

Scaling Out Alternating Direction Isogeometric L2 Projections Solver

Grzegorz Gurgul (AGH)

Marcin Łoś (AGH)

Danuta Szeliga (AGH)

Maciej Paszyński (AGH)

**14th U.S. National Congress on Computational
Mechanics**

July 17-20, 2017, Montreal, Canada

Agenda

- Background
- Isogeometric L2 projections algorithm
- Algorithm
- Conclusions

- Isogeometric L2 projections algorithm

Proposed by prof. Victor Calo: L. Gao, V.M. Calo, *Fast Isogeometric Solvers for Explicit Dynamics*, **Computer Methods in Applied Mechanics and Engineering** (2014).

- Applications to time-dependent problems

Tumor growth simulations (C++ sequential): M. Łoś, M. Paszyński, A. Kłusek, W. Dzwiniel, Application of fast isogeometric L2 projection solver for tumor simulations, **Computer Methods in Applied Mechanics and Engineering** (2017)

Non-linear flow (Fortran+MPI, parallel): M. Woźniak, M. Łoś, M. Paszyński, L. Dalcin, V. Calo, Parallel fast isogeometric solvers for explicit dynamics, **Computing and Informatics** (2017)

- Improving performance of time-dependent applications of ADS

Step 1: CUDA implementation:

G. Gurgul, M. Paszyński, D. Szeliga, Open source JAVA implementation of the parallel multi-thread alternating direction isogeometric L2 projections solver for material science simulations, **KomPlasTech** (2017)

Step 2: Object-Oriented shared memory implementation:

G. Gurgul, M. Paszyński, D. Szeliga, Open source JAVA implementation of the parallel multi-thread alternating direction isogeometric L2 projections solver for material science simulations, **KomPlasTech** (2017)

Step 3: Cloud implementation:

G. Gurgul, M. Paszyński, D. Szeliga, Scaling Out Alternating Direction Isogeometric L2 Projections Solver, 14th U.S. National Congress on Computational Mechanics

Isogeometric L2 projections

In general: non-stationary problem of the form

$$\partial_t u - \mathcal{L}(u) = f(x, t)$$

with some initial state u_0 and boundary conditions

\mathcal{L} – well-posed linear spatial partial differential operator

Weak form: $(\partial_t u + \mathcal{L}u, v)_{L^2} = (f, v)_{L^2}$

Discretization:

- spatial discretization: isogeometric finite element method

$$(\partial_t u_h + \mathcal{L}u_h, v_h)_{L^2} = (f, v_h)_{L^2}$$

$$u_h = \sum_i \phi_i, v_h \in V_h = \text{span}\{\phi_1, \dots, \phi_n\} \text{ (B-splines)}$$

- time discretization with explicit method e.g. forward Euler scheme

$$\mathcal{M}u_h^{(t+1)} = \mathcal{M}u_h^{(t)} + \Delta t \left(\mathcal{L}u_h^{(t)} + \mathcal{F} \right)$$

$$(u_h^{(t+1)}, v_h)_{L^2} = (u_h^{(t)} - \Delta t * \mathcal{L}u_h^{(t)} + \Delta t * \mathcal{F}, v_h)_{L^2}$$

Isogeometric basis functions:

- 1D B-splines basis $B_1(x), \dots, B_n(x)$
- higher dimensions: tensor product basis

$$B_{i_1 \dots i_d}(x_1, \dots, x_d) \equiv B_{i_1}^{x_1}(x_1) \cdots B_{i_d}^{x_d}(x_d)$$

Gram matrix of B-spline basis on 2D domain $\Omega = \Omega_x \times \Omega_y$:

$$\mathcal{M}_{ijkl} = (B_{ij}, B_{kl})_{L^2} = \int_{\Omega} B_{ij} B_{kl} \, d\Omega$$

Standard multi-frontal solver: $O(N^{1.5})$ in 2D, $O(N^2)$ in 3D

L^2 projections – tensor product basis

Isogeometric basis functions:

- 1D B-splines basis $B_1(x), \dots, B_n(x)$
- higher dimensions: tensor product basis

$$B_{i_1 \dots i_d}(x_1, \dots, x_d) \equiv B_{i_1}^{x_1}(x_1) \cdots B_{i_d}^{x_d}(x_d)$$

Gram matrix of B-spline basis on 2D domain $\Omega = \Omega_x \times \Omega_y$:

$$\mathcal{M}_{ijkl} = (B_{ij}, B_{kl})_{L^2} = \int_{\Omega} B_{ij} B_{kl} \, d\Omega$$

$$= \int_{\Omega} B_i^x(x) B_j^y(y) B_k^x(x) B_l^y(y) \, d\Omega$$

$$= \int_{\Omega} (B_i B_k)(x) (B_j B_l)(y) \, d\Omega$$

$$= \left(\int_{\Omega_x} B_i B_k \, dx \right) \left(\int_{\Omega_y} B_j B_l \, dy \right) \\ = \mathcal{M}_{ik}^x \mathcal{M}_{jl}^y$$

$$\mathcal{M} = \mathcal{M}^x \otimes \mathcal{M}^y \quad (\text{Kronecker product})$$

Alternating Direction Solver – 2D

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & 0 \\ A_{21} & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} y_{11} & y_{21} & \cdots & y_{m1} \\ y_{12} & y_{22} & \cdots & y_{m1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1n} & y_{2n} & \cdots & y_{mn} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{21} & \cdots & b_{m1} \\ b_{12} & b_{22} & \cdots & b_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1n} & b_{2n} & \cdots & b_{mn} \end{bmatrix}$$

$$\begin{bmatrix} B_{11} & B_{12} & \cdots & 0 \\ B_{21} & B_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{mm} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{21} & \cdots & x_{2n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix}$$

Alternating Direction Solver – 2D

Gram matrix of tensor product basis

B-spline basis functions have **local support** (over $p + 1$ elements)

$\mathcal{M}^x, \mathcal{M}^y, \dots$ – banded structure

$$\mathcal{M}_{ij}^x = 0 \iff |i - j| > 2p + 1$$

Exemplary basis functions and matrix for cubics

$$\begin{bmatrix} (B_1, B_1)_{L^2} & (B_1, B_2)_{L^2} & (B_1, B_3)_{L^2} & (B_1, B_4)_{L^2} & 0 & 0 & \dots & 0 \\ (B_2, B_1)_{L^2} & (B_2, B_2)_{L^2} & (B_2, B_3)_{L^2} & (B_2, B_4)_{L^2} & (B_2, B_5)_{L^2} & 0 & \dots & 0 \\ (B_3, B_1)_{L^2} & (B_3, B_2)_{L^2} & (B_3, B_3)_{L^2} & (B_3, B_4)_{L^2} & (B_3, B_5)_{L^2} & (B_3, B_6)_{L^2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & (B_n, B_{n-3})_{L^2} & (B_n, B_{n-2})_{L^2} & (B_n, B_{n-1})_{L^2} & (B_n, B_n)_{L^2} \end{bmatrix}$$

Alternating Direction Solver – 2D

Two steps – solving systems with **A** and **B** in different *directions*

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & 0 \\ A_{21} & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} y_{11} & y_{21} & \cdots & y_{m1} \\ y_{12} & y_{22} & \cdots & y_{m1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1n} & y_{2n} & \cdots & y_{mn} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{21} & \cdots & b_{m1} \\ b_{12} & b_{22} & \cdots & b_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1n} & b_{2n} & \cdots & b_{mn} \end{bmatrix}$$

$$\begin{bmatrix} B_{11} & B_{12} & \cdots & 0 \\ B_{21} & B_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{mm} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{21} & \cdots & x_{2n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix}$$

Two one dimensional problems with multiple RHS:

- $n \times n$ with m right hand sides $\rightarrow O(n * m) = O(N)$
- $m \times m$ with n right hand sides $\rightarrow O(m * n) = O(N)$

Linear computational cost $O(N)$

Hitting elastic material (1/2)

Snapshots from the simulation

Hitting elastic material (2/2)

Snapshots from the simulation

The main goal of our work was to create highly performant solution for an exemplary computational problem in material science simulations which could serve as a template for dealing with similar problems.

The code should:

- serve as a living description of the algorithm
- match the expected performance of the solver
- be easy to adapt to solve a different problem

There are three general design principles which can make the implementation of ADS isogeometric L2 projections solver satisfy those requirements. Namely:

- choose tree as a backing data structure
- reuse the same logic for every direction
- think in terms of objects (productions) - not procedures - follow object-oriented coding style guidelines

A language that can help satisfy those requirements should be:

- object-oriented
- with thread-based concurrency model with little overhead
- popular (and relatively easy to learn)
- with no platform dependencies

Java language is a good fit.

The backing data structure is a tree. Each of its nodes holds coefficients of a simple linear equation being a portion of the original system.

Partition of
the problem
matrix into
sub-matrices

Graph of matrices the solver will operate on

We can identify a set of basic tasks applicable on any vertex. We call them **productions**. They can:

- branch a vertex into two $\{P1, P2\}$ or three $\{P3\}$ child vertices
- initialize a vertex with particular coefficients $\{A1, A, AN\}$
- merge two vertices and eliminate unknowns - $\{A2_3, E1_2_5, A2_2, E2_2_6, Aroot, Eroot\}$
- backward substitute parts of solution - $\{BS_2_6, BS_1_5\}$

To obtain a solution for a 1-dimensional problem with 12 elements it is enough to execute the following productions, set by set, going from left to right, on respective vertices.

We can solve any problem of size $3 * 2^{n-1}$ in one dimension. This is being done by repeating intermediate production sequence $n - 2$ times.

For multidimensional problems we first solve in one direction and then use the results to construct the second tree. Its solution is solution to the master problem. We can do this for any dimension at a cost of having more right hand sides.

This **twofold scalability** greatly broadens the range of problems which can be solved using this technique.

Maximum of $6144 \times 6144 = 37$ million elements in less than 3 minutes on 12-core machine with 12GB of RAM.

JAVA implementation (9/10) Testing

The quality of software can be measured by how easy it is to test it.

In the implementation shown nearly everything can be tested in isolation - productions being the best example.

We can derive all levels of testing:

- system - testing the algorithm against some well known problems
- integration - testing the algorithm against simple problems in single direction using a single thread
- unit - testing simplest tasks - mostly productions

This encourages to use well known and recognized techniques like Test Driven Development to create solutions to other problems.

What is a gain of using production based approach over classical procedural-style design?

- **flexibility** as we are able to solve any problem size in any dimensional space
- **performance** as productions are applied in parallel on every set of vertices
- **testability** as every production is an independent piece of code
- **readability** as production *does one task and does it well*

Thank you!

Our research is funded by Polish National Science Centre
grant no. DEC-2015/19/B/ST8/01064