

# Scaling Out Alternating Direction Isogeometric L2 Projections Solver

**Grzegorz Gurgul (AGH)**

Bartosz Baliś (AGH)

Marcin Łoś (AGH)

Danuta Szeliga (AGH)

Maciej Paszyński (AGH)

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

July 17-20, 2017, Montreal, Canada



- Background
- Isogeometric L2 projections solver - theory
- Isogeometric L2 projections solver - algorithm
- Isogeometric L2 projections solver - implementation
- 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*

*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** (2015)

*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)



- Improving performance of time-dependent applications of ADS

## Step 1: CUDA implementation:

G. Gurgul, M. Paszyński, W. Dzwiniel, Łoś, *GPGPU accelerations of tumor growth simulation using isogeometric L2 - projections solver*, **USACM Conference on Isogeometric Analysis and Meshfree Methods (2016)**

## 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*, **Computer Methods in Material Science (2017)**

## Step 3: Cloud implementation:

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



**In general:** non-stationary problem of the form

$$\partial_t u - \mathcal{L}(u) = f(x, t) \implies \sum_{ij} u_{ij} (B_{ij}, B_{kl})_{L^2} = RHS$$

with some initial state  $u_0$  and boundary conditions

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

Discretization:

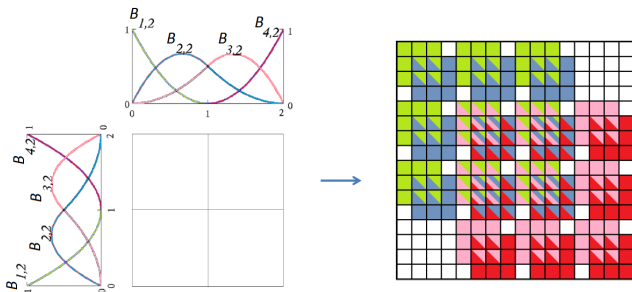
- spatial discretization: isogeometric FEM

Basis functions:  $\phi_1, \dots, \phi_n$  (tensor product B-splines)

- time discretization with explicit method
- implies isogeometric  $L^2$  projections in every time step



# $L^2$ 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  $\Omega = \Omega_x \times \Omega_y$ :

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



## $L^2$ projections – tensor product basis

Isogeometric basis functions:

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

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

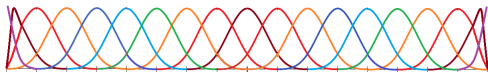
Gram matrix of B-spline basis on 2D domain  $\Omega = \Omega_x \times \Omega_y$ :

$$\begin{aligned}\mathcal{M}_{ijkl} &= (B_{ij}, B_{kl})_{L^2} = \int_{\Omega} B_{ij} B_{kl} \, d\Omega \\ &= \int_{\Omega} B_i^x(x) B_j^y(y) B_k^x(x) B_l^y(y) \, d\Omega \\ &= \int_{\Omega} (B_i B_k)(x) (B_j B_l)(y) \, d\Omega \\ &= \left( \int_{\Omega_x} B_i B_k \, dx \right) \left( \int_{\Omega_y} B_j B_l \, dy \right) \\ &= \mathcal{M}_{ik}^x \mathcal{M}_{jl}^y\end{aligned}$$

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



# 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

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





# Alternating Direction Solver – 2D

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

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

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

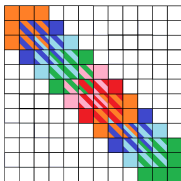
Two one dimensional problems with multiple RHS:

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

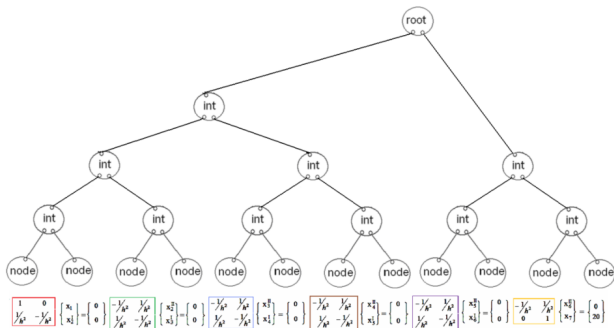
Linear computational cost  $O(N)$  as opposed to  $O(N^{1.5})$  or  $O(N^2)$

# Algorithm - data structure

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



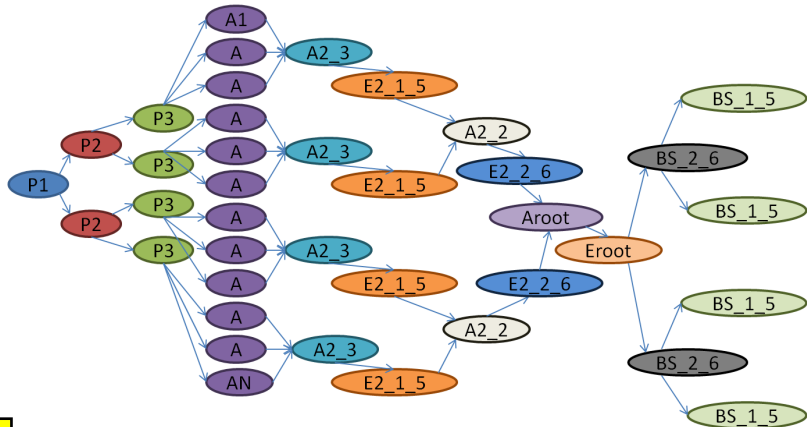
We can identify a set of basic tasks applicable on any vertex. We call them **productions**. They can:

- 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 -  $\{BS\_2\_6, BS\_1\_5\}$



## Algorithm - flow

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.



# Implementation

## CUDA (GPU):

- extremely fast for problems which *fit in on-board memory*
- verbose and fragile
- hardware dependent

## Shared memory (Multicore CPU)

- fast for problems which fit in RAM (37 million elements require 16GB of RAM)
- portable, easy to understand and adapt
- scales up only

## In memory grid (Cloud)

- can solve problems of any size
- slower for relatively small problems
- *scales out*



# Implementation - performance considerations

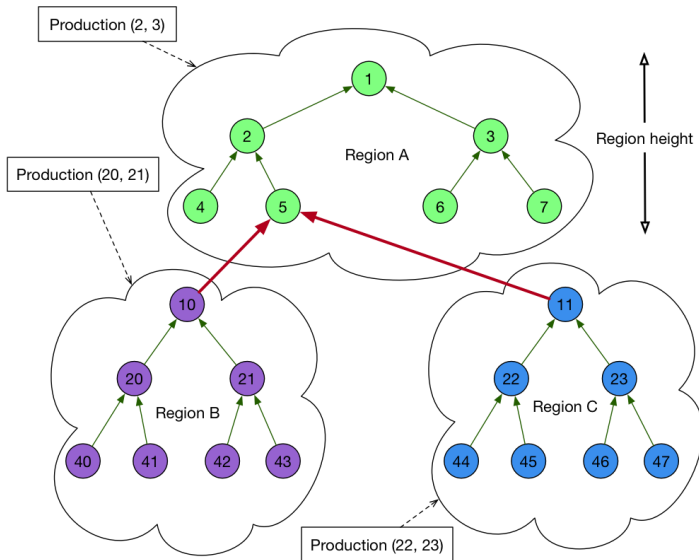
Distributed environment implies *heavy network traffic* and increased memory consumption due to *extensive serialization*. This has to be reduced to the absolute minimum to let solver run bigger problems.

The following measures have been used:

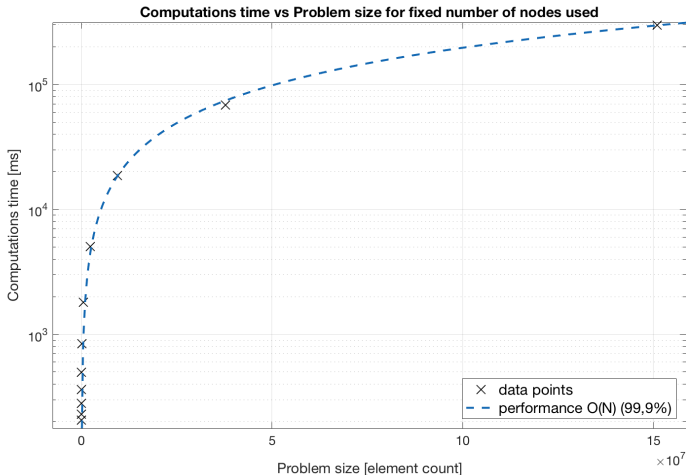
- split tree into set of subtrees of a given height and store them on a single node
- use localized map-reduce to extract columns from the solution rows
- reuse same operation on multiple vertices
- run subsequent operations applied on same vertex in one invocation
- schedule tasks in batches
- use near-caches
- keep solution in deserialized form



# Implementation - partition tree



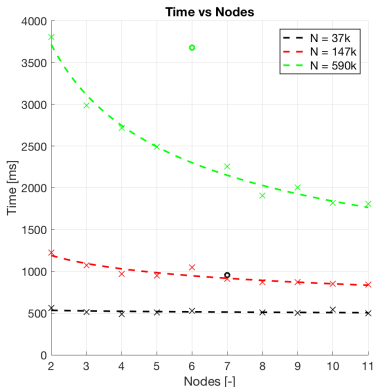
# Implementation - time complexity - $O(N)$



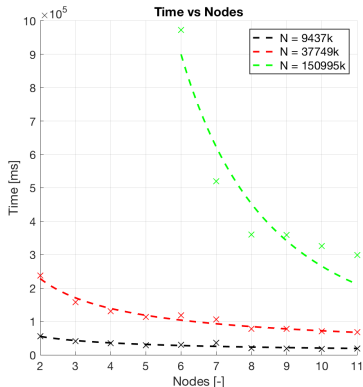
\* Run on a cluster of 11 machines - Westmere (Nehalem-C 2.7GHz) CPU with 4 cores / 8GB RAM each



# Implementation - scaling out



Small problems (2m elem.)



Larger problems (2m to 150m elem.)

\* Each node has Westmere (Nehalem-C 2.7GHz) CPU with 4 cores / 8GB RAM each

- Distributed memory implementation is slower than shared memory one for small problem sizes which fit into physical memory of a single machine
- It can solve any problem by adding additional nodes (4 - 6144, 6 - 12288, 12 - 24576)
- Given there is a shared memory machine with sufficient physical memory, IMDG implementation outperforms it starting from a certain problem size
- This implementation can be made significantly faster by pushing some logic into worker nodes

# Thank you!

Our research is supported from Dean's grant from Faculty of Computer Science, Electronics and Telecommunication, AGH University, Krakow, Poland