# Vectorisation for high-order simplicial elements

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

Roman Amici, Robert M. (Mike) Kirby
Scientific Computing and Imaging Institute, University of Utah

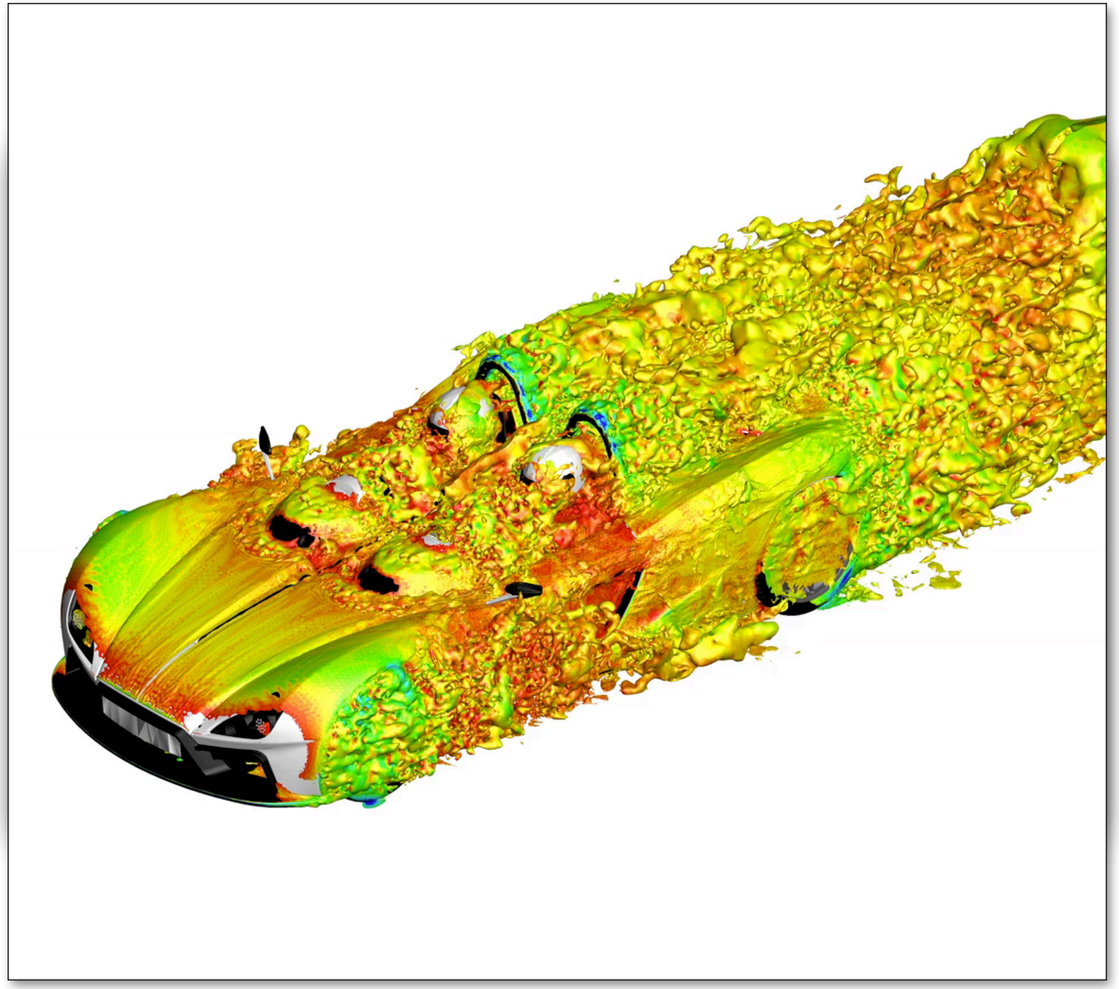
SIAM Conference on Computational Science & Engineering,
Spokane, WA, USA

27th February 2019

# Motivation

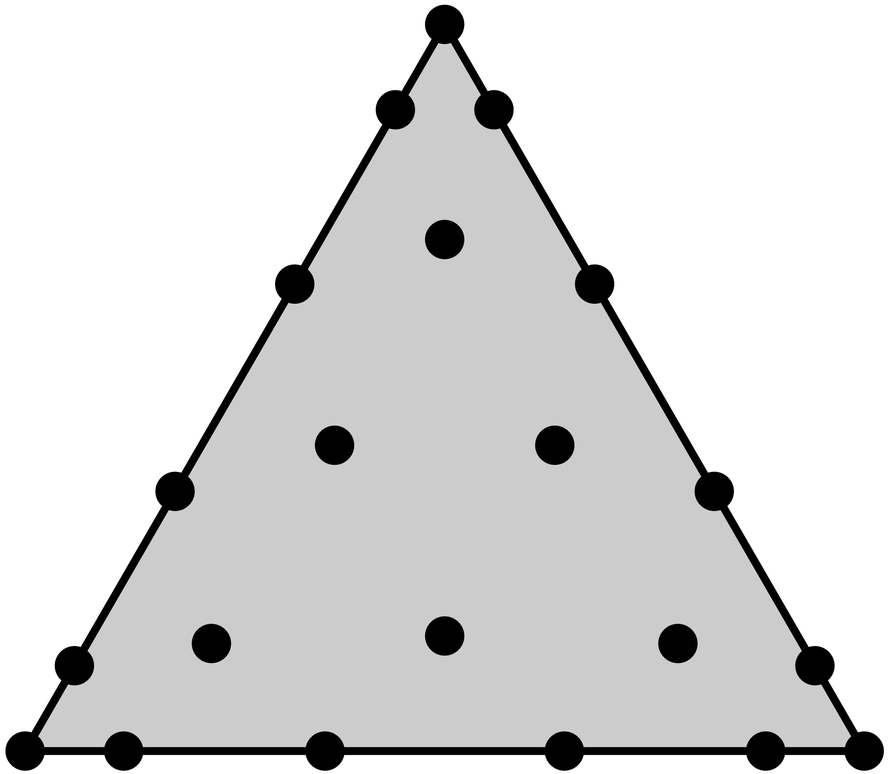
- High-order methods potentially map well onto current/future hardware challenges in three key ways.
  - Use of **matrix-free** formulations of operators to reduce memory bandwidth requirements.
  - Use of **sum-factorisation** to reduce operator count, which relies on the use of tensor-product basis.
  - Can effectively used **SIMD instructions** to achieve very high performance.

# Unstructured simulations



- Common knowledge: hex/quad elements yield best performance.
- However highly complex geometries presently require unstructured meshes.
- How to improve performance?
- Possible answer: tensor-product basis on unstructured elements: enable matrix-free operators.

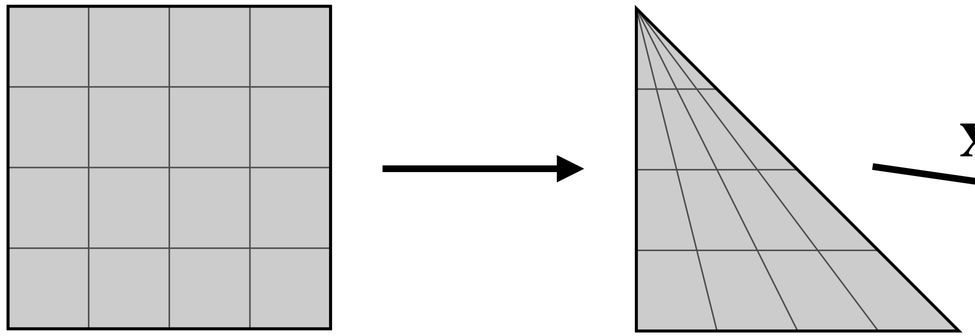
# Unstructured elements



P5 triangle, Fekete points

- Typically unstructured elements make use of Lagrange basis functions (although not always).
- Combine this with a suitable set of quadrature (cubature) points.
- However this loses tensor-products structure: i.e. no sum factorisation possible.

# Spectral/*hp* element formulation

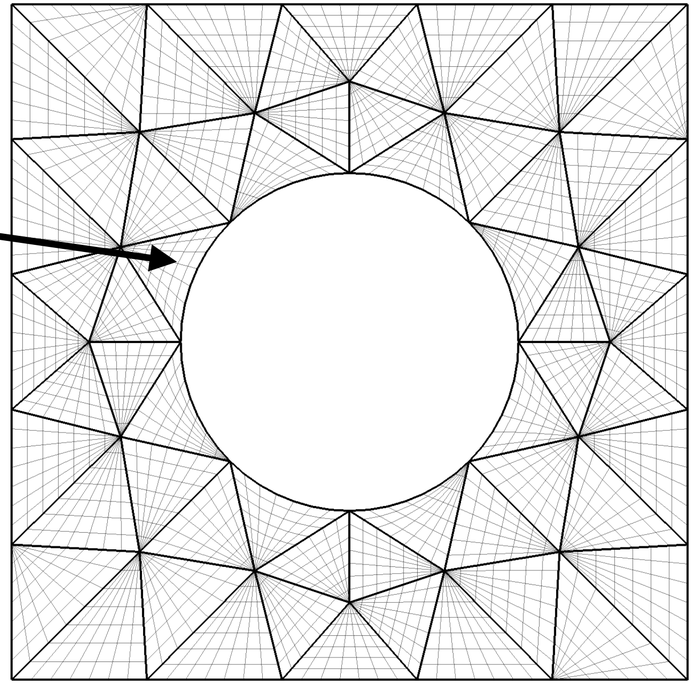


collapsed coordinates
$(\eta_1, \eta_2) \in [-1,1]^2$

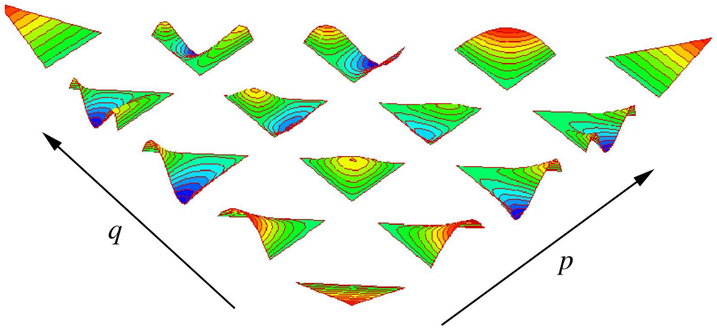


reference element
$(\xi_1, \xi_2) \in \Omega_{st}$

$\mathbf{x} = \chi^e(\xi)$



(C⁰) tensor product basis
$\phi_p^a(\eta_1)\,\phi_{pq}^b(\eta_2)$



# "Defining" features

Generally *not* collocated

$$u(\xi_{1i}, \xi_{2j}) = \sum_{n=0}^{P^2} \hat{u}_n \phi_n(\xi) = \sum_{p=0}^{P} \sum_{q=0}^{Q} \hat{u}_{pq} \phi_p(\xi_{1i}) \phi_q(\xi_{2j})$$

quadrature points

modal coefficients

Uses tensor products of 1D basis functions, even for non-tensor product shapes

$$u(\xi_{1i}, \xi_{2j}, \xi_{3j}) = \sum_{p=0}^{P} \sum_{q=0}^{Q-p} \sum_{r=0}^{R-p-q} \hat{u}_{pqr} \phi_p^a(\xi_{1i}) \phi_{pq}^b(\xi_{2j}) \phi_{pqr}^c(\xi_{3k})$$

basis function indexing harder

# Sum-factorisation

Key to performance at high polynomial orders: complexity $O(P^{2d})$ to $O(P^{2d-2})$

$$\sum_{p=0}^{P}\sum_{q=0}^{Q}\hat{u}_{pq}\phi_p(\xi_{1i})\phi_q(\xi_{2j}) = \sum_{p=0}^{P}\phi_p(\xi_{1i})\left[\sum_{q=0}^{Q}\hat{u}_{pq}\phi_q(\xi_{2j})\right]$$

**store this**

This works in essentially the same way for more complex indexing:

$$\sum_{p=0}^{P}\sum_{q=0}^{Q-p}\hat{u}_{pq}\phi_p^a(\xi_{1i})\phi_{pq}^b(\xi_{2j}) = \sum_{p=0}^{P}\phi_p^a(\xi_{1i})\left[\sum_{q=0}^{Q-p}\hat{u}_{pq}\phi_{pq}^b(\xi_{2j})\right]$$

**store this**

# Key questions

- Under spectral/*hp* approach, sum-factorisation matrix-free operators are certainly possible for any element type.
- However, **how much performance** do we lose relative to hex/quad?
- How should **SIMD** be used?
- Developed a benchmarking utility for Helmholtz operator to test viability of this approach.

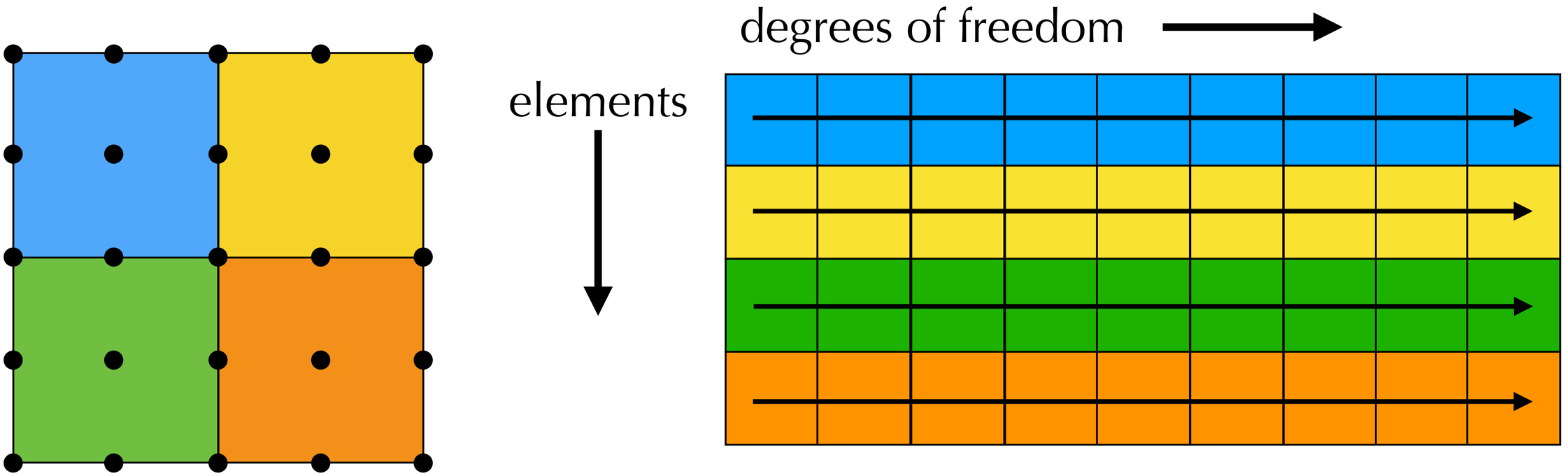
$$\nabla^2 u - \lambda u = f(x) \quad \rightarrow \quad (\mathbf{L} + \lambda \mathbf{M})\hat{\mathbf{u}} = \hat{\mathbf{f}}$$

# Implementation particulars

- Hand-written kernels for each element type to implement three key components: **interpolation**, **derivatives** and **inner products**.
- Written using C++: templating on (maybe heterogeneous) polynomial order and quadrature order.
- Also templates on **affine elements** (spatially-constant Jacobian) vs. **curvilinear** (spatially-varying); no consideration for Cartesian meshes.
- Templating gives *significant* improvements in runtime performance, particularly for complex loop structures found in this regime.

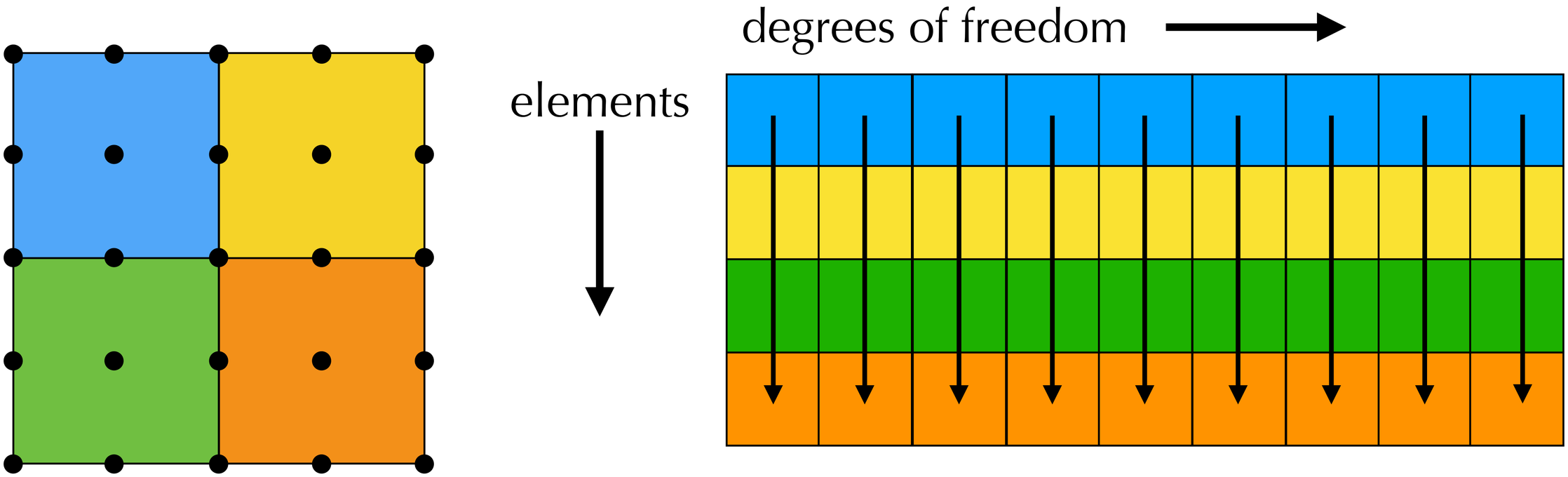
# Data layout

Natural to consider data laid out element by element



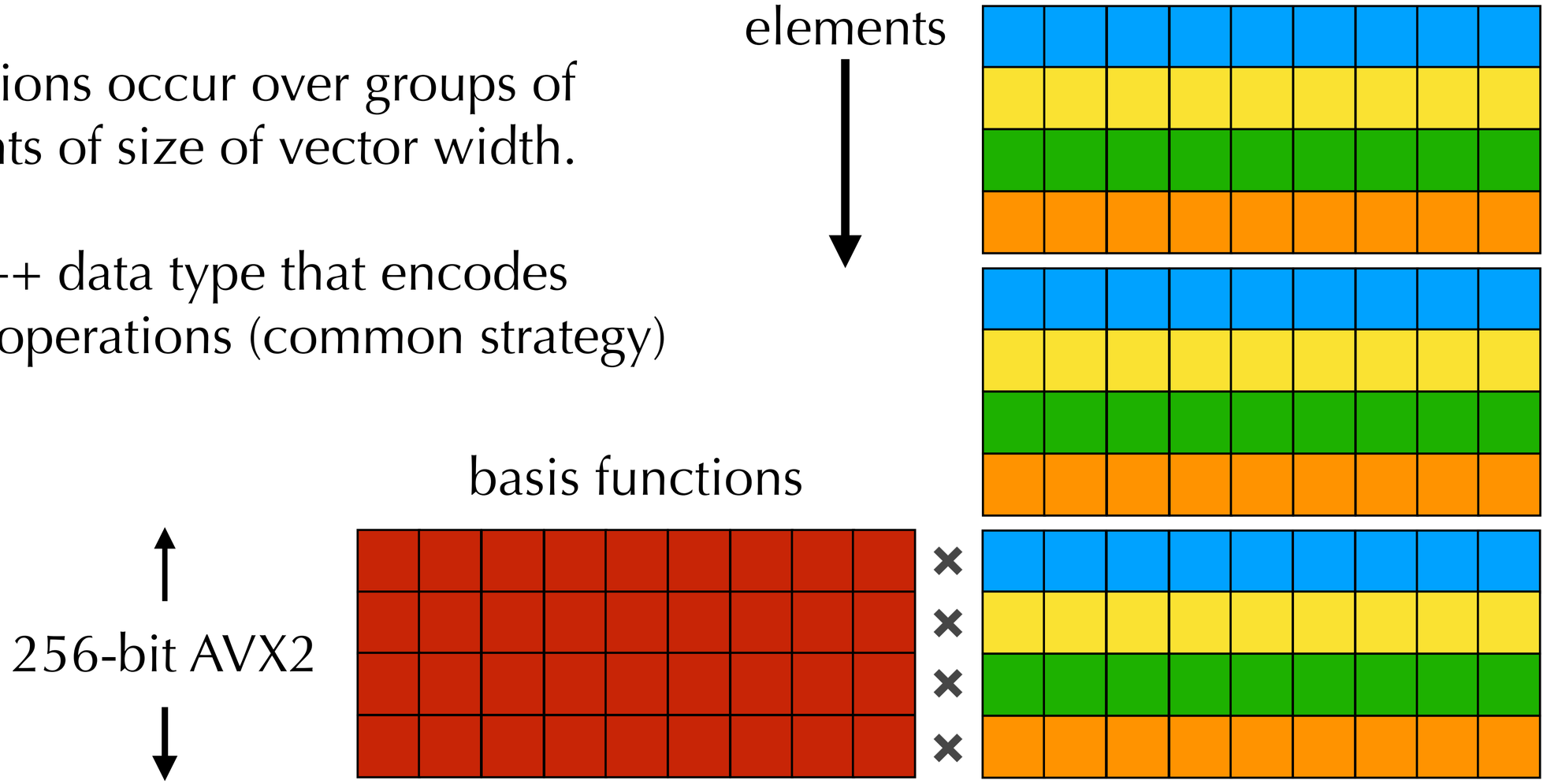
# Data layout

Exploit vectorisation by grouping DoFs by vector width



# Data layout

- Operations occur over groups of elements of size of vector width.
- Use C++ data type that encodes vector operations (common strategy)



# Tests

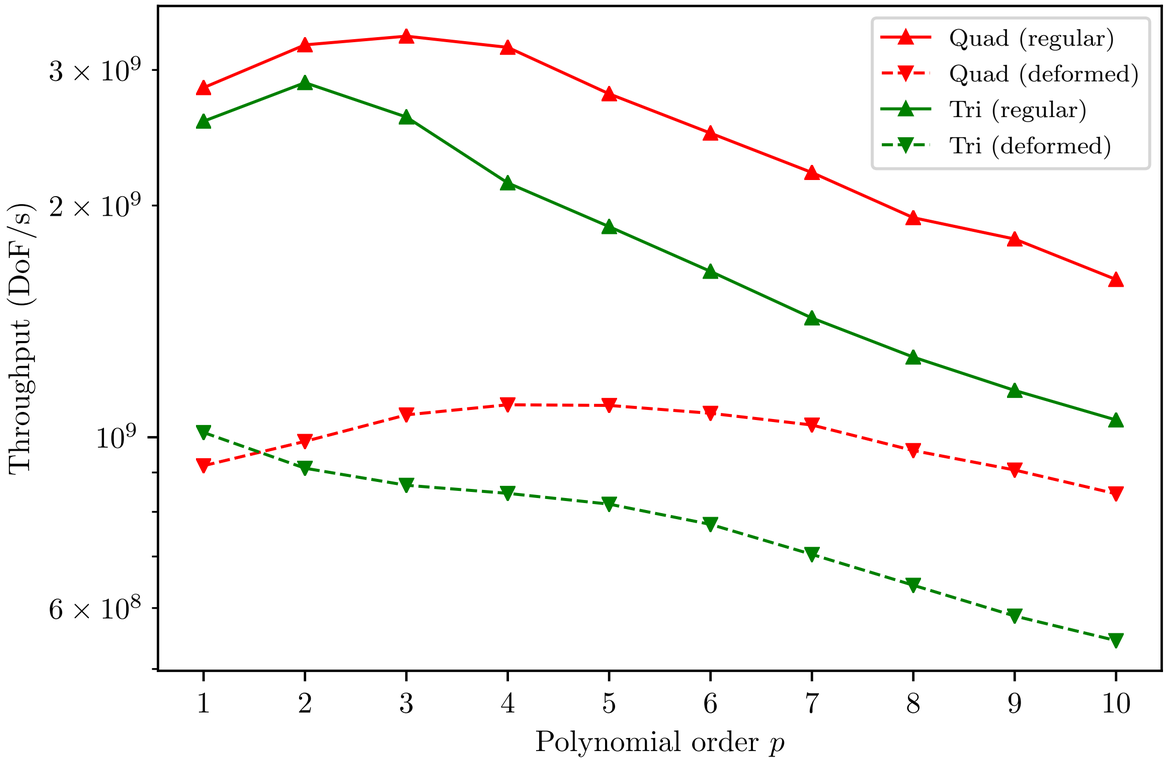
- Benchmarking of Helmholtz operator performed on two architectures with varying SIMD widths.

| | Broadwell (AVX2) | Skylake (AVX512) |
|---|---|---|
| **Model** | E5-2697 v4 | Xeon Gold 6130 |
| **Clock speed(s)** | 2.3 / 2.0 GHz<br>standard / AVX2 | 2.1 / 1.7 / 1.3 GHz<br>standard / AVX / AVX512 |
| **Cores / sockets** | 18 / 2 | 16 / 2 |
| **Max node GFLOP/s** | 1152 | 870 (AVX2)<br>1331 (AVX512) |

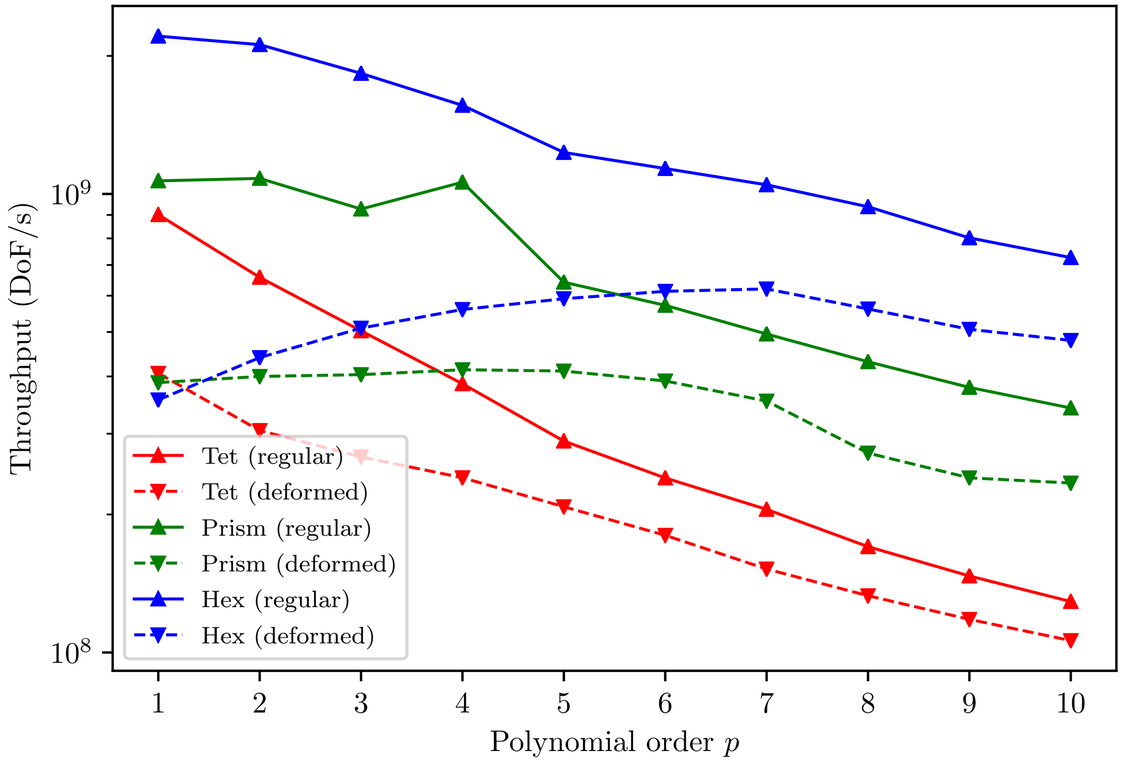
# Assessing performance

- Various techniques used to assess kernel performance:
  - **Throughput**: number of local DoF/s processed, for a mesh whose sizes exceeds available cache.
  - GFLOP/s gives some indication of capabilities, provided we are not memory-bound.
  - Better is **roofline analysis**: where do we sit in terms of memory bandwidth to arithmetic intensity?
- Note all results for local elemental operation evaluation only.

# Throughput (AVX2, Broadwell)




2D: Quads, triangles

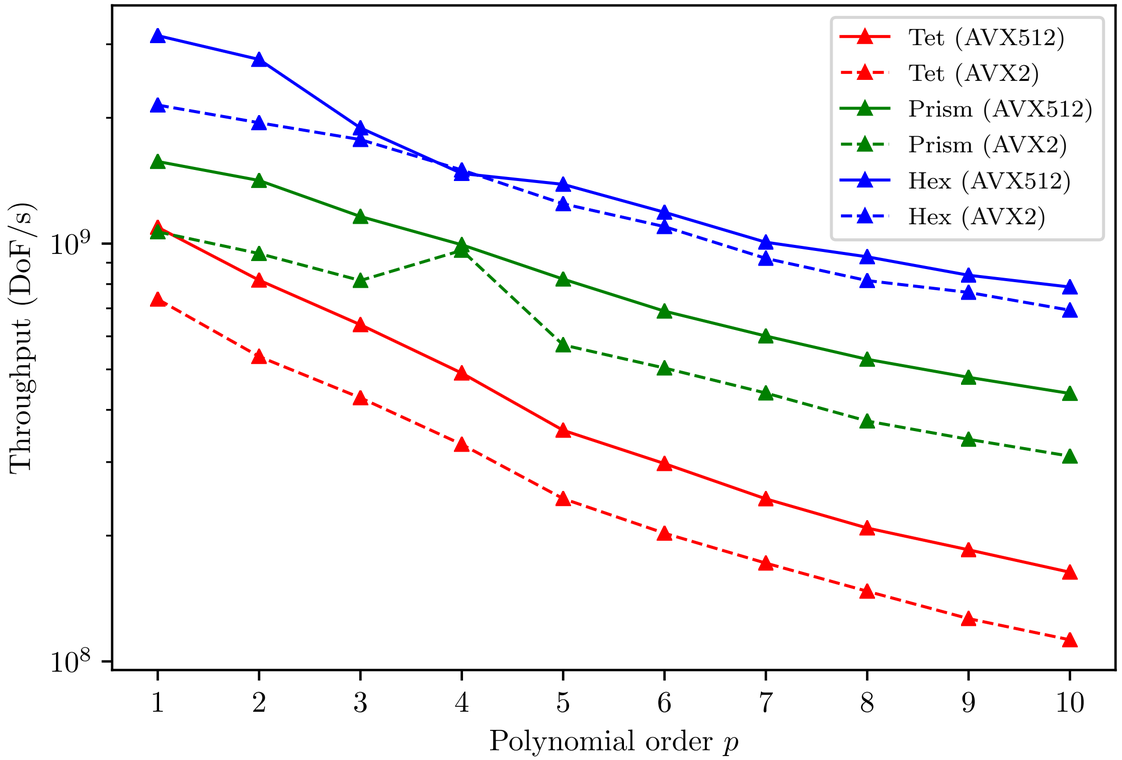



3D: Hexahedra, prisms, tetrahedra

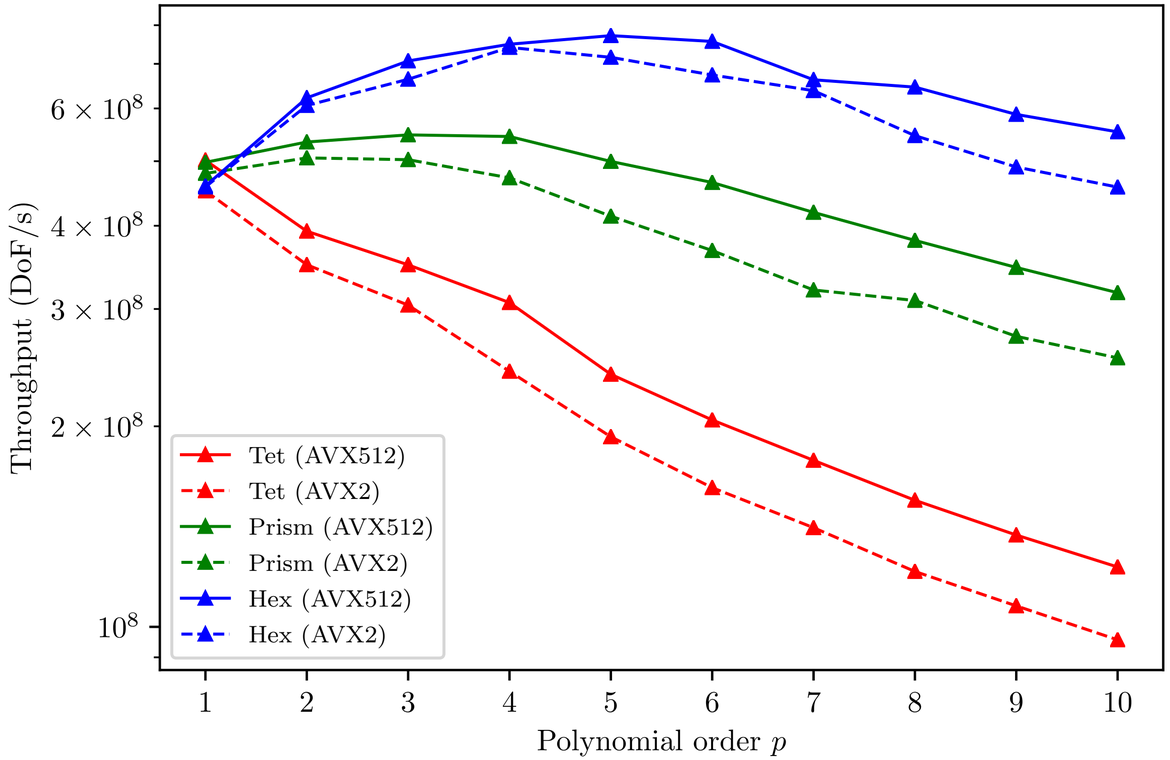
# Some clear trends

- This behaves pretty much as you might anticipate:
  - For regular elements, clear hierarchy of element type/dimension, where throughput is lost as dimension/complexity of indexing increases.
  - Regular elements outperform deformed elements: memory bandwidth.
  - Relative performance gap between deformed elements decreases at moderate polynomial orders.
- However, doesn’t really tell us how efficient our kernels are.

# Throughput (AVX512/AVX2, Skylake)




3D: ‘Regular’ elements




3D: ‘Deformed’ elements

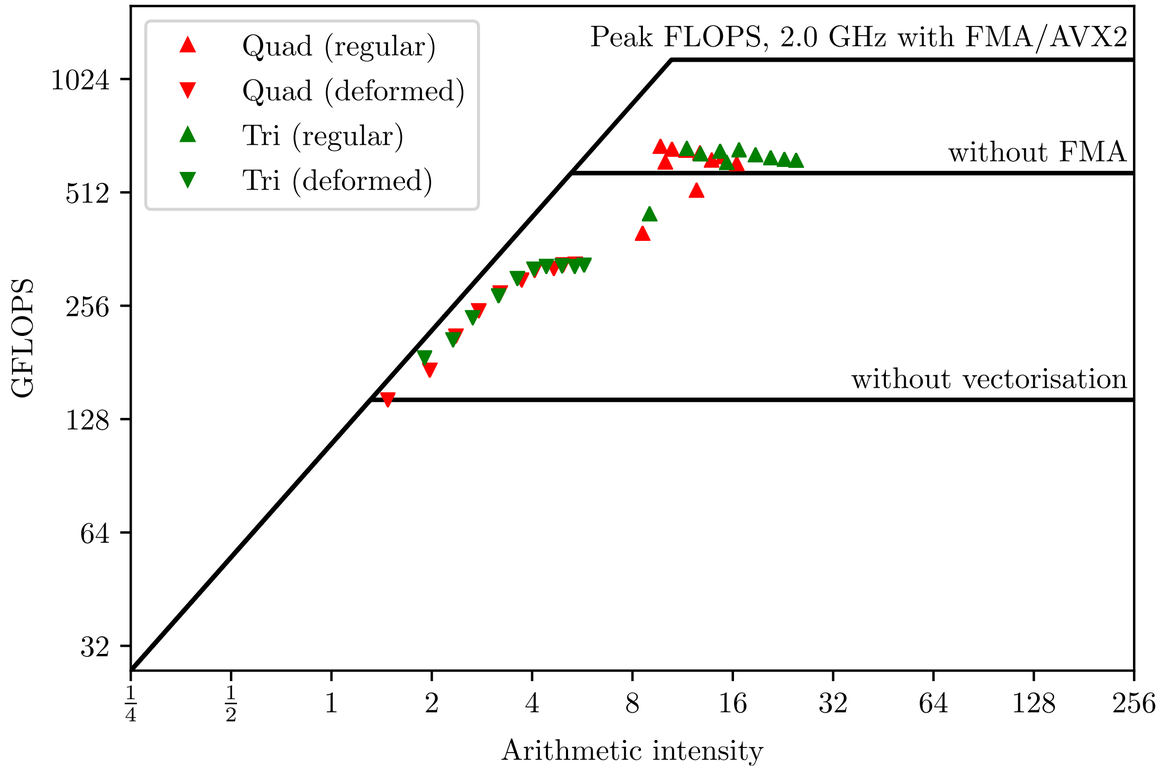
# Roofline model

- Roofline modelling should give better indication of ability of kernel to hit peak computational performance:

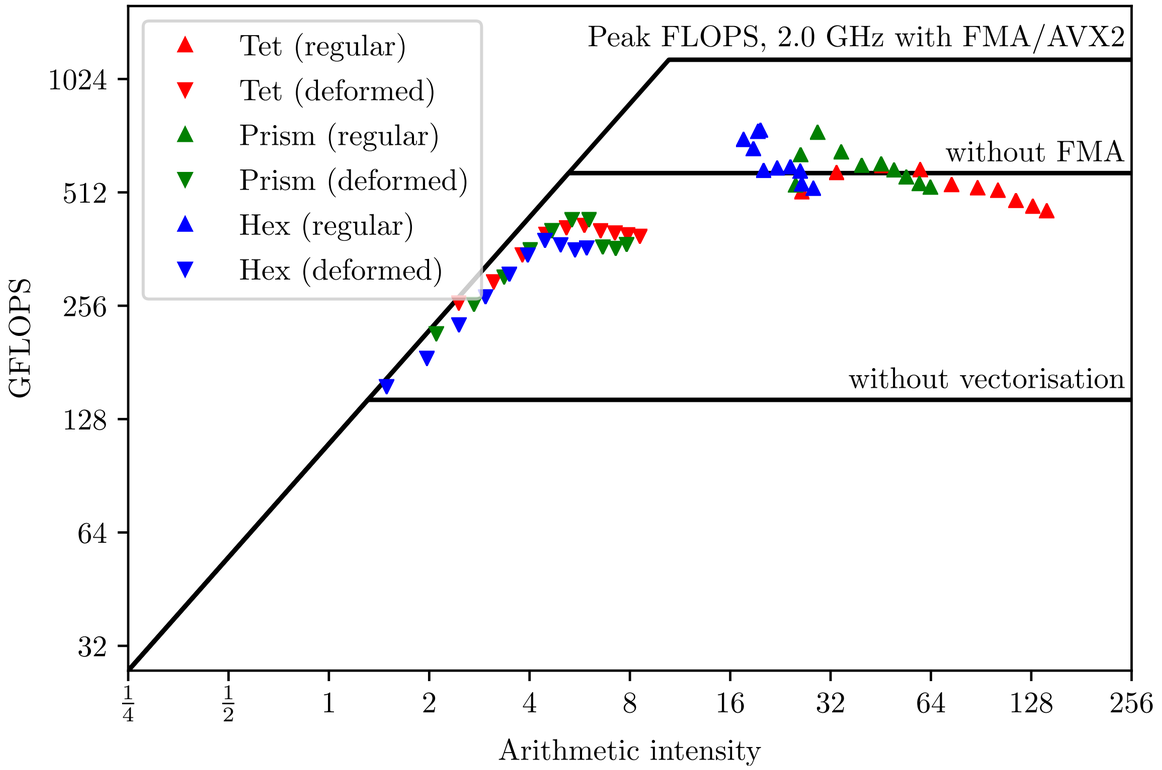
  Max GFLOPS/s = min(peak GFLOPS/s, peak memory bandwidth × $\alpha$)

- Profiled on Broadwell architecture using `likwid` performance profiling tools; hardware counters observed for GFLOPS/s and memory transfer.

# Roofline results




2D: Quads, triangles

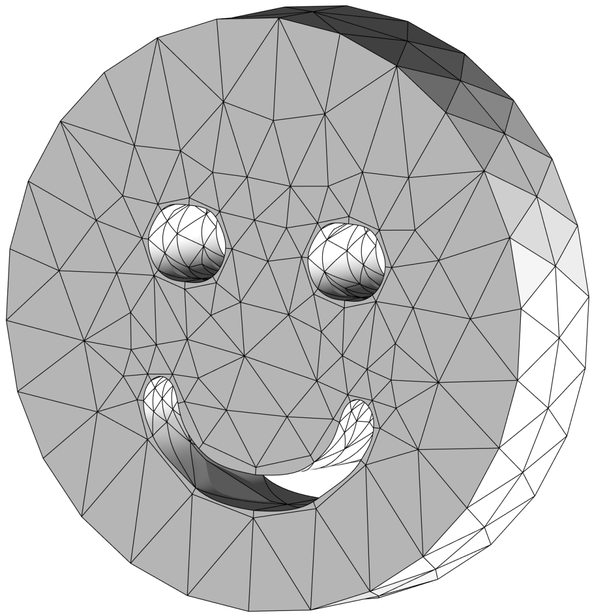



3D: Hexahedra, prisms, tetrahedra

# Summary

- Efficient matrix-free implementations of key finite element operators are certainly achievable on modern architectures for unstructured elements.
- Inevitable drop in performance from quads/hexahedra: complexity of indexing, additional cache pressure, etc.
- However relative performance of e.g. hex/prism and quad/tri is actually pretty good, particularly for deformed elements; important for e.g. boundary layer problems with large proportion of BL prisms.
- Clear future direction for this work!

Thanks for listening!



https://davidmoxey.uk/ @davidmoxey

d.moxey@exeter.ac.uk

www.nektar.info