup phenomenon could happen under these conditions.

Branch-and-bound algorithms are the example of parallel algorithms with specific properties, which may result in super linear speedup. In some case, the process, which is being executed by a processor over a node of the domain, can affect the processing of other processors over other nodes. In other words, the result of one processor's work can decrease the workload of other processors significantly, again leading to super linear speedup.

It should be noted that Amdahl's law may only be applied when the resulting speedup in parallel processing is not caused by the data being placed in cache memory, the special conditions occurred in the algorithm, or super linear speedup [5].

3. Hardware and software platforms

Parallel execution of the programs requires special hardware and software platforms. Distributed and shared memory systems are two examples of these platforms. Since distributed systems are cheaper than shared memory ones, these systems are of more use nowadays [6]. Two elements of the used software platforms are the operating system and a consistent library for writing parallel programs. The operating system, which is usually used for executing parallel programs, is Linux. Microsoft Windows could also be used as the operating system, but due to the significant disadvantages of Microsoft Windows compared to Linux, the former is not being used very frequently [7]. In the present study, Ubuntu operating system, which is a Linux distribution, has been used. Besides, the C++ programming language is used for implementing the algorithms. Additionally, GCC is one of the most powerful and free compilers available under Linux operating system; therefore, it is chosen to compile the source codes. Furthermore, MPI library, which enables the developed software to use distributed memory, has been deployed as the framework for producing parallel program. Availability and being free of charge are two main reasons for using the aforementioned software combination.

4. Problem definition and assumptions

The problem that is being discussed in this study is performing the modal analysis of two-dimensional beams using parallel processing. During the analysis, it is assumed that the modal analysis is linear. The main part of the problem is to find eigenpairs of the beam in a parallel manner. It is assumed that the beam is made of materials which do not damp, but it can have arbitrary dimensions and support specifications. Here, the assumption is that the beam acts like a cantilever, although its boundary conditions can be arbitrary. Since the beam is assumed two-dimensional, its thickness should be considered low enough compared to its height and length. A two-dimensional beam example, considered in this paper, is shown in Figure 1.

5. Governing equations

The governing equation of free oscillation of undamped Multiple-Degrees-Of-Freedom (MDOF) systems can be written as follows [8]:

$$M\ddot{u} + Ku = 0, \tag{2}$$

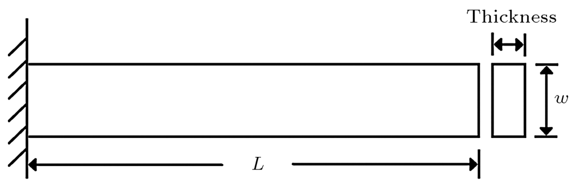

Figure 1. Example of the studied problem.

where K is the stiffness matrix, M is the mass matrix, and u is the displacement matrix of the structure. The free oscillation of an undamped Multiple-Degrees-Of-Freedom (MDOF) system can be expressed as follows:

$$u(t) = \sum_{i=1}^{N} q_i(t)\phi_l, \tag{3}$$

where ϕ_i is the ith mode shape function and $q_i(t)$ is called time coordinate or briefly the ith mode coordinate, which is a function of time. N is the number of the degrees of freedom. The time coordinate functions are defined in Eq. (4):

$$q_i(t) = F_i \sin \omega_l t + G_i \cos \omega_l t. \tag{4}$$

In Eq. (4), F_i and G_i are integrating constants, and ω_t is the natural resonance frequency. These parameters can be determined using the boundary conditions. The combination of Eqs. (3) and (4) results in Eq. (5):

$$u(t) = \sum_{i=1}^{N} \phi_t \left(F_i \sin \omega_l t + G_i \cos \omega_l t\right), \tag{5}$$

where ω_l and ϕ_l are unknown. By replacing $u(t)$ from Eq. (5) in Eq. (2), the following equation is derived:

$$\left[-\omega_l^2 M\phi_l + K\phi_l\right] q_i(t) = 0. \tag{6}$$

The acceptable answer to Eq. (6), which is not the trivial answer $u(t) = 0$, is:

$$K\phi_l = \omega_i^2 M\phi_l. \tag{7}$$

Useful numerical conclusions can be drawn from the above equation. The given algebraic problem is called matrix eigenvalue problem. Sometimes, it is necessary to add "real" suffix to distinguish it from complex eigenvalue problem. Stiffness matrix K and mass matrix M are known, and the unknown parameters of the problem are scalar ω_l^2 and vector ϕ_l, respectively. To make it similar to the classic solution, Eq. (7) is rewritten as follows:

$$\left[K - \omega_i^2 M\right] \phi_l = 0. \tag{8}$$

The above equation could be interpreted as N homogeneous algebraic equations for determining N number of ϕ_ls ($i = 1, 2, ..., N$). The nontrivial solution to the above equation is:

$$det\left[K - \omega_l^2 M\right] = 0. \tag{9}$$

Eq. (9) is a polynomial of degree N in ω_l^2, which is called characteristic or frequency equation. For practical structures in Civil Engineering, which have enough support conditions for stability and cannot have a rigid motion, matrix K is positive definite. Due to the characteristics of stiffness and mass matrices of the structure, Eq. (9) has N number of real and positive roots for ω_l^2. However, this expression is not true for structures like a flying airplane [8].

6. Solving the problem

6.1. Mesh, stiffness matrix, and mass matrix generation

The first step of solving the problem is to discretize or equivalently generate the Finite Element mesh of the beam, shown in Figure 1. Four-node rectangular plane stress element with eight degrees of freedom has been used to discretize the domain. Since the considered model does not contain any kind of geometrical inhomogeneity, finite-element mesh generation for calculating eigenvalues and eigenvectors is easy and efficient in the case of using four-node rectangular elements. Due to the points that will be given in the next sections, especially in Modified Checkered Decomposition section, it is necessary to number the nodes of the finite-element mesh in such a way that the nonzero elements of the resultant sparse matrix be concentrated around the main diagonal. For this reason, the numbering of the mentioned nodes is done along the width of the beam.

Subsequently, stiffness and mass matrices of the structure are created using the same finite-element mesh. It should be noted that the mass matrix has diagonalization ability. This ability results in an efficient use of memory, especially in large-scale problems. In fact, after diagonalization, only the main diagonal is stored in memory. Additionally, the inverse of the mass matrix will be easy to calculate. In this study, Hinton-Rock-Zienkiewicz (HRZ) scheme is used for diagonalization. This scheme is presented in [9].

6.2. Standard form of the equations

The standard form means a form of the eigenvalue problem, $Av = \lambda v$, in which both stiffness and mass matrices are included in matrix A. In this way, the eigenpairs of equation $K\phi = \omega^2 M\phi$ will be determined with less computational effort using matrix A. It is noteworthy that most of the proposed algorithms for solving eigenvalue problems try to solve the standard form. This is very advantageous, especially in large-scale problems. In fact, less memory will be needed during the implementation of this form. Additionally, when the eigenvalue solver algorithm deals with one matrix instead of two, the computational effort reduces significantly. Moreover, iterative algorithms for calculating eigenpairs of non-symmetric matrices usually have slower convergence and more computational complexity compared to the iterative algorithms for calculating eigenpairs of symmetric matrices. Fortunately, because stiffness and mass matrices are symmetric, matrix A will be symmetric. Therefore, the main characteristics of stiffness and mass matrices will be transported to matrix A, and this matrix will have real eigenvalues. Eventually, the algorithms developed for symmetric matrices can solve it.

In this paper, the following equation is used for calculating matrix A using stiffness and mass matrices [8]:

$$A = M^{-1/2} K M^{-1/2}. \tag{10}$$

6.3. Standard matrix compression

By performing matrix compression and not storing zero elements, besides the efficient use of the available memory, processing time could also be reduced notably by reducing the number of operations, performed just for nonzero elements. One of the most usual methods of matrix compression is Compressed Sparse Row (CSR) format. Another usual method is Symmetric Sparse Skyline (SSS). As it can be indicated from the name of this storing format, unlike CSR that is used for storing general matrices, SSS method is employed only for symmetric sparse matrices. One of the advantages of this method over CSR is its more efficient use of memory. Explanation on compression using the two mentioned methods can be found in [10].

If the SSS method is used in order to store matrix A and a multiplication of A to a vector is desired, each processor will not be capable of computing an independent part of the answer. Therefore, at the end of the multiplication, it is necessary to gather the outcomes of all processors to make the correct answer of the operation. Parallel function MPI_Reduce in MPICH library could be used for doing such an operation. Unfortunately, using the mentioned function imposes a significant overhead on the parallel processing performance. Additionally, the number of the executed commands for doing a multiplication task using both CSR and SSS methods is equal. Considering the overhead of the SSS method, caused by MPI_Reduce, if the SSS method is used, the execution speed will reduce compared to CSR. Therefore, despite all other advantages of using SSS, CSR method is used in this study.

6.4. Davidson algorithm

In most of the large-scale problems, iterative methods are used for finding a limited number of eigenpairs. One of the most important families of iterative methods, which are used for this end, are subspace methods. In these methods, the main matrix is projected onto a subspace. Then, the eigenpairs of this subspace are calculated and used for computing the eigenpairs of the main matrix. Davidson algorithm is a kind of subspace algorithms. The main idea of this algorithm is to increase the dimension of the subspace to calculate a limited number of the desired eigenpairs. If there exist specific eigenvectors in the subspace associated to that iteration in one of the iterations of this algorithm, eigenpairs of the main matrix could be calculated using eigenpairs of that subspace [11].

```
//**Choose an initial orthonormal matrix**
$B_1 = [b_1, b_2, ..., b_N]$
for $j = 1, 2, ...$ do
1- $H_j = B_j^T A B_j$
2- $(\rho_{j,i}, x_{j,i})_{1 \leq i \leq N}$
3- for $i = 1, ..., N$ do $u_{j,i} = B_j x_{j,i}$
4- for $i = 1, ..., N$ do $r_{j,i} = A u_{j,i} - \rho_{j,i} u_{j,i}$
   **if** convergence **then** exit
5- $\delta_{j,i} = C_{j,i} r_{j,i}$ **for** $i = 1, ..., N$
6- **if** $dim(B_j) \leq l - N$ **then**
   $B_{j+1} = MGS(B_j, \delta_{j,1}, ..., \delta_{j,N})$
    else
   $B_{j+1} = MGS(u_{j,1}, ..., u_{j,N}, \delta_{j,1}, ..., \delta_{j,N})$
    end if
 end for
```

Figure 2. Davidson algorithm.

Davidson algorithm's pseudocode is shown in Figure 2. In this pseudocode, N denotes the number of the desired eigenpairs and L is the maximum dimension of the subspace which is set by the user at the beginning of the procedure. The main steps of these algorithm are numbered to evaluate parallelization potential. Based on this numbering, each of the steps is explained comprehensively in the following steps.

- **Step 1:** *Subspace matrix creation.* One of the most resource-consuming steps of Davidson algorithm is performing a matrix by vector multiplication to produce subspace vector. Since the discussed matrix in this step is the main matrix of the problem, it could contain a large number of elements. Because of the parallelization potential of the multiplication, performing this multiplication in a parallel way is one of the most important parallelization steps of Davidson algorithm [12]. To perform this parallelization using master-slave pattern, the main matrix of the problem should first be properly distributed between the processors of the grid.

  Since the main matrix is used for multiplication in all iterations of the problem, every processor could be given its share of the main matrix before running the algorithm. The vectors by which the main matrix is multiplied will differ in each iteration, so the master processor should properly share these vectors among the processors of the grid. Then, each processor performs its share of matrix by vector multiplication and sends the result to the master processor. Eventually, the master processor puts the results of the network's processors in the right order and produces the resultant vector. It should be noted that in this form of master-slave parallelization, the master processor could also participate in parallel multiplication, i.e. this processor could have a share of the main matrix and help solve the problem in addition to its main task of sharing, receiving, and putting the vectors in the right order. This action will improve the efficiency of parallelization [12].

- **Step 2:** *Subspace eigenpairs calculation.* Since the maximum dimension of the subspace is set to be relatively low, the calculation of the eigenpairs of each subspace could be done easily with low computational effort and in a serial manner. Indeed, executing this operation in a parallel manner would not be effective. In this study, the maximum dimension of the subspace is set to be 15; therefore, serial QR algorithm is used in this step.
- **Step 3:** *Ritz vector calculation.* Ritz vector is calculated by multiplying a matrix containing eigenspace vectors, with dimensions limited to those of the subspace and main matrix, by the eigenvectors calculated from the subspace. Since the dimension of the subspace is limited and considered low (here, it is 15), the mentioned product is performed serially and with low computational effort. In fact, parallelization is not effective in this step.
- **Step 4:** *Residual vector calculation.* Like subspace creation, in the calculation of the residual vector, there exists a matrix by vector multiplication. In this step, similar to what was previously mentioned, the matrix by which the vector is multiplied is the main matrix of the problem which could contain a large number of elements. This matrix is shared among the processors from the beginning. Nonetheless, the vector by which the main matrix is multiplied will differ in each iteration; thus, in every iteration, it is shared among the processors before multiplication, and eventually, the master processor will help other processors to calculate the resultant vector in a parallel manner like in subspace matrix creation.
- **Step 5:** *Correction vector calculation.* If Jacobian correction vector is used in Davidson algorithm, the computational effort will be so low that parallelization cannot be effective. In fact, in this step, calculation only includes the multiplication of a vector by another vector. Actually, the first vector is made by inversing the diagonal elements of a diagonal matrix. However, using the correction vector is effective only when the main matrix is almost diagonal; otherwise, the convergence of Davidson algorithm will be very slow. Depending on the problem's details, if general correction vector is used in Davidson algorithm, this step can be considered as a potential for parallelization of the algorithm. In other words, when using general correction matrix, the problem to solve is a system of equations, in which the matrix of coefficients is the main matrix that can have a large number of elements. The resultant vector is the following residual vector:

$$A\delta = r. \tag{11}$$

In Eq. (11), A, δ, and r denote the standard matrix of the problem, the correction vector, and the residual vector, respectively. Due to the conditions of the problem studied in this paper, general correction vector is used. As a result, this step is one of the most time-consuming parts of Davidson algorithm, which also has the potential of parallelization [13]. For taking this step, Parallelized Preconditioned Conjugate Gradient method (PPCG) is used when solving the system of equations. This parallel algorithm will be discussed later.

- **Step 6:** *Orthogonalization of the subspace vectors.* This step is done using Gram-Schmidt algorithm. The matrix, which is being orthogonalized in this step, consists of vectors with the same dimension as the main matrix. The number of these vectors equals the maximum dimension of the subspace. Since this number has to be relatively small, i.e. 15, this step can be done easily in a serial manner and parallelization is not efficient.

6.5. Parallel preconditioned conjugate gradient algorithm

Because of the problem's specifications, Jacobi preconditioner matrix has been used. In this section, the parallelization potentials of preconditioned conjugate gradient algorithm are being discussed. These potentials can be studied for the two following factors:

- Dot product of two vectors;
- Matrix by vector multiplication.

Using state-of-the-art processors, parallelization of dot product is efficient only when the vectors are previously stored in the slave processors. If this condition holds, there will be no need for the master processor to send data to the slaves; otherwise, parallelization will increase data traffic. As a result, the dot product of two vectors has been done serially.

Considering the matrix by vector multiplication, which shows up in two steps of each iteration of this algorithm, the matrix that is multiplied by the vector is the main matrix of the problem, shared among the processors before running the algorithm because it will be needed in other parts of the problem. Thus, the master processor only has to share the vector among the slave processors, and the multiplication can be performed in a parallel way. Since the computational effort of this multiplication grows by an increase in the size of the system of equations, where performing it in a parallel way can increase the speed of the algorithm, this part of the multiplication algorithm has been parallelized.

6.6. Sharing the matrix among processors

For doing a parallel multiplication, the matrix involved in multiplication should be shared among the processors. The method of partitioning and sharing this matrix has a significant impact on the speedup of the parallel program. In this paper, two methods, namely row-wise and modified checkered decomposition, have been studied. There exist some other algorithms, e.g. column-wise and checkered decomposition, which have not been used due to their disadvantages, or because they do not fit the problems studied in this paper.

6.6.1. Row-wise decomposition

In this method, which is represented in Figure 3, the rows of the matrix are shared almost equally among the processors. It is considered "almost equally" because the number of the processors might not be an integer factor of the dimension of the matrix. So, the following formulas are used:

$$i_1 = (cp - 1) \times qr + 1$$

$$i_2 = \begin{cases} cp \times qr & cp < np \\ cp \times qr + mr & cp = np \end{cases} \tag{12}$$

where:

i_1 :	The first number of the rows allocated to each processor;
i_2 :	The last number of the rows allocated to each processor;
cp :	Number of the processor involved in the equations;
np :	Total number of the processors;
qr :	The integer quotient of matrix dimension and number of the processors;
mr :	Remainder of matrix dimension divided by number of the processors.

To perform the parallel multiplication with this method of sharing, the whole vector by which the matrix is multiplied should be sent to all processors.

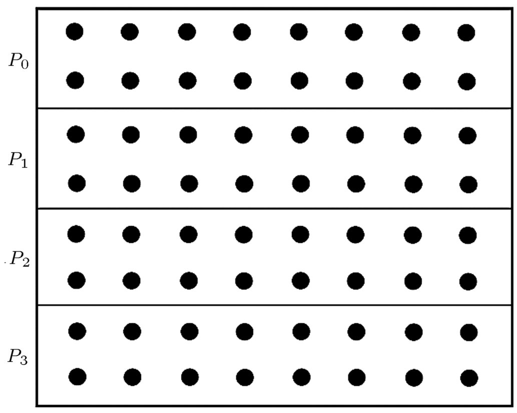

Figure 3. Row-wise sharing of matrix elements among the processors.

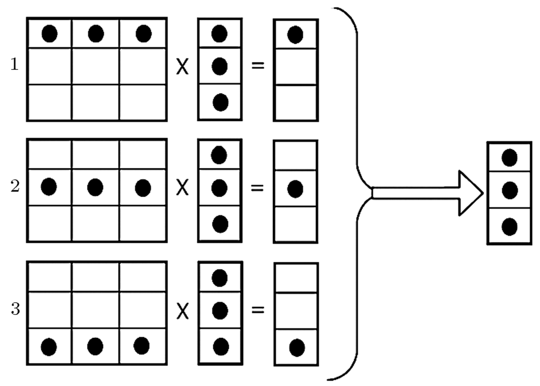

Figure 4. Parallel multiplication in row-wise sharing of the matrix.

Eventually, each processor will produce an independent part of the resultant vector. This procedure is depicted in Figure 4.

In the end, the resultant vectors of the processors should just be put in the right order so that the overall resultant vector is achieved.

6.6.2. Modified checkered decomposition

In the present study, the original checkered decomposition is altered so that the method can be parallelized significantly. In the original checkered method, the whole matrix is divided into blocks, and the elements of each block are sent to a processor. As Figure 5 represents, in parallel multiplication using checkered decomposition, the vector is divided among the processors of each row of the shared matrix. Each part of the resultant vector of this multiplication is calculated by adding the resultant vectors of the processors of each row.

In finite-element related problems, by taking some measures in numbering of the finite-element mesh, the main matrix of the problem could be changed into a well-defined form, such that its elements become concentrated around the main diagonal. As a result, in checkered sharing method where the whole matrix is divided into blocks, one can only send the blocks

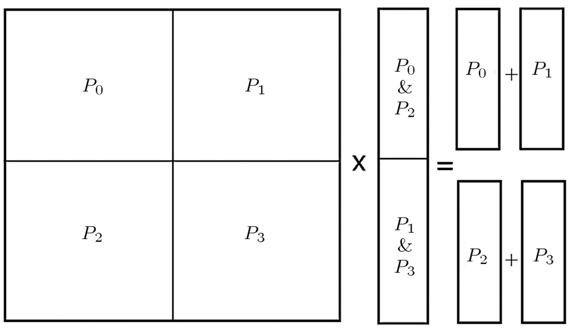

Figure 5. Sharing the elements among the processors and performing parallel multiplication using original checkered decomposition.

which are on the main diagonal plus small parts which are in the beginning and ending of a block to the processors. The algorithm, which is implemented in such a way in this paper is named Modified Checkered Decomposition. According to this method (Figure 6), in parallel multiplication, each processor is given a part of the vector with the same length as the dimension of the main block, plus another part with a length equal to the dimension of the small block next to the main block. As a result, a part of the shared data among the processors will be the same. In Figure 6, the hatched parts of the vector depict common data transferred to the surrounding blocks. As Figure 7 represents, in parallel multiplication using modified checkered sharing method, each processor produces an independent part of the overall resultant vector. The length of this vector is equal to the dimension of the main block sent to the processor. Eventually, the resultant vectors of each processor should be put in right order so that the overall resultant vector is created. Therefore, in modified checkered sharing method, despite checkered method, the vector is sent to the processors only once, but with a small amount of overlap, and different parts of the resultant vector are created by each processor independently.

Another advantage of the modified checkered decomposition over the original checkered version is the better use of the available storage without employing matrix compression methods. The difference between modified and original versions of the checkered method is better understood by comparing Figures 5 and 7.

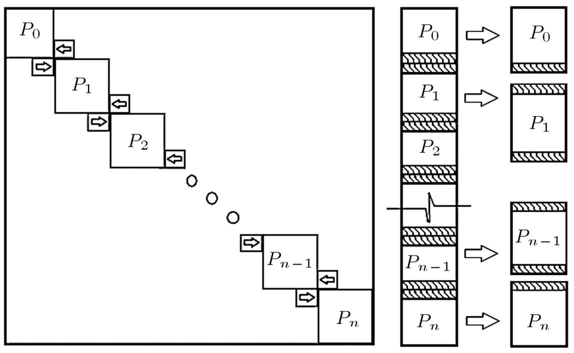

Figure 6. Sharing the multiplicand vector among processors using modified checkered decomposition.

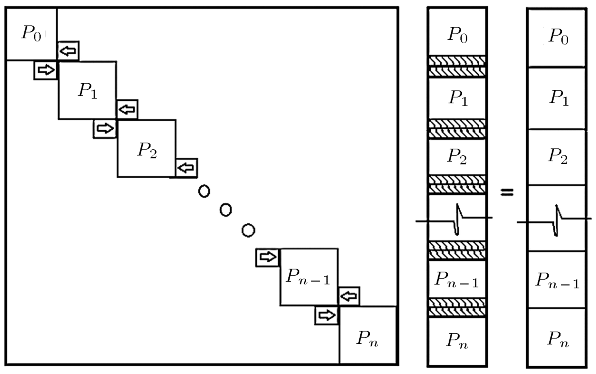

Figure 7. Multiplication using modified checkered decomposition.

7. Numerical results

7.1. Problems' specifications, solving method and employed system

In this section, the results of analyzing eight problems, solved using the developed parallel program, are presented. The goal is to calculate the first four eigenpairs of the beam in each problem using parallel processing and to perform its modal analysis. All beams are considered cantilever.

An example of such beams is shown in Figure 1. The problems are solved by the following methods:

- Method A: No matrix compression is used in this method. The matrices are shared among the processors using row-wise sharing method.
- Method B: No matrix compression is used in this method. The matrices are shared among the processors using modified checkered sharing method.
- Method C: CSR technique is used for matrix compression. The matrices are shared among the processors using row-wise sharing method.
- Method D: CSR technique is used for matrix compression. The matrices are shared among the processors using modified checkered sharing method.

The specifications of the materials of the beam in the problems are given in Tables 1 and 2.

7.2. Verification of the results

Before interpreting the amount of speedup caused by parallelization and in order to verify the results,

Table 1. Dimensional specifications and used materials.

Thickness (m)	W (m)	L (m)	ρ $(\frac{\text{kg}}{\text{m}^3})$	v	E (GPa)
0.025	0.2	2	7850	0.3	21

Table 2. Mesh specifications of the solved problems.

Problems	No. of elements	No. of DOF
A(612)	250	612
A(4832)	2250	4832
B(612)	250	612
B(4832)	2250	4832
C(8442)	4000	8442
C(51102)	25000	51102
D(8442)	4000	8442
D(51102)	25000	51102

the first four natural frequencies calculated using the parallel program are compared to the results of "spec" function of Scilab, which is an open source software for numerical computation. It should be noted that just the main matrix is used as the argument for "spec" function. The percentage of relative error, which is shown in Tables 3-7 by the term "error", is calculated using Eq. (13):

$$\text{Error} = \frac{v_{\text{Scilab}} - v_{\text{parallel}}}{v_{\text{Scilab}}} \times 100, \qquad (13)$$

Table 3. Verification of the results for problems A(612) and B(612).

Frequencies	Parallel code	Scilab	Error (%)
ω_{n1}	83.28	83.27	0.01
ω_{n2}	499.36	499.36	0.00
ω_{n3}	1286.31	1286.31	0.00
ω_{n4}	1314.90	1314.90	0.00

Table 4. Verification of the results for problems A(4832) and B(4832).

Frequencies	Parallel code	Scilab	Error (%)
ω_{n1}	82.55	82.55	0.00
ω_{n2}	495.18	495.18	0.00
ω_{n3}	1286.11	1286.11	0.00
ω_{n4}	1304.27	1304.28	0.00

Table 5. Verification of the results for problems C(8442) and D(8442).

Frequencies	Parallel code	Scilab	Error (%)
ω_{n1}	82.51	82.51	0.00
ω_{n2}	494.94	494.94	0.00
ω_{n3}	1286.65	1286.09	0.04
ω_{n4}	1303.10	1303.65	0.04

Table 6. Verification of the results for problems C(51102) and D(51102).

Frequencies	Parallel code	Scilab	Error (%)
ω_{n1}	82.48	82.46	0.02
ω_{n2}	494.66	494.66	0.00
ω_{n3}	1287.27	1286.06	0.09
ω_{n4}	1301.73	1302.93	0.09

Table 7. Verification of maximum displacement for the beam in problem D(51102).

Parallel code	Scilab	Error (%)
0.52	0.52	0.00

where V_{Scilab} is the outcome of Scilab and V_{parallel} is the result of the developed parallel program.

7.3. Speedup results

The speedup diagrams with respect to the solution method and degrees of freedom of each problem are depicted in Figures 8-11.

As can be seen in Figure 8, the speedup reduces drastically when the number of the involved processors increases from 7 to 8 in problem A(612). The reason for this reduction may be described by the use of two views: hardware concepts and parallelism theory.

From hardware point of view, it should be noted that the Intel-based CPUs, which have been used in this research, support Turbo Boost Technology. Turbo Boost Technology is a capability, which dynamically changes the processors' processing speed. As an example of how it works, when the process is executed using one core of CPU and there is no energy issue

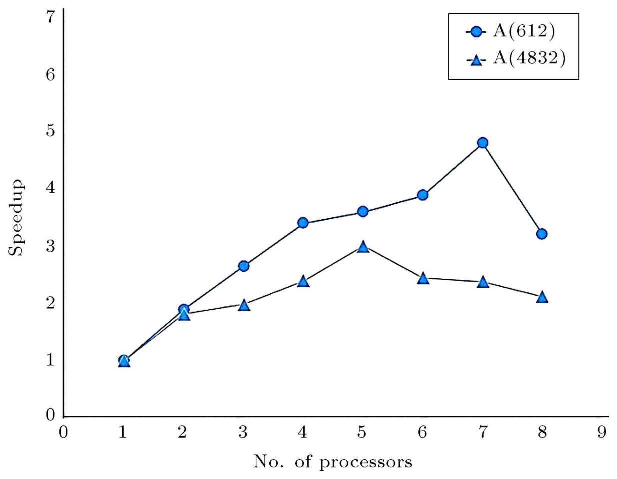

Figure 8. Speedup in Method A.

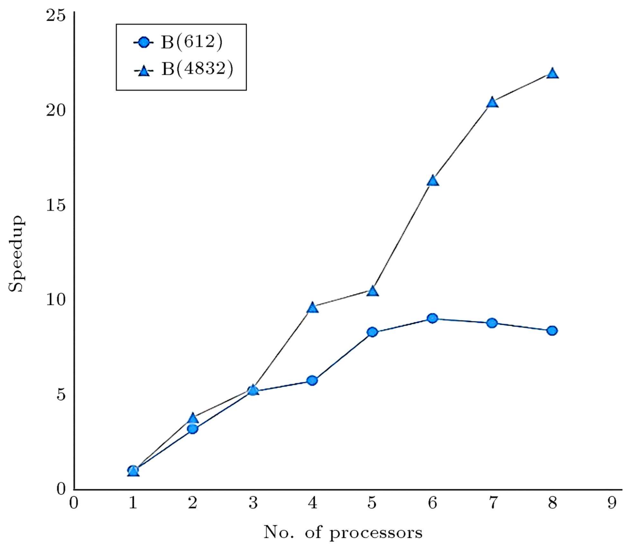

Figure 9. Speedup in Method B.

Table 8. Theoretical speedup of Method A (row-wise without matrix compression).

Execution strategy	**Code segments**	p_i	s_i	S
Serial	Stiffness and mass matrix generation	0.34	1.00	
	Gram-Schmidt orthogonalization	0.03	1.00	
	Built-in solver	0.03	1.00	6.84
Parallel	Parallel multiplication and subspace generation	0.77	8.00	
	Parallel PCG solver	98.83	7.00	

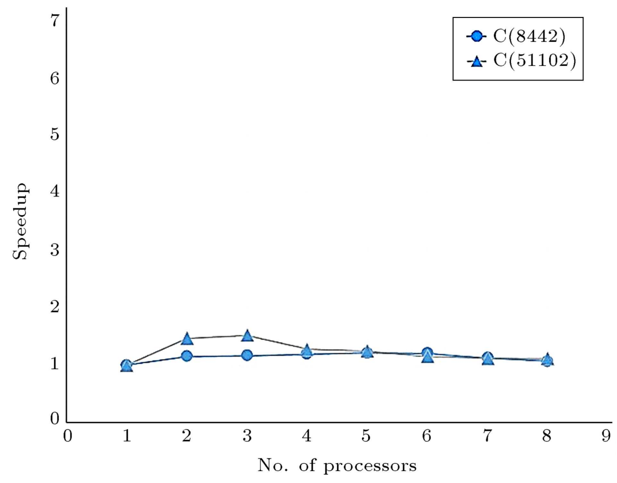

Figure 10. Speedup in Method C.

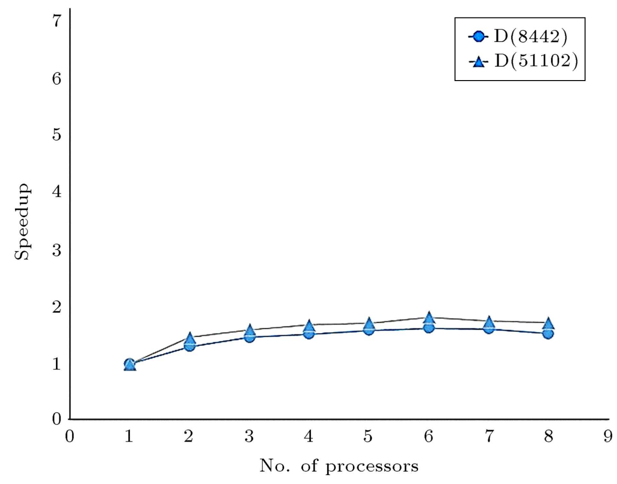

Figure 11. Speedup in Method D.

and thermal problem, the processing speed of that core reaches its maximum. If two cores are processing simultaneously, the speed of the cores increases, but each speed may be less than the potential maximum speed, which can be reached when a core works alone. In the same way, when more processors are involved, the speed of each core decreases to control the maximum allowable temperature. Generally, in processors with Turbo Boost capability, the frequency of each of the involved cores may be reduced when more processors get involved in order to deal with system overheat. In problem A(612), since the number of degrees of freedom is low, speedup limitation is mainly due to Turbo Boost and not the communication traffic among the processors. Therefore, Turbo Boost effect becomes more visible when more cores are involved in process. In other words, by increasing the number of processors from 7 to 8, Turbo boost has possibly reduced the frequency of the cores to avoid overheating. The Turbo Boost effect may be reduced in other examples with a larger number of degrees of freedom or larger data traffic because there are some time intervals whose cores are waiting to receive data from each other or RAM. Subsequently, the cooling system is able to control the heat: The more time is consumed in data traffic, the more idle time is assigned to each core resulting in cooler CPUs.

The amount of speedup for A(612) problem is presented in the next section. From Tables 8-11, it can be seen that A(612) is experiencing super linear speedup. From parallelism point of view, all points in the diagram of problem A(612), which are between one and eight cores, should not be compared with Amdahl's law or non-super linear conditions. In this case, there are some constant data and all cores need them. When the number of cores increases, the cache of CPU is divided into a larger number of CPUs. As a result, when the number of cores increases, the portion of each core decreases and they cannot contain all required data. Consequently, at the last point, with eight cores, all cores have to communicate via RAM resulting in a decreased but real speedup. To summarize, the speedup in the diagram of A(612) should not have increased in such a way that a sudden fall be detected in a non-super linear point (with eight cores).

7.4. The results of modal analysis

The response of the structure to various dynamic loads can be calculated by the computed eigenpairs. As an example, in problem D(51102), the beam displacement under the impact of a 50 N.s impulse, with persistence time of 0.05 s at 0.1 s after the hit, is questioned. This impulse is depicted in Figure 12.

It should be noted that the mentioned impulse is applied to the lower right corner of the beam in Figure 1. In Figure 13, the displacement of the beam under the impact of the impulse, at 0.1 s after

Table 9. Theoretical speedup of Method C (row-wise with matrix compression).

Execution strategy	Code segments	p_i	s_i	S
Serial	Stiffness and mass matrix generation	0.05	1.00	
	Gram-Schmidt orthogonalization	0.98	1.00	
	Built-in solver	1.07	1.00	2.88
Parallel	Parallel multiplication and subspace generation	0.44	4.00	
	Parallel PCG solver	97.46	3.00	

Table 10. Theoretical speedup of Method D (MCM with matrix compression).

Execution strategy	Code segments	p_i	s_i	S
Serial	Stiffness and mass matrix generation	0.07	1.00	
	Gram-Schmidt orthogonalization	0.65	1.00	
	Built-in solver	0.67	1.00	2.92
Parallel	Parallel multiplication and subspace generation	0.80	4.00	
	Parallel PCG solver	97.81	3.00	

Table 11. Comparing maximum theoretical and practical speedup values.

Approach	A	B	C	D
Practical	4.80	22.00	1.52	1.82
Theoretical (Amdahl's law)	6.84	N/A	2.88	2.92

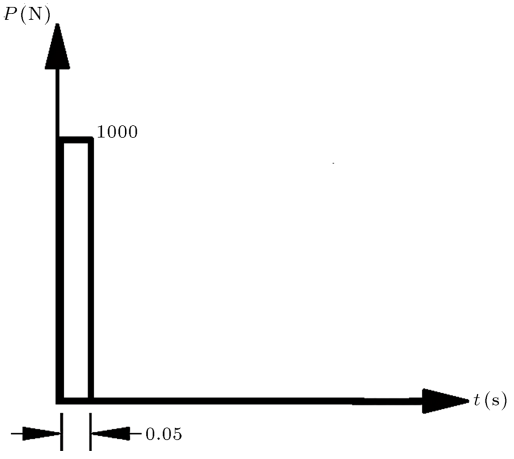

Figure 12. Graph of the applied impulse.

the impulse, is depicted. Additionally, the vertical displacement of this beam in evenly spaced points, with 0.5 m distance, is provided.

The results of the verification of the developed software are shown in Table 7. In this table, the vertical displacement of the impulse point is compared to the outcome of Scilab.

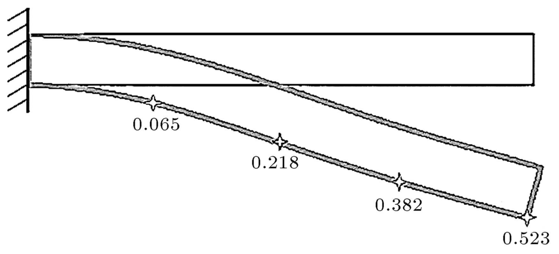

Figure 13. Deflection of the beam in problem D(51102) at 0.1 s after applying the impulse.

8. Interpreting the results

8.1. Comparing the results with Amdahl's law

According to Amdahl's law, the theoretical speedup is presented as follows:

$$S = \frac{1}{\frac{p_1}{s_1} + \frac{p_2}{s_2} + \frac{p_3}{s_3} + ...}, \tag{14}$$

where p_i is the percentage of the ith code segment, and the amount of speedup for the ith code segment is equal to s_i. Tables 8-10 summarize the execution times of serial and parallel code segments in terms of relative parameters p_i and s_i. According to these values, the theoretical speedup is computed and presented in the tables.

It is important to emphasize that Method B does not use matrix compression and contains relatively small degrees of freedom. Therefore, it is likely that super linear speedup takes place in this case. Therefore, comparing the results with those of theoretical Amdahl's law may lead to incorrect results.

Table 11 summarizes the maximum theoretical and practical speedups for applicable methods.

As can be seen in Table 11, theoretical speedup is larger than its practical counterpart. Amdahl's law ignores all time intervals which are consumed for sending and receiving messages and data among all processors, real traffic loads, mandatory operating system services which increase system overhead, etc. Therefore, the theoretical speedup is never achieved unless a super linear speedup has been occurred.

In addition, Tables 8-10 show that the difference between theoretical and practical speedups using Method A is larger compared to those obtained by Methods B and C. This indicates that in Method A, the processors consume more time to send and receive messages to or from each other in comparison to Methods C and D. Here, practical measurements indicate that Amdahl's law may be unable to predict the real speedup because of its assumptions which are made for simplifying the formula.

8.2. Discussion

First, the precision of the natural frequencies and displacement calculated using parallel modal analysis could be noted. According to the percentages of relative error in Tables 3-7, the results are calculated with good precision.

Considering the speedup of the analyses using Method A, it is noteworthy that according to the represented graphs, by increasing the number of degrees of freedom, the speedup becomes limited and the maximum speedup occurs when using 5 and 7 processors. The reasoning is somehow related to the way the multiplicand vectors are divided among the processors. Since the mentioned vectors should be sent to all of the processors completely in row-wise sharing method, bigger problems will yield bigger dimension for the shared vector. As a result, in large-scale problems, communication traffic among processors will become significant, which leads to a limitation in the speedup. On the other hand, by increasing the involved processors, the number of the vectors that should be transferred grow; thus, maximum speedup occurs in a small number of processors. According to the presented graphs, by using Method B, the speedup moves beyond the ideal speedup limit of parallel processing. It happens because the resulted speedup is related not only to parallelization, but also to the modified checkered sharing method of the main matrix of the problem and also due to ignoring blocks with zero elements which are far from the main diagonal. The advantage of this method over Method A is that there is no need to send the complete multiplicand vectors to all of the processors. In fact, each processor receives only a share of the multiplicand vector, which is assigned to its block. However, a part of these vectors will have overlap due to the nonzero elements outside the main diagonal blocks. This overlap could be limited by generating a proper mesh of finite elements. Therefore, during the analysis, the grid of the processors will not be congested by vectors of main dimension, and only one vector with a small amount of overlap is distributed among the processors. As a result, despite the analysis using Method A, by adding degrees of freedom and enlarging the problem, the speedup increases. As it can be seen in Figures 10 and 11, the speedup of Methods D and C is lower than that of Methods A and B. The main reason for this is performing matrix compression. By performing matrix compression, computational effort and required data increase significantly. The cause of speedup rate reduction in the case of using Method C can be explained by referring to what was mentioned about Method A. In Method C, the complete multiplicand vector is distributed among processors. Consequently, in large-scale problems, the data traffic among processors will be increased and the speedup will be limited. On the other hand, the speedup of Method D is higher than that of Method C. It seems to happen due to using the Modified Checkered method and dividing the multiplicand vector among processors. Therefore, the communication grid will be less crowded in Method D compared to Method C, and the speedup will increase. However, in Methods C and D, the essence of the problem does not change by increasing the number of the processors. Therefore, all of the resulting speedup is due to parallelization, although the amount of increase in the speedup is low. More accurately, in Method D, the required time to analyze the fine mesh decreases approximately by 45% when the number of processors increases; nevertheless, by increasing the number of processors, the speedup does not significantly change in both fine and course meshes. A similar conclusion may be drawn from Figure 10 for Method C. In fact, the results show that Methods C and D do not have significant capability of being parallelized, at least by the use of the presented algorithms.

Moreover, in Tables 12 and 13, the percentages of maximum decrease of processing time in problems

Table 12. Maximum decrease of processing time in problems solved using Methods A and B.

Method	**A**	**B**	**A**	**B**
No. of DOF	612	612	4832	4832
Maximum decrease of processing time	79.20%	88.92%	66.77%	95.45%

Table 13. Maximum decrease of processing time in problems solved using Methods C and D.

Method	**C**	**D**	**C**	**D**
No. of DOF	18662	18662	51102	51102
Maximum decrease of processing time	19.10%	33.34%	34.02%	45.35%

without compression (A and B) and problems with compression (C and D) are compared to each other. Based on the mentioned results, it can be concluded that regardless of using or not using matrix compression, modified checkered sharing method, proposed in this paper, results in a high amount of decrease in the processing time. The reason for this advantage is the decrease of the data traffic among the processors in the grid. In fact, despite row-wise sharing method, in modified checkered method, the multiplicand vector is not completely shared among all of the processors. Another advantage of the modified checkered method is its better use of the available storage without employing matrix compression methods.

9. Conclusion

In the present study, the problem of modal analysis of two-dimensional beams is performed using parallel processing. The importance of the problem is due to its application in the design of structures, especially when the intrinsic dynamic characteristics of the model change in each step of a trial-and-error design manner, and it is required to have access to the accurate eigenpairs in each iteration. In such a situation, a modal analysis may be required to be performed several times. Logically, it would be very efficient if the computation time of a modal analysis could be reduced. As mentioned, Davidson method has been used in order to implement the parallel program. In general, two main types of approaches have been implemented. The first type does not compress the main matrix of the standard form of the eigenvalue problem, whereas the second type uses CSR matrix compression. Each of the above-mentioned types is divided into two versions. The first version uses row-wise sharing method, while the second version implements a modified checkered sharing model, which has been proposed in this paper. The accuracy of all types and corresponding versions has been verified using Scilab built-in functions, which is an open-source software for numerical computation. The results of this study indicate that the combination of modified checkered method and parallel processing could be used as a powerful tool for high performance modal analysis. Although the presented method has been applied only to a cantilever beam, the developed finite-element source code is capable of being applied to a wide range of two-dimensional problems. Nevertheless, in the present study, the domain has been discretized using a structured mesh.

References

1. Garinei, A. "Vibrations of simple beam-like modelled bridge under harmonic moving loads", *International Journal of Engineering Science*, **44**(11-12), pp. 778-787 (2006).

2. Bozdaga, E., Sunbuloglua, E. and Ersoy, H. "Vibration analysis of new Galata bridge experimental and numerical results", *Journal of Computers & Structures,* **84**(5-6), pp. 283-292 (2006).

3. Heng, B.C.P. and Mackie, R.I. "Parallel modal analysis with concurrent distributed objects", *Journal of Computers & Structures*, **88**(23-24), pp. 1444-1458 (2010).

4. Subramanian, C., Van Criekingen, S., Heuveline, V., Nataf, F. and Havé, P. "The Davidson method as an alternative to power iterations for criticality calculations", *Journal of Annals of Nuclear Energy*, **38**(12), pp. 2818-2823 (2011).

5. Nefedev, K.V. and Peretyatko, A.A. "Superlinear speedup of parallel calculation of finite number Ising spins partition function", *3th International Conference on High Performance Computing*, **1**, Kyiv, Ukraine, pp. 231-243 (2013).

6. Baker, M. and Buyyaz, R. "Cluster computing at a glance", in *Computer Science*, 1th Edn., pp. 9-13, University of Portsmouth Press, Southsea, Hants, UK (2006).

7. Al-Rayes, H.T. "Studying main differences between linux & windows operating systems", *International Journal of Electrical and Computer Engineering*, **12**(4), pp. 25-31 (2004).

8. Chopra, A.K. "Dynamics of structures: Theory and applications to earthquake engineering", in *Civil Engineering and Engineering Mechanics*, 4th Edn., pp. 440-478, University of California Press, Berkeley, California, USA (2004).

9. Chandrupatla, T.R. and Belegundu, A.D. "Introduction to Finite Element Methods in Engineering", in *Mathematics and Engineering,* 4th Edn., pp. 113-121, Pennsylvania State University Press, Pennsylvania, USA (2012).

10. Batista, V.H.F., Ainsworth, G.O. and Ribeiro, F.L.B. "Parallel structurally-symmetric sparse matrix-vector products on multi-core processors", *3th International*

Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, **1**, Pécs, Hungary, pp. 1-22 (2013).

11. Feng, Y.T. "An integrated Davidson and multigrid solution approach for very large scale symmetric eigenvalue problems", *Journal of Computer Methods in Applied Mechanics and Engineering*, **190**(28), pp. 3543-3563 (2001).

12. Giménez, D., Jiménez, C., Majado, M.J., Marín, N. and Martín, A. "Solving eigenvalue problems on networks of processors", *Lecture Notes in Computer Science (LNCS)*, **1573**, pp. 58-99 (1999).

13. Xuanhua, F., Pu, C., Ruian, W. and Shifu, X. "Parallel computing study for the large-scale generalized eigenvalue problems in modal analysis", *Journal of Mechanics and Astronomy*, **57**(3), pp. 477-489 (2014).

Biographies

Soroush Heydari was born in Ahwaz, Iran in 1990. He earned his BSc (2012) from Shahid Chamran University, Ahwaz, Iran, and MSc (2014) from K. N. Toosi University of Technology, Tehran, Iran. He is interested in Parallel Programming and accomplished his thesis on Parallel Finite Element Method.

Saeed Asil Gharebaghi was born in Tehran, Iran, in 1973. He earned his BSc (1996), MSc (1998), and PhD (2007) from Sharif University of Technology, Tehran, Iran. The applications of mathematics in Civil Engineering as well as numerical simulations using a wide variety of software products are his interests. He is now a faculty member of Civil Engineering Department in K. N. Toosi University of Technology, Tehran, Iran.