Parallel Computing Technologies in the Finite Element Method

Sergey Choporov[1]

[1]Zaporizhzhya National University (ZNU), Zaporizhzhya, Ukraine

s.choporoff@gmail.com

Abstract. The finite element method is a powerful tool for the numerical simulation of a wide range of problems. Implementation of the finite element method in CAD systems on the basis of modern computers allows researchers to solve large scale problems. The article describes parallel algorithms for assembly of stiffness matrix and for solution of linear equations. Also this article contains two numerical experiments: Dirichlet problem on the complex domain (gear); Elasticity problem of three-layer shell. In the last section of the article author compares the performance of solutions of these problems with one, two and four parallel cores.

Keywords

Parallel processing, the finite element method, mathematical model, numerical method.

1 Introduction

Nowadays engineers and researchers are faced with solving of complex problems in a mathematical modeling and design. Numerical analysis naturally finds applications in all fields of engineering and the physical sciences. The finite element method (FEM) is a powerful tool for the numerical simulation of a wide range of problems from analysis of the stress and strain state of components, constructions and buildings, heat transfer, fluid dynamics analysis to microelectromechanical systems. Implementation of the finite element method in CAD systems on the basis of modern computers allows researchers to solve large scale problems [1, 2, 3].

The complexity of many components and constructions leads to complex geometric models (mathematical models of geometric areas). The subsequent application of discretization techniques [4] with the requirement of considering features of the geometric model and the physical formulation of the problem (for example, nonuniform mesh in areas of contact, damage and fractures, etc.) can lead to meshes with a large number of finite elements. To process mathematical models with a large number of finite elements we need significant computational resources.

A natural solution of the problem of processing of large meshes is optimization of the number of elements in discrete models. However, when we reduce the number of finite elements in some cases we lost the accuracy of the model. In these cases we can use parallel processing to reduce time spent on modeling. The rapid growth of a number functional units (processors, cores) in modern computational systems makes parallel processing urgent.

Thus, the aim is to develop a parallel algorithm for the main stages of the finite element method and to analyze the results of their implementation in numerical experiments.

2 Related Works

Today we can see many researches in the field of parallel processing. Scientists propose different approaches for applications of parallel technologies in different mathematical and engineering problems. The main idea of such researches is growing of performance by using parallel computations instead serial computations for independent step of algorithm.

The finite element method is numerical method that used to solve a large scale problems. Thus parallel processing in the finite element method is urgent. Main steps to develop low cost FEM cluster are described in [3]. Parallel algorithms for FEM are proposed to implement FEM on a cluster of workstations (10 DEC ALPHA 300X) and on a CRAY C98 [5]. A research concept of parallel finite element (FE) simulation for moving boundary and adaptive refinement problems using graphics processing unit is proposed by [6]. Main steps for development FEM software using MPI are described in [7, 8] and other. Acceleration of FEM using OpenMP is described by [9, 10] and other. Hybrid approaches for parallel processing of main steps of FEM also presented (see [11, 12, 13] and other). Parallel conjugate gradient solver for the finite element method also developed in [14].

Thus, parallel processing of the finite element method is fast growing direction of research. This article describes parallel algorithms for main steps of FEM and compares performance of obtained algorithms on multi-cores computer.

3 Parallel implementation of the finite element method

The basic idea of the finite element method is that any unknown continuous quantity (e.g., temperature, strain, and others) can be approximated by a piecewise continuous function defined on non-overlapping elementary geometric shapes. Elementary geometric shapes are called finite elements. Finite elements mesh is discrete model of the domain. The shape of the finite element completely defined by it nodes. Nodes are the connection points between adjacent elements. System of linear equations is constructed to determine the values of the unknown continuous quantity in nodes. This system of linear equation is called a global stiffness matrix.

The process of constructing a global stiffness matrix is called assembly. Assembly is the special sum of local stiffness matrices (element matrices). An element matrix is the results of the integration of matrix differential equations on the finite element. The size of a global stiffness matrix is the product of the number of nodes in a discrete model on the number degrees of freedom.

Thus, in the case of a sufficiently complex discrete models, the stages of construction of the global stiffness matrix and the subsequent linear algebraic solutions are the most demanding computing requirements. Therefore, the introduction of parallel computing in these stages is able to give the most significant result in terms of saving time designing and modeling.

Finite element analysis starts with loading and initialization of data (mesh, boundary values, force vector). The next step is determination of the number functional units (value K). This value is the basis for determining the number of parallel threads. Input-output operations are used only on the firs step (loading and initialization). The global stiffness matrix is sparse. Sparse matrices are usually compressed to reduce memory usage. Thus, in case of multiprocessor system the number of cores (value K) equal to the number of threads.

Assembly process is a special sum of local matrices. Thus, each of K parallel threads should produce assembly N/K elements (N is the number of elements in the model). In tis case all variables for integration should be local for parallel threads (to local variable can access only it thread-owner; see figure 1).

Update of global stiffness matrix is not thread-safe. Thus, access to global stiffness matrix should be blocked for other threads when one thread update it.

The time need to build the global stiffness matrix using parallel computing process, we can estimate the value T_p.

$$T_p = N(I/K + m) + K \cdot t_{thread}, \tag{1}$$

where I is CPU time for building of an element matrix (integration); m is CPU time for assembly element matrix in global matrix (time for update of compressed matrix); t_{thread} is CPU time for thread creation/destruction (overhead).

Similarly, time required to build a global stiffness matrix on the basis of traditional sequential algorithm can be estimated by value T_s.

$$T_s = N(I + m), \tag{2}$$

Using parallel calculating makes sense if $T_s > T_p$:

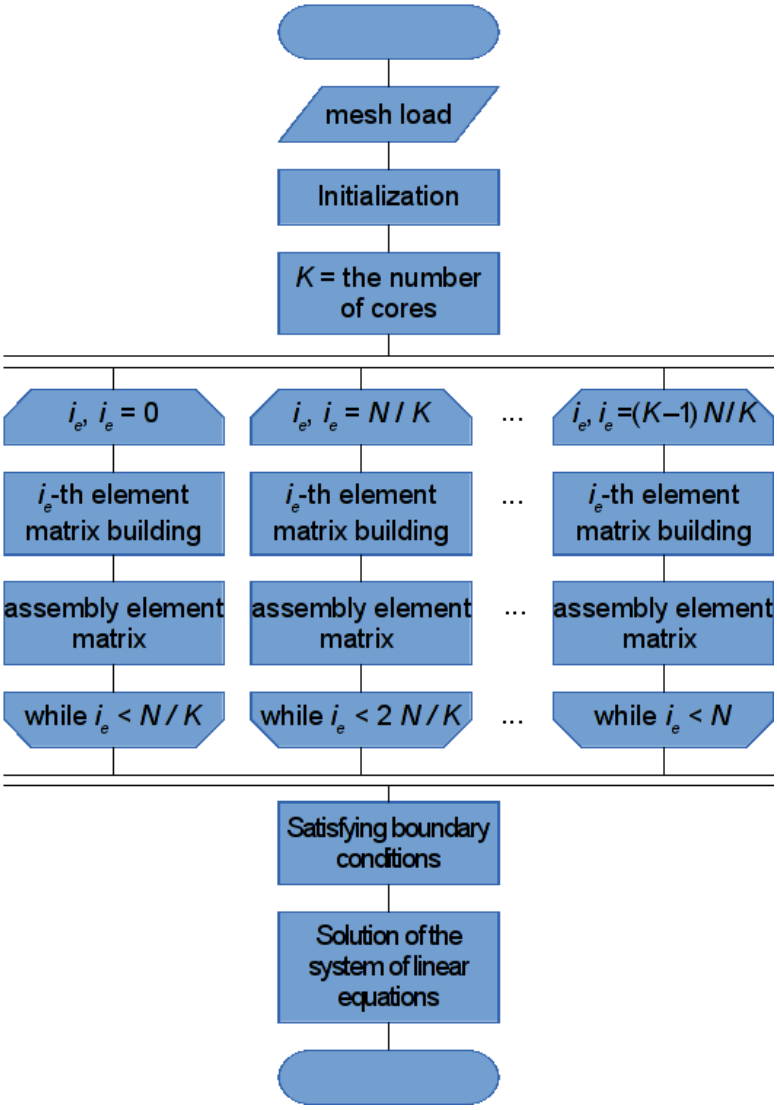

Figure 1. Parallel assembly of stiffness matrix.

$$N(I+m) > N(I/K+m) + K \cdot t_{thread}, \tag{3}$$

simplifying inequality 3 we'll get similar to Amdahl's law formulas.

$$NI > \frac{K^2 \cdot t_{thread}}{K-1}. \tag{4}$$

Thus, for example, in case of two threads CPU time for serial building elements matrices should be fourfold greater than CPU time for thread creation/destruction (which can not be performed with a small number of finite elements).

For solving obtained from the global stiffness matrix system of linear equations iterative algorithms (e.g., the conjugate gradient method, see [15, 16], etc.) are commonly used. In case of the conjugate gradient method result of current iteration is initial data for next iteration. Therefore, to reduce the CPU time for system of equations solving we should speed up inner operations of iterative method (matrix-vector product, calculation of residual, etc.).

Global stiffness matrix is sparse. An algorithm for matrix-vector multiplication should process only nonzero cell's of the matrix. Thus, algorithm of parallel matrix-vector multiplication on the basis of K threads is shown on figure 2. On the figure 2 A is the stiffness matrix (sparse), D is size of A, x is the vector for multiplication,

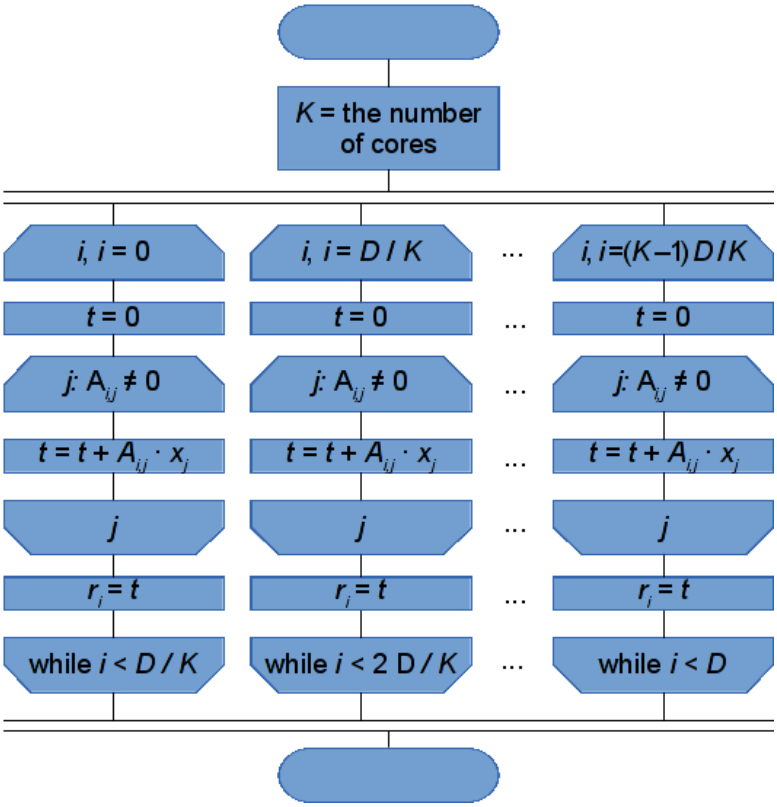

Figure 2. Parallel sparse matrix / vector multiplication.

r is the result of multiplication. A, D, x and r are common data for all threads. Since threads update different cells of r, then there is no need to lock access to r while it updating by thread.

4 Numerical experiment

For numerical experiment We used computer with Intel Core i3-380 (2.53 GHz, 2 physical cores + 2 virtual cores) CPU, 3 GB of RAM, openSUSE 12.2 operating system (compiler gcc 4.7), openMP library for threads management.

4.1 Dirichlet problem

Consider the solution of Laplace's equation 5 with Dirichlet boundary conditions 6.

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} = 0, \tag{5}$$

$$U_{\delta\Omega} = f(x, y, z). \tag{6}$$

Consider that domain is the gear (see figure 3). Function $f(x, y, z)$ described below.

$$f(x, y, z) = \begin{cases} 0, \text{ if } (x, y, z) \text{ is coordinate of the hole of the gear,} \\ 4\sin^2 5\phi, \text{ if } (x, y, z) \text{ is coordinate of the teeth of the gear,} \end{cases} \tag{7}$$

where ϕ is an angle of polar radius to the node of teeth in Oxy plane.

Formulas for evaluation of local matrix $[K^e]$ described below.

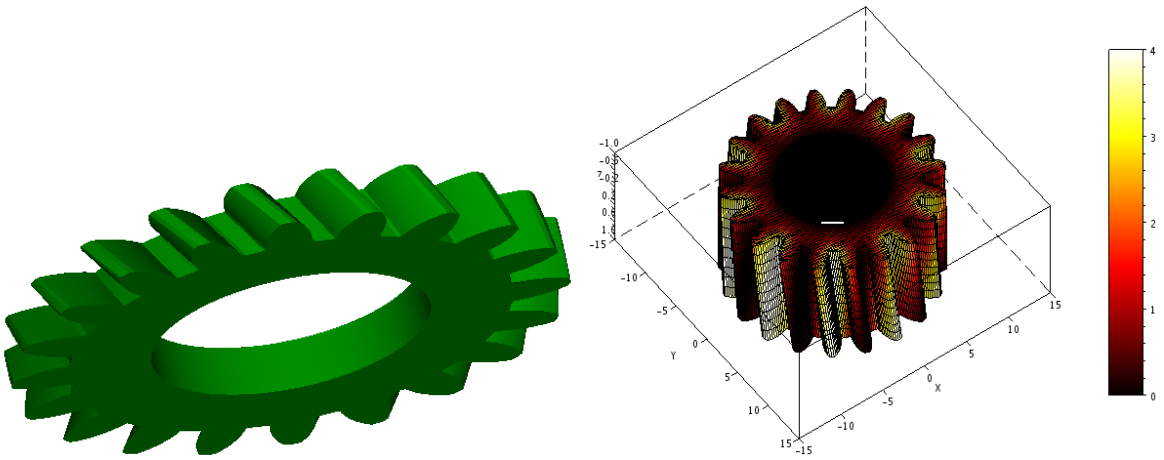

Figure 3. Dirichlet problem: the gear.

$$[K^e] = \iiint (\begin{Bmatrix} \frac{\partial H_1}{\partial x} \\ \frac{\partial H_2}{\partial x} \\ \dots \\ \frac{\partial H_8}{\partial x} \end{Bmatrix} \begin{bmatrix} \frac{\partial H_1}{\partial x} & \frac{\partial H_2}{\partial x} & \dots & \frac{\partial H_8}{\partial x} \end{bmatrix} + \begin{Bmatrix} \frac{\partial H_1}{\partial y} \\ \frac{\partial H_2}{\partial y} \\ \dots \\ \frac{\partial H_8}{\partial y} \end{Bmatrix} \begin{bmatrix} \frac{\partial H_1}{\partial y} & \frac{\partial H_2}{\partial y} & \dots & \frac{\partial H_8}{\partial y} \end{bmatrix} + \begin{Bmatrix} \frac{\partial H_1}{\partial z} \\ \frac{\partial H_2}{\partial z} \\ \dots \\ \frac{\partial H_8}{\partial z} \end{Bmatrix} \begin{bmatrix} \frac{\partial H_1}{\partial z} & \frac{\partial H_2}{\partial z} & \dots & \frac{\partial H_8}{\partial z} \end{bmatrix}) dxdydz, \tag{8}$$

where H_i are shape functions of elements.

Obtained from numerical experiment result are shown in the table 1.

Table 1. Dirichlet problem.

Step	**Number of cores**		
	1	2	4
	time, seconds		
The global stiffness matrix assembly	3.07639	1.56709	1.25949
The solution of linear equations	0.304023	0.240893	0.264904

4.2 Three-layer cylindrical shell under uniform load

Consider three-layer cylindrical shell under uniform load. The edge of the shell is fixed. Radius $r = 0.4$ m, thickness $h = 0.01$ m. Thickness of inner layer $h_i = 0.008$ m, Young's modulus of inner layer $E_i = 72017.3327$ MPa, Poisson's ratio $\nu_i = 0.29995185$. Thickness of outer layers $h_o = 0.001$ m, Young's modulus of outer layers $E_o = 203200$ MPA,Poisson's ratio $\nu_o = 0.27$. Uniform load (pressure on top of the shell) $q = -0.05$ MPa.

We built mesh of a sector of the shell (problem is symmetric, see 4) in 3D. We need to use 10 layers of finite elements (thickness of each layer 0.001 m) for modeling of shell structure. Obtained mesh consists of 59213 nodes (52240 finite elements).

Formulas for evaluation of local matrix $[K^e]$ described below.

$$[K^e] = \iiint B^T D B dxdydz, \tag{9}$$

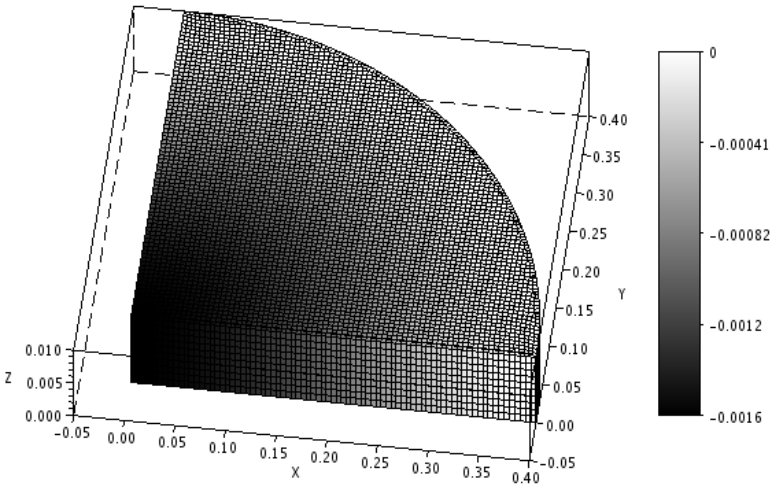

Figure 4. Three-layer shell: displacements.

$$B = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} H_1 & H_2 & \dots & H_8 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & H_1 & H_2 & \dots & H_8 & 0 & 0 & \dots & 0 \end{bmatrix}, \tag{10}$$

$$D = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 0 & 0 & 0 \\ \frac{\nu}{1-\nu} & 1 & \frac{\nu}{1-\nu} & 0 & 0 & 0 \\ \frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix} \tag{11}$$

$$E = \begin{cases} E_o, \text{ if } z \geq 0.009 \text{ or } z \leq 0.001, \\ E_i, \text{ if } 0.001 \leq z \leq 0.009, \end{cases} \quad \nu = \begin{cases} \nu_o, \text{ if } z \geq 0.009 \text{ or } z \leq 0.001, \\ \nu_i, \text{ if } 0.001 \leq z \leq 0.009, \end{cases} \tag{12}$$

Obtained from numerical experiment result are shown in the table 2.

Table 2. Three-layer shell.

Step	**Number of cores**		
	1	2	4
	time, seconds		
The global stiffness matrix assembly	83.7293	41.5063	31.7687
The solution of linear equations	298.287	191.299	189.134

5 Conclusion

Thus, numerical experiments show that parallel processing in the finite element allows to reduce time for modeling. In tables we can see not significant difference between two and four cores because tested CPU has two physical cores and two virtual cores (Hyper-Threading). In the article presented first step to develop parallel FEM processor. Next steps will be improved of FEM algorithms for parallel processing on ZNU's cluster. Another direction of improvement of algorithms is using GPU power in FEM processor. A perspective

direction of research is determination of an optimal number of parallel cores for finite element analysis on different parallel architectures (multiprocessors, multicomputers, clusters, etc.).

References

[1] O. Z. Zienkiewicz, R.L. Taylor: The Finite Element Method. V. 1: The Basis. Butterworth-Heinemann, Oxford, 2000.

[2] I. M. Smith, D. V. Griffits: Programming the finite element method. Wiley, Chichester, 2004.

[3] A. Ahmad: Parallel Programming in the Finite Element Method. *Proceedings of Failure of Engineering Materials and Structures*: 87-93, 2007.

[4] J. F. Thompson, B. K. Soni, N. P. Weatheril: Hand book of grid generation. CRC Press, New York, 1999.

[5] C. Vollaire, L. Nicolas, A. Nicolas: Parallel computing for the finite element method. *The European Physical Journal Applied Physics*, 1(3): 305-314, 1998.

[6] V. Kandasamy: Parallel FEM Simulation Using GPUs. *23rd European Conference Forum Bauinformatik*: 2011.

[7] P. K. Jimack, N. Touheed: Developing Parallel Finite Element Software Using MPI. *High Performance Computing for Computational Mechanics*: 15-38, 2000.

[8] F. Okulicka-Dłużewska: Parallelization of Finite Element Package by MPI Library. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2131: 427-436, 2001.

[9] S. V. Savic, A. Z. Ilic, B. M. Notaros, M. M. Ilic: Acceleration of higher order FEM matrix filling by OpenMP parallelization of volume integrations. *Telecommunications Forum (TELFOR)*: 1183-1184, 2012.

[10] O. Pantalé: Parallelization of an object-oriented FEM dynamics code: influence of the strategies on the Speedup. *Advances in Engineering Software*, 36(6): 361-373, 2005.

[11] G. Mahinthakumar, F. Saied: A Hybrid MPI-OpenMP Implementation of an Implicit Finite-Element Code on Parallel Architectures. *International Journal of High Performance Computing Applications*, 16(4): 371-393, 2002.

[12] M. Hayashi, K. Nakajima: OpenMP/MPI Hybrid Parallel ILU(k) Preconditioner for FEM Based on Extended Hierarchical Interface Decomposition for Multi-core Clusters. *High Performance Computing for Computational Science*, 7851: 278-291, 2013.

[13] M. Vargas-Félix, S. Botello-Rionda: Solution of Finite Element Problems Using Hybrid Parallelization with MPI and OpenMP. *Acta Universitaria*, 22(7): 14-24, 2012.

[14] G. Bencheva, S. Margenov, J. Stary: Parallel PCG solver for nonconforming FE problems: overlapping of communications and computations. *Springer LNCS*, 3743: 646-654, 2006.

[15] Y. Notay: Flexible Conjugate Gradients. *SIAM Journal on Scientific Computing*, 22(4): 1444-1460, 2001.

[16] A. V. Knyazev, I. Lashuk: Steepest Descent and Conjugate Gradient Methods with Variable Preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 29(4): 1267–1280, 2008.