Efficient sparse matrix-matrix multiplication for computing periodic responses by shooting method on Intel Xeon Phi

S. Stoykov, E. Atanassov, and S. Margenov

Citation: AIP Conference Proceedings **1773**, 110012 (2016); doi: 10.1063/1.4965016
View online: http://dx.doi.org/10.1063/1.4965016
View Table of Contents: http://aip.scitation.org/toc/apc/1773/1
Published by the American Institute of Physics

Efficient Sparse Matrix-matrix Multiplication for Computing Periodic Responses by Shooting Method on Intel Xeon Phi

S. Stoykov[a], E. Atanassov[b] and S. Margenov[c]

Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Acad. G. Bonchev str., bl. 25A, 1113 Sofia, Bulgaria

[a] Corresponding author: stoykov@parallel.bas.bg
[b]emanouil@parallel.bas.bg
[c]margenov@parallel.bas.bg

Abstract. Many of the scientific applications involve sparse or dense matrix operations, such as solving linear systems, matrix-matrix products, eigensolvers, etc. In what concerns structural nonlinear dynamics, the computations of periodic responses and the determination of stability of the solution are of primary interest. Shooting method iswidely used for obtaining periodic responses of nonlinear systems. The method involves simultaneously operations with sparse and dense matrices. One of the computationally expensive operations in the method is multiplication of sparse by dense matrices. In the current work, a new algorithm for sparse matrix by dense matrix products is presented. The algorithm takes into account the structure of the sparse matrix, which is obtained by space discretization of the nonlinear Mindlin's plate equation of motion by the finite element method. The algorithm is developed to use the vector engine of Intel Xeon Phi coprocessors. It is compared with the standard sparse matrix by dense matrix algorithm and the one developed by Intel MKL and it is shown that by considering the properties of the sparse matrix better algorithms can be developed.

INTRODUCTION

The computation of periodic responses of elastic structures and its parametric study in frequency domain are of primary interest for engineers and researchers. Special attention is taken into account when the equation of motion of the elastic structureis nonlinear. Due to the nonlinearities, bifurcation points may appear, the shape of vibration can change, the period of vibration can become double or the motion can become quasi-periodic or chaotic [1]. Apart from these complexities which arise from the nonlinearities, the computation of periodic responses is time consuming process when it is applied to large-scale models. One of the computationally expensive parts of the algorithm is the multiplication of matrices. The aim of the current work is to develop and present efficient algorithms for multiplication of sparse matrix by dense matrix and by taking into account the specific structure of the sparse matrix, to achieve better performance than the algorithm from Intel MKL library. From here on, everywhere in the paper where matrix-matrix product is written, it will be assumed multiplication of sparse matrix by dense matrix.

The shooting method is applied to the equation of motion of plates based on Mindlin's theory [2]. Geometrical type of nonlinearity is introduced in the model, *i.e.*, nonlinear strain-displacement relations are considered. The equation of motion is discretized by the finite element method. Quadrilateral finite elements are used with bilinear shape functions. These elements are appropriate for plates with complex geometries and to avoid shear locking, reduced and selective integration is performed to the terms related with the shear strain energy. The resulting system of nonlinear ordinary differential equations is solved by shooting method, where several matrix-matrix multiplicationsare involved in each shooting iteration step. The presented algorithm takes into account the structure of the sparse matrices. A comparison between the proposed algorithm with standard matrix-matrix multiplication

Application of Mathematics in Technical and Natural Sciences
AIP Conf. Proc. 1773, 110012-1–110012-7; doi: 10.1063/1.4965016
Published by AIP Publishing. 978-0-7354-1431-0/$30.00

algorithm and withIntel MKL library is presented and the efficiency of the proposed algorithm is outlined. The experiments are performed on one Xeon Phi coprocessor and improvement is achieved by increasing the number of the processes.

The paper is organized in the following way: in the next section is presented the equation of motion of Mindlin's plate and its discretization by the finite element method. The shooting method is also presented by outlining the parts where matrix-matrix multiplications appear. Then, three algorithms for matrix-matrix multiplications are presented and their performance on Intel Xeon Phi is analyzed.

MATHEMATICAL MODEL

The equation of motion of plate, based on Mindlin's theory and considering geometrical type of nonlinearity is presented in this section. The plate is assumed to be of isotropic material with constant thickness denoted by h. The equation is derived on the middle plane of the plate, *i.e.*, for $z = 0$ (Figure 1). The primary unknowns are the transverse displacement denoted by w_0 and the rotations of the middle plane about the axes x and y, denoted by ϕ_x and ϕ_y, respectively.

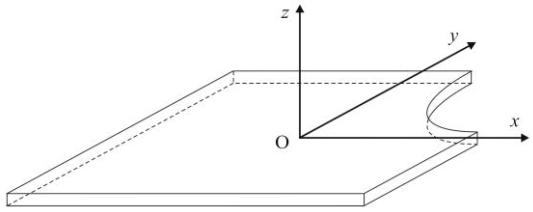

FIGURE1. Plate with complex geometry

The equation of motion is given by [2]:

$$\begin{aligned} \frac{\partial Q_x}{\partial x} + \frac{\partial Q_y}{\partial y} + N(w_0, \phi_x, \phi_y) + q &= \rho h \frac{\partial^2 w_0}{\partial t^2}, \\ \frac{\partial M_x}{\partial x} + \frac{\partial M_{xy}}{\partial y} - Q_x &= \rho \frac{h^3}{12} \frac{\partial^2 \phi_x}{\partial t^2}, \\ \frac{\partial M_{xy}}{\partial x} + \frac{\partial M_y}{\partial y} - Q_y &= \rho \frac{h^3}{12} \frac{\partial^2 \phi_y}{\partial t^2}, \end{aligned} \tag{1}$$

where $\{M_x, M_y, M_{xy}\}$ are the moment resultants:

$$\{M_x, M_y, M_{xy}\} = \int_{-\frac{h}{2}}^{\frac{h}{2}} \{\sigma_x z, \sigma_y z, \tau_{xy} z\} dz, \tag{2}$$

$\{Q_x, Q_y\}$ are the shear stress resultants:

$$\{Q_x, Q_y\} = \lambda \int_{-\frac{h}{2}}^{\frac{h}{2}} \{\tau_{xz}, \tau_{yz}\} dz, \tag{3}$$

λ is the shear correction factor. The nonlinear terms are represented by $N(w_0, \phi_x, \phi_y)$:

$$N(w_0, \phi_x, \phi_y) = \frac{\partial}{\partial x}\left(N_x \frac{\partial w_0}{\partial x} + N_{xy} \frac{\partial w_0}{\partial y}\right) + \frac{\partial}{\partial y}\left(N_{xy} \frac{\partial w_0}{\partial x} + N_y \frac{\partial w_0}{\partial y}\right), \tag{4}$$

where $\{N_x, N_y, N_{xy}\}$ are the in-plane stress resultants:

$$\{N_x, N_y, N_{xy}\} = \int_{-\frac{h}{2}}^{\frac{h}{2}} \{\sigma_x, \sigma_y, \tau_{xy}\} dz. \tag{5}$$

Quantity $q = q(x, y, t)$ is the force vector on the middle plane in transverse direction.σ_x and σ_y are the direct stresses and τ_{xy}, τ_{xz} and τ_{yz} are the shear stresses. They are related to the strains by Hooke's law.

The partial differential equations (PDE) of motion, Eq.(1), are discretized by the finite element method. Quadrilateral finite elements are used. These elements allow one to discretize plates with complex geometries. Bilinear functions are considered for the transverse displacement and the rotations. A square element with local coordinates ξ and η is used as a reference element, $\xi, \eta \in [-1, 1]$. The element has four nodes at the corners and

each node has three degrees of freedom, the transverse displacement w_0 and the rotations of the middle plane ϕ_x and ϕ_y.To avoid shear locking, reduced and selective integration (RSI) is used for the terms related to the shear forces.The PDEis discretized by the finite element method and the following system of second order ordinary differential equations (ODE) is obtained:

$$\mathbf{M}\ddot{\mathbf{q}}(t) + \mathbf{C}\dot{\mathbf{q}}(t) + \mathbf{K}_{\mathrm{l}}\mathbf{q}(t) + \mathbf{K}_{\mathrm{nl}}\big(\mathbf{q}(t)\big)\mathbf{q}(t) = \mathbf{a}\cos(\omega t), \tag{6}$$

with initial conditions:

$$\begin{aligned} \mathbf{q}(0) &= \mathbf{q}_0, \\ \dot{\mathbf{q}}(0) &= \dot{\mathbf{q}}_0, \end{aligned} \tag{7}$$

where $\mathbf{q}(t)$ represents the vector of generalized coordinates,$\mathbf{M}$ represents the mass matrix, $\mathbf{C}$ – the damping matrix, $\mathbf{K}_{\mathrm{l}}$ – the stiffness matrix of constant terms, $\mathbf{K}_{\mathrm{nl}}\big(\mathbf{q}(t)\big)$ – the stiffness matrix that depends on the vector $\mathbf{q}(t)$, it introduces cubic nonlinearity in the equation of motion, harmonic forces are applied in transverse direction with excitation frequency ω and $\mathbf{a}$ is the generalized vector of external forces.

The shooting method finds the initial conditions that lead to periodic steady-state response, without any transient motion. The method finds corrections $\delta\mathbf{q}_0$ and $\delta\dot{\mathbf{q}}_0$of the initial conditions $\mathbf{q}_0$ and $\dot{\mathbf{q}}_0$, such that the response of the system is equal to the initial conditions after one period of vibration T:

$$\begin{aligned} \mathbf{q}(T, \mathbf{q}_0 + \delta\mathbf{q}_0, \dot{\mathbf{q}}_0 + \delta\dot{\mathbf{q}}_0) &= \mathbf{q}_0 + \delta\mathbf{q}_0, \\ \dot{\mathbf{q}}(T, \mathbf{q}_0 + \delta\mathbf{q}_0, \dot{\mathbf{q}}_0 + \delta\dot{\mathbf{q}}_0) &= \dot{\mathbf{q}}_0 + \delta\dot{\mathbf{q}}_0. \end{aligned} \tag{8}$$

The period of vibration is assumed to be known and related to the excitation frequency ω. Details of the shooting method are given in [3]. The method performs a time integration for one period of time, and at the end of the time integration it defines a linear system which solution gives the corrections of the initial conditions, *i.e.*, $\delta\mathbf{q}_0$ and $\delta\dot{\mathbf{q}}_0$. The method uses Newmark's time integration scheme for solving the initial value problem, and Newton-Raphson's method for solving the nonlinear problem. A pseudo-code of the shooting method is given here:

Pseudo-code of time integration part of shooting method, a_0, a_1, a_2, a_3, a_4, a_5 are the constants of Newmark's method.

Define the following matrices:

$$\begin{aligned} \mathbf{RM}_1 &= a_0\mathbf{M} + a_1\mathbf{C} \\ \mathbf{RM}_2 &= a_2\mathbf{M} + a_4\mathbf{C} \\ \mathbf{RM}_3 &= a_3\mathbf{M} + a_5\mathbf{C}. \end{aligned}$$

For each time integration step $t + \Delta t$ from 0 to T do (Newmark's time integration scheme)

1. Solve the nonlinear system

$$\big(a_0\mathbf{M} + a_1\mathbf{C} + \mathbf{K}_{\mathrm{l}} + \mathbf{K}_{\mathrm{nl}}(\mathbf{q}_{t+\Delta t})\big)\mathbf{q}_{t+\Delta t} = \mathbf{F}_{t+\Delta t} + \mathbf{RM}_1\mathbf{q}_t + \mathbf{RM}_2\dot{\mathbf{q}}_t + \mathbf{RM}_3\ddot{\mathbf{q}}_t$$

by Newton-Raphson's method:

use the solution from the previous step as initial guess for the nonlinear solution of the current step:

$$\mathbf{q}_{t+\Delta t} = \mathbf{q}_t$$

while ($\|\mathbf{r}\| > \varepsilon_1$ and $\|\Delta\mathbf{q}\| > \varepsilon_2$) do

compute the nonlinear matrix $\mathbf{K}_{\mathrm{nl}}(\mathbf{q}_{t+\Delta t})$ and the Jacobian $\mathbf{J}(\mathbf{q}_{t+\Delta t})$

solve the following linear equation:

$$\big(a_0\mathbf{M} + a_1\mathbf{C} + \mathbf{K}_{\mathrm{l}} + \mathbf{J}(\mathbf{q}_{t+\Delta t})\big)\Delta\mathbf{q} =$$
$$= \mathbf{F}_{t+\Delta t} + \mathbf{RM}_1\mathbf{q}_t + \mathbf{RM}_2\dot{\mathbf{q}}_t + \mathbf{RM}_3\ddot{\mathbf{q}}_t - \big(a_0\mathbf{M} + a_1\mathbf{C} + \mathbf{K}_{\mathrm{l}} + \mathbf{K}_{\mathrm{nl}}(\mathbf{q}_{t+\Delta t})\big)\mathbf{q}_{t+\Delta t}$$

update the solution:

$$\mathbf{q}_{t+\Delta t} = \mathbf{q}_{t+\Delta t} + \Delta\mathbf{q}$$

compute $\dot{\mathbf{q}}_{t+\Delta t}and\ddot{\mathbf{q}}_{t+\Delta t}$

compute the residual $\mathbf{r}$:

$$\mathbf{r} = \mathbf{M}\ddot{\mathbf{q}}_{t+\Delta t} + \mathbf{C}\dot{\mathbf{q}}_{t+\Delta t} + \mathbf{K}_{\mathrm{l}} + \mathbf{K}_{\mathrm{nl}}(\mathbf{q}_{t+\Delta t}) - \mathbf{F}_{t+\Delta t}$$

end do

2. Compute the following matrix-matrix products

$$\begin{aligned} \mathbf{R_D} &= \mathbf{RM}_1\mathbf{Q}_t + \mathbf{RM}_2\mathbf{DQ}_t + \mathbf{RM}_3\mathbf{DDQ}_t \\ \mathbf{R_V} &= \mathbf{RM}_1\mathbf{QD}_t + \mathbf{RM}_2\mathbf{DQD}_t + \mathbf{RM}_3\mathbf{DDQD}_t \end{aligned}$$

3. Solve the following systems

$$\begin{aligned} \Big(a_0\mathbf{M} + a_1\mathbf{C} + \mathbf{K}_{\mathrm{l}} + \mathbf{J}\big(\mathbf{q}(t+\Delta t)\big)\Big)\,\mathbf{Q}_{t+\Delta t} &= \mathbf{R_D} \\ \Big(a_0\mathbf{M} + a_1\mathbf{C} + \mathbf{K}_{\mathrm{l}} + \mathbf{J}\big(\mathbf{q}(t+\Delta t)\big)\Big)\,\mathbf{QD}_{t+\Delta t} &= \mathbf{R_V} \end{aligned}$$

Compute $\mathbf{DQ}_{t+\Delta t}$, $\mathbf{DDQ}_{t+\Delta t}$, $\mathbf{DQD}_{t+\Delta t}$and $\mathbf{DDQD}_{t+\Delta t}$

End of time integration

SPARSE MATRIX-MATRIX MULTIPLICATION

The matrix-matrix products are performed in step 2. Numerical experiments are carried out to analyze what part of the total CPU time of the algorithm take the matrix-matrix products. It wasfound that the matrix-matrix products take more than 20% of the computational time of the shooting method on sequential code. The reduction of the computational time of the matrix-matrix products will influence respectively on the computational time of the whole algorithm.

The presented algorithms of matrix-matrix products are implemented considering one sparse matrix denoted by $\mathbf{RM}_1$. The sparse matrices are presented in compressed sparse row (CSR) format. The array $\mathbf{RM}_1$ contains the nonzero values of the sparse matrix in row by row order, the array **AI** is of integer type and it has the column indices of the elements of $\mathbf{RM}_1$, the array **AP** is of integer type and contains pointers to the beginning of each row in the arrays of $\mathbf{RM}_1$ and **AI**. The dense matrix is denoted by $\mathbf{Q}_t$ and the resulting matrix from the multiplication matrix is denoted by $\mathbf{R_D}$. Matrices **M** and **C**, which compose the matrix $\mathbf{RM}_1$, are the mass and the damping matrices from Eq. (6). The mass, the damping, the linear and nonlinear stiffness matrices and the Jacobian have the same structure. Thus, for all these matrices, the arrays **AI** and **AP** are common.

The numerical computations are carried out on Avitohol supercomputer [4], located at IICT-BAS. The supercomputer has 300 CPUs (Intel Xeon E5-2650 v2 8C 2600 GHz) organized on 150 servers with non-blocking InfiniBand FDR and 64 GB RAM. Each processor has 8 CPU cores with hyper-threading. Each server has two Intel Xeon Phi 7120P coprocessors, each with 61 coresat 1.238GHz with 4-way Intel hyper-threading and 8 GB RAM. The presented algorithms for matrix-matrix multiplications are run on the CPUs and on the coprocessors by using shared memory parallel realization by OpenMP.

The size of the sparse matrices is denoted by N. The dense matrices are also with dimension $N\times N$, but the previously derived parallel realization of the method, spreads the dense matrices in blocks of vector columns among the available processes by MPI. Thus, the algorithm is presented to multiply sparse $N\times N$ matrix by dense $N\times M$ matrix. The standard algorithm for multiplication of sparse matrix by dense matrix is given here [5]:

ALGORITHM 1. Sparse matrix by dense matrix multiplication from [5].

```
for (j = 0; j < N; j++){
  for (i = 0; i < M; i++){
    k1 = AP[j];
    k2 = AP[j+1];
    for (k = k1; k < k2; k++){
      R_D[j*M + i] += RM_1[k] * Q_t[M*(AI[k]) + i];
    }
  }
}
```

The parallel implementation is achieved by OpenMP on the external loop. The code is run on the CPUs and on the Intel Xeon Phi coprocessors. Meshes with different sizes are used. The results of the algorithm are summarized in Tables 1-4. Because the algorithm involves dense matrix and because the experiments are run on one coprocessor with 16 GB RAM, one cannot do experiments with very large matrices because of lack of memory.

TABLE 1. Computational time(s) of the algorithms on CPU by OpenMP. Mesh with $N = 14259$, $M = 14259$.

Number of processes	8	16	32
ALGORITHM 1	0.8494	0.7763	2.2047
ALGORITHM 2	0.6047	0.7149	0.9330
ALGORITHM 3	0.5619	0.3873	0.3994
MKL	0.7182	0.9029	0.9098

TABLE 2. Computational time(s) of the algorithms on Intel Xeon Phi by OpenMP. Mesh with $N = 14259$, $M = 14259$.

Number of processes	30	61	122	183	244
ALGORITHM 1	9.1947	4.6724	2.5483	1.7851	1.5055
ALGORITHM 2	0.4392	0.3082	0.3137	0.7855	0.8363
ALGORITHM 3	0.3392	0.1755	0.6760	0.7374	0.7500
MKL	0.4914	0.3239	0.3243	0.8141	0.8745

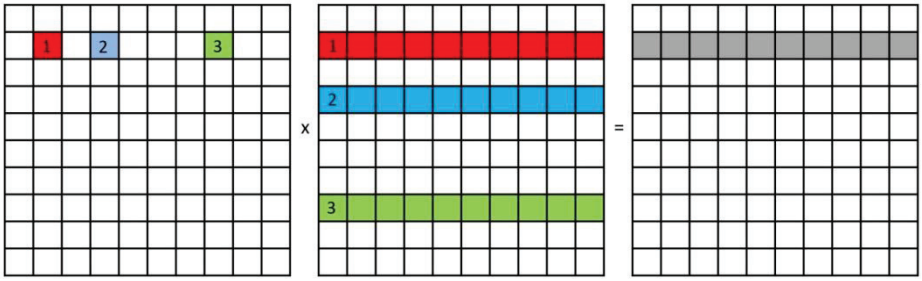

FIGURE 2. Sparse matrix by dense matrix multiplication

Algorithm 1 is improved and adapted for vectorization, in order to achieve the advantages of the Intel Xeon Phi coprocessors.For that purpose, the matrix-matrix multiplication is achieved on the following way: the elements of a given row from the sparse matrix are multiplied by all elements of the corresponding row of the dense matrix. The results are accumulated in the corresponding row of the resulting dense matrix. The procedure is shown in Figure 2. In this way, the multiplication of the elements and the summationis achieved by using single instruction, multiple data (SIMD) on the Intel Xeon Phi. The vector engine supports 512-bit vector, which means that 8 double precision numbers are processed simultaneously. The proposed algorithm is denoted as Algorithm 2 and it is shown as pseudo-code bellow:

ALGORITHM 2. Sparse matrix by dense matrix multiplication.

```
for (j = 0; j < N; j++){
  k1 = AP[j];
  k2 = AP[j+1];
    for (k = k1; k < k2; k++){
      for (i = 0; i < M; i++){
        R_D[j*M+i] += RM_1[k] * Q_t[M*AI[k] + i];
      }
    }
}
```

The dense matrices are aligned to 64-byte address boundaries, which additionally increase the performance on Intel Xeon Phi coprocessors. Different strategies for alignments were carried out, like organizing the sparse data in the rows of blocks of 32, and it was concluded that there is no significant improvement of the performance of the algorithm. The most essential part is the organization of the memory access within the iterations of the matrix-matrix product and the storage of the data in the cache.

TABLE 3. Computational time(s) of the algorithms on CPU by OpenMP. Mesh with $N = 56163$, $M = 4096$.

Number of processes	8	16	32
ALGORITHM 1	2.3719	1.2356	1.3280
ALGORITHM 2	0.5696	0.3470	0.4145
ALGORITHM 3	0.4831	0.2977	0.3627
MKL	0.6764	0.4451	0.4431

TABLE 4. Computational time(s) of the algorithms on Intel Xeon Phi by OpenMP. Mesh with $N = 56163$, $M = 4096$.

Number of processes	30	61	122	183	244
ALGORITHM 1	62.7641	32.1431	16.6409	11.3927	8.7824
ALGORITHM 2	0.4603	0.2982	0.2826	0.2853	0.3036
ALGORITHM 3	0.3293	0.1664	0.1361	0.1822	0.4209
MKL	0.5469	0.3713	0.3174	0.3157	0.3285

The algorithm can be further improved by using the structure of the sparse matrix for the specific problem. Because the plate has three degrees of freedom per node, the structure of the rows of the sparse matrix is organized in groups of three, *i.e.*, each three rows have the same positions of nonzero elements. This knowledge about the structure of the sparse matrices can be used and the access to the arrays **AP** and **AI** can be reduced by three. The code that uses Algorithm 2 and the multiplication illustrated in Figure 2, but with reduced access to the arrays **AP** and **AI** is presented as Algorithm 3. On this way the computational time of this algorithm is reduces, as it is demonstrated by the results from Tables 1-4.

ALGORITHM 3. Sparse matrix by dense matrix multiplication.

```
for (j = 0; j < N/3; j++){
  k1 = AP[3*j];
  k2 = AP[3*j+1];
    for (k = k1; k < k2; k++){
      for (i = 0; i < M; i++){
        R_D[3*j*M+i] += RM_1[k] * Q_t[M*AI[k] + i];
        R_D[(3*j+1)*M + i] += RM_1[k+k2-k1] * Q_t[M*AI[k] + i];
        R_D[(3*j+2)*M + i] += RM_1[k+2*( k2-k1)] * Q_t[M*AI[k] + i];
      }
    }
}
```

The presented algorithms are compared with mkl_dcsrmm routine from Intel MKL library [6] which computes matrix-matrix product of sparse matrix stored in CSR format. The results show that the MKL library behaves well, but the proposed algorithm which takes into account the structure of the sparse matrices is faster than the one from the Intel MKL library.

CONCLUSIONS

An algorithm for matrix-matrix products is presented and applied to the shooting method for computing periodic responses of plates. The algorithm is developed to use the 512-bit SIMD registers of Intel Xeon Phi coprocessors. Additional improvementis achieved by taking into account the structure of the sparse matrix. It is shown that the proposed algorithm has better performance on the coprocessors than on the CPU and it behaves better than the Intel MKL library. The proposed algorithm has an improvement of about 45% over the computational time of the MKL library.

One of the most time consuming operations in the matrix-matrix multiplication are the access to the memory and the storage of the new data. Future improvements of the algorithm will be considered by optimization of the memory access. The load request data for the multiplication will be organized to stay in L1 memory of the coprocessors and to be reused for all necessary operations and only then new data will be fetched into the cache. In this way the transfer of data between the cache and the main memory will be reduced and better performance will be achieved.

ACKNOWLEDGMENTS

The research work carried out in the paper is partially supported through the program for young scientists in the Bulgarian Academy of Sciences, grant No. DFNP-93/04.05.2016 and through the project VI-SEEM (Contract No. 675121) supported by the European Commission H2020 program.

REFERENCES

1. A. Nayfeh and B. Balachandran, *Applied Nonlinear Dynamics: Analytical, Computational and Experimental Methods* (John Wiley & Sons, New York, 1995).
2. J. N. Reddy, *Mechanics of Laminated Composite Plates and Shells, Theory and Analysis* (CRC Press, 2004).
3. S. Stoykov and S. Margenov (2014) *Computers and Mathematics with Applications* **67**, 2257-2267.
4. Supercomputer System Avitohol at IICT-BAS, http://www.iict.bas.bg/avitohol (last time visited 31.07.2016).
5. Y. Saad, *Iterative Methods for Sparse Linear Systems* (SIAM, Philadelphia, 2003).
6. Intel® Math Kernel Library, http://software.intel.com/intel-mkl (last time visited 31.07.2016).