Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE
Informatyka w Technologii Materiałów

Vol. 17, 2017, No. 1

OPEN SOURCE JAVA IMPLEMENTATION OF THE PARALLEL MULTI-THREAD ALTERNATING DIRECTION ISOGEOMETRIC L2 PROJECTIONS SOLVER FOR MATERIAL SCIENCE SIMULATIONS

GRZEGORZ GURGUL[1], MACIEJ WOŹNIAK[1], MARCIN ŁOŚ[1],
DANUTA SZELIGA[2], MACIEJ PASZYŃSKI[1]*

[1] *Department of Computer Science AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland,*
[2] *Department of Applied Computer Science and Modeling, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland*
**Corresponding author: paszynsk@agh.edu.pl*

Abstract

This paper describes multi-thread parallel open source JAVA implementation of alternating directions isogeometric L2 projections solver. The solver enables for fast numerical simulations of time-dependent problems. To apply our solver, the time-dependent problem must be discretized using isogeometric finite element method with B-spline basis functions in the spatial domain. The problem is solved using an explicit method with respect to time. The application of the explicit method with B-spline based spatial discretization results in a sequence of isogeometric L2 projections that can be solved using our fast solver. The computational cost of the solution of either 2D or 3D problem is linear O(N) in every time step. This cost is lower than the cost of traditional multi-frontal solvers, delivering O($N^{1.5}$) computational cost for 2D problems and O(N^2) computational cost for 3D problems. This cost is also lower from any iterative solver, delivering O(Nk) computational cost, where k is the number of iterations, which depends on the particular iterative solver algorithm, and the k parameter is also a function of N, so cannot be ignored. Our algorithm is used for numerical solution of 3D elasticity problem.

Key words: Alternating directions solver, Isogeometric L2 projections, Material science eling

1. INTRODUCTION

The Alternating Directions Implicit (ADI) solver has been originally introduced in (Paceman et al., 1956; Wachspress & Habetler, 1960; Birkhoff et al., 1962) to solve parabolic, hyperbolic and elliptic PDE-s with finite difference method. The ADI solver has been re-invented as a tool for the fast solution of isogeometric L2 projections for explicit dynamics in regular (Gao & Calo, 2014), as well complicated geometries (Gao & Calo, 2015). The isogeometric finite element method L2 projections utilize B-spline functions as a basis for the approximation. The ADI solver can be applied for solution of non-stationary problems discretized with B-spline basis functions. This is because such the time-dependent problem solved with an explicit method is reduced to a sequence of isogeometric L2 projections. We developed a parallel, implementation of the ADI solver in Fortran plus MPI for distributed memory parallel machines (Woźniak et al., 2016) and applied it to the solution of the non-linear flow in heterogeneous media (Alotaibi et al., 2015). We have also utilized a sequential Fortran based implementation for the solution of the heat transfer or linear elasticity prob-

ISSN 1641-8581

lems, solved with Newmark explicit dynamics scheme (Łoś et al., 2015).

In this paper, we focus on parallel multi-thread JAVA implementation of the ADI solver. This is an alternative implementation targeting shared memory parallel machines. The main advantage of the ADI method is a linear O(*N*) computational cost of both 2D and 3D isogeometric L2 projections. This is a huge reduction of the computational cost with respect to both traditional multi-frontal direct solvers (Duff & Reid, 1983; Duff & Reid, 1984; Geng et al., 2006) and iterative solvers (Saad, 2003). Namely, the multi-frontal solvers for isogeometric finite element method have computational cost of the order of O($N^{1.5}p^3$) for 2D problems and O(N^2p^3) for 3D problems (Collier et al., 2015). Iterative solvers, in turn, have the computational cost of the order of O(*Nk*) for a single time step, where *k* is the number of iterations, depending on the method (Saad, 2003), and *k* is also a function of *N,* so cannot be ignored.

2. ALTERNATING DIRECTION ISOGEOMETRIC L2 PROJECTIONS METHOD

In this chapter, we describe the alternating direction solver for the fast solution of the isogeometric L2 projections over a 2D mesh of size $N_x * N_y$. The solver reduces the 2D L^2 projection problem into two one-dimensional problems with multiple right-hand sides. The projection problem can be summarized as

$$\min \left\| \sum_{i=1} b_i B_i - f \right\|_{L^2}$$

(expressing a given function *f* as a linear combination of B-spline basis functions span over the 2D mesh. This problem is equivalent to the solution of the following system of linear equations (which results from the orthogonality condition of the projection in L2 norm)

$$\begin{bmatrix} \int_\Omega \left(B_1^y B_1^x\right)\left(B_1^y B_1^x\right) & \cdots & \int_\Omega \left(B_1^y B_1^x\right)\left(B_{N_y}^y B_{N_x}^x\right) \\ \vdots & \ddots & \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)\left(B_1^y B_{1_x}^x\right) & \cdots & \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)\left(B_{N_y}^y B_{N_x}^x\right) \end{bmatrix}$$

$$\begin{bmatrix} b_{1,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} = \begin{bmatrix} \int_\Omega \left(B_1^y B_1^x\right) f \\ \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right) f \end{bmatrix} \tag{1}$$

$$\begin{pmatrix} \begin{bmatrix} \int_\Omega \left(B_1^y B_1^x\right)\left(B_1^y B_1^x\right) & \cdots & \int_\Omega \left(B_1^y B_1^x\right)\left(B_1^y B_{N_x}^x\right) \\ \vdots & \ddots & \vdots \\ \int_\Omega \left(B_1^y B_{N_x}^x\right)\left(B_1^y B_{1_x}^x\right) & \cdots & \int_\Omega \left(B_1^y B_{N_x}^x\right)\left(B_1^y B_{N_x}^x\right) \end{bmatrix} & \cdots & \begin{bmatrix} \int_\Omega \left(B_1^y B_1^x\right)\left(B_{N_y}^y B_1^x\right) & \cdots & \int_\Omega \left(B_1^y B_1^x\right)\left(B_{N_y}^y B_{N_x}^x\right) \\ \vdots & \ddots & \vdots \\ \int_\Omega \left(B_1^y B_{N_x}^x\right)\left(B_{N_y}^y B_{1_x}^x\right) & \cdots & \int_\Omega \left(B_1^y B_{N_x}^x\right)\left(B_{N_y}^y B_{N_x}^x\right) \end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix} \int_\Omega \left(B_{N_y}^y B_1^x\right)\left(B_{N_y}^y B_1^x\right) & \cdots & \int_\Omega \left(B_{N_y}^y B_1^x\right)\left(B_1^y B_{N_x}^x\right) \\ \vdots & \ddots & \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)\left(B_{N_y}^y B_{1_x}^x\right) & \cdots & \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)\left(B_1^y B_{N_x}^x\right) \end{bmatrix} & \cdots & \begin{bmatrix} \int_\Omega \left(B_{N_y}^y B_1^x\right)\left(B_{N_y}^y B_1^x\right) & \cdots & \int_\Omega \left(B_{N_y}^y B_1^x\right)\left(B_{N_y}^y B_{N_x}^x\right) \\ \vdots & \ddots & \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)\left(B_{N_y}^y B_{1_x}^x\right) & \cdots & \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)\left(B_{N_y}^y B_{N_x}^x\right) \end{bmatrix} \end{pmatrix} \tag{2}$$

The size of the system is equal to the 2D mesh size. This system can be expressed in the following way, where we emphasize the local block numbering of B-spline basis functions.

$$\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y 1,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} \int_\Omega \left(B_1^y B_1^x\right) f \\ \vdots \\ \int_\Omega \left(B_1^y B_{N_x}^x\right) f \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \int_\Omega \left(B_{N_y}^y B_1^x\right) f \\ \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right) f \end{bmatrix} \end{pmatrix} \tag{3}$$

The two-dimensional B-spline basis functions are tensor products of two one-dimensional B-splines from one-dimensional basis along *x*- and *y*-axis $B^x = \{B_1, \cdots, B_{N_x}\}$ and $By = \{B_1, \cdots, B_{N_y}\}$.

We notice that the integral can be decomposed into the multiplication of two integrals

$$\int_\Omega g_1(x) g_2(y) = \int_x \int_y g_1(x) g_2(y) = \int_x g_1(x) \int_y g_2(y) \tag{4}$$

which implies

$$\left(\begin{matrix}\begin{bmatrix}\int_y B_1^y B_1^y \int_x B_1^x B_1^x & \cdots & \int_y B_1^y B_1^y \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_y B_1^y B_1^y \int_x B_{N_x}^x B_1^x & \cdots & \int_y B_1^y B_1^y \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix} & \cdots & \begin{bmatrix}\int_y B_1^y B_{N_y}^y \int_x B_1^x B_1^x & \cdots & \int_y B_1^y B_{N_y}^y \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_y B_1^y B_{N_y}^y \int_x B_{N_x}^x B_1^x & \cdots & \int_y B_1^y B_{N_y}^y \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix}\int_y B_{N_y}^y B_1^y \int_x B_1^x B_1^x & \cdots & \int_y B_{N_y}^y B_1^y \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y \int_x B_{N_x}^x B_1^x & \cdots & \int_y B_{N_y}^y B_1^y \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix} & \cdots & \begin{bmatrix}\int_y B_{N_y}^y B_{N_y}^y \int_x B_1^x B_1^x & \cdots & \int_y B_{N_y}^y B_{N_y}^y \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_{N_y}^y \int_x B_{N_x}^x B_1^x & \cdots & \int_y B_{N_y}^y B_{N_y}^y \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix}\end{matrix}\right)\left(\begin{matrix}\begin{bmatrix}b_{1,1}\\ \vdots \\ b_{1,N_x}\end{bmatrix}\\ \vdots \\ \begin{bmatrix}b_{N_y,1}\\ \vdots \\ b_{N_y,1,N_x}\end{bmatrix}\end{matrix}\right)=\left(\begin{matrix}\begin{bmatrix}\int_\Omega \left(B_1^y B_1^x\right)f\\ \vdots \\ \int_\Omega \left(B_1^y B_{N_x}^x\right)f\end{bmatrix}\\ \vdots \\ \begin{bmatrix}\int_\Omega \left(B_{N_y}^y B_1^x\right)f\\ \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)f\end{bmatrix}\end{matrix}\right) \quad (5)$$

Now comes the essential observation, that our matrix with integrals from B-splines along x- and y-axis can be expressed as the multiplication of two matrices, where in the first one we only have B-splines along x-axis, and in the second one we only have B-splines along y-axis:

$$\left(\begin{matrix}\begin{bmatrix}\int_x B_1^x B_1^x & \cdots & \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_x B_{N_x}^x B_1^x & \cdots & \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix}\int_x B_1^x B_1^x & \cdots & \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_x B_{N_x}^x B_1^x & \cdots & \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix}\end{matrix}\right)$$

$$*\left(\begin{matrix}\begin{bmatrix}\int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_1^y \\ \vdots & \ddots & \vdots \\ \int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_1^y\end{bmatrix} & \cdots & \begin{bmatrix}\int_y B_1^y B_{N_y}^y & \cdots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_1^y B_{N_y}^y & \cdots & \int_y B_1^y B_{N_y}^y\end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix}\int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_1^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_1^y\end{bmatrix} & \cdots & \begin{bmatrix}\int_y B_{N_y}^y B_{N_y}^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_{N_y}^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y\end{bmatrix}\end{matrix}\right)\left(\begin{matrix}\begin{bmatrix}b_{1,1}\\ \vdots \\ b_{1,N_x}\end{bmatrix}\\ \vdots \\ \begin{bmatrix}b_{N_y,1}\\ \vdots \\ b_{N_y,1,N_x}\end{bmatrix}\end{matrix}\right)=\left(\begin{matrix}\begin{bmatrix}\int_\Omega \left(B_1^y B_1^x\right)f\\ \vdots \\ \int_\Omega \left(B_1^y B_{N_x}^x\right)f\end{bmatrix}\\ \vdots \\ \begin{bmatrix}\int_\Omega \left(B_{N_y}^y B_1^x\right)f\\ \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)f\end{bmatrix}\end{matrix}\right) \quad (6)$$

Now, we multiply the left-hand side by the matrix with the inverse of the diagonal blocks. We use the fact that

$$\begin{pmatrix}A & & \\ & \ddots & \\ & & A\end{pmatrix}\begin{pmatrix}A^{-1} & & \\ & \ddots & \\ & & A^{-1}\end{pmatrix}=\begin{pmatrix}I & & \\ & \ddots & \\ & & I\end{pmatrix} \quad (7)$$

Which implies the following system of linear equations

$$\left(\begin{matrix}\begin{bmatrix}\int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_1^y \\ \vdots & \ddots & \vdots \\ \int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_1^y\end{bmatrix} & \cdots & \begin{bmatrix}\int_y B_1^y B_{N_y}^y & \cdots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_1^y B_{N_y}^y & \cdots & \int_y B_1^y B_{N_y}^y\end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix}\int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_1^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_1^y\end{bmatrix} & \cdots & \begin{bmatrix}\int_y B_{N_y}^y B_{N_y}^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_{N_y}^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y\end{bmatrix}\end{matrix}\right)*$$

$$*\begin{pmatrix}\begin{bmatrix} b_{1,1}\\ \vdots \\ b_{1,N_x}\end{bmatrix}\\ \vdots \\ \begin{bmatrix} b_{N_y,1}\\ \vdots \\ b_{N_y1,N_x}\end{bmatrix}\end{pmatrix}=\begin{pmatrix}\begin{bmatrix}\int_x B_1^x B_1^x & \cdots & \int_x B_1^x B_{N_x}^x\\ \vdots & \ddots & \vdots\\ \int_x B_{N_x}^x B_1^x & \cdots & \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix}^{-1} & & \\ & \ddots & \\ & & \begin{bmatrix}\int_x B_1^x B_1^x & \cdots & \int_x B_1^x B_{N_x}^x\\ \vdots & \ddots & \vdots\\ \int_x B_{N_x}^x B_1^x & \cdots & \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix}^{-1}\end{pmatrix}\begin{pmatrix}\begin{bmatrix}\int_\Omega \left(B_1^y B_1^x\right)f\\ \vdots \\ \int_\Omega \left(B_1^y B_{N_x}^x\right)f\end{bmatrix}\\ \vdots \\ \begin{bmatrix}\int_\Omega \left(B_{N_y}^y B_1^x\right)f\\ \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)f\end{bmatrix}\end{pmatrix}=$$

$$=\begin{pmatrix}\begin{bmatrix}\int_x B_1^x B_1^x & \cdots & \int_x B_1^x B_{N_x}^x\\ \vdots & \ddots & \vdots\\ \int_x B_{N_x}^x B_1^x & \cdots & \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix}^{-1}\begin{bmatrix}\int_\Omega \left(B_1^y B_1^x\right)f\\ \vdots \\ \int_\Omega \left(B_1^y B_{N_x}^x\right)f\end{bmatrix}\\ \vdots \\ \begin{bmatrix}\int_x B_1^x B_1^x & \cdots & \int_x B_1^x B_{N_x}^x\\ \vdots & \ddots & \vdots\\ \int_x B_{N_x}^x B_1^x & \cdots & \int_x B_{N_x}^x B_{N_x}^x\end{bmatrix}^{-1}\begin{bmatrix}\int_\Omega \left(B_{N_y}^y B_1^x\right)f\\ \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right)f\end{bmatrix}\end{pmatrix}=\begin{pmatrix}\begin{bmatrix} t_{1,1}\\ \vdots \\ t_{1,N_x}\end{bmatrix}\\ \vdots \\ \begin{bmatrix} t_{N_y,1}\\ \vdots \\ t_{N_y,N_x}\end{bmatrix}\end{pmatrix} \tag{8}$$

Now, we perform the re-ordering in the block system

$$\begin{pmatrix}\begin{bmatrix} b_{1,1}\\ \vdots \\ b_{1,N_x}\end{bmatrix}\\ \vdots \\ \begin{bmatrix} b_{N_y,1}\\ \vdots \\ b_{N_y,N_x}\end{bmatrix}\end{pmatrix}\rightarrow\begin{pmatrix}\begin{bmatrix} b_{1,1}\\ \vdots \\ b_{N_y,1}\end{bmatrix}\\ \vdots \\ \begin{bmatrix} b_{1.N_x}\\ \vdots \\ b_{N_y,N_x}\end{bmatrix}\end{pmatrix}\qquad\begin{pmatrix}\begin{bmatrix} t_{1,1}\\ \vdots \\ t_{1,N_x}\end{bmatrix}\\ \vdots \\ \begin{bmatrix} t_{N_y,1}\\ \vdots \\ t_{N_y,N_x}\end{bmatrix}\end{pmatrix}\rightarrow\begin{pmatrix}\begin{bmatrix} t_{1,1}\\ \vdots \\ t_{N_y,1}\end{bmatrix}\\ \vdots \\ \begin{bmatrix} t_{1.N_x}\\ \vdots \\ t_{N_y,N_x}\end{bmatrix}\end{pmatrix} \tag{9}$$

to get

$$\begin{pmatrix}\begin{bmatrix}\int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_{N_y}^y\\ \vdots & \ddots & \vdots\\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y\end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix}\int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_{N_y}^y\\ \vdots & \ddots & \vdots\\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y\end{bmatrix}\end{pmatrix}\begin{pmatrix}\begin{bmatrix} b_{1,1}\\ \vdots \\ b_{N_y,1}\end{bmatrix}\\ \vdots \\ \begin{bmatrix} b_{1,N_x}\\ \vdots \\ b_{N_y,N_x}\end{bmatrix}\end{pmatrix}=\begin{pmatrix}\begin{bmatrix} t_{1,1}\\ \vdots \\ t_{N_y,1}\end{bmatrix}\\ \vdots \\ \begin{bmatrix} t_{1,N_x}\\ \vdots \\ t_{N_y,N_x}\end{bmatrix}\end{pmatrix} \tag{10}$$

which implies

$$\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{1,N_x} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} \int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y \end{bmatrix}^{-1} \begin{bmatrix} t_{1,1} \\ \vdots \\ t_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y \end{bmatrix}^{-1} \begin{bmatrix} t_{1,N_x} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix} \end{pmatrix} \tag{11}$$

We denote

$$\begin{pmatrix} \begin{bmatrix} d_{1,1} \\ \vdots \\ d_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} d_{1,N_x} \\ \vdots \\ d_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} \int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y \end{bmatrix}^{-1} \begin{bmatrix} t_{1,1} \\ \vdots \\ t_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y \end{bmatrix}^{-1} \begin{bmatrix} t_{1,N_x} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix} \end{pmatrix}$$

and reorder $\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{1.N_x} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix}$ back to $\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix}$ to get finally $\begin{pmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,N_x} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} b_{N_y,1} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} d_{1,1} \\ \vdots \\ d_{N_y,1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} d_{1.N_x} \\ \vdots \\ d_{N_y,N_x} \end{bmatrix} \end{pmatrix}$.

The practical consequence from these algebraic transformations are the following. The two-dimensional projection problem (1) can be replaced by two one-dimensional projection problems with multiple right-hand sides:

$$\begin{bmatrix} \int_x B_1^x B_1^x & \cdots & \int_x B_1^x B_{N_x}^x \\ \vdots & \ddots & \vdots \\ \int_x B_{N_x}^x B_1^x & \cdots & \int_x B_{N_x}^x B_{N_x}^x \end{bmatrix} \begin{bmatrix} t_{1,1} \\ \vdots \\ t_{1,N_x} \end{bmatrix} \cdots \begin{bmatrix} t_{N_y,1} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix} = \begin{bmatrix} \int_\Omega \left(B_1^y B_1^x\right) f \\ \vdots \\ \int_\Omega \left(B_1^y B_{N_x}^x\right) f \end{bmatrix} \cdots \begin{bmatrix} \int_\Omega \left(B_{N_y}^y B_1^x\right) f \\ \vdots \\ \int_\Omega \left(B_{N_y}^y B_{N_x}^x\right) f \end{bmatrix} \tag{12}$$

$$\begin{bmatrix} \int_y B_1^y B_1^y & \cdots & \int_y B_1^y B_{N_y}^y \\ \vdots & \ddots & \vdots \\ \int_y B_{N_y}^y B_1^y & \cdots & \int_y B_{N_y}^y B_{N_y}^y \end{bmatrix} \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{N_y,1} \end{bmatrix} \cdots \begin{bmatrix} b_{1,N_x} \\ \vdots \\ b_{N_y,N_x} \end{bmatrix} = \begin{bmatrix} t_{1,1} \\ \vdots \\ t_{N_y,1} \end{bmatrix} \cdots \begin{bmatrix} t_{1,N_x} \\ \vdots \\ t_{N_y,N_x} \end{bmatrix} \tag{13}$$

The nice feature of the one-dimensional projection problems is that the matrices are banded. Namely, for the B-splines of order p they have $2p+1$ diagonals. This means that they can be factorized in computational cost linear $O(N_x)$ and $O(N_y)$, respectively. We have N_y right-hand sides for the first problem, and N_x for the second one. Total cost is either $O(N_x*N_y)$ or $O(N_y*N_x)$ which is equal to $O(N)$ where N is the number of basis functions over the 2D mesh.

3. EXPRESSING OF THE SOLVER ALGORITHM BY GRAPH GRAMMAR PRODUCTIONS

In general, the computational problem, in here the factorization of the banded matrix followed by backward substitution, can be represented as a set of tasks, and the dependency graph is plotted between them, compare figure 1. For example, for the one-dimensional multi-frontal direct solver for isogeometric finite element method with quadratic B-splines, we define the following tasks, compare chapter 3.2 in (Paszyński, 2016). First, we define the tasks responsible for construction of the elimination tree

[(P1)]

[(P2)1, (P2)2]

[(P3)1, (P3)2, (P3)3, (P3)4]

Next, we define tasks responsible for generation of element frontal matrices

[(A1)(A)1(A)2(A)3(A)4(A)5(A)6(A)7(A)8(A)9(A)10(A)11(AN)]

We merge three element local matrices to have one row fully assembled

[(A2_3)1(A2_3)2(A2_3) 3(A2_3)4]

This row corresponds to the single B-spline that spreads over three neighbors elements.

We can eliminate the degrees of freedom related to this fully assembled B-spline

[(E2_1_5) 1(E2_1_5)2(E2_1_5)3(E2_1_5)4]

solver for isogeometric finite element method with multiple right-hand sides, call it once, reorder the result, and call it second time. Having the solver algorithm decomposed into basic undividable tasks that can be executed in parallel, we need to design a scheduler for concurrent executions of sets of parallel tasks.

Following (Paszyński, 2016) we introduce the following JAVA based scheduler, executing the groups of independent tasks, one set after another, in concurrent.

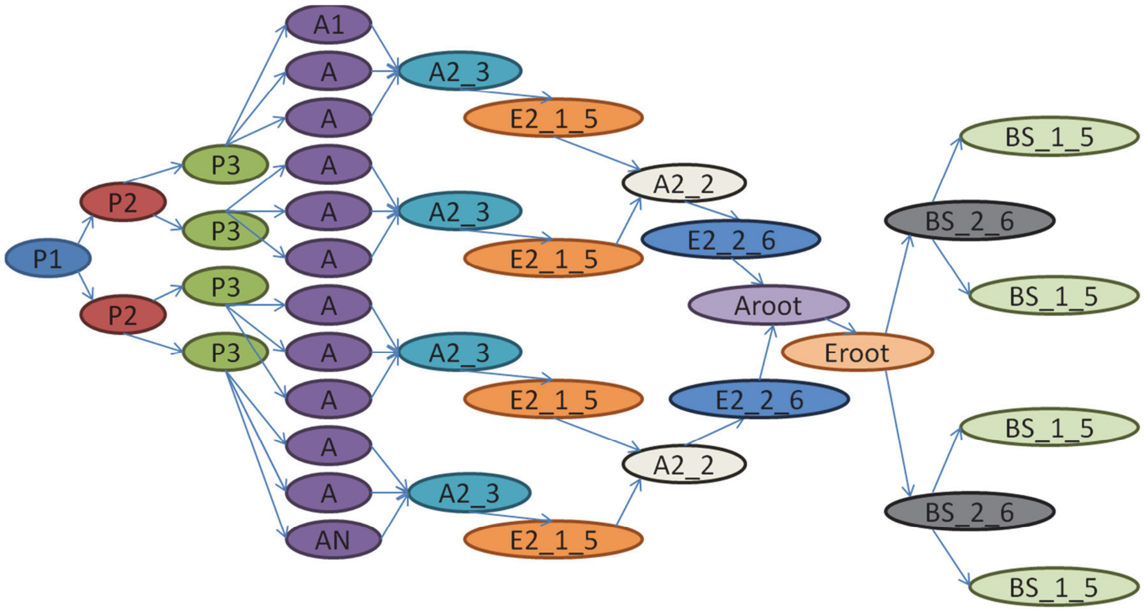

Fig. 1. *The tasks representing the generation of the element partition tree (P1,P2,P3), the generation of element matrices (A1,A,AN), the merging of frontal matrices at parent level (A2_3, A2_2), elimination of the fully assembled degrees of freedom (E2_1_5,E2_2_6), followed by the constructing and solving root problem (Aroot, Eroot), and finally backward substitutions (BS_2_6, BS_1_5).*

Next, we assemble the resulting pairs frontal matrices

[(A2_2)1(A2_2)2]

and eliminate two degrees of freedom related to the two fully assembled B-splines

[(E2_2_6)1(E2_2_6)2].

We formulate and solve the root problem

[(Eroot)]

[(Aroot)]

and we perform backward substitutions

[(BS_2_6)1(BS_2_6)2]

[(BS_1_5)1(BS_1_5)2(BS_1_5)3(BS_1_5)4]

Thus, the parallel execution of the solver consists of several steps, where independent tasks can be executed in concurrent, interchanged with the synchronization barriers.

4. MULTI-THREAD JAVA IMPLEMENTATION

Basically, to implement the ADI solver it is necessary to implement one-dimensional multi-frontal

```
class Executor extends Thread {
public synchronized void run() {
// BUILDING ELEMENT PARTITION TREE
try {
int n = 12; // number of elements
      along x axis
int m = 12; // number of elements
      along y axis
double dx = 1.0 / n; // mesh size
      along x
double dy = 1.0 / m; // mesh size
      along y
int p = 2; // polynomial order of ap-
      proximation
// Mesh
MeshData mesh = new MeshData(dx, dy,
      n, m, p);
Vertex S = new Vertex ("S", mesh);
// Build tree along x
// [(P1)]
CyclicBarrier barrier = new CyclicBar-
      rier(1 + 1);
P1 p1 = new P1(S, barrier, mesh);
p1.start();
barrier.await();
```

```
// [(P2)1(P2)2]
barrier = new CyclicBarrier(2 + 1);
P2 p2a = new P2(p1.m_vertex.m_left,
     barrier, mesh);
P2 p2b = new P2(p1.m_vertex.m_right,
     barrier, mesh);
p2a.start(); p2b.start();
barrier.await();
// [(P3)1(P3)2(P3)3(P3)4]
barrier = new CyclicBarrier(4 + 1);
P3 p3a1 = new P3(p2a.m_vertex.m_left,
     barrier, mesh);
P3 p3a2 = new P3(p2a.m_vertex.m_right,
     barrier, mesh);
P3 p3b1 = new P3(p2b.m_vertex.m_left,
     barrier, mesh);
P3 p3b2 = new P3(p2b.m_vertex.m_right,
     barrier, mesh);
p3a1.start(); p3a2.start();
p3b1.start(); p3b2.start();
barrier.await();
// MFS along x
// [A^12]
barrier = new CyclicBarrier(12 + 1);
A1 a1 = new A1(p3a1.m_vertex.m_left,
     barrier, mesh);
A a2 = new A(p3a1.m_vertex.m_middle,
     barrier, mesh);
A a3 = new A(p3a1.m_vertex.m_right,
     barrier, mesh);
A a4 = new A(p3a2.m_vertex.m_left,
     barrier, mesh);
A a5 = new A(p3a2.m_vertex.m_middle,
     barrier, mesh);
A a6 = new A(p3a2.m_vertex.m_right,
     barrier, mesh);
A a7 = new A(p3b1.m_vertex.m_left,
     barrier, mesh);
A a8 = new A(p3b1.m_vertex.m_middle,
     barrier, mesh);
A a9 = new A(p3b1.m_vertex.m_right,
     barrier, mesh);
A a10 = new A(p3b2.m_vertex.m_left,
     barrier, mesh);
A a11 = new A(p3b2.m_vertex.m_middle,
     barrier, mesh);
AN a12 = new AN(p3b2.m_vertex.m_right,
     barrier, mesh);
a1.start();a2.start();a3.start();a4.st
     art();
a5.start();a6.start();a7.start();a8.st
     art();
a9.start();a10.start();a11.start();a12
     .start();
barrier.await();
// [A2_3^4]
barrier = new CyclicBarrier(4 + 1);
A2_3 a2a = new A2_3(p3a1.m_vertex,
     barrier, mesh);
A2_3 a2b = new A2_3(p3a2.m_vertex,
     barrier, mesh);
A2_3 a2c = new A2_3(p3b1.mx`_vertex,
     barrier, mesh);
A2_3 a2d = new A2_3(p3b2.m_vertex,
     barrier, mesh);
a2a.start();a2b.start();a2c.start();a2
     d.start();
barrier.await();
// [E2_1_5^4]
barrier = new CyclicBarrier(4 + 1);
E2_1_5 e2a = new E2_1_5(p3a1.m_vertex,
     barrier, mesh);
E2_1_5 e2b = new E2_1_5(p3a2.m_vertex,
     barrier, mesh);
E2_1_5 e2c = new E2_1_5(p3b1.m_vertex,
     barrier, mesh);
E2_1_5 e2d = new E2_1_5(p3b2.m_vertex,
     barrier, mesh);
e2a.start();e2b.start();e2c.start();e2
     d.start();
barrier.await();
// [A2_2^2]
barrier = new CyclicBarrier(2 + 1);
A2_2 a22a = new A2_2(p2a.m_vertex,
     barrier, mesh);
A2_2 a22b = new A2_2(p2b.m_vertex,
     barrier, mesh);
a22a.start();a22b.start();
barrier.await();
// [E2_2_6^2]
barrier = new CyclicBarrier(2 + 1);
E2_2_6 e26a = new E2_2_6(p2a.m_vertex,
     barrier, mesh);
E2_2_6 e26b = new E2_2_6(p2b.m_vertex,
     barrier, mesh);
e26a.start();e26b.start();
barrier.await();
// [Aroot]
barrier = new CyclicBarrier(1 + 1);
Aroot aroot = new Aroot(p1.m_vertex,
     barrier, mesh);
aroot.start();
barrier.await();
// [Eroot]
barrier = new CyclicBarrier(1 + 1);
Eroot eroot = new Eroot(p1.m_vertex,
     barrier, mesh);
eroot.start();
barrier.await();
printMatrix(eroot.m_vertex, 6,
     mesh.m_dofsy);
// [BS_2_6^2]
barrier = new CyclicBarrier(2 + 1);
BS_2_6 bs1a = new BS_2_6(p2a.m_vertex,
     barrier, mesh);
BS_2_6 bs1b = new BS_2_6(p2b.m_vertex,
     barrier, mesh);
bs1a.start();bs1b.start();
barrier.await();
// [BS_1_5^4]
barrier = new CyclicBarrier(4 + 1);
BS_1_5 bs2a = new
     BS_1_5(p3a1.m_vertex, barrier,
     mesh);
BS_1_5 bs2b = new
     BS_1_5(p3a2.m_vertex, barrier,
     mesh);
```

```
BS_1_5 bs2c = new
     BS_1_5(p3b1.m_vertex, barrier,
     mesh);
BS_1_5 bs2d = new
     BS_1_5(p3b2.m_vertex, barrier,
     mesh);
bs2a.start();bs2b.start();bs2c.start()
     ;bs2d.start();
barrier.await();
//REORDER FOR SECOND SOLVER CALL
double[][] rhs = new double[n * 3 + p
     + 1][];
rhs[1] = bs2a.m_vertex.m_x[1];
rhs[2] = bs2a.m_vertex.m_x[2];
rhs[3] = bs2a.m_vertex.m_x[3];
rhs[4] = bs2a.m_vertex.m_x[4];
rhs[5] = bs2a.m_vertex.m_x[5];
rhs[6] = bs2b.m_vertex.m_x[3];
rhs[7] = bs2b.m_vertex.m_x[4];
rhs[8] = bs2a.m_vertex.m_x[5];
rhs[9] = bs2c.m_vertex.m_x[3];
rhs[10] = bs2c.m_vertex.m_x[4];
rhs[11] = bs2c.m_vertex.m_x[5];
rhs[12] = bs2d.m_vertex.m_x[3];
rhs[13] = bs2d.m_vertex.m_x[4];
rhs[14] = bs2d.m_vertex.m_x[5];
// HERE WE CALL THE ONE DIMENSIONAL
     MULTI-FRONTAL SOLVER
// WITH MULTIPLE RHS FOR ANOTHER
     DIRECTION
// WE UTILIZE SIMILAR TASKS AS FOR THE
     FIRST CALL
// ...

// PRINT OUT THE SOLUTION
for (int i = 1; i <= mesh.m_dofsx; ++
     i) {
 for (int j = 1; j <= mesh.m_dofsy; ++
     j) {
 System.out.printf("%6.2f ",
     rhs[i][j]);
 }
 System.out.println();
}
System.exit(0);
} catch
(InterruptedException | BrokenBarri-
     erException e)
{ e.printStackTrace();}
}
}
```

5. EFFICIENCY TESTS

In this section, we show the scalability of the ADI solver executed on a single Linux cluster node equipped AMD Opteron processor with 6 physical (12 virtual) cores, 2,4 GHz and 16 GB of RAM. By using the ADI algorithm we are able to solve the problem of size 148,996 (single time step) within 5 seconds using 1 core, and 1 seconds using 12 virtual cores. We are able to solve the problem of size 2,365,444 (single time step) within 17 seconds using 1 core, and 2 seconds using 12 virtual cores. The exemplary scalability of the solver is depicted in figures 2 and 3.

The justification for choosing JAVA as a programming language is motivated by its simplicity, elegance, and portability to different computational platforms. It can be also used as a base for developing more optimized implementation in other languages. The JAVA language implementation, as implies from our experiments, is up to one order of magnitude slower than corresponding fortran + MPI implementation.

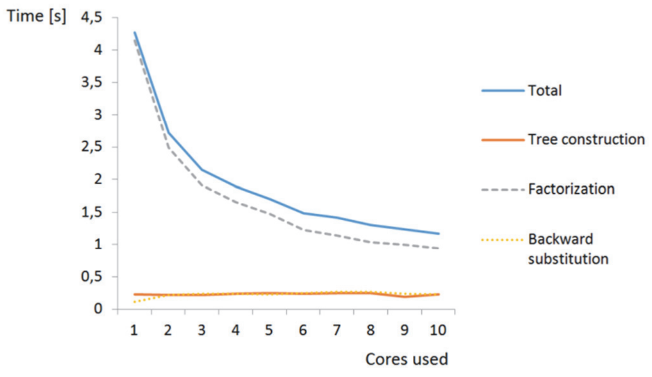

Fig. 2. Scalability of the JAVA code executed on the problem with N=148,996 unknowns.

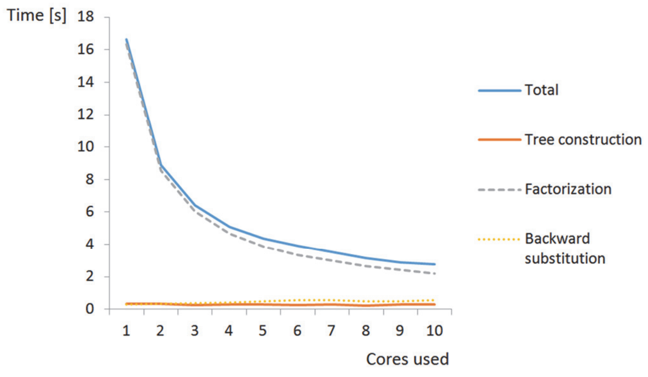

Fig. 3. Scalability of the JAVA code executed on the problem with N=2,365,444 unknowns.

6. ALTERNATING DIRECTION SOLVER FOR TIME DEPENDENT PROBLEMS

In this section, we show how to transform a time-dependent problem into a sequence of isogeometric L2 projection to be used with several ADI solver calls. We start with the definition of an arbitrary second order time dependent problem

$$\frac{\partial u}{\partial t} - L(u) = f(x,t) \text{ in } \Omega \times (0,T) \tag{14}$$

$$u(x,0) = u_0(x) \text{ in } \Omega \tag{15}$$

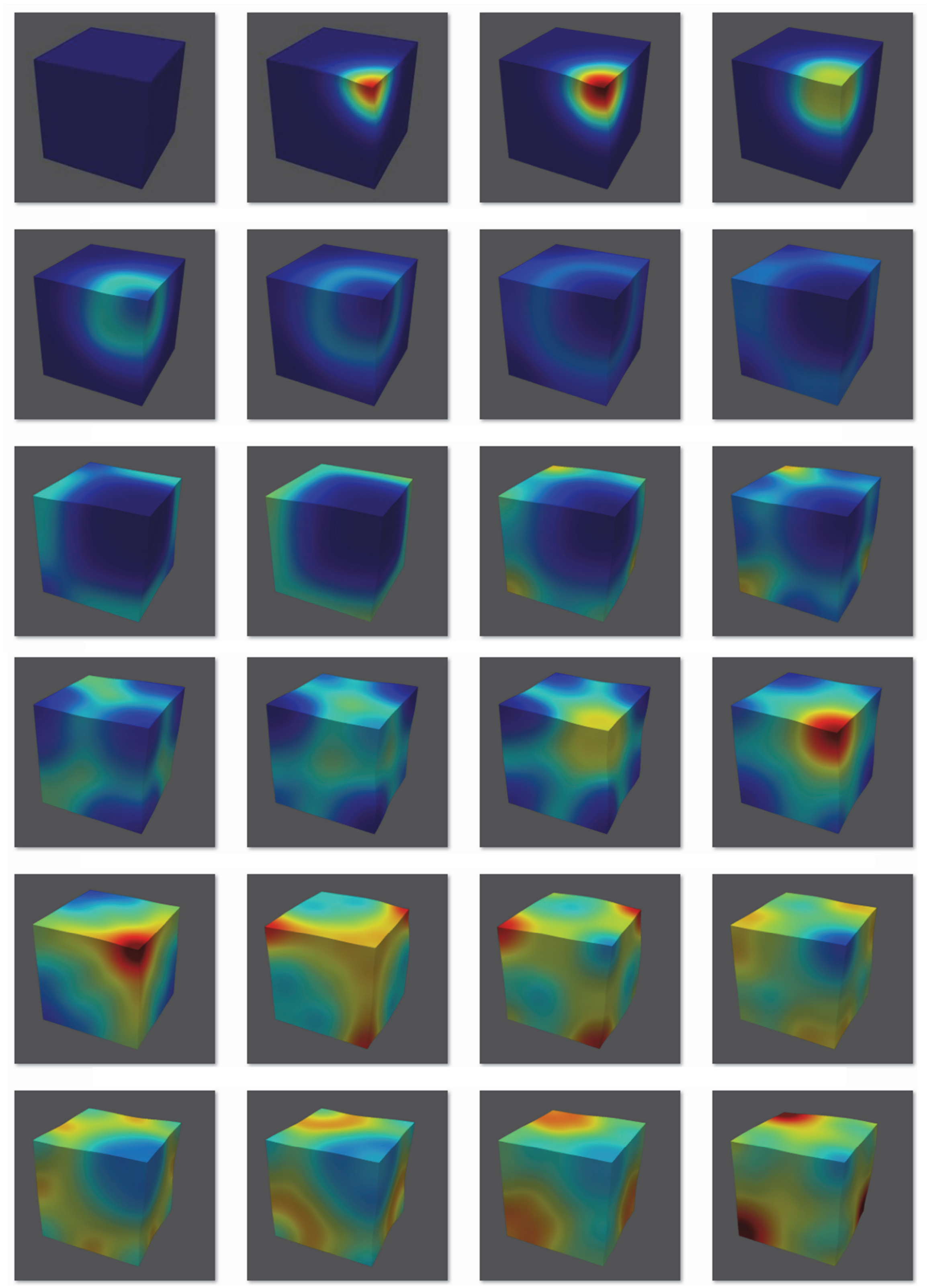

Fig. 4. Numerical simulation of the propagation of elastic wave after hitting the corner of the cube. 18,000 time steps of the Euler method

Next we transform time dependent problem into a weak form. We multiply by test functions $v \in V$, and integrate by parts

$$(v, \frac{\partial u}{\partial t})_\Omega + b(v,u) = (v,f)_\Omega \forall v \in V \tag{16}$$

where

$$b(u,v) = \left(v, L(u)\right)_\Omega \text{ and}$$

$$(f_1, f_2)\Omega = \int_\Omega f_1 f_2 dx \tag{17}$$

Finally we can utilize Forward Euler scheme with respect to time

$$\frac{\partial u}{\partial t} \approx \frac{u_{t+1}-u_t}{\Delta t} \tag{18}$$

$$v, \frac{u_{t+1}-u_t}{\Delta t} + \left(v, L(u_t)\right)_\Omega = (v, f_t)_\Omega \forall v \in V \tag{19}$$

$$(v, u_{t+1})_\Omega = (v, u_t + \Delta t[r_t - L(u_t)])_\Omega \; \forall v \in V \tag{20}$$

which results in a sequence of L2 projections to be solved at every time step.

We make it isogeometric L2 projections by replacing the arbitrary test functions by B-spline basis and using a linear combination of B-spline basis funtions to approximate the solutions in previous and current time steps

$$u_{t+1} = \sum_i B_i^x B_j^y a_{i,j}^{t+1} \; u_t = \sum_i B_i^x B_j^y a_{i,j}^t \; v = B_k^x B_l^y$$

Thus, we have a sequence of isogeometric L2 projections to be solved in linear computational cost, with the right-hand side representing the interaction with the previous time step solution.

7. NUMERICAL RESULTS

We summarize the paper with numerical results concerning the propagation of the elastic waves over the elastic cube after hitting one of its corners, presented in figure 2. The problem has been solver using ADS solver, with three one-dimensional multi-frontal solvers with multiple right-hand sides, implemented and tested as described in this paper.

For more details on the problem formulation (on the definition of the $L(u)$ operator with initial / boundary conditions) we refer to (Łoś et al., 2015). The simulation took 18,000 iterations of the explicit Euler method.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we described a multi-thread parallel open source JAVA implementation of the alternating directions isogeometric L2 projections solver. The solver translates the 3D problem into three one-dimensional solvers with multiple right-hand sides, resulting in a linear computational cost. The solver is utilized for the solution of time-dependent problems, discretized using isogeometric finite element method with B-spline basis functions in spatial domain, and Euler method with respect to time. The one-dimensional multi-frontal solver utilized for the numerical solution of the time-dependent problem has been expressed as set of independent tasks that can be executed in parallel. The resulting tasks have been partitioned into sets of independent tasks that can be executed in parallel.

We also provided a JAVA implementation of the scheduler for the tasks, constructing the elimination tree and performing factorizations and backward substitutions. The JAVA codes are available at `home.agh.edu.pl/paszynsk/CADCAE/ADS`

The JAVA code can be utilized as a reference for designing and building ADI solver in more sophisticated tools, like e.g. NVIDIA CUDA implementation for GPGPU. The future work may include the development of GPGPU accelerators, as described in Kuźnik et al. (2013).

ACKNOWLEDGEMENTS

This work is supported by grant NCN 2015/19/B/ST8/01064.

REFERENCES

Alotaibi, M, Calo, V. M., Efendiev, Y., Galvis, J., Ghommem, M., 2015, Global-Local Nonlinear Model Reduction for Flows in Heterogeneous Porous Media, *Computer Methods in Applied Mechanics and Engineer*ing, 292, 122-137.

Birkhoff, G, Varga, R.S., Young, D., 1962, Alternating direction implicit methods, *Advanced Computing*, 3,189-273.

Collier, N, Pardo, D., Dalcin, L., Paszynski, M., Calo, V. M., 2015, The cost of continuity: a study of the performance of isogeometric finite elements using direct solvers, *Computer Methods in Applied Mechanics and Engineering*, 213-216, 353-361.

Gao, L., Calo, V. M., 2014, Fast Isogeometric Solvers for Explicit Dynamics. *Computer Methods in Applied Mechanics and Engineering*, 1, 19-41.

Gao, L., Calo, V. M., 2015, Preconditioners based on the alternating-direction-implicit algorithm for the 2D steady-state diffusion equation with orthotropic heterogenous coefficients. *Journal of Computational and Applied Mathematics*, 273(1), 274-295.

Douglas, J., Rachford, H., 1956, On the numerical solution of heat conduction problems in two and three space variables. *Transactions of American Mathematical Society* 82, 421-439.

Duff, I. S. Reid, J. K., 1983, The multifrontal solution of indefinite sparse symmetric linear. *ACM Transactions on Mathematical Software* 9(3), 302–325.

Duff, I. S. Reid, J. K., 1984, The multifrontal solution of unsymmetric sets of linear equations. *Journal on Scientific and Statistical Computing* 5, 633–641.

Geng , P., Oden, T. J. van de Geijn, R. A., 2006, A parallel multifrontal algorithm and its implementation. *Computer Methods in Applied Mechanics and Engineering* 149, 289–301.

Kuźnik, K., Paszyński, M., Calo, V. M., 2013, Grammar-Based Multi-Frontal Solver for One Dimensional Isogeometric Analysis with Multiple Right-Hand-Sides. *Procedia Computer Science*, 18, 1574-1583.

Łoś, M., Woźniak, M., Paszyński, M., Dalcin, L., Calo, V. M., 2015, Dynamics with Matrices Possessing Kronecker Product Structure. *Procedia Computer Science*, 51, 286-295.

Paszyński, M., 2016, *Fast Solvers for Mesh Based Computations*, Taylor & Francis, CRC Press., New York.

Wachspress, E.L., Habetler, G., 1960, An alternating-direction-implicit iteration technique. *Journal of Society of Industrial and Applied Mathematics* 8, 403-423.

Saad, Y, 2003, *Iterative methods for sparse linear systems*, SIAM.

OTWARTE OPROGRAMOWANIE ZAWIERAJĄCE IMPLEMENTACJĘ W JĘZYKU JAVA RÓWNOLEGŁEGO SOLWERA WIELOWĄTKOWEGO METODY ZMIENNO-KIERUNKOWYCH IZOGEOMETRYCZNYCH L2 PROJEKCJI W ZASTOSOWANIACH DO SYMULACJI INŻYNIERII MATERIAŁOWEJ

Streszczenie

W artykule tym opisujemy otwarte oprogramowanie zawierające implementację w języku JAVA równoległego solwera wielowątkowej metody izogeometrycznych L2 projekcji. Solwer ten umożliwia przeprowadzanie szybkich symulacji problemów zależnych od czasu. Aby dało się zastosować proponowany solwer, problem niestacjonarny musi zostać zdyskretyzowany za pomocą izogeometrycznej metody elementów skończonych zawierającej funkcje bazowe B-spline w dziedzinie przestrzennej. Problem niestacjonarny rozwiązywany jest za pomocą metody explicite w dziedzinie czasowej. Zastosowanie metody explicite względem czasu redukuje problem obliczeniowy do sekwencji izogeometrycznych L2 projekcji które mogą zostać rozwiązane za pomocą naszego szybkiego solwera. Koszt obliczeniowy szybkiego solwera projekcji izogeometrycznych dla dwu- oraz trójwymiarowych problemów niestacjonarnych jest liniowy O(N) w każdym kroku czasowym. Koszt ten jest niższy niż koszt klasycznego solwera wielo-frontalnego $O(N^{1.5})$ dla problemów 2D oraz $O(N^2)$ dla problemów 3D. Koszt ten jest również niższy niż koszt solwera iteracyjnego O(Nk) gdzie k oznacza liczbę iteracji, która zależy od rodzaju algorytmu solwera iteracyjnego. Solwer nasz zastosowany jest do rozwiązania trójwymiarowego problemu 3D liniowej sprężystości.

Received: October 10, 2016
Received in a revised form: November 9, 2016
Accepted: December 16, 2016