Isogeometric Analysis for Compressible Flow Problems in Industrial Applications

Matthias Möller

Delft Institute of Applied Mathematics, Numerical Analysis
Delft University of Technology

I³MS Seminar, 13 November 2017, CATS@RWTH Aachen
Numerical Analysis group: Prof.dr.ir. Kees Vuik, Kees Oosterlee
- 12 assistant/associated professors, 30 PhDs, 5 guest researchers
- Industrial flows, FSI, biomath, finance, iterative solvers, HPC, ...

My research team:
- J. Hinz: *IgA-based elliptic grid generation for industrial applications*
- R. Tielen: *IgA-inspired high-order material point method*
- J. v.d. Meer: *foam enhanced oil recovery*
- A. Jaeschke (TU Lodz, PL): *IgA in turbomachinery applications*

My own research interests:
- High-order high-resolution FEM/IgA schemes and efficient solvers for compressible flow problems, hardware-oriented numerics on unconventional hardware, quantum computing
Overview

1 Introduction
   Scientific computing from a hardware perspective
   Isogeometric design-simulation-optimization loop

2 Elliptic 'grid' generation [J. Hinz]
   Planar parameterizations
   Volumetric parameterizations

3 Compressible flow solver [A. Jaeschke]
   Problem formulation
   Linearized FCT limiter
   Numerical examples

4 Implementation aspects
   Meta-programming techniques for heterogeneous HPC
Today: Heterogeneous compute hardware at system and node level

- Distributed cluster systems with heterogeneous compute nodes
  - Multi-core CPUs: Intel 28, AMD 32, ARM 48, IBM 16x12
  - Many-core accelerators: GPUs, Intel MICs, FPGAs, ...
  - Memory subsystem is the bottleneck in data-intensive applications
Scientific computing from a hardware perspective

Today: Heterogeneous compute hardware at system and node level
- Distributed cluster systems with heterogeneous compute nodes
  - Multi-core CPUs: Intel 28, AMD 32, ARM 48, IBM 16x12
  - Many-core accelerators: GPUs, Intel MICs, FPGAs, ...
- Memory subsystem is the bottleneck in data-intensive applications

Future trends: 'novel' hardware architectures and paradigm shifts
- In-memory-computing for data-intensive applications (DBs, FEM?)
- Data-flow computing in space vs. control-flow computing in time
- Special-purpose accelerators: analogue or quantum computers
Scientific computing from a hardware perspective

**Today:** Heterogeneous compute hardware at system and node level
- Distributed cluster systems with heterogeneous compute nodes
  - Multi-core CPUs: Intel 28, AMD 32, ARM 48, IBM 16×12
  - Many-core accelerators: GPUs, Intel MICs, FPGAs, ...
  - Memory subsystem is the bottleneck in data-intensive applications

**Future trends:** 'novel' hardware architectures and paradigm shifts
- In-memory-computing for data-intensive applications (DBs, FEM?)
- Data-flow computing in space vs. control-flow computing in time
- Special-purpose accelerators: analogue or quantum computers

**Philosophy:** Bottom-up design of hardware-oriented numerics *for* next-gen hardware instead of squeezing existing methods *into* it.
An example of data-flow computing in space

Example: 2d Poisson equation

- IgA with tensor-product B-spline basis functions of order $p = 2$.
- Matrix-free iterative CG/BiCGStab solver

\[ \int_{[0,1]^2} g_{\alpha,\beta}(\xi) \partial_\beta \hat{B}_j(\xi) \left( \partial_\alpha \hat{B}_i(\xi) \right) \partial \xi \approx \sum_{\alpha,\beta=1}^{d} Q_{\alpha,\beta,i} \left( g_{\alpha,\beta}(\cdot) \partial_\beta \hat{B}_j(\cdot) \partial \right) \]

\(^1\)CMAME 316, (2017) 606-622.
Example: 2d Poisson equation

- IgA with tensor-product B-spline basis functions of order $p = 2$.
- Matrix-free iterative CG/BiCGStab solver
- Weighted quadrature approach by Calabrò et al.\textsuperscript{1} is used for the on-the-fly generation matrix entries in the SpMv subroutine

$$S_{i,j} = \sum_{\alpha,\beta=1}^{d} \int_{[0,1]^2} g_{\alpha,\beta}(\xi) \partial_{\beta} \hat{B}_j(\xi) \left( \partial_{\alpha} \hat{B}_i(\xi) d\xi \right)$$

$$\simeq \sum_{\alpha,\beta=1}^{d} Q_{\alpha,\beta,i}^{WQ} \left( g_{\alpha,\beta}(\cdot) \partial_{\beta} \hat{B}_j(\cdot) \right)$$

\textsuperscript{1}CMAME 316, (2017) 606-622.
An example of data-flow computing in space

R.v.Nieuwoort: Implementation of the weighted quadrature approach by Tani et al. on MAX5 Lima DFE
An example of analogue computing

Example: 1d Poisson equation

- Solution to the linear system $Ax = b$ can be interpreted as the steady state limit of the initial value problem

$$\frac{dx(t)}{dt} = b - Ax(t), \quad x(0) = x_0$$
An example of analogue computing

Example: 1d Poisson equation

- Solution to the linear system $Ax = b$ can be interpreted as the steady state limit of the initial value problem

$$\frac{dx(t)}{dt} = b - Ax(t), \quad x(0) = x_0$$

- Analogue computers are efficient in modelling differential equations using op-amps but very expensive to operate at large scales
An example of analogue computing

Example: 1d Poisson equation

- Solution to the linear system $Ax = b$ can be interpreted as the steady state limit of the initial value problem

$$\frac{dx(t)}{dt} = b - Ax(t), \quad x(0) = x_0$$

- Analogue computers are efficient in modelling differential equations using op-amps but very expensive to operate at large scales
- Analogue computing can bring to new ideas, e.g., for data-flow computing architectures like FPGAs and quantum computers
An example of virtual analogue computing
Vision: Isogeometric DSO loop

**Our objective** is to develop an IGA toolbox for the efficient **Design**, **Simulation** and **Optimization** of screw machines with variable pitches

![Image of screw machine](image)

Courtesy of Andreas Brümmer, Dortmund University of Technology.

**Collaboration:** TU Kaiserslautern, TU Dortmund, TU Delft, JKU Linz
Vision: Isogeometric DSO loop

Challenges:

- Rotating geometries with tiny gaps (< 0.4mm) between rotors
- Multi-physics problem: compressible flows, thermal deformation, ...
- Prevent topology changes of the multi-patch structure

Design criteria:

- High-resolution capturing of shocks and discontinuities
- Support for current and future HPC platforms
- KISS (no unmaintainable hacks)
A 'novel' hardware-oriented numerics approach

Multi-patch Isogeometric Analysis:

- B-spline based iso-parametric FEM on unstructured coarse grid
- Fully structured high-order discretizations with hardware-optimized implementations on individual patches
- Multi-patch DG-coupling with minimal communication overhead
Overview

1. Introduction
   Scientific computing from a hardware perspective
   Isogeometric design-simulation-optimization loop

2. Elliptic 'grid' generation
   Planar parameterizations
   Volumetric parameterizations

3. Compressible flow solver
   Problem formulation
   Linearized FCT limiter
   Numerical examples

4. Implementation aspects
   Meta-programming techniques for heterogeneous HPC
Sliding grid technology - planar case

Approach: Create multi-patch parameterizations of fluid domain (O- or C-type + separator) and let the rotors slide along the inner boundaries

Pros: no topology changes, conforming/’nested’ nonconforming patches
Cons: Change of separator parameterization is not continuous in time
Elliptic ‘grid’ generation

**Algorithm²**: Given a *point cloud* describing the patch boundary create an analysis-suitable *parameterization* $\hat{G}_f : [0, 1]^2 \rightarrow \Omega_f$ as follows:

1. Generate boundary parameterizations $\gamma_k$, $k \in \{N, S, E, W\}$
2. Reparameterize opposite boundaries at ‘small-gap regions’ by constrained cord-length parameterization algorithm
3. Compute union of basis functions of opposite boundaries
4. Generate a bi-variate parameterization by solving with IgA
5. Adaptively refine knot spans where $\det D \hat{G}_m, f \leq \text{tol}$ and repeat

---

²J. Hinz, MM, C. Vuik, submitted to GMP 2018
Elliptic 'grid' generation

**Algorithm** \(^2\): Given a *point cloud* describing the patch boundary create an analysis-suitable *parameterization* \( \hat{G}_f : [0, 1]^2 \rightarrow \Omega_f \) as follows:

1. Generate boundary parameterizations \( \gamma_k, k \in \{N, S, E, W\} \)

---

\(^2\)J. Hinz, MM, C. Vuik, submitted to GMP 2018
Elliptic 'grid' generation

**Algorithm**: Given a *point cloud* describing the patch boundary create an analysis-suitable *parameterization* $\hat{G}_f : [0, 1]^2 \rightarrow \Omega_f$ as follows:

1. Generate boundary parameterizations $\gamma_k, k \in \{N, S, E, W\}$
2. Reparameterize opposite boundaries at 'small-gap regions' by constrained cord-length parameterization algorithm

---

$^2$J. Hinz, MM, C. Vuik, submitted to GMP 2018
Elliptic 'grid' generation

Algorithm\textsuperscript{2}: Given a point cloud describing the patch boundary create an analysis-suitable parameterization $\hat{G}_f : [0, 1]^2 \rightarrow \Omega_f$ as follows:

1. Generate boundary parameterizations $\gamma_k, k \in \{N, S, E, W\}$
2. Reparameterize opposite boundaries at 'small-gap regions' by constrained cord-length parameterization algorithm
3. Compute union of basis functions of opposite boundaries

\textsuperscript{2}J. Hinz, MM, C. Vuik, submitted to GMP 2018
Elliptic 'grid' generation

Algorithm\(^2\): Given a \textit{point cloud} describing the patch boundary create an analysis-suitable \textit{parameterization} \(\hat{G}_f : [0, 1]^2 \rightarrow \Omega_f\) as follows:

1. Generate boundary parameterizations \(\gamma_k, k \in \{N, S, E, W\}\)
2. Reparameterize opposite boundaries at 'small-gap regions' by constrained cord-length parameterization algorithm
3. Compute union of basis functions of opposite boundaries
4. Generate a bi-variate parameterization by solving with IgA

\[
\begin{align*}
\hat{g}_{22}x_{\xi\xi} - 2\hat{g}_{12}x_{\xi\eta} + \hat{g}_{11}x_{\eta\eta} &= 0 \\
\hat{g}_{22}y_{\xi\xi} - 2\hat{g}_{12}y_{\xi\eta} + \hat{g}_{11}y_{\eta\eta} &= 0
\end{align*}
\]

\(s.t. \quad \hat{G}_{m,f} \bigg|_{\hat{\Gamma}_{m,f}} = \Gamma_{m,f}\)

\(^2\)J. Hinz, MM, C. Vuik, submitted to GMP 2018
Elliptic 'grid' generation

Algorithm\(^2\): Given a point cloud describing the patch boundary create an analysis-suitable parameterization \( \hat{G}_f : [0,1]^2 \rightarrow \Omega_f \) as follows:

1. Generate boundary parameterizations \( \gamma_k, k \in \{N, S, E, W\} \)
2. Reparameterize opposite boundaries at 'small-gap regions' by constrained cord-length parameterization algorithm
3. Compute union of basis functions of opposite boundaries
4. Generate a bi-variate parameterization by solving with IgA

\[
\begin{align*}
\hat{g}_{22}x_{\xi \xi} - 2\hat{g}_{12}x_{\xi \eta} + \hat{g}_{11}x_{\eta \eta} &= 0 & \text{s.t. } \hat{G}_{m,f} \big|_{\hat{\Gamma}_{m,f}} &= \Gamma_{m,f} \\
\hat{g}_{22}y_{\xi \xi} - 2\hat{g}_{12}y_{\xi \eta} + \hat{g}_{11}y_{\eta \eta} &= 0
\end{align*}
\]

5. Adaptively refine knot spans where \( \det D\hat{G}_{m,f} \leq \text{tol} \) and repeat

\(^2\)J. Hinz, MM, C. Vuik, submitted to GMP 2018
Parameterizations of male/female rotors
One-patch separator geometry

**Approach:** Create parameterization for separator geometry

![Image showing separator geometry with CUSP markers and a transformation arrow]
One-patch separator geometry

**Approach:** Create parameterization for separator geometry, compute splitting curve

--- good

--- better
One-patch separator geometry

**Approach:** Create parameterization for separator geometry, compute splitting curve and (a) update point cloud and regenerate O-grids, or (b) adjust O-grids to match along separator using linear elasticity.
Two-patch separator geometry

**Approach:** Create two-patch separator
Two-patch separator geometry

**Approach:** Create two-patch separator with $C^0$ continuity along splitting curve
Two-patch separator geometry

**Approach:** Create two-patch separator with $C^0$ continuity along splitting curve, combine into one patch and run 'repair' algorithm.

run algorithm that allows for $C^0$ but requires bijective initial guess
Two-patch separator geometry

**Approach:** Create two-patch separator with $C^0$ continuity along splitting curve, combine into one patch and run 'repair' algorithm.
Generation of classical FEM meshes

**Approach:** Evaluate parameterization in parameter domain to obtain block-structured/unstructured (globally conforming) grids
Generation of classical FEM meshes

**Approach:** Evaluate parameterization in parameter domain to obtain block-structured/unstructured (globally conforming) grids
Volumetric grid generation

**Approach**: Interpolate (linearly) between planar parameterizations at different rotation angles to generate static volumetric parameterization.
Overview

1. Introduction
   Scientific computing from a hardware perspective
   Isogeometric design-simulation-optimization loop

2. Elliptic 'grid' generation [J. Hinz]
   Planar parameterizations
   Volumetric parameterizations

3. Compressible flow solver [A. Jaeschke]
   Problem formulation
   Linearized FCT limiter
   Numerical examples

4. Implementation aspects
   Meta-programming techniques for heterogeneous HPC
Compressible Euler equations

Divergence form

\[ U, t + \nabla \cdot F(U) = 0 \]

Quasi-linear form

\[ U, t + A(U) \cdot \nabla U = 0 \]

Conservative variables, inviscid fluxes, flux-Jacobian matrices

\[ U = \begin{bmatrix} \rho \\ \rho v \\ \rho E \end{bmatrix}, \quad F = \begin{bmatrix} \rho v \\ \rho v \otimes v + Ip \\ v(\rho E + p) \end{bmatrix}, \quad A = \frac{\partial F}{\partial U} \]

Equation of state (here for an ideal gas)

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

Similar formulations exist for primitive and entropy variables.
Compressible Euler equations

Divergence form

\[ U_t + \nabla \cdot F(U) = 0 \]

Quasi-linear form

\[ U_t + A(U) \cdot \nabla U = 0 \]

Conservative\(^a\) variables, inviscid fluxes, flux-Jacobian matrices

\[
U = \begin{bmatrix}
u_1 \\
\vdots \\
u_{d+2}
\end{bmatrix}, \quad F = \begin{bmatrix}
f^1_1 & \cdots & f^1_d \\
\vdots & \ddots & \vdots \\
f^d_{d+2} & \cdots & f^d_{d+2}
\end{bmatrix}, \quad A = \frac{\partial F}{\partial U}
\]

Notation

\[
f_k = [f^1_k, \ldots, f^d_k], \quad f^l = \begin{bmatrix}f^l_1 \\
\vdots \\
f^l_{d+2}\end{bmatrix}
\]

\(^a\)Similar formulations exist for primitive and entropy variables
Galerkin ansatz

Find solution $U$ at fixed time $t$ s.t. for all $W$

$$
\int_{\Omega} WU_{t} - \nabla W \cdot \mathbf{F}(U) \, d\Omega + \int_{\Gamma} WF^b(U, \cdot) \, ds = 0
$$

with boundary fluxes

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

Fletcher’s group formulation\(^3\)

$$
U_h = \sum_j B_j(x)U_j(t), \quad \mathbf{F}_h = \sum_j B_j(x)\mathbf{F}_j(t), \quad \mathbf{F}_j = \mathbf{F}(U_j)
$$

\(^3\)Fletcher, CMAME 37 (1983) 225–244.
SpMV-representation

Semi-discretized problem

\[
\begin{bmatrix}
M & \cdot & \cdot \\
\cdot & M & \cdot \\
\cdot & \cdot & \cdot
\end{bmatrix}
\begin{bmatrix}
\dot{u}_1 \\
\cdot \\
\cdot \\
\dot{u}_{d+2}
\end{bmatrix}
- \begin{bmatrix}
C & \cdot & \cdot \\
\cdot & C & \cdot \\
\cdot & \cdot & \cdot
\end{bmatrix}
\begin{bmatrix}
f_1^\top \\
\cdot \\
\cdot \\
f_{d+2}^\top
\end{bmatrix}
+ \begin{bmatrix}
S & \cdot & \cdot \\
\cdot & S & \cdot \\
\cdot & \cdot & \cdot
\end{bmatrix}
\begin{bmatrix}
f_{b1}^\top \\
\cdot \\
\cdot \\
f_{b_{d+2}}^\top
\end{bmatrix} = 0
\]

Constant coefficient matrices

\[
M = \left[ \int_\Omega B_i B_j \, d\Omega \right]_{i,j} \quad C = \left[ \int_\Omega \nabla B_i B_j \, d\Omega \right]_{i,j} \quad S = \left[ \int_\Gamma B_i B_j n \, ds \right]_{i,j}
\]

Stabilization of convective term by \textbf{Algebraic Flux Correction}\(^4\)

Linearized FCT\textsuperscript{5}

1. Perform variational mass lumping $M \rightarrow M_l = \text{diag}\{m_i\}$

$$m_i := \int_\Omega \varphi_i(x) \, d\Omega = \int_{\hat{\Omega}} \hat{B}_i(\xi) | \det D\hat{G}| d\xi > 0$$

Since

- B-spline basis functions satisfy PU property $\sum_j \hat{B}_j(\xi) \equiv 1$
- $\det D\hat{G} > \text{tol} > 0$ by design of the grid generation algorithm
- B-spline basis functions $\hat{\varphi}_i(\xi) > 0$ over their entire support

\textsuperscript{5}Kuzmin, MM, Shadid, Shashkov, JCP 229 (2010) 8766-8779.
Linearized FCT\(^5\)

1. Perform variational mass lumping \(M \rightarrow M_l = \text{diag}\{m_i\}\)
   
2. **Eliminate all negative eigenvalues from the flux-Jacobian**

\[
R_i^{\text{high}} := \sum_j c_{ij} \cdot F_j = \sum_{j \neq i} e_{ij} \cdot A_{ij}^{\text{Roe}}(U_j - U_i), \quad e_{ij} = 0.5(c_{ij} - c_{ji})
\]

by adding artificial viscosities \(D_{ij} := \|e_{ij}\| R_{ij} |\Lambda_{ij}| R_{ij}^{-1}\)

\[
R_i^{\text{low}} := \sum_{j \neq i} [e_{ij} \cdot A_{ij}^{\text{Roe}} + D_{ij}](U_j - U_i)
\]

Linearized FCT

1. Perform variational mass lumping \( M \rightarrow M_l = \text{diag}\{m_i\} \)
2. Eliminate all negative eigenvalues from the flux-Jacobian
3. Solve \( M_l \dot{U}^{\text{low}} = R^{\text{low}} + S^{\text{low}} \) by an explicit SSP-RK method

\[
M_l U^{(1)} = M_l U^n + \Delta t[R^n + S^n] \\
M_l U^{\text{low}} = \frac{1}{2} M_l U^n + \frac{1}{2} \left( M_l U^{(1)} + \Delta t[R^{(1)} + S^{(1)}] \right)
\]

Linearized FCT

1. Perform variational mass lumping $M \rightarrow M_l = \text{diag}\{m_i\}$
2. Eliminate all negative eigenvalues from the flux-Jacobian
3. Solve $M_l \dot{U}^{\text{low}} = R^{\text{low}} + S^{\text{low}}$ by an explicit SSP-RK method
4. Linearize antidiffusive fluxes about the low-order predictor

$$F_{ij} := m_{ij}(\dot{U}_i^{\text{low}} - \dot{U}_j^{\text{low}}) + D_{ij}^{\text{low}}(U_i^{\text{low}} - U_j^{\text{low}})$$

---

Linearized FCT\textsuperscript{5}

1. Perform variational mass lumping $M \rightarrow M_l = \text{diag}\{m_i\}$
2. Eliminate all negative eigenvalues from the flux-Jacobian
3. Solve $M_l \dot{U}^{\text{low}} = R^{\text{low}} + S^{\text{low}}$ by an explicit SSP-RK method
4. Linearize antidiffusive fluxes about the low-order predictor
5. Apply a generalized version of Zalesak’s multidimensional ‘nodal’ flux limiter to scalar indicator variables $\rho$ and $p$ ($p > 0 \Rightarrow \rho E > 0$) and add a limited antidiffusive correction

$$U_i^{n+1} = U_i^{\text{low}} + \frac{\Delta t}{m_i} \sum_{j \neq i} \min\{\alpha_i^\rho, \alpha_i^p\} F_{ij}$$

\textsuperscript{5}Kuzmin, MM, Shadid, Shashkov, JCP 229 (2010) 8766-8779.
Zalesak’s nodal FCT limiter

’Nodal’ flux limiter for \( u \in \{\rho, p\} \) is designed to ensure that for all \( i \)

\[
\min_{j: m_{ij} \neq 0} u_j^{\text{low}} =: u_i^{\text{min}} \leq u_i^{n+1} \leq u_i^{\text{max}} := \max_{j: m_{ij} \neq 0} u_j^{\text{low}}
\]

\[u_i^{n+1} = u_i^L + \frac{\Delta t}{m_i} \sum_{j \neq i} \alpha_{ij} f_{ij}^L\]

---

\(^6\)S. Zalesak, JCP 1979, 31(3), pp. 335–362
Positivity proof for B-spline based IgA

**Theorem**

When the same constraints are imposed on the *weights* of the B-spline function then the end-of-step solution stays within the upper and lower bounds set up by the low-order predictor.

**Proof:** Note that it is not the coefficients $u_{i \text{min}}$ that are the local bounds (as in nodal FEM) but the functions $u_{\text{min}}(x) = \sum_j u_{j \text{min}} \varphi(x)$.

Assume that for some $x^* \in \Omega$ we have $u^{n+1}(x^*) > u^{\text{max}}(x^*)$:

$$0 > u^{\text{max}}(x^*) - u^{n+1}(x^*) = \sum_j [u_{j \text{max}} - u_{j}^{n+1}] \varphi_j(x^*) > 0$$

q.e.d.
Numerical examples

Sod’s shock tube problem\(^7\)

Quadratic bi-variate B-spline basis functions with expl. SSP-RK(2).

Numerical examples

Sod’s shock tube problem

Quadratic bi-variate B-spline basis functions with expl. SSP-RK(2).

\[ \rho, \quad v_x, \quad p \]

\[ G.A. \ Sod, \ JCP \ 27 \ (1978) \ 1–31. \]
Overview

1 Introduction
   Scientific computing from a hardware perspective
   Isogeometric design-simulation-optimization loop

2 Elliptic 'grid' generation [J. Hinz]
   Planar parameterizations
   Volumetric parameterizations

3 Compressible flow solver [A. Jaeschke]
   Problem formulation
   Linearized FCT limiter
   Numerical examples

4 Implementation aspects
   Meta-programming techniques for heterogeneous HPC
Meta-programming techniques for heterogeneous HPC

**Strategy:** Implement one device-independent single-patch compute kernel and just-in-time compile it into hardware-, formulation- and algorithm-optimized dynamically loadable libraries per patch

**Computational building blocks**

- Open-Source IGA library G+Smo\(^8\) (JKU, RICAM)
- 3rd party linear algebra libraries: ArrayFire, Blaze, Eigen, VexCL, ...
- Fluid Dynamics Building Blocks expression templates library\(^9\)

\(^8\)https://gs.jku.at/trac/gismo
\(^9\)https://gitlab.com/mmoelle1/FDBB
Expression templates

Code that you write

```cpp
define vex::vector<float> x, y, z;
auto expr = x*y + x/y;
z = expr;
```

All arithmetic operations are kept symbolically as expression tree whose evaluation is delayed until the assignment to the vector z. All arithmetic operation are then fused into a single evaluation (≡loop).
Expression templates, cont’d

Compute kernel from VexCL

```cpp
kernel void vexcl_vector_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] + prm_2[idx] / prm_3[idx];
    }
}
```
# Fluid Dynamics Building Blocks

<table>
<thead>
<tr>
<th>High-level</th>
<th>ET’s for conservative/primitive variables, EOS, inviscid/viscous fluxes, flux Jacobians, and Riemann invariants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-level</td>
<td>Unified wrapper function API to core functionality of ETL’s: make_temp, tag, tie, +, -, *, /, abs, sqrt, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Armadillo</th>
<th>ArrayFire</th>
<th>Blaze</th>
<th>Blitz++</th>
<th>Eigen</th>
<th>IT++</th>
<th>MTL4</th>
<th>uBLAS</th>
<th>VexCL</th>
<th>ViennaCL</th>
<th>...</th>
</tr>
</thead>
</table>
\[ y \leftarrow (m_x \cdot m_x + m_y \cdot m_y + m_z \cdot m_z)/(\rho \cdot \rho) \]

Double precision performance

Performance [mflops]

Problem size [bytes]

Double precision performance

- Armadillo specific fdbb
- ArrayFire specific fdbb
- Blaze specific fdbb
- Blitz++ specific fdbb
- Eigen specific fdbb
- IT++ specific fdbb
- uBLAS specific fdbb
- VexCL specific fdbb
y ← (m_x * m_x + m_y * m_y + m_z * m_z) / (ρ * ρ)  
7 flop
Factory method design pattern

Code example: *single-patch inviscid fluxes*

```cpp
using var = Variables<eos::idealGas<T>, dim,
    EnumForm::conservative>;
using fty = Factory<config, device>;
auto U = fty::stateVector<var>();
auto dF = fty::inviscidFluxes<var>();
```

1-2 The **Variable** typedef specifies all internals of the formulation like
the equation of state, the spatial dimension, and the type of state
vector variables (thereby defining fluxes and flux-Jacobians)
Factory method design pattern

Code example: *single-patch inviscid fluxes*

```cpp
using var = Variables<eos::idealGas<T>, dim,
                      EnumForm::conservative>;
using fty = Factory<config, device>;
auto U = fty::stateVector<var>();
auto dF = fty::inviscidFluxes<var>();
```

3 The **Factory** typedef selects optimal LA backend (Eigen, VexCL), data structures (SoA, AoS) and matrix formats (CSR, ELL, Band) for the particular **device** and problem **config**.
Factory method design pattern

Code example: *single-patch inviscid fluxes*

```cpp
1 using var = Variables<eos::idealGas<T>, dim, EnumForm::conservative>;
2 using fty = Factory<config, device>;
3 auto U = fty::stateVector<var>();
4 auto dF = fty::inviscidFluxes<var>();
```

4-5 Vector and operator objects are created from the `Factory` typedef delegating memory allocation, etc. to the underlying LA backend.
Factory method design pattern

**Code example: single-patch inviscid fluxes**

```cpp
using var = Variables<eos::idealGas<T>, dim, EnumForm::conservative>;
using fty = Factory<config, device>;
auto U = fty::stateVector<var>();
auto dF = fty::inviscidFluxes<var>();
```

The operator* is overloaded such that dF*U unfolds to the SpMV representation of the discretized $\nabla \cdot \mathbf{F}(U)$ term, whereby fluxes, eos, etc. are implemented as lightweight expression templates in FDBB.
Factory method design pattern

Code example: *single-patch inviscid fluxes*

```cpp
using var = Variables<eos::idealGas<T>, dim,
EnumForm::conservative>; 
using fty = Factory<config, device>;
auto U = fty::stateVector<var>();
auto dF = fty::inviscidFluxes<var>();
```

**Is it worth the effort?** Yes, because you can tune the implementation 'behind the scenes', e.g., by switching from AoS to SoA, i.e.

$$\text{SpMatrix}<T> \ C[\text{dim}]; \quad \longrightarrow \quad \text{SpMatrix}<\text{CST}<T,1,\text{dim}> \ > \ \ C;$$

$$\text{Vector}<T> \ U[\text{dim}+2]; \quad \longrightarrow \quad \text{Vector}<\text{CST}<T,\text{dim}+2,1> \ > \ U;$$

to save memory and computing time by factor $O(\text{dim})$. 
Topics discussed:

- Parameterizations for screw machines with variable pitch
- Positivity-preserving IGA-solver for compressible flows
- Efficient implementation for heterogeneous HPC systems

Future work:

- THB-splines for solution-adapted parameterizations

Acknowledgements:
B. Jüttler, A. Mantzaflaris, D. Demidov, K. Iglberger
Financial support by EC under GA nº 678727 is acknowledged.
Consider sums of positive/negative antidiffusive fluxes into each node.

Limit antidiffusive flux if it exceeds the distance to upper/lower bounds.

Compute nodal correction factors:

\[ R^+_i = \min\{1, \frac{Q^+_i}{P^+_i}\} \quad \text{and} \quad \frac{R^-_i}{P^-_i}\]

Limit antidiffusive flux for edge \(ij\) by:

\[ \alpha_{ij} = \begin{cases} \min\{R^+_i, R^-_j\} & \text{for positive fluxes} \\ \min\{R^-_i, R^+_j\} & \text{for negative fluxes} \end{cases} \]

---

\(^{10}\) S. Zalesak, JCP 1979, 31(3), pp. 335–362
Zalesak’s flux limiter\textsuperscript{10}

- Consider sums of positive/negative antidiffusive fluxes into each node
- Limit antidiffusive flux if it exceeds the distance to upper/lower bounds

\textsuperscript{10}S. Zalesak, JCP 1979, 31(3), pp. 335–362
Zalesak’s flux limiter\textsuperscript{10}

- Consider sums of positive/negative antidiffusive fluxes into each node
- Limit antidiffusive flux if it exceeds the distance to upper/lower bounds

- Compute nodal correction factors

\[ R_i^+ = \min\{1, \frac{Q_i^+}{P_i^+}\} \quad \text{and} \quad R_i^- = \min\{1, \frac{Q_i^-}{P_i^-}\} \]

\textsuperscript{10}S. Zalesak, JCP 1979, 31(3), pp. 335–362
Zalesak’s flux limiter\textsuperscript{10}

- Consider sums of positive/negative antidiffusive fluxes into each node
- Limit antidiffusive flux if it exceeds the distance to upper/lower bounds

- Compute nodal correction factors

\[
R_i^+ = \min\{1, \frac{Q_i^+}{P_i^+}\} \quad \text{and} \quad R_i^- = \min\{1, \frac{Q_i^-}{P_i^-}\}
\]

- Limit antidiffusive flux for edge $ij$ by

\[
\alpha_{ij} = \begin{cases} 
\min\{R_i^+, R_j^-\} & \text{for positive fluxes} \\
\min\{R_i^-, R_j^+\} & \text{for negative fluxes}
\end{cases}
\]

\textsuperscript{10}S. Zalesak, JCP 1979, 31(3), pp. 335–362
Extended version of Zalesak’s FCT limiter

**Input:** predictor $u^L$ and antidiffusive fluxes $f_{ij}^u$, where $f_{ij}^u \neq f_{ji}^u$

1. Sums of positive/negative antidiffusive fluxes into node $i$

   $$P_i^+ = \sum_{j \neq i} \max\{0, f_{ij}^u\}, \quad P_i^- = \sum_{j \neq i} \min\{0, f_{ij}^u\}$$

2. Upper/lower bounds based on the local extrema of $u^L$

   $$Q_i^+ = m_i(\max_{i} u_i^\text{max} - u_i^L), \quad Q_i^- = m_i(\min_{i} u_i^\text{min} - u_i^L)$$

3. Correction factors $\alpha_{ij}^u = \alpha_{ji}^u$ to satisfy the FCT constraints

   $$\alpha_{ij}^u = \min\{R_{ij}, R_{ji}\}, \quad R_{ij} = \begin{cases} \min\{1, Q_i^+ / P_i^+\} & \text{if } f_{ij}^u \geq 0 \\ \min\{1, Q_i^- / P_i^-\} & \text{if } f_{ij}^u < 0 \end{cases}$$
Node-based transformation of control variables\textsuperscript{11}

- **Conservative variables**: density, momentum, total energy

\[
U_i = [\rho_i, (\rho v)_i, (\rho E)_i],\quad F_{ij} = [f^\rho_{ij}, f^{\rho v}_{ij}, f^{\rho E}_{ij}],\quad F_{ji} = -F_{ij}
\]

- **Primitive variables** $V = TU$: density, velocity, pressure

\[
V_i = [\rho_i, v_i, p_i],\quad v_i = \frac{(\rho v)_i}{\rho_i},\quad p_i = (\gamma - 1) \left[ (\rho E)_i - \frac{|(\rho v)_i|^2}{2\rho_i} \right]
\]

\[
G_{ij} = \left[ f^\rho_{ij}, f^{\rho v}_{ij}, f^p_{ij} \right] = T(U_i)F_{ij},\quad T(U_j)F_{ji} = G_{ji} \neq -G_{ij}
\]

- Raw antidiffusive fluxes for the velocity and pressure

\[
f^{\rho v}_{ij} = \frac{f^{\rho v}_{ij} - v_i f^\rho_{ij}}{\rho_i},\quad f^p_{ij} = (\gamma - 1) \left[ \frac{|v_i|^2}{2} f^\rho_{ij} - v_i \cdot f^{\rho v}_{ij} + f^{\rho E}_{ij} \right]
\]