Fast and smooth simulation of time dependent problems

Maciej Paszyński

Department of Computer Science
AGH University of Science and Technology, Kraków, Poland
home.agh.edu.pl/paszynsk

Outline

- Projection with isogeometric finite element method
- Generalization to time-dependent problems
- Explicit dynamics
- Example 1: Heat transfer
- Example 2: Non-linear flow in heterogenous media
- Example 3: Tumor growth
- Parallel shared-memory scalability
- Implicit dynamics
- Example 4: Linear elasticity
- Example 5: Pollution problem
- Conclusions

Software

Program Title: IGA-ADS (Isogeometric Analysis Alternating Directions Solver)
Code: `git clone https://github.com/marcinlos/iga-ads`
Licensing provisions: MIT license (MIT)
Programming language: C++
Nature of problem: Solving non-stationary problems in 1D, 2D and 3D
Solution method: Alternating direction solver with isogeometric finite element method

If you use this software in your work, please cite

Marcin Łoś, Maciej Woźniak, Maciej Paszyński, Andrew Lenharth, Keshav Pingali *IGA-ADS : Isogeometric Analysis FEM using ADS solver*, **Computer & Physics Communications** 217 (2017) 99-116 (available on researchgate.org)

Software

Program Title: IGA-ADI-SM (Isogeometric Alternating Directions Implicit Shared Memory Solver)
Code `https://github.com/kboom/iga-adi-sm`
Programming language: JAVA
Nature of problem: Solving 2D projections and non-stationary problems
Solution method: Alternating direction solver with isogeometric finite element method

Details described in

Grzegorz Gurgul, Maciej Paszyński, *Object-oriented implementation of the Alternating Directions Implicit Solver for Isogeometric Analysis*, submitted to **Advances in Engineering Software** (2018)

Recursive definition of 1D B-splines

J.A. Cottrel, T.J.R. Hughes, Y. Bazilevs, *Isogeometric Analysis. Toward Integration of CAD and FEA*, Wiley, (2009).

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \le \xi < \xi_{i+1}, \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)$$

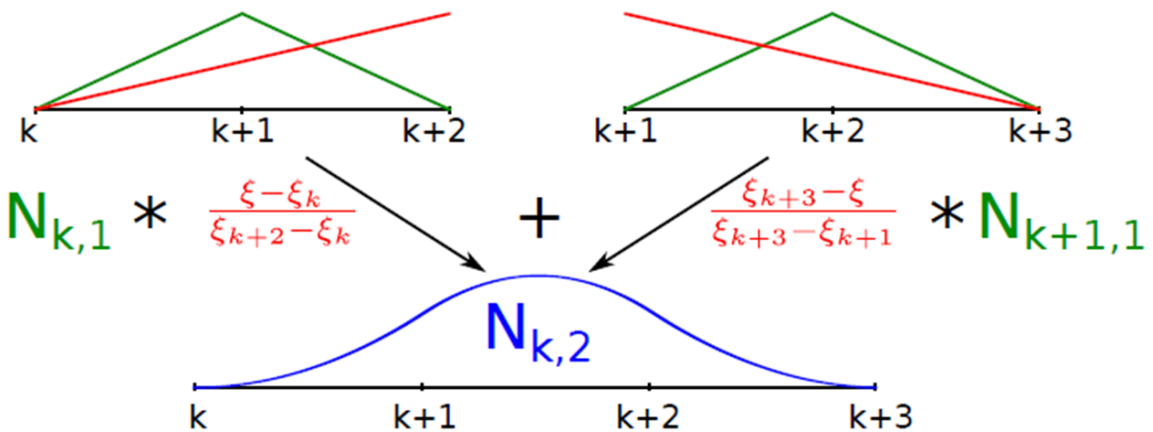


Figure: Recursive formulae for B-spline basis functions and its explanation

Representation of B-splines by knot vectors

J.A. Cottrel, T.J.R. Hughes, Y. Bazilevs, *Isogeometric Analysis. Toward Integration of CAD and FEA*, Wiley, (2009).

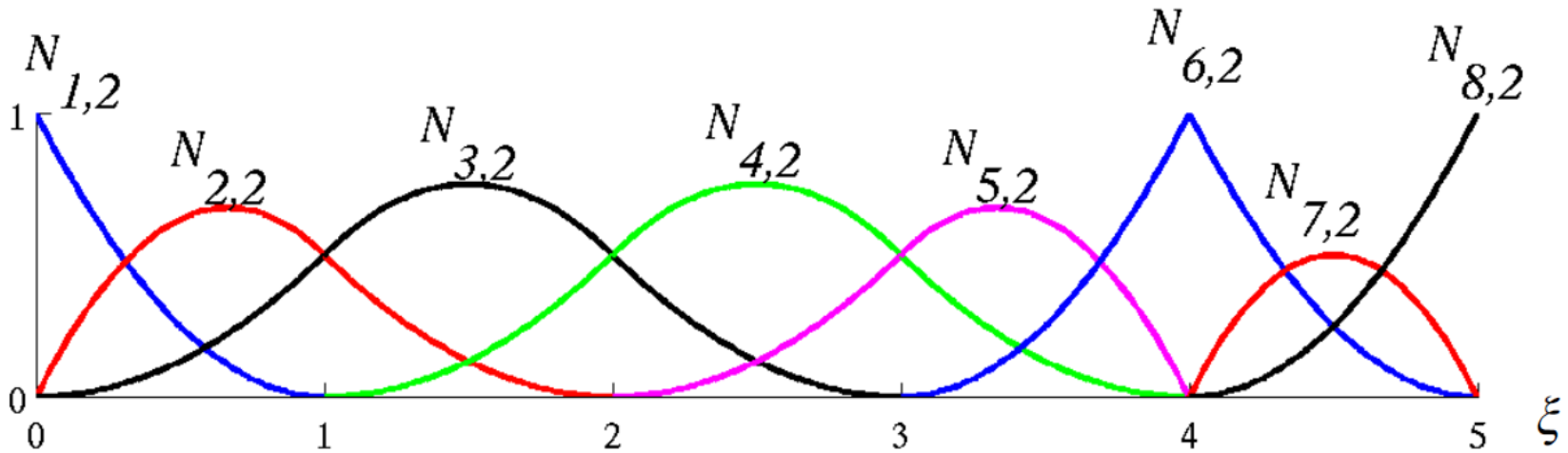


Figure: B-spline basis functions represented by knot vector $\{0,0,0,1,2,3,4,4,5,5,5\}$

Tensor product definition of 2D B-spline basis functions

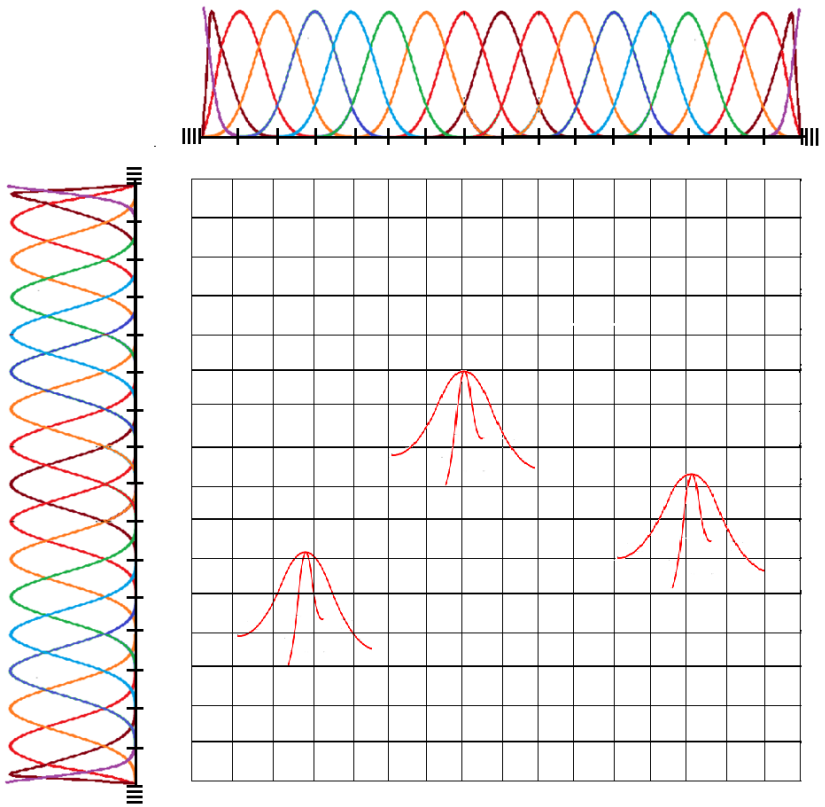


Figure: Tensor products B-splines basis funcitions

- 1D B-splines basis $B_1^x(x), \ldots, B_{N_x}^x(x)$, $B_1^y(y), \ldots, B_{N_y}^y(y)$,
- 2D B-splines basis $B_{i,j}(x, y) = B_i^x(x) * B_j^y(y)$

Projection with isogeometric finite element method

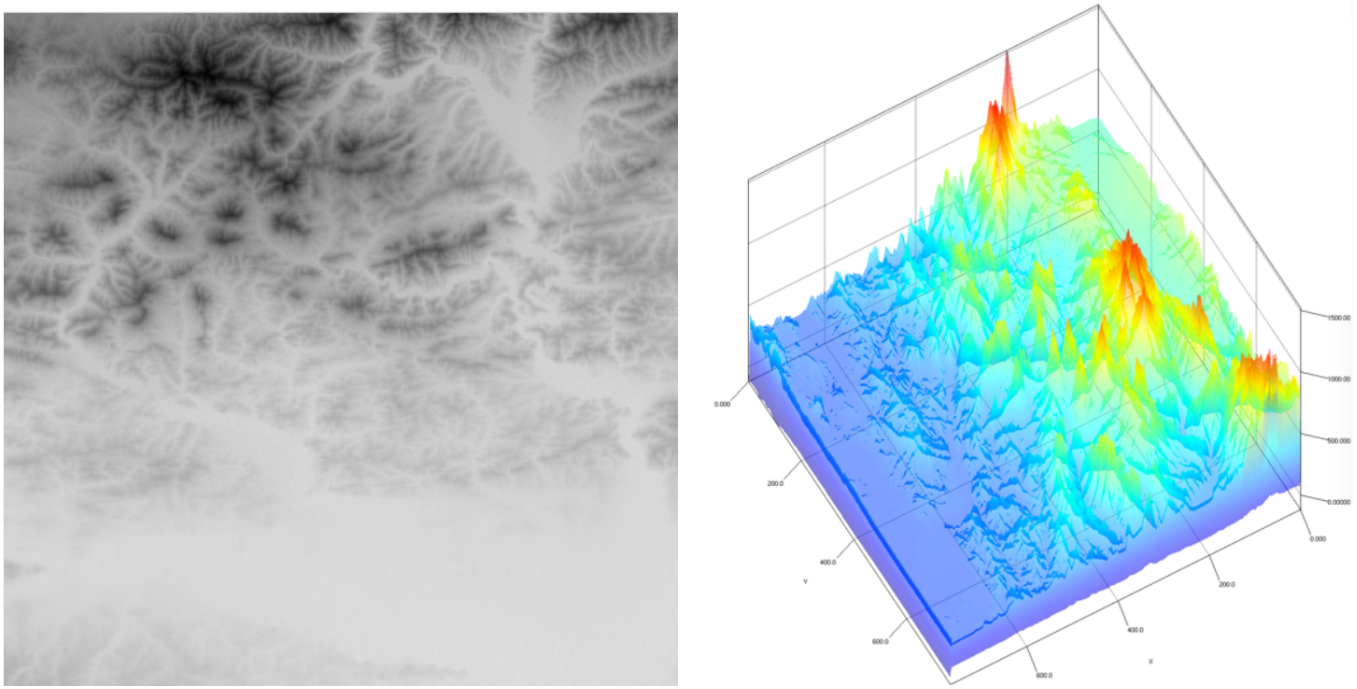


Figure: 2D terrain bitmap and its continuous B-spline approximation

- We want to approximate a BITMAP(x,y) with a linear combination of B-splines
  $u(x, y) \approx BITMAP(x, y)$
  where $u(x, y) = \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y)$
- How to construct a system of linear equations to get the coefficients $u_{i,j}$?

Projection with isogeometric finite element method

- We choose several "test functions" v, and use them to average the BITMAP at test functions supports
  $\int u(x,y)v(x,y)dx = \int BITMAP(x,y)v(x,y)dx$
- Each selection of $v = B_i^x(x)B_j^y(y)$ leads to one equation
  $\int u(x,y)B_i^x(x) * B_j^y(y)dx = \int BITMAP(x,y)B_i^x(x)B_j^y(y)dx$

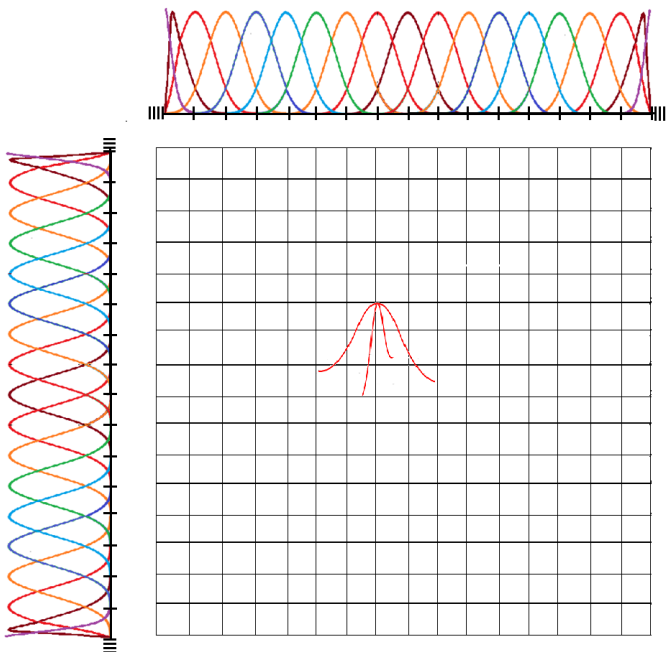


Figure: One exemplary selection of $v = B_i^x(x)B_j^y(y)$

Projection with isogeometric finite element method

We end up with several equations
(one equation per one testing B-spline in 2D)

$$\int u(x,y) B_1^x(x) B_1^y(y) dx = \int BITMAP(x,y) B_1^x(x) B_1^y(y) dx$$
$$\int u(x,y) B_1^x(x) B_2^y(y) dx = \int BITMAP(x,y) B_1^x(x) B_2^y(y) dx$$
$$\vdots$$
$$\int u(x,y) B_k^x(x) B_l^y(y) dx = \int BITMAP(x,y) B_k^x(x) B_l^y(y) dx$$
$$\vdots$$
$$\int u(x,y) B_{N_x}^x(x) B_{N_y-1}^y(y) dx = \int BITMAP(x,y) B_{N_x}^x(x) B_{N_y-1}^y(y) dx$$
$$\int u(x,y) B_{N_x}^x(x) B_{N_y}^y(y) dx = \int BITMAP(x,y) B_{N_x}^x(x) B_{N_y}^y(y) dx$$

Projection with isogeometric finite element method

We approximate $u(x,y) \approx \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y)$

$$\int u(x,y) B_1^x(x) B_1^y(y) dx = \int BITMAP(x,y) B_1^x(x) B_1^y(y) dx$$
$$\int u(x,y) B_1^x(x) B_2^y(y) dx = \int BITMAP(x,y) B_1^x(x) B_2^y(y) dx$$
$$\vdots$$
$$\int u(x,y) B_k^x(x) B_l^y(y) dx = \int BITMAP(x,y) B_k^x(x) B_l^y(y) dx$$
$$\vdots$$
$$\int u(x,y) B_{N_x}^x(x) B_{N_y-1}^y(y) dx = \int BITMAP(x,y) B_{N_x}^x(x) B_{N_y-1}^y(y) dx$$
$$\int u(x,y) B_{N_x}^x(x) B_{N_y}^y(y) dx = \int BITMAP(x,y) B_{N_x}^x(x) B_{N_y}^y(y) dx$$

Projection with isogeometric finite element method

We approximate $u(x,y) \approx \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y)$ to get

$$\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_1^x(x) * B_1^y(y) dx = \int BITMAP(x,y) B_1^x(x) * B_1^y(y) dx$$

$$\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_1^x(x) * B_2^y(y) dx = \int BITMAP(x,y) B_1^x(x) * B_2^y(y) dx$$

$$\vdots$$

$$\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_k^x(x) * B_l^y(y) dx = \int BITMAP(x,y) B_k^x(x) * B_l^y(y) dx$$

$$\vdots$$

$$\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_{N_x}^x(x) * B_{N_y-1}^y(y) dx = \int BITMAP(x,y) B_{N_x}^x(x) * B_{N_y-1}^y(y) dx$$

$$\int \sum_{i,j} u_{i,j} B_i^x(x) B_j^y(y) B_{N_x}^x(x) * B_{N_y}^y(y) dx = \int BITMAP(x,y) B_{N_x}^x(x) * B_{N_y}^y(y) dx$$

Projection with isogeometric finite element method

We take the sum out

$$\sum_{i,j} u_{i,j} \int B_i^x(x) B_j^y(y) B_k^x(x) * B_l^y(y) dx = \int BITMAP(x,y) B_k^x(x) * B_l^y(y) dx$$

and we end up with a system of linear equations

$$\begin{bmatrix} \int B^x_{1,p} B^y_{1,p} B^x_{1,p} B^y_{1,p} & \int B^x_{1,p} B^y_{1,p} B^x_{2,p} B^y_{1,p} & \cdots & \int B^x_{1,p} B^y_{1,p} B^x_{N_x,p} B^y_{N_y,p} \\ \int B^x_{2,p} B^y_{1,p} B^x_{1,p} B^y_{1,p} & \int B^x_{2,p} B^y_{1,p} B^x_{2,p} B^y_{1,p} & \cdots & \int B^x_{2,p} B^y_{1,p} B^x_{N_x,p} B^y_{N_y,p} \\ \vdots & \vdots & \vdots & \vdots \\ \int B^x_{N_x,p} B^y_{N_y,p} B^x_{1,p} B^y_{1,p} & \int B^x_{N_x,p} B^y_{N_y,p} B^x_{2,p} B^y_{1,p} & \cdots & \int B^x_{N_x,p} B^y_{N_y,p} B^x_{N_x,p} B^y_{N_y,p} \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ \vdots \\ u_{N_x,N_y} \end{bmatrix}$$

$$= \begin{bmatrix} \int BITMAP(x,y) B^x_1(x) * B^y_1(y) dx \\ \int BITMAP(x,y) B^x_1(x) * B^y_2(y) dx \\ \vdots \\ \int BITMAP(x,y) B^x_{N_x}(x) * B^y_{N_y}(y) dx \end{bmatrix}$$

Projection with isogeometric finite element method

$$\begin{bmatrix} \int B^x_{1,p} B^y_{1,p} B^x_{1,p} B^y_{1,p} & \int B^x_{1,p} B^y_{1,p} B^x_{2,p} B^y_{1,p} & \cdots & \int B^x_{1,p} B^y_{1,p} B^x_{N_x,p} B^y_{N_y,p} \\ \int B^x_{2,p} B^y_{1,p} B^x_{1,p} B^y_{1,p} & \int B^x_{2,p} B^y_{1,p} B^x_{2,p} B^y_{1,p} & \cdots & \int B^x_{2,p} B^y_{1,p} B^x_{N_x,p} B^y_{N_y,p} \\ \vdots & \vdots & \vdots & \vdots \\ \int B^x_{N_x,p} B^y_{N_y,p} B^x_{1,p} B^y_{1,p} & \int B^x_{N_x,p} B^y_{N_y,p} B^x_{2,p} B^y_{1,p} & \cdots & \int B^x_{N_x,p} B^y_{N_y,p} B^x_{N_x,p} B^y_{N_y,p} \end{bmatrix}$$

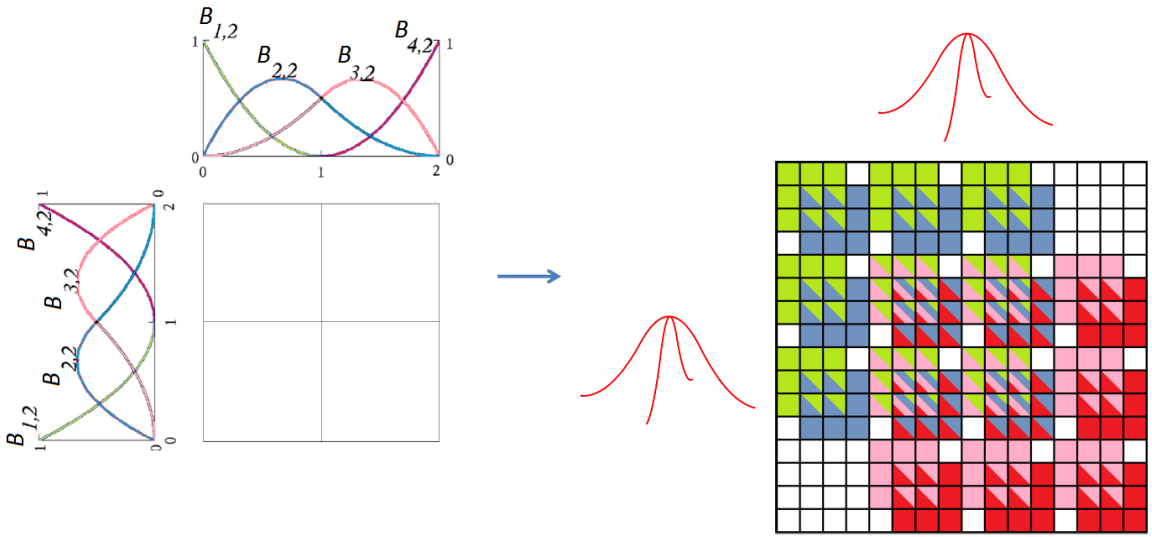
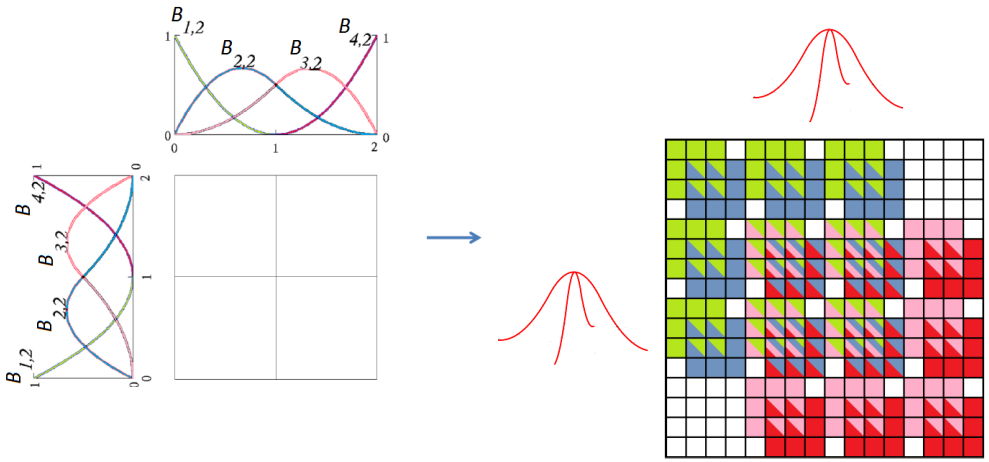


Figure: The resulting system of linear equations can be solved with $\mathcal{O}(N^{1.5})$ time in 2D, or $\mathcal{O}(N^2)$ time in 3D

Fast isogeometric L2 projections

Longfei Gao, *Kronecker Products on Preconditioning*, PhD. Thesis, KAUST (supervised by Prof. Victor Calo), 2013.



Projection problem matrix on 2D domain $\Omega = \Omega_x \times \Omega_y$:

$$\mathcal{M}_{ijkl} = \int_\Omega B_{ij} B_{kl} \, \mathrm{d}\Omega = \int_\Omega B_i^x(x) B_j^y(y) B_k^x(x) B_l^y(y) \, \mathrm{d}\Omega =$$

$$\int_\Omega (B_i B_k)(x) \, (B_j B_l)(y) \, \mathrm{d}\Omega = \int_{\Omega_x} B_i B_k \, \mathrm{d}x \int_{\Omega_y} B_j B_l \, \mathrm{d}y = \mathcal{M}_{ik}^x \mathcal{M}_{jl}^y$$

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

Fast isogeometric L2 projections

Longfei Gao, *Kronecker Products on Preconditioning*, PhD. Thesis, KAUST (supervised by Prof. Victor Calo), 2013.

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

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

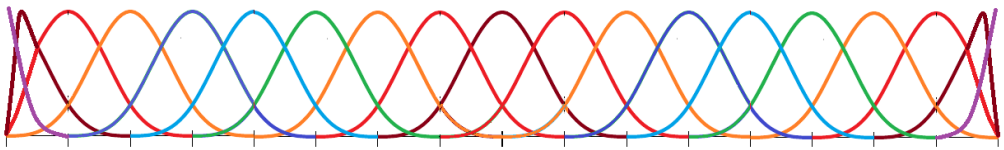
$$\begin{bmatrix} B_{11} & B_{12} & \cdots & 0 \\ B_{21} & B_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{mm} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{21} & \cdots & x_{2n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix}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

Two 1D problems with multiple RHS, linear cost $O(N)$

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

Fast isogeometric L2 projections

Longfei Gao, *Kronecker Products on Preconditioning*, PhD. Thesis, KAUST (supervised by Prof. Victor Calo), 2013.

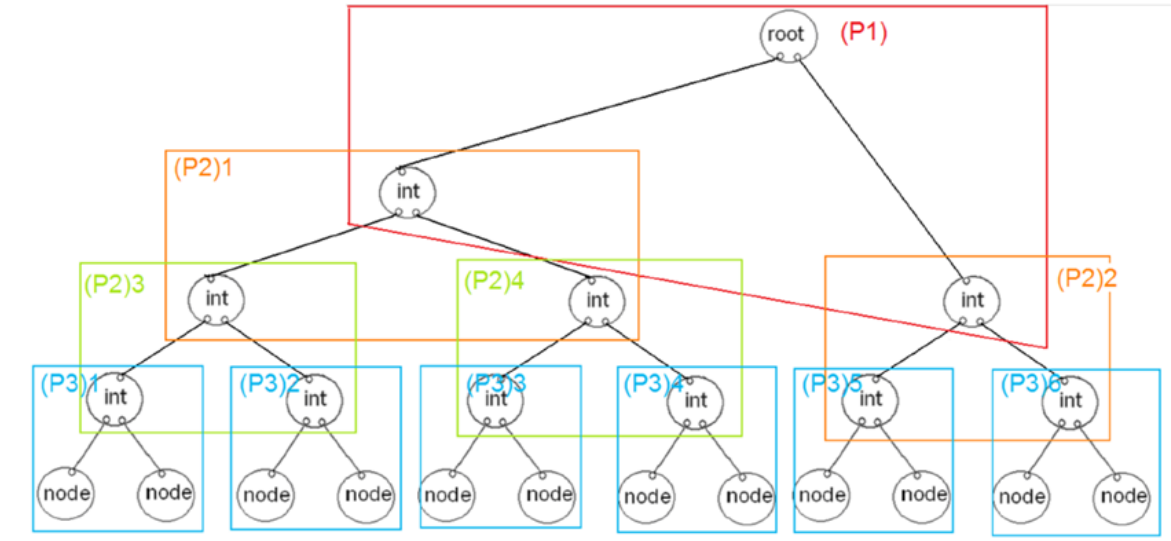


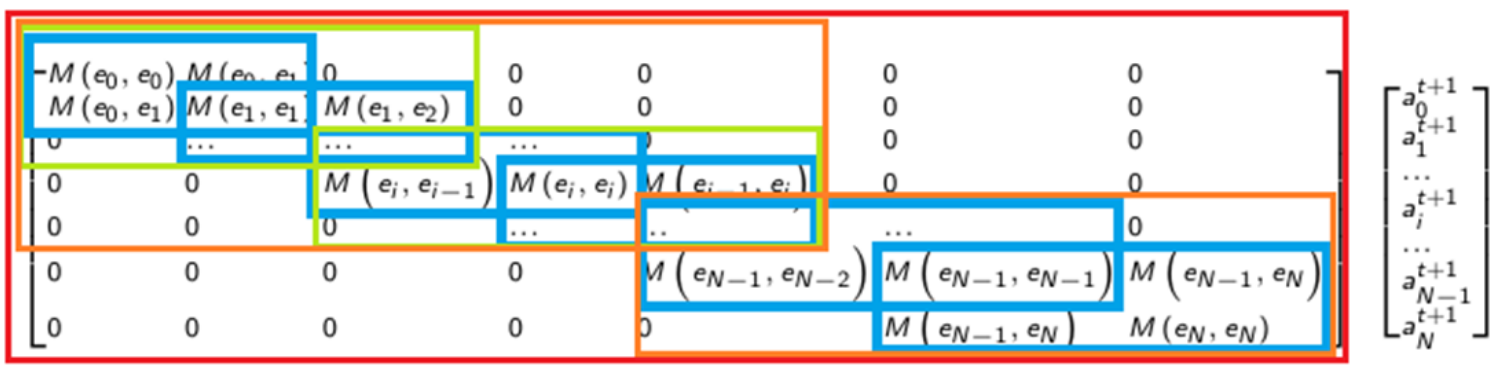
B-spline basis functions have **local support** (over $p+1$ elements)
$\mathcal{M}^x$, $\mathcal{M}^y$, ... – banded structure
$\mathcal{M}^x_{ij} = 0 \iff |i-j| > p$
Exemplary basis functions and matrix for cubics

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

Multi-diagonal matrix can be factorized in linear $\mathcal{O}(N)$ cost.

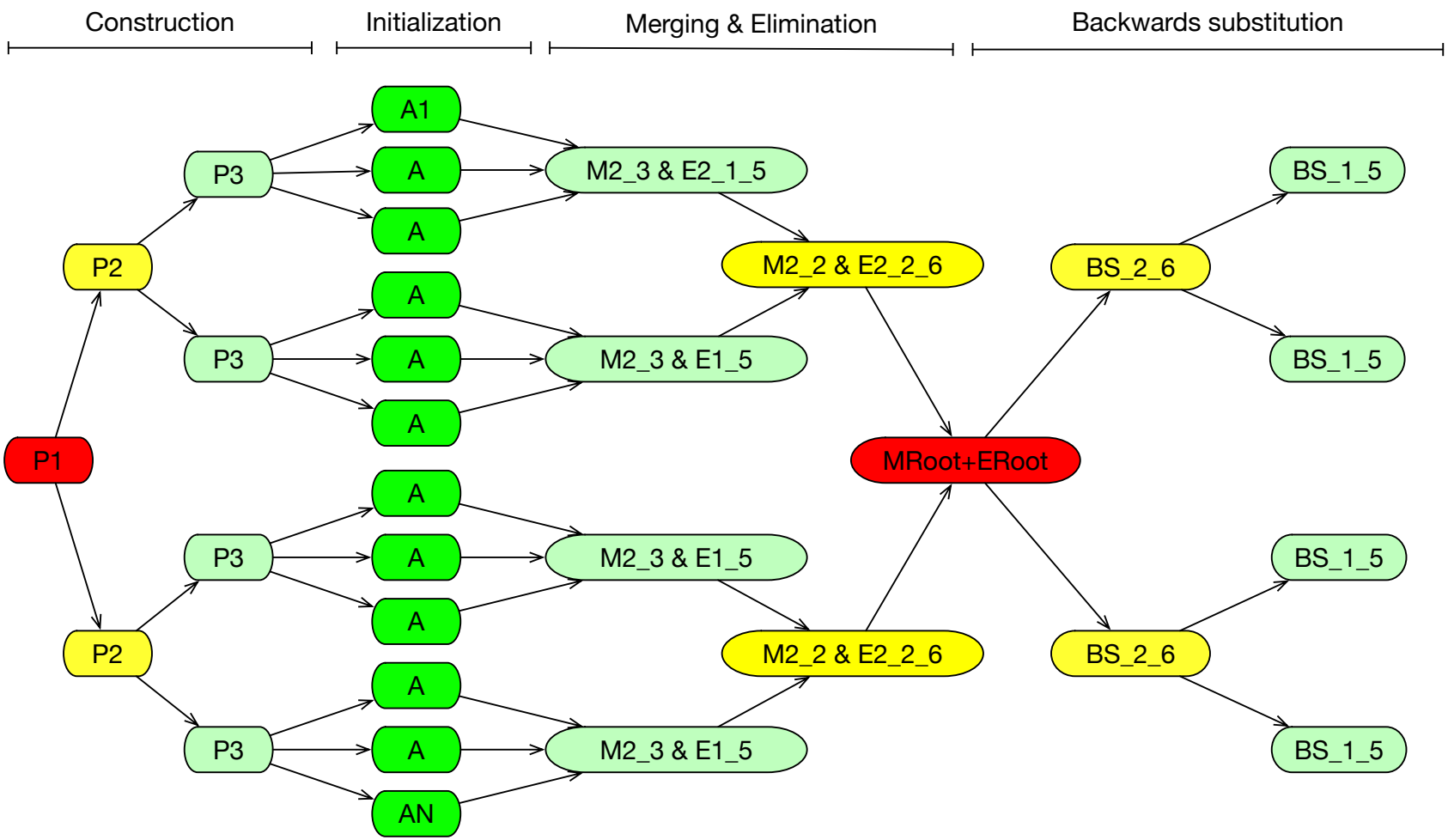
Parallel shared-memory task-based implementation





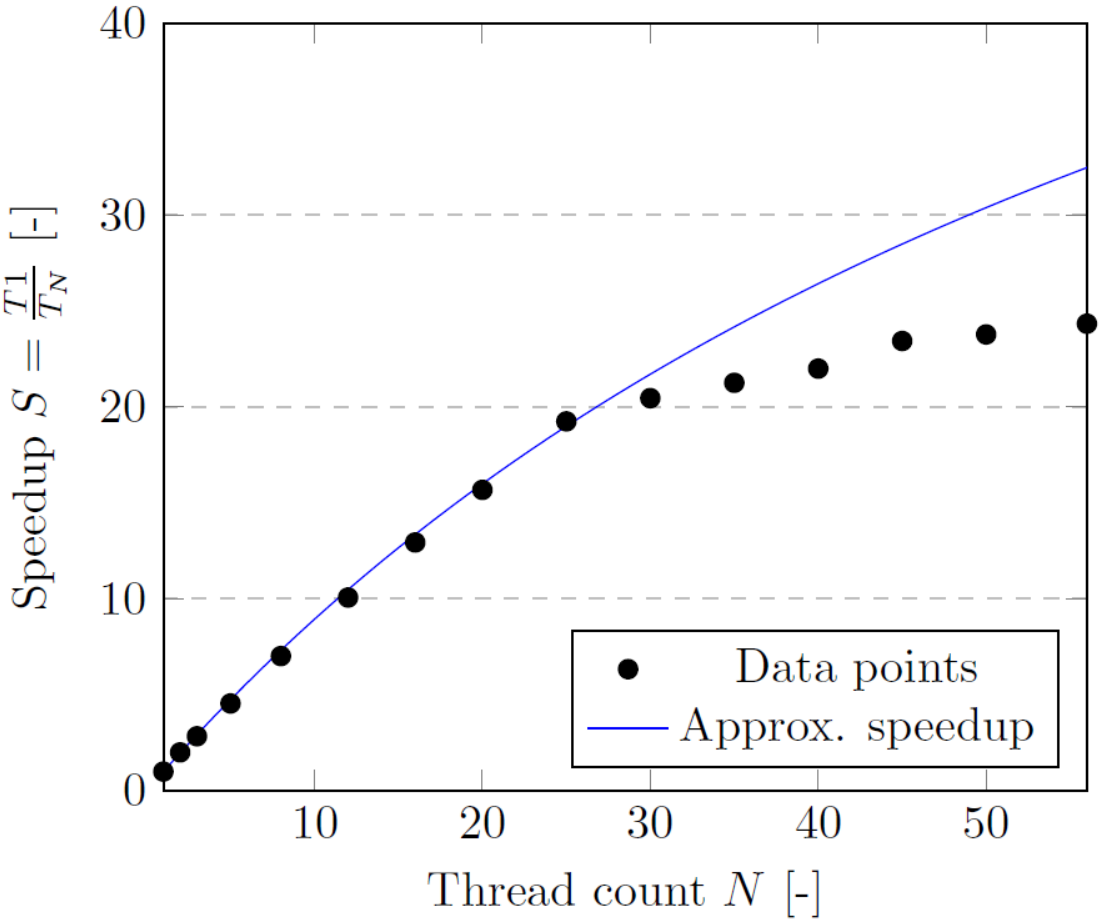
M. Paszyński, Fast solvers for mesh-based computations, Taylor & Francis, CRC Press (2016)

Parallel shared-memory task-based implementation



M. Paszyński, Fast solvers for mesh-based computations, Taylor & Francis, CRC Press (2016)

Parallel shared-memory task-based implementation



Node with two Intel Xeon E5-2680 CPUs with 56 threads in total (two processors with 14 physical cores and 28 threads each) with 64 gigabytes of RAM.

Parallel shared-memory task-based implementation

For $16,000,000$ dofs which is 2^{12} tasks along axis
$(1 - P)$=percentage of the algorithm which does not benefit from parallel speedup
P = percentage of the algorithm which benefits from speedup
$\frac{P}{n}$ = how much does it benefit using n threads
$S(n)$ = measured speedup when using n threads

$$S(n) = \frac{1}{(1 - P) + \frac{P}{n}}$$

We substitute $n = 20$ threads and $S(20) = 16$ into the Amdahl's law

$$P = \frac{n * (S(n) - 1)}{(n - 1) * S(n)} = \frac{20 * (S(20) - 1)}{(20 - 1) * S(20)} = \frac{75}{76} > 98\%$$

The percentage of the algorithm that benefits from parallelization is 98%.

$$\lim_{n \to \infty} \frac{1}{(1 - \frac{75}{76}) + \frac{\frac{75}{76}}{n}} = 76$$

The speedup limit is equal to 76.

Generalization to time-dependent problems

Maciej Woźniak, Marcin Łoś, Maciej Paszyński, Lisandro Dalcin, Victor Manuel Calo, Parallel fast isogeometric solvers for explicit dynamics, **Computing and Informatics**, 36(2) (2017) 423–448. (Fortran code)

Marcin Łoś, Maciej Woźniak, Maciej Paszyński, Andrew Lenharth, Keshav Pingali, *IGA-ADS : Isogeometric Analysis FEM using ADS solver*, **Computer & Physics Communications** 217 (2017) 99-116 (C++ code)

In general: time dependent problem of the form

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

$u(x,t)$ - the unknown field
(its meaning depends on the simulated physical phenomena)

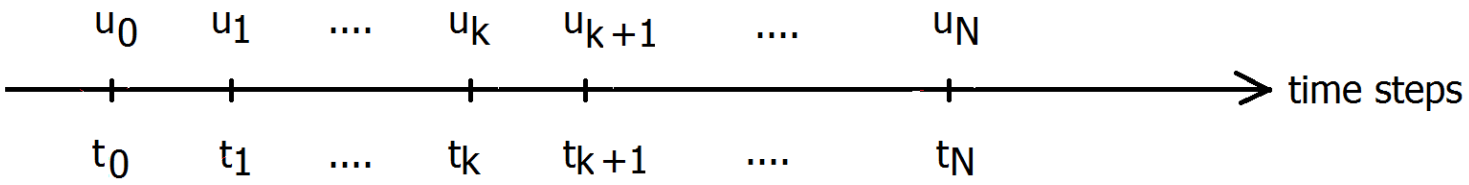
$\frac{\partial u(x,t)}{\partial t}$ - changes of the phenomena in time

$\mathcal{L}$ - the physics (well-posed linear spatial PDE)

$f(x,t)$ - the forcing
with some initial state $u_0(x,t)$ and boundary conditions

Generalization to time-dependent problems

We introduce the time steps



We approximate the time derivatives using two consequtive time steps $\frac{\partial u}{\partial t} \approx \frac{u_{t+1}-u_t}{dt}$

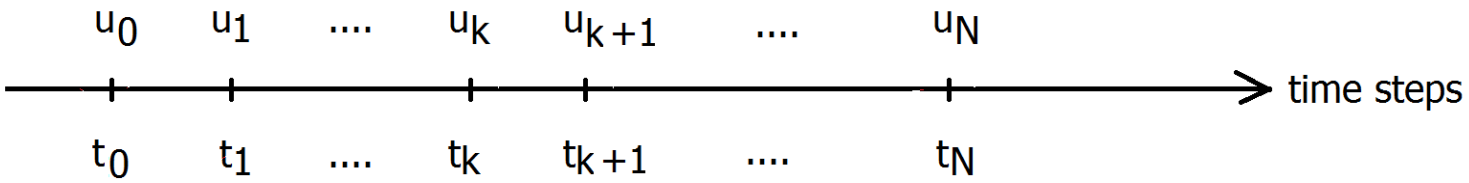
$$\frac{u_{t+1} - u_t}{dt} - \mathcal{L}(u) = f$$

We also introduce the solution from the previous time step into the operator and the forcing (so-called explicit scheme)

$$\frac{u_{t+1} - u_t}{dt} - \mathcal{L}(u_t) = f_t$$

$$u_{t+1} = u_t + dt * \mathcal{L}(u_t) + f_t$$

Generalization to time-dependent problems



Previous state Time step Physics Forcing

$$u_{t+1} = u_t + dt * \mathcal{L}(u_t) + f_t(x,t)$$

Projection „Bitmap"

Sequence of projections:
previous time step state + changes enforced by physics and forcing
$\rightarrow$ next time step state

Generalization to time-dependent problems

Applications to time-dependent problems (C++ code)

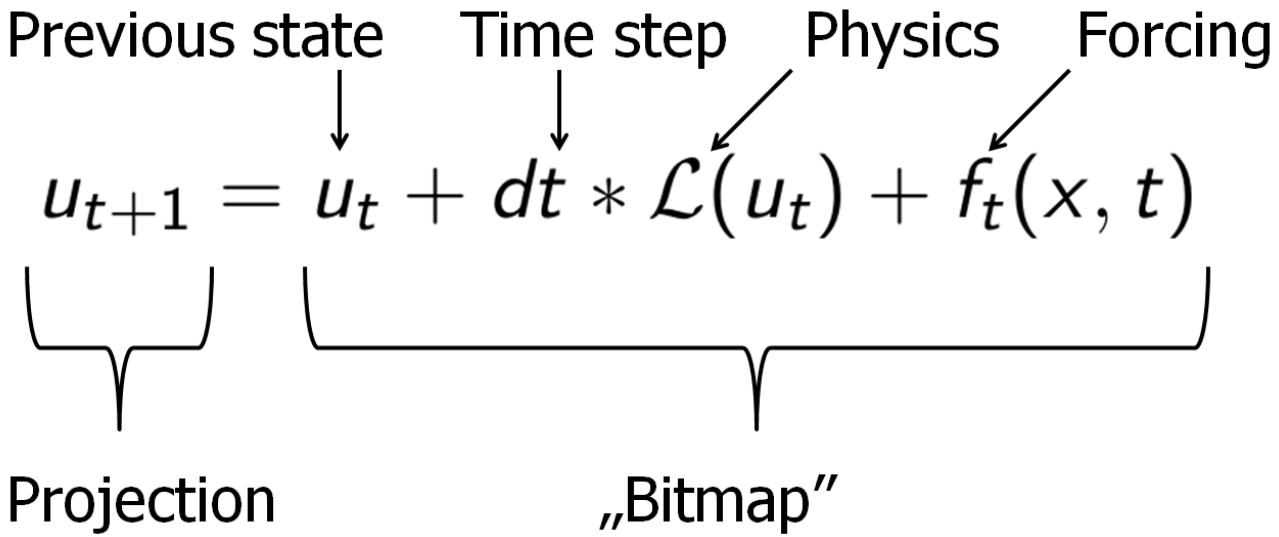
Marcin Łoś, Maciej Woźniak, Maciej Paszyński, Andrew Lenharth, Keshav Pingali *IGA-ADS : Isogeometric Analysis FEM using ADS solver*, **Computer & Physics Communications** 217 (2017) 99-116

Examples

The "physics" operator $\mathcal{L} =$

- heat transfer problem $\mathcal{L}(u) = \Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$
- non-linear flow in heterogeneous media (oil extraction problem) $\mathcal{L}(u) = \nabla \cdot (\kappa(\mathbf{x}, u) \exp(\mu u) \nabla u) =$ $\frac{\partial}{\partial x}\left(\kappa(\mathbf{x}, u) \exp(\mu u) \frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\kappa(\mathbf{x}, u) \exp(\mu u) \frac{\partial u}{\partial y}\right)$
- propagation of the pollutant $\mathcal{L}(u) = \beta \cdot \nabla u - \nabla \cdot (K \nabla u) =$ $\beta_x \frac{\partial u}{\partial x} + \beta_y \frac{\partial u}{\partial y} - \frac{\partial}{\partial x}\left(K_{11} \frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_{22} \frac{\partial u}{\partial y}\right)$
- propagation of elastic waves ($\mathcal{L}$=complicated)
- tumor growth ($\mathcal{L}$=very complicated)

Example 1: Heat transfer equation



The "physics" operator $\mathcal{L}(u_t) = \Delta u_t = \frac{\partial^2 u_t}{\partial x^2} + \frac{\partial^2 u_t}{\partial y^2}$

The forcing $f_t(x,t) = 0$

The initial state = ball of heat in the center of the domain

5000 time steps, time step size $dt = 10^{-7}$

Example 1: Heat transfer equation

Click in the middle

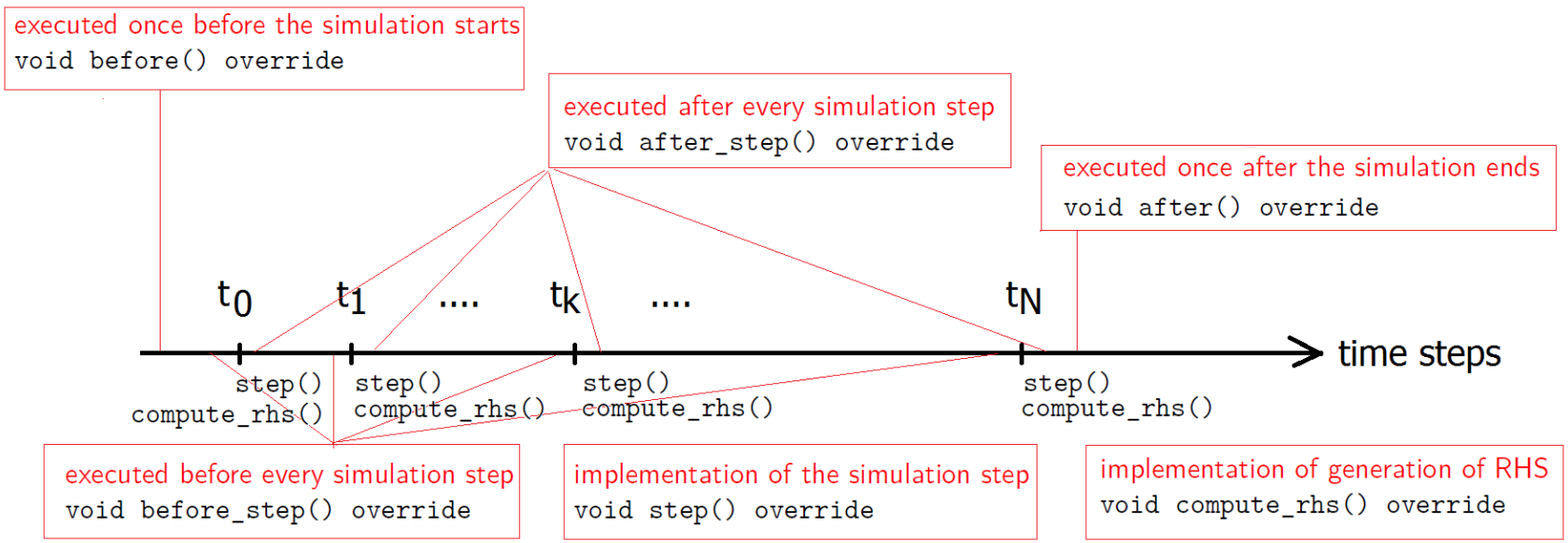
Code for Example 1 (Heat transfer equation)

"problems/heat/heat_2d.cpp"

```
#include "problems/heat/heat_2d.hpp"
using namespace ads;
using namespace ads::problems;
pilot for the simulation
int main() {
quadratic B-splines, 12 elements along axis
  dim_config dim{ 2, 12 };
5000 time steps, time step size 10^-7
  timesteps_config steps{ 5000, 1e-7 };
we will need to compute first derivatives during the computations
  int ders = 1;
some auxiliary objects for configuration and simulation
config_2d c{dim, dim, steps, ders};
heat_2d sim{c};
run the simulation
sim.run();
```

Implementation

Marcin Łoś, Maciej Woźniak, Maciej Paszyński, Andrew Lenharth, Keshav Pingali *IGA-ADS : Isogeometric Analysis FEM using ADS solver*, **Computer & Physics Communications** 217 (2017) 99-116



Code for Example 1 (Heat transfer equation)

"problems/heat/heat_2d.hpp"

```
#include "ads/simulation.hpp"
using namespace ads;
using namespace problems;
class heat_2d :  public simulation_2d {
...
```

implementation of the initial state

```
double init_state(double x, double y, double z)
```

executed once before the simulation starts

```
void before() override
```

executed before every simulation step

```
void before_step() override
```

implementation of the simulation step

```
void step() override
```

executed after every simulation step

```
void after_step() override
```

implementation of generation of RHS

```
void compute_rhs() override
```

executed once after the simulation ends

```
void after() override
```

Code for Example 1 (Heat transfer equation)

"problems/heat/heat_2d.hpp"

this functions is called from the *before* at the beginning of the simulation
the function returns the value of $u_0 = u(x, y)|_{t=0}$)
computed at point (x, y)

```
double init_state(double x, double y) {
    double dx = x - 0.5;
    double dy = y - 0.5;
    double r2 = std::min(8*(dx*dx+dy*dy),1.0);
    return (r2 - 1) * (r2 - 1) * (r2 + 1) * (r2 + 1);
  };
```

Example 1: Heat transfer equation

$$\int_\Omega u_{t+1} v = \int_\Omega u_t v - dt * \int_\Omega \Delta u_t v$$

Integration by parts $-\int_\Omega \Delta u_t v = \int_\Omega \nabla u_t \cdot \nabla v + \int_\Gamma \frac{\partial u}{\partial n} v ds$
and incorporation of boundary condition $\frac{\partial u}{\partial n} = 0$

$$\int_\Omega u_{t+1} v = \int_\Omega u_t v - dt * (\nabla u_t \cdot \nabla v)$$

value of test function a over element e at Gauss point q

```
value_type v = eval_basis(e, q, a);
```

value of u_t at Gauss point

```
value_type u = eval_fun(u_prev, e, q);
```

computations of double gradient

```
double gradient = u.dx*v.dx+u.dy*v.dy;
```

$RHS = u_t - dt \nabla u_t \cdot \nabla v$

```
double val = u.val*v.val - steps.dt * gradient;
```

scale by Jacobian and weight

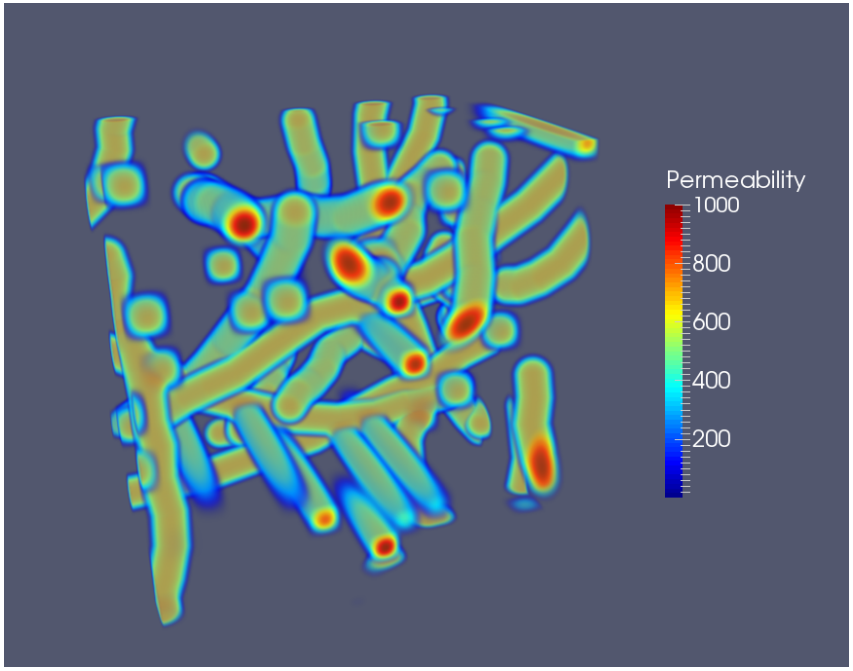
```
rhs(a[0],a[1])+=val*w*J;
```

Code for Example 1 (Heat transfer equation)

```
void compute_rhs() {
auto& rhs = u; zero(rhs);
for (auto e :  elements()) {  loop through elements
  double J = jacobian(e); compute Jacobian
  for (auto q:quad_points()){ loop through Gauss points
   double w = weigth(q);  Gauss weight
   for (auto a :  dofs_on_element(e)){loop through dofs
value of basis function q over element e at Gauss point q
    value_type v = eval_basis(e, q, a);
value of u_t at Gauss point
this also computes derivatives and stored at *.dx
    value_type u = eval_fun(u_prev, e, q);
computations of double gradient
    double gradient = u.dx*v.dx+u.dy*v.dy;
RHS = u_t - dt∇u·∇v
    double val = u.val*v.val - steps.dt * gradient;
scale by Jacobian and weight
    rhs(a[0],a[1])+=val*w*J;
```

Example 2: Non-linear flow in heterogenous media

Hydraulic fracturing - oil/gas extraction technique consisting in high-pressure fluid injection into the deposit



M. Alotaibi, V.M. Calo, Y. Efendiev, J. Galvis, M. Ghommem, *Global-Local Nonlinear Model Reduction for Flows in Heterogeneous Porous Media* **arXiv:1407.0782 [math.NA]**

Example 2: Non-linear flow in heterogenous media

Previous state, Time step, Physics, Forcing

$$u_{t+1} = u_t + dt * \mathcal{L}(u_t) + f_t(x,t)$$

Projection, „Bitmap"

u represents pressure

The "physics" operator $\mathcal{L}(u) = \nabla \cdot (\kappa(\mathbf{x}, u) \exp(\mu u) \nabla u) = \frac{\partial}{\partial x}\left(\kappa(\mathbf{x}, u) \exp(\mu u) \frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\kappa(\mathbf{x}, u) \exp(\mu u) \frac{\partial u}{\partial y}\right)$

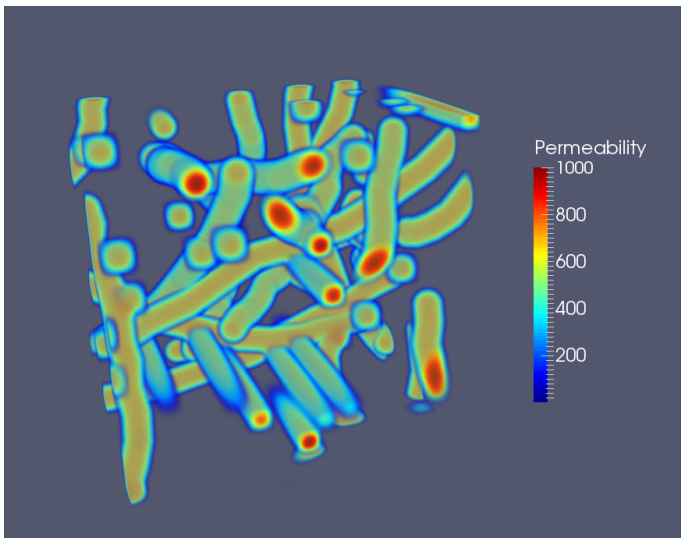


Figure: κ property of the terrain (permeability), $\mu = 10$

Example 2: Non-linear flow in heterogenous media

Previous state — Time step — Physics — Forcing

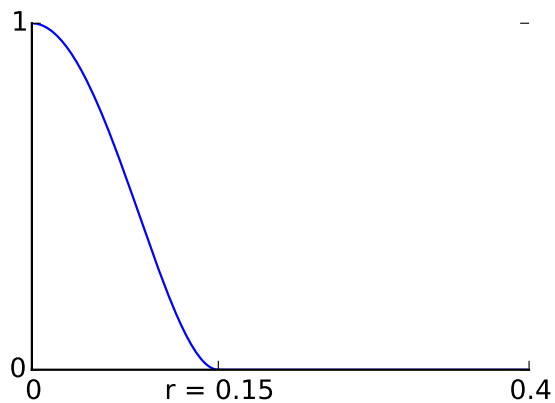
$$u_{t+1} = u_t + dt * \mathcal{L}(u_t) + f_t(x,t)$$

Projection — „Bitmap"

Forcing f represents **pumps** P and **sinks** S, increasing and decreasing the pressure locally at $x_p \in P$, $x_s \in S$ locations

$$f(x,t) = \sum_{p \in P} \phi\left(\|x_p - x\|\right) - \sum_{s \in S} u(x,t)\phi\left(\|x_s - x\|\right)$$

$$\phi(t) = \begin{cases} \left(\frac{t}{r} - 1\right)^2 \left(\frac{t}{r} + 1\right)^2 & \text{for } t \leq r \\ 0 & \text{for } t > r \end{cases}$$



Example 2: Non-linear flow in heterogenous media

Click in the middle

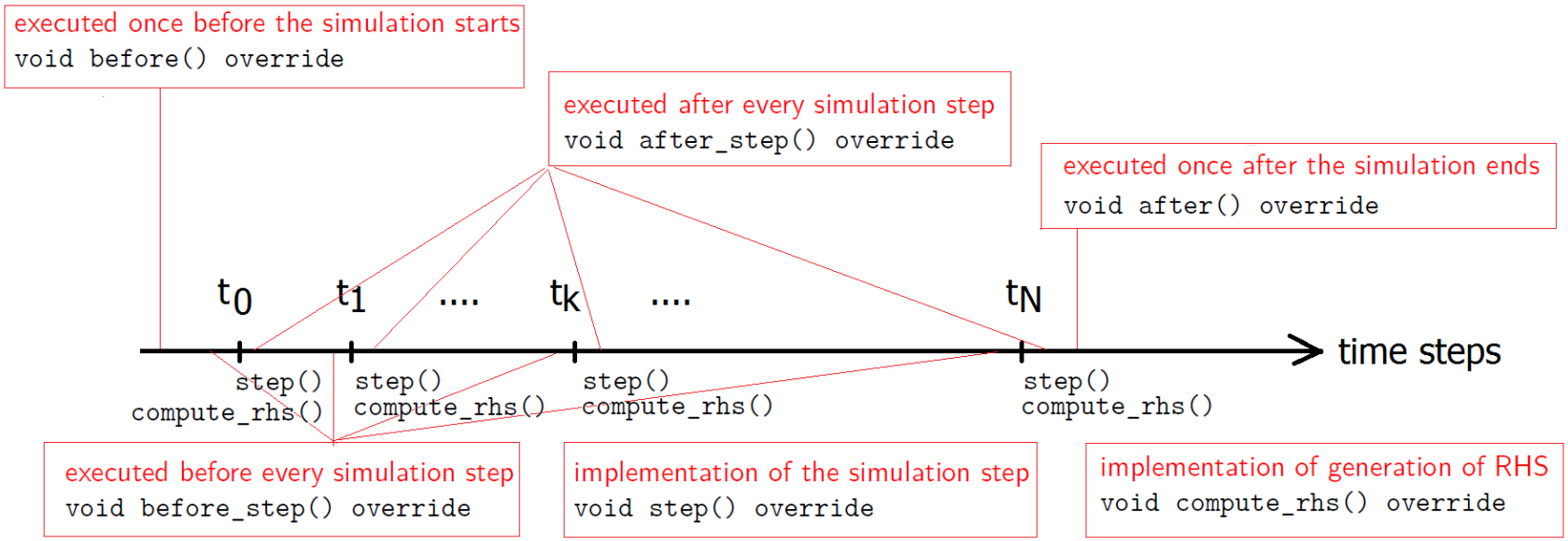
Code for Example 2 (Non-linear flow)

"problems/flow/flow.cpp"

```
#include "problems/flow/flow.cpp"
using namespace ads;
using namespace ads::problems;
pilot for the simulation
int main() {
quadratic B-splines, 20 elements along axis
  dim_config dim{ 2, 20 };
10000 time steps, time step size 10^-7
  timesteps_config steps{ 10000, 1e-7 };
we will need to compute first derivatives during the computations
  int ders = 1;
some auxiliary objects for configuration and simulation
config_3d c{dim, dim, dim, steps, ders};
flow_3d sim{c};
run the simulation
sim.run();
```

Implementation

Marcin Łoś, Maciej Woźniak, Maciej Paszyński, Andrew Lenharth, Keshav Pingali *IGA-ADS : Isogeometric Analysis FEM using ADS solver*, **Computer & Physics Communications** 217 (2017) 99-116



Code for Example 2 (Non-linear flow)

"problems/flow/geometry.hpp"

this functions is called from the *before* at the beginning of the simulation
the function returns the value of $u_0 = u(x, y, z)|_{t=0}$) computed at (x, y, z)

```
double init_state(double x, double y, double z) {
    double r = 0.1;
    double R = 0.5;
    return ads::bump(r, R, x, y, z);
  };

// r < R in [0, 1]
inline double bump(double r, double R, double x, double y, double
z) {     double dx = x - 0.5, dy = y - 0.5, dz = z - 0.5;
    double t = std::sqrt(dx * dx + dy * dy + dz * dz);
    return falloff(r / 2, R / 2, t);
}
```

this functions is called from the *compute_rhs*

```
inline double falloff(double r, double R, double t) {
    if (t < r) return 1.0;
    if (t > R) return 0.0;
    double h = (t - r) / (R - r);
    return std::pow((h - 1) * (h + 1), 2);
```

Code for Example 2 (Non-linear flow)

"problems/flow/flow.hpp"

this functions is also called from the *before* at the beginning of the simulation

```
void fill_permeability_map() {
for(auto e:elements()) {  loop through elements
  for(auto q:quad_points()){loop through Gauss points
   auto x = point(e, q);
   kq(e[0], e[1], e[2], q[0], q[1], q[2]) =
    env.permeability(x[0], x[1], x[2]);
  }
}

double permeability(index_type e, index_type q) const {
  return kq(e[0], e[1], e[2], q[0], q[1], q[2]);
}
```

Code for Example 2 (Non-linear flow)

"problems/flow/flow.hpp"

```
double forcing(point_type x, double /*t*/) const {
  using std::sin;
  double pi4 = 4 * M_PI;
  four pumps one sink
  double dx1=x-0.25; dy1=y-0.25; dz1=z-0.25;
  double dx2=x-0.75; dy2=y-0.25; dz2=z-0.25;
  double dx3=x-0.25; dy3=y-0.75; dz3=z-0.25;
  double dy4=x-0.25; dy4=y-0.25; dz4=z-0.75;
  double dz5=x-0.5; dz5=y-0.5; dz5=z-0.5;
  return
   max(0,sin(pi4*dx1)*sin(pi4*dy1)*sin(pi4*dz1))
   +max(0,sin(pi4*dx2)*sin(pi4*dy2)*sin(pi4*dz2))
   +max(0,sin(pi4*dx3)*sin(pi4*dy3)*sin(pi4*dz3))
   +max(0,sin(pi4*dx4)*sin(pi4*dy4)*sin(pi4*dz4))
   -max(0,sin(pi4*dx5)*sin(pi4*dy5)*sin(pi4*dz5));
}
```

Code for Example 2 (Non-linear flow)

$$\int_\Omega u_{t+1} v = \int_\Omega (u_t + f)v + dt * \int_\Omega \nabla \cdot (\kappa(\mathbf{x}, u) \exp(\mu u) \nabla u) v$$

integration by parts $\int_\Omega \nabla \cdot (\kappa(\mathbf{x}, u) \exp(\mu u) \nabla u) v =$
$\int_\Omega (\kappa\, e^{10 * u_t} \nabla u_t \cdot \nabla v) + \int_\Gamma \kappa\, e^{10 * u_t} \frac{\partial u_t}{\partial n} v$ and using Neumann b.c.

$$\int_\Omega u_{t+1} v = \int_\Omega (u_t + f)v - dt * \int_\Omega (K_q(x)\, e^{10 * u_t} \nabla u_t \cdot \nabla v)$$

u_t value over element e at Gauss point q

```
value_type u = eval_fun(u_prev, e, q);
```

the forcing is based on the location of pumps and sinks

```
h = forcing(x, t);
```

value of test function a over element e at Gauss point q

```
  value_type v = eval_basis(e, q, a);
```

$(-K_q(x)\, e^{10 * u_t}, \nabla u_t)_{L_2} + (h, \nabla v)_{L^2}$

```
  val= -k*std::exp(10*u.val)*grad_dot(u, v)+h*v.val;
```

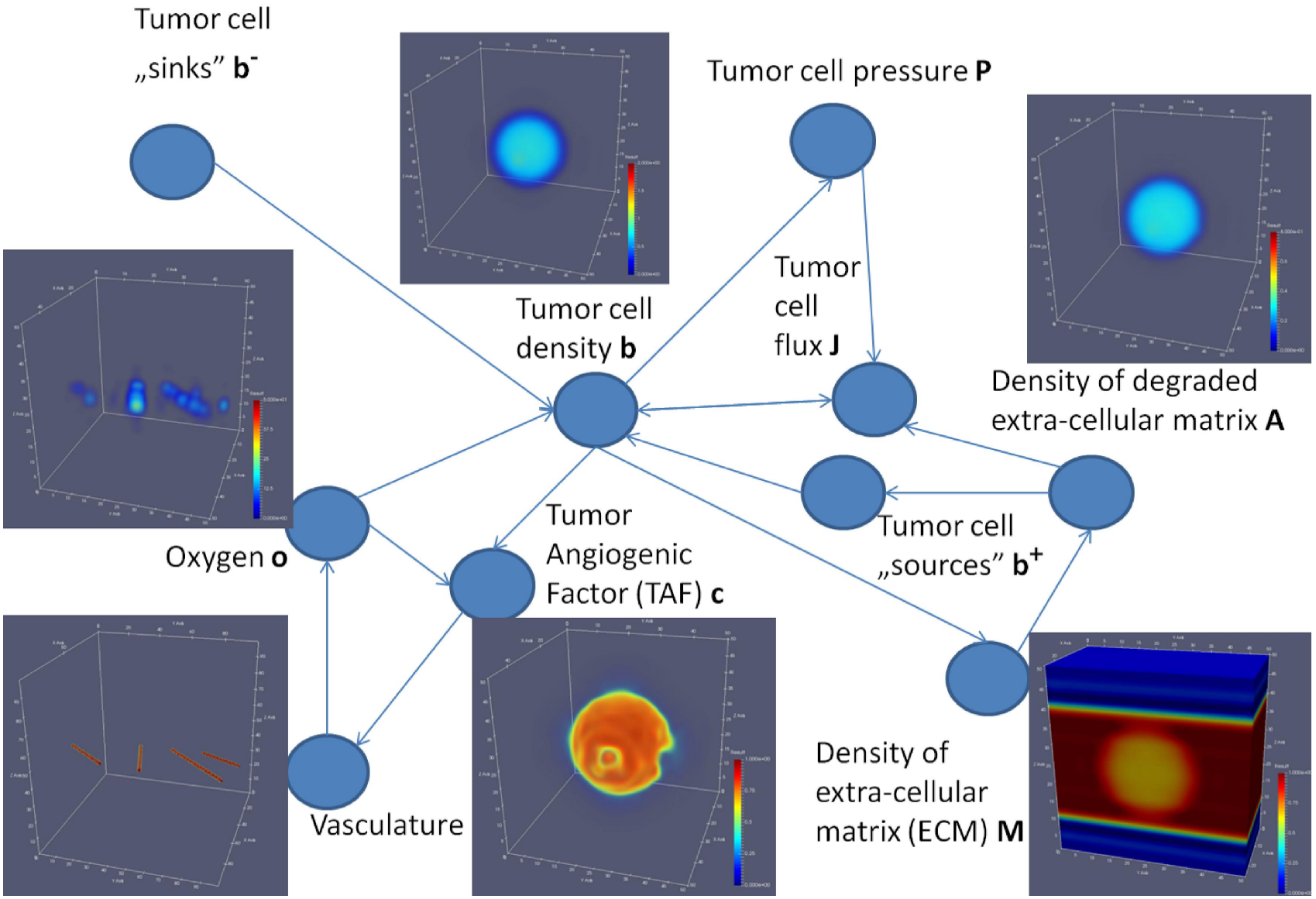
$(u_t + h, w)_{L_2} - dt * (\kappa\, e^{10 * u_t} \nabla u_t, \nabla w)_{L^2}$ *Jacobian*weight

```
U(aa[0],aa[1],aa[2])+=(u.val*v.val+steps.dt*val)*w*J;
```

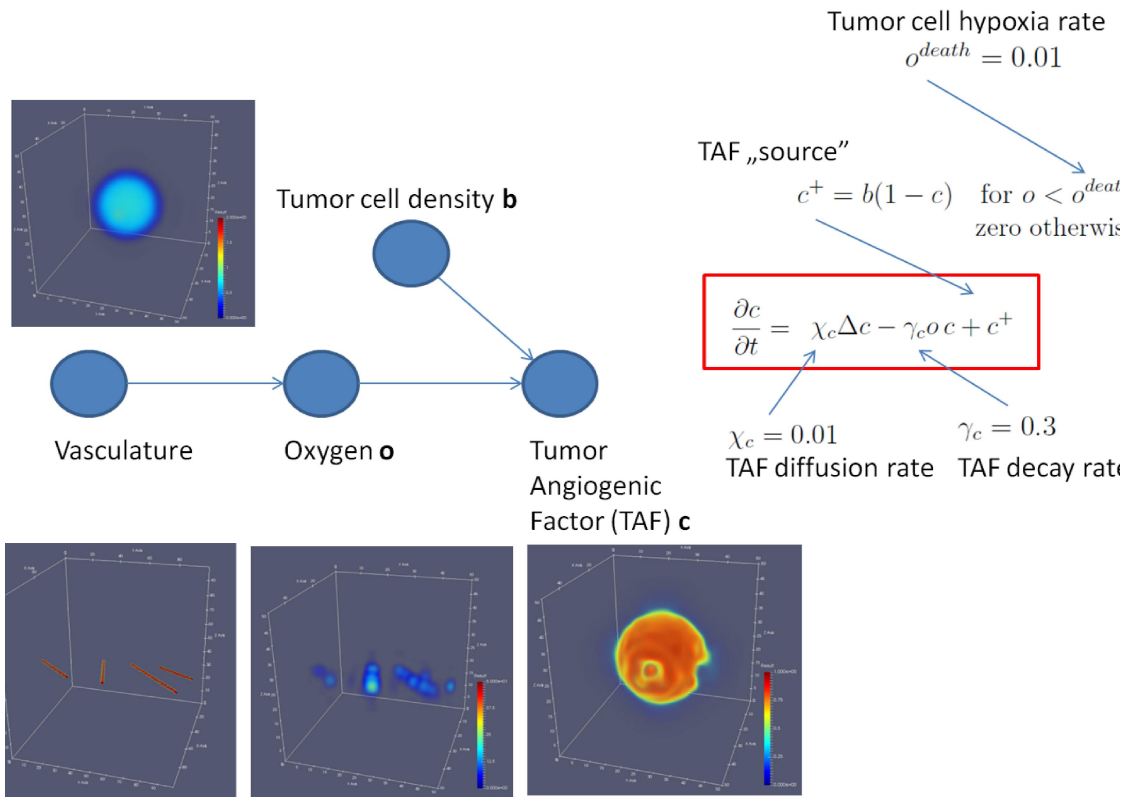
Code for Example 2 (Non-linear flow)

```
parallel processing of loop through elements
executor.for_each(elements(), [&](index_type e) {
  double J = jacobian(e);
  for (auto q :  quad_points()) { Gauss points
weight and pointt for Gaussian quadrature
   double w = weigth(q); auto x = point(e, q);
value of permeability at Gauss point
   double k = permeability(e, q);
u_t value over element e at Gauss point q
   value_type u = eval_fun(u_prev, e, q);
the forcing is based on the location of pumps and sinks
   double h = forcing(x, t);
   for (auto a:dofs_on_element(e)) { test functions
remapping local to global index for aggregation of RHS
    auto aa = dof_global_to_local(e, a);
value of test function a over element e at Gauss point q
    value_type v = eval_basis(e, q, a);
    double val = - k*std::exp(10*u.val)*grad_dot(u, v)+h*v.val;
    U(aa[0],aa[1],aa[2])+=(u.val*v.val+steps.dt*val)*w*J;
the update of RHS must be synchronized when processed in parallel
executor.synchronized( [ & ] { update_global_rhs(rhs, U, e); });
```

Example 3: Melanoma growth model

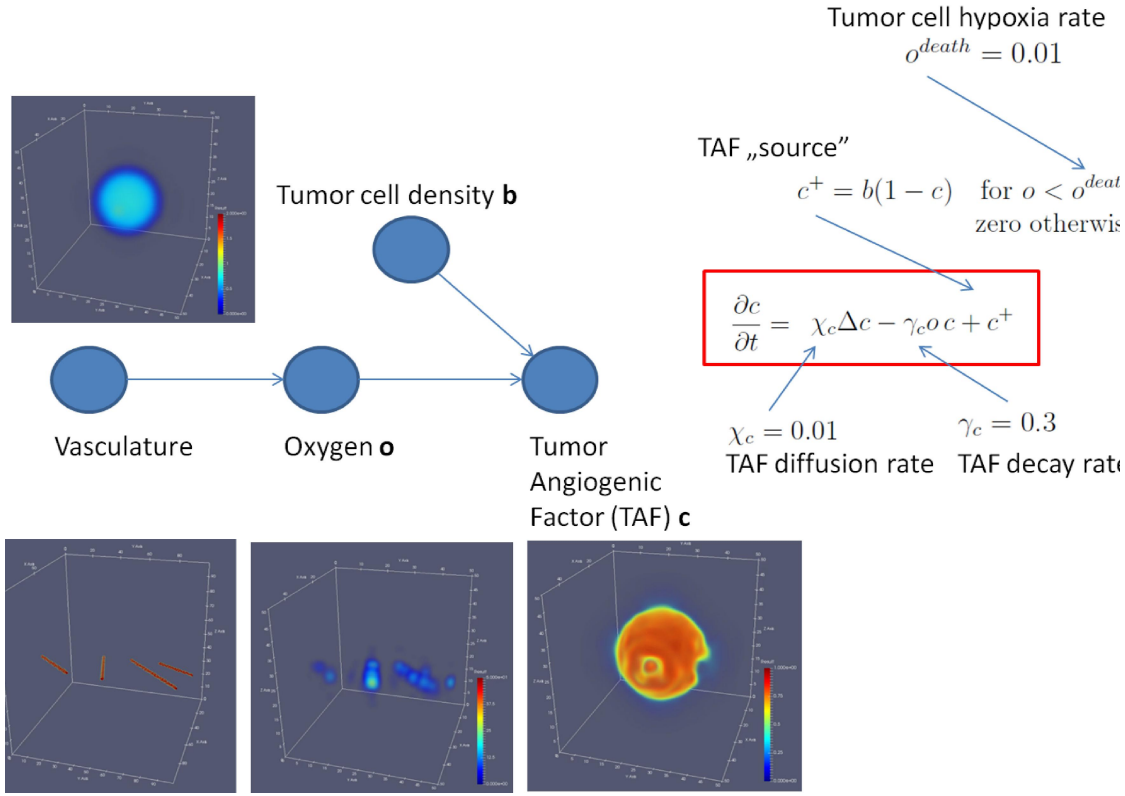


Example 3: Tumor Angiogenic Factor (TAF) c



- produced by oxygen-starved tumor cells
- signal to the vasculature – „more oxygen is needed here"
- influences vasculature evolution (discrete model)

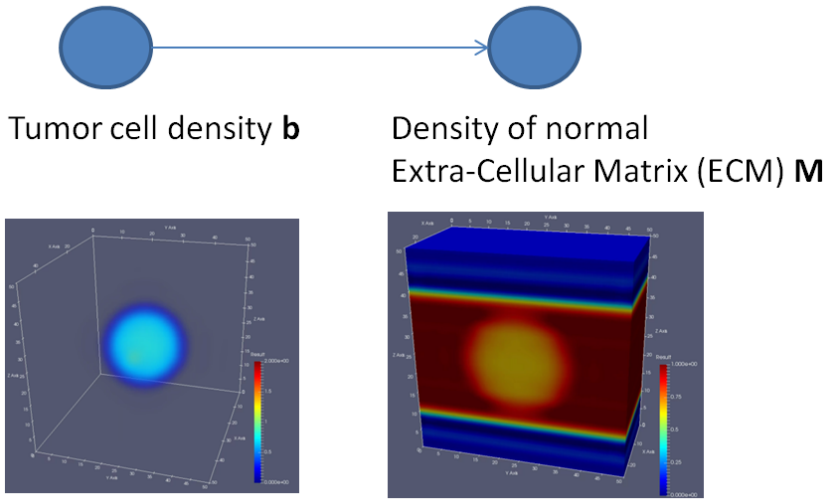
Example 3: Tumor Angiogenic Factor (TAF) c



$$\frac{\partial c}{\partial t} = \chi_c \Delta c - \gamma_c o\, c + c^+$$

- $\chi_c = 0.01$ TAF diffusion rate, $\gamma_c = 0.3$ TAF decay rate
- c^+ – TAF "source" $c^+ = b(1-c) \quad \text{for } o < o^{death}$
- $o^{death} = 0.01$ TAF hypoxia rate

Example 3: Density of normal extra-cellular matrix (ECM)

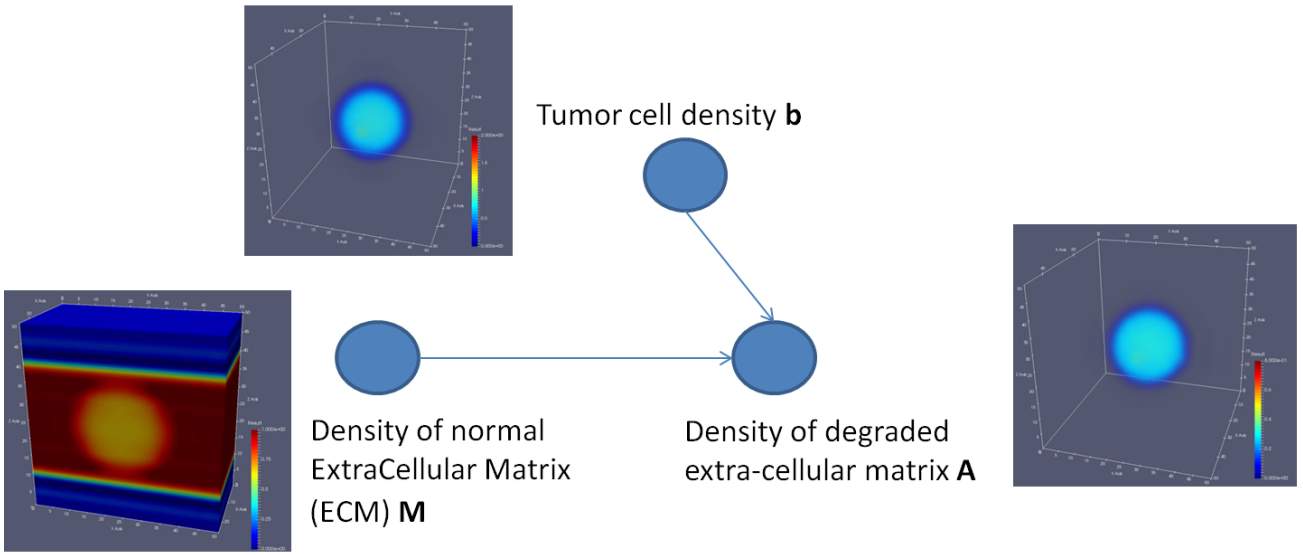


- provides support for the cell structures

$$\frac{\partial M}{\partial t} = -\beta_M M b$$

- $\beta_M = 1.0$ – ECM decay rate

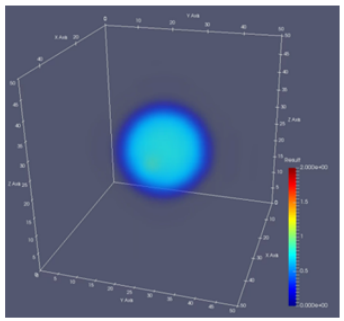
Example 3: Density of degraded extra-cellular matrix A

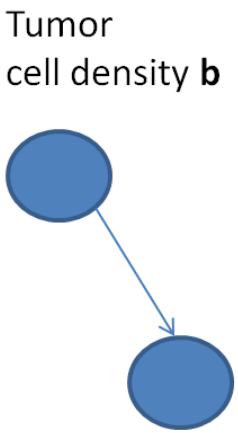


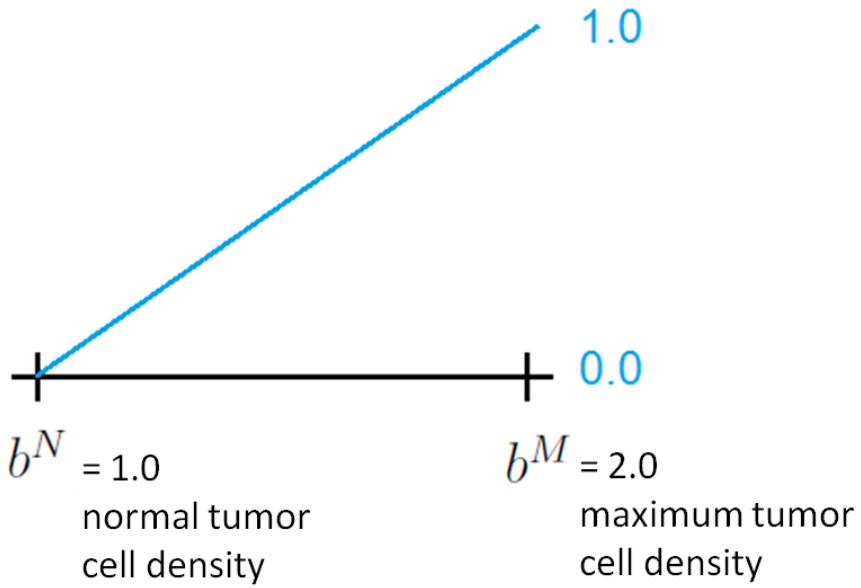
$$\frac{\partial A}{\partial t} = \gamma_A M b + \chi_{aA} \Delta A - \gamma_{oA} A$$

- $\gamma_A = 0.5$ – production rate of artefacts
- $\chi_{aA} = 0.01$ – diffusion rate of degraded extra-cellular matrix
- $\gamma_{oA} = 0.01$ – decay rate of degraded extra-cellular matrix

Example 3: Tumor cell pressure



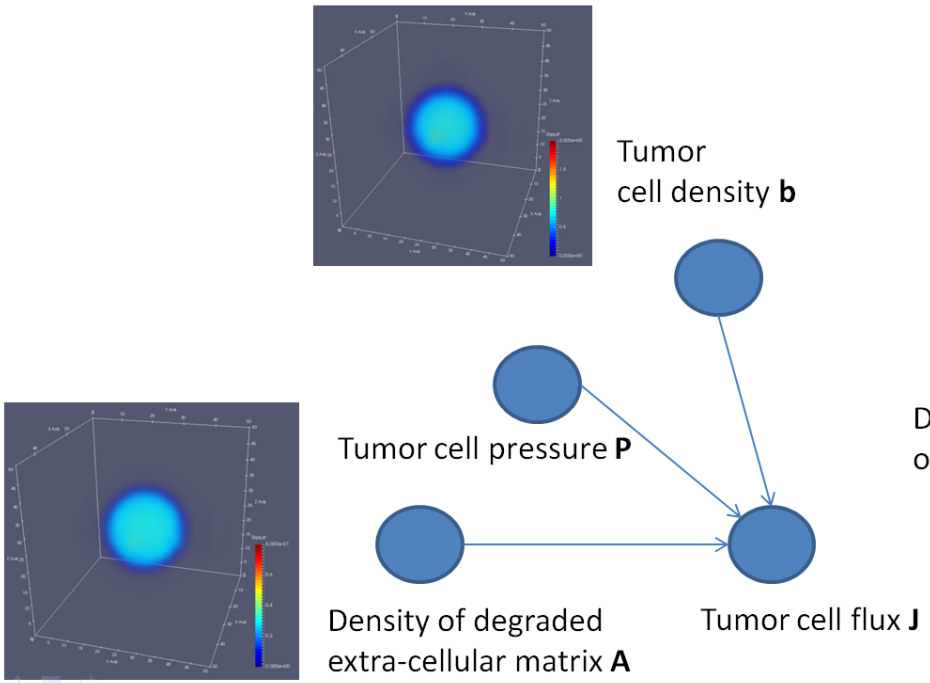


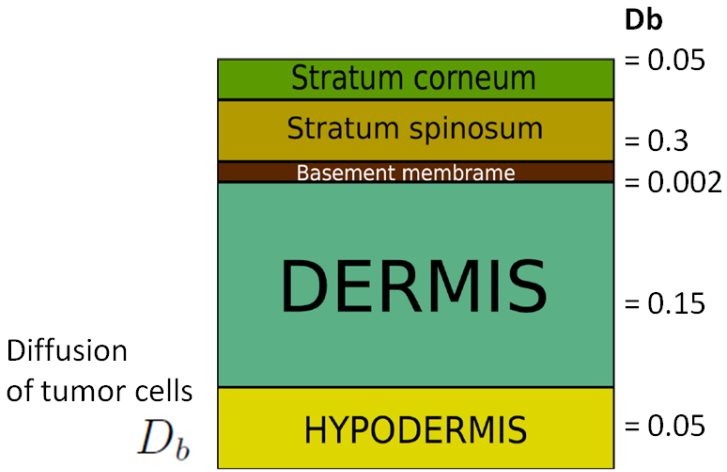


- P – tumor pressure, present for tumor cell density exceeding b^N

$$P = \begin{cases} 0 & \text{for } b < b^N \\ \frac{b - b^N}{b^M - b^N} & \text{for } b^N \leq b \leq b^M \end{cases}$$

Example 3: Tumor cell flux



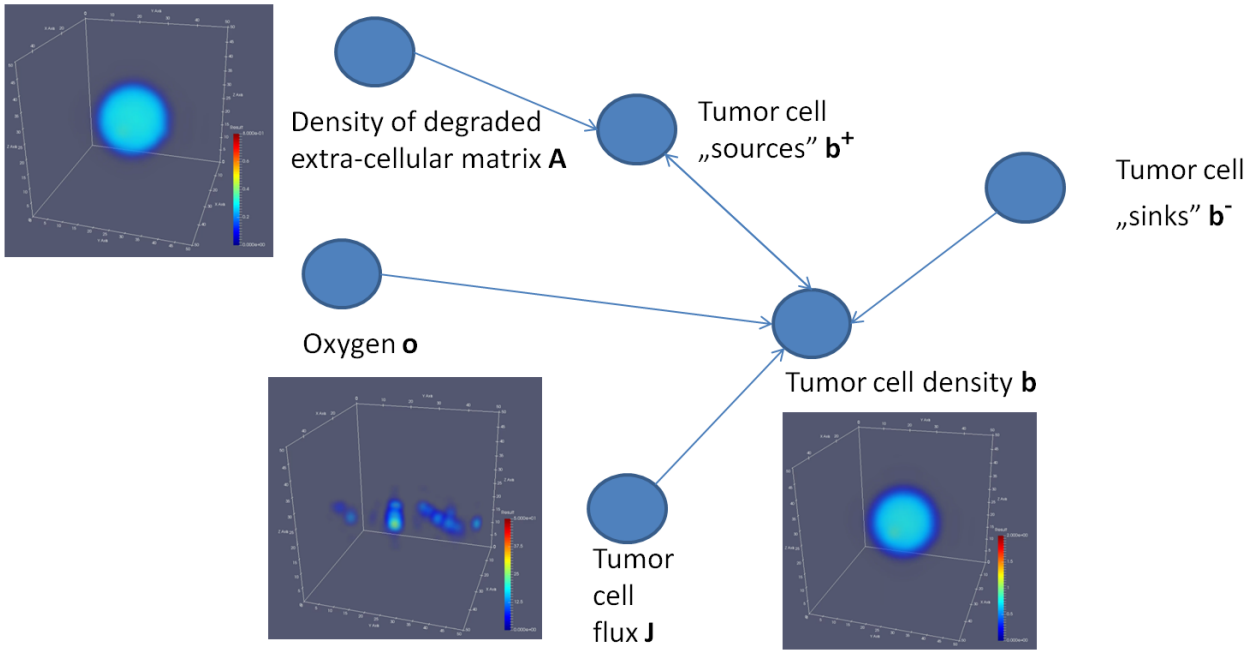


- J – induced by pressure of tumor and extracellular matrix

$$J = -D_b\, b\, (\nabla P + r_b \nabla A)$$

- D_b – cell diffusion coefficient
- $r_b = 0.0001$ – tumor cell chemoattractand sensitivity

Example 3: Tumor cell density



- values between $b^m = 0$ (no cancer cells) and $b^M = 2$
- $b^N = 1$ – *normal* tumor cell density

$$\frac{\partial b}{\partial t} = -\nabla \cdot J + b^- + b^+$$

- b^+, b^- – tumor cell proliferation and apopthosis factors

Example 3: Tumor cell proliferation/death

b^+, b^- – governed by the oxygen concentration o

- $o > o^{prol} = 0.1$ – tumor cells multiply ($b^+ > 0$)
- $o < o^{death} = 0.01$ – tumor cells die ($b^- > 0$)

$$b^+ = \frac{b}{T^{prol}} \left(1 + \frac{\tau_b A}{\tau_b A + 1} P_b\right) \left(1 - \frac{b}{b^M}\right) \qquad \text{for } o > o^{prol}$$

$$b^- = -\frac{b}{T^{death}} \qquad \text{for } o < o^{death}$$

- $T^{prol} = 10$ tumor cell proliferation time
- $T^{death} = 100$ tumor cell survival time
- $\tau_b = 0.5$ instantaneous reaction rate constant
- tumor cell grows if the density is not too high

Numerical results

Initial state:

- tumor concentrated in the center of the domain
- constant ECM in each skin layer
- no TAF, no degraded ECM

Parameters:

- $32 \times 32 \times 32$ elements
- cubic B-splines ($p = 3$)
- $\Delta t = 1$
- 1000 time steps
- 30 minutes with parallel GALOIS solver on 16 cores of GILBERT

M. Łoś, M. Paszyński, A. Kłusek, W. Dzwinel, Application of fast isogeometric l2 projection solver for tumor growth simulations, Computer Methods in Applied Mechanics and Engineering 316 (2017) 1257-1269.

3D simulation (1/5) TAF c

Click in the middle

3D simulation (2/5) Vasculature

Click in the middle

3D simulation (3/5) Oxygen

Click in the middle

3D simulation (4/5) Tumor cell density b

Click in the middle

3D simulation (5/5) Density of extra cellular matrix

Click in the middle

Numerical formulation

Explicit time discretization:

$$\begin{cases} b_{t+1} = b_t + \Delta t \left(-\nabla \cdot J_t + b_t^- + b_t^+\right) \\ c_{t+1} = c_t + \Delta t \left(\chi_c \Delta c_t - \gamma_c o_t c_t + c_t^+\right) \\ o_{t+1} = o_t + \Delta t \left(\alpha_0 \Delta o_t - \gamma_o b_t o_t + \delta_o (o^{\max} - o_t)\, o^{\text{src}}\right) \\ M_{t+1} = M_t + \Delta t \left(-\beta_M M_t b_t\right) \\ A_{t+1} = A_t + \Delta t \left(\gamma_A M_t b_t + \chi_{OA} \Delta A_t - \gamma_{OA} A_t\right) \end{cases}$$

$$J = -D_b\, b \left(\nabla P + r_b \nabla A\right)$$

$$P = \begin{cases} 0 & \text{for } b < b^N \\ \frac{b - b^N}{b^M - b^N} & \text{for } b^N \le b \le b^M \end{cases}$$

$$b^+ = \frac{b}{T^{prol}} \left(1 + \frac{\tau_b A}{\tau_b A + 1} P_b\right) \left(1 - \frac{b}{b^M}\right) \qquad \text{for } o > o^{prol}$$

$$b^- = -\frac{b}{T^{death}} \qquad \text{for } o < o^{death}$$

Numerical formulation

Sequence of isogeometric L^2-projections to be solved in every time step

$$\begin{cases} (b_{t+1}, B_i)_{L^2} = (b_t, B_i)_{L^2} + \Delta t\,(-\nabla \cdot J_t + b_t^- + b_t^+, B_i)_{L^2} \\ (c_{t+1}, B_i)_{L^2} = (c_t, B_i)_{L^2} + \Delta t\,(\chi_c \Delta c_t - \gamma_c o_t c_t + c_t^+, B_i)_{L^2} \\ (o_{t+1}, B_i)_{L^2} = (o_t, B_i)_{L^2} + \Delta t\,(\alpha_0 \Delta o_t - \gamma_o b_t o_t + \delta_o (o^{\max} - o_t)\, o^{\mathrm{src}}, \\ (M_{t+1}, B_i)_{L^2} = (M_t, B_i)_{L^2} + \Delta t\,(-\beta_M M_t b_t, B_i)_{L^2} \\ (A_{t+1}, B_i)_{L^2} = (A_t, B_i)_{L^2} + \Delta t\,(\gamma_A M_t b_t + \chi_{OA} \Delta A_t - \gamma_{OA} A_t, B_i)_{L^2} \end{cases}$$

$u = (b, c, o, M, A) \Rightarrow (u_{t+1}, B_i)_{L^2} = (u_t, B_i)_{L^2} + F(u_t, B_i)$

The most expensive part is not the factorization, but the numerical integration.

Integration loop – parallel version

for *each element* $E = [\xi_{l_x}, \xi_{l_x+1}] \times \left[\xi_{l_y}, \xi_{l_y+1}\right] \times [\xi_{l_z}, \xi_{l_z+1}]$ *in parallel* **do**
  $U^{loc} \leftarrow 0$;
  for *each quadrature point* $\boldsymbol{\xi} = \left(X_{k_x}, X_{k_y}, X_{k_z}\right)$ **do**
    $\mathbf{x} \leftarrow \Psi_E(\boldsymbol{\xi})$;
    $W \leftarrow w_{k_x} w_{k_y} w_{k_z}$;
    $u, Du \leftarrow 0$;
    for $I \in \mathcal{I}(E)$ **do**
      $u \leftarrow u + U_I^{(t)} \mathcal{B}_I(\boldsymbol{\xi})$;
      $Du \leftarrow Du + U_I^{(t)} \nabla \mathcal{B}_I(\boldsymbol{\xi})$;
    end
    for $I \in \mathcal{I}(E)$ **do**
      $v \leftarrow \mathcal{B}_I(\boldsymbol{\xi})$;
      $Dv \leftarrow \nabla \mathcal{B}_I(\boldsymbol{\xi})$;
      $U_I^{loc} \leftarrow U_I^{loc} + W\,|E|\,(uv + \Delta t\, F(u, Du, v, Dv))$;
    end
  end
  synchronized
    for $I \in \mathcal{I}(E)$ **do**
      $U_I^{(t+1)} \leftarrow U_I^{(t+1)} + U_I^{loc}$
    end
  end
end

Implementation: `Galois::for_each`, `Galois::Runtime::LL::SimpleLock`

Strong scaling for quadratic B-splines

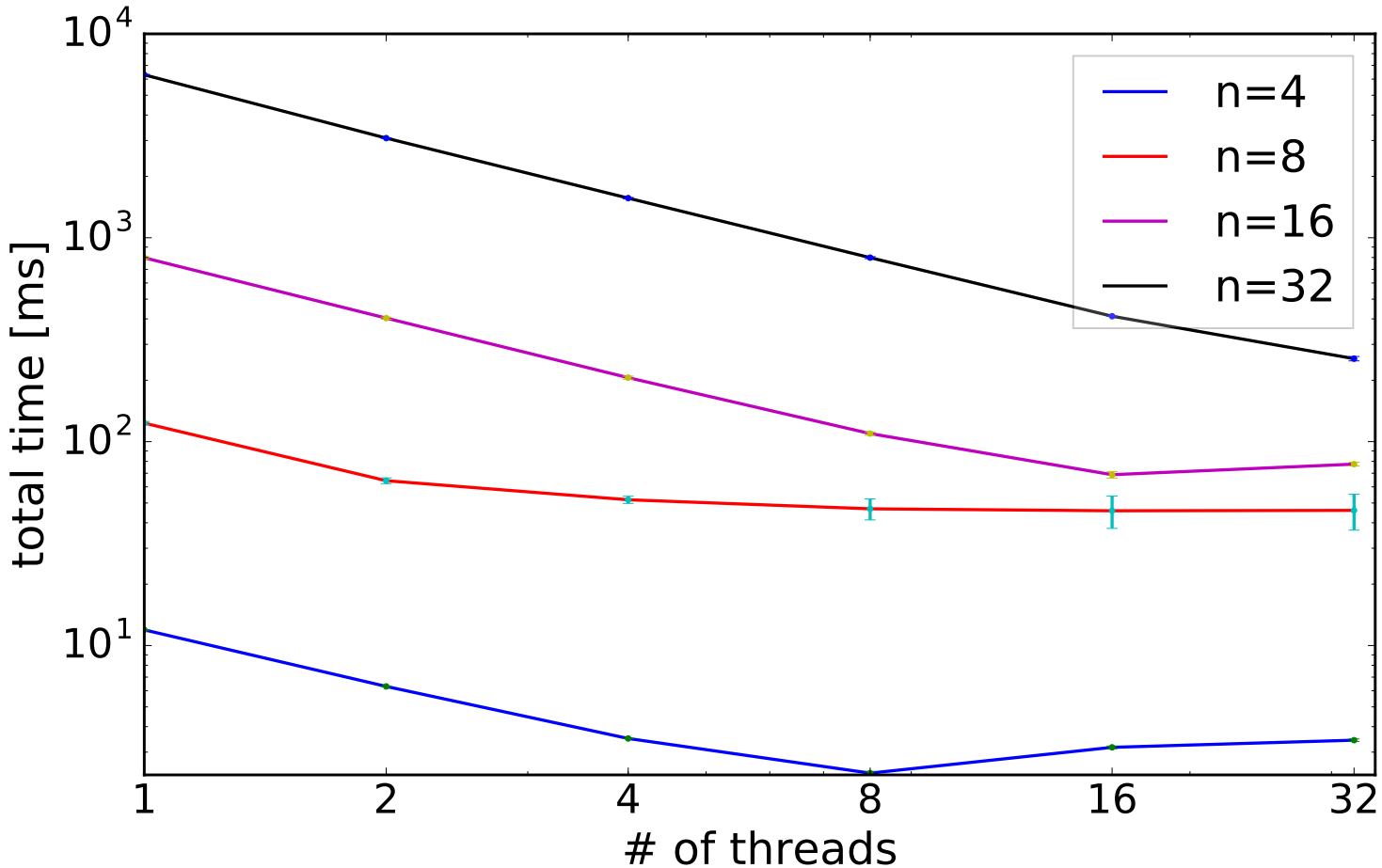


Figure: Strong scaling for quadratic B-splines

A straight line denotes ideal strong scalability, while any upward deviation from the line denotes limited strong scalability.

Strong scaling for cubic B-splines

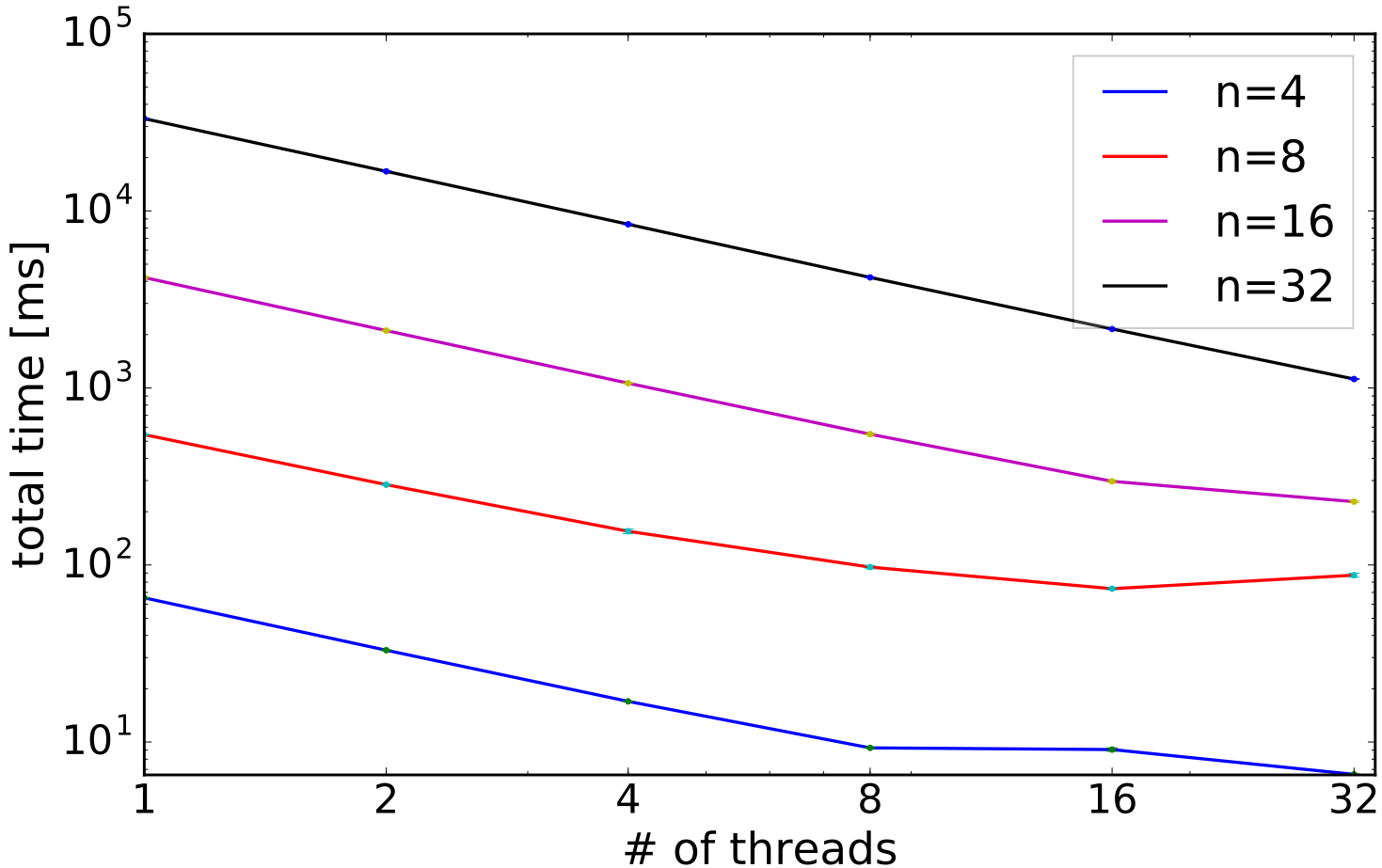


Figure: Strong scaling for cubic B-splines

A straight line denotes ideal strong scalability, while any upward deviation from the line denotes limited strong scalability.

Serial fraction

The serial fraction of the algorithm can be determined from the Amdahl's law

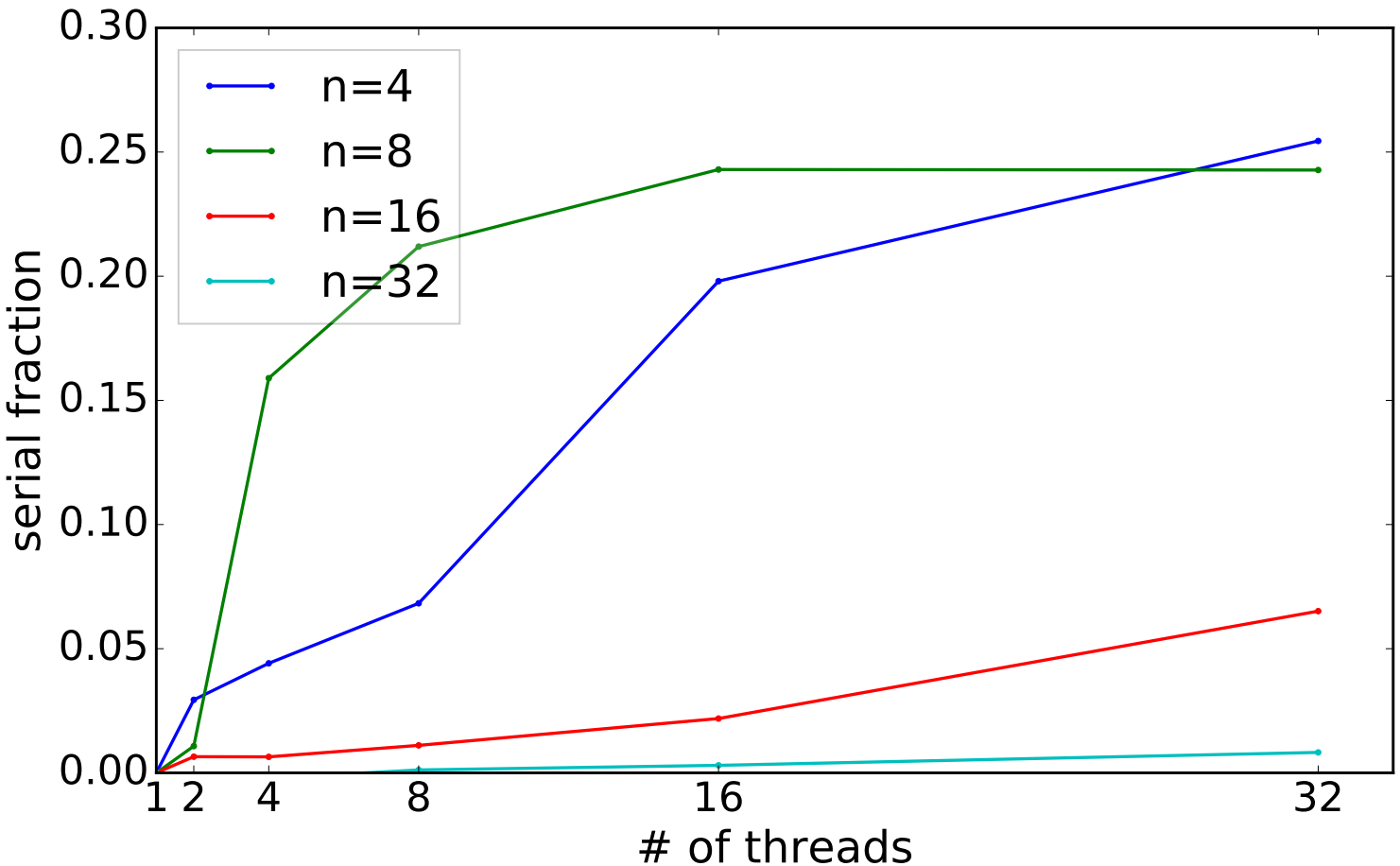
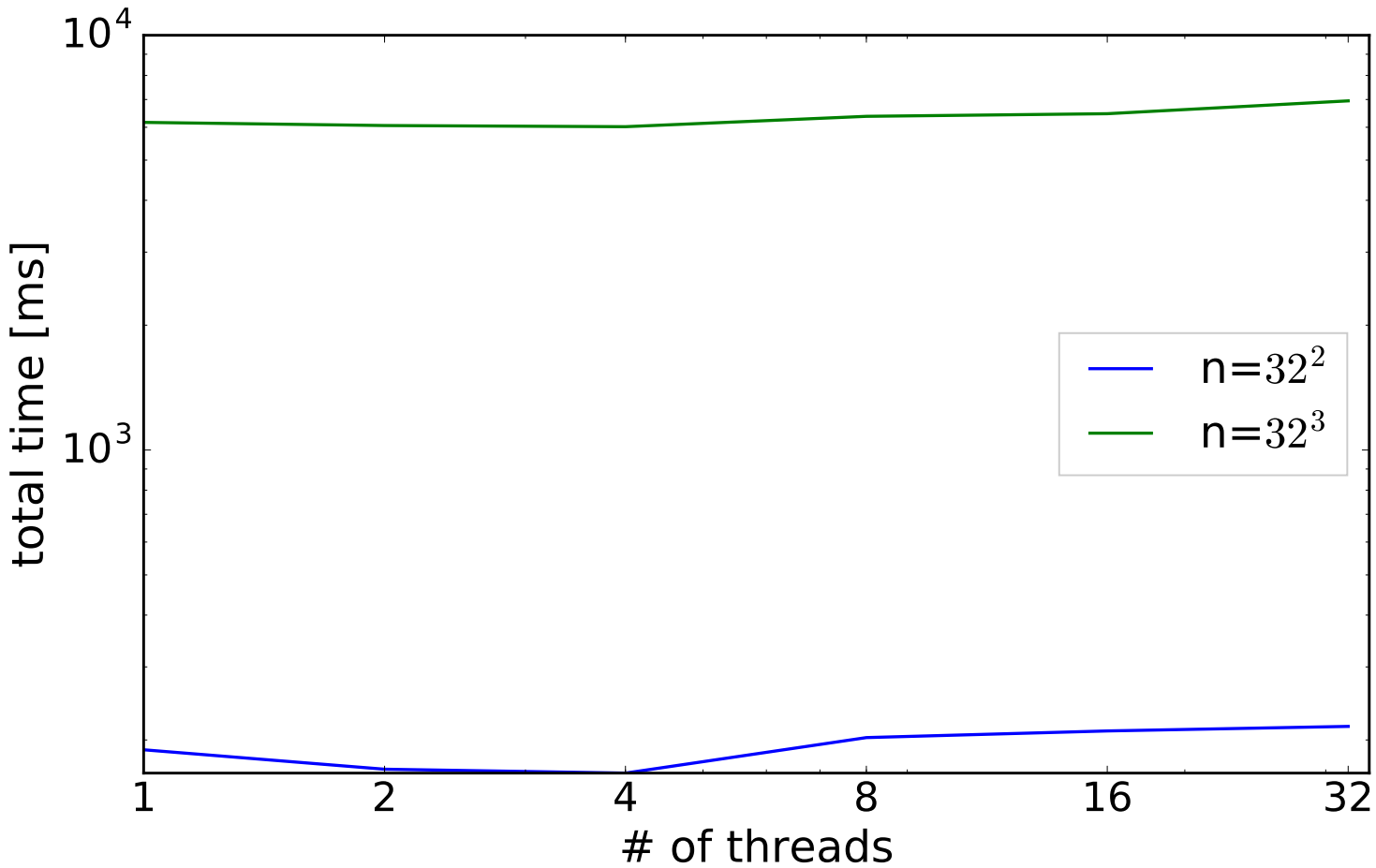


Figure: Sequential fraction of the parallel code for quadratic B-splines

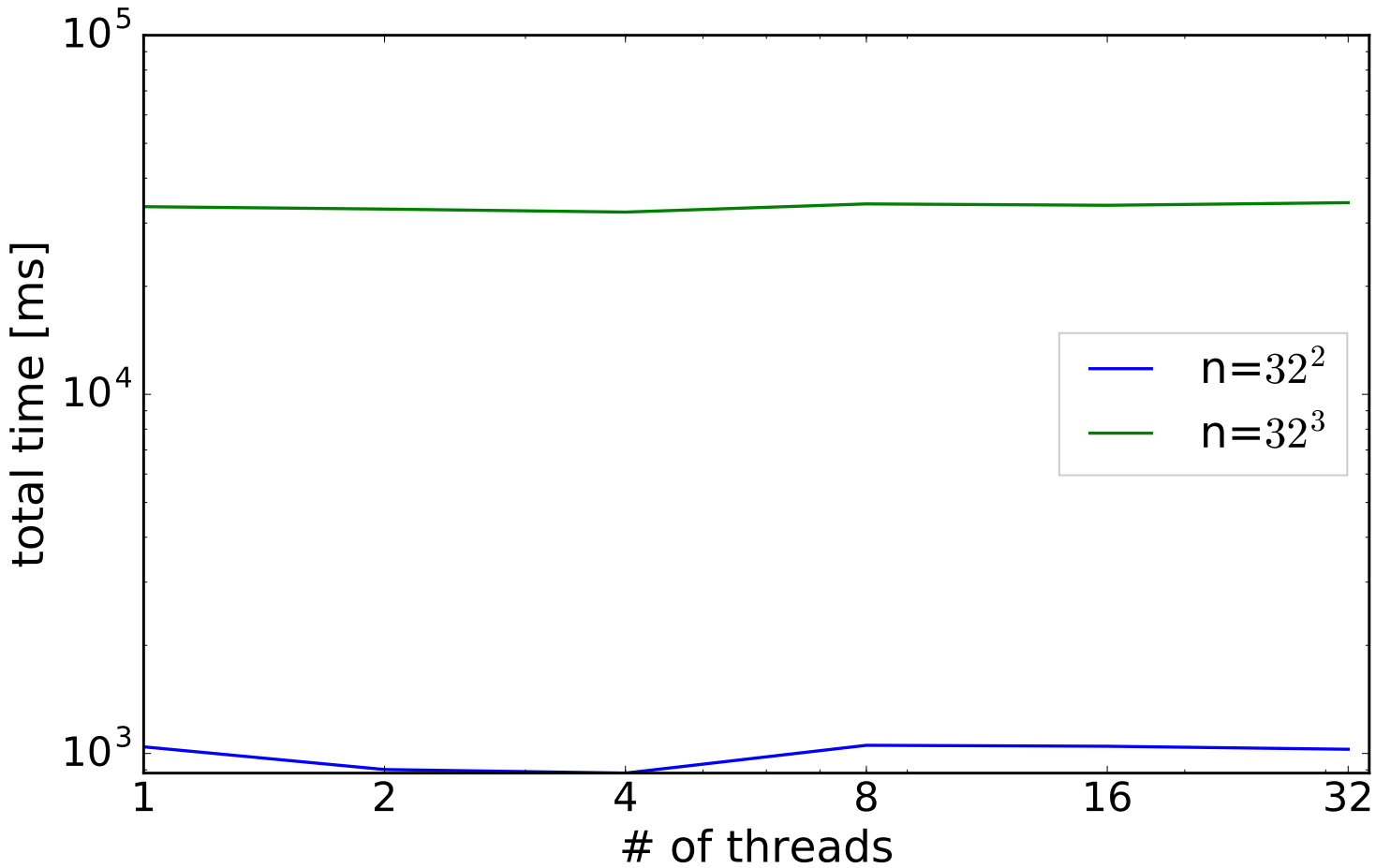
Weak scaling for quadratic B-splines

Each core is processing either $32 * 32$ elements (first experiment) or $32 * 32 * 32$ elements (second experiment), and we increase the problem size and number of cores. The horizontal line denotes ideal weak scaling, while the upward deviation represents limited scaling.



Weak scaling for cubic B-splines

Each core is processing either $32 * 32$ elements (first experiment) or $32 * 32 * 32$ elements (second experiment), and we increase the problem size and number of cores. The horizontal line denotes ideal weak scaling, while the upward deviation represents limited scaling.



Implicit dynamics

$$\frac{du}{dt} - Lu = f$$

assuming constant coefficients and regular cube shape domain, where $L = L_x + L_y$ is a separable differential operator, e.g. Laplacian, where $L = L_x + L_y = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$. First, we apply the alternating direction method with respect to time. We introduce the intermediate time steps

$$\frac{u_{t+0.5} - u_t}{dt} - L_x u_{t+0.5} - L_y u_t = f_t$$

$$\frac{u_{t+1} - u_{t+0.5}}{dt} - L_x u_{t+0.5} - L_y u_{t+1} = f_{t+0.5}$$

We obtain

$$u_{t+0.5} - dt * L_x u_{t+0.5} = u_t + dt * L_y u_t + dt * f_t$$

$$u_{t+1} - dt * L_y u_{t+0.1} = u_{t+0.5} + dt * L_x u_{t+0.5} + dt * f_{t+0.5}$$

Implicit dynamics

We multiply by test functions v

$$\int (u_{t+0.5} - dt * \frac{\partial^2 u_{t+0.5}}{\partial x^2}) v = \int (u_t + dt * L_y u_t + dt * f_t) v$$

and we integrate by parts the second term

$$\int u_{t+0.5} v + dt \int u_{t+0.5} \frac{\partial u_{t+0.5}}{\partial x} \frac{\partial v}{\partial x} = \int (u_t + dt * L_y u_t + dt * f_t) v$$

We test and approximate with 2D B-splines

$$\sum_{i,j} u_{i,j}^{t+0.5} \int B_i^x(x) B_j^y(y) B_k^x(x) * B_l^y(y)$$

$$+dt * \int \frac{\partial (B_i^x(x) B_j^y(y))}{\partial x} \frac{\partial (B_k^x(x) B_l^y(y))}{\partial x} =$$

$$\int (u_t + dt * L_y u_t + dt * f_t) B_k^x(x) B_l^y(y)$$

Implicit dynamics

We have $\frac{\partial}{\partial x} B_j^y(y) = 0$ which results in

$$\frac{\partial}{\partial x}(B_i^x(x) * B_j^y(y)) = \frac{\partial}{\partial x}(B_i^x(x)) B_j^y(y) + B_i^x(x) \frac{\partial}{\partial x}(B_i^y(y))$$

and

$$\sum_{i,j} u_{i,j}^{t+0.5} \int B_i^x(x) * B_j^y(y) B_k^x(x) * B_l^y(y)$$

$$+dt * \int \frac{\partial((B_i^x(x))}{\partial x} B_j^y(y) \frac{\partial(B_k^x(x))}{\partial x} B_l^y(y) =$$

$$\int (u_t + dt * L_y u_t + dt * f_t) B_k^x(x) * B_l^y(y)$$

so our left-hand-side matrix is the Kronecker product of

$$[\int (B_i^x(x) B_k^x(x) + dt * (\frac{\partial}{\partial x} B_i^x(x))(\frac{\partial}{\partial x} B_k^x(x)))dx] *$$

$$[\int B_j^y(y) B_l^y(y) dy]$$

and it can be factorized in a linear $O(N)$ cost.

Direction splitting for linear elasticity

We solve linear elasticity problem given by

$$\begin{cases} \rho\,\partial_{tt}\mathbf{u} = \nabla\cdot\boldsymbol{\sigma} + \mathbf{F} & \text{on } \Omega\times[0,T] \\ \boldsymbol{u}(x,0) = u_0 & \text{for } x\in\Omega \\ \boldsymbol{\sigma}\cdot\widehat{\mathbf{n}} = 0 & \text{on } \partial\,\Omega \end{cases} \tag{1}$$

where $\Omega = [0,1]^3$ is an unit cube, $\mathbf{u}$ is a 3-dimensional displacement vector to be calculated, ρ is material density, $\mathbf{F}$ is the applied external force, and $\boldsymbol{\sigma}$ is the Cauchy stress tensor, given by

$$\sigma_{ij} = c_{ijkl}\epsilon_{lk}, \qquad \epsilon_{ij} = \frac{1}{2}\left(\partial_j u_i + \partial_i u_j\right) \tag{2}$$

and $\mathbf{c}$ is the elasticity tensor.

Direction splitting for linear elasticity

The above second-order system can be converted to system of 6 first-order equations by introducing additional variable $\mathbf{v} = \partial_t \mathbf{u}$:

$$\begin{cases} \partial_t \mathbf{u} = \mathbf{v} \\ \rho \partial_t \mathbf{v} = \nabla \cdot \boldsymbol{\sigma} + \mathbf{F} \end{cases} \tag{3}$$

We assume an isotropic elastic material and thus the Lame parameters,

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\epsilon} + \lambda \operatorname{tr} \boldsymbol{\epsilon}\, I \tag{4}$$

thus

$$\nabla \cdot \boldsymbol{\sigma} = 2\mu(\nabla \cdot \boldsymbol{\epsilon}) + \lambda \nabla \operatorname{tr} \boldsymbol{\epsilon} \tag{5}$$

Direction splitting for linear elasticity

Furthermore,

$$\nabla \cdot \boldsymbol{\epsilon} = \frac{1}{2} \begin{pmatrix} 2\partial_{xx} u_x & + \partial_{yy} u_x + \partial_{yx} u_y & + \partial_{zz} u_x + \partial_{zx} u_z \\ \partial_{xx} u_y + \partial_{xy} u_x & + 2\partial_{yy} u_y & + \partial_{zz} u_y + \partial_{zy} u_z \\ \partial_{xx} u_z + \partial_{xz} u_x & + \partial_{yy} u_z + \partial_{yz} u_y & + 2\partial_{zz} u_z \end{pmatrix} \tag{6}$$

and

$$\nabla \operatorname{tr} \boldsymbol{\epsilon} = \begin{pmatrix} \partial_{xx} u_x + \partial_{xy} u_y + \partial_{xz} u_z \\ \partial_{yx} u_x + \partial_{yy} u_y + \partial_{yz} u_z \\ \partial_{zx} u_x + \partial_{zy} u_y + \partial_{zz} u_z \end{pmatrix} \tag{7}$$

Direction splitting for linear elasticity

We split every timestep into three substeps

$$\begin{cases} \mathbf{u}^{(t+\frac{1}{3})} = \mathbf{u}^{(t)} + \dfrac{dt}{3}\mathbf{v}^{(t)} \\ \rho\mathbf{v}^{(t+\frac{1}{3})} = \rho\mathbf{v}^{(t)} + \dfrac{dt}{3}\left(\nabla\cdot\boldsymbol{\sigma}^{(t+\frac{1}{3})} + \mathbf{F}\right) \end{cases} \tag{8}$$

$$\begin{cases} \mathbf{u}^{(t+\frac{2}{3})} = \mathbf{u}^{(t+\frac{1}{3})} + \dfrac{dt}{3}\mathbf{v}^{(t+\frac{1}{3})} \\ \rho\mathbf{v}^{(t+\frac{2}{3})} = \rho\mathbf{v}^{(t+\frac{1}{3})} + \dfrac{dt}{3}\left(\nabla\cdot\boldsymbol{\sigma}^{(t+\frac{2}{3})} + \mathbf{F}\right) \end{cases} \tag{9}$$

Direction splitting for linear elasticity

$$\begin{cases} \mathbf{u}^{(t+1)} = \mathbf{u}^{(t+\frac{2}{3})} + \dfrac{dt}{3}\mathbf{v}^{(t+\frac{2}{3})} \\ \rho\mathbf{v}^{(t+1)} = \rho\mathbf{v}^{(t+\frac{2}{3})} + \dfrac{dt}{3}\left(\nabla\cdot\boldsymbol{\sigma}^{(t+1)} + \mathbf{F}\right) \end{cases} \tag{10}$$

and $\boldsymbol{\sigma}^{(t+\frac{k}{3})}$ are constructed using

$$\mathbf{u} = \mathbf{u}^{(t+\frac{k-1}{3})} + \frac{dt}{3}\,\mathbf{v}^{(t+\frac{k-1}{3})}$$

in most places, except when u_i appears inside i-th component of $\nabla\cdot\boldsymbol{\sigma}$ under a double derivative with respect to x ($k=1$), y ($k=2$) or z ($k=3$).

Direction splitting for linear elasticity

These cases are marked in equations (6) and (7) with red, brown, and blue color, respectively. In other words, we separate the operator into its diagonal part and the off-diagonal part, the diagonal part we treat implicitly, while the remainder we treat explicitly. After moving all the terms with values to be computed to the left-hand side, substep equations have the following form:

$$\begin{cases} \rho\, v_x^{(t+\frac{1}{3})} - \dfrac{dt}{3}(\lambda + 2\mu)\, \partial_{xx} v_x^{(t+\frac{1}{3})} & = \dots \\ \rho\, v_y^{(t+\frac{1}{3})} - \dfrac{dt}{3}\mu\, \partial_{xx} v_y^{(t+\frac{1}{3})} & = \dots \\ \rho\, v_z^{(t+\frac{1}{3})} - \dfrac{dt}{3}\mu\, \partial_{xx} v_z^{(t+\frac{1}{3})} & = \dots \end{cases} \tag{11}$$

Direction splitting for linear elasticity

$$\begin{cases} \rho\, v_x^{(t+\frac{2}{3})} - \dfrac{dt}{3}\mu\, \partial_{yy}\, v_x^{(t+\frac{2}{3})} & = \dots \\ \rho\, v_y^{(t+\frac{2}{3})} - \dfrac{dt}{3}(\lambda+2\mu)\, \partial_{yy}\, v_y^{(t+\frac{2}{3})} & = \dots \\ \rho\, v_z^{(t+\frac{2}{3})} - \dfrac{dt}{3}\mu\, \partial_{yy}\, v_z^{(t+\frac{2}{3})} & = \dots \end{cases} \tag{12}$$

$$\begin{cases} \rho\, v_x^{(t+1)} - \dfrac{dt}{3}\mu\, \partial_{zz}\, v_x^{(t+1)} & = \dots \\ \rho\, v_y^{(t+1)} - \dfrac{dt}{3}\mu\, \partial_{zz}\, v_y^{(t+1)} & = \dots \\ \rho\, v_z^{(t+1)} - \dfrac{dt}{3}(\lambda+2\mu)\, \partial_{zz}\, v_z^{(t+1)} & = \dots \end{cases} \tag{13}$$

This formulation is then transformed into a sequence of isogeometric projections in a standard way.

Example 4: Propagation of elastic waves

Click in the middle

Example 5: Pollution from a chimney with a wind

We seek the pollution concentration scalar field $c\colon \Omega \to \mathbb{R}$ such as:

$$\begin{cases} \dfrac{\partial c}{\partial t} + u \cdot \nabla c - \nabla \cdot (K \nabla c) = e & \text{on } \Omega \times [0, T] \\ \nabla c \cdot \hat{\mathbf{n}} = 0 & \text{on } \partial\Omega \times [0, T] \\ c(\mathbf{x}, 0) = c_0(\mathbf{x}) & \text{on } \Omega \end{cases} \tag{14}$$

where $\Omega = [0, 1]^3$,
$\hat{\mathbf{n}}$ is a normal vector of the domain boundary,
T is a length of the time interval for the simulation,
u is the prescribed wind,
e is the prescribed emission from the chimney,
K is the diffusion,
and c_0 is an initial state.

Example 5: Pollution from a chimney with a wind

$\Omega = 5km \times 5km \times 5km$
Mesh size $= 100 \times 100 \times 100$
Wind $= F * (cosa(t), sina(t), v(t))$ where
$a(t) = \pi/3(sin(s) + 0.5sin(2.3s)) + 3/8\pi$
$v(t) = 1/3sin(s)$
$s = t/150$
chimney $e(p) = (r-1)^2(r+1)^2$ where $r = min(1, (|p - p_0|/25)^2)$
$p_0 = (3000, 2000, 2000)$
Diffusion $K = (50, 50, 0.5)$
We run 300 time steps of the implicit method

Space instability - oscillations and reflections

Click in the middle

Stabilization

Click in the middle

Conclusions and future work

Conclusions

- Explicit dynamics with isogeometric L2 projections
- Multiple applications (e.g. heat transfer, propagation of elastic waves, tumor growth)
- Possible extension to implicit dynamics (heat transfer, propagation of elastic waves, pollution simulations)
- Limitations: tensor product geometry, Kronecker product material coefficients

Future work

- GPGPU implementation
- hybrid implementation
- Stabilized simulations of Navier-Stokes and Maxwell