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 - theory
- Isogeometric L2 projections algorithm - algorithm
- Isogeometric L2 projections algorithm - implementation
- Conclusions

Background

- 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. Dzwinel, 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)

Background

- 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

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

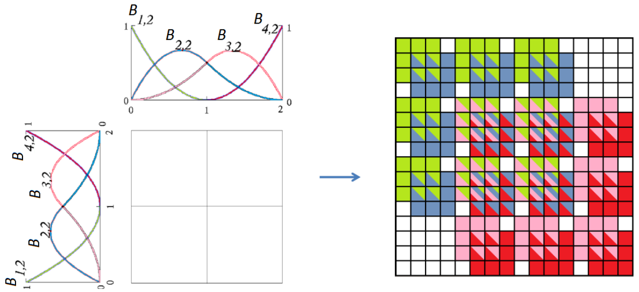
$u_h = \sum_i \phi_i,\ v_h \in V_h = span\{\phi_1, \ldots, \phi_n\}$ (B-splines)

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

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

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

L^2 projections – tensor product basis



Isogeometric basis functions:

- 1D B-splines basis $B_1(x), \ldots, B_n(x)$
- higher dimensions: tensor product basis
  $B_{i_1 \cdots i_d}(x_1, \ldots, x_d) \equiv B^{x_1}_{i_1}(x_1) \cdots B^{x_d}_{i_d}(x_d)$

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

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

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

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

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

$$= \int_\Omega B^x_i(x) B^y_j(y) B^x_k(x) B^y_l(y) \, \mathrm{d}\Omega$$

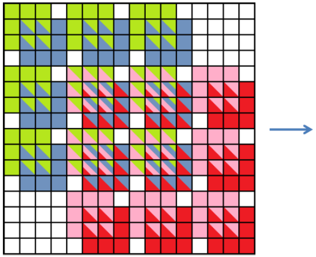
$$= \int_\Omega (B_i B_k)(x) \, (B_j B_l)(y) \, \mathrm{d}\Omega$$

$$= \left(\int_{\Omega_x} B_i B_k \, \mathrm{d}x \right) \left(\int_{\Omega_y} B_j B_l \, \mathrm{d}y \right)$$

$$= \mathcal{M}^x_{ik} \mathcal{M}^y_{jl}$$

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

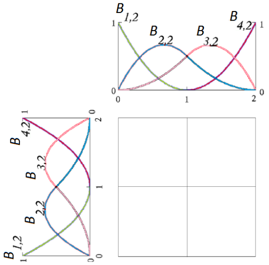
Alternating Direction Solver – 2D



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

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

Alternating Direction Solver – 2D



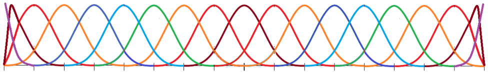
$$(u_h^{(t+1)}, v_h)_{L^2} = (\underbrace{u_h^{(t)} - \Delta t * \mathcal{L}u_h^{(t)} + \Delta t * \mathcal{F}}_{\boldsymbol{F}}, v_h)_{L^2}$$

$$b_{ij} = \int_\Omega B_{ij}\ \boldsymbol{F}\ \mathrm{d}\Omega = \int_\Omega B_{i,2}\ B_{j,2}\ \boldsymbol{F}\ \mathrm{d}\Omega$$

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

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

Gram matrix of tensor product basis



B-spline basis functions have **local support** (over $p + 1$ elements)
$\mathcal{M}^x$, $\mathcal{M}^y$, $\ldots$ – banded structure
$\mathcal{M}^x_{ij} = 0 \iff |i - j| > 2p + 1$
Exemplary basis functions and matrix for cubics

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

Alternating Direction Solver – 2D

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

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

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

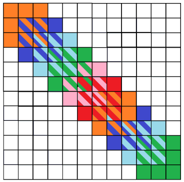
Two one dimensional problems with multiple RHS:

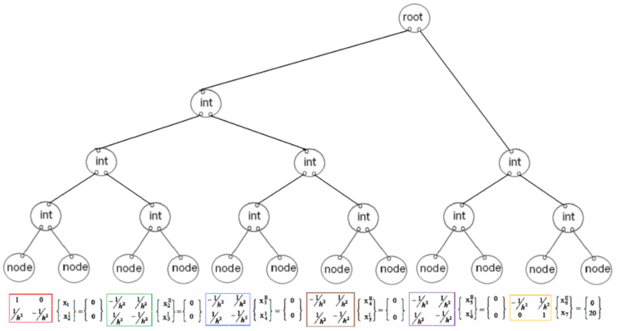
- $n \times n$ with m right hand sides $\rightarrow O(n * m) = O(N)$
- $m \times m$ with n right hand sides $\rightarrow O(m * n) = O(N)$

Linear computational cost $O(N)$

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

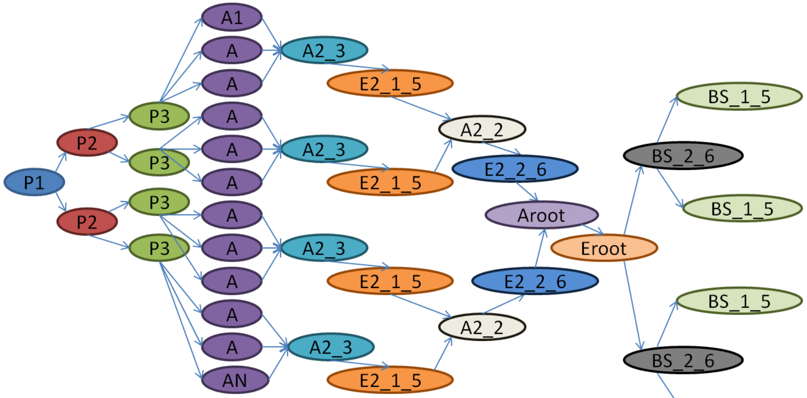
Algorithm - set of operations

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.



Algorithm - applications

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. This is also relevant to time-dependent problems.

Implementation

CUDA (GPU):

- extremely fast but only for problems which fit in integrated memory
- low-level and fragile
- not portable

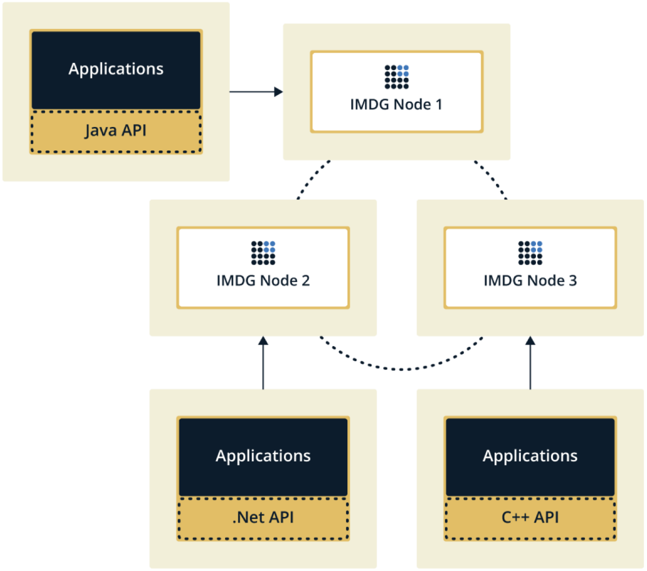
Shared memory (multicore CPU)

- fast but only for problems which fit in RAM ($6144x6144 = 37$ million elements requires 12GB of RAM)
- portable, easy to understand and adapt
- scales up only

In memory grid (cloud)

- slower than both GPU and shared memory version but can solve much larger problems which do not fit into a physical memory of single machine
- scales both up and out

Implementation - IMDG



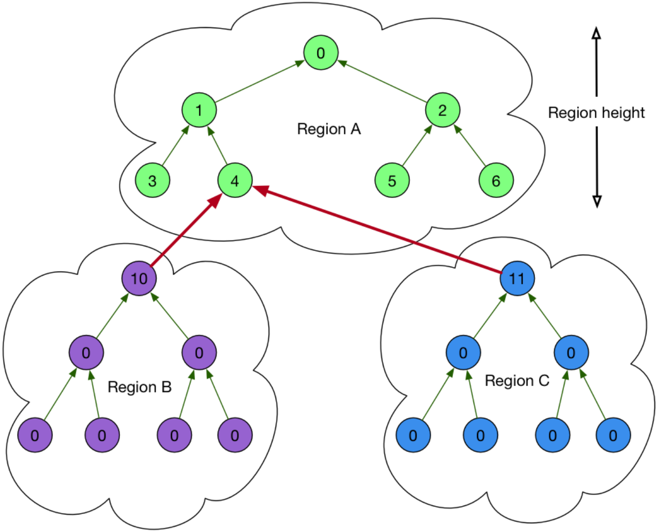
Implementation - performance considerations

Distributed environment implies heavy network traffic. This has to be reduced to the absolute minimum to let solver run have acceptable execution times.

The following measures have been used:

- split tree into subtrees of a given height and store them on a single node (region-height)
- reuse same operation on multiple vertices (max-batch-size)
- run subsequent operations applied on same vertex in one invocation
- schedule tasks in batches (max-job-count)

Implementation - partition tree

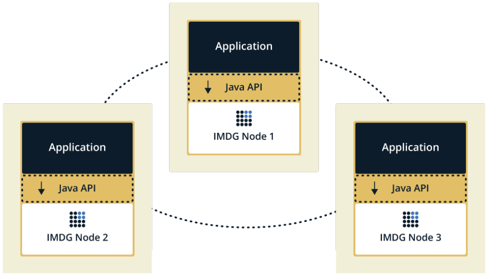


Implementation - performance results

TBD, probably 2-3 slides (show that region partitioning is really very important and so on)

Conclusions

- Distributed memory implementation is much slower than shared memory one for relatively small problem sizes which fit into physical memory of a single machine
- Given there is a shared memory machine with sufficient physical memory, IMDG implementation is expected to outperform it starting from a certain problem
- This implementation can be made significantly faster by pushing some logic into worker nodes



Thank you!

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