Hardware-Oriented Numerics for PDEs

Solving Compressible Flow Problems by Isogeometric Analysis

Matthias Möller (m.moller@tudelft.nl)

Delft Institute of Applied Mathematics, Delft University of Technology, Netherlands

Talk in Scientific Computing Seminar at TU Kaiserslautern, November 10, 2016

Acknowledgements

Core team

J. Hinz, A. Jaeschke (Lodz), F. Khatami, R. van Nieuwpoort, D. Pouw

Collaborators

N. Gauger & M. Sagebaum (CoDiPack), B. Jüttler & A. Mantzaflaris (G+Smo), C. Sand (Armadillo), K. Iglberger (Blaze), P. Gottschling (MTL4), D. Demidov (VexCL), K. Rupp (ViennaCL), C. Strydis (EMC Rotterdam), G. Gaydadjiev (Maxeler) and many others

Financial support by EC

MOTOR

Multi-ObjecTive design Optimization of fluid eneRgy machines

Overview

1. From Numerical Analysis to Hardware-Oriented Numerics
2. Computational Building Blocks: Smart Fast Expression Templates
3. Application: Compressible flow solver
4. Isogeometric Analysis
5. Applications: Flow Problems, Meshing, and Optimization

Numerical Analysis: Past, Present, and Future(?)

Given a *problem* $p \in \mathcal{P}$:

1. Find a *method* $m \in \mathcal{M}$ that solves problem p

2. Find an *algorithm* $a \in \mathcal{A}$ that realizes method m

QoI: errors, rate of convergence, FLOP, stability, monotonicity, ...

Numerical Analysis: Past, Present, and Future(?)

Given a *problem* $p \in \mathcal{P}$:

1. Find a *method* $m \in \mathcal{M}$ that solves problem p

2. Find an *algorithm* $a \in \mathcal{A}$ that realizes method m

QoI: errors, rate of convergence, FLOP, stability, monotonicity, ...

Given a *hardware* $h \in \mathcal{H}$:

3. Find an *implementation* $i \in \mathcal{I}$ that realizes algorithm a

QoI: FLOPS, memory bandwidth, parallel speed-up, ...

Numerical Analysis: Past, Present, and Future(?)

Given a *problem* $p \in \mathcal{P}$: $-\Delta u = f$ + bc's

1. Find a *method* $m \in \mathcal{M}$ that solves problem p
   continuous Galerkin P_1-FEM
2. Find an *algorithm* $a \in \mathcal{A}$ that realizes method m
   matrix-free CG solver with element-wise Gaussian quadrature

QoI: errors, rate of convergence, FLOP, stability, monotonicity, ...

Given a *hardware* $h \in \mathcal{H}$:

3. Find an *implementation* $i \in \mathcal{I}$ that realizes algorithm a
   OpenMP parallelized SHMEM C++ code using Eigen library

QoI: FLOPS, memory bandwidth, parallel speed-up, ...

Numerical Analysis: Past, Present, and Future(?)

Given a *problem* $p \in \mathcal{P}$:

1. Find a *method* $m \in \mathcal{M}$ that solves problem p
   continuous Galerkin P_1-FEM
2. Find an *algorithm* $a \in \mathcal{A}$ that realizes method m
   matrix-free CG solver with element-wise Gaussian quadrature

Given a *hardware* $h \in \mathcal{H}$:

3. Find an *implementation* $i \in \mathcal{I}$ that realizes algorithm a
   OpenMP parallelized SHMEM C++ code using Eigen library

THE metric that matters: time to solution for prescribed accuracy

Hardware-Oriented Numerics

State of the art

Given a problem $p \in \mathcal{P}$ and a target hardware $h \in \mathcal{H}$:

1. Find an *optimal combination* $(m, a, i)_{p,h} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ that solves problem p on hardware h in shortest time with prescribed accuracy

Hardware-Oriented Numerics

State of the art

Given a problem $p \in \mathcal{P}$ and a set of target hardware $\{h_1, h_2, \dots\} \subset \mathcal{H}$:

1. Find *optimal combinations* $(m, a, i)_{p,h_k} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ that solve problem p on hardware h_k in shortest time with prescribed accuracy

Hardware-Oriented Numerics

State of the art

Given a problem $p \in \mathcal{P}$ and a set of target hardware $\{h_1, h_2, \dots\} \subset \mathcal{H}$:

1. Find *optimal combinations* $(m, a, i)_{p,h_k} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ that solve problem p on hardware h_k in shortest time with prescribed accuracy

Next step

2. Develop a strategy that automatically inspects the available hardware and chooses the optimal combinations $(m, a, i)_{p,h_k}$

Hardware-Oriented Numerics

State of the art

Given a problem $p \in \mathcal{P}$ and a set of target hardware $\{h_1, h_2, \dots\} \subset \mathcal{H}$:

1. Find *optimal combinations* $(m, a, i)_{p,h_k} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ that solve problem p on hardware h_k in shortest time with prescribed accuracy

Next step

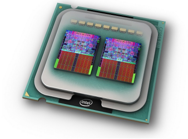
2. Develop a strategy that automatically inspects the available hardware and chooses the optimal combinations $(m, a, i)_{p,h_k}$

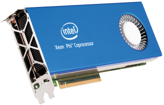
Future vision

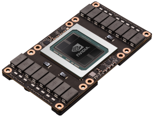
3. Automatically determine and schedule optimal combinations $(m, a, i)_{p_j,h_k} \in \mathcal{M} \times \mathcal{A} \times \mathcal{I}$ for multi-physics problems $\{p_1, p_2, \dots\} \subset \mathcal{P}$ and target hardware $\{h_1, h_2, \dots\} \subset \mathcal{H}$

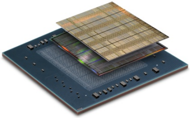
HPC hardware

Current and (most probably) future HPC hardware is diversified:





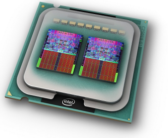




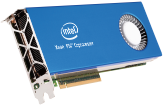
multi-core CPUs many-core MICs and GPUs FPGAs

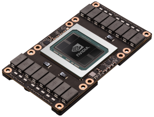
HPC hardware

Current and (most probably) future HPC hardware is diversified:

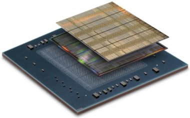


multi-core CPUs





many-core MICs and GPUs



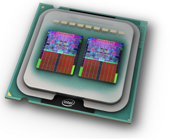
FPGAs

There are good reasons (performance-per-watt, low-latency) to believe that the future of HPC lies in heterogeneous and hybrid technologies:

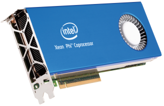
CPUs + accelerators

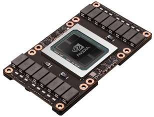
HPC hardware

Current and (most probably) future HPC hardware is diversified:

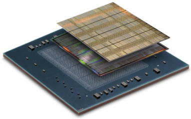


multi-core CPUs



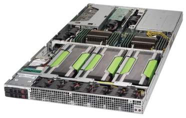


many-core MICs and GPUs



FPGAs

There are good reasons (performance-per-watt, low-latency) to believe that the future of HPC lies in heterogeneous and hybrid technologies:

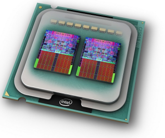


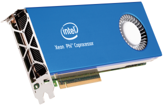
CPUs + accelerators

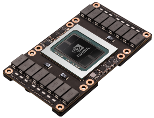
Stand-alone Xeon Phi

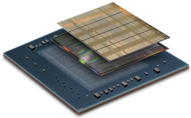
HPC hardware

Current and (most probably) future HPC hardware is diversified:



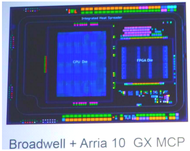






multi-core CPUs | many-core MICs and GPUs | FPGAs

There are good reasons (performance-per-watt, low-latency) to believe that the future of HPC lies in heterogeneous and hybrid technologies:



CPUs + accelerators | Stand-alone Xeon Phi | Hybrid CPU/FPGA

HPC hardware and beyond

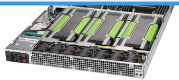
Quantum Computing

Strong effort in the Netherlands to establish quantum computers and algorithms as key technology in future scientific computing

QuSoft QUTech

Application in PDE-constrained optimization

- $\exists$ QA to estimate $x^\top Mx$ s.t. $Ax = b$ in poly$(\log N, \log(1/\epsilon))$
- best classical algorithm requires $\mathcal{O}(N\sqrt{\kappa})$



CPUs + accelerators

Stand-alone Xeon Phi

Hybrid CPU/FPGA

Question 1

Can we come up with a **unified programming approach** to exploit the performance of the different hardware architectures automatically with minimal effort for code development and maintenance?

Computational building blocks

- Highly optimized dense and sparse linear algebra libraries

$$y \leftarrow \alpha * x + y, \quad y \leftarrow A^{-1} * x$$

BLAS/LAPACK implementation of $y \leftarrow A^{-1}(x - y)$

```
call xSCAL(n, -1.0, y, 1)
call xAXPY(n,  1.0, x, 1, y, 1)
call xGESV(n, 1, A, n, IPIV, y, 1, INFO)
```

Here: 3 function calls, 5× fetching data, 3× storing data.

Ideal: no call (inlining!), 3× fetching data, 1× storing data.

And it's the memory transfer that is the bottleneck!

Computational building blocks

- Highly optimized dense and sparse linear algebra libraries

$$y \leftarrow \alpha * x + y, \quad y \leftarrow A^{-1} * x$$

- Expression template libraries (ETLs)

$$y \leftarrow A * ((m. * m)./(\rho) + p)$$

Note: against common belief, the use of ETLs does not automagically lead to high-performance C++ code

Computational building blocks

- Highly optimized dense and sparse linear algebra libraries

$$y \leftarrow \alpha * x + y, \quad y \leftarrow A^{-1} * x$$

- Expression template libraries (ETLs)

$$y \leftarrow A * ((m. * m)./(\rho) + p)$$

- *Smart and fast expression template libraries* which combine classical ETL concepts with vector intrinsics, node-level parallelization, cache-size/architecture optimized compute kernels

Computational building blocks

- Highly optimized dense and sparse linear algebra libraries

$$y \leftarrow \alpha * x + y, \quad y \leftarrow A^{-1} * x$$

- Expression template libraries (ETLs)

$$y \leftarrow A * ((m. * m)./(\rho) + p)$$

- *Smart and fast expression template libraries* which combine classical ETL concepts with vector intrinsics, node-level parallelization, cache-size/architecture optimized compute kernels
- *Just-in-time compilation* ('reconfigurable computing')

SFET concept

Code that you write[1]

```
vex::vector<float> x,y,z;   z = x * y;
```

OpenCL compute kernel generated by VexCL

```
kernel void vexcl_kernel(...) {
  for(size_t idx = get_global_id(0);
             idx < n;
             idx += get_global_size(0)) {
    prm_1[idx] = prm_2[idx] * prm_3[idx]; }}
```

[1] https://github.com/ddemidov/vexcl

SFET concept

Code that you write[1]

```
vex::vector<float> x,y,z;    z = x * y;
```

CUDA compute kernel generated by VexCL

```
extern "C" __global__ void vexcl_kernel(...) {
  for(size_t idx = blockDim.x * blockIdx.x
                 + threadIdx.x,
             grid_size = blockDim.x * gridDim.x;
             idx < n;
             idx += grid_size) {
    prm_1[idx] = prm_2[idx] * prm_3[idx]; }}
```

[1] https://github.com/ddemidov/vexcl

SFET concept

Code that you write[1]

```
vex::vector<float> x,y,z;    z = x * y;
```

MaxJ compute kernel to be generated by VexCL (D.Pouw)

```
class vexcl_kernel extends Kernel {
  public vexcl_kernel(...) {
    DFEVar x = io.input("x", dfeFloat(8, 24));
    DFEVar y = io.input("y", dfeFloat(8, 24));

    DFEVar result = x * y;
    io.output("z", result, dfeFloat(8, 24)); }}
```

[1] https://github.com/ddemidov/vexcl

Compressible Euler equations

Divergence form

$$\partial_t U + \nabla \cdot \mathbf{F}(U) = 0$$

Quasi-linear form

$$\partial_t U + \mathbf{A}(U) \cdot \nabla U = 0$$

Conservative[a] **variables**, inviscid **fluxes**, flux-Jacobian **matrices**

$$U = \begin{bmatrix} \rho \\ \rho \mathbf{v} \\ \rho E \end{bmatrix}, \qquad \mathbf{F} = \begin{bmatrix} \rho \mathbf{v} \\ \rho \mathbf{v} \otimes \mathbf{v} + \mathcal{I} p \\ \mathbf{v}(\rho E + p) \end{bmatrix}, \qquad \mathbf{A} = \frac{\partial \mathbf{F}}{\partial U}$$

Equation of state (here for an ideal gas)

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}\rho \|\mathbf{v}\|^2\right), \quad \gamma = C_p / C_v$$

[a] Similar formulations exist for primitive and entropy variables

Compressible Euler equations

Divergence form

$$\partial_t U + \nabla \cdot \mathbf{F}(U) = 0$$

Quasi-linear form

$$\partial_t U + \mathbf{A}(U) \cdot \nabla U = 0$$

Conservative[a] **variables**, inviscid **fluxes**, flux-Jacobian **matrices**

$$U = \begin{bmatrix} u_1 \\ \vdots \\ u_{d+2} \end{bmatrix}, \qquad \mathbf{F} = \begin{bmatrix} f_1^1 & \cdots & f_1^d \\ \vdots & \ddots & \vdots \\ f_{d+2}^1 & \cdots & f_{d+2}^d \end{bmatrix}, \qquad \mathbf{A} = \frac{\partial \mathbf{F}}{\partial U}$$

Notation

$$\mathbf{f}_k = \left[f_k^1, \ldots, f_k^d \right], \qquad \mathbf{f}^k = \begin{bmatrix} f_1^k \\ \vdots \\ f_{d+2}^k \end{bmatrix}$$

[a]Similar formulations exist for primitive and entropy variables

Fluid Dynamics Building Blocks[2]

High-level

SFET's for conservative/primitive variables, EOS, inviscid/viscous fluxes, flux Jacobians, and Riemann solvers

Low-level

Unified wrapper function API to core functionality of ETL's: make_temp, tag, tie, +, -, *, /, abs, sqrt, ...

Armadillo | ArrayFire | Blaze | Blitz++ | Eigen | IT++ | MTL4 | uBLAS | VexCL | ViennaCL | ...

[2] https://gitlab.com/mmoelle1/FDBB.git

FDBB at work

Implementation of $\|\mathbf{v}\|^2$

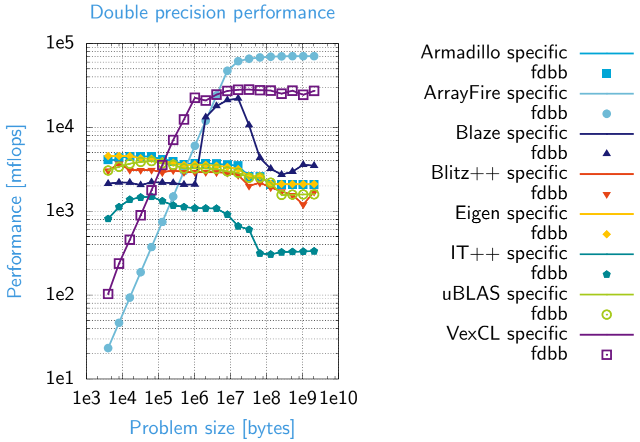
```
// EOS for ideal gas (gamma=1.4)
typedef fdbb::fdbbEOSidealGas<T> eos;

// Conservative variables in 3d
typedef fdbb::fdbbVariables<eos,3,
        fdbb::EnumVar::conservative> var;

// VexCL backend
vex::vector<T> u1,u2,u3,u4,u5,v;

// Generic implementation
v = var::v_mag2(u1,u2,u3,u4,u5);
```

FDBB μ-benchmark

- All tests were run under CentOS Linux 6.7, GCC 5.3.0, nvcc 7.5.17 with thread pinning (`likwid-pin -c N:0-15 benchmark`)
- CPU benchmarks
  - 2x Intel E5-2670 (16 cores), 2.60GHz, 20MB Cache, 64GB RAM
  - ETL's: Armadillo, Blaze, Blitz++, Eigen, IT++, uBLAS
- GPU benchmarks
  - 1x NVIDIA Tesla K20Xm, ECC off, 6GB (DriVer: 352.93)
  - ETL's: ArrayFire and VexCL with CUDA backend enabled

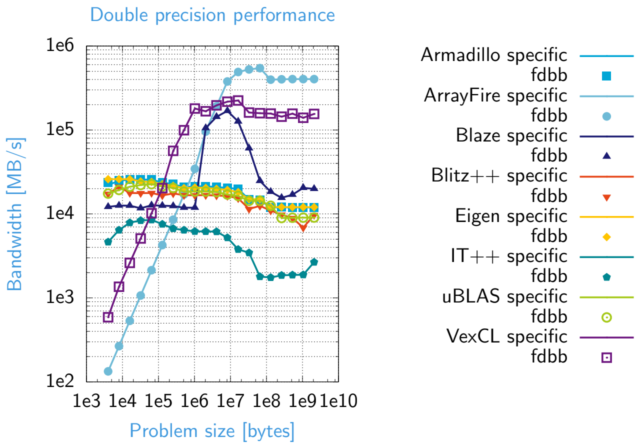
FDBB μ-benchmark

$$y \leftarrow (m_x . * m_x + m_y . * m_y + m_z . * m_z)./(\rho . * \rho) \qquad \text{7 flop}$$



FDBB μ-benchmark

$$y \leftarrow (m_x .* m_x + m_y .* m_y + m_z .* m_z)./(\rho .* \rho) \qquad \text{7 flop}$$



Question 2

Given we have a highly tuned SFET library (and FDBB), how can we design a **compressible flow solver** based on SpMV and at the same time flexible enough for practical applications?

Compressible Euler equations

Galerkin ansatz ("find solution U s.t. for all W")

$$\int_\Omega W\partial_t U - \nabla W \cdot \mathbf{F}(U)\,\mathrm{d}\Omega + \int_\Gamma W F^b(U,\cdot)\,\mathrm{d}s = 0$$

with boundary fluxes

$$F^b = \begin{cases} [0, pn_1, pn_2, pn_3, 0]^\top & \text{at solid walls} \\ \frac{1}{2}(F_n(U_-) + F_n(U_+)) - \frac{1}{2}|A_n(\mathrm{Roe}(U_-, U_+))| & \text{otherwise} \end{cases}$$

[3] C.A.J. Fletcher, CMAME 37 (1983) 225–244.

Compressible Euler equations

Galerkin ansatz ("find solution U s.t. for all W")

$$\int_\Omega W \partial_t U - \nabla W \cdot \mathbf{F}(U) \, \mathrm{d}\Omega + \int_\Gamma W F^b(U, \cdot) \, \mathrm{d}s = 0$$

with boundary fluxes

$$F^b = \begin{cases} [0, pn_1, pn_2, pn_3, 0]^\top & \text{at solid walls} \\ \frac{1}{2}(F_n(U_-) + F_n(U_+)) - \frac{1}{2}|A_n(\mathrm{Roe}(U_-, U_+))| & \text{otherwise} \end{cases}$$

Fletcher's group formulation[3]

$$U_h = \sum_A (\mathcal{I} \otimes \varphi_A(\mathbf{x})) U_A(t), \quad \mathbf{F}_h = \sum_A (\mathcal{I} \otimes \varphi_A(\mathbf{x})) \mathbf{F}_A(t), \quad \mathbf{F}_A = \mathbf{F}(U_A)$$

[3] C.A.J. Fletcher, CMAME 37 (1983) 225–244.

Compressible Euler equations

Semi-discretized problem

$$\begin{bmatrix} M & & \\ & \ddots & \\ & & M \end{bmatrix} \begin{bmatrix} \dot{u}_1 \\ \vdots \\ \dot{u}_{d+2} \end{bmatrix} + \begin{bmatrix} \mathbf{C} & & \\ & \ddots & \\ & & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^\top \\ \vdots \\ \mathbf{f}_{d+2}^\top \end{bmatrix} + \begin{bmatrix} \mathbf{S} & & \\ & \ddots & \\ & & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^{b^\top} \\ \vdots \\ \mathbf{f}_{d+2}^{b^\top} \end{bmatrix} = 0$$

Read the above as

$$\mathbf{C}\mathbf{f}_k^\top = \left[C^1, \dots, C^d\right] \begin{bmatrix} f_k^1 \\ \vdots \\ f_k^d \end{bmatrix} = \sum_{l=1}^{d} C^k f_k^l \qquad \text{for } k = 1, \dots, d+2$$

and the same for $\mathbf{S}\mathbf{f}_k^{b^\top}$

Compressible Euler equations

Semi-discretized problem

$$\begin{bmatrix} M & & \\ & \ddots & \\ & & M \end{bmatrix} \begin{bmatrix} \dot{u}_1 \\ \vdots \\ \dot{u}_{d+2} \end{bmatrix} + \begin{bmatrix} \mathbf{C} & & \\ & \ddots & \\ & & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^\top \\ \vdots \\ \mathbf{f}_{d+2}^\top \end{bmatrix} + \begin{bmatrix} \mathbf{S} & & \\ & \ddots & \\ & & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^{b^\top} \\ \vdots \\ \mathbf{f}_{d+2}^{b^\top} \end{bmatrix} = 0$$

Constant coefficient matrices

$$M = \left[\int_\Omega \varphi_A \varphi_B \, \mathrm{d}\Omega \right] \quad \mathbf{C} = \left[- \int_\Omega \nabla \varphi_A \varphi_B \, \mathrm{d}\Omega \right] \quad \mathbf{S} = \left[\int_\Gamma \varphi_A \varphi_B \mathbf{n} \, \mathrm{d}s \right]$$

Compressible Euler equations

Semi-discretized problem

$$\begin{bmatrix} M & & \\ & \ddots & \\ & & M \end{bmatrix} \begin{bmatrix} \dot{u}_1 \\ \vdots \\ \dot{u}_{d+2} \end{bmatrix} + \begin{bmatrix} \mathbf{C} & & \\ & \ddots & \\ & & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^\top \\ \vdots \\ \mathbf{f}_{d+2}^\top \end{bmatrix} + \begin{bmatrix} \mathbf{S} & & \\ & \ddots & \\ & & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^{b^\top} \\ \vdots \\ \mathbf{f}_{d+2}^{b^\top} \end{bmatrix} = 0$$

Constant coefficient matrices

$$M = \left[\int_\Omega \varphi_A \varphi_B \, \mathrm{d}\Omega \right] \quad \mathbf{C} = \left[- \int_\Omega \nabla \varphi_A \varphi_B \, \mathrm{d}\Omega \right] \quad \mathbf{S} = \left[\int_\Gamma \varphi_A \varphi_B \mathbf{n} \, \mathrm{d}s \right]$$

whereby

$$- \int_\Omega \nabla \varphi_A \varphi_B \, \mathrm{d}\Omega = \int_\Omega \varphi_A \nabla \varphi_B \, \mathrm{d}\Omega + \int_\Gamma \varphi_A \varphi_B \mathbf{n} \, \mathrm{d}s \quad \Rightarrow \quad \mathbf{C} + \mathbf{C}^\top = \mathbf{S}$$

Stabilization by algebraic flux correction

$$(\mathcal{I} \otimes m_A)\dot{U}_A + \sum_B \left(\mathbf{c}_{AB} \cdot \mathbf{F}_B + \mathbf{s}_{AB} \cdot \mathbf{F}_B^b\right) + \sum_{B \in \mathcal{J}_A} D_{AB}(U_B - U_A) = \sum_{B \in \mathcal{J}_A} \alpha_{AB} \mathcal{F}_{AB}$$

1. Perform row-sum mass lumping to decouple the degrees of freedom
2. Add discrete artificial dissipation to prevent spurious oscillations
3. Decompose anti-diffusion into fluxes and apply a limited correction

Details:

- Kuzmin, M., Gurris, AFC II. Compressible Flow Problems. In: Flux-Corrected Transport, Springer, 2012

Stabilization by algebraic flux correction

$$(\mathcal{I} \otimes m_A)\dot{U}_A + \sum_B \left(\mathbf{c}_{AB} \cdot \mathbf{F}_B + \mathbf{s}_{AB} \cdot \mathbf{F}_B^b\right) + \sum_{B \in \mathcal{J}_A} D_{AB}(U_B - U_A) = \sum_{B \in \mathcal{J}_A} \alpha_{AB} \mathcal{F}_{AB}$$

Compute kernels

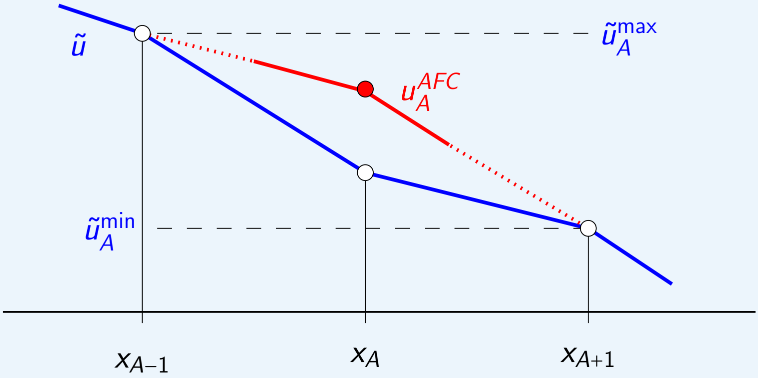
- block-VV and block-SpMV
- edge-loops over non-zero entries of sparsity graph

$$\mathcal{I}_A := \{B \,:\, \mathrm{supp}\varphi_A \cap \mathrm{supp}\varphi_B \neq \emptyset\}, \quad \mathcal{J}_A := \mathcal{I}_A \setminus \{A\}$$

  - symmetric operators D_{AB} and α_{AB}
  - skew-symmetric fluxes $U_B - U_A$ and $\mathcal{F}_{AB}$

$\Rightarrow$ can be expressed as block-SpMV

Illustration of Zalesak's flux limiter[4]



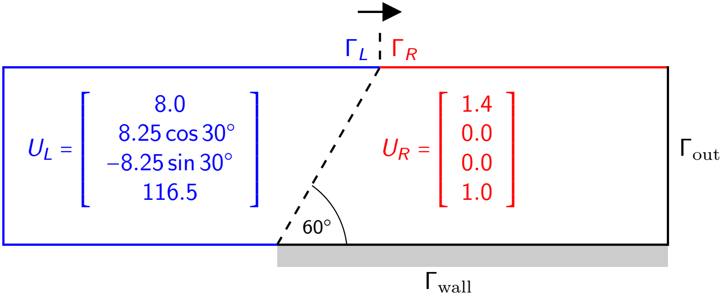
- Mass-lumped low-order predictor yields nodal bounds $\tilde{u}_A^{\overset{max}{min}}$
- AFC-corrected solution is allowed to vary within the bounds

[4] S. Zalesak, JCP 1979, 31(3), 335–362

Double Mach reflection[5]

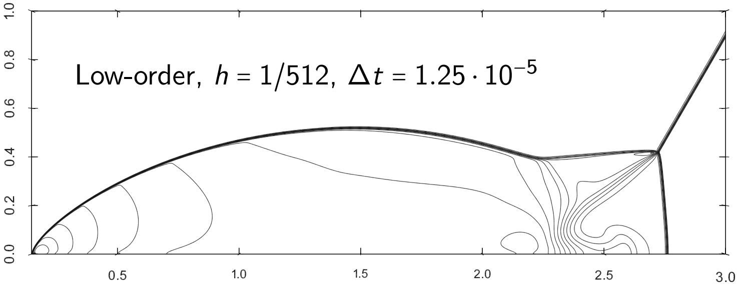
Test: Roe-linearization + FCT, structured mesh, Q_1 finite elements

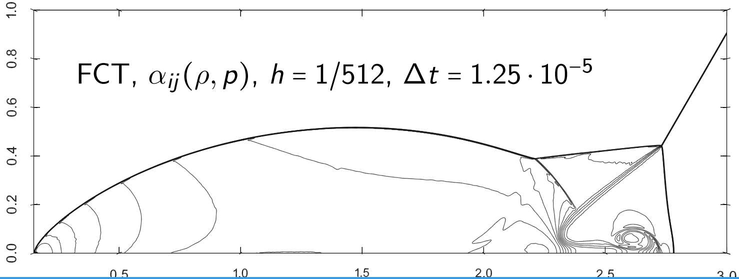
$T = 0.2$, Crank Nicolson time stepping ($\theta = 0.5$)



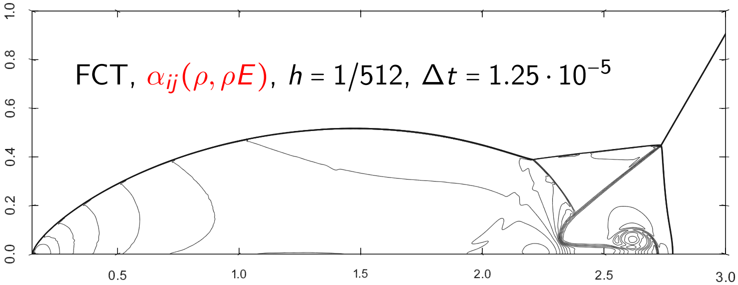
[5] P.R. Woodward, P. Colella, JCP 54, 115 (1984), 115–173.

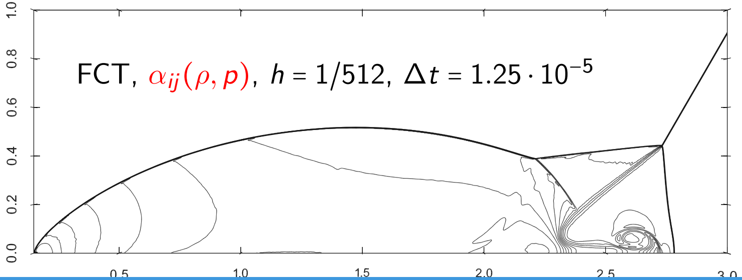
Double Mach reflection





Double Mach reflection





Question 3

The presented approach is applicable to unstructured meshes and general FE spaces except for AFC which is limited to P_1 and Q_1!

It there a way to extend AFC to **higher-order approximations**?

Polynomial spaces

Definition

The space of polynomials of degree p over the interval $[a,b]$ is

$$\Pi^p([a,b]) := \{q(x) \in \mathcal{C}^\infty([a,b]) \,:\, q(x) = \sum_{i=0}^{p} c_i x^i,\, c_i \in \mathbb{R}\}$$

Example: $\Pi^2([0,1])$

- Canonical basis

$$\mathcal{B} = \{1, x, x^2\}$$

- Polynomials

$$q(x) = c_0 + c_1 x + c_2 x^2$$

Spline space

Definition

Let $\mathcal{P} = \{a = x_1 < \cdots < x_{p+1} = b\}$ be a partition of the interval Ω_0 and $\mathcal{M} = \{1 \leq m_i \leq p+1\}$ a set of positive integers. The polynomial spline of degree p is defined as $s : \Omega_0 \mapsto \mathbb{R}$ if

$$s\big|_{[x_i, x_{i+1}]} \in \Pi^p([x_i, x_{i+1}]), \qquad i = 1, \ldots, k$$

$$\frac{d^j}{dx^j} s_{i-1}(x_i) = \frac{d^j}{dx^j} s_i(x_i), \qquad \begin{array}{l} i = 2, \ldots, k, \\ j = 0, \ldots, p - m_i \end{array}$$

Polynomial splines of degree p form the spline space $\mathcal{S}(\Omega_0, p, \mathcal{M}, \mathcal{P})$.

Knot vectors

Definition

A knot vector is a sequence of non-decreasing values $\xi_i \in [a, b] \subset \mathbb{R}$ in the parameter space $\Omega_0 = [a, b]$

$$\Xi = (\xi_1, \xi_2, \ldots, \xi_{n+p+1})$$

where

- p is the polynomial order of the B-splines
- n is the number of B-spline functions
- ξ_i is the i-th knot with knot index i

Knots ξ_i can have multiplicity $1 \leq m_i \leq p + 1$. The knot vector is called open if the first and last knot have multiplicity $p + 1$.

B-spline basis functions

Cox-de Boor recursion formula

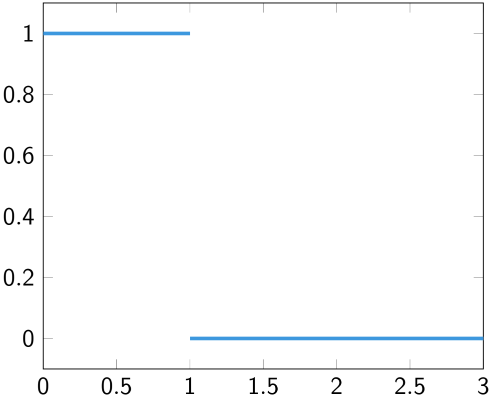
$p = 0$

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

$p > 0$

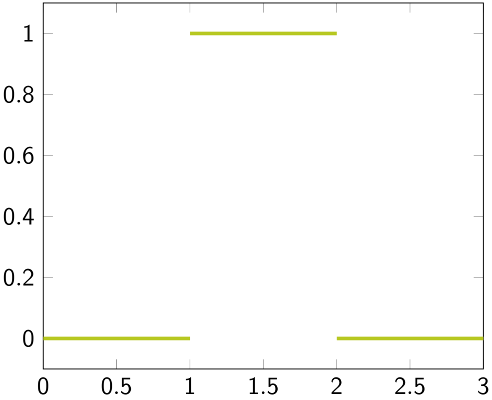
$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)$$

B-spline basis functions



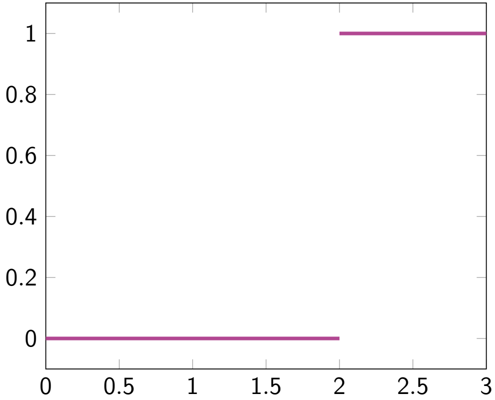
Constant basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



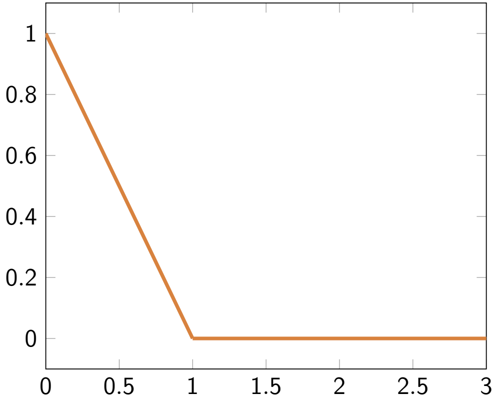
Constant basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



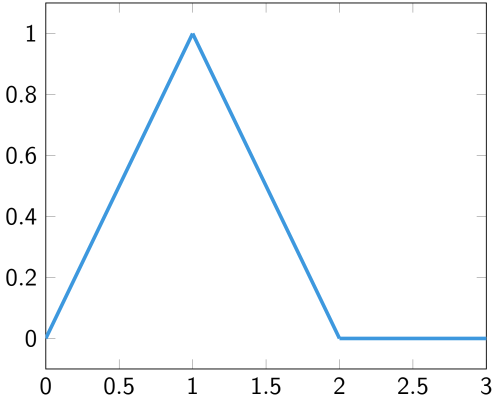
Constant basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



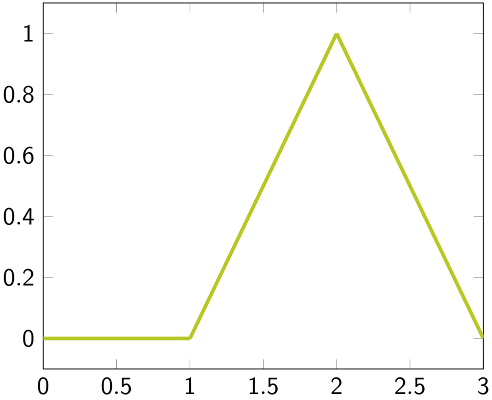
Linear basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



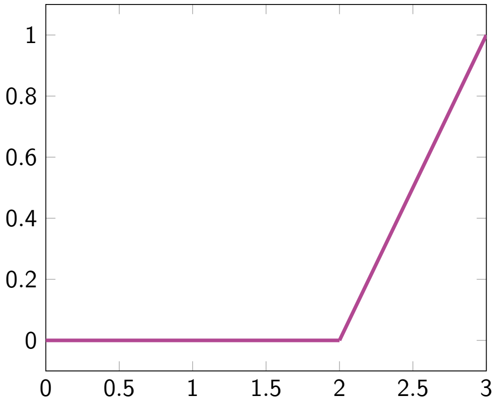
Linear basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



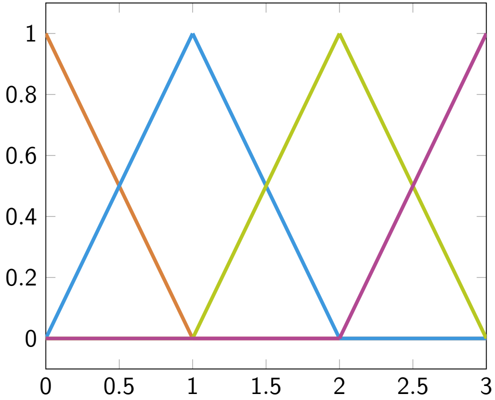
Linear basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



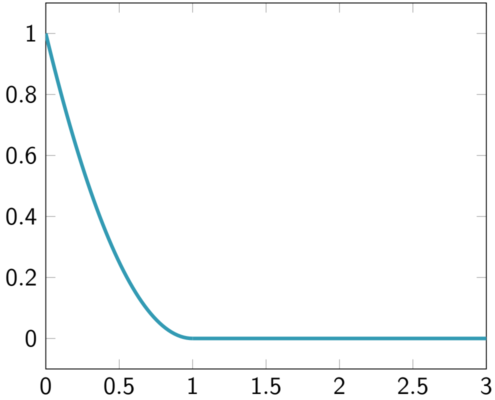
Linear basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



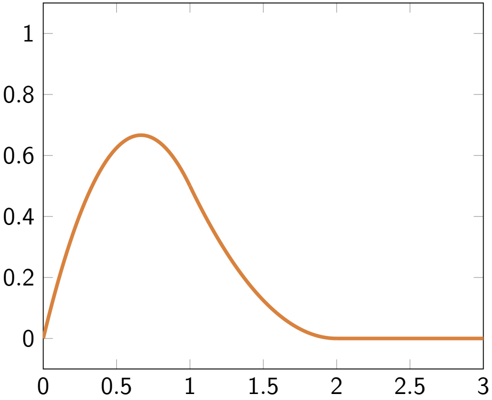
Linear basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



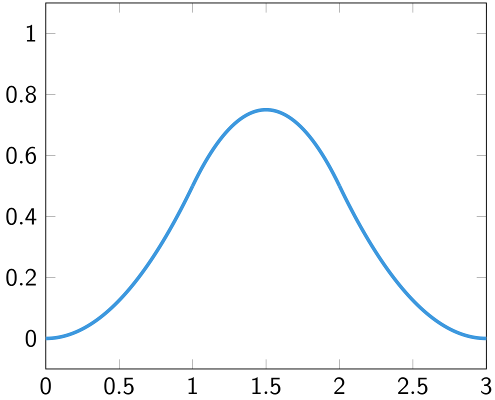
Quadratic basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



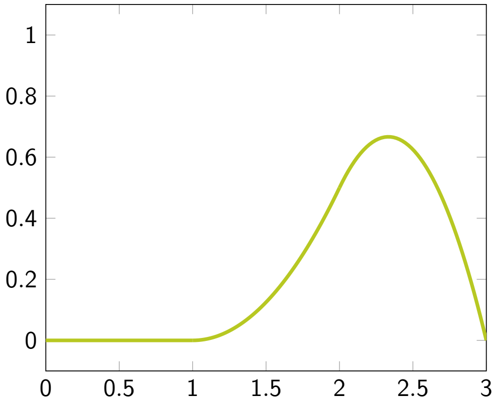
Quadratic basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



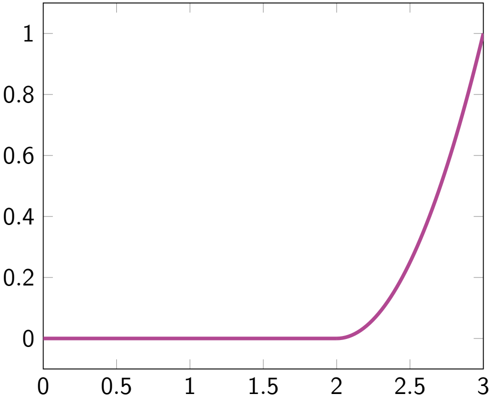
Quadratic basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



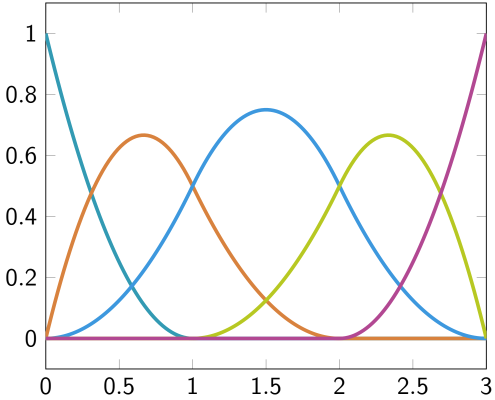
Quadratic basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



Quadratic basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

B-spline basis functions



Quadratic basis functions corresponding to $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$

Properties of B-spline basis functions

Compact support

$$\text{supp } N_{i,p}(\xi) = [\xi_i, \xi_{i+p+1}), \quad i = 1, \dots, n$$

- System matrices are sparse like in the standard FEM
- Support grows with the polynomial order so that system matrices have a slightly broader stencil due to the coupling of degrees of freedom over multiple element layers (good for HPC)

Properties of B-spline basis functions

Compact support

$$\text{supp } N_{i,p}(\xi) = [\xi_i, \xi_{i+p+1}), \quad i = 1, \ldots, n$$

Strict positiveness

$$N_{i,p}(\xi) > 0 \quad \text{for } \xi \in (\xi_i, \xi_{i+p+1}), \quad i = 1, \ldots, n$$

- Consistent mass matrix has no negative off-diagonal entries
- Lumped mass matrix is not singular (no zero diagonal entries)

Properties of B-spline basis functions

Compact support

$$\text{supp } N_{i,p}(\xi) = [\xi_i, \xi_{i+p+1}), \quad i = 1, \ldots, n$$

Strict positiveness

$$N_{i,p}(\xi) > 0 \quad \text{for } \xi \in (\xi_i, \xi_{i+p+1}), \quad i = 1, \ldots, n$$

Partition of unity

$$\sum_{i=1}^{n} N_{i,p}(\xi) = 1 \quad \text{for all } \xi \in [a, b]$$

Properties of B-spline basis functions

Derivatives

Derivative is a B-spline of order $p-1$

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

Expression for k^{th} derivative

$$\frac{d^k}{d^k\xi}N_{i,p}(\xi) = \frac{p!}{(p-k)!}\sum_{j=0}^{k}\alpha_{k,j}N_{i+j,p-k}(\xi)$$

with recursively defined coefficients $\alpha_{k,j}$[a]

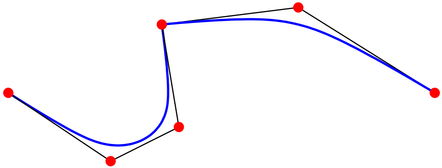
[a] L. Piegl, W. Tiller. The NURBS book (1997).

Spline curves

Geometric mapping $\mathbf{G} : \Omega_0 \mapsto \Omega_h \simeq \Omega$

$$\mathbf{G}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{B}_i$$

set of control points $\mathbf{B}_i \in \mathbb{R}^d, d \geq 1$

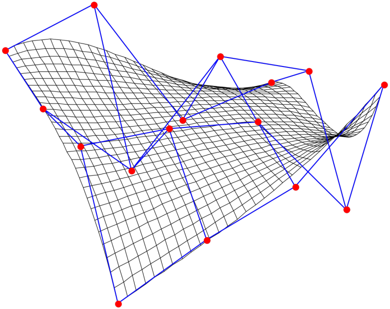


- C^{p-m_i} continuous curve (m_i is the multiplicity of knot ξ_i)
- Convex hull property
- Variation diminishing property
- Knot insertion (h-adaptivity), order elevation (p-adaptivity) preserve shape of geometry

Spline surfaces

Geometric mapping $\mathbf{G}: \Omega_0 \mapsto \Omega_h \simeq \Omega$

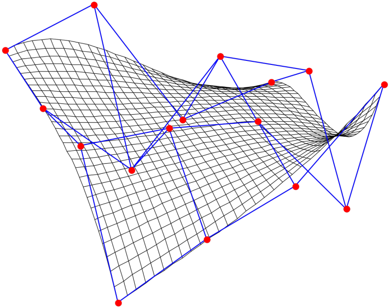
$$\mathbf{G}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\xi)N_{j,q}(\eta)\mathbf{B}_{i,j} \qquad \mathbf{B}_{i,j} \in \mathbb{R}^d, d \geq 2$$



Spline surfaces

Geometric mapping $\mathbf{G} : \Omega_0 \mapsto \Omega_h \simeq \Omega$

$$\mathbf{G}(\xi) = \sum_{\mathbf{A}} \hat{\varphi}_{\mathbf{A}}(\xi)\mathbf{B}_{\mathbf{A}} \qquad \mathbf{B}_{\mathbf{A}} \in \mathbb{R}^d, d \geq 2, \text{ multi-index } \mathbf{A}$$



- Computational 'mesh' is a multi-variate parameterization of Ω_h. It can be canonically generated from the geometry by knot insertion and/or order elevation $(\hat{\varphi}_{\mathbf{A}}, \mathbf{B}_{\mathbf{A}}) \to (\tilde{\varphi}_{\mathbf{A}}, \tilde{\mathbf{B}}_{\mathbf{A}})$

Marriage of geometry and discretization

Geometric mapping

$$\mathbf{G}(\boldsymbol{\xi}) = \sum_{\mathbf{A}} \hat{\varphi}_{\mathbf{A}}(\boldsymbol{\xi}) \mathbf{B}_{\mathbf{A}} \qquad \text{'push-forward'} \ \mathbf{G} : \Omega_0 \mapsto \Omega_h$$

Ansatz space

$$V_h = \text{span}\{\varphi_{\mathbf{A}}(\mathbf{x}) = \tilde{\varphi}_{\mathbf{A}} \circ \mathbf{G}^{-1}(\mathbf{x})\} \qquad \text{'pull-back'} \ \mathbf{G}^{-1} : \Omega_h \mapsto \Omega_0$$

Question 4

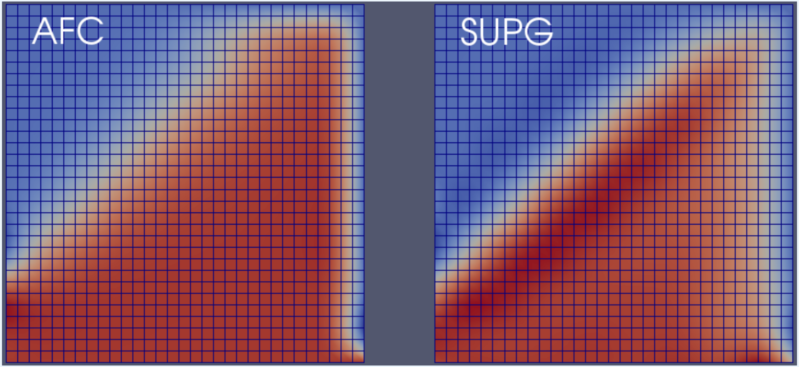
Bézier extraction is commonly promoted as 'the' way to integrate isogeometric analysis into classical finite element codes. But doesn't this contradict the concept of hardware-oriented numerics?

Our research is based on genuine IgA tools:

- C++ library G+SMO developed at JKU/RICAM, Linz
- Python library Nutils (Numerical Utilities) by Evalf Computing, Delft

Application: Convection-diffusion equation

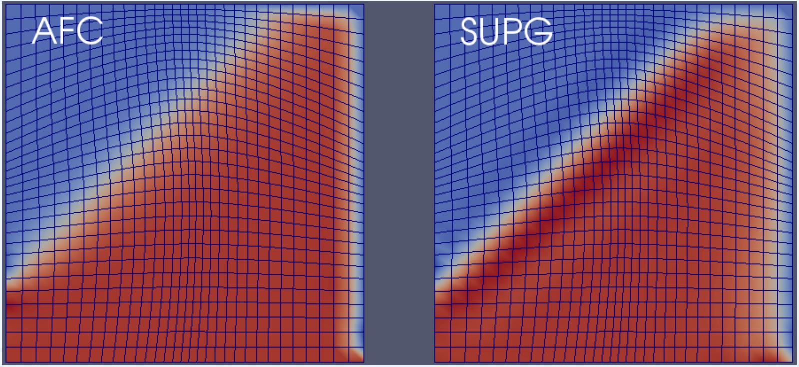
Convection skew to the mesh



Quadratic bi-variate B-spline basis functions.

Application: Convection-diffusion equation

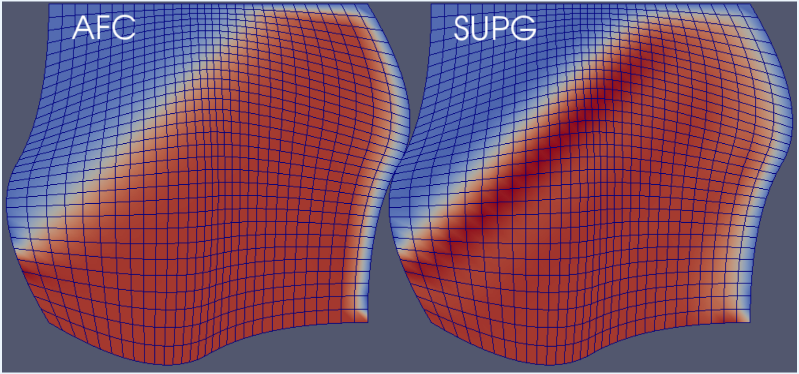
Convection skew to the mesh



Quadratic bi-variate B-spline basis functions.

Application: Convection-diffusion equation

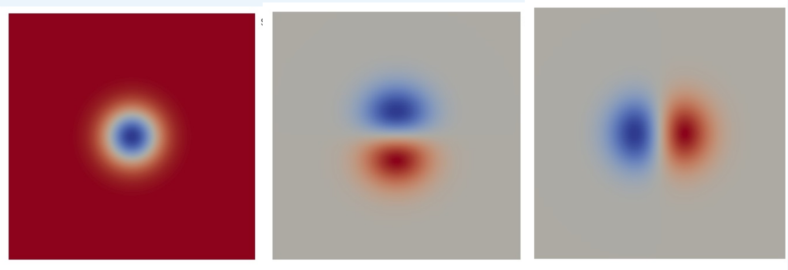
Convection skew to the mesh



Quadratic bi-variate B-spline basis functions.

Application: Compressible Euler equations

Convection of isentropic vortex[6]



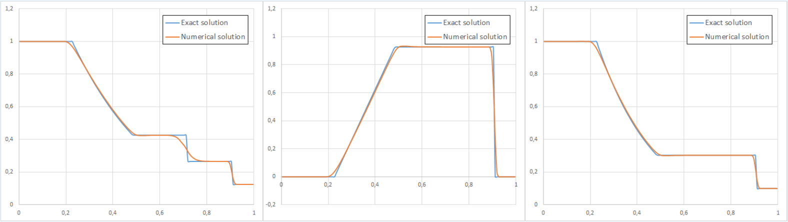
ρ $\qquad$ v_x $\qquad$ v_y

Quadratic bi-variate B-spline basis functions.

[6] H-C. Yee, N. Sandham, M. Djomehri, JCP 150 (1999) 199-238.

Application: Compressible Euler equations

Sod's shock tube problem[7]



ρ v_x p

Quadratic bi-variate B-spline basis functions.

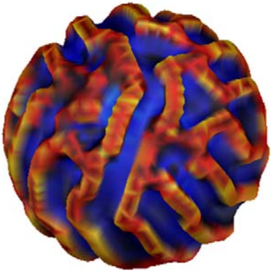
[7] G.A. Sod, JCP 27 (1978) 1–31.

Application: IgA on evolving manifolds

Gray-Scott reaction-diffusion model

$$u_t + u(\ln\sqrt{g_t})_t - d_1\Delta u = F(1-u) - uv^2$$
$$v_t + v(\ln\sqrt{g_t})_t - d_2\Delta v = -(F+H)v + uv^2$$
$$\mathbf{s} = Kv\mathbf{n}$$

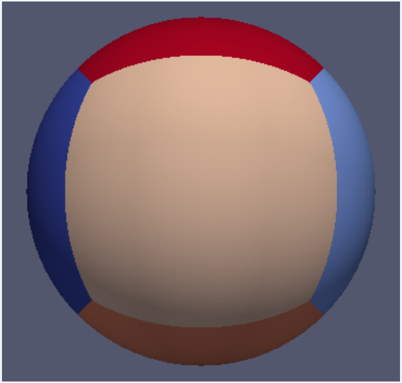
MSc-thesis project by J. Hinz



J. Lefèvre, J-F. Mangin, PLoS Comput. Biol. 6(4) e1000749.

Application: IgA on evolving manifolds

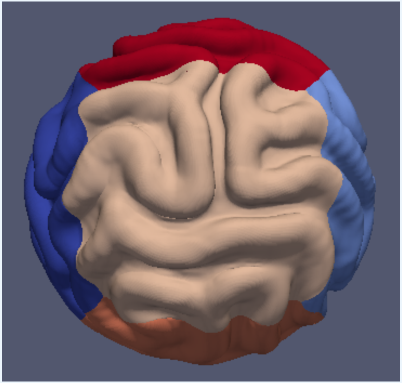
Phenomenological human brain development model



- multi-patch geometry $\Omega_h \simeq \Omega$ approximated by quadratic (hierarchical) B-spline basis functions
- C^{p-1} continuity along patch boundaries due to periodic basis functions
- C^0 continuity in the vicinity of the triple points

Application: IgA on evolving manifolds

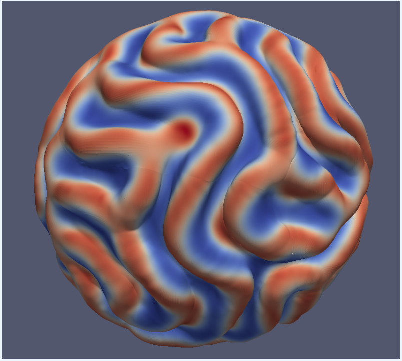
Phenomenological human brain development model

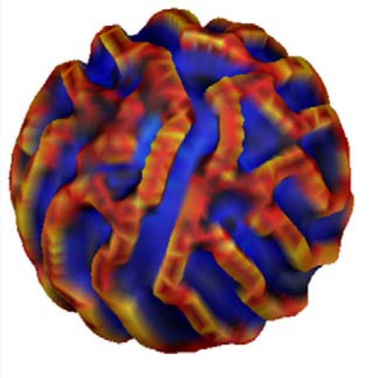


- multi-patch geometry $\Omega_h \simeq \Omega$ approximated by quadratic (hierarchical) B-spline basis functions
- C^{p-1} continuity along patch boundaries due to periodic basis functions
- C^0 continuity in the vicinity of the triple points

Application: IgA on evolving manifolds

Phenomenological human brain development model





Application: Isogeometric 'mesh generation'[8]

Create a valid mapping (= diffeomorphism, e.g., $\det J > 0$ on Ω_0)

$$\mathbf{G} : \Omega_0 \mapsto \Omega_h \simeq \Omega$$

starting from the boundary parameterization $\bigcup_i \gamma_i$ of Ω by solving

$$\begin{cases} \Delta x(\xi,\eta) & = 0 \\ \Delta y(\xi,\eta) & = 0 \end{cases} \quad \text{s.t.} \quad \mathbf{S}|_{\partial\Omega_i} = \gamma_i.$$

Theory: Ω_h must be convex for $\mathbf{G}$ to be a diffeomorphism.

[8] PhD project by J. Hinz

Application: Isogeometric 'mesh generation'[8]

Create a valid mapping (= diffeomorphism, e.g., $\det J > 0$ on Ω_0)

$$\mathbf{G} : \Omega_0 \mapsto \Omega_h \simeq \Omega$$

starting from the boundary parameterization $\bigcup_i \gamma_i$ of Ω by solving

$$\begin{cases} \Delta\xi(x,y) &= 0 \\ \Delta\eta(x,y) &= 0 \end{cases} \qquad \text{s.t.} \quad \mathbf{S}^{-1}\big|_{\gamma_i} = \partial\Omega_i$$

for the inverse mapping $\mathbf{G}^{-1} : \Omega_h \mapsto \Omega_0$. Inversion yields

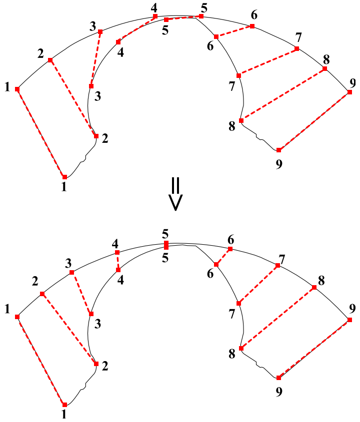
$$\begin{cases} g_{22}x_{\xi\xi} - 2g_{12}x_{\xi\eta} + g_{11}x_{\eta\eta} &= 0 \\ g_{22}y_{\xi\xi} - 2g_{12}y_{\xi\eta} + g_{11}y_{\eta\eta} &= 0 \end{cases} \qquad \text{s.t.} \quad \mathbf{G}\big|_{\partial\Omega_i} = \gamma_i,$$

where $g_{11} = x_\xi^2 + y_\xi^2$, $g_{12} = x_\xi x_\eta + y_\xi y_\eta$ and $g_{22} = x_\eta^2 + y_\eta^2$.

[8] PhD project by J. Hinz

Application: Isogeometric 'mesh generation'[9]

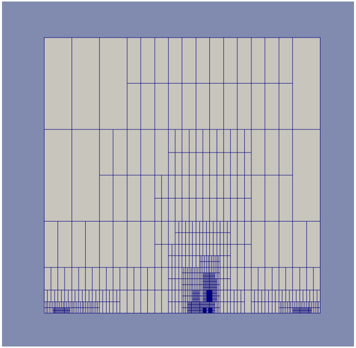
1. Boundary reparameterization



[9] PhD project by J. Hinz

Application: Isogeometric 'mesh generation'[9]

1. Boundary reparameterization
2. Defect detection, e.g., where $\det J(\boldsymbol{\xi}^*) < 0$ or using the dual-weighted residual approach by Becker and Rannacher and refine the parameterization locally (THB-splines by Giannelli *et al.*)



[9] PhD project by J. Hinz

Application: Isogeometric 'mesh generation'[9]

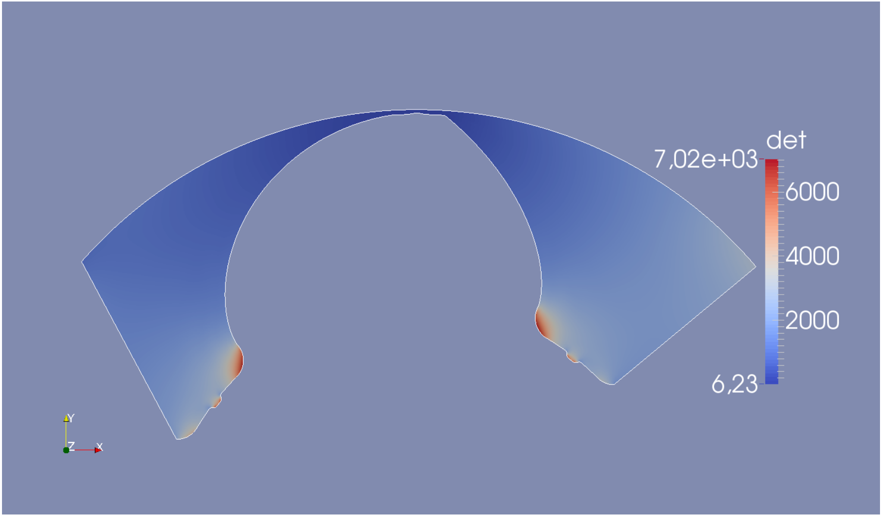
1. Boundary reparameterization

2. Defect detection, e.g., where $\det J(\boldsymbol{\xi}^*) < 0$ or using the dual-weighted residual approach by Becker and Rannacher and refine the parameterization locally (THB-splines by Giannelli *et al.*)

3. Possible extensions:
   - optimization of 'mesh properties'
   - multi-patch segmentation
   - 4th order PDE-problem

[9] PhD project by J. Hinz

Application: Isogeometric 'mesh generation'[9]



[9] PhD project by J. Hinz

Application: Adjoint-based optimization[10]

Proof-of-concept: AD of G+Smo using CoDiPack

$$-\Delta u + \nabla \cdot (\mathbf{v} u) = f \quad \text{in } \Omega_h, \quad u \equiv 1 \quad \text{on } \partial\Omega_h$$

with exact solution $u \equiv 1$.

Goal: Maximize area $A = \|u_h\|_{L^2(\Omega_h)}$ of geometry Ω_h while preserving the circumference $C = \|u_h\|_{L^2(\Gamma_h)}$ of the initial geometry $\Omega_0 = [0,1]^2$.

Gradient based optimization using `IpOpt` with cost functional

$$L = -A + \eta |C_0 - C|$$

[10] PhD project by A. Jaeschke (Lodz)

Conclusion and outlook

1. Open-source Fluid Dynamic Building Blocks library
   `https://gitlab.com/mmoelle1/FDBB.git`
2. IgA-based solver for compressible flows
3. Isogeometric 'mesh generation'
4. Proof-of-concept AD of G+Smo code

Ongoing and future work:

- Distributed JIT compilation of multi-patch geometries
- Embedding of linear algebra SFETs into CoDiPack
- Extension towards FPGAs (reconfigurable computing)

Appendix

Further applications of the AFC framework

Idealized Z-pinch implosion model[11]

- Generalized Euler system coupled with scalar tracer equation

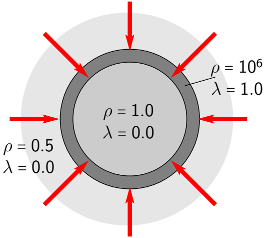
$$\frac{\partial}{\partial t}\begin{bmatrix}\rho \\ \rho\mathbf{v} \\ \rho E \\ \rho\lambda\end{bmatrix} + \nabla\cdot\begin{bmatrix}\rho\mathbf{v} \\ \rho\mathbf{v}\otimes\mathbf{v} + p\mathcal{I} \\ \rho E\mathbf{v} + p\mathbf{v} \\ \rho\lambda\mathbf{v}\end{bmatrix} = \begin{bmatrix}0 \\ \mathbf{f} \\ \mathbf{f}\cdot\mathbf{v} \\ 0\end{bmatrix}$$

- Equation of state

$$p = (\gamma - 1)\rho(E - 0.5|\mathbf{v}|^2)$$

- Non-dimensional Lorentz force

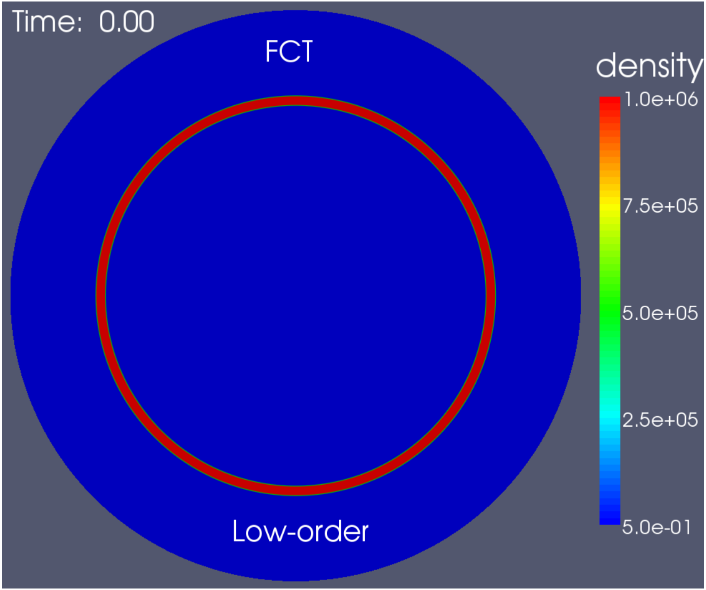
$$\mathbf{f} = (\rho\lambda)\left(\frac{I(t)}{I_{\max}}\right)^2 \frac{\hat{\mathbf{e}}_r}{r_{\text{eff}}}, \quad 0 \le \lambda \le 1$$



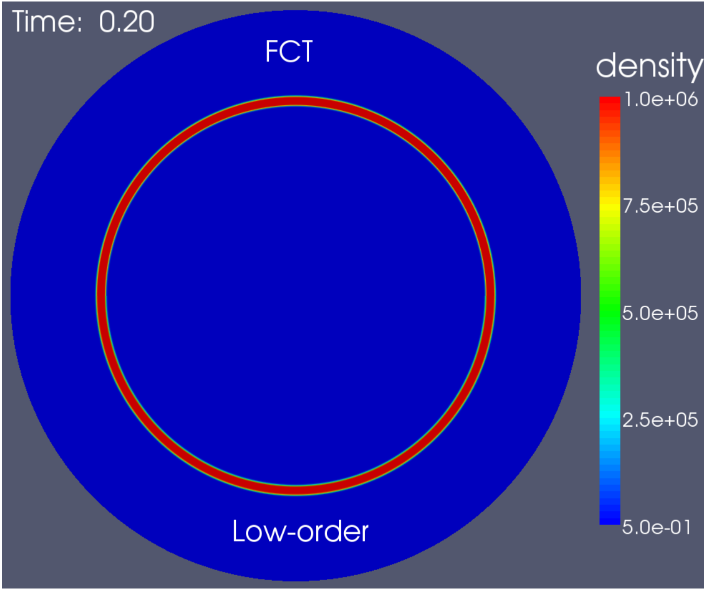
$\mathbf{v} = 0.0$, $p = 1.0$ everywhere

[11] J.W. Banks, J.N. Shadid, IJNMF 2009, 61(7), 725–751

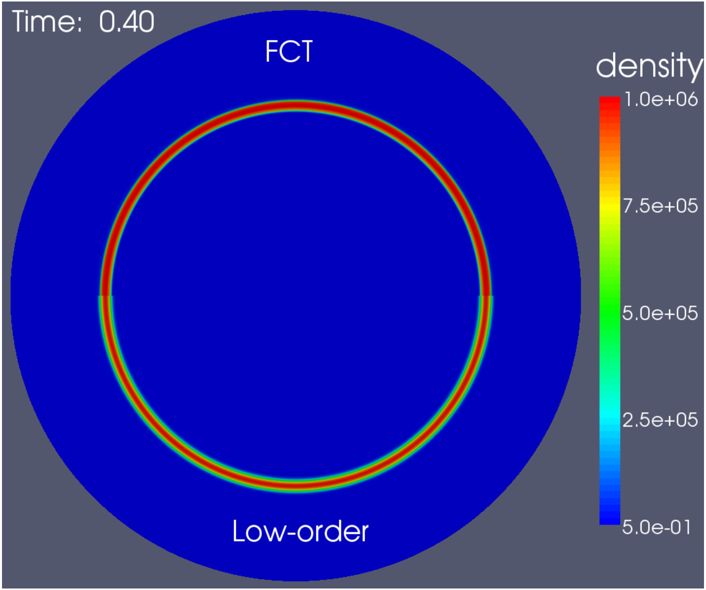
Idealized Z-pinch implosion



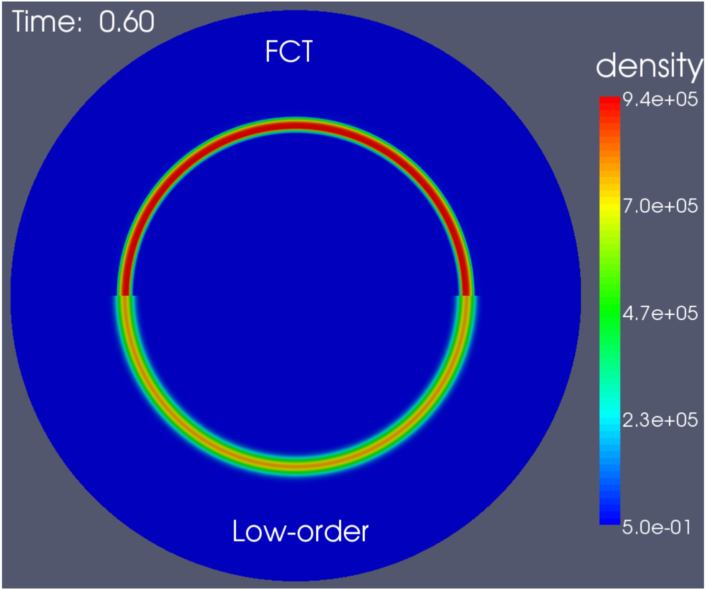
Idealized Z-pinch implosion



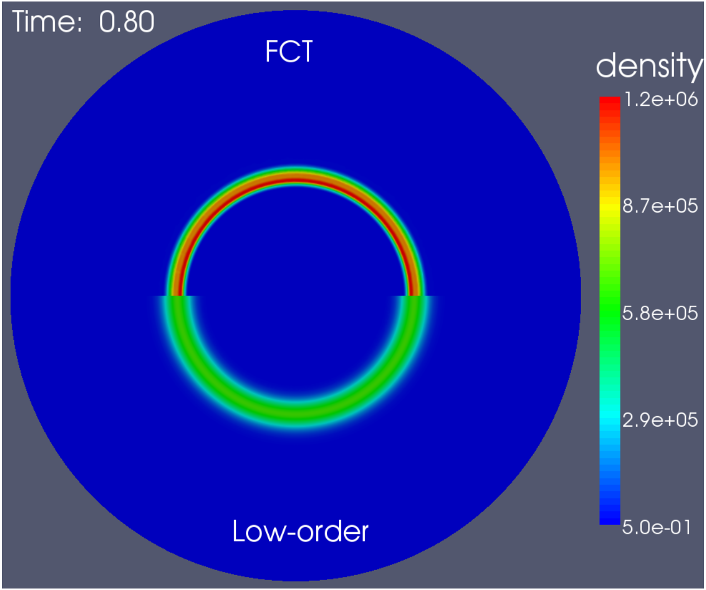
Idealized Z-pinch implosion



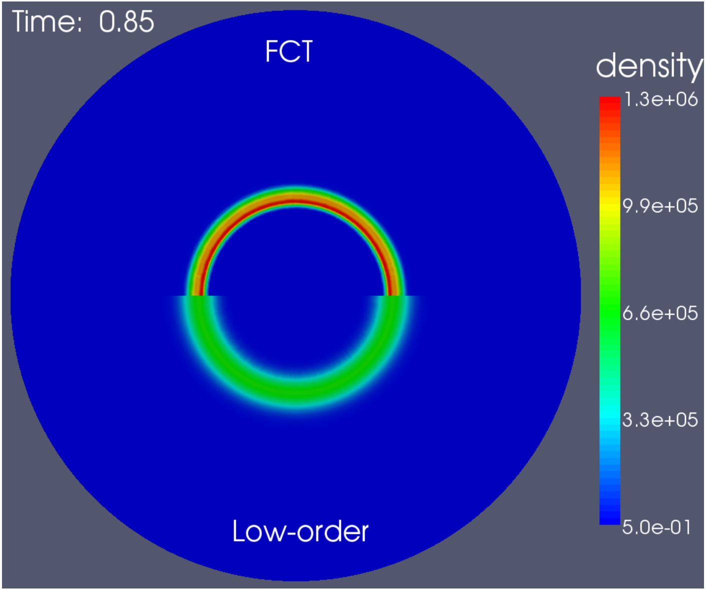
Idealized Z-pinch implosion



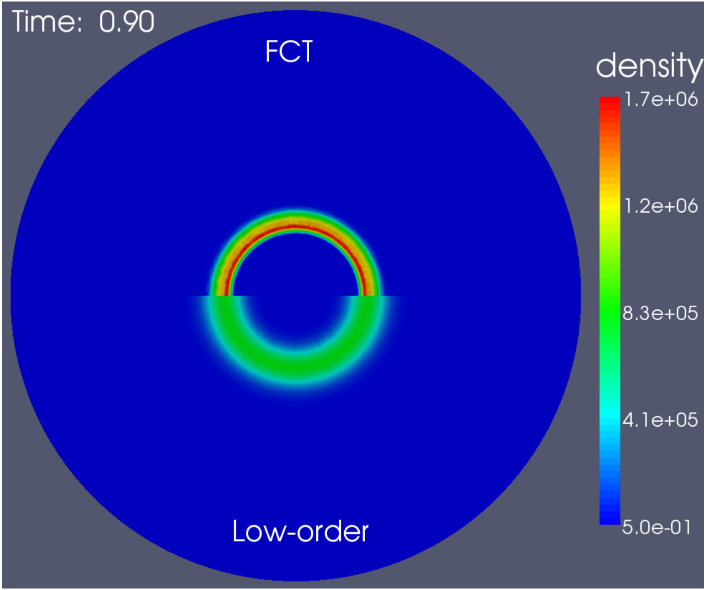
Idealized Z-pinch implosion



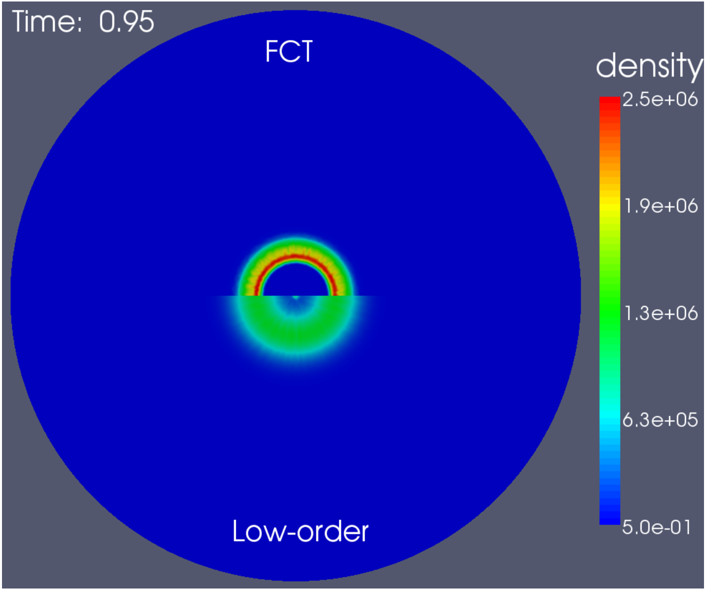
Idealized Z-pinch implosion



Idealized Z-pinch implosion



Idealized Z-pinch implosion



Idealized Z-pinch implosion