Acta Polytechnica CTU Proceedings **15**:6–11, 2018
doi:10.14311/APP.2018.15.0006

available online at http://ojs.cvut.cz/ojs/index.php/app

PERFORMANCE EVALUATION OF DIFFERENT LINEAR EQUATION SOLVERS FOR SOLVING NONLINEAR FE PROBLEMS ON MULTICORE ARCHITECTURES

Michal Bošanský*, Bořek Patzák

Czech Technical University in Prague, Faculty of Civil Engineering, Thákurova 7, 166 29 Praha 6, Czech Republic

* corresponding author: michal.bosansky@fsv.cvut.cz

Abstract. The aim of this paper is to evaluate the performance of existing parallel linear equation solvers to solving large-scale, nonlinear finite element analysis problems on systems with distributed memory. The parallel approach allows us to take an advantage of the distributed memory enabling forming large system matrices and of multiple processing units to achieve significant speedups. Our study is based on comparison of parallel direct solver and parallel iterative solver implemented in SuperLU DIST library from Portable, Extensible Toolkit for Scientific Computation (PETSc). Both considered solvers are designed for distributed system memory model and are based on a Massage Passing Interface (MPI).

The efficiency of individual solvers is evaluated on a selected benchmark problems, with different solution strategies by comparing computation times and obtained speedups.

Keywords: Nonlinear system, distributed memory, direct solver.

1. Introduction

Computational advancements have started a new trend in computational science and engineering that were powered by development and increases in the power and pervasiveness of computer and communication. Capitalization on those advances by developing techniques for modern hardware enable the solution of large and complex problems. Traditional finite element solvers run in serial mode and are limited by available resources represented by a single machine. The serial computers permit us to run simulation codes only sequentially and often have limited available resources represented by the single processing unit and available system memory. Parallelization can significantly reduce the solution time by more efficient use of modern hardware and enable to solve large problems by utilizing distributed memory resources.

Parallel computers can be classified by type of system memory architecture. The shared, distributed and hybrid (combination of shared and distributed memory) memory systems exist [1]. This paper is focused on distributed memory systems. In distributed memory systems, the memory is physically distributed on individual processing units and there is no global shared memory as on shared memory systems, where all computing units can access the same physical global address space. The explicit communications between processing units are needed to establish data exchange. It is a task of the programmer to explicitly define how and when data is communicated. The cost of communication, compared to a shared memory systems, can be very high, on the other hand, the advantage is that overall memory is scalable with increasing number of processors an memory access is not limited by a single memory bus. In recent years a new trend is based on using graphics processing units together with traditional processors to accelerate the solution process is enabling. Graphics processing unit computes intensive portions of the solution while the remainder of the code still runs on the processor.

The idea of parallel algorithms is based on partitioning the problem into a set of smaller tasks, that can be solved simultaneously. Scalability of computation is the most important goal in parallel computing [2]. The scalable parallel algorithm allows achieving decreasing execution time by using an increasing number of processing units, ideally in a linear trend. The ideal scalability is difficult to obtain due to the overhead cost of the parallel algorithm (synchronization and communication) and due to the fact that some parts of the problem are essentially sequential. Moreover, despite obtained speed-up, the parallel computing allows solving large, complex problems that cannot be solved at a single, well-equipped machine.

The Finite Element Method (FEM) has become a widely used tool for solving problems described by partial differential equations and it has been widely adopted by engineering and scientific communities as a reliable numerical tool. In FEM the differential equations are converted to the algebraic system of equations by using variational methods with the help of decomposition of the problem domain into sub-domains called elements and smart choice of interpolation functions.

Numerical solutions of many engineering problems lead to the solution of nonlinear models consisting of very complex geometries and many degrees of free-

dom. Nonlinear problems add additional complexity. In structural mechanics the nonlinearity can originate from nonlinear geometrical relations (large deformations), nonlinearity of constitutive relations on from boundary conditions (follower type of loading, for example) problem, typically using Newton-Raphson algorithm. This makes the nonlinear problem solution more demanding, compared to linear problems. The schema of common process for iteration processes is presented on next figure Fig. 1.

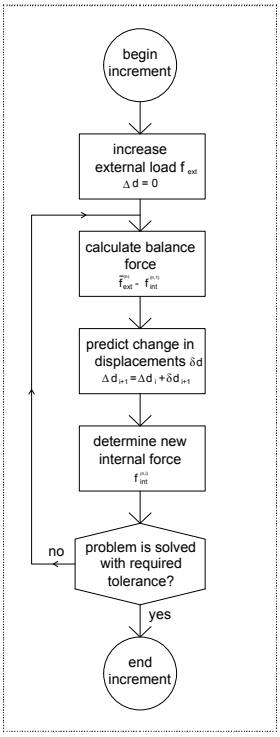

FIGURE 1. Iteration process.

Nonlinear structural analysis is the prediction of the response of nonlinear structures by modelbased simulation. The concept of equilibrium path explaining the solution process of nonlinear structural analysis. This concept lends itself to graphical representation in the form of response diagrams. The form that is used in this paper is load-deflection response diagram. The mechanical behavior of structures can be characterized by load-deflection or force-displacement response. This response is represented in two dimensions as representative force quantity marked by **f** against a representative displacement quantity represented by **d** as illustrated in Fig. 2.

Term path is a continuous curve in load deflection diagram and in typically the path is smooth. Smooth path has a continuous tangent except at exceptional points. The state or configuration of the structure represents each point in the path. The equilibrium path represents configurations in static equilibrium. The inception of the response diagram (load is equal to zero, displacement is equal to zero) is characterized reference state and this state represents configuration from which loads and displacements are measured. For structural analysis there are four sources of nonlinear behavior. The corresponding nonlinear effects are identified by the term material, geometric, force boundary conditions and displacement boundary conditions.

In this paper, material nonlinearity we consider that only being the source of nonlinear behavior. Material behavior depends on current deformation state and possibly past history of deformation. Other constitutive variables, prestress, temperature, time, etc. may be involved [3]. The problem representing discrete equilibrium equations at nodes is described by system of nonlinear algebraic equations

$$\mathbf{f}_{int}(\mathbf{d}) = \mathbf{f}_{ext}, \tag{1}$$

where the $\mathbf{f}_{int}$ represents the internal forces vector depending on unknown displacements $\mathbf{d}$ and $\mathbf{f}_{ext}$ representing external load. The problem can be linearized by means of Taylor series expansion of $\mathbf{f}_{int}(\mathbf{d})$. The internal force vector find represent nodal equivalent of internal stresses and defined as

$$\mathbf{f}^{\sigma}_{int}(\mathbf{d}) = B^T \sigma(\varepsilon_{(d)}) dV, \tag{2}$$

where B is strain-displacement matrix, σ is stress and ε is displacement. The Taylor Series expansion of $\mathbf{f}_{int}(\mathbf{d})$ around the point $(\overline{\mathbf{d}})$ is given by

$$\mathbf{f}_{int}(\mathbf{d}) + \frac{\partial \mathbf{f}_{int}(\mathbf{d})}{\partial \mathbf{d}} \Delta \mathbf{d} = \mathbf{f}_{ext}, \tag{3}$$

where $\frac{\partial \mathbf{f}_{int}(\mathbf{d})}{\partial \mathbf{d}}$ represents the tangential stiffness matrix $\mathbf{K}$ of the structure. After substitution the equation 2 has a form

$$\mathbf{K}\Delta\mathbf{d} = \mathbf{f}_{ext} - \mathbf{f}_{int}(\mathbf{d}), \tag{4}$$

Newton's method is a commonly used to iteratively solve nonlinear systems. In the general form, load movement is applied and the incremental displacement is solved from linearized, discrete problem. The residual loading is evaluated from the difference of external and internal forces corresponding to achieved displacement, structure is loaded by residual and corresponding incremental change of displacement is evaluated. This iterative process is repeated, until required tolerance is achieved. In this contribution, we consider different in variants of the Newton-Raphson method, the modified Newton-Raphson method and the initial stiffness method [4]. The full Newton-Raphson method is based on using the tangent stiffness matrix in each iteration which is formed and factorized in each iteration. The advantage of the method is fast convergence, but it can be computationally expensive for some types of problems.

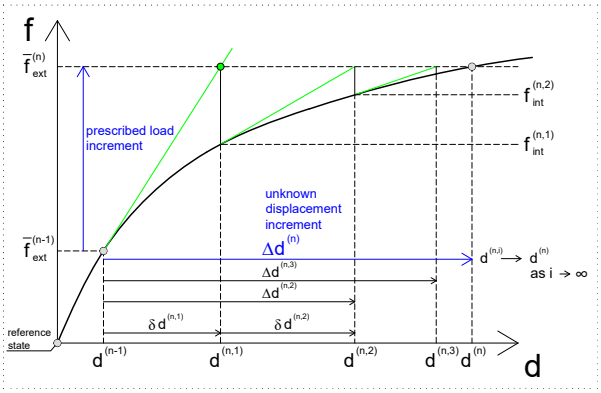

Figure 2. Newton - Raphson method.

The modified Newton-Raphson method has the same algorithm, but stiffness matrix is loaded only after certain number of iterations or out the forming of each loading step. This approach can be computationally expensive as the stiffness matrix should be factorized only when changed however, the solution process has slower convergence rate compared to the full Newton-Raphson method.

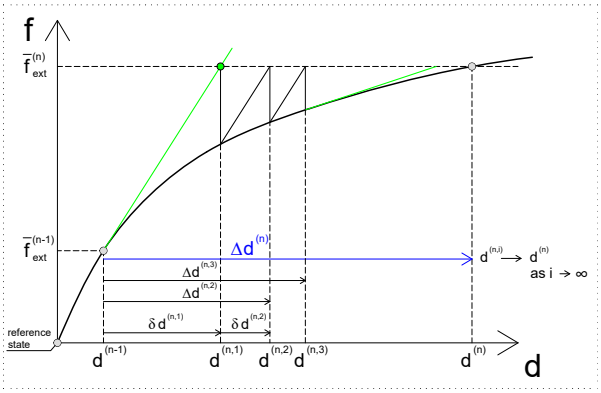

Figure 3. Modified Newton - Raphson method.

The initial stiffness method is using initial, elastic stiffness matrix constructed only once and which is held constant throughout the solution process. This method is robust but requires a large number of iterations to converge. The optimal choice is problem dependent. The iterative solver has basically the same solution cost of each problem, while direct solver can profit from existing factorization of system matrix in case there is no change (also iterative solver can profit, as no preconditioner setup is needed), but the effect is not in common with profit from existing factorization of system matrix,

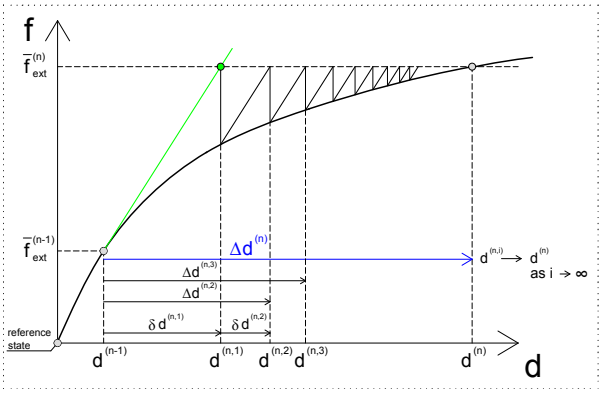

Figure 4. Initial stiffness method.

2. Implementation

SuperLU is a general purpose library for the direct solution of a large sparse symmetric or non-symmetric system of linear equations [5]. SuperLU library provides serial, multithreaded (shared-memory), and distributed memory versions. The performance of the distributed version of SuperLU solver based on Message Passing Interface (MPI) [6] is evaluated nonlinear structural analysis and compared with implemented Portable, Extensible Toolkit for Scientific Computation (PETSc) as the iterative solver [7] in this paper.

The comparison was made using the OOFEM open-source finite element solver [8], distributed under GNU public license, written in C++ programming language. This solver has been extended to support the SuperLU distributed version. The interface to SuperLU DIST library consists of new classes to represent interface to SuperLU DIST (direct solver based on distributed memory model) and classes implementing SuperLU DIST compatible sparse matrix storage. The PETSc (iterative solver based on distributed memory) support already exists. The SuperLU driver advantageously uses the symmetry of the matrix in the nonlinear system. The sparse matrix and the right-hand side vector are distributed among all the processes using the distribution based on block rows. That is, each process owns a block of consecutive rows of matrices. Distributed sparse matrix is stored in a compressed row format. This choice comes from SuperLU library, which requires the sparse matrix in this format on input. In the compressed row format, only nonzero entries of the sparse matrix are stored in one dimensional array. Additional integer array is needed to store column indices of the stored values. We assume, that the portioning of the discretized problem domain has been established and individual, non-overlapping sub-domains (partitions) are assigned to and stored locally on individual computing nodes. The so called node-cut strategy is assumed, where the cut dividing the problem domain runs through the nodes. Nodes on mutual partition boundaries are called shared nodes, the nodes inside individual partitions are so-called local nodes.

The process of parallel assembly is using group

of processing nodes assembling the problem simultaneously. Each processing node is responsible for assembling its contribution to the global system matrix by summing up the contributions from individual elements assembled to the node by portioning. To minimize the communication, it is natural that each processing node will assemble and maintain in its local memory corresponding block of global stiffness matrix. The ownership is uniquely defined for local nodes, which are exclusively shared by local elements on individual partitions. For shared nodes, which are shared by elements from multiple partitions, the ownership has to be defined by convention. The process of assembling the global right hand side vector is very similar to the process of assembling the stiffness matrix, but in many aspects is simpler, do to the assembly of local vector contributions.

3. Results

The mentioned nonlinear solution strategies have been evaluated on 3D finite element model of anchor pull out test using two different sparse linear solvers (SuperLU DIST and PETSc) using distributed memory programming model. The anchor is located close to the boundary, requiring 3D analysis with only one plane of symmetry. As the stell anchor is pulled out of concrete, the crack surface is initiated at the anchor head and starts to propagate toward the boundary as the loading increases. To model concrete fracture, an anisotropic, nonlocal damage based model has been used. The anchor pullout test mesh consists of 444 nodes and 799 triangular elements with nonlinear interpolation. The loading was controlled by prescribed increments of anchor pull out. In total 15 loading steps have been analyzed, corresponding to almost fully pulled anchor. The individual approaches were tested on Linux workstation (running Ubuntu 14.04 OS) with the two CPU Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz and 126GB RAM. Each of the two CPU units consists of eight physical and sixteen logical cores, allowing up to thirty-two threads to run simultaneously on one workstation. All the tests fit into a system memory.

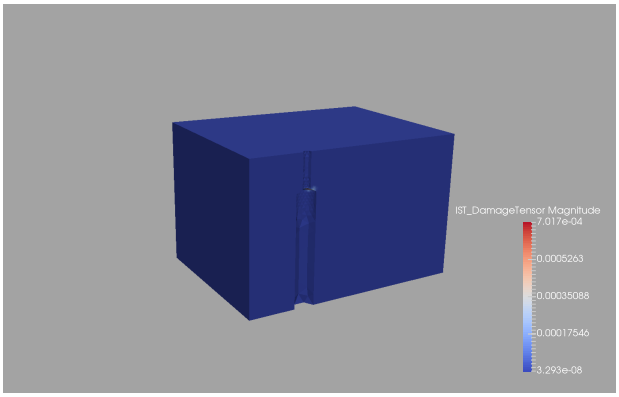

Figure 5. Anchor pullout test.

The evaluated nonlinear solution strategies include Newton-Raphson method with stiffness matrix update after every iteration, Modified Newton-Raphson method with stiffness matrix updated after every two and ten iterations, and initial stiffness method.

The total number of iterations in our testing benchmark problem for all considered strategies (Newton-Raphson method, modified Newton-Raphson method with all cases, initial stiffness method) is presented in Tab.1. It is clear that the number of iteration which is needed to successfully solve our benchmark problem is higher for Initial stiffness matrix method than the Newton-Raphson method. The number of iterations in one loading step is in the range from 2 (Newton-Raphson method) to 30 (Initial stiffness method) iterations to successfully solve this loading step as a part of the solution process.

Solution method	Num. of iteration	Matrix update
N-R	204	241
mod. N-R s. 2	251	131
mod. N-R s. 10	297	37
initial stiffness	6049	1

Table 1. Number of iterations for different methods.

To illustrate the performance of direct SuperLU DIST solver and iterative solver PETSc the execution times and speedups are presented. In case of iterative conjugated gradient solver from PETSc, the convergence criteria based on relative solution error equal to 10^{-6} has been used. The performance has been evaluated for case without any preconditioning, as well as for the case when block Jacobi preconditioning has been used. The obtained solution times (averaged over five consecutive runs) and corresponding speedups (relative to 2 CPU) are presented in Fig. 6, Fig. 7, Fig. 8, Fig. 9, Fig. 10 and Fig 11.

The achieved results show that the effect of the parallelization using both solvers is quite substantial. In case of full Newton-Raphson method where the stiffness is updated after every iteration, the performance of direct solver is lower than performance of iterative solvers. This is because of dense character of stiffness matrix, from which iterative solver can profit. Also direct solver could not profit from existing factorization, as the system matrix is updated after every iteration. The performance of preconditioned iterative solver is better compared to performance of non-preconditioned solver, see Fig. 6 for reference.

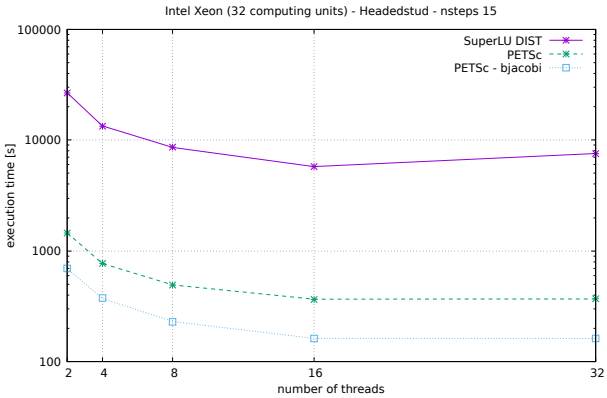

Figure 6. Execution times using the Newton-Raphson method with secant stiffness updated after every iteration.

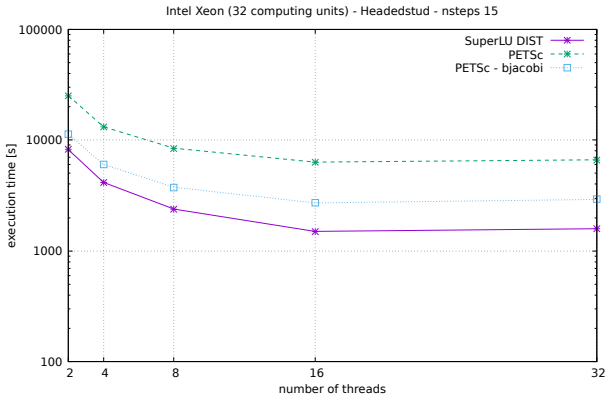

Figure 8. Execution times using the Initial Stiffness Matrix method.

The similar trend can be observed in case of Modified Newton-Raphson method. Again the performance of iterative solver is superior to performance of direct solver, see Fig. 7, comparing solution times for both solver and stiffness update every 2 or 10 iterations.

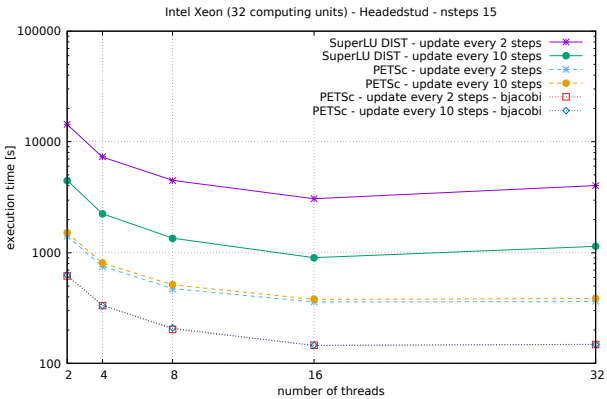

Figure 7. Execution times using the Modified Newton-Raphson method with secant stiffness updated after every 2nd and 10th iteration.

Finally, the achieved results of using Initial Stiffness Matrix method with using direct solver and iterative solver are presented. In this case the best performance has been obtained using direct solver, which is in this case extremely profits from existing factorization.

It is difficult to generalize obtained results, as the relative performance of direct and iterative solvers is highly depend on particular problem. However it is possible to evaluate the scalability of both solvers. The results for different settings are presented in Figs. 9, 10 and 11. Although far from ideal scalability, the results show reasonable linear scalability up to 16 cores. The significant decrease of performance for 32 threads, observed in all cases, can be attributed to hyper-threading technology of Intel processors, assembling to sharing some CPU resources between hyper-threaded cores, which takes place only for 32 threads (note that workstation had 16 physical hyper-threaded cores)

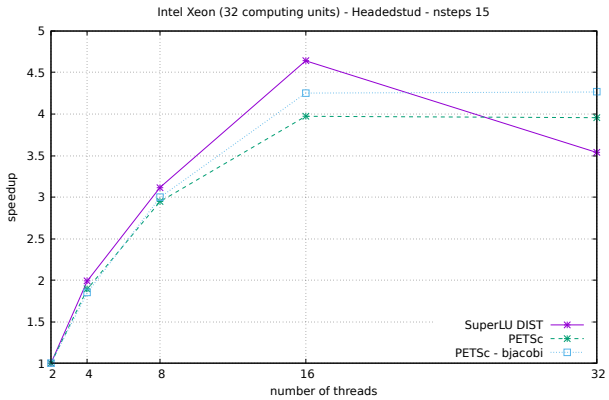

Figure 9. Speedups using the Newton-Raphson method with secant stiffness updated every iteration.

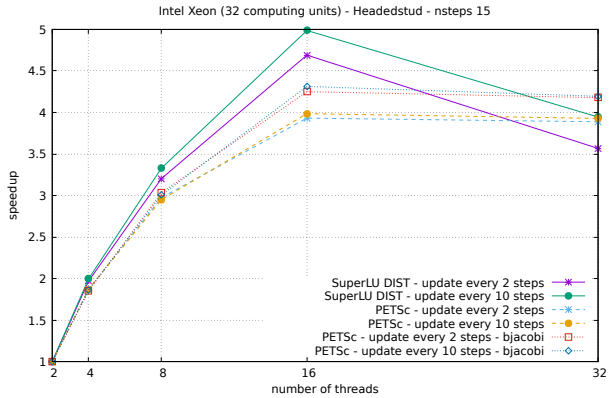

FIGURE 10. Speedups using the Modified Newton-Raphson method with secant stiffness updated after every 2nd and 10th iteration.

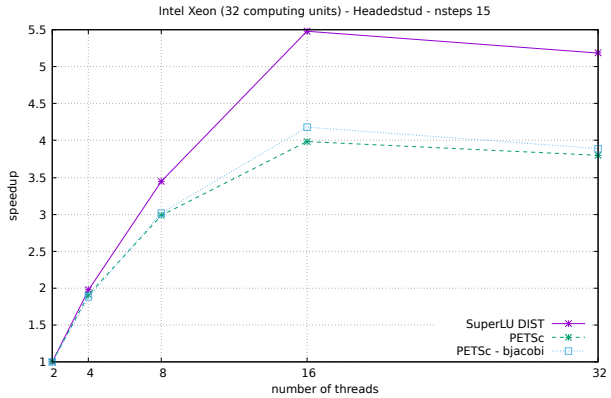

FIGURE 11. Speedups with using the Initial Stiffness Matrix method.

4. CONCLUSIONS

The paper evaluates the performance of a direct SuperLU and PETSc iterative solvers based on distributed system memory model in object-oriented, FEM framework. The parallelization strategy is based on message passing programming model. The performance of direct and iterative solvers are compared on complex nonlinear benchmark problem of 3D simulation of an anchor pullout test. For the particular benchmark case considered, the performance of iterative solvers has been superior, except the case of Initial Stiffness method, where direct solver had better performance. Despite particular problem considered, more general can depend or can be made regarding the scalability of both solvers. However, we can make general conclusion regarding scalability of both solvers. Both solvers show relatively good scalability. As the relative performance of individual solvers anyway for different use case and problems, it is definitely an advantage of having both solver types implemented in any Finite Element code.

ACKNOWLEDGEMENTS

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS16/038/OHK1/1T/11 - Advanced algorithms for numerical modelling in mechanics of structures and materials.

REFERENCES

[1] C. Hufhes, T. Hughes. *Parallel and Distributed programming Using C++*. Pearson Education, 2003.

[2] B. Patzak. *Parallel computations in structural mechanics*. Ceske vysoke uceni technicke v Praze, 2010.

[3] C. Fellipa. Nonlinear finite element methods. https://www.colorado.edu/, 2017.

[4] A. Ibrahimbegovic. *Nonlinear Solid Mechanics*. Springer Dordrecht Heidelberg London New York, 2009.

[5] X. S. Li, J. W. Demmel, J. R. Gilbert, et al. Superlu users‘ guide program interface. http://crd.lbl.gov/xiaoye/SuperLU/, September 1999.

[6] P. S. Pacheco. *A User‘s Guide to MPI*. Univ. of San Francisco, 1997.

[7] S. Balay, S. Abhyankar, M. F. Adams, et al. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2018.

[8] B. Patzak. Oofem. http://www.oofem.org, 2000.