DOI https://doi.org/10.30525/978-9934-26-019-3-5

APPLICATION OF PROTOTYPE DESIGN PATTERN IN PARALLEL IMPLEMENTATION OF FINITE ELEMENT ANALYSIS SYSTEMS

Ihnatchenko M. S., Kudin O. V., Homeniuk S. I.

INTRODUCTION

The development of modern mechanical engineering and construction, in particular, the creation of new aerospace technology, transport vehicles, power plants and other complex engineering systems is impossible without the use of automated design systems (ADS). Today's level of development of computer technology in every way contributes to the constant increase of its role. It is now virtually impossible to design and build complex machines, mechanisms and structures without the use of computers.

Automation of design work is usually used in the following cases:

1) when performing routine engineering work (creation of drawings, preparation of various input and output documentation, etc.);

2) when analyzing the properties of the design object (study of compliance of the characteristics of the created object to the requirements of the customer);

3) when performing such design tasks that are not subject to full formalization (tracing of electronic boards; creation of various schedules, etc.) [1].

The most complex and important in practice is the automation of the analysis of the properties of the design object, which allows you to replace expensive and time-consuming experimental study of a prototype with a virtual experiment, the essence of which is to build and study appropriate mathematical model of the design object using computer technology. In addition, actual tests of prototypes can lead to serious economic costs and sometimes to catastrophic consequences (for example, in construction or in the creation of rocket and space technology).

When designing complex engineering systems, the main thing is usually to study their strength and durability. This leads to the need to analyze the stress-strain state of the design objects, which, in turn,

requires the solution of different classes of problems in the mechanics of a deformable solid.

Since the actual problems of mathematical physics in most cases cannot be solved by analytical methods, in practice, a variety of approximate numerical methods is used, the most common of which is the finite element method (FEM) [2].

The practical application of FEM without the use of a computer is virtually impossible. Therefore, for its application today a large number of different software that automates various aspects of the application of FEM is created: from the generation of two- and three-dimensional finite element (discrete) models of the object to visualization of large arrays of numerical results.

The most well-known commercial software tools for finite element analysis of different classes of boundary value problems include Abaqus [3], Ansys [4], COMSOL [5], MSC Nastran [6] etc. [7]. Alternative to them are open source systems, among which there are dial.II [8], FreeFEM [9], OpenCAD [10], etc. [11].

Modern ADSs that implement different finite element analysis algorithms can be divided into three separate subsystems:

– preprocessor – automates the preparation of output for further numerical calculation of data (most often at this stage a geometric model of the object of calculation is created and the process of its sampling for a given type of finite elements is performed);

– processor – the central part (core) of finite element software that performs direct numerical calculation of the problem (at this stage, global matrices of stiffness, mass and damping are usually built, boundary conditions are taken into account and the corresponding systems of linear algebraic equations are solved);

– postprocessor – automates the analysis of the obtained results (most often at this stage the visualization of large arrays of numerical data, as well as the synthesis of new information on the basis of previously obtained are performed).

Despite the large amount of existing software for finite element analysis of different classes of mechanics problems, there is often a need to improve them or create new programs. This can be explained, for example, by the fact that non-existent structural materials (carbon plastics, elastomers, ceramics, etc.) are regularly created, to take into account the features of which you need to create new or adapt existing models and calculation methods. In addition, recently almost all computer systems are equipped with several separate processors (or one

multi-core). To use all available computation resources effectively, it is necessary to develop parallel (distributed) versions of finite element analysis algorithms for certain classes of boundary value problems.

Thus, the development of parallel algorithms and software for automation of various aspects of finite element analysis today is a very complex and urgent task that requires the development of new mathware and software.

Preprocessor

The main function of the preprocessor is to build a finite element model of the geometric domain occupied by the object of calculation. This problem can be divided into two separate tasks:

1) development of a formal description of the geometry of the original object of calculation in a certain form suitable for further automatic processing using computer technology;

2) automation of construction of a finite element model according to the previously obtained geometric model [12].

The first task is quite complex and creative, especially for geometric domains of non-standard shape. Usually in engineering practice boundary representation [13] and solid modeling (solid modeling) are most often used for geometric modeling [14]. The main disadvantage of these approaches is the rather high work content and complexity of the description of objects of non-standard shape.

The most universal and natural way of geometric modeling of domains of arbitrary shape is the use of functional representation [15], which is based on the use of R-functions [16]. According to it, an arbitrary geometric domain Ω in R^3 can be described as such a function $F(x,y,z)$, for which the following relations are fulfilled: $F(x,y,z) \geq 0$, if $(x,y,z) \in \Omega$ (whereby $F(x,y,z) = 0$, if the point $(x,y,z) \in \partial\Omega$, where $\partial\Omega$ is the boundary of the domain Ω), i $F(x,y,z) < 0$, if $(x,y,z) \notin \Omega$.

V.L. Rvachev proved [16] that for any geometric domain Ω you can construct a function $F(x,y,z)$ by means of elementary mathematical functions and logical operations of conjunction, disjunction and inversion.

The function $F_1 \wedge F_2$ is called the R-conjunction and is determined by the ratio:

$$F_1 \wedge F_2 = \left(F_1 + F_2 - \sqrt{F_1^2 + F_2^2} \right) / 2 .$$

Similarly, the function $F_1 \vee F_2$ is called R-disjunction and is determined by the formula:

$$F_1 \vee F_2 = \left(F_1 + F_2 + \sqrt{F_1^2 + F_2^2}\right) / 2 .$$

The function $\underline{F}$ is called R-inversion, and is calculated by the following expression:

$$\underline{F} = -F .$$

For example, a functional model of a three-dimensional geometric region formed by the intersection of two spheres of different radii (Figure 1), can be described as follows:

$$F = F_1 \wedge \underline{F_2} ,$$

where $F_1(x, y, z) = R^2 - x^2 - y^2$,

$$F_2(x, y, z) = r^2 - (x - a)^2 - (y - b)^2 ,$$

R, r are the radii of larger and smaller spheres (respectively);
(a, b) are the coordinates of the center of a smaller sphere.

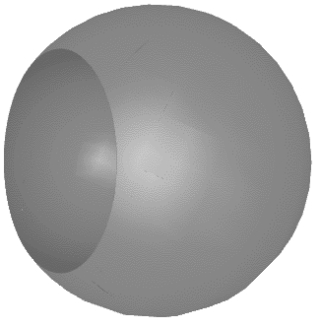

Fig. 1. The figure formed by intersection of two spheres of different radii

The practical application of the functional approach to geometric modeling is complicated by the fact that the R-function $F(x, y, z)$ is implicit. To visualize or discretize a functionally given geometric domain, it is necessary to construct a certain approximation of its surface, which requires finding the coordinates of some set of boundary nodes (for which the relation $F(x, y, z) = 0$ holds). An overview of sequential algorithms for finding points on the surface of a functionally given domain is presented in [17, 18].

To increase their efficiency, it is necessary to develop appropriate parallel algorithms. The simplest for practical implementation is the method of parallel decomposition "Divide and rule" [19]. Accordingly, the initial task of finding a set of points belonging to the boundary $\partial\Omega$ implicitly given geometric domain Ω, is divided into a set of subtasks that are solved in parallel, and the results are later combined.

The sequential algorithm for finding the boundary of the domain given by the R-function can be described using pseudocode as follows:

Algorithm *Procedure for finding boundary nodes in a geometric domain*

procedure *FindBnPts*(*BoundaryPoints*, *Box*, *N*)

BnPts is the requested vector of coordinates of points on the boundary of the domain

$Box = (X_{min}, Y_{min}, Z_{min}, X_{max}, Y_{max}, Z_{max})$ are cuboid coordinates (search area)

$N = (N_x, N_y, N_z)$ is the number of steps along the coordinate axes

begin

$$H_x = \frac{(X_{max} - X_{min})}{N_x}$$

$$H_y = \frac{(Y_{max} - Y_{min})}{N_y}$$

$$H_z = \frac{(Z_{max} - Z_{min})}{N_z}$$

while $i \in [0, N_x - 1]$ **do**

$$x_0 = X_{min} + i \cdot H_x$$

$$x_1 = X_{min} + (i + 1) \cdot H_x$$

while $j \in \lfloor 0, N_y - 1 \rfloor$ **do**

$y_0 = Y_{min} + j \cdot H_y$

$y_1 = Y_{min} + (j+1) \cdot H_y$

while $k \in [0, N_k - 1]$ **do**

$z_0 = Z_{min} + k \cdot H_z$

$z_1 = Z_{min} + (k+1) \cdot H_z$

if $F(x_0, y_0, z_0) \le 0$ **and** $F(x_1, y_1, z_1) \ge 0$ **then**

$Find(x_0, y_0, z_0, x_1, y_1, z_1) \rightarrow BnPts$

endif

end while

end while

end while

end procedure

Here the procedure *Find()* implements search at a segment of coordinates $(x_0, y_0, z_0) - (x_1, y_1, z_1)$ of the point for which the R-function describing the original geometric domain takes a value of zero.

A corresponding parallel implementation of the above sequential algorithm can be obtained by dividing the original search cuboid into a number of individual subdomains, the number of which is selected depending on the available number of processors in the computer system. Then the results of parallel algorithms should be combined into one resulting array of boundary nodes (Figure 2).

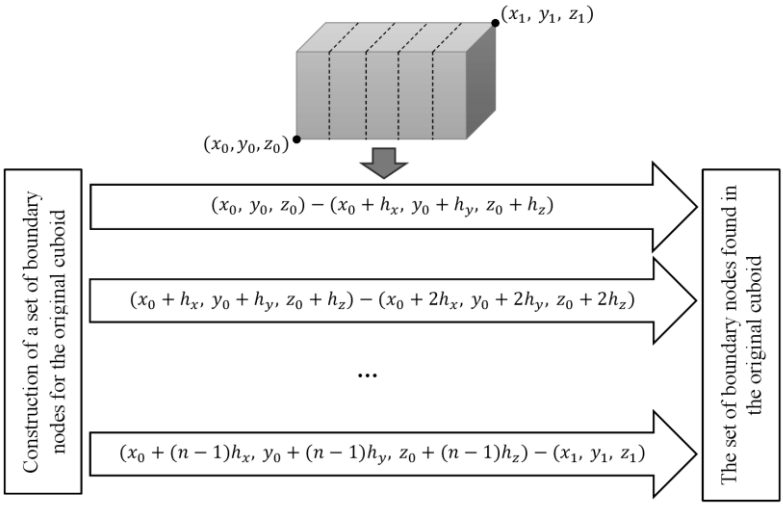

Fig. 2. An example of parallelism in the implementation of the search for boundary nodes

The software implementation of this procedure is convenient to perform using the creative design pattern Prototype [20], which allows you to copy objects of any complexity without going into the details of their software implementation. Thus, when using this template to develop a finite element analysis program, you can, for example, perform cloning of an object that implements the above algorithm on a particular node of a computation cluster or processor (processor core), without reference to its class (Figure 3).

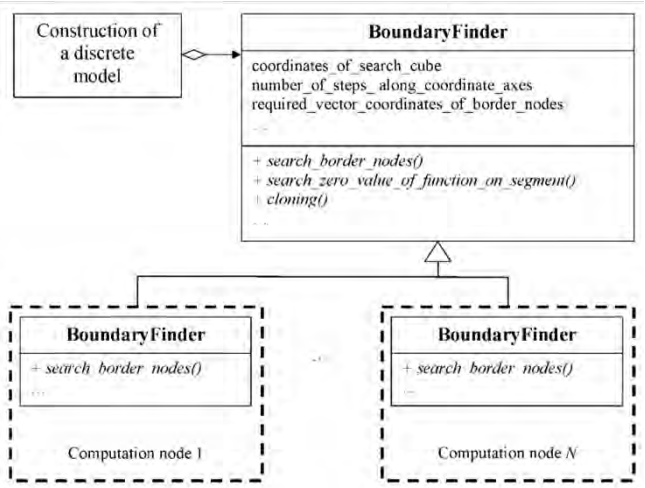

Fig. 3. An example of cloning procedures for constructing a set of boundary nodes

After obtaining a set of nodes belonging to the boundaries of the functionally given source domain, using the algorithm Marching cubes [21], you can obtain a preliminary triangulation of the surface of the source domain. As an example, consider the geometric domain “Cup”, which is given by the following relationship:

$$F(x,y,z) = F_1(x,y,z) \vee F_2(x,y,z),$$

where

$$F_1(x,y,z) = F_3(x,y,z) \wedge F_4(x,y,z);$$

$$F_2(x,y,z) = F_5(x,y,z) \vee F_6(x,y,z);$$

$$F_3(x,y,z) = a\sqrt{(x+a)^2 + (y+b)^2} - (x+a)^2 - (y+b)^2 - z^2 - b;$$

$$F_4(x,y,z) = -x - b;$$

$$F_5(x,y,z) = F_7(x,y,z) \wedge F_8(x,y,z);$$

$$F_6(x,y,z) = 2b\sqrt{x^2 + z^2} - x^2 - y^2 - z^2 - 2a;$$

$$F_7(x,y,z) = F_9(x,y,z) \wedge F_{10}(x,y,z);$$

$$F_8(x,y,z) = -y;$$

$$F_9(x,y,z) = 1 - \left(x^2 + z^2\right)^2 / c - y^4 / (16c);$$

$$F_{10}(x,y,z) = \left(x^2 + z^2\right)^2 / (4c) + y^4 / (5c) - 1;$$

$$a = 4;$$

$$b = 3;$$

$$c = 256.$$

Figure 4 shows the boundary representation of the "Cup", described by the above function, which has been obtained using the above parallel algorithm.

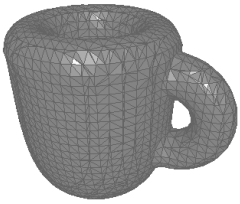

Fig. 4. Preliminary boundary representation of the R-function "Cup"

The operating time of the algorithm depending on the number of used computational flows is shown in Figure 5. The computational experiment was performed on a computer with an AMD Ryzen 7 2700X Octa-Core Processor clock rate 3.70 GHz and 32 GB RAM running Windows 10. To find the boundary of the domain, a grid consisting of 100 × 100 × 100 nodes was built.

The graph shows that the execution time of the parallel algorithm decreases logarithmically with increasing number of threads involved in the calculations until such time as their number does not equal the number of physical processor cores (AMD Ryzen 7 is equipped with eight physical cores). Then the speed does not change, which is due to the growing overhead of the operating system scheduler to switch controls between threads.

Having a boundary representation in the form of a triangulated surface and applying to it a frontal algorithm [22], we can obtain a discrete finite element model of the original geometric domain (Figure 6).

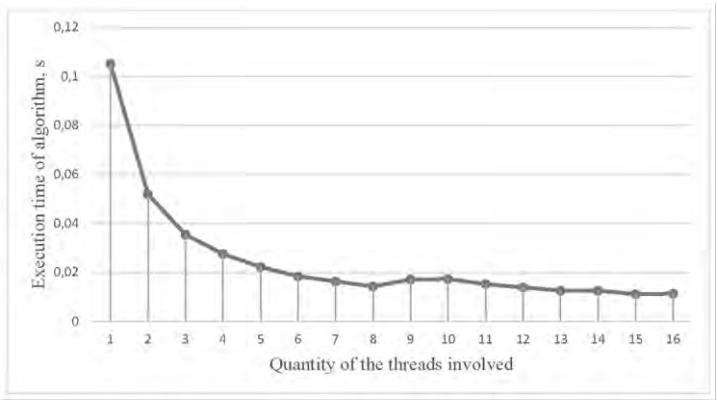

Fig. 5. Execution time of parallel algorithm of visualization of geometric domain "Cup" depending on quantity of the threads involved

Processor

The processor is the core of any ADS in mechanical engineering and construction. It directly implements computer analysis of models of design objects using a computational method (FEM, the method of boundary elements, the finite difference method, etc.).

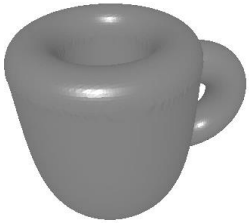

Fig. 6. Discrete model of R-function "Cup"

The most common in practice is FEM. Numerical analysis of the behavior of the designed structure using this method consists of the following steps [2]:

1) generation of local stiffness, mass and damping matrices for each finite element and their ensemble to the corresponding global matrices;

2) formation of a column vector – the right part of the system of linear algebraic equations (SLAE);

3) formation of a system of linear algebraic equations from global matrices;

4) taking into account boundary conditions;

5) SLAE solution;

6) calculation of additional nodal results (for example, deformations and stresses on the basis of the received displacements).

Each of the above stages can be quite long with a large number of finite elements, so modern ADSs use the technology of parallel and distributed computation in their implementation.

A typical object-oriented structure of a finite element processor ADS can be represented as follows (Figure 7). It consists of the following structural elements:

1) Message – a class that implements the output of information about the calculation, possible errors, results, etc.;

2) Mesh – a class that contains descriptions of all necessary information for the calculation of a discrete model of the projected object (coordinates of nodes, boundary and finite elements, connections between nodes, etc.);

3) Parameters – a class that implements the storage of a database of parameters of the current task, such as elastic and physical characteristics of the material; boundary conditions; load, etc.;

4) FEM – a base class describing the generalized FEM;

5) FEMStatic, FEMDynamic, FEMNonLinear, … – classes derived from FEM, which implement a certain calculation algorithm using the FEM of a given type of problem (statics, dynamics, viscoelasticity, contact interaction, etc.).

As in the development of the preprocessor, an effective technology for creating the ADS processor with support for parallel calculations is the use of a creative design pattern Prototype [20]. We consider its application in the development of ADS with support for parallel data processing. One of the longest stages of finite element computation is the process of formation of global matrices of stiffness, mass and damping. It usually consists of sequentially constructing local matrices for each element and assembling them into appropriate global ones. Therefore, when using object-oriented programming technology to implement this template, it is necessary to design a class of constructing local matrices of a finite element so that they support the implementation of exact copies of their objects on a particular processor, core, or computation node (Figure 8).

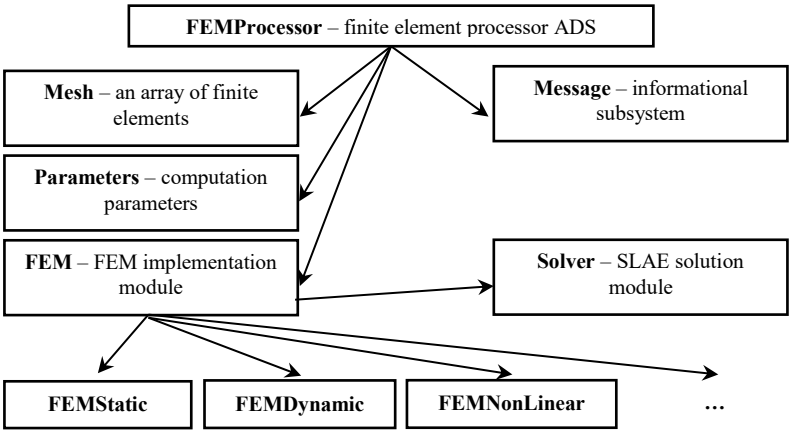

Fig. 7. Typical finite element processor structure

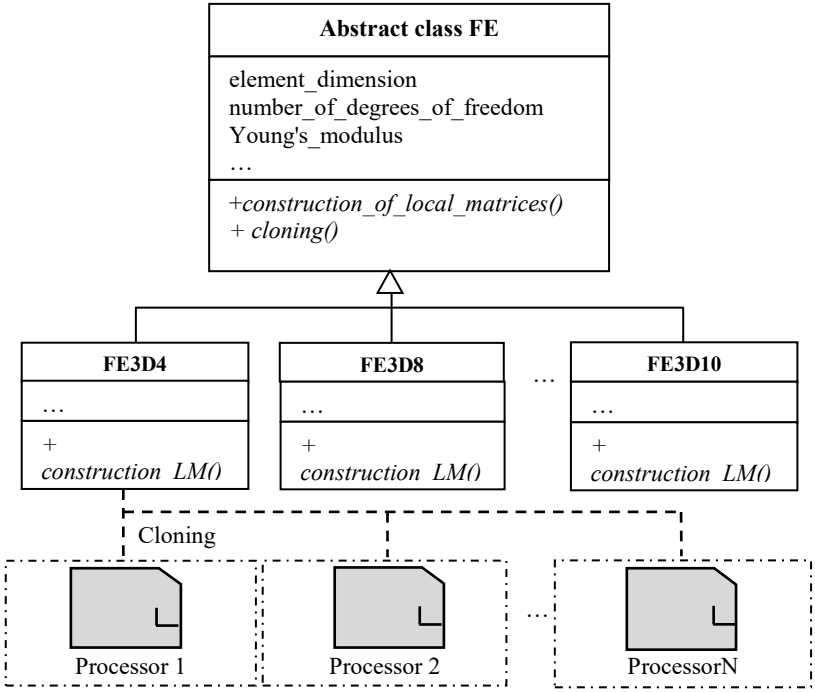

Fig. 8. Cloning scheme of the process of constructing local stiffness matrices on different processors of finite element

Similarly, appropriate classes are designed for implementing other stages of finite element analysis, such as solving SLAE, where, for example, the matrix factorization procedure can be designed so that it runs in parallel on a number of available processors, performing the cloning procedure of the object of the corresponding class (Figure 9).

Similarly, you can design other classes that implement the algorithm of finite element computation of a certain type of problem.

The above structure of the processor was implemented programmatically in the C ++ programming language using standard tools for parallel calculations of the STL library [23]. As a test, the problem of finding the parameters of the stress-strain state of an I-beam with circular holes, the upper surface of which was under the action of a uniformly distributed load, was solved. For the calculation, a corresponding discrete model was constructed, which consisted of 1855 nodes and 6991 linear tetrahedral finite elements (Figure 10).

The calculation was performed with a different number of threads involved on a computer with an Intel (R) Core (TM) i7-3630QM processor clock rate 2.40 GHz and 16 GB RAM running Windows 10. The dependence of the calculation time on the number of involved cores is shown in Figure 11.

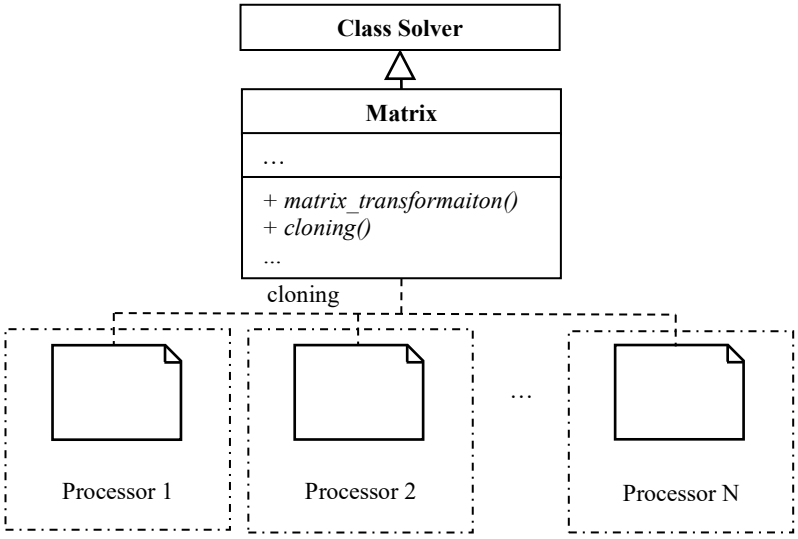

Fig. 9. Application of the Clone design pattern in the implementation of the SLAE solution subsystem

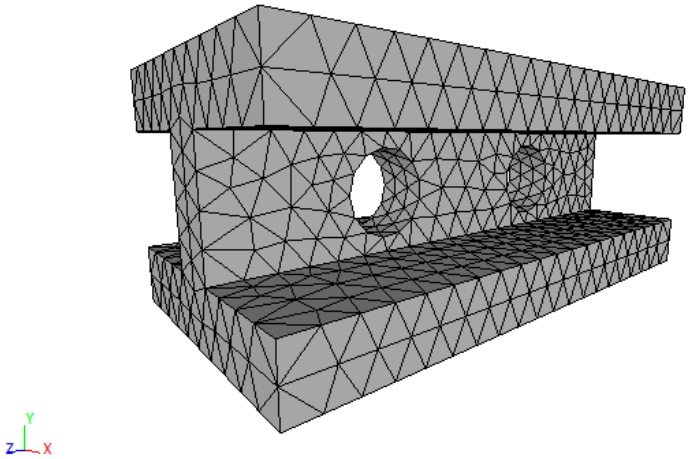

Fig. 10. Discrete model of I-beam with circular holes

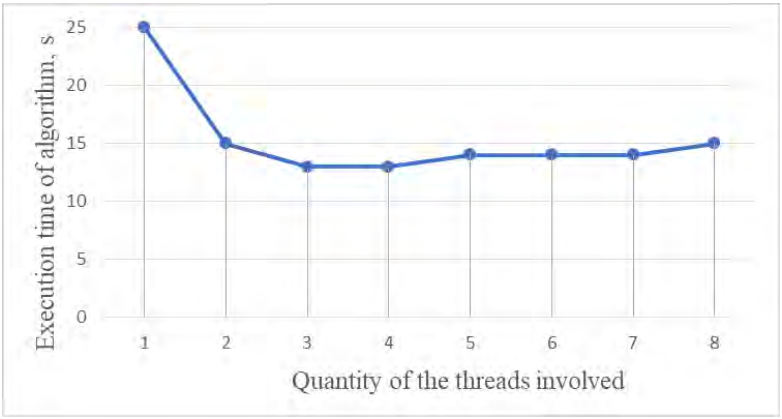

Fig. 11. Time of calculation of an I-beam with holes at various quantity of the threads involved

As in the previous experiment, the calculation time decreased with increasing number of threads involved until the number does not equal the number of physical cores. Then the speed of calculations does not change.

Postprocessor

When analyzing the numerical solutions obtained as a result of the application of the FEM, there are usually two main tasks:

– a large amount of numerical information to be processed (for example, research on the accuracy, reliability and adequacy of the content);

– the need to synthesize additional information, such as standard results of a finite element.

To automate the solution of these problems, specialized subsystems of finite element analysis systems – postprocessors – are used.

The most effective means of visual representation of the results obtained during the calculation is their certain visualization, when the distribution of the studied value in the calculation area is encoded with a certain color (Figure 12).

To construct such a figure, it is necessary to implement an algorithm for visualizing the distribution of the studied quantity over a separate boundary segment (for example, a triangle). The idea of such an algorithm is to divide the boundary segment into a number of triangles, the value of the studied value of which will be the same (Figure 13).

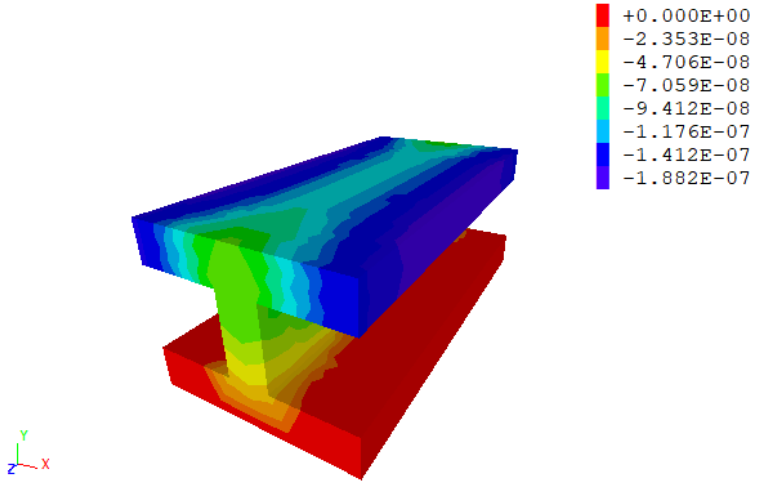

Fig. 12. Distribution of the vertical component of the displacement vector along the I-beam

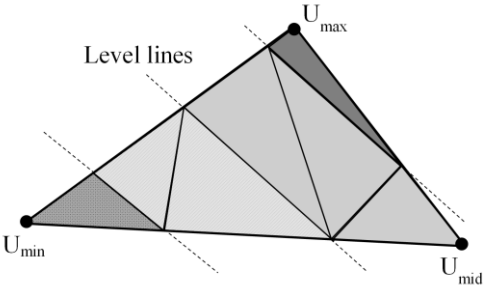

Fig. 13. The scheme of dividing the boundary segment into triangles, which correspond to the same value of the studied quantity

This algorithm can be described using pseudocode as follows:
Algorithm *Visualization of the boundary segment (triangle)*
procedure *ShowBoundarySegment*(*Coord*, *U*)
$U = (U_{min}, U_{mid}, U_{max})$ are nodal values of the required function
$Coord = (X_0, Y_0, Z_0, X_1, Y_1, Z_1, X_2, Y_2, Z_2)$ are coordinates of nodes of the boundary segment
$P02, P012$ are auxiliary vectors
begin
$C_{min} = ColorIndex(U_{min})$
$C_{mid} = ColorIndex(U_{mid})$
$C_{max} = ColorIndex(U_{max})$
if $C_{min} = C_{mid}$ **and** $C_{mid} = C_{max}$ **then**
ShowTriangle(*Coord* , C_{min})
return
endif

$Step = C_{max} - C_{min} + 1$
$H_x = (X_{max} - X_{min}) / Step$
$H_y = (Y_{max} - Y_{min}) / Step$
$H_z = (Z_{max} - Z_{min}) / Step$
$H_c = (C_{max} - C_{min}) / Step$

while $i \in [0,\ Step-1]$ **do**

$P02.add(X_0 + i \cdot H_x, Y_0 + i \cdot H_y, Z_0 + i \cdot H_z, C_{min} + i \cdot H_c)$

end while

$P02.add(X_2, Y_2, Z_2, C_{max})$

$Step = C_{mid} - C_{min} + 1$

$H_x = (X_{mid} - X_{min}) / Step$

$H_y = (y_{mid} - y_{min}) / Step$

$H_z = (z_{mid} - z_{min}) / Step$

$H_c = (C_{mid} - C_{min}) / Step$

while $i \in [1,\ Step-1]$ **do**

$P012.add(X_0 + i \cdot H_x, Y_0 + i \cdot H_y, Z_0 + i \cdot H_z, C_{min} + i \cdot H_c)$

end while

$P012.add(x_1, y_1, z_1, C_{mid})$

$step = C_{max} - C_{mid} + 1$

$h_x = (x_{max} - x_{mid}) / step$

$h_y = (y_{max} - y_{mid}) / step$

$h_z = (z_{max} - z_{mid}) / step$

$h_c = (C_{max} - C_{min}) / step$

while $i \in [1, step-1]$ **do**

$P012.add(x_1 + i \cdot h_x, y_1 + i \cdot h_y, z_1 + i \cdot h_z, C_{mid} + i \cdot h_c)$

end while

while $i \in [0,\ P02.length() - 2]$ **do**

if $i < P012.length()$ **then**

$$CurCoord = \begin{matrix} (P02[i+1][0], P02[i+1][1], P02[i+1][2], \\ P02[i][0], P02[i][1], P02[i][2], \\ P012[i][0], P012[i][1], P02[i][2]) \end{matrix}$$

$ShowTriangle(CurCoord,\ P02[i][3])$

endif

if $i + 1 < P012.length()$ **then**

$$CurCoord = \begin{matrix} (P02[i+1][0], P02[i+1][1], P02[i+1][2], \\ P012[i][0], P012[i][1], P012[i][2], \\ P012[i+1][0], P012[i+1][1], P02[i+1][2]) \end{matrix}$$

ShowTriangle(*CurCoord*, $P012[i+1][3]$)

endif
end while
end procedure

Here the procedure *ShowTriangle()* directly implements the reflection of a single-color triangle.

An example of the proposed algorithm is shown in Figure 12. It should be noted that when using modern graphics cards to make the visualization algorithm work in parallel does not make sense, because this procedure is automatically performed by the graphics processor. However, if necessary, using the Prototype pattern, you can build a class that divides the total set of boundary segments into subsets, for which processing is performed in parallel similarly to the scheme described above in the description of the preprocessor or processor.

CONCLUSIONS

The presented scheme of designing a finite element analysis system using the Prototype design pattern allows you to develop appropriate software for parallel computation systems quickly and efficiently.

SUMMARY

The article explores the application of prototype design pattern in parallel implementation of finite element analysis systems.

The development of modern mechanical engineering and construction, in particular, the creation of new aerospace technology, transport vehicles, power plants and other complex engineering systems is impossible without the use of automated design systems (ADS). Today's level of development of computer technology in every way contributes to the constant increase of its role. It is now virtually impossible to design and build complex machines, mechanisms and structures without the use of computers.

Thus, the development of parallel algorithms and software for automation of various aspects of finite element analysis today is a very complex and urgent task that requires the development of new mathware and software.

REFERENCES

1. Norenkov I.P. Fundamentals of computer-aided design: Textbook for universities. 2nd edition, revised and expanded. Moscow: Bauman University Publishing House, 2002. 336 p.
2. Zienkiewicz O. C., Taylor R. L., Zhu J. Z. The Finite Element Method: Its Basis and Fundamentals. Sixth edition. Butterworth-Heinemann, 2016. 753 p.
3. Design and Engineering Simulation | SIMULIA – Dassault Systemes. URL: https://www.3ds.com/products-services/simulia/
4. Engineering Simulation & 3D Design Software | Ansys. URL: https://www.ansys.com/
5. COMSOL Multiphysics ® Modelling Software. URL: https://www.comsol.com/
6. MSC Nastran – Multidisciplinary Structural Analysis. URL: https://www.mscsoftware.com/product/msc-nastran
7. Top Finite Element Analysis (FEA) Software List, Reviews, Comparison & Price | TEC. URL: https://www3.technologyevaluation.com/sd/category/finite-element-analysis-fea
8. The dial.II Finite Element Library. URL: https://www.dealii.org/
9. FreeFEM – An open-source PDE Solver using The Finite Element Method: URL: https://freefem.org/
10. OpenCAD The Programmers Solid 3D CAD Modeller. URL: https://www.openscad.org/
11. qzCAD. URL: https://github.com/qzcad
12. Choporov S. V., Hrebeniuk S. N., Homeniuk S. I. Functional approach to geometric modeling of technical systems. Zaporizhzhia: ZNU, 2016. 177 p.
13. Stroud I. Boundary Representation Modelling Techniques. London: Springer-Verlag. 2006. 788 p.
14. LaCourse D. E. Handbook of solid modeling., editor in chief. New York: McGraw-Hill, 1995. 308 p.
15. Pasko A., Adzhiev V., Sourin A. Savchenko V. Function representation in geometric modeling: concepts, implementation and applications. The visual computer. 1995. Vol. 11. P. 429-446.
16. Rvachov V. L. Theory of R-functions and some of its applications. Kyiv: Naukova Dumka, 1982. 106 p.
17. Tolok A. V. Functional-voxel method in computer modeling. Moscow: FIZMATLIT, 2016. 112 p.

18. Homeniuk S. I., Choporov S. V., Al-Atamneh B. G. M. Mathematical modeling of geometric objects in parallel computer systems: monograph. Kherson: Publishing house “Helvetyka”, 2018. 112 p.
19. Cormen Th., Leiserson Ch., Rivest R., Stein C. Algorithms: construction and analysis. Moscow: Williams, 2005. 1296 p.
20. Gamma E., Helm R., Johnson R., Vlissides J. Techniques of object-oriented design. Design patterns. Saint Petersburg: Piter, 2015. 368 p.
21. Lorensen W. E., Cline H. E. Marching Cubes: A high resolution 3D surface construction algorithm. In: Computer Graphics, Vol. 21(4), July 1987, P. 163-169.
22. Liseikin V. D. Grid Generation Methods. Springer International Publishing AG, 2017. 541 p.
23. Standard Template Library Programmer’s Guide. URL: https://justinmeiners.github.io/sgi-stl-docs/

Information about the authors:
Ihnatchenko M. S.,
Post-graduate student,
Zaporizhzhia National University
66, Zhukovsky str., Zaporizhzhia, 69600, Ukraine

Homeniuk S. I.,
Doctor of Technical Sciences, Professor,
Dean of the Mathematical Faculty,
Zaporizhzhia National University
66, Zhukovsky str., Zaporizhzhia, 69600, Ukraine

Kudin O. V.,
Candidate of Physical and Mathematical Sciences (PhD),
Associated Professor at the Software Engineering Department,
66, Zhukovsky str., Zaporizhzhia, 69600, Ukraine