COMPUTATIONAL SCIENCE AND TECHNIQUES
Volume 1, Number 2, 2013, 214-221

Publisher: Klaipėda University
http://journals.ku.lt/index.php/CST
Online ISSN: 2029-9966

PARALLEL COMPUTING FOR THE FINITE ELEMENT METHOD IN MATLAB

Aurimas Šimkus, Sigita Turskienė
Department of Informatics, Šiauliai University, P. Višinskio St. 19, LT-77156, Šiauliai
saurimas@hotmail.com, sigita@fm.su.lt

Abstract. In this research, parallel computing capabilities of MATLAB and the capabilities for the finite element method were analyzed. A program for solving a heat transfer problem by the finite element method was implemented. Three different parallel algorithms using CPU and GPU for solving steady state and transient heat transfer problems were proposed and implemented. A maximal speedup of around 2.3 times for steady state and 2 times for transient problem solving time was achieved by using a quad-core CPU.
Keywords: parallel computing, finite element method, heat transfer problem, MATLAB.

Introduction

While a complexity of engineering and scientific research problems is increasing, a need of computational resources of computers is increasing as well. An appropriate example for this case is the finite element method (FEM) that is widely used in engineering (Lewis, 2004; Sergerlind, 1976). Sequential computing of such problems may be too expensive even for modern computers if a fairly complex problem is needed to be solved. Parallel computations can often reduce a computation time even if a single multicore computer is used and increase an amount of available memory if a distributed system is used (Leopold, 2000). Furthermore, computer mathematic systems, such as MATLAB, that are oriented for solving mentioned problems, support parallel computing (Martin, Sharma, 2009).

Although some researches have been done in this field, most of it did not cover a comparative analysis of all capabilities for solving engineering problems by FEM in computer mathematic systems. Kuczmann (2011) analyzed solution of two-dimensional electrostatic field problem by diving the finite element mesh into equal sub-meshes for parallel assembly. Hosagrahara, Tamminana and Sharma (2010) analyzed MATLAB parallel computing capabilities for solving electrostatic field problem, however they applied only one parallelization strategy (parallel loop). Butrylo B. et al. (2004) analyzed parallel solvers for systems of linear equations in FEM for electromagnetic analysis. Cecka, Lew and Darve (2011) analyzed multiple strategies for assembly and solving sparse linear systems strategies in FEM by using Nvidia CUDA technology for GPU computing.

The main aim of the present paper is to analyze the parallel computing capabilities (CPU and GPU) for FEM in MATLAB by implementing some parallel approaches for solving the heat transfer problem.

Testing tools: MATLAB R2012b 64 bit, Parallel computing toolbox (PCT), quad-core Intel i7 3630QM, 8 GB DDR3, and Nvidia GT640M 2 GB with 384 CUDA cores.

1. Parallel computing concepts in MATLAB

Concepts of parallel computing have been reviewed in different scientific articles, comparing both CPU and GPU (Krušna, Denisov, 2013). Parallel computing in MATLAB can be described in some different ways depending on a type of the parallelism generalized in Figure 1 (Luszczek, 2009; Moler, 2007).

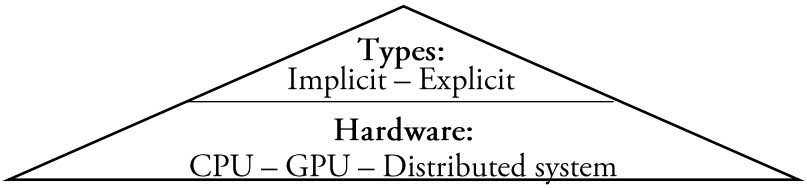

Figure 1. Parallelism in MATLAB.

Implicit parallel calculations are performed by MATLAB core-integrated parallel functions which use parallel computer resources directly and automatically. Explicit parallel constructs have to be formed explicitly by a programmer. These constructs are not included in the core of MATLAB – they are implemented in MATLAB toolboxes. In this research, Parallel Computing Toolbox (PCT) was used. GPU support is also implemented in PCT, and is based on Nvidia CUDA (Reese, Zaranek, 2011).

When using explicit PCT constructs (e.g., parfor or spmd), a pool of MATLAB sessions (matlabpool) for parallel computations should be initialized first. The pool uses a profile which describes the workspace where parallel computations are performed. A local profile for a single computer is set by default, but a distributed system (a cluster, a grid or a cloud) profile can be set if MATLAB Distributed Computing Server (MDSC) is used. Up to 12 MATLAB computational engines (workers) can be initialized per single computer in the pool (4 local workers were used for testing).

2. Parallel computing in the finite element method analysis

In FEM, two main components: matrix [K] and vector {F} are assembled before calculating the answer of a problem (a temperature vector {T} in the case of the heat transfer problem). It is called the assembly procedure where [K] is a global thermal conductivity matrix and {F} is a global thermal vector. They both are assembled from separate finite elements as shown in Figure 2 and are used in matrix equation

$$[K]\cdot\{T\}=\{F\}. \qquad (1)$$

$$[K]=\begin{bmatrix} K_{1,1}^e & K_{1,2}^e & 0 & 0 & 0 & 0 \\ K_{1,3}^e & \left[K_{1,4}^e+K_{2,1}^e\right] & K_{2,2}^e & 0 & 0 & 0 \\ 0 & K_{2,3}^e & \left[K_{2,4}^e+K_{3,1}^e\right] & \dots & 0 & 0 \\ 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \dots & \left[K_{NE-1,4}^e+K_{NE,1}^e\right] & K_{NE,2}^e \\ 0 & 0 & 0 & 0 & K_{NE,3}^e & K_{NE,4}^e \end{bmatrix} \quad \{F\}=\begin{Bmatrix} F_{1,1}^e \\ \left\{F_{1,2}^e+F_{2,1}^e\right\} \\ F_{2,2}^e+F_{3,1}^e \\ \dots \\ \left\{F_{NE-1,2}^e+F_{NE,1}^e\right\} \\ F_{NE,2}^e \end{Bmatrix}$$

Figure 2. Example of [K] and {F} assembly (NE – number of the finite elements)

According to Figure 2, global matrix [K] and vector {F} are assembled from separate elements {Fe} and [Ke] as well as their intersections. In the parallel part of this algorithm, non-overlapping parts of these elements can be included into the global components. After the parallel part, the intersections can be included as well. There are two possible types of matrix [K] in FEM that are implemented in MATLAB. One of them is a full matrix and another is a band (sparse) matrix. Only used (non-zero) elements are saved in the band matrix.

In our solved heat transfer problem, the band matrix was used; however, the full matrix can be used in some problems as well. Both types of matrices were compared by consumptions of computer memory and solving time using a double-precision data type.

Table 1. Comparison of using full and band matrices with 15000 finite elements.

Type of matrix [K]	Size of [K] (KB)	Size of {F} (KB)	(1) solving time (s)	Assembly time (s)
Full	175805	117	19,981	0,130
Band	820		0,002	0,142
Difference (times):	214	-	9990	0,915

According to Table 1, the band matrix obviously saves a lot of computer memory and so the problem is solved much quicker. Then, the matrix [K] and vector {F} assembly procedure has to be parallelly computed (this strategy is analyzed in our paper in Section 3). Though MATLAB implicit parallelism was revealed to be actually not working efficiently when the band matrix is used, it has a significant importance when the full matrix is used (Table 2).

Table 2. Effect of implicit MATLAB parallelism when the full matrix [K] is used.

	1 core	2 cores	3 cores	4 cores
10 000 FE (s)	16.5560	9.2100	7.3310	5.9560

3. Implementation of parallel algorithms

3.1. Parallel loop *parfor*

The most intuitive algorithm of the parallel assembly procedure was implemented by using a parallel loop construct *parfor*. This construct was also discussed in the paper by

Hosagrahara, Sharma and Tamminana (2010). Our implemented algorithm is shown in Figure 3.

As shown in Figure 3 and discussed in Section 2, each finite element consisting of {Fe} and [Ke] is computed separately in parallel loop iterations by independent MATLAB workers.

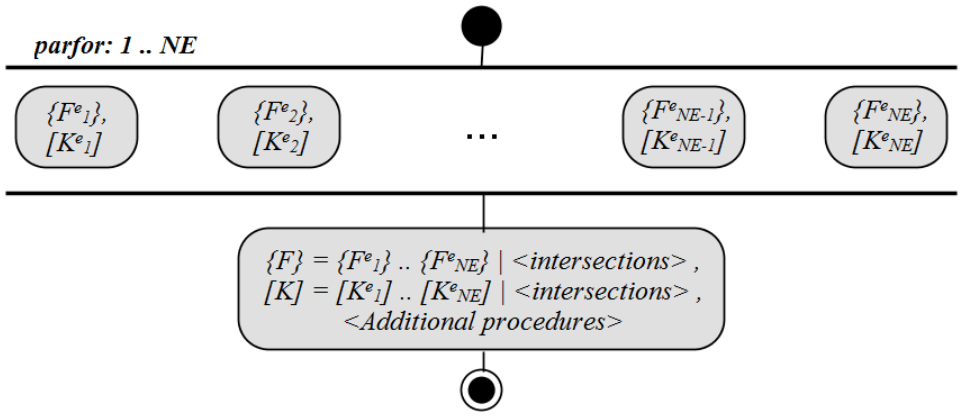

Figure 3. Algorithm for the parallel assembly procedure using the *parfor* loop.

3.2. Construct of *SPMD* model

The second algorithm for the parallel assembly procedure was implemented by using a *spmd* (Single Program Multiple Data computational model) construct (Figure 4).

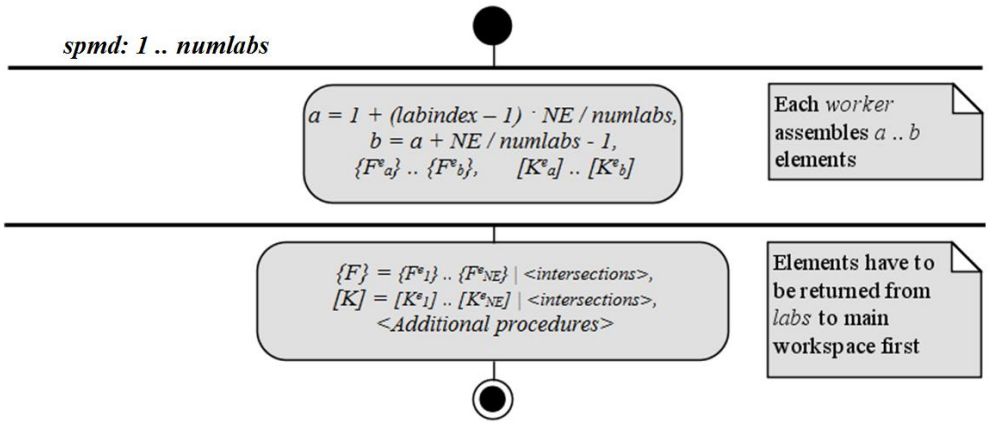

Figure 4. Algorithm for the parallel assembly procedure using the *spmd* construct.

In this algorithm, computations of the assembly procedure are divided in a number of MATLAB workers used. Each worker corresponds to a separate execution laboratory (*numlabs* is a number of the laboratories, *labindex* is an identification of the laboratory). Although *spmd* is less limited and has more control comparing to *parfor*, its composite data structure consumes more computer resources. On the other hand, it is designed to maintain data in distributed systems. It is possible to implement a similar procedure for the *parfor* construction if no such data is needed.

3.3. Element-wise function *arrayfun* for GPU

An element-wise function *arrayfun* was used for GPU-based algorithm (Figure 5). This function can be implemented for CPU as well, however, this case implementation has not produced positive results and is not discussed further.

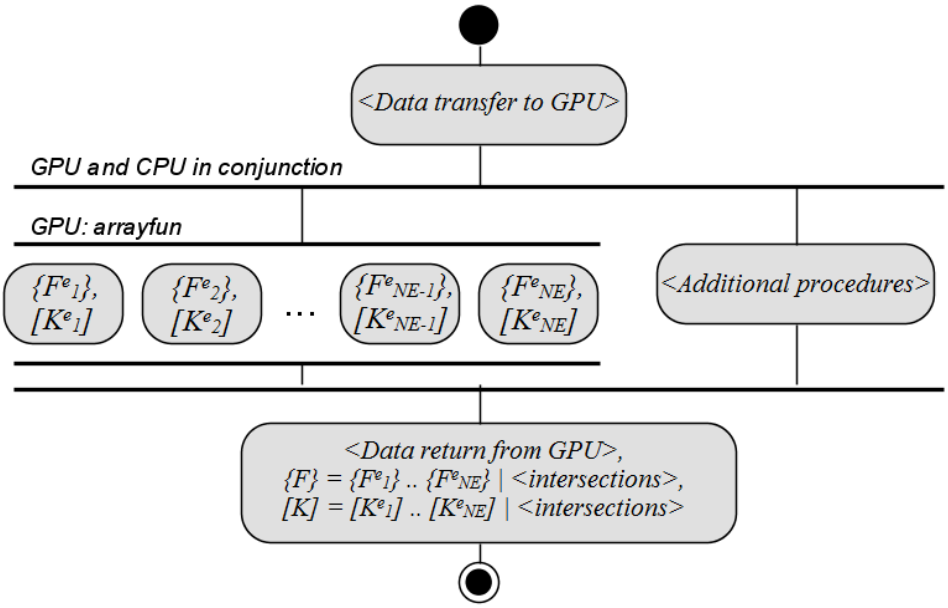

Figure 5. Algorithm for the parallel assembly procedure using *arrayfun* for GPU.

In *arrayfun*, the assembly procedure is applied to each finite element. Since GPU and CPU procedures are asynchronous, both of them may be computed in conjunction. a*rrayfun* for GPU also has a couple of significant limitations: only scalar data can be computed and it is CUDA-based only.

4. Numerical results and discussion

Steady state and transient heat transfer problems were solved for testing the implemented parallel assembly algorithms based on *parfor* and *spmd* (CPU), and *arrayfun* (GPU).

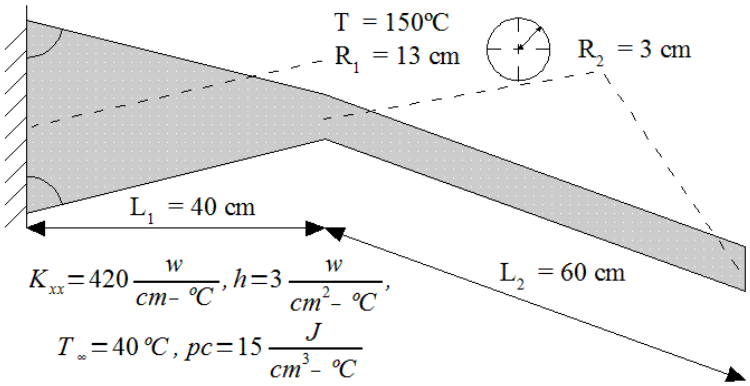

Figure 6. Model of the heat transfer problem.

A one-dimensional variable cross-sectional area finite element was used for the discretization of the structure shown in Figure 6. The temperature of the external environment T∞ was assumed to be changing in each time iteration in the transient problem, hence vector {F} has to be reassembled every time. Comparative results of speeding up the problem solving are further discussed in this section.

Steady state heat transfer problem solving times were measured by applying the implemented parallel assembly algorithms based on *parfor*, *spmd* and *arrayfun*. The speedup results are given in Figure 7.

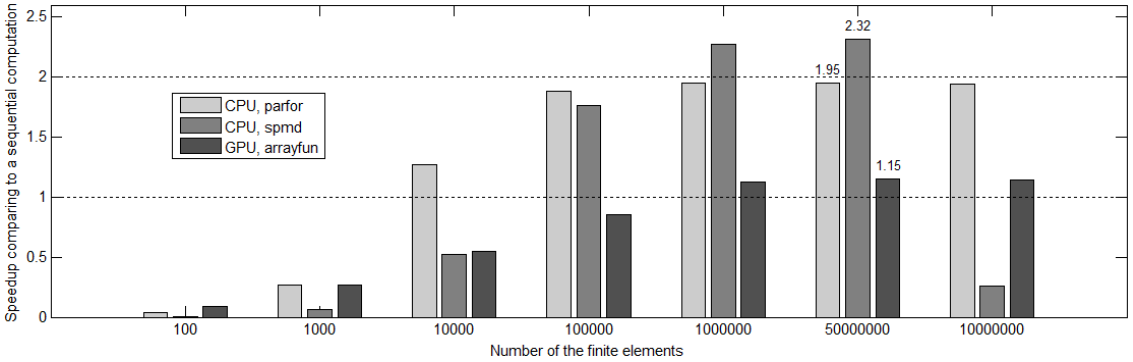

Figure 7. Comparison of the steady state problem solving times.

According to Figure 7, parallel assembly is inefficient for a small number of the finite elements (FE). However, it can reduce the problem solving time more than twice when the number of FE is big enough. Spmd-based algorithm failed for solving 10 million FE problem due to a lack of the computer memory, which proves the drawback of using composite data on a single computer. Because of limitations for solving FEM problem with GPU *arrayfun*, its parallel results are not very significant but positive.

Transient heat transfer problem solving time results are given in Figure 8. The implemented parallel assembly algorithms based on *parfor*, *spmd* and *arrayfun* were applied. A sequential Crank-Nicolson algorithm (Sergerlind, 1976, p. 218–219) was used for the time iterations.

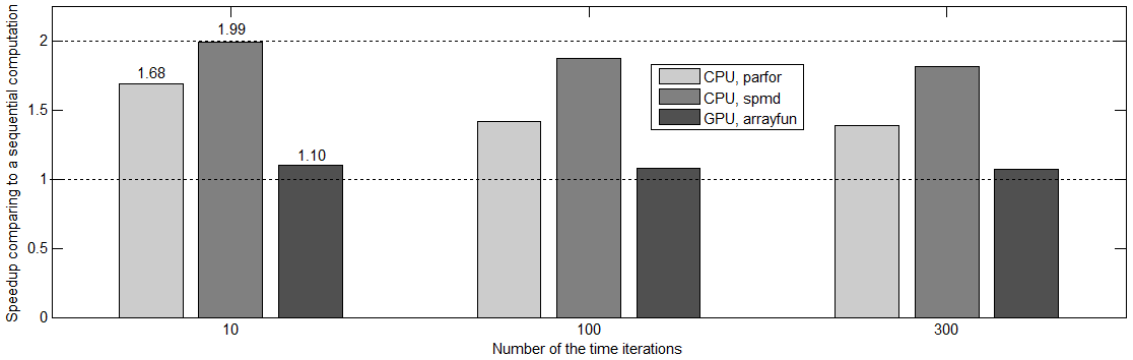

Figure 8. Comparison of the transient problem solving times for 1 million FE.

To solve the transient heat transfer problem, the heat capacity matrix [C] is needed. The time results are quite similar for different numbers of time iterations (Figure 8). It produces a slightly less speedup for more iterations, because matrices [K] and [C] are assembled only

once before iterations, and only vector {F} is reassembled over iterations. Maximal speedup of around twice was achieved, but it should be bigger for a more dynamic problem.

Results of the implemented algorithms were also tested for an analytically and Sergerlind's (1976, p. 144–146) solved heat transfer problem solution to satisfy their correctness. The solution of the algorithms converged similarly as solved in earlier research for this problem (Šimkus, Turskienė, 2013).

Conclusions

During the analysis of parallel computations for FEM, three MATLAB constructs were used for implementation of algorithms for parallel assembly: *parfor*, *spmd* for quad-core CPU, and *arrayfun* for GPU with 384 CUDA cores. The following conclusions correspond to the analysis and results of 1-D heat transfer problem.

1. Comparative results revealed that the fastest implementation is the *spmd*-based assembly algorithm with up to 2.3 times speedup for the steady state and up to 2 times for the transient heat transfer problem solving time.
2. The *parfor*-based algorithm is the most intuitive to implement with up to 1.95 times speedup for the steady state and up to 1.7 times for the transient problem solving time.
3. The GPU *arrayfun*-based algorithm is quite limited and not very effective (up to 1.15 times speedup for the steady state and up to 1.10 times for the transient problem solving time) for parallelizing the assembly procedure at the moment.

For the future research, the implemented algorithms should be tested on Grid or Cloud. More complex 2-D or 3-D finite element problems should be solved. A part of this research was presented (Šimkus, Turskienė, 2013).

References

Butrylo, B., Musy, F., Nicolas, L., Perrussel, R., Scorretti, R., Vollaire, C. (2004). A Survey of Parallel Solvers for the Finite Element Method in Computational Electromagnetics. The International Journal for Computation and Mathematics in Electrial and Electronic Engineering, 23 (2), 531–546.

Cecka, C., Lew, A. J., Darve, E. (2011). Assembly of the Finite Element Methods on Graphics Processors. International Journal for Numerical Methods in Engineering, 85 (5), 640–669.

Hosagrahara, V., Sharma, G., Tamminana, K. (2010). Accelerating Finite Element Analysis in MATLAB with Parallel Computing. The MathWorks News and Notes. Available from: http://www.mathworks.com/tagteam/66859_91826v00_FEM_final.pdf [Accessed: 22 July 2013].

Krušna, D., Denisov, V. (2013). General Purpose Parallel Programing Using New Generation Graphic Processors: CPU vs GPU Comparative Analysis and Opportunities Research. Computational Science and Techniques, 1 (1), 36–44.

Kuczmann, M. (2011). Parallel Finite Element Method. Przegląd Elektrotechniczny (Electrical Review), 12b, 100–102.

Leopold, C. (2000). Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches. New York: Wiley.

Lewis, R. W., Nithiarasu, P., Seetharamu, K. N., (2004). Fundamentals of the Finite Element Method for Heat and Fluid Flow. New York: Wiley.

Luszczek, P. (2009). Parallel Programming in MATLAB. *International Journal of High Performance Computing Applications*, 23 (3), 277–283.

Martin, J., Sharma, G. (2009). MATLAB: A Language for Parallel Computing. *International Journal of Parallel Programming*, 37 (1), 3–36.

Moler, C. (2007). Parallel MATLAB: Multiple Processors and Multiple Cores. The MathWorks News and Notes. Available from: http://www.mathworks.com/tagteam/42682_91467v00_NNR_Cleve_US.pdf [Accessed: 22 July 2013].

Reese, J., Zaranek, S. (2011). GPU Programming in MATLAB. The MathWorks News and Notes. Available from: http://www.mathworks.com/tagteam/74240_91967v01_gpu-programming-in-matlab.pdf [Accessed: 22 July 2013].

Sergerlind, L. J. (1976). Applied Finite Element Analysis. New York: Wiley.

Šimkus, A., Turskienė, S. (2013). Comparative Analysis of Possibilities for Object-oriented Programming in MATLAB Language. Jaunųjų mokslininkų darbai (Journal of Young Scientists), 39 (1), 148–151.

Šimkus, A., Turskienė, S. (2013). Parallel Computing for Finite Element Method in MATLAB. In: NorduGrid 2013, Distributed Systems and Big Data – Towards New Horizons, 4–6 June 2013, Šiauliai, Lithuania. Available from: http://indico.hep.lu.se//contributionDisplay.py?contribId=17&sessionId=0&confId=1273 [Accessed: 22 July 2013].

A. Šimkus is a Bachelor of Computer Science currently seeking a Master's Degree in Software Engineering at Vilnius University in Lithuania. His main research field is in computer modeling and parallel computing.

S. Turskienė is a Doctor of Technology Science and an Associate Professor of the Department of Informatics at Šiauliai University in Lithuania. Her main research field is in computer modelling of systems.

LYGIAGRETIEJI SKAIČIAVIMAI BAIGTINIŲ ELEMENTŲ METODUI MATLAB SISTEMA

Aurimas Šimkus, Sigita Turskienė

Santrauka

Šiame darbe buvo išanalizuotos lygiagrečių skaičiavimų galimybės baigtinių elementų metodui MATLAB sistema. Sukurta programa šilumos pernešimo uždaviniui spręsti baigtinių elementų metodu. Pasiūlyti ir realizuoti trys skirtingi algoritmai skirti CPU ir GPU lygiagretinimui sprendžiant stacionaraus ir nestacionaraus šilumos pernešimo uždavinius. Maksimalus stacionaraus proceso pagreitinimas iki 2.3 karto, nestacionaraus – iki 2 kartų buvo pasiektas naudojant keturių branduolių procesorių.

Pagrindiniai žodžiai: lygiagretieji skaičiavimai, baigtinių elementų metodas, šilumos pernešimo uždavinys, MATLAB sistema.