Developing methods for exascale CFD simulations at high orders

David Moxey
College of Engineering, Maths & Physical Sciences
University of Exeter

Chris Cantwell, Martin Vymazal & Spencer Sherwin
Department of Aeronautics, Imperial College London

Platform for Advanced Scientific Computing Conference,
Basel, Switzerland

2nd July 2018

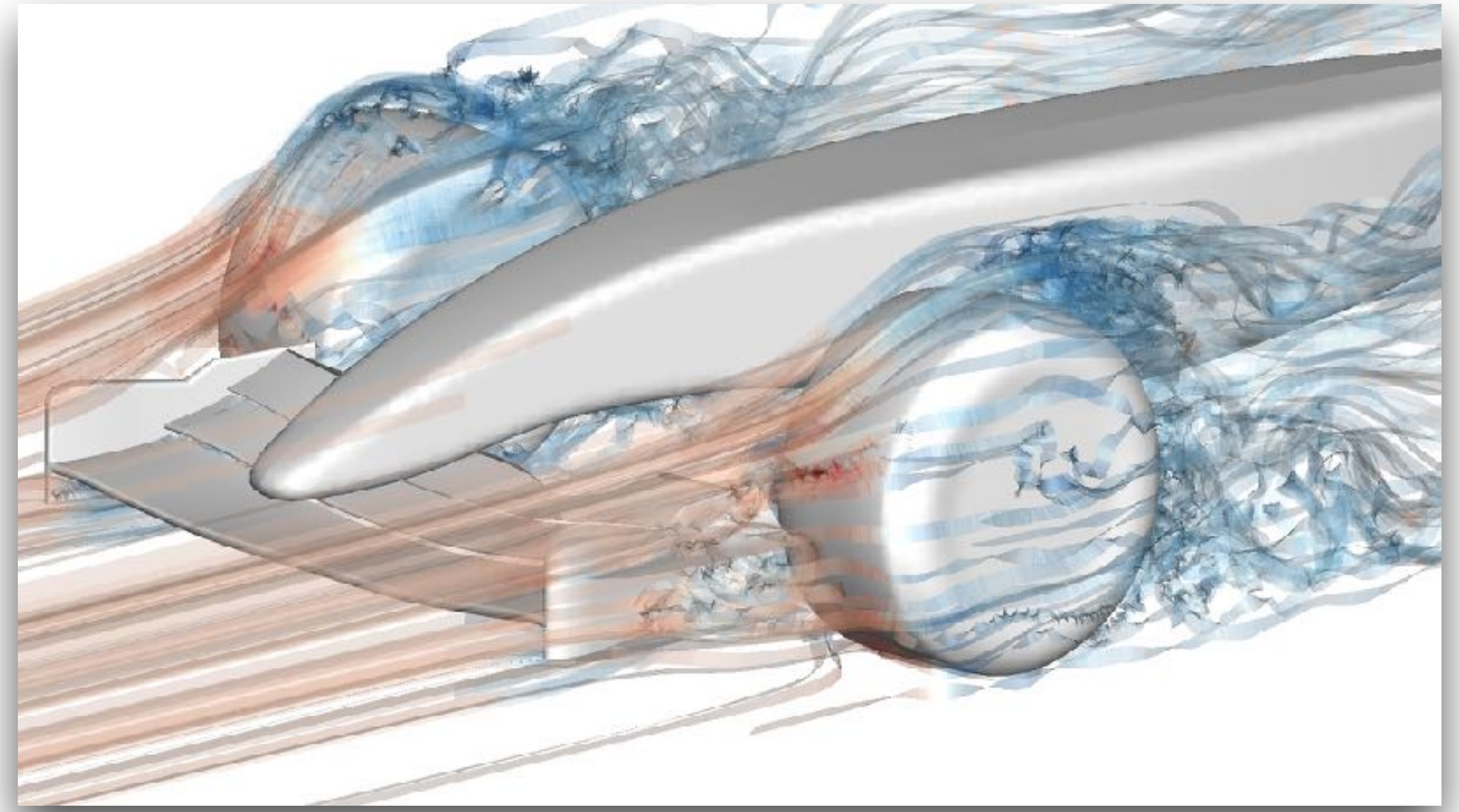
ExaFLOW

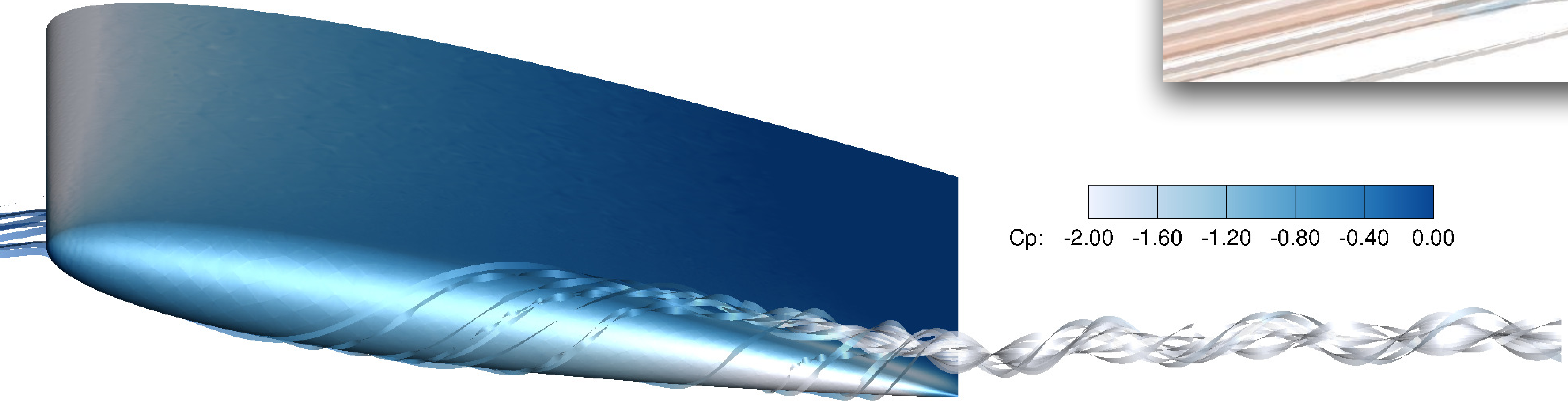
Outline

- Challenges for exascale: hardware landscape
- The spectral/*hp* element method
- Exploiting vectorisation
- Performance results
- Summary

What CFD do we want to do at exascale?

- Industrial simulations at high Reynolds numbers
- Things that RANS struggles with: high-fidelity, detachment, vortex interaction
- SVV LES formulation of incompressible NS





Lombard, Moxey, Hoessler, Dhandapani, Taylor and Sherwin
Implicit large-eddy simulation of a wingtip vortex, AIAA Journal **54** (2), 2016

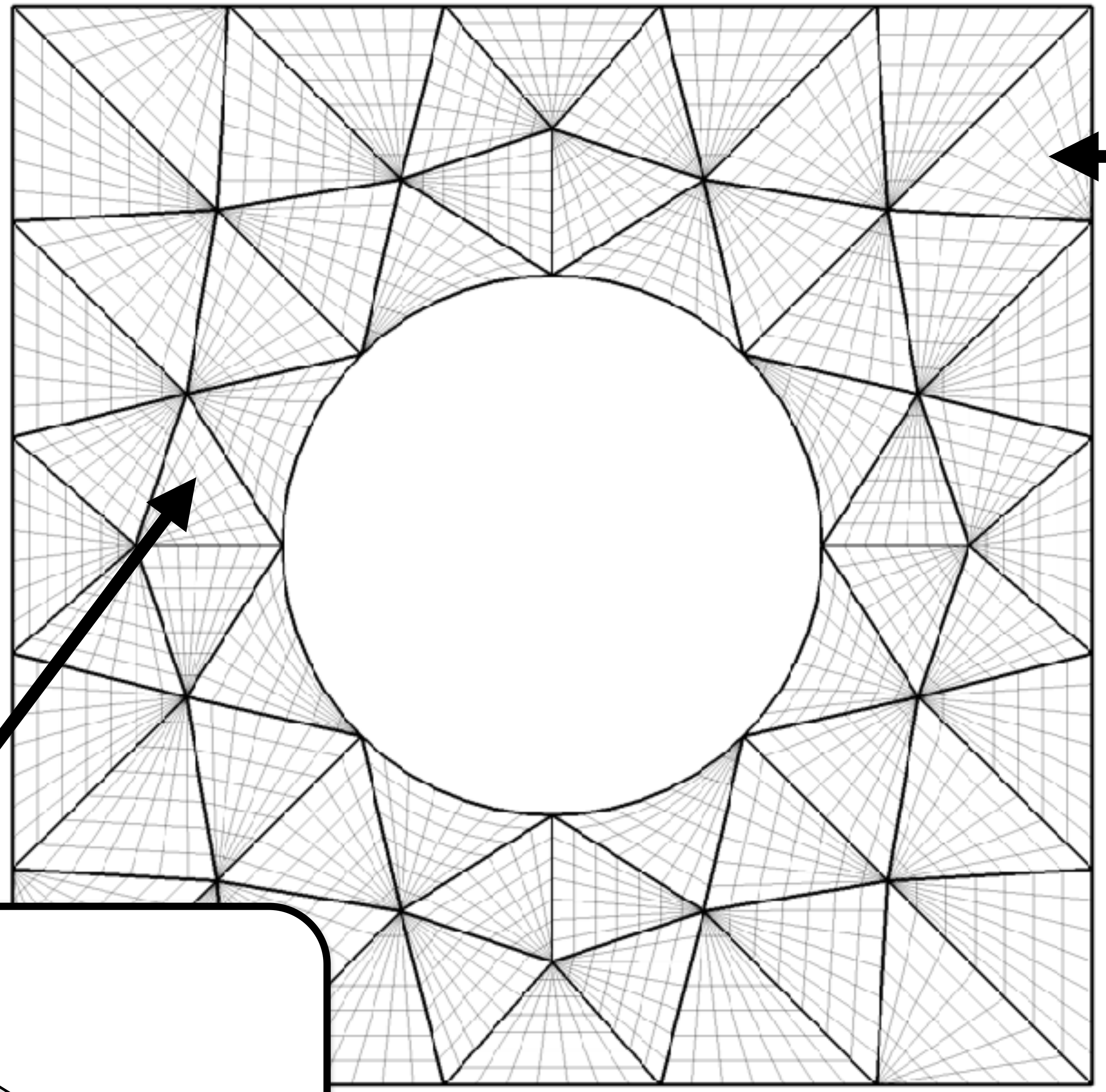
Why is exascale CFD hard?

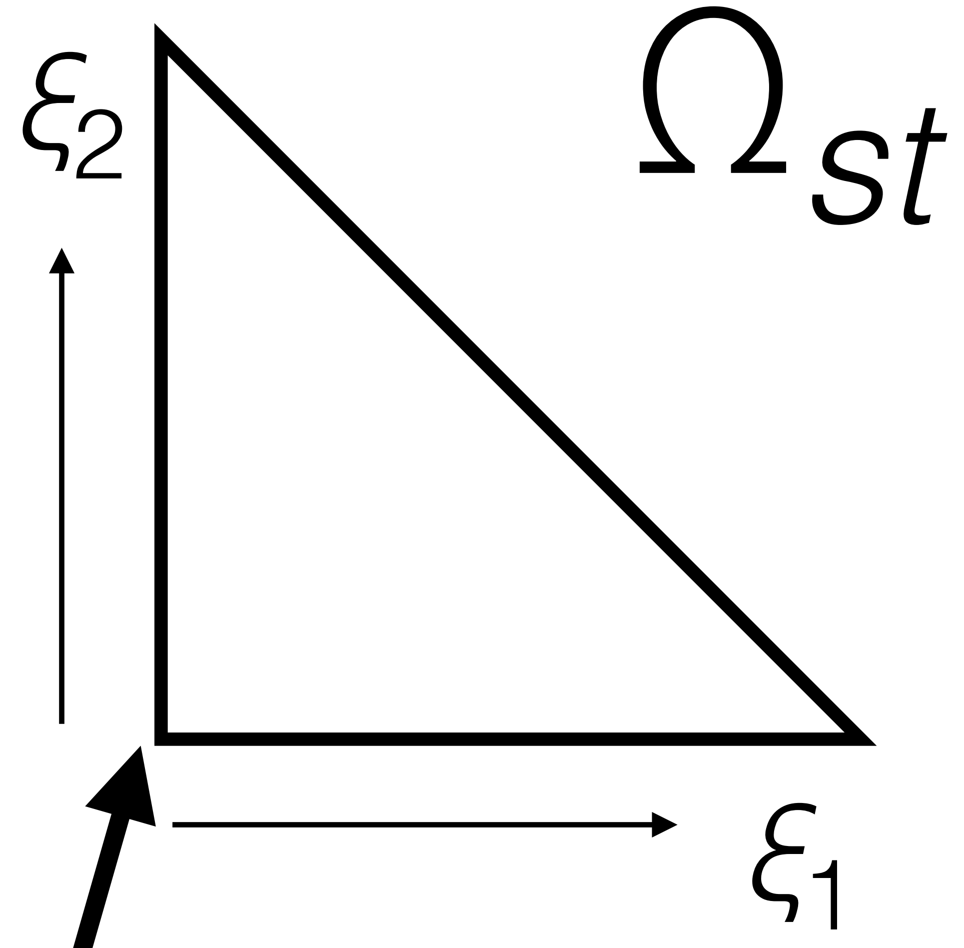
- Ideally, want really fast single-core nodes with lots of memory bandwidth
- Instead, many cores per node @ lower clock speed
- Very limited memory bandwidth, complicated memory hierarchies

Therefore need algorithms with **high arithmetic intensities** that can actually use FLOPS available

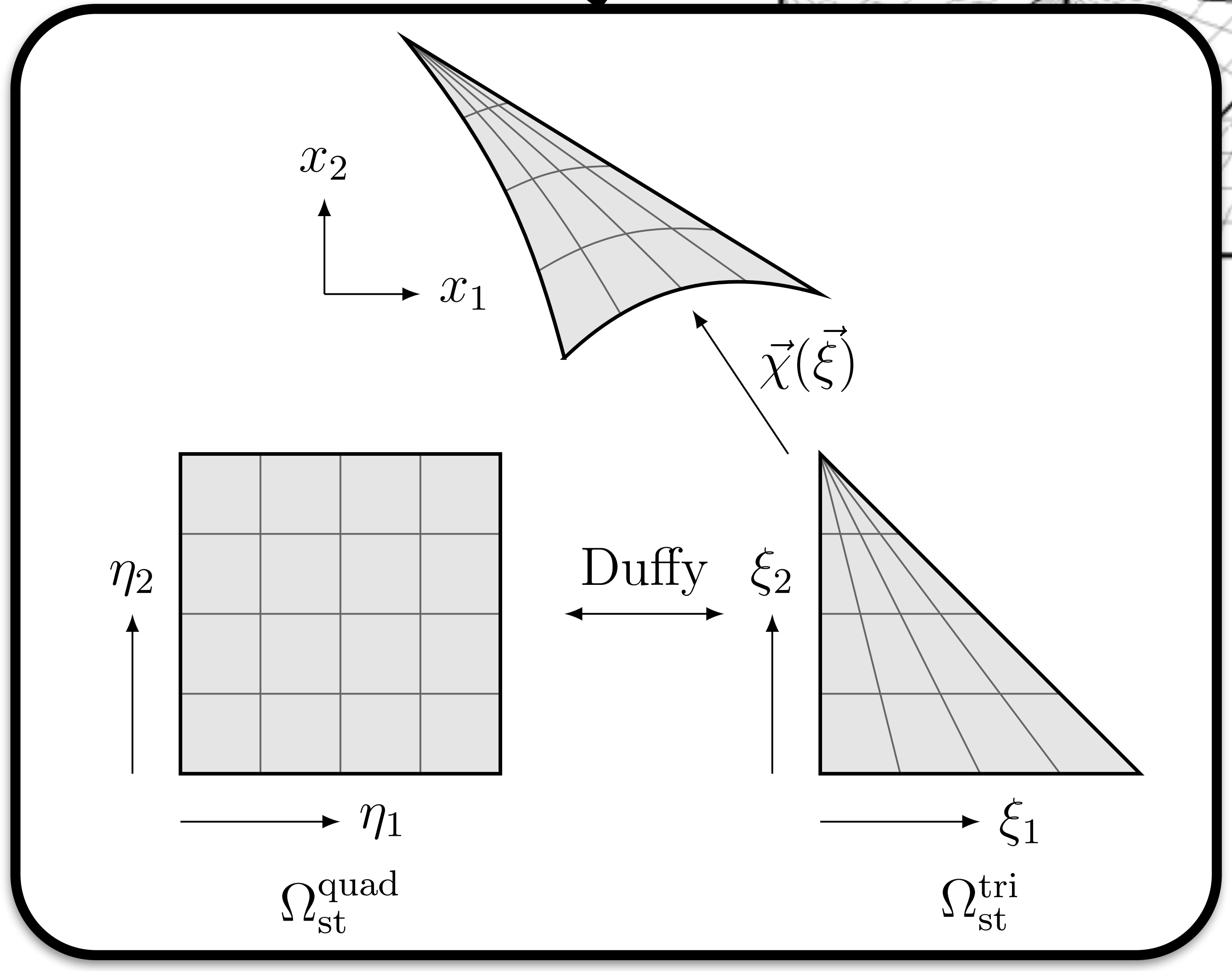
✓ high-order methods

Spectral/*hp* element method

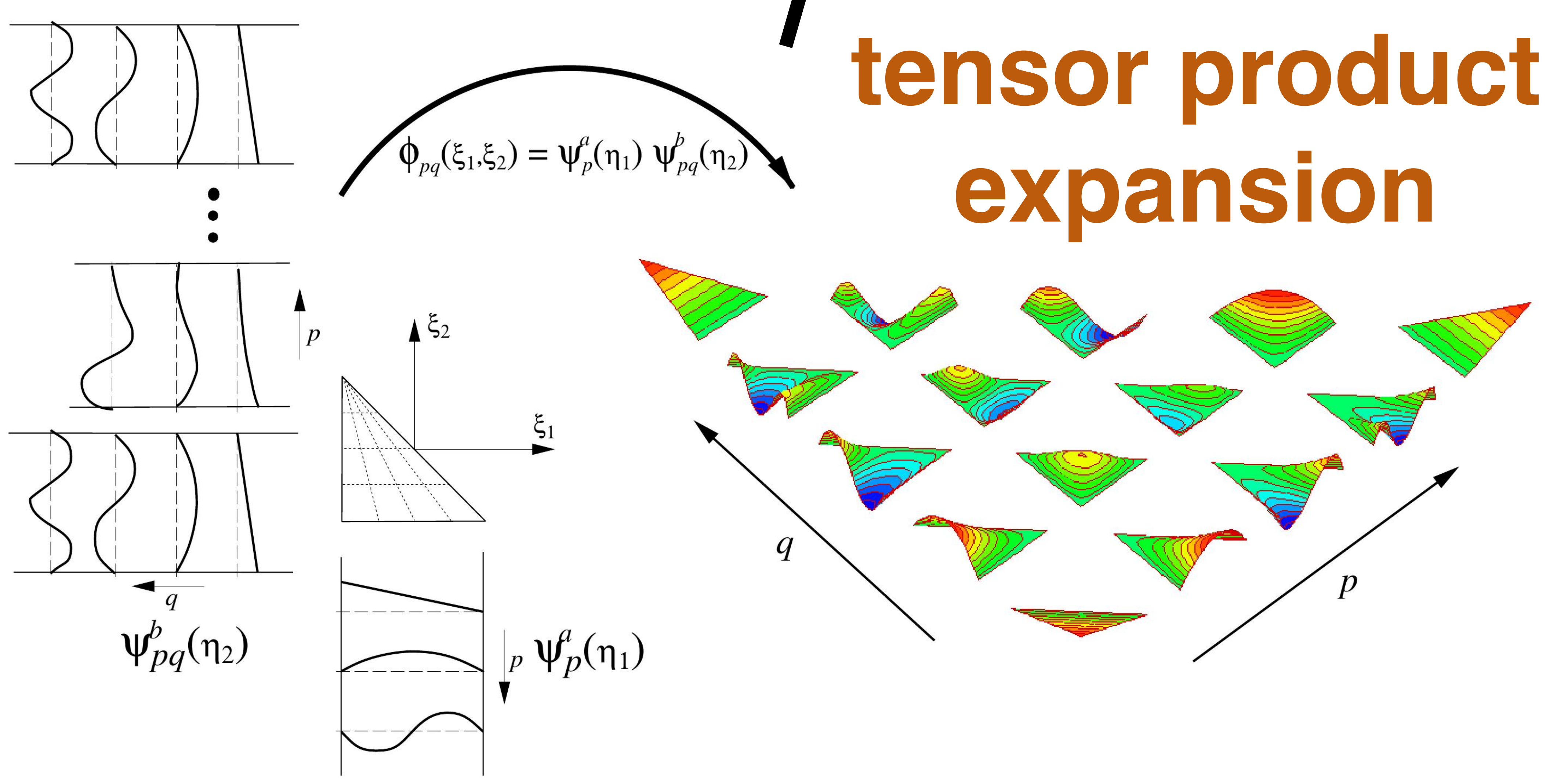




map from
reference
element

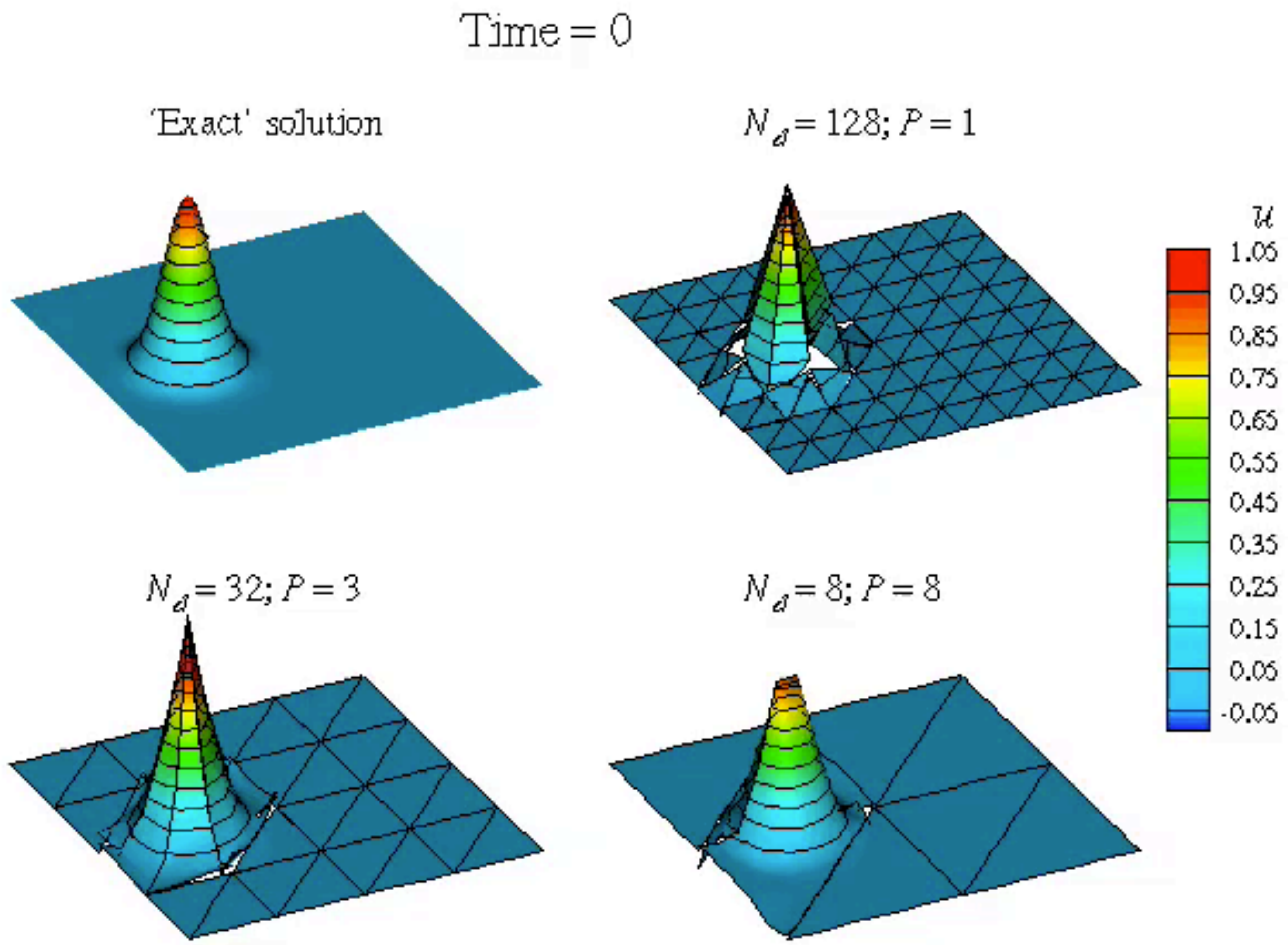


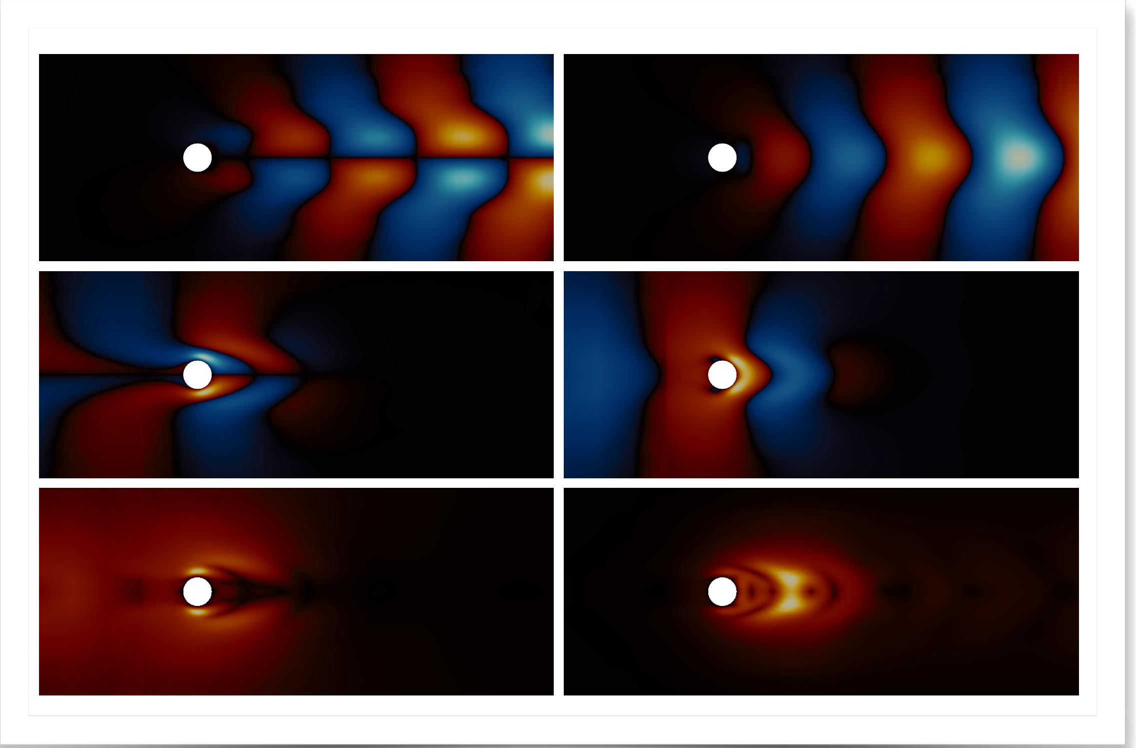
collapsed coordinates for quadrature

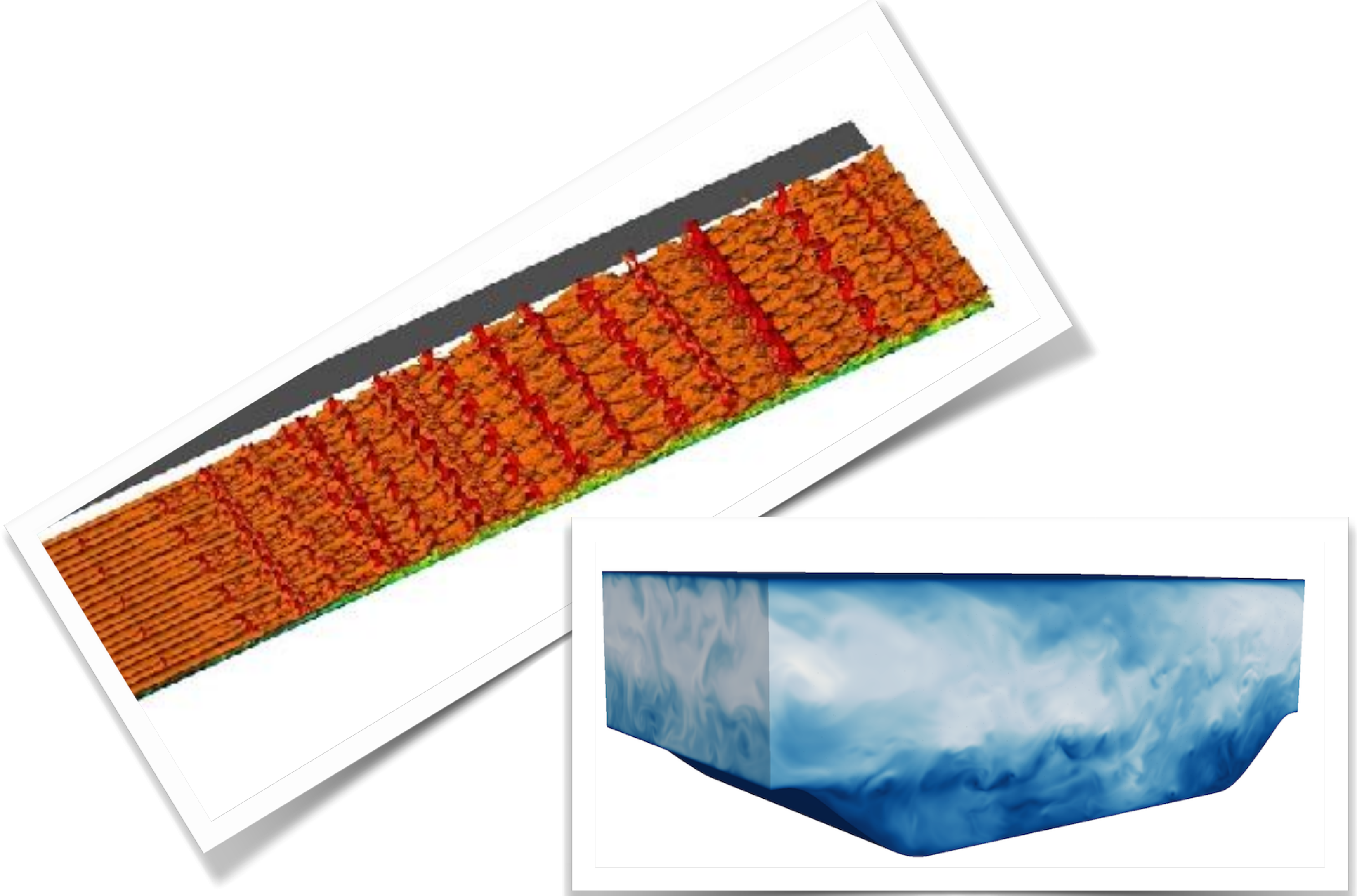


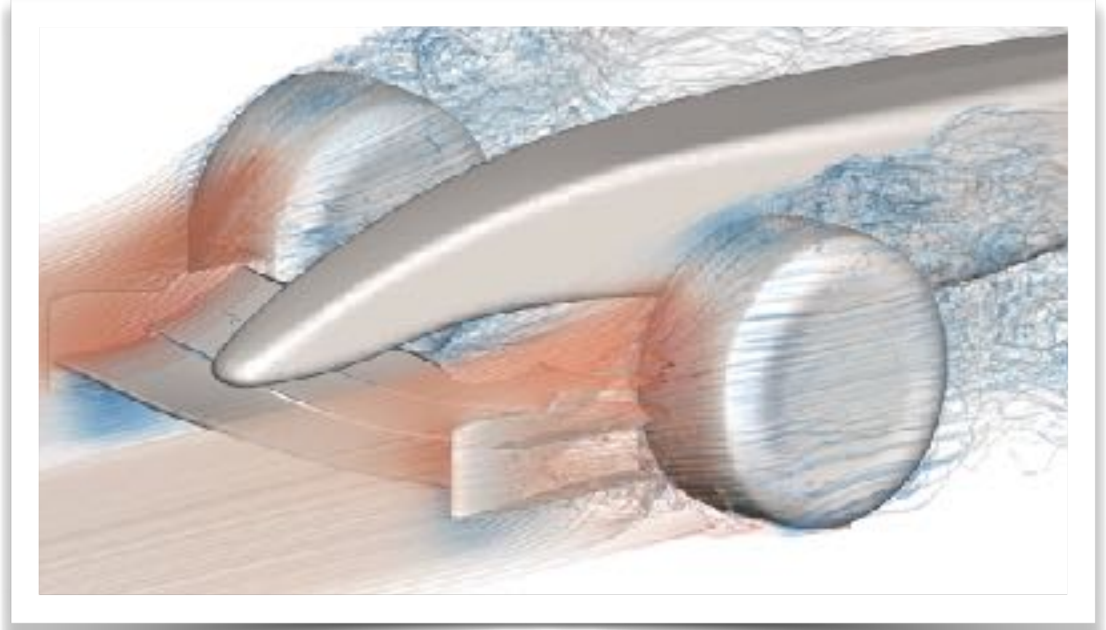
tensor product expansion

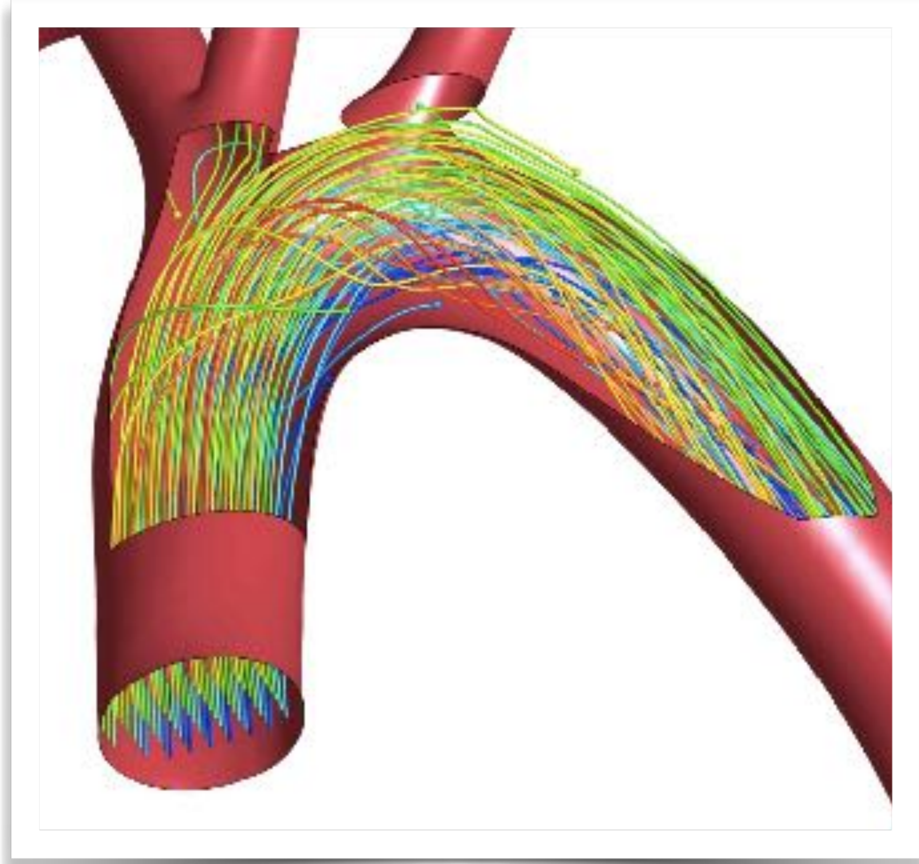
Why high order?

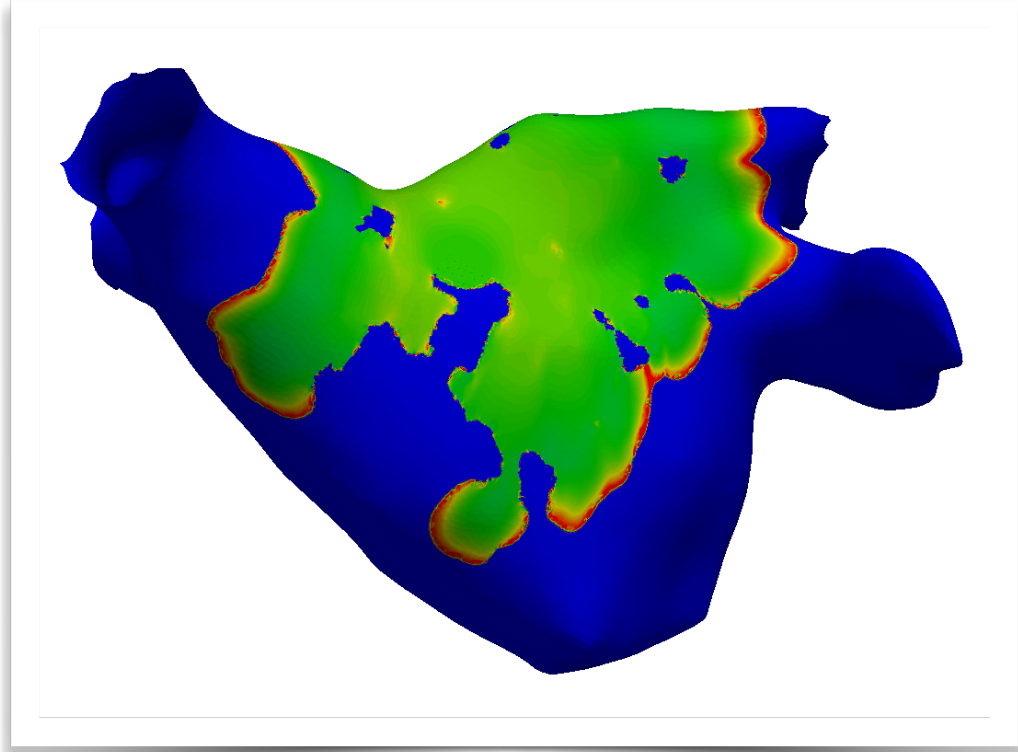


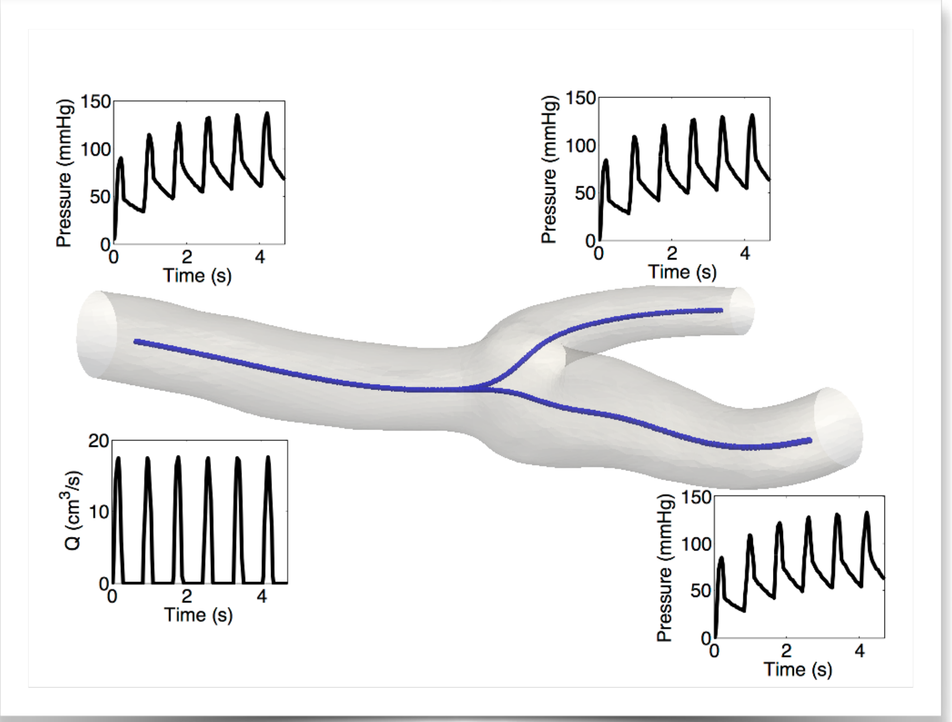


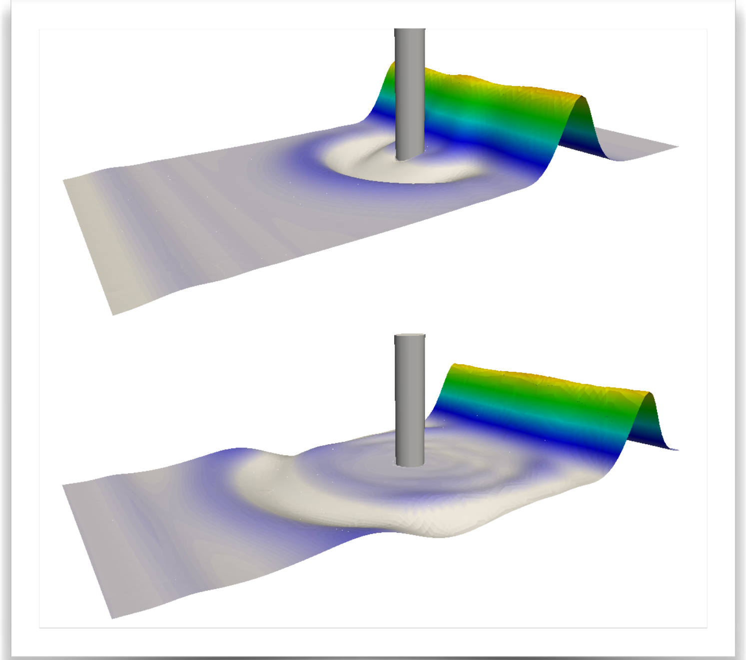


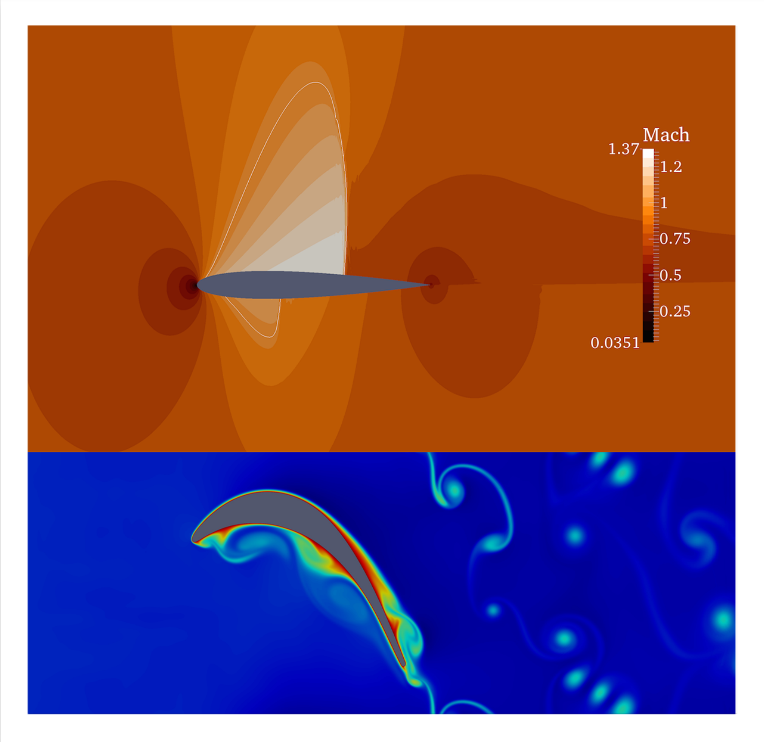


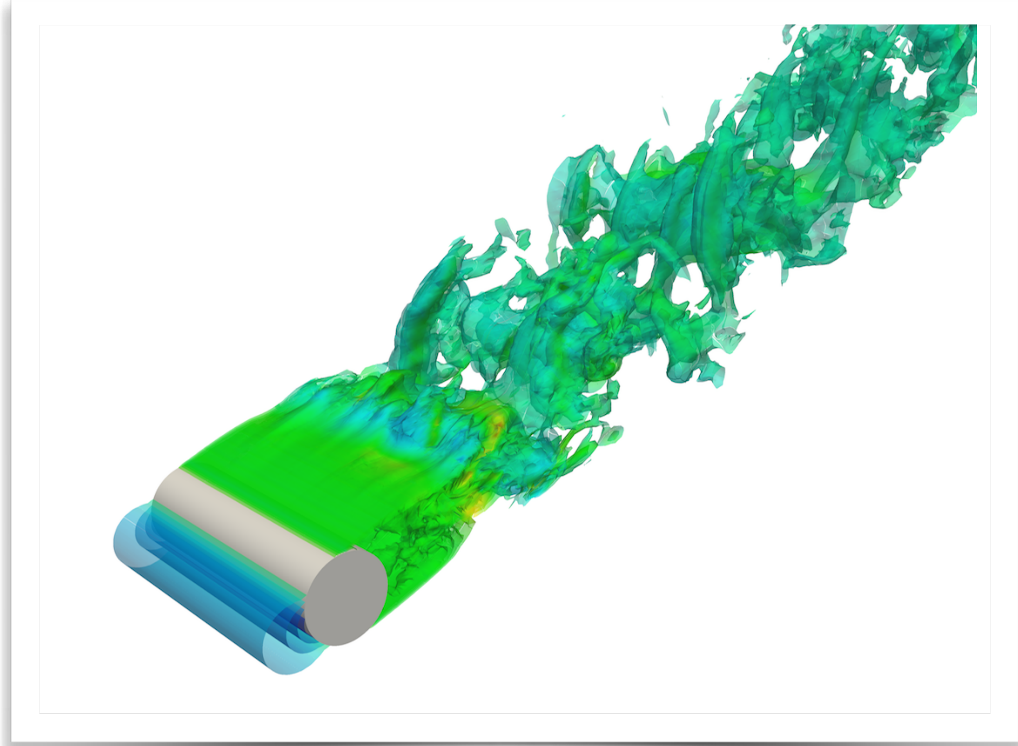




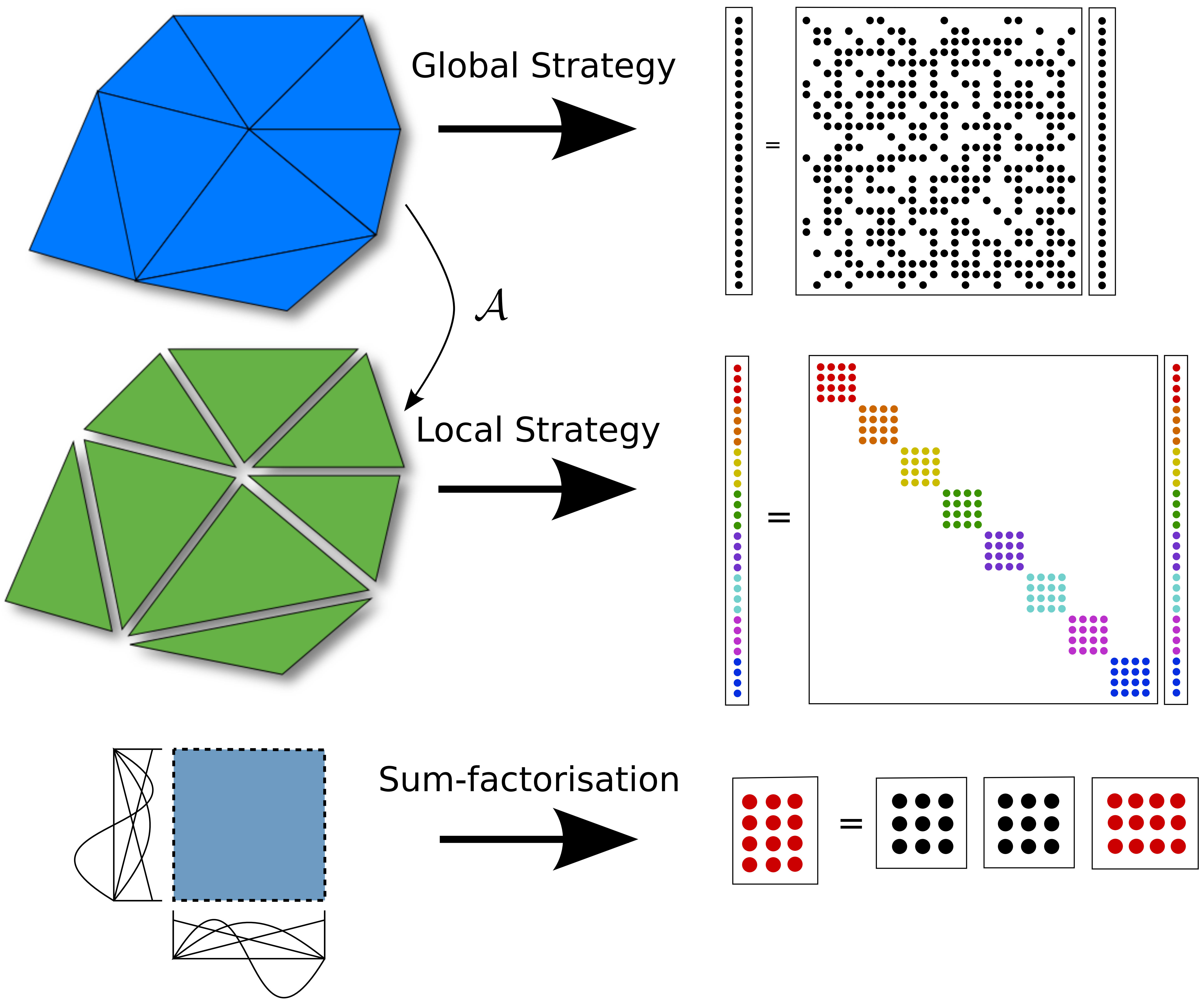

Pressure (mmHg)
150
100
50
0
0
2
4
Time (s)
Q (cm^3/s)
20
10




Mach
1.37
1.2
1
0.75
0.5
0.25
0.0351



Implementation choices

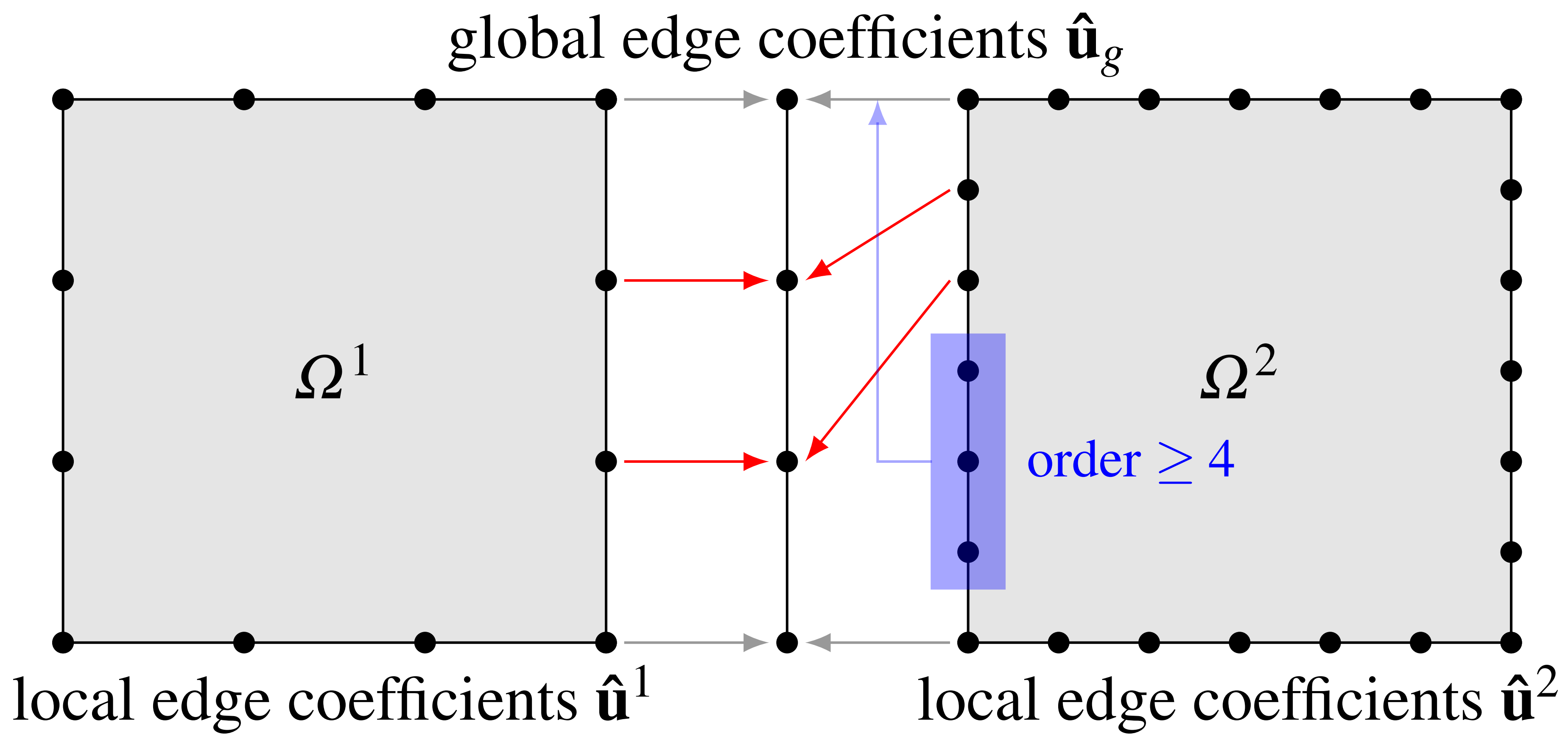


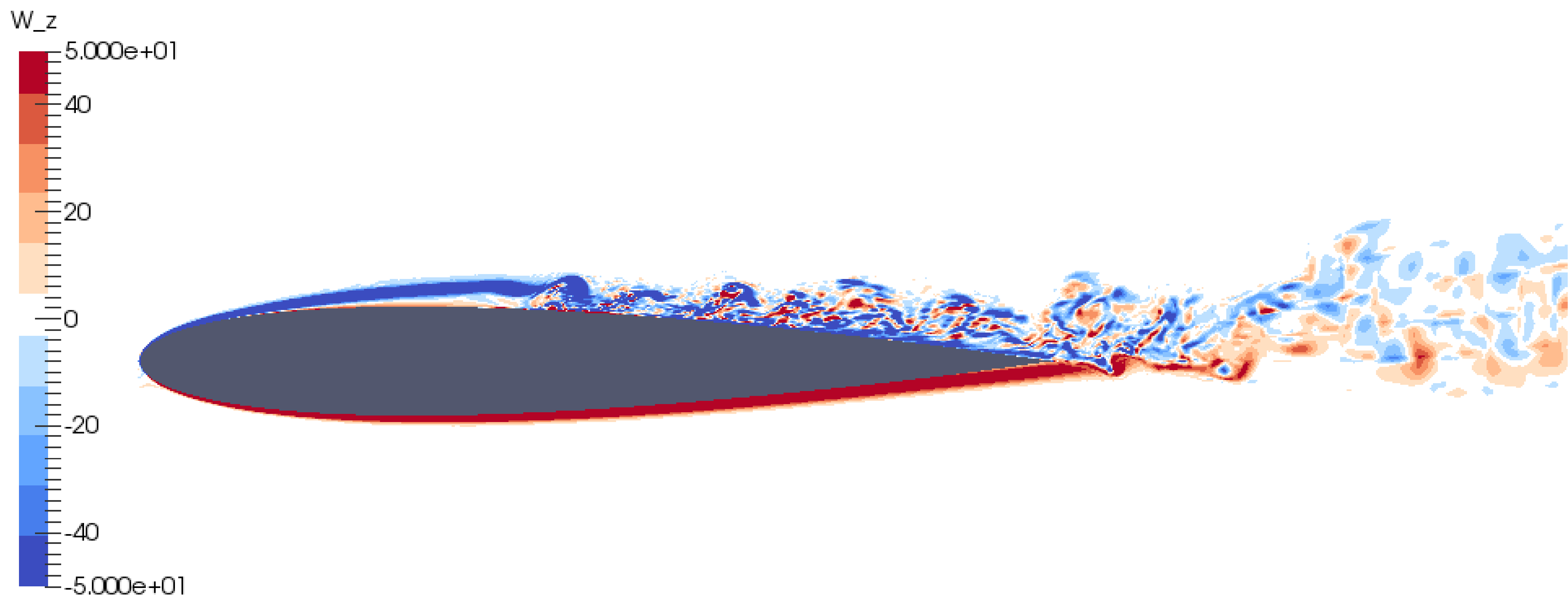
Increasing polynomial order

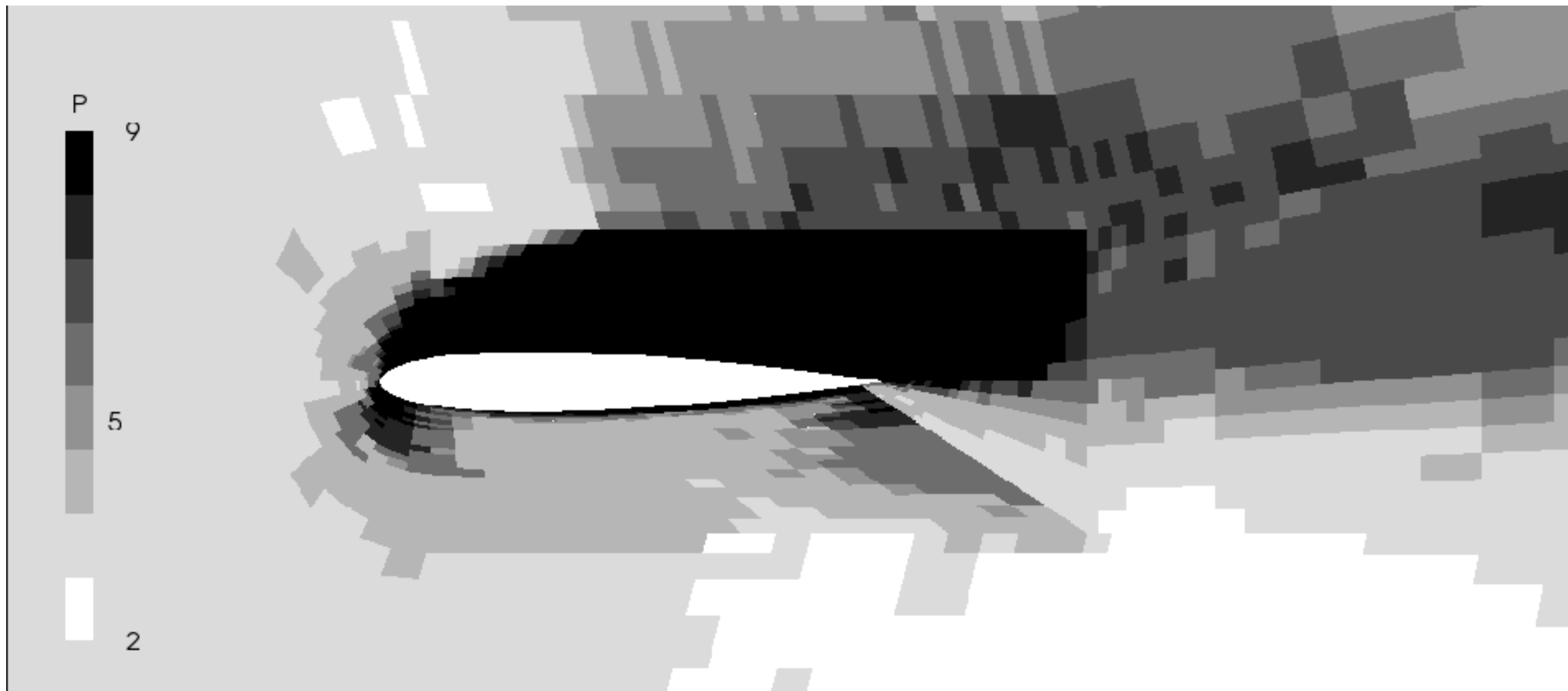
More localised memory access

h-to-*p* efficiently

- Approach performance varies wildly depending on many factors that are not *a priori* determinable
- Allow us to explore the space of flops/byte ratio
- Also important for e.g. variable-*p* simulations







Sum-factorisation

Essential for performance at high polynomial orders

$$\sum_{p=1}^{P}\sum_{q=1}^{Q}\hat{u}_{pq}\phi_p(\xi_{1i})\phi_q(\xi_{2j}) = \sum_{p=1}^{P}\phi_p(\xi_{1i})\left[\sum_{q=1}^{Q}\hat{u}_{pq}\phi_q(\xi_{2j})\right]$$

store this

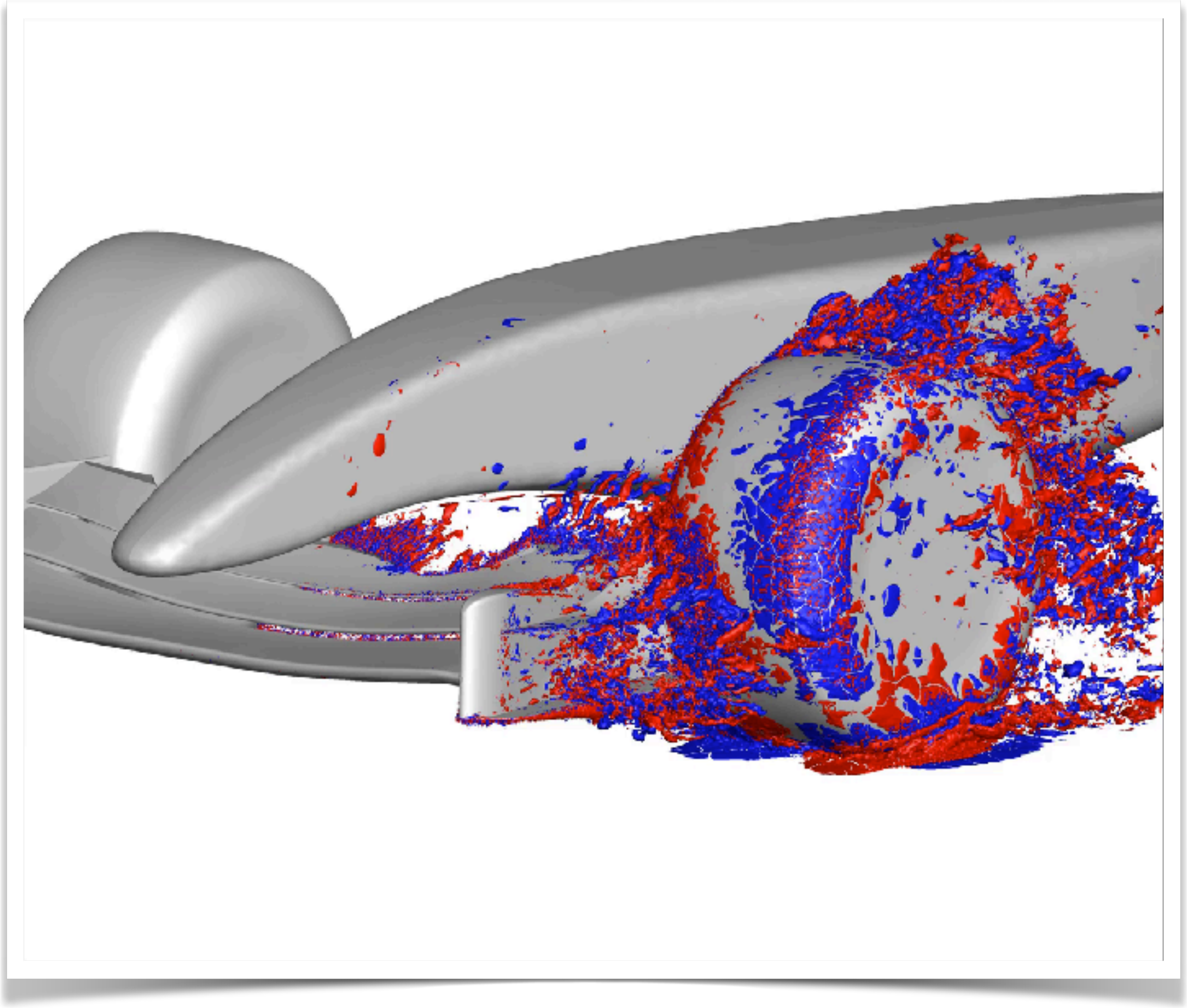
2D: $O(P^4) \rightarrow O(P^3)$ 3D: $O(P^6) \rightarrow O(P^4)$

Can still do this for tets: harder indexing

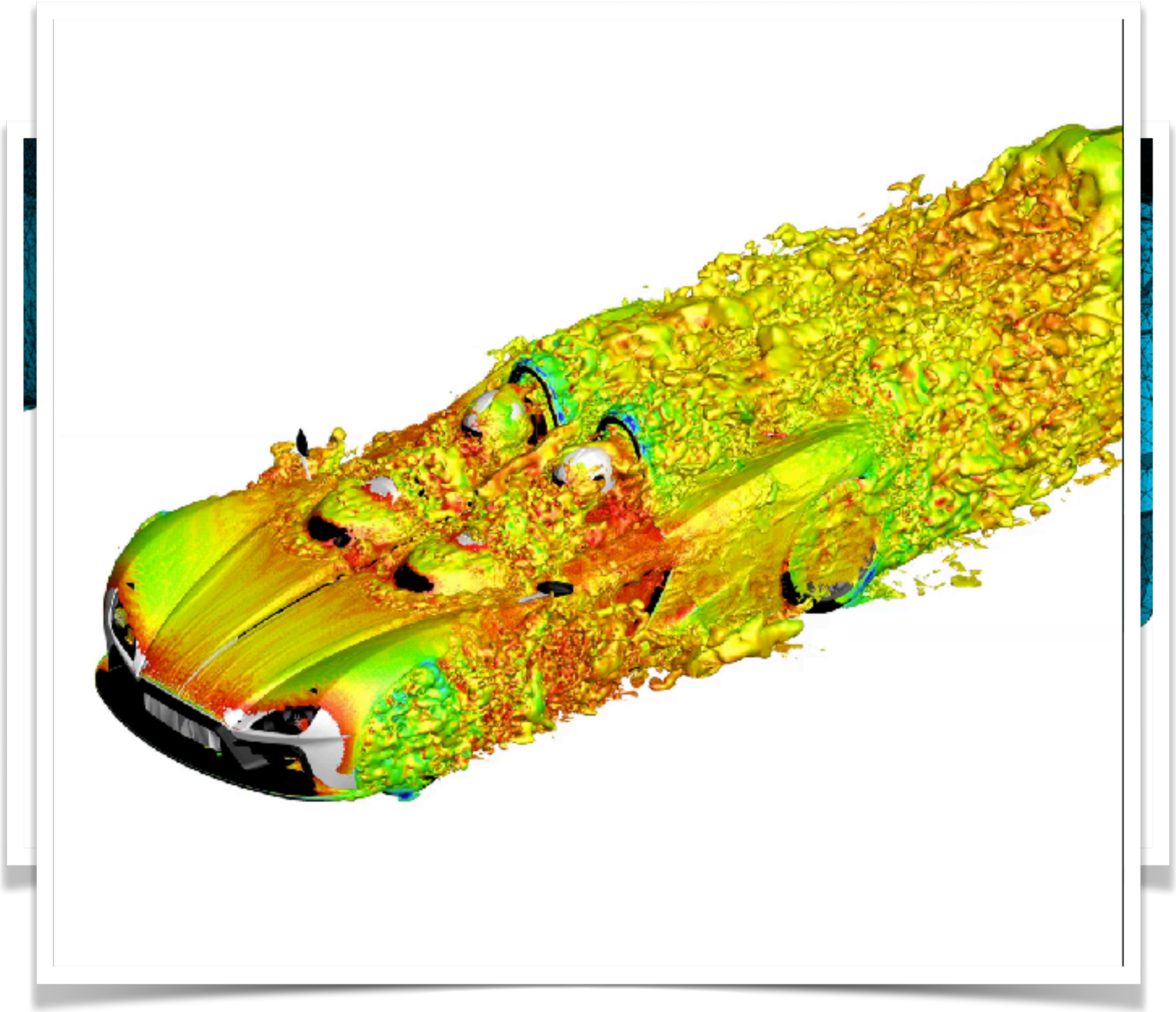
$$u(\xi_{1i},\xi_{2j},\xi_{3k}) = \sum_{p=1}^{P}\sum_{q=1}^{Q-p}\sum_{r=1}^{R-p-q}\hat{u}_{pqr}\phi^a_p(\xi_{1i})\phi^b_{pq}(\xi_{2j})\phi^c_{pqr}(\xi_{3k})$$

Unstructured simulations

Hexes yield best performance
can efficiently exploit sum factorisation



Can’t use for complex geometries
How to improve performance?



Exploiting vectorisation

- Key to achieving peak performance is exploiting vectorisation (SSE, FMA, AVX, AVX-512, …)
- Recent work has focused on achieving this with tuned kernels for key operators
- Particular focus on matrix-free approaches that avoid construction of a matrix per-element
- Try to exploit tensor-product construction of basis

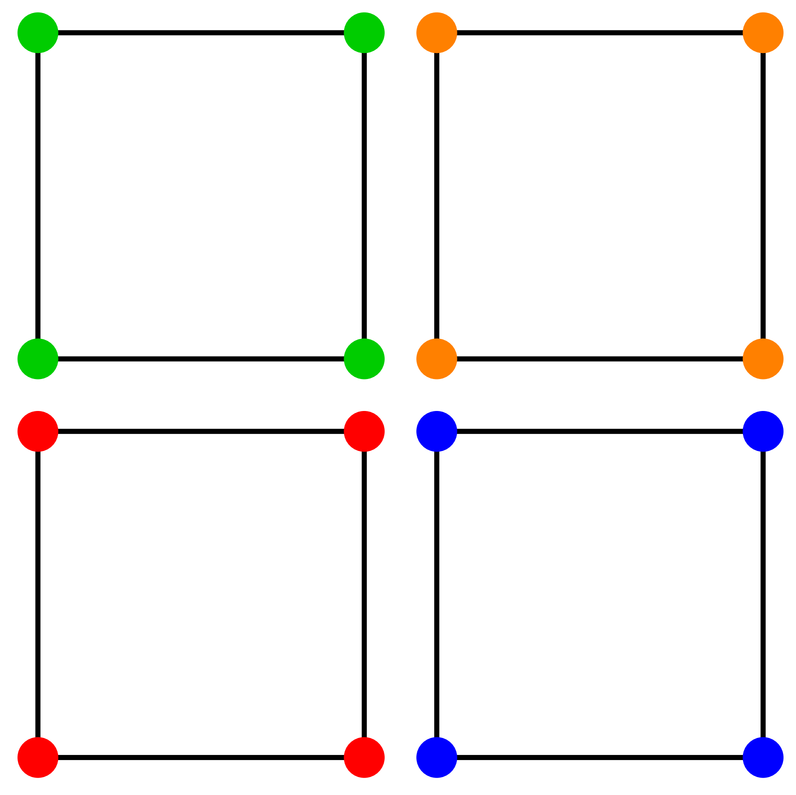
Collections

- Reformulate implementation choices into kernel operations over multiple elements
- Group geometric terms $\dfrac{\partial x_i}{\partial \xi_j}$
- Focus around key components of Laplacian:
  - ➡ Backward transformation: $u_e^\delta = \sum_p \hat{u}_p \phi_p(x)$
  - ➡ Inner product: (Φ_i, Φ_j)
  - ➡ Derivatives: $\partial u / \partial x_i$
  - ➡ Inner product w.r.t. derivative: $(\Phi_i, \nabla \Phi_j)$

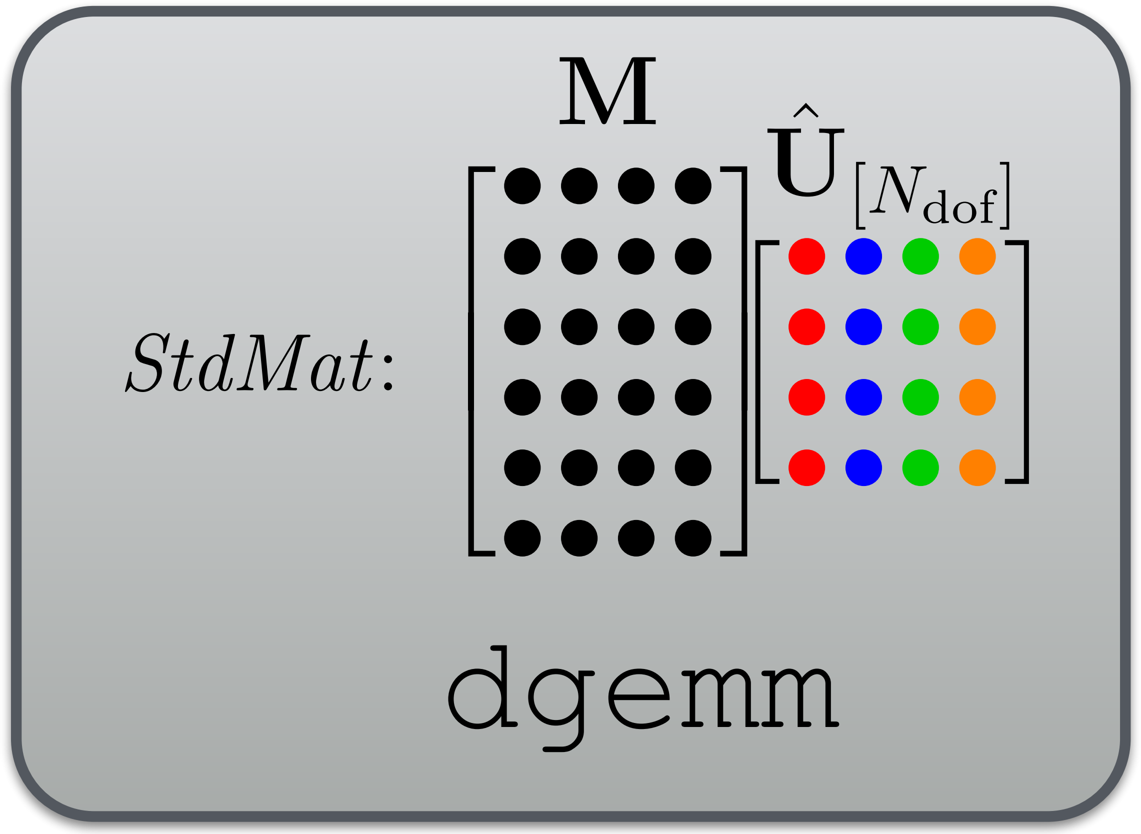
Collections

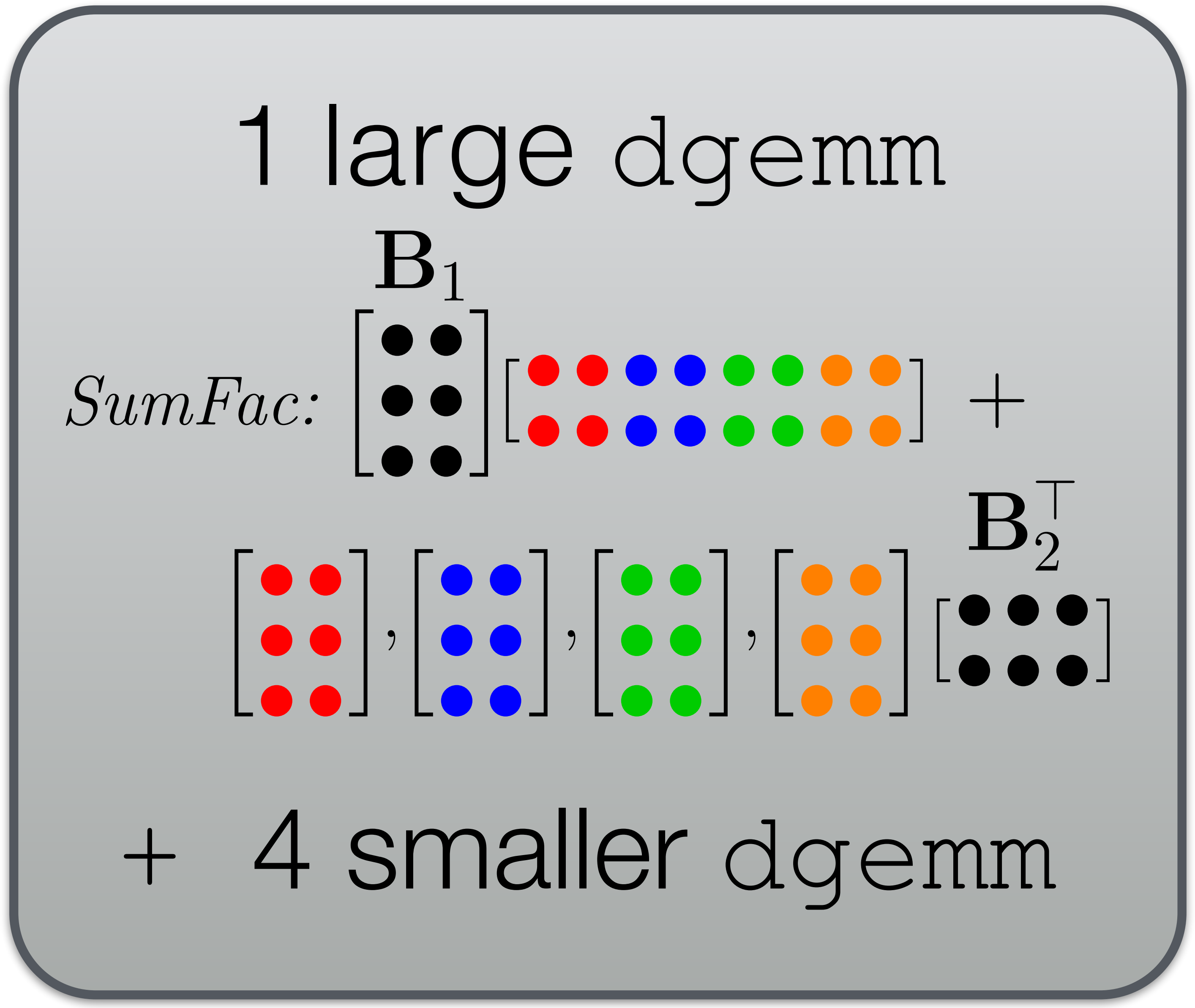
Use BLAS calls throughout

Various implementation strategies for performance across *p*



4 quad mesh





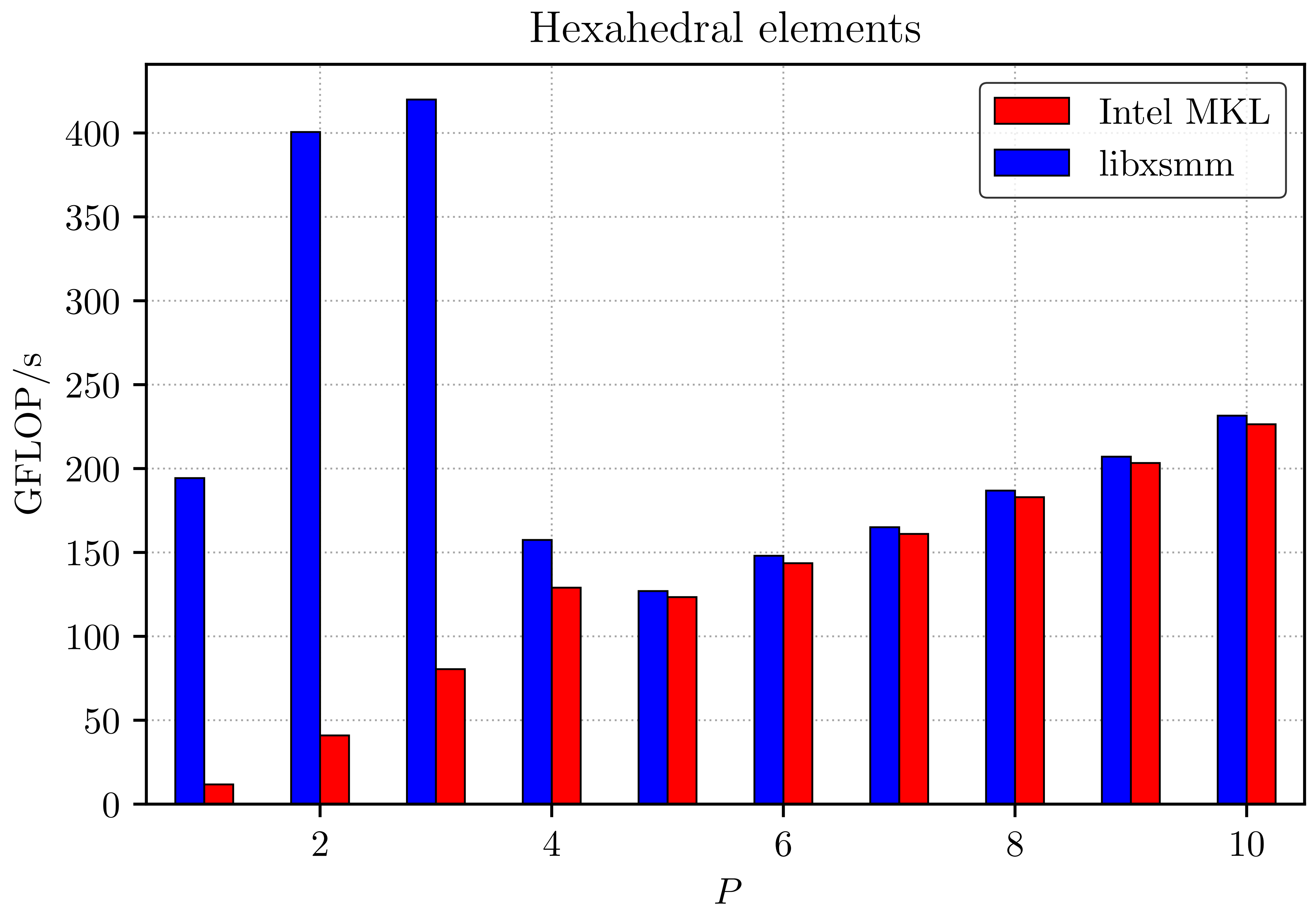
Can also do this for non-TP elements, but
data ordering harder, matrices smaller (bad for BLAS)

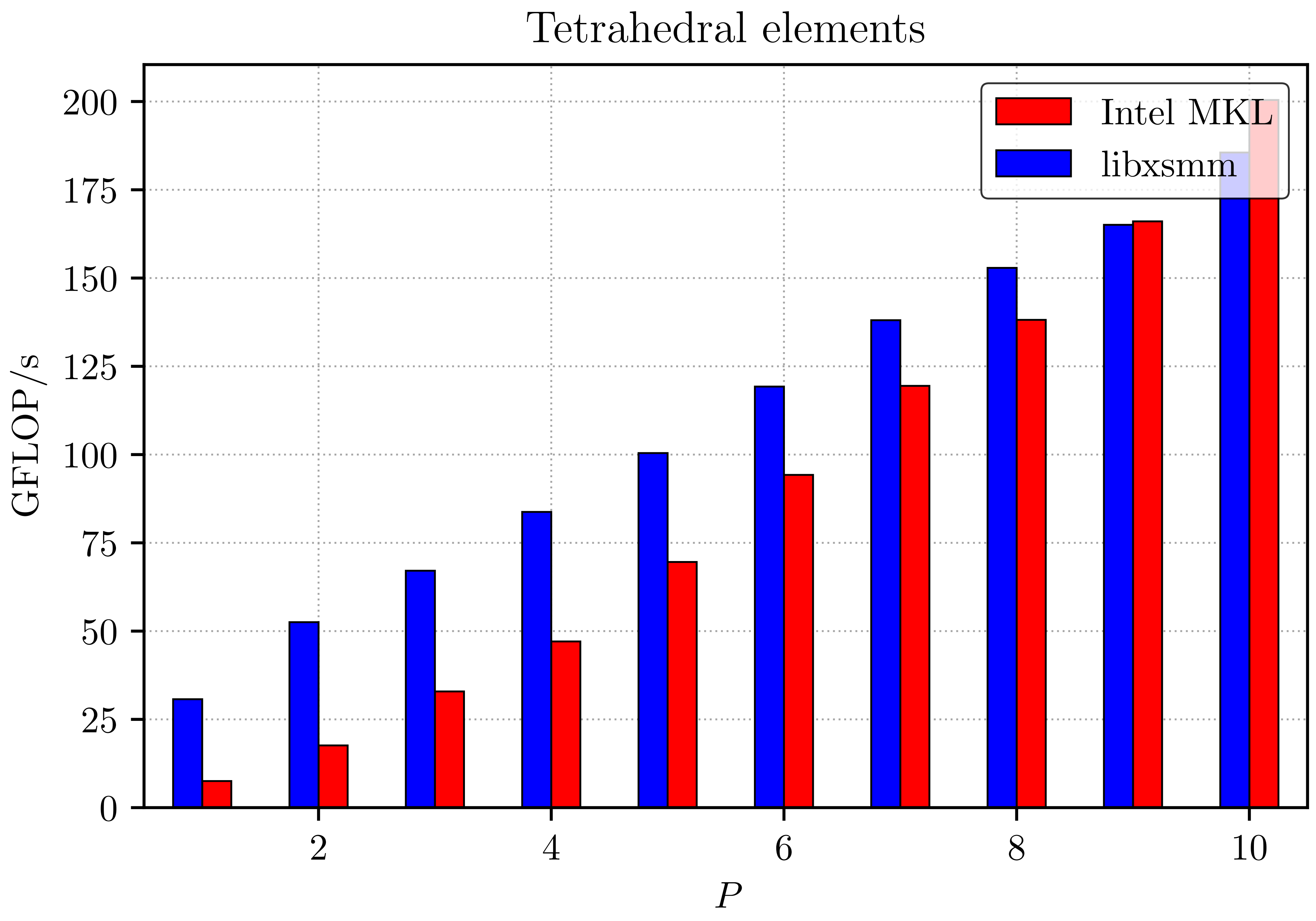
Small matrices with `libxsmm`

- Most of the matrix-matrix multiplies done in collections are small, at least in one rank
- Trialling use of `libxsmm` for small matrix multiplications
- libxsmm yields encouraging performance gains over standard MKL/BLAS, particularly for non-TP elements
- **Bottleneck:** transposes (current out-of-place)

➡ Appears to be very challenging for non-tensor product elements

libxsmm vs Intel MKL





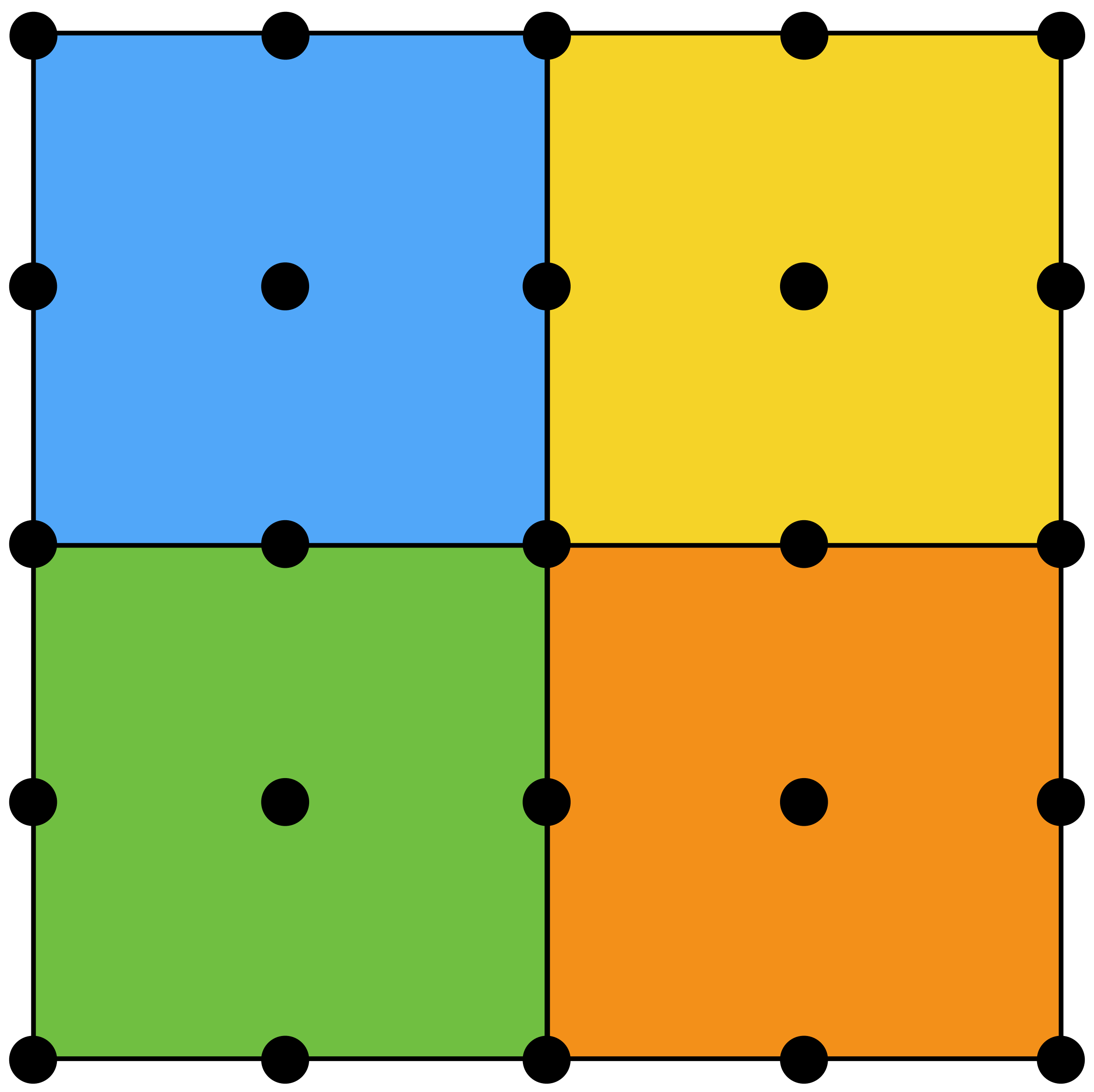
2 x Intel E5-2670v4
~1.2 TFLOP/s peak

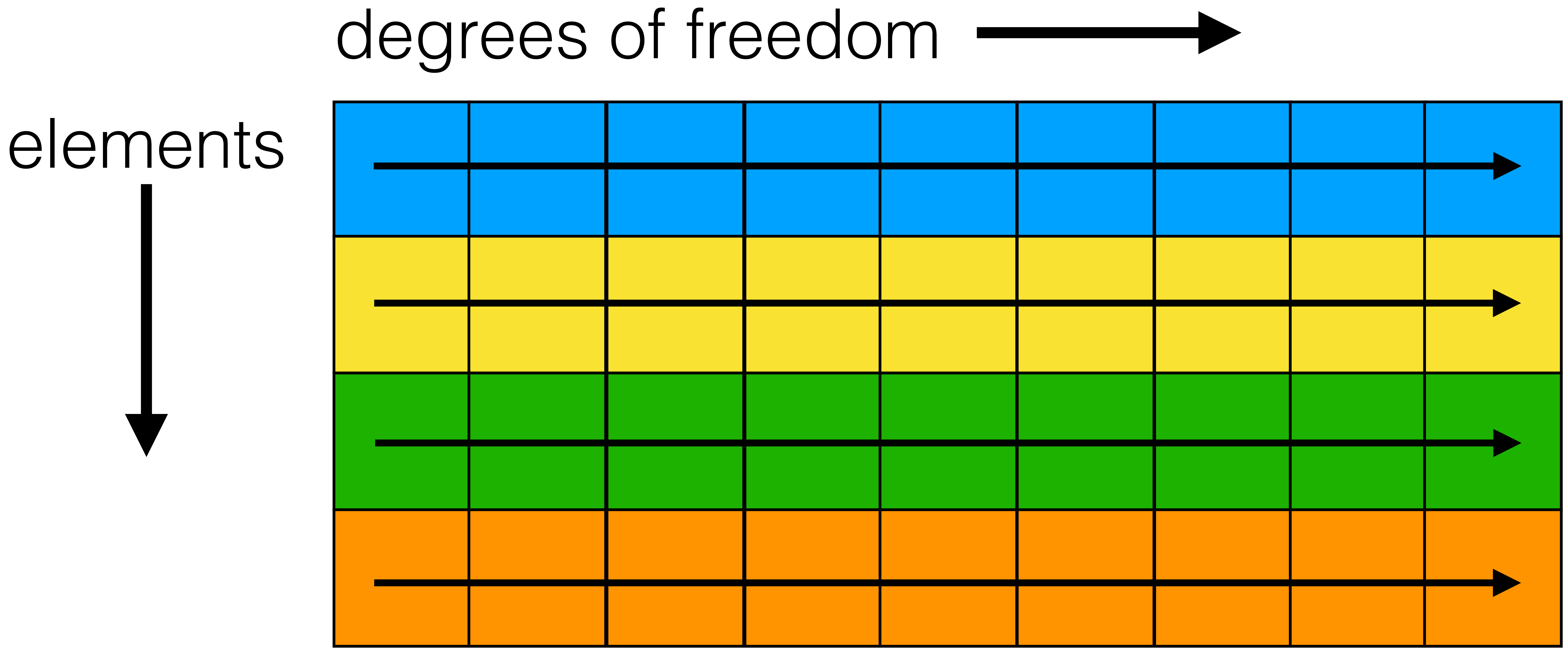
Good performance gains
at low/moderate orders

Anything else we can do?

Data layout

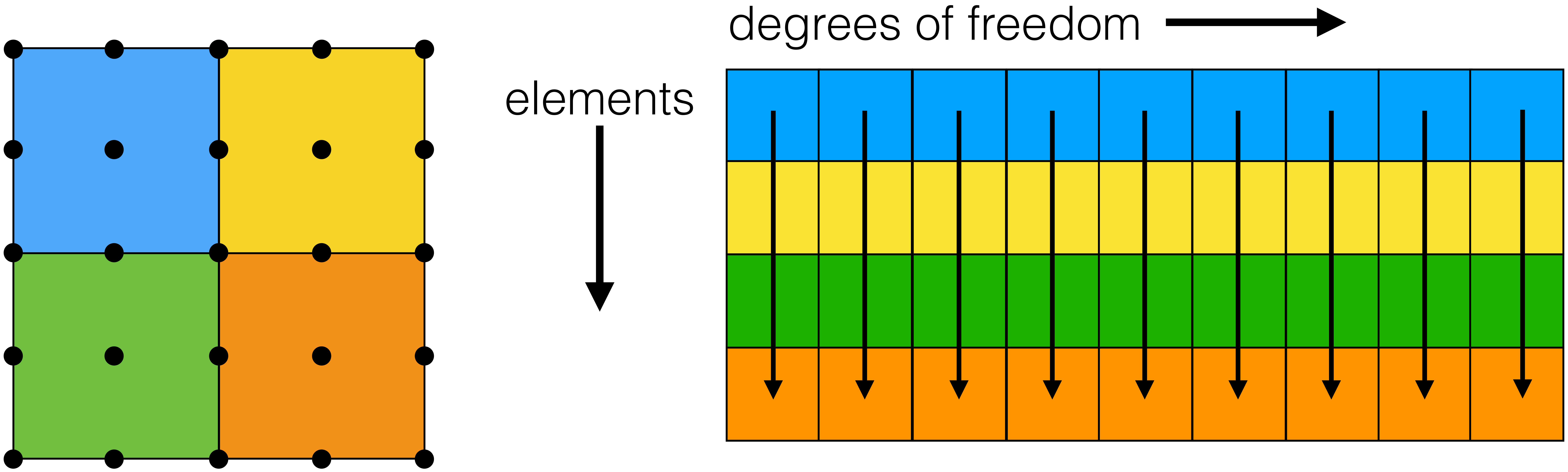
Natural to consider data laid out element by element





Data layout

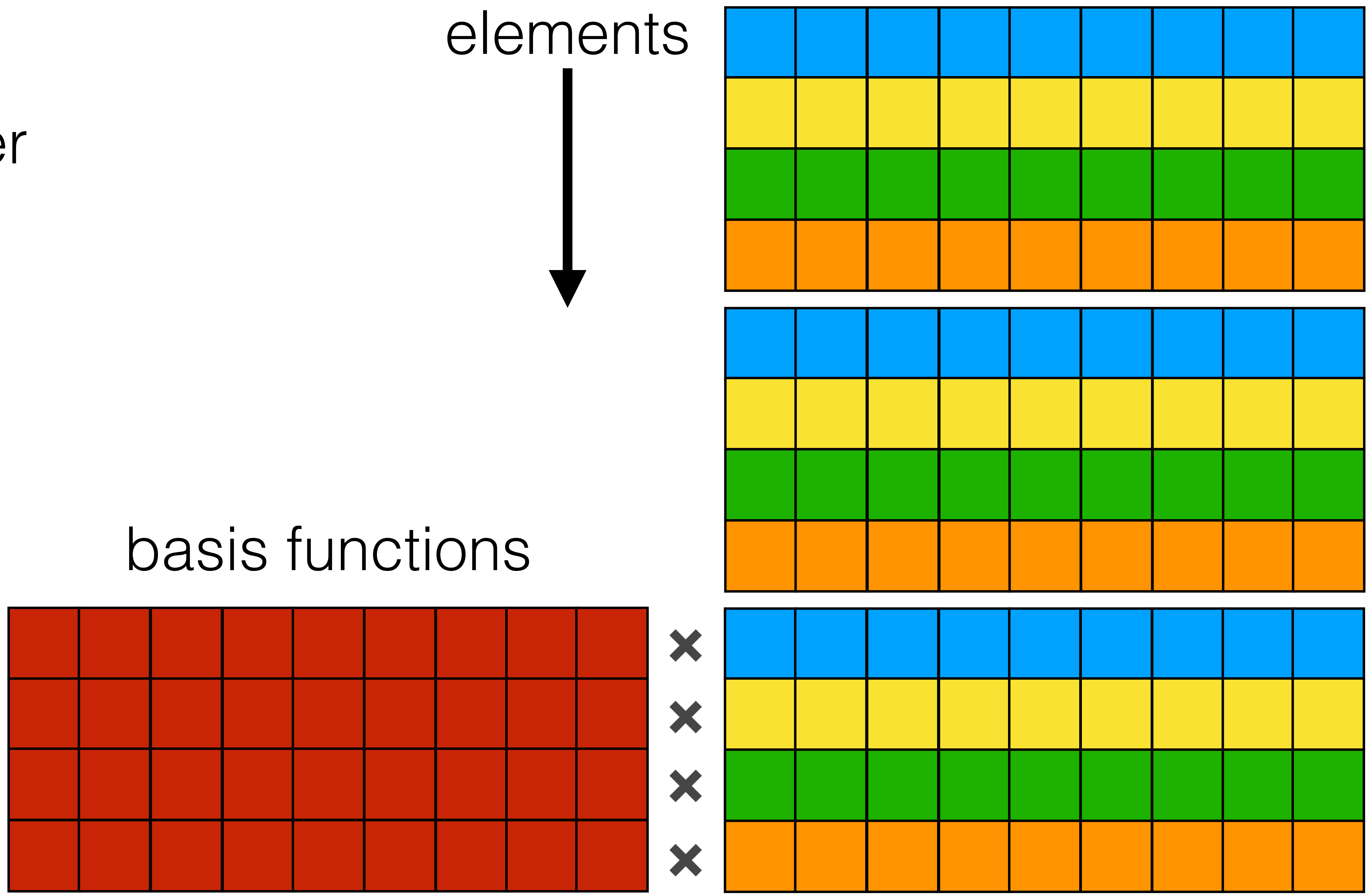
May be able to exploit vectorisation by grouping DoFs by vector width



Data layout

Operations then occur over groups of elements of size of vector width

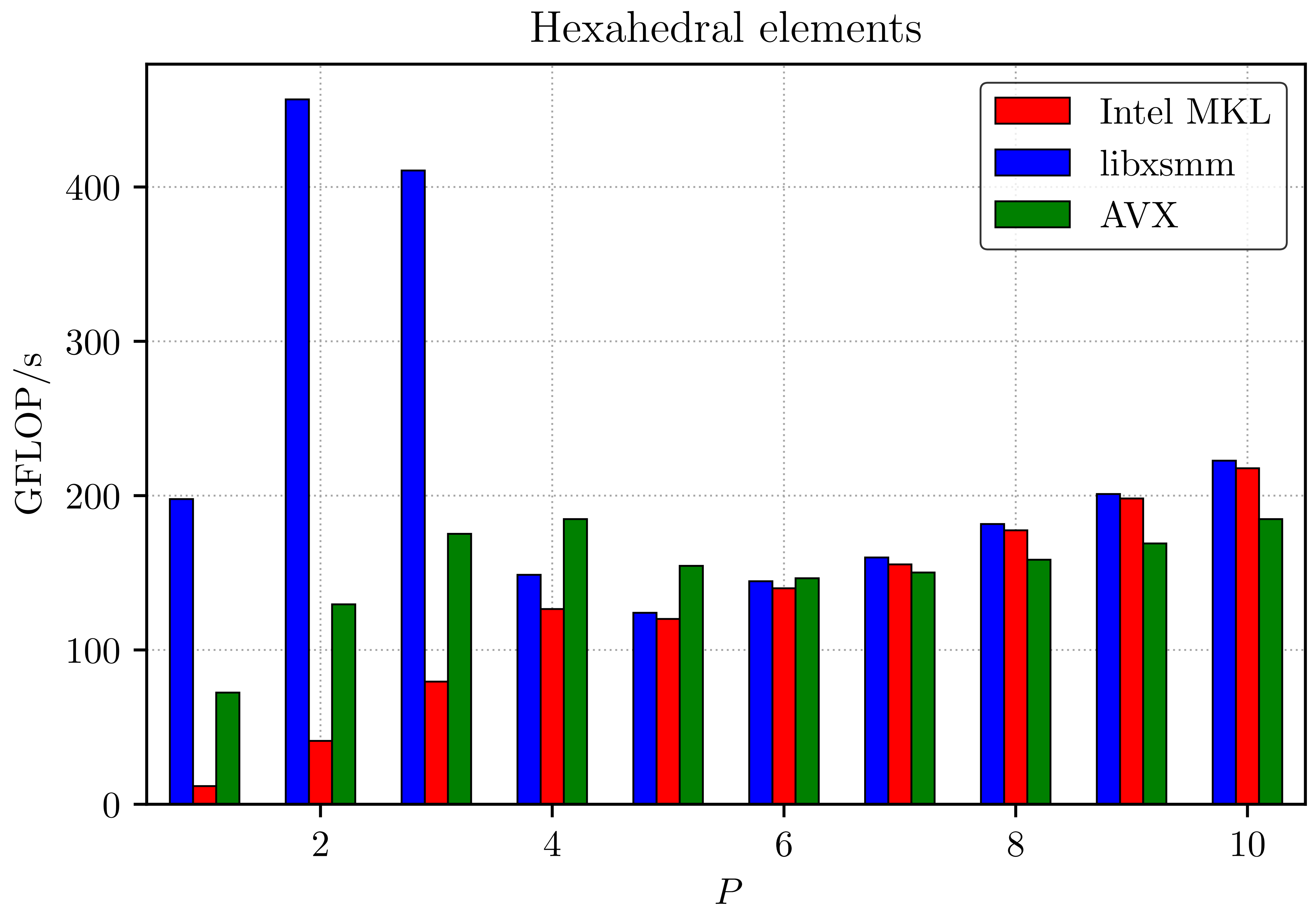
Possible downside: requires duplicating basis data for each vector lane



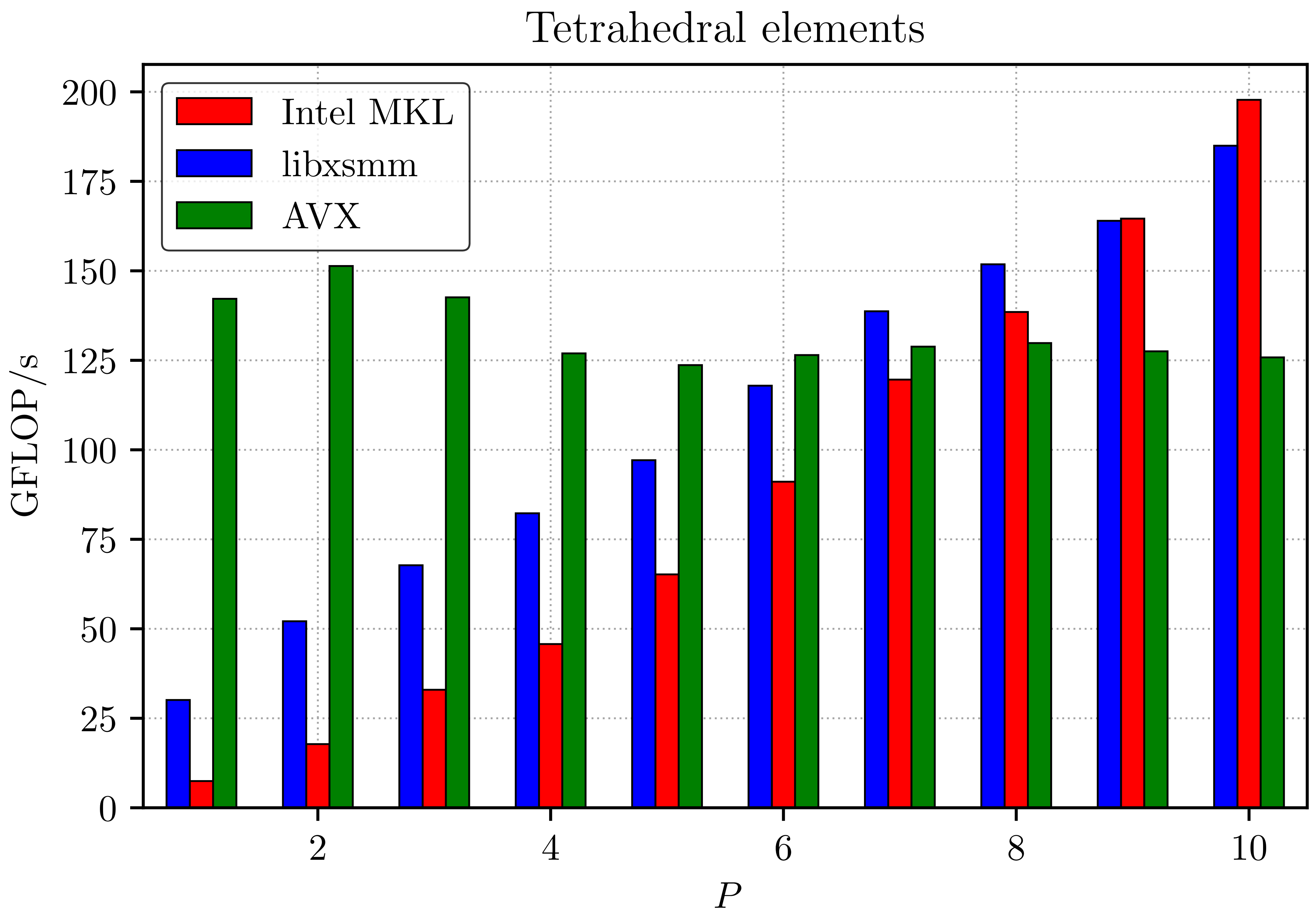
Implementation details

- Benchmark against `libxsmm` and MKL
- Hand-written loops for sum-factorisation
- Explicit intrinsics for vector operations
  - AVX: 4 double multiplications/cycle
  - combined with FMA
  - non-temporal stores where appropriate

Performance



Fairly mediocre hex performance

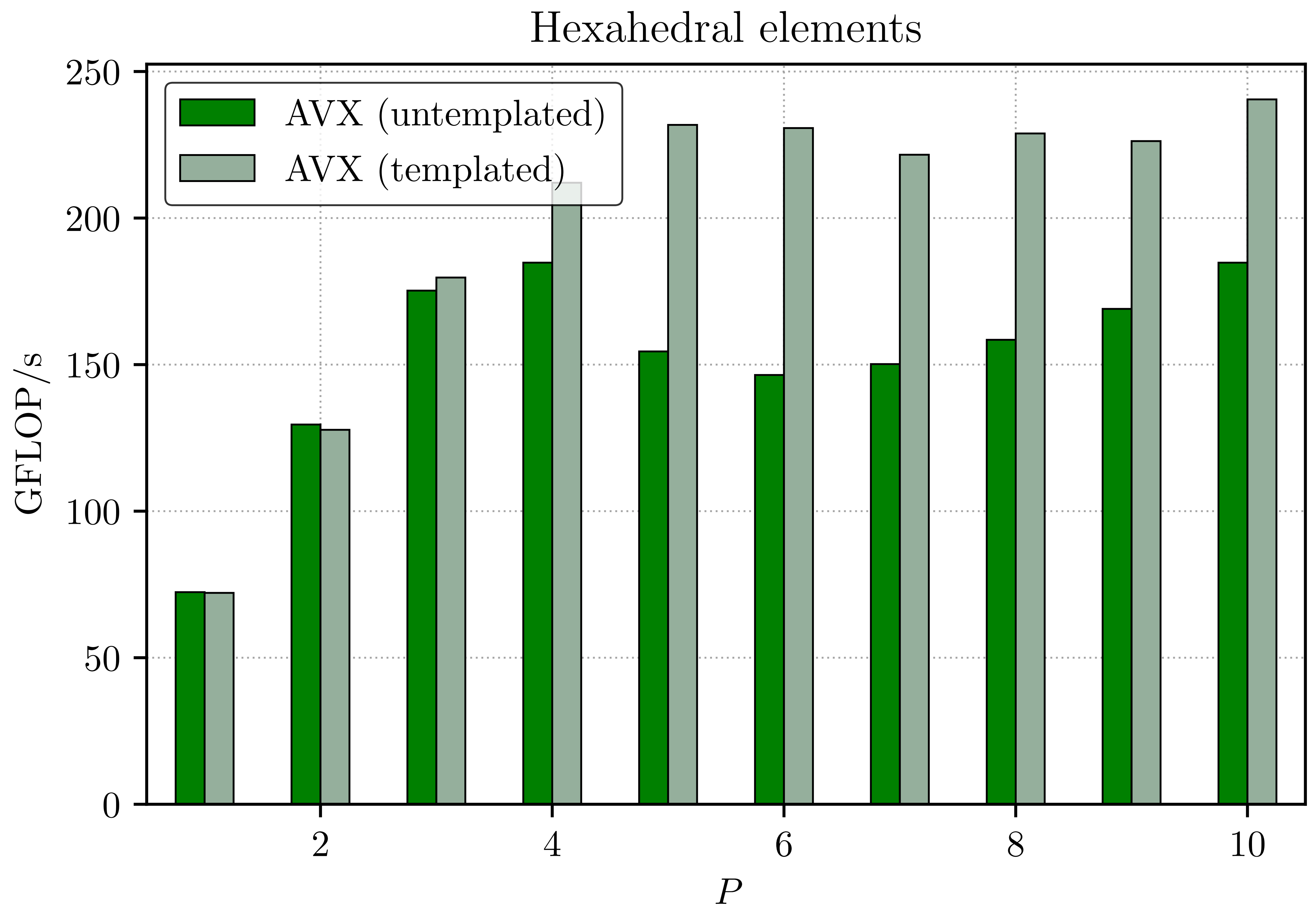


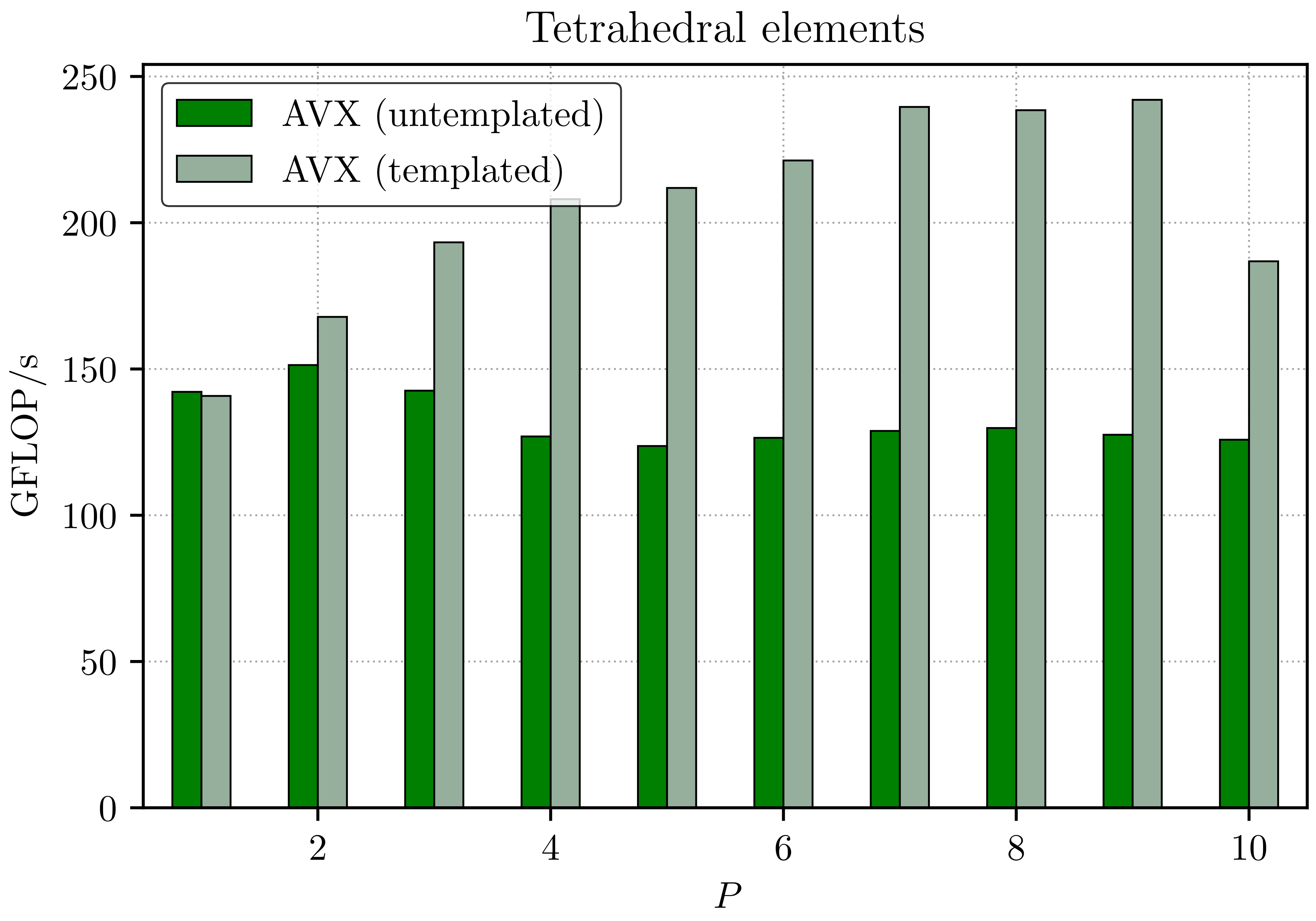
Pretty good for tets at low-order

Indexing

- Indexing for tets is a bit complex
- Could we do better by giving compiler more information for unrolling loops?
- Might also help hex performance
- Use some C++ templating

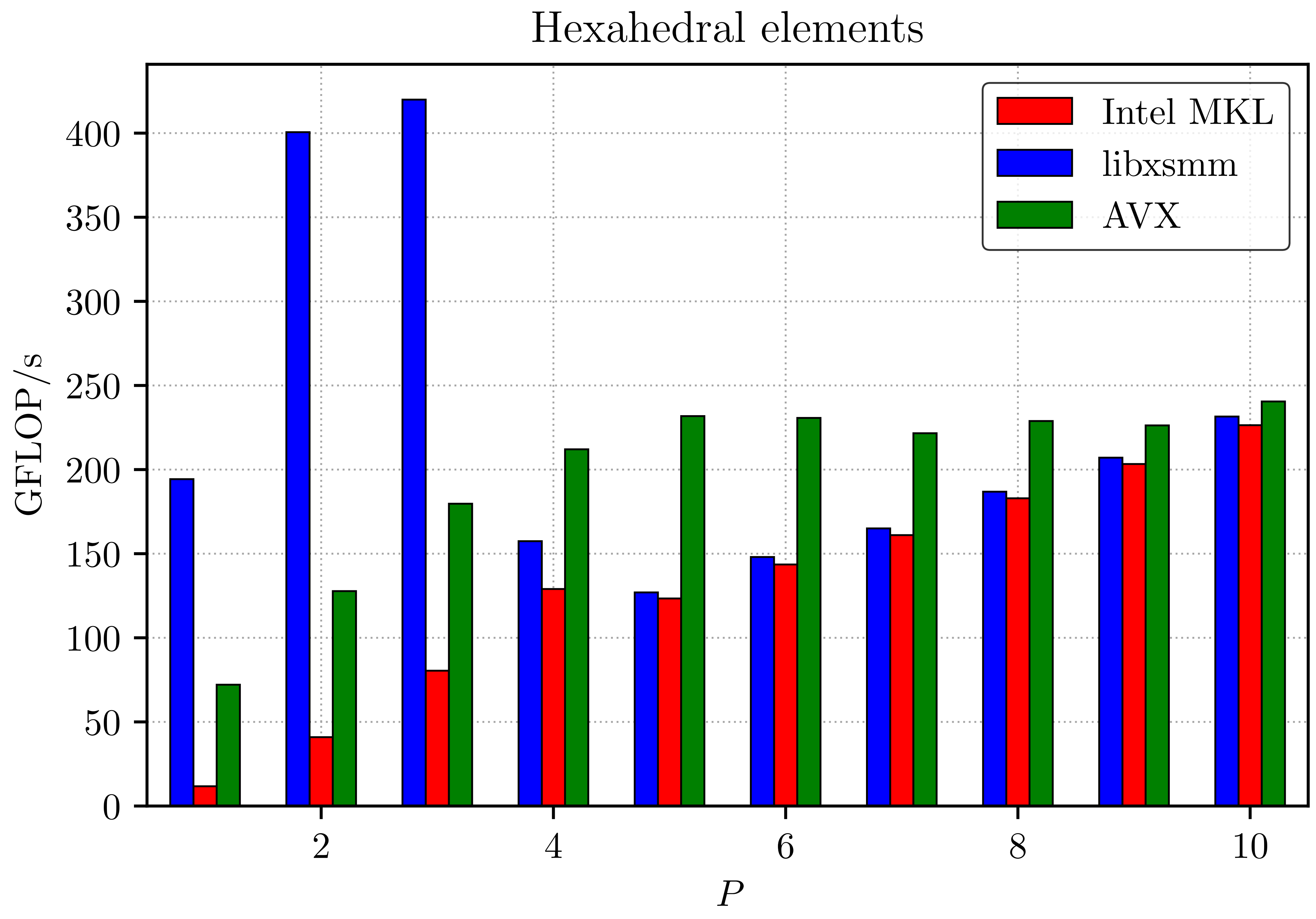
Templated performance

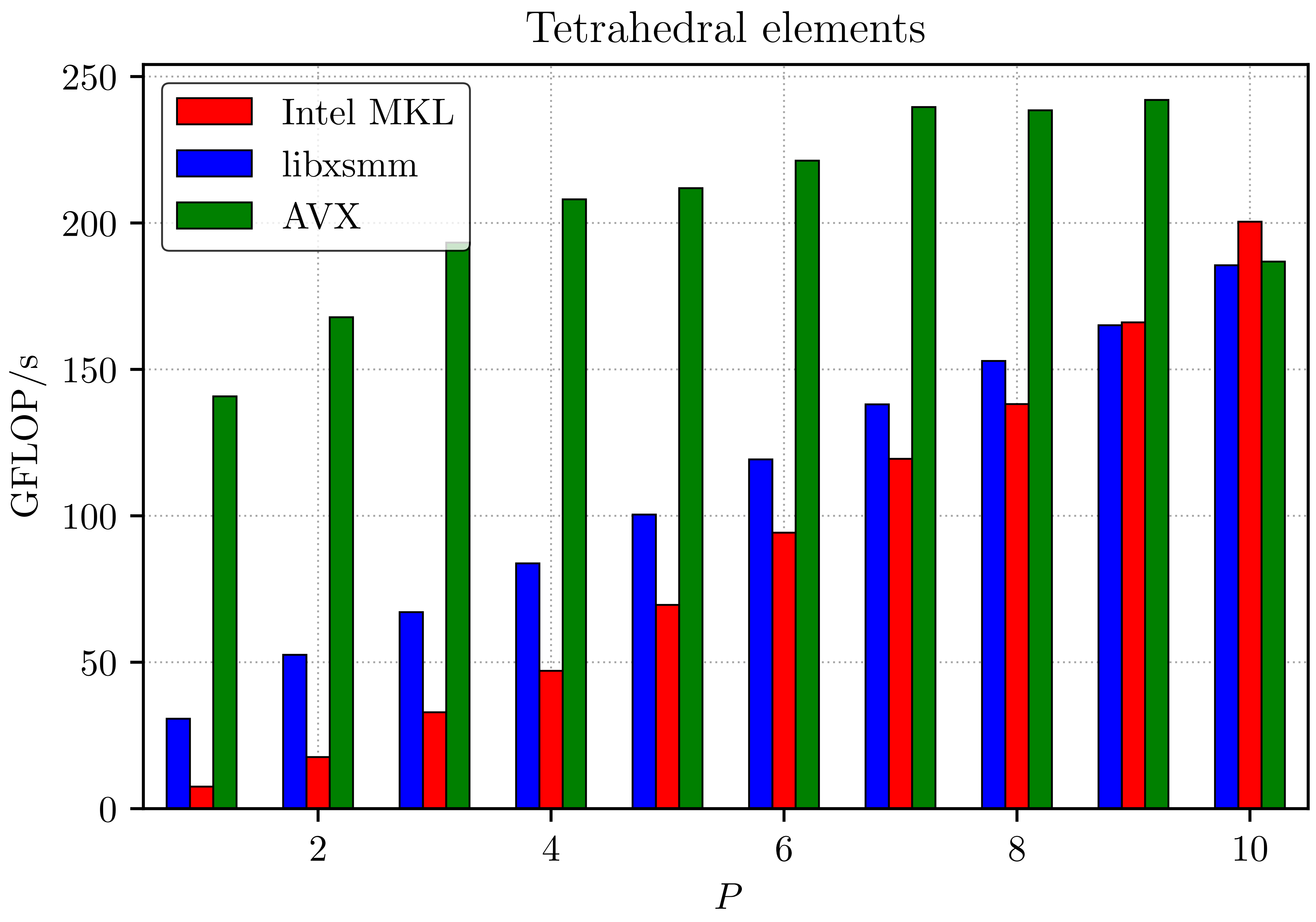




Templating gives good performance gains across the board

Final comparison

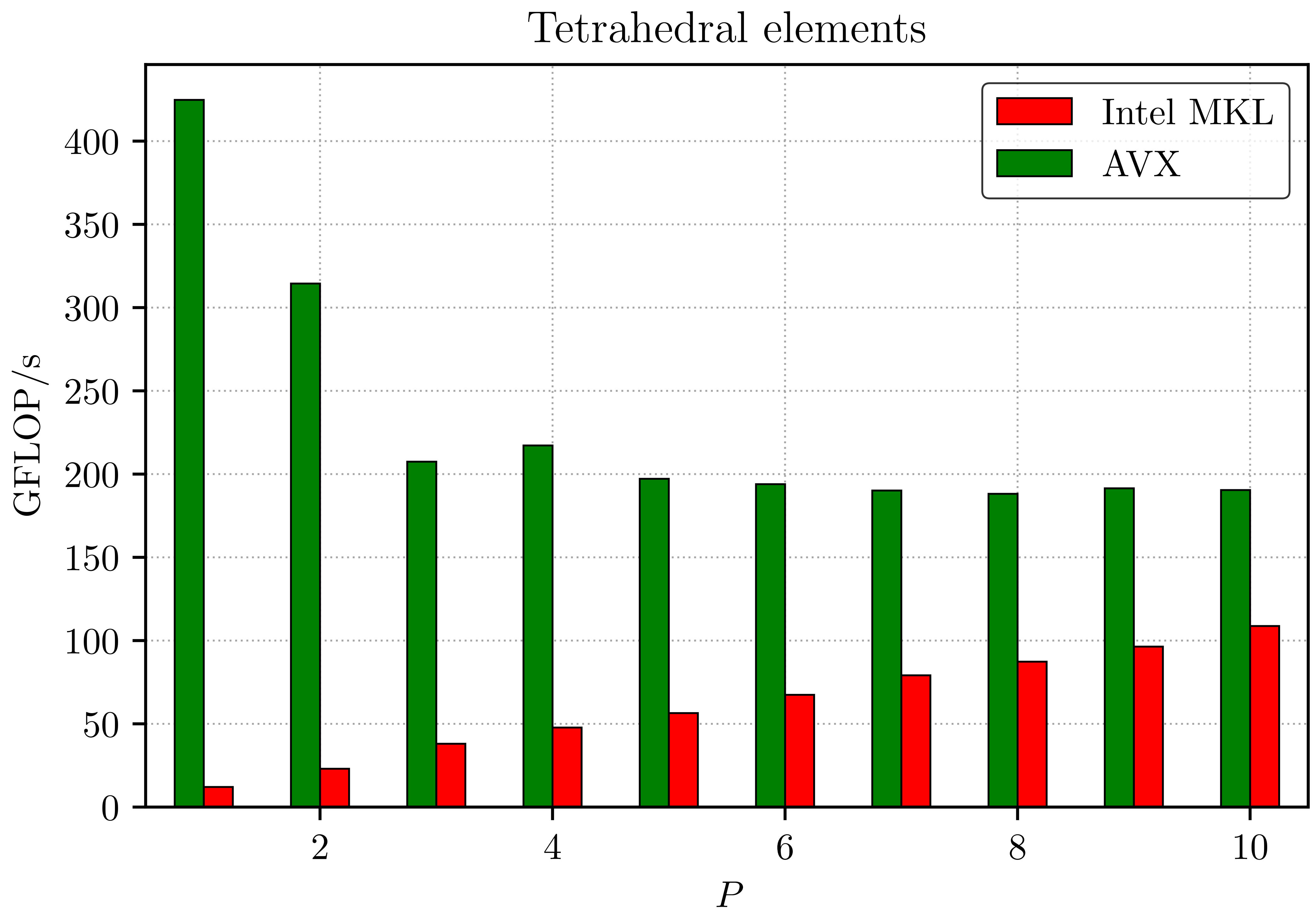




Templating gives good performance gains across the board

Other operators

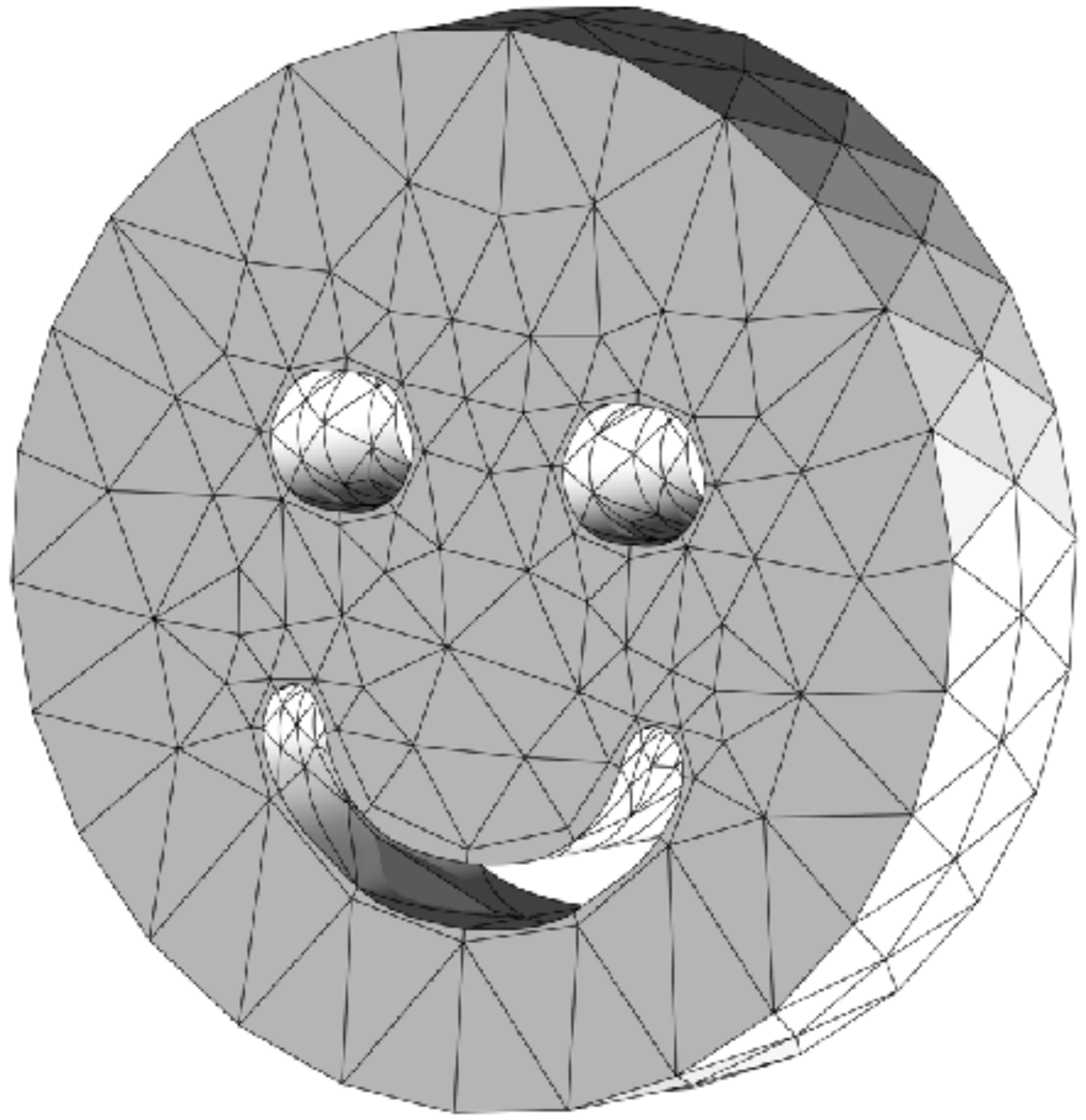
- Inner product requires more work:
  - multiply by elemental Jacobian
  - multiply by quadrature weights
  - avoid storage of premultiplied quadrature weights
- Initial results seem fairly promising



Summary

- Need to think very hard about data layout to properly exploit underlying hardware
- Efficient use of data layout can significantly improve performance, at least for operators considered here
- A work in progress!
  - Full Helmholtz operator
  - Wider vector lanes (AVX-512)

Thanks for listening!



ExaFLOW

https://davidmoxey.uk/ @davidmoxey

d.moxey@exeter.ac.uk

www.nektar.info

Nektar++ high-order framework

Framework for spectral/hp element method:

- Dimension independent and supports various discretisations (CG/DG/HDG)
- Mixed elements (quads/tris, hexes, prisms, tets, pyramids) using hierarchical modal and classical nodal formulations
- Solvers for (in)compressible Navier-Stokes, advection-diffusion-reaction, shallow water equations, …
- Parallelised with MPI, tested scaling up to ~10k cores

http://www.nektar.info/   nektar-users@imperial.ac.uk   https://gitlab.nektar.info/