Algebraic implementation of a flux limiter for heterogeneous computing

N. Valle, X. Álvarez, F.X. Trias, J. Castro and A. Oliva

Heat and Mass Transfer Technological Centre (CTTC), Universitat Politècnica de Catalunya - BarcelonaTech (UPC)

July 16, 2018

10th International Conference in Computational Fluid Dynamics
July, 9-13, 2018, Barcelona

Overview

Portability. Why?

Software

- Legacy codes
- Architecture-dependent
- Non-standard kernels

Portability. Why?

Software

- Legacy codes
- Architecture-dependent
- Non-standard kernels

Hardware

- New architectures
- Hybrid platforms
- Power consumption

Portability. Why?

Software

- Legacy codes
- Architecture-dependent
- Non-standard kernels

Hardware

- New architectures
- Hybrid platforms
- Power consumption

Challenge

How to design portable/hybrid platform codes?

Portability. How?

Idea

Mathematics are always portable!

Portability. How?

Idea

Mathematics are always portable!

“The reason MATLAB is so good at signal processing is that it was not designed for signal processing. It was designed to do mathematics.“

Jim McClellan
GeorgiaTech

Portability. How?

Idea

Mathematics are always portable!

"The reason MATLAB is so good at signal processing is that it was not designed for signal processing. It was designed to do mathematics."

Jim McClellan
GeorgiaTech

May approaching dedicated scientific computing codes from an algebraic perspective help?

Portability. What for?

Casting **computational** operations into **algebraic** forms provides with several advantages:

- fewer number of computing kernels $\rightarrow$ portability
- mathematical formality $\rightarrow$ analysis

Remark[1]

For a typical DNS simulation of an incompressible flow, almost 90% of the operations can be cast into 3 basic kernels:

- `SpMV`
- `DOT`
- `axpy`

[1] Guillermo Oyarzun et al. "Portable implementation model for CFD simulations. Application to hybrid CPU/GPU supercomputers". In: *Int. J. Comut. Fluid Dyn.* 31.9 (2017), pp. 396–411.

Scope

How to design a flux limiter kernel from an algebraic approach?

Advantages

With this approach we aim at:

- High portability
- High degree of abstraction

Scope

How to design a flux limiter kernel from an algebraic approach?

Advantages

With this approach we aim at:

- High portability
- High degree of abstraction

Disclaimer

We are not after:

- Discussing Flux Limiters
- Optimize performance

Mathematical Machinery

Algebraic Topology

Your mesh. A starting point.

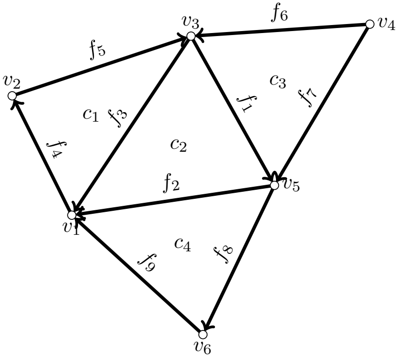


Figure 1: Primal mesh

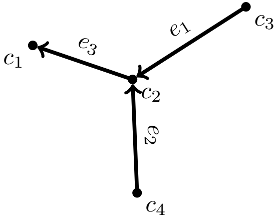
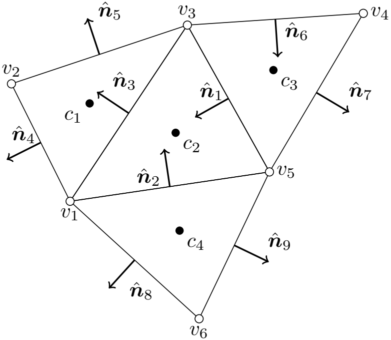


Figure 2: Dual mesh

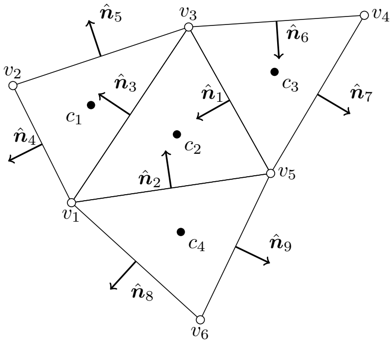
Algebraic Topology



DeRahm Cohomology

$$\begin{array}{ccccccccccc}
\mathbb{R} & \rightarrow & \Lambda_0(\Omega) & \rightarrow & \Lambda_1(\Omega) & \rightarrow & \Lambda_2(\Omega) & \rightarrow & \Lambda_3(\Omega) & \rightarrow & 0 \\
 & & \updownarrow \star & & \updownarrow \star & & \updownarrow \star & & \updownarrow \star & & \\
0 & \leftarrow & \Lambda^3(\Omega) & \leftarrow & \Lambda^2(\Omega) & \leftarrow & \Lambda^1(\Omega) & \leftarrow & \Lambda^0(\Omega) & \leftarrow & \mathbb{R}
\end{array}$$

Algebraic Topology

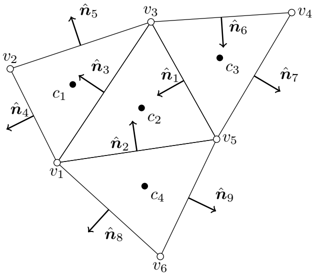


DeRahm Cohomology

Under the hood of:

- Staggered grid
- Symmetry preserving

A graph to rule them all



Operators

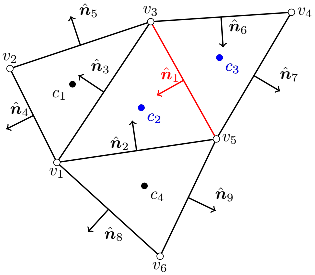
Operators can be constructed from graph information.

$$
T_{cf} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array}
\begin{array}{c} \begin{array}{ccccccccc} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 & f_9 \end{array} \\
\begin{pmatrix}
0 & 0 & +1 & -1 & -1 & 0 & 0 & 0 & 0 \\
+1 & +1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & +1 & -1 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & -1
\end{pmatrix} \end{array}
$$

A graph to rule them all

Example: Gradient operator

$$\int_{c_3}^{c_2} \nabla P = P_2 - P_3$$

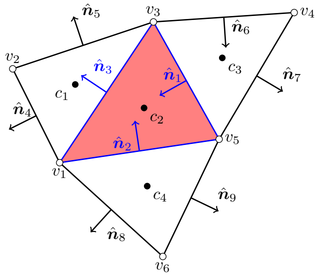


$$T_{cf} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} \begin{array}{ccccccccc} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 & f_9 \end{array} \\ \begin{pmatrix} 0 & 0 & +1 & -1 & -1 & 0 & 0 & 0 & 0 \\ +1 & +1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & +1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{pmatrix} \end{array}$$

A graph to rule them all

Example: Divergence operator

$$\int_{c_2} \nabla \cdot \vec{u} = \int_{\partial c_2} \vec{u}\hat{n}_f \approx \sum_{f \in c_2} S_f u_f$$



$$T_{cf} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} \begin{matrix} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 & f_9 \end{matrix} \\ \begin{pmatrix} 0 & 0 & +1 & -1 & -1 & 0 & 0 & 0 & 0 \\ +1 & +1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & +1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{pmatrix} \end{array}$$

Summary

Metric

Development of numerical method in terms of geometric entities.

$$\Omega_f = \Delta_x S_f$$

Symmetry-preserving

$$GRAD = -\Omega_f^{-1} DIV^T = -(\Delta_x S_f)^{-1} (T_{cf} S_f)^T = -(\Delta_x)^{-1} T_{cf}^T$$

Highlights

- The definition of **star ($\star$)** determines **dual operators**
- Preserve important quantities

Flux Limiters

Flux Limiters

Typically, flux limiters are stated in the following form:

$$\theta_f = \theta_C + \Psi(r)\left(\frac{\theta_D - \theta_U}{2}\right)$$

$$r_f = \frac{\theta_C - \theta_U}{\theta_D - \theta_C} = \frac{\Delta_U \theta_c}{\Delta_u \theta_c}$$

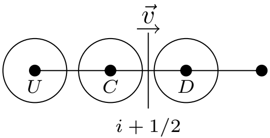


Figure 3: Classical stencil for the computation of the gradient ratio at face $i + 1/2$. U, C and D correspond to the upstream, centered and downstream nodes.

Flux Limiters

Reformulation in terms of matrices [2]

Rearrangement:

$$\theta_f = \frac{\theta_D + \theta_U}{2} + \frac{\Psi(r) - 1}{2} (\theta_D - \theta_U)$$

Matrix formulation:

$$\theta_f = (\Pi_{C \to F} + F(r)_{C \to F})\, \theta_c$$

Dynamic addition of artificial diffusivity as a function of r

[2] F.X. Trias et al. "Symmetry-preserving discretization of Navier–Stokes equations on collocated unstructured grids". In: *J. Comput. Phys.* 258 (Feb. 2014), pp. 246–267.

Gradient Ratio

How to compute the gradient ratio?

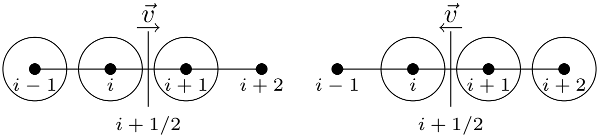


Figure 4: Switched stencil for a typical flux limiter

Gradient Ratio

How to compute the gradient ratio?

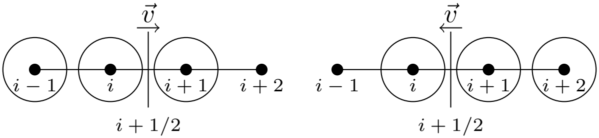


Figure 4: Switched stencil for a typical flux limiter

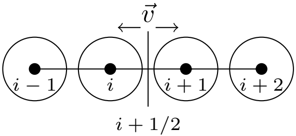


Figure 5: Algebraic stencil for an algebraic flux limiter

Gradient Ratio

How to compute the gradient ratio?

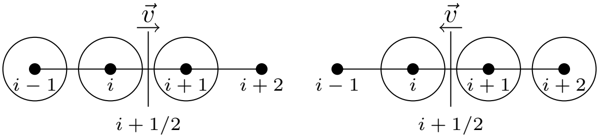


Figure 4: Switched stencil for a typical flux limiter

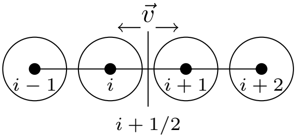


Figure 5: Algebraic stencil for an algebraic flux limiter

$$r_f = \frac{\theta_C - \theta_U}{\theta_D - \theta_C} = \frac{\Delta_U \theta_c}{\Delta_u \theta_c}$$

Gradient Ratio

How to compute the gradient ratio?

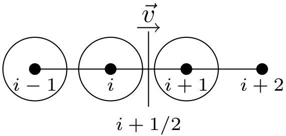


Figure 4: Switched stencil for a typical flux limiter

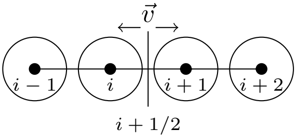


Figure 5: Algebraic stencil for an algebraic flux limiter

$$r_f = \frac{\theta_C - \theta_U}{\theta_D - \theta_C} = \frac{\Delta_U \theta_c}{\Delta_u \theta_c}$$

$$\Delta_u = S(u) T_{cf} \theta_c$$

Gradient Ratio

How to compute the gradient ratio?

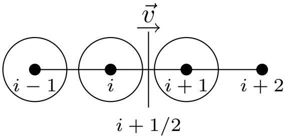


Figure 4: Switched stencil for a typical flux limiter

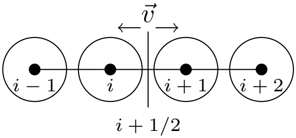


Figure 5: Algebraic stencil for an algebraic flux limiter

$$r_f = \frac{\theta_C - \theta_U}{\theta_D - \theta_C} = \frac{\Delta_U \theta_c}{\Delta_u \theta_c}$$

$$\Delta_u = S(u) T_{cf} \theta_c$$

How to compute Δ_U?

Gradient Ratio

Computing Δ_U

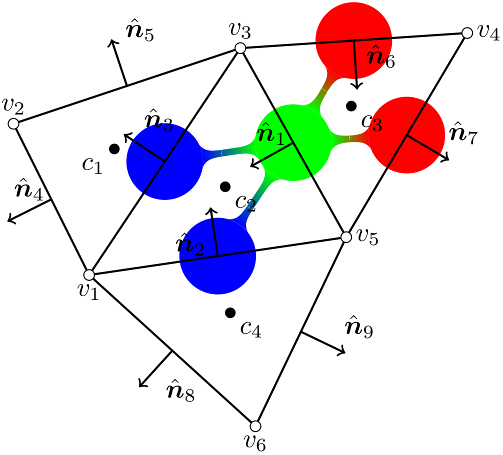
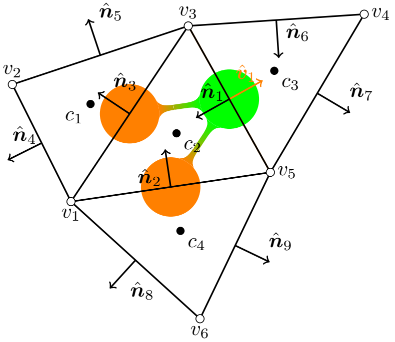


Figure 6: Upstream and Downstream adjacency faces

Gradient Ratio

Idea

1. Vectorize differences
2. Sum up upstream faces
3. Project over the normal

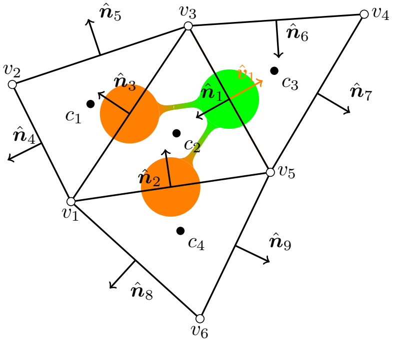


Gradient Ratio

Idea

1. Vectorize differences
2. Sum up upstream faces
3. Project over the normal

$$\Delta_U = N \left(\mathbb{I}_d \otimes A^U_{FF(u)} \right) N^T \Delta$$



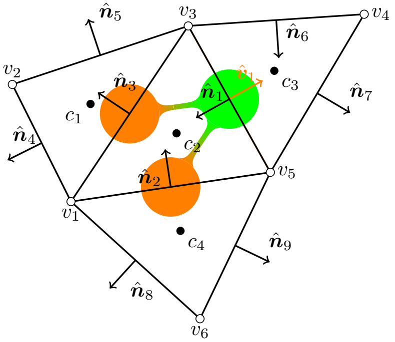
Gradient Ratio

Idea

1. Vectorize differences
2. Sum up upstream faces
3. Project over the normal

$$\Delta_U = N \left(\mathbb{I}_d \otimes A^U_{FF(u)} \right) N^T \Delta$$

$$N = \begin{pmatrix} n_{1x} & 0 & 0 & n_{1y} & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & n_{9x} & 0 & \dots & n_{9y} \end{pmatrix}$$



Gradient Ratio

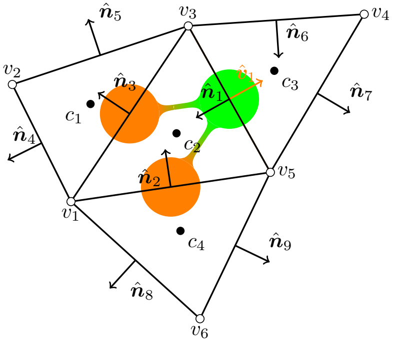
Idea

1. Vectorize differences
2. Sum up upstream faces
3. Project over the normal

$$\Delta_U = N\left(\mathbb{I}_d \otimes A^U_{FF(u)}\right) N^T \Delta$$

$$N = \begin{pmatrix} n_{1x} & 0 & 0 & n_{1y} & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & n_{9x} & 0 & \dots & n_{9y} \end{pmatrix}$$

$$A^U_{FF(u)} = \frac{1}{2}\left(S(u)A_{FF} - A^D_{FF}\right)$$



Gradient Ratio

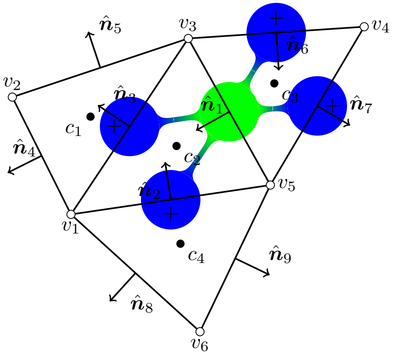
Idea

1. Vectorize differences
2. Sum up upstream faces
3. Project over the normal

$$\Delta_U = N\left(\mathbb{I}_d \otimes A^U_{FF(u)}\right) N^T \Delta$$

$$N = \begin{pmatrix} n_{1x} & 0 & 0 & n_{1y} & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & n_{9x} & 0 & \dots & n_{9y} \end{pmatrix}$$

$$A^U_{FF(u)} = \frac{1}{2}\left(S(u)A_{FF} - A^D_{FF}\right)$$



Gradient Ratio

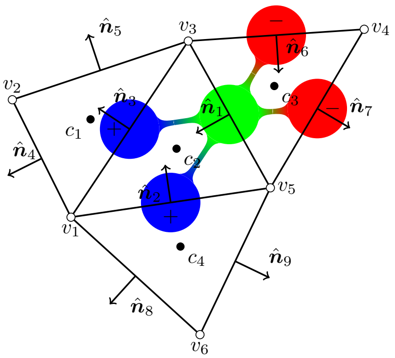
Idea

1. Vectorize differences
2. Sum up upstream faces
3. Project over the normal

$$\Delta_U = N \left(\mathbb{I}_d \otimes A^U_{FF(u)} \right) N^T \Delta$$

$$N = \begin{pmatrix} n_{1x} & 0 & 0 & n_{1y} & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & n_{9x} & 0 & \dots & n_{9y} \end{pmatrix}$$

$$A^U_{FF(u)} = \frac{1}{2} \left(S(u) A_{FF} - A^D_{FF} \right)$$



Gradient Ratio

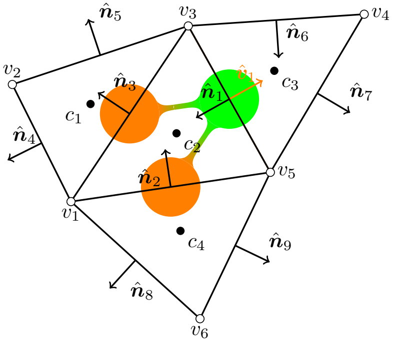
Idea

1. Vectorize differences
2. Sum up upstream faces
3. Project over the normal

$$\Delta_U = N \left(\mathbb{I}_d \otimes A^U_{FF(u)} \right) N^T \Delta$$

$$N = \begin{pmatrix} n_{1x} & 0 & 0 & n_{1y} & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & n_{9x} & 0 & \dots & n_{9y} \end{pmatrix}$$

$$A^U_{FF(u)} = \frac{1}{2} \left(S(u) A_{FF} - A^D_{FF} \right)$$



Computational framework

The code has been implemented into HPC^2 - a fully-portable, algebra-based framework for heterogeneous computing [3].

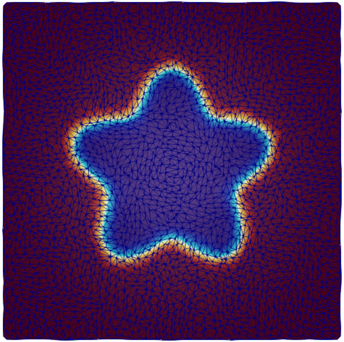
Operation	SpMV	axpy	axdy	shft	scal	vmax vmin	smax smin	sign
$S(u)$	0	0	0	0	0	0	0	1
$\Delta_U\theta$	3	1	0	0	0	0	0	0
$\Delta_u\theta$	2	0	0	0	0	0	0	0
r_f	0	0	1	0	0	0	0	0
$SUPERBEE(r)$	0	0	0	1	1	1	3	0
$F(r)_{C\to F}$	0	0	0	0	1	0	0	0
Euler	6	2	0	0	0	0	0	0
total	11	3	1	1	2	1	3	1

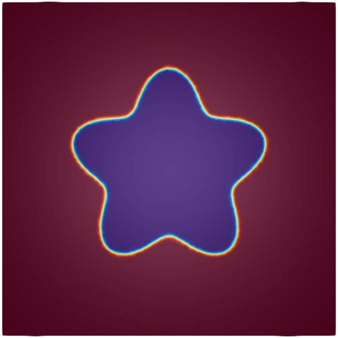
Table 1: Operation count per time step with SUPERBEE and Euler integration [4].

[3] X Álvarez et al. "HPC^2 - a fully-portable, algebra-based framework for heterogeneous computing. Application to CFD". . In: *Comput. Fluids (published online)* (2018).

[4] X Álvarez et al. "Integration of a flux limiter into a fully-portable, algebra-based framework for heterogeneous computing". In: *Tenth Int. Conf. Comput. Fluid Dyn.* Barcelona, 2018.

Advection of a scalar field





Profiles

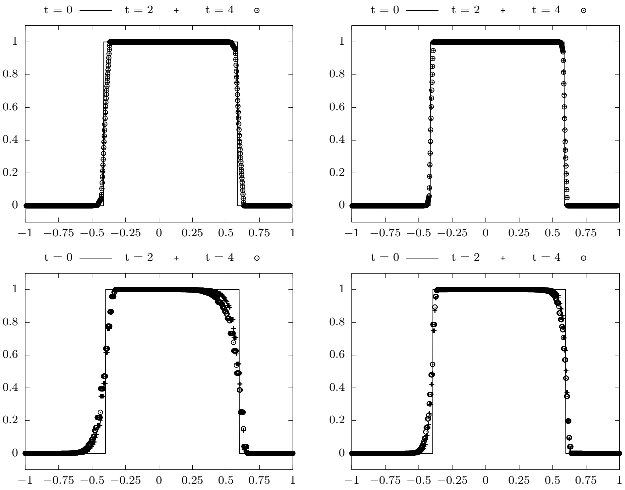


Figure 7: Left column corresponds to a meshes with a characteristic lenght of $\Delta x = 1/32$, while right columns are produced with a characteristic length of $\Delta x = 1/64$.

Conclusions

Highlights

- Flux limiters have been implemented into a portable platform
- Flux limiters **CAN** be cast in an algebraic form
- New conceptual platform developed

Future Work

- Analyze flux limiters properties
- Assess flux limiters design
- Improve gradient reconstruction strategies

Conclusions

Thank you for your attention!