IMPERIAL

Nektar++: Enabling high-fidelity modelling on heterogeneous HPC platforms

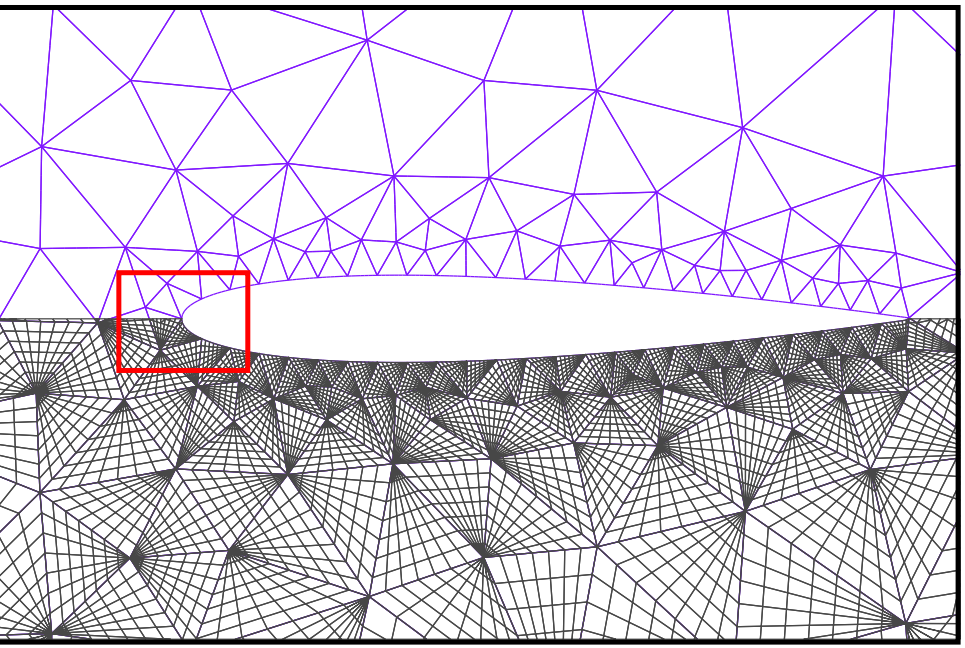
Chris Cantwell (ICL), David Moxey (KCL),
Jacques Xing (ICL), Boyang Xia (KCL), Alexandra Liosi (ICL),
Henrik Wustenberg (ICL), Chi Hin Chan (ICL),
Mike Kirby (Utah), Spencer Sherwin (ICL)

ARCHER2 Celebration of Science, Edinburgh
15th May 2025

Overview

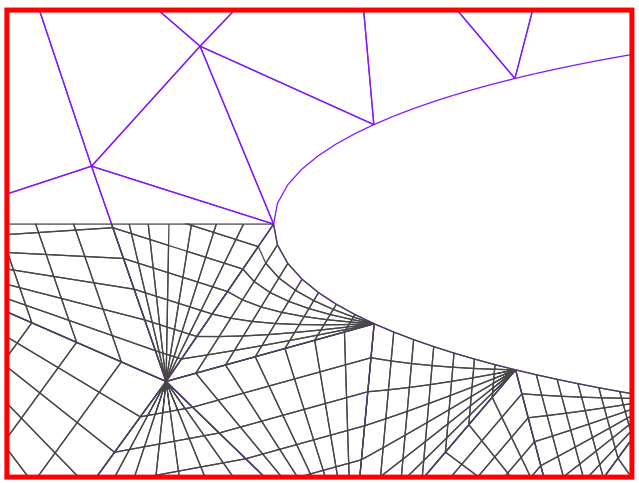
- Spectral/hp element methods and applications
- Nektar++ and its design philosophy
- Core code design
- Current performance results

Nektar++ and Spectral/hp element methods

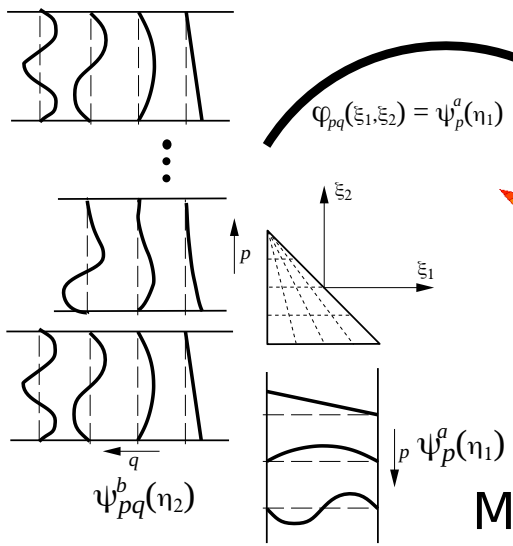


NACA 0012 Wing

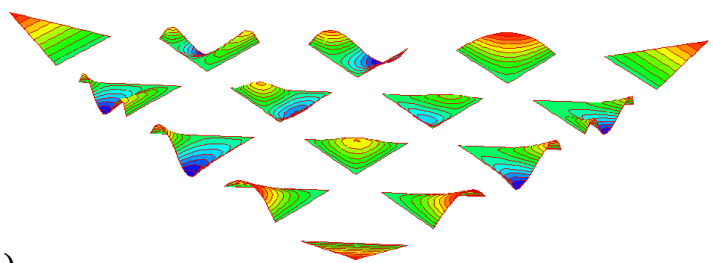
- High-order extension of linear finite element methods
- Enrich each element with high-order polynomial shape functions
- Coarser elements
- Nodal/Modal bases
- Tensor-product bases



High-order curved elements



$$P_n^{\alpha,\beta}(\xi)$$

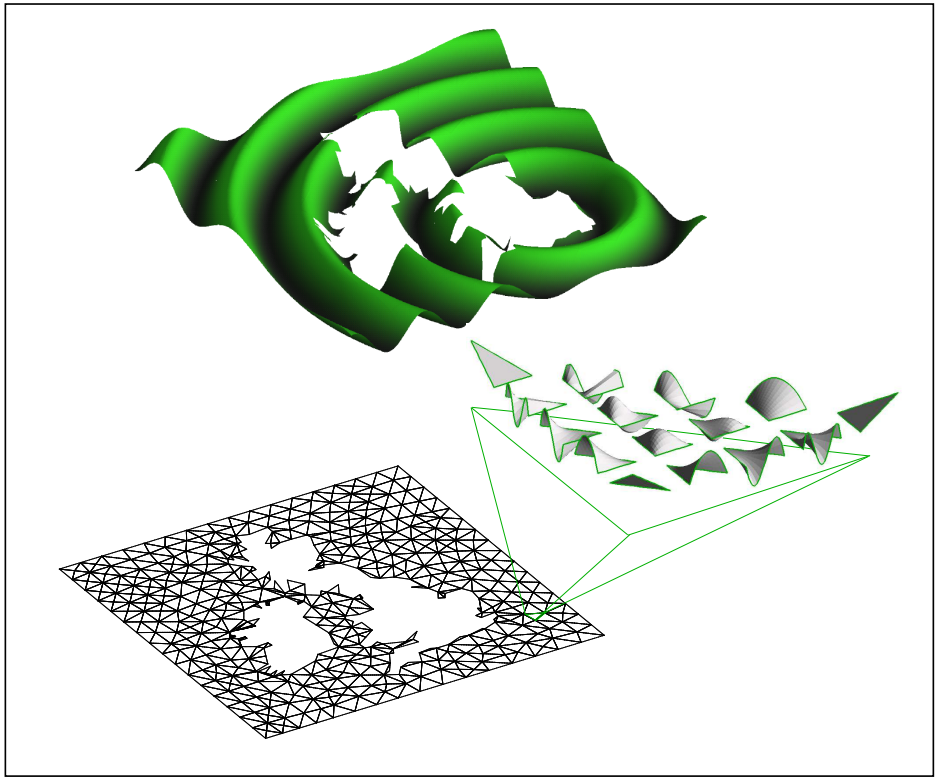


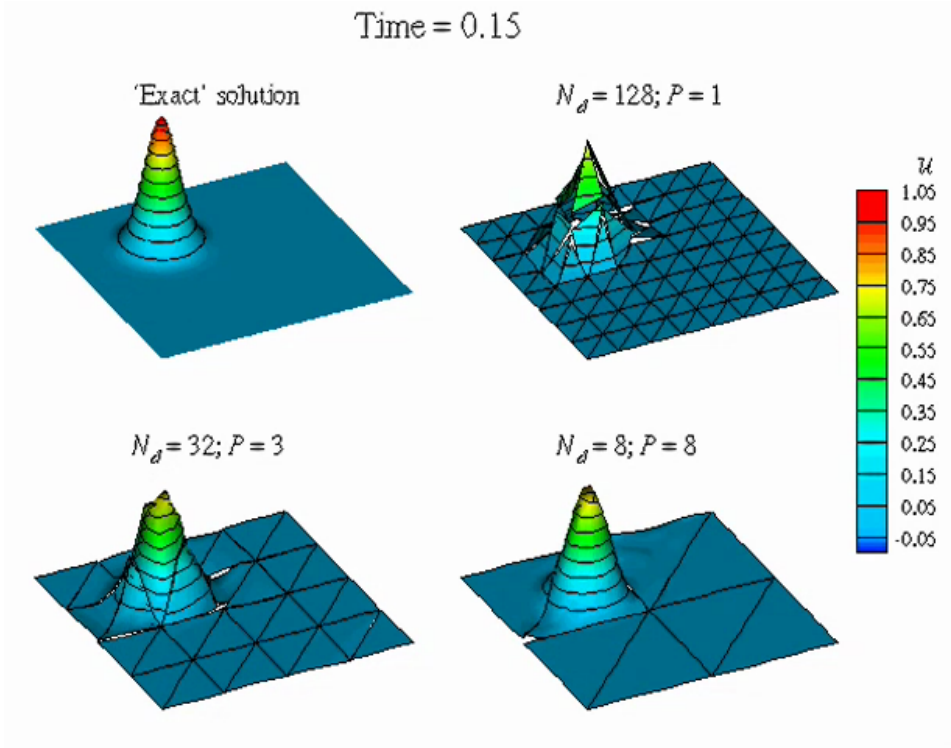
Modified Jacobi: $\alpha = 1, \beta = 1$

C. D. Cantwell, et al, Nektar++: An open-source spectral/hp element framework, Comput. Phys. Commun., vol.192, 2015 , p205–219

Nektar++ and spectral/hp element methods

Why are spectral/hp methods beneficial?

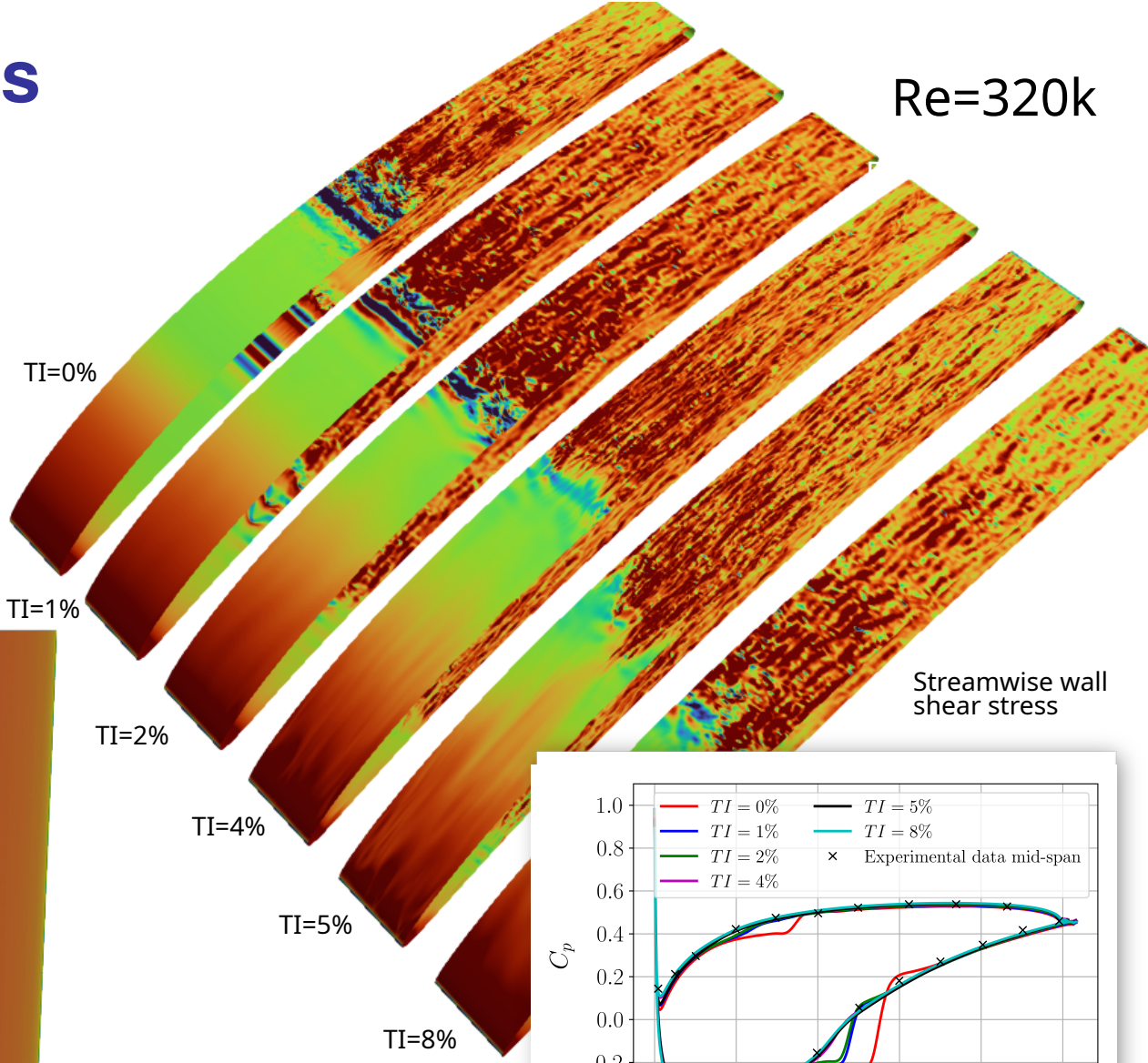




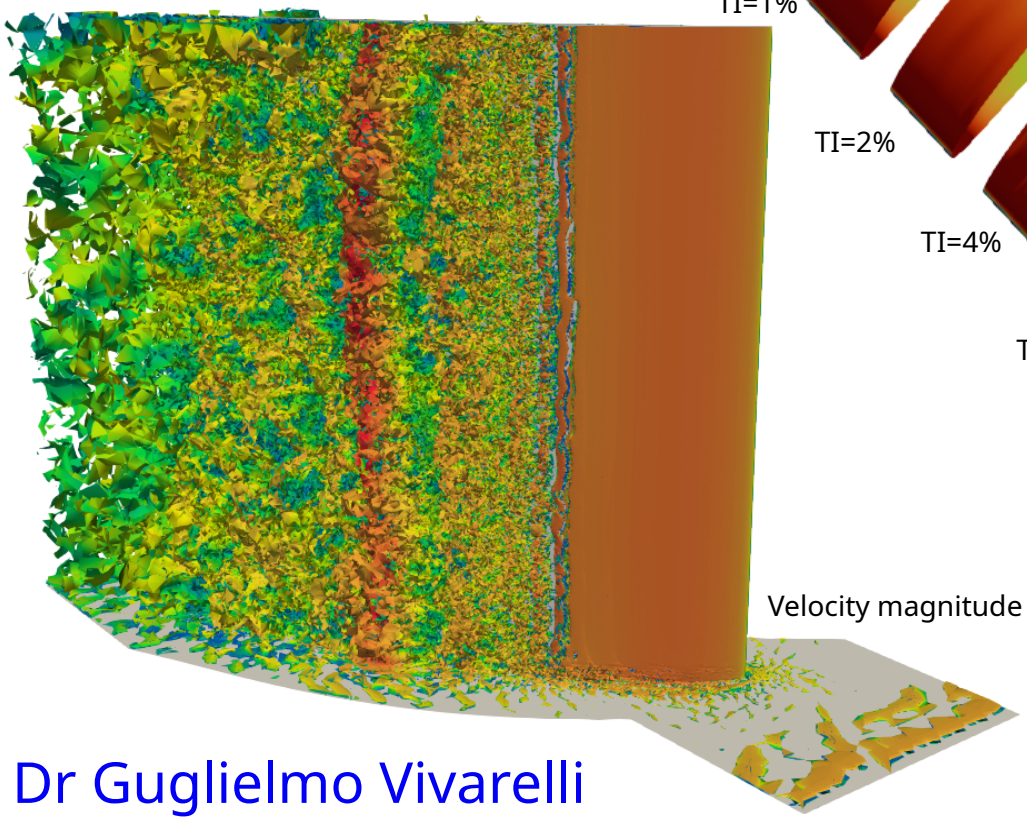
Enables improved capturing of multi-scale phenomena over longer time periods than low-order methods.

Compressor blades

Re=320k



Streamwise wall shear stress

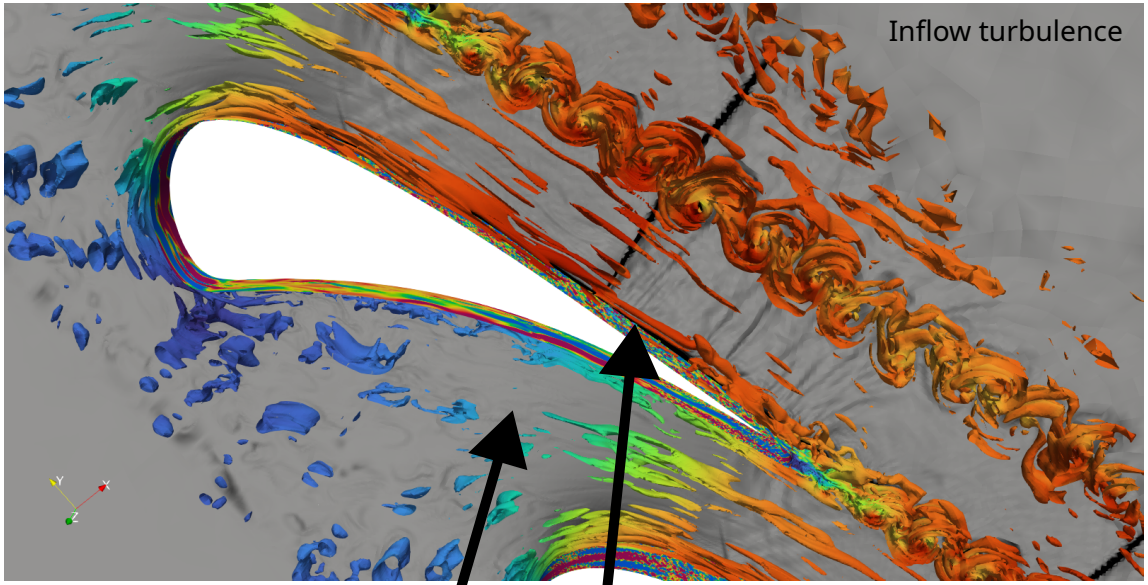


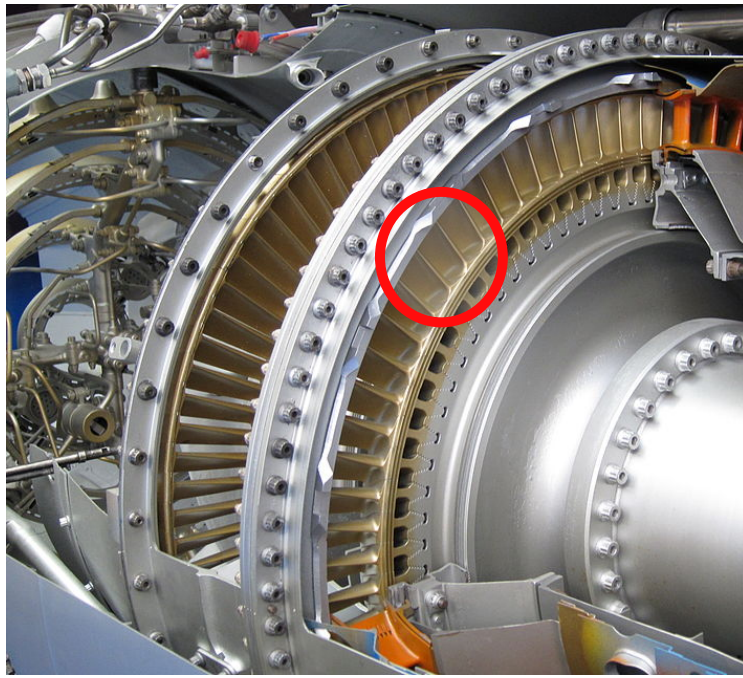
Velocity magnitude

Dr Guglielmo Vivarelli

Rolls-Royce®

High-pressure Turbine blades





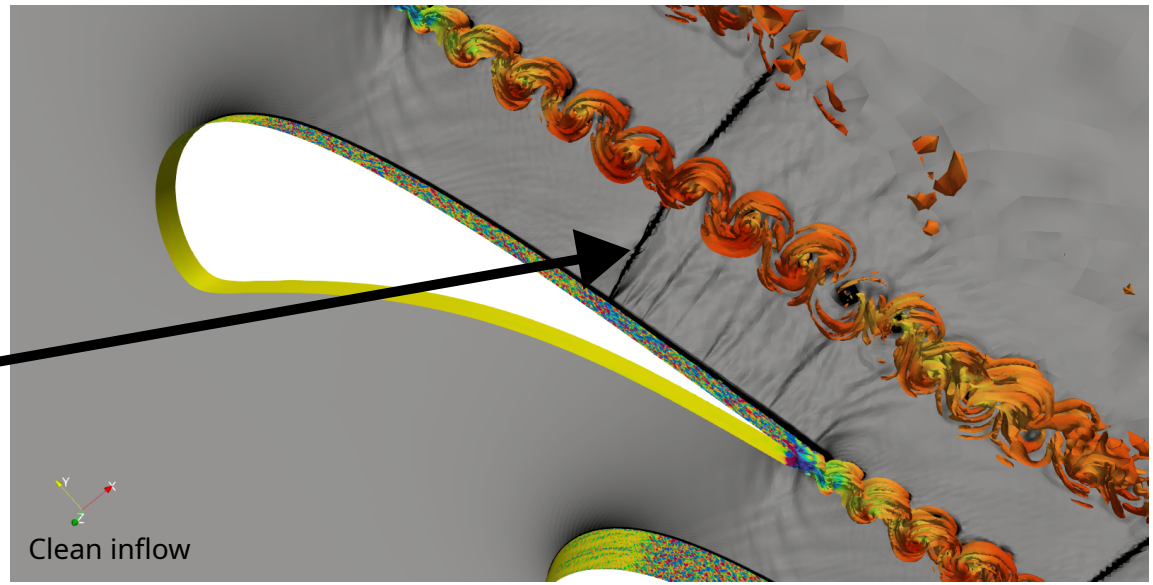
Re=800k~1M

Ma 0.15 to 0.9-1.1
in 5-10cm

Boundary layer ~1mm

Shocks between blades

High resolution and
stabilisation required

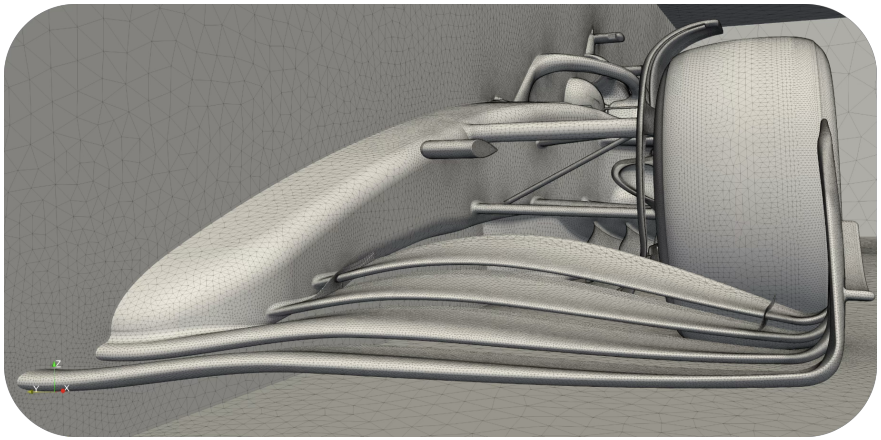


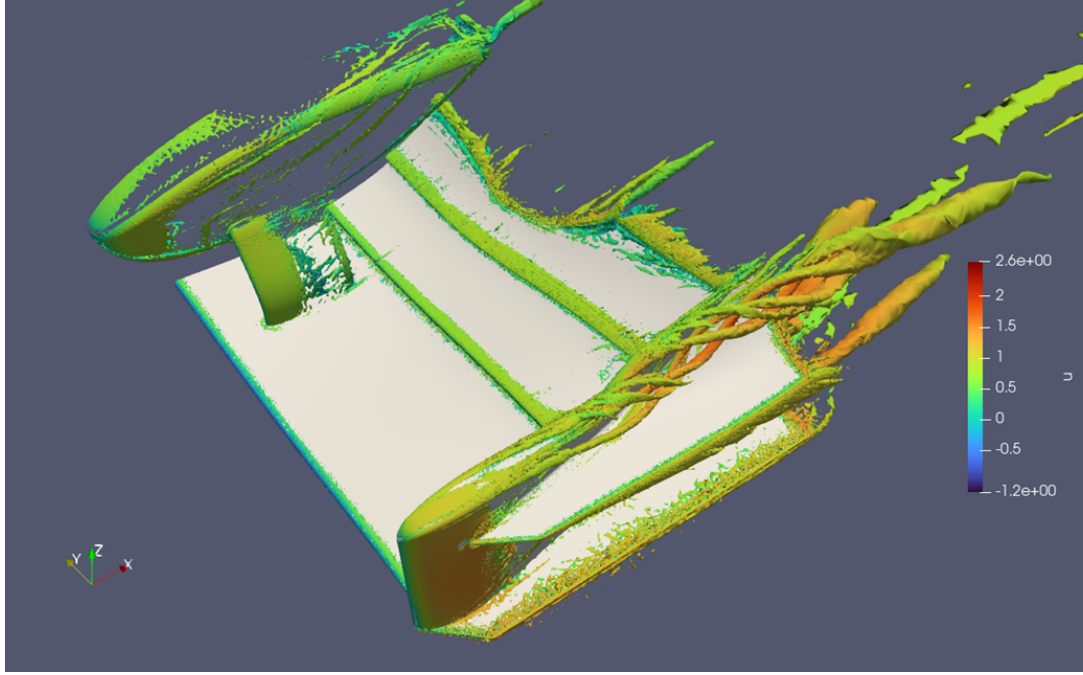
LS89 Test Case

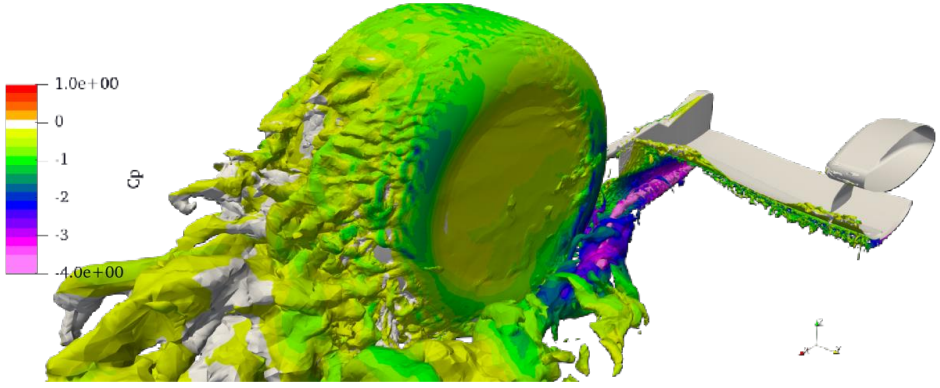
Dr Guglielmo Vivarelli

Rolls-Royce®

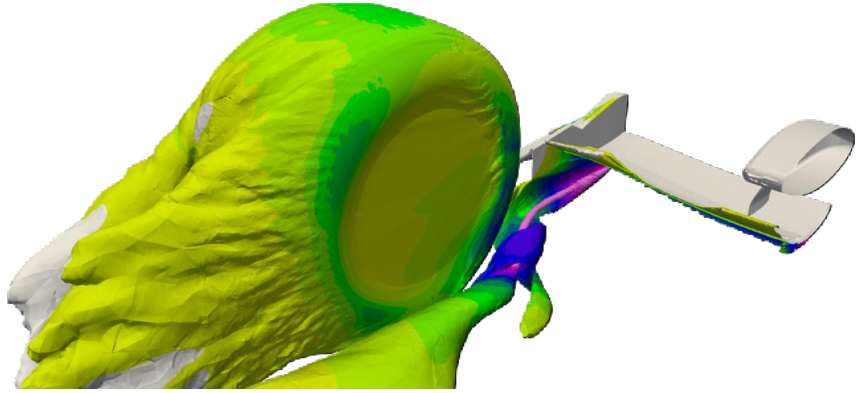
Formula 1



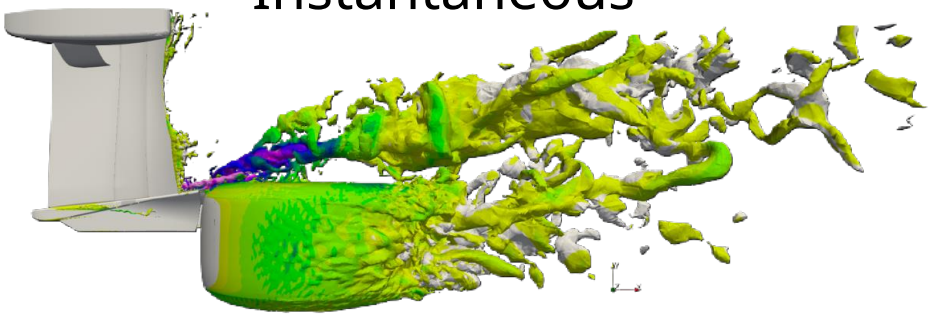


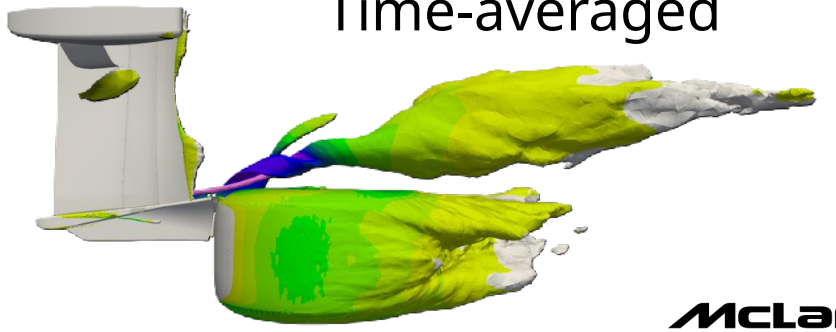


Instantaneous



Time-averaged

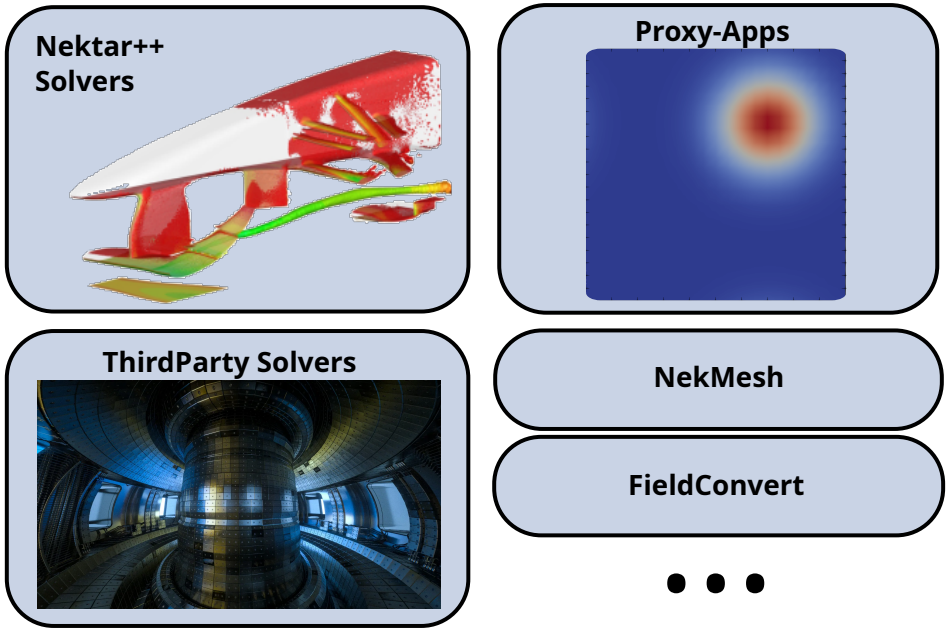




Parv Khurana; Alexandra Liosi

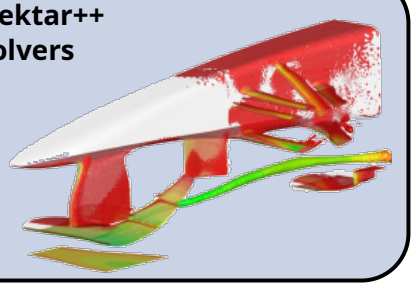
McLaren
FORMULA 1 TEAM

Nektar++ framework

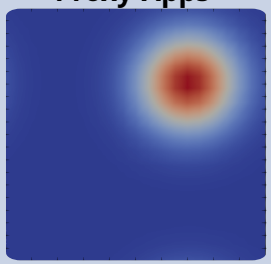


Nektar++ framework

Nektar++ Solvers



Proxy-Apps



ThirdParty Solvers

NekMesh

FieldConvert

• • •

Nektar++ Libraries

BLAS	LAPACK	Scotch
METIS	Boost	TinyXML
HDF5	gslib	MPI
Tetgen	FFTW	Arpack
MUI	CWIPI	OpenCascade

APPL. DOMAIN

SolverUtils
$\nabla^2 u - \lambda u = f$

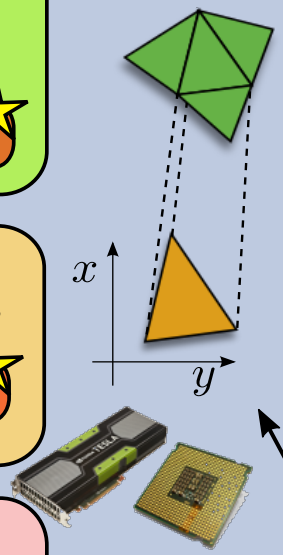
NekMeshUtils **FieldUtils**

MultiRegions
$u^\delta(x) = \sum_n^{N_{\text{dof}}} \Phi_n(x)\hat{u}_n$

Operators

LocalRegions
$u^\delta(x) = \sum_p^P \phi_p([\chi_e]^{-1}(x))\hat{u}_p$

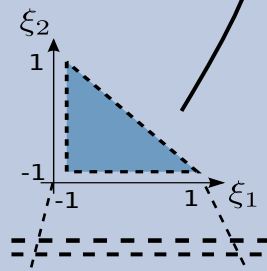
BlockOperators



SpatialDomains
$\mathbf{x} = \chi_e(\xi)$

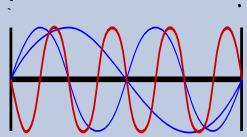
StdRegions
$u^\delta(\xi) = \sum_p^P \phi_p(\xi)\hat{u}_p$

Kernels



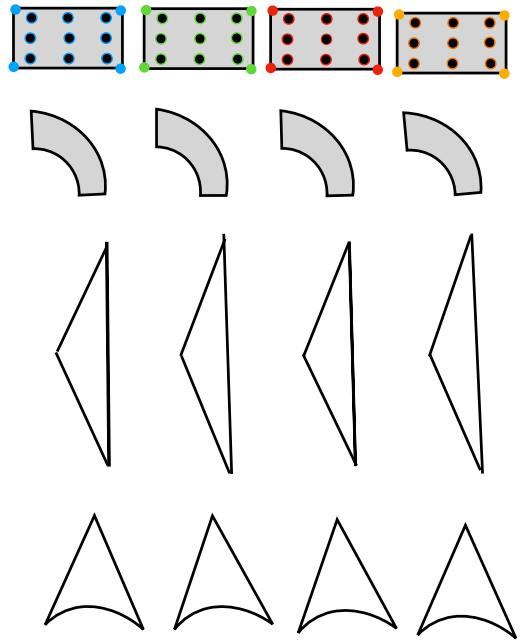
DISCRETISATION

LibUtilities
$\phi_p(x)$



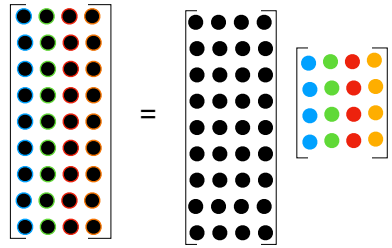
AUX

Need for algorithmic flexibility

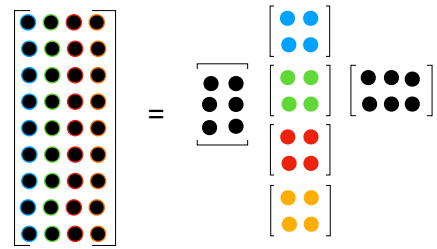


Polynomial order, Shape, Deformation

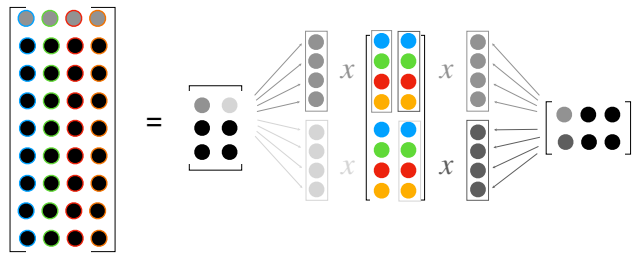
Interpolation/Backwards Transformation



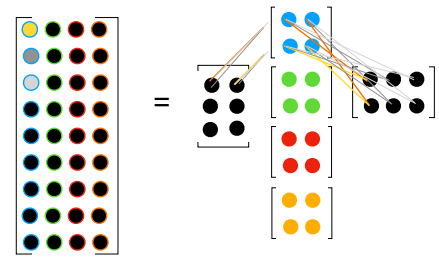
StdMatrix



Sum-factorisation



Sum-factorisation by Element
(vectorised)



Sum-factorisation by Entry
(threaded)

- Different elements shapes / orders / deformation.
- Target architecture / CPU / GPU.
- Performance of algorithms a function of both.

Nektar++ on heterogeneous platforms

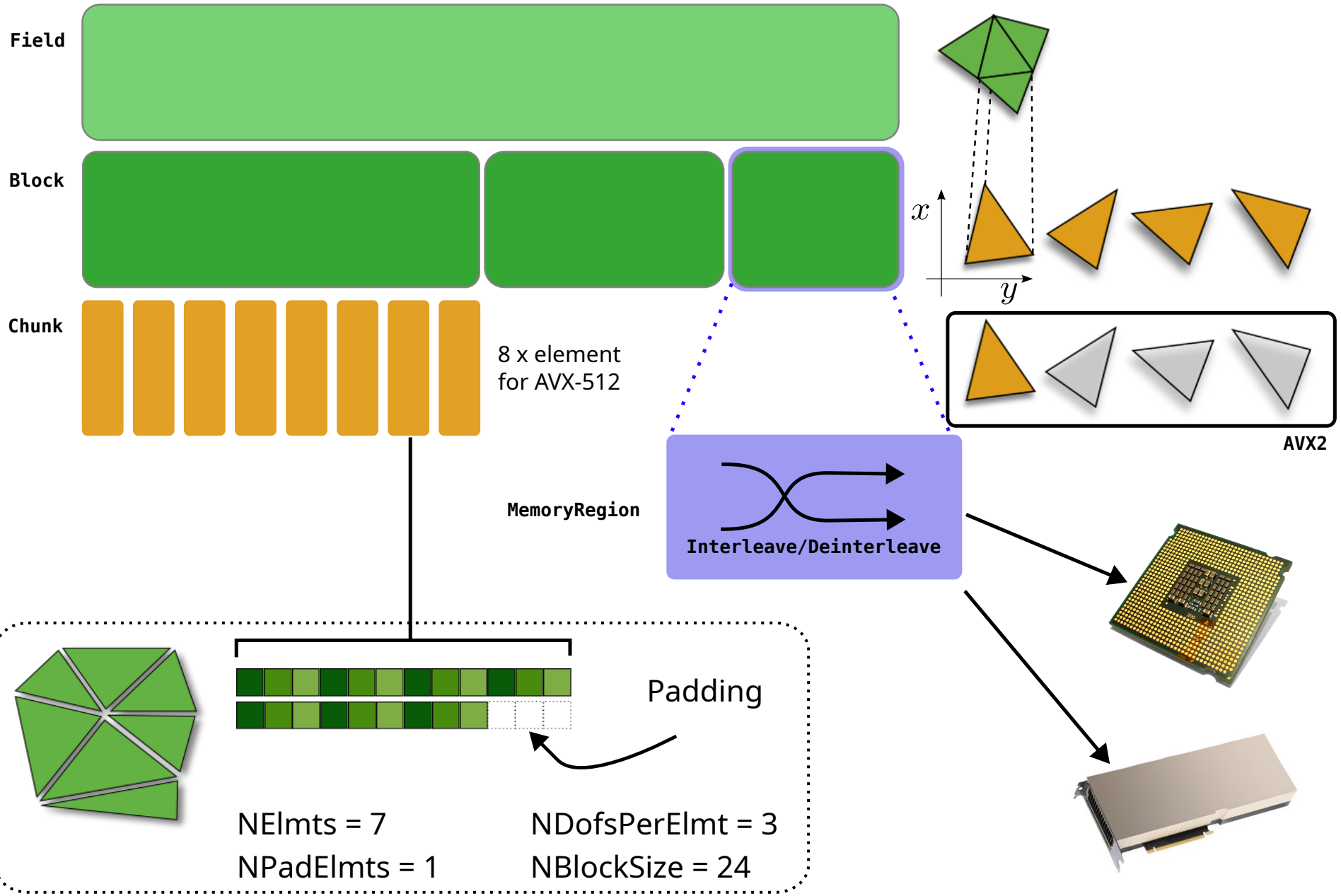
Design considerations

- Minimise data movement
- Align/interleave data appropriately to support SIMD / SIMT approaches
- Improve separation of data and algorithms
- Portability (hardware platforms / heterogeneity) – SYCL / Kokkos?
- Support platform-targeted optimisation (e.g. native kernels)
- Future-proofing: capability to support future hardware / programming models

Design decisions

- Abstraction layer to support multiple architectures / programming models and native kernels
- SYCL support to leverage both current and future platforms
- Templating of kernels to enable maximum compiler optimisation

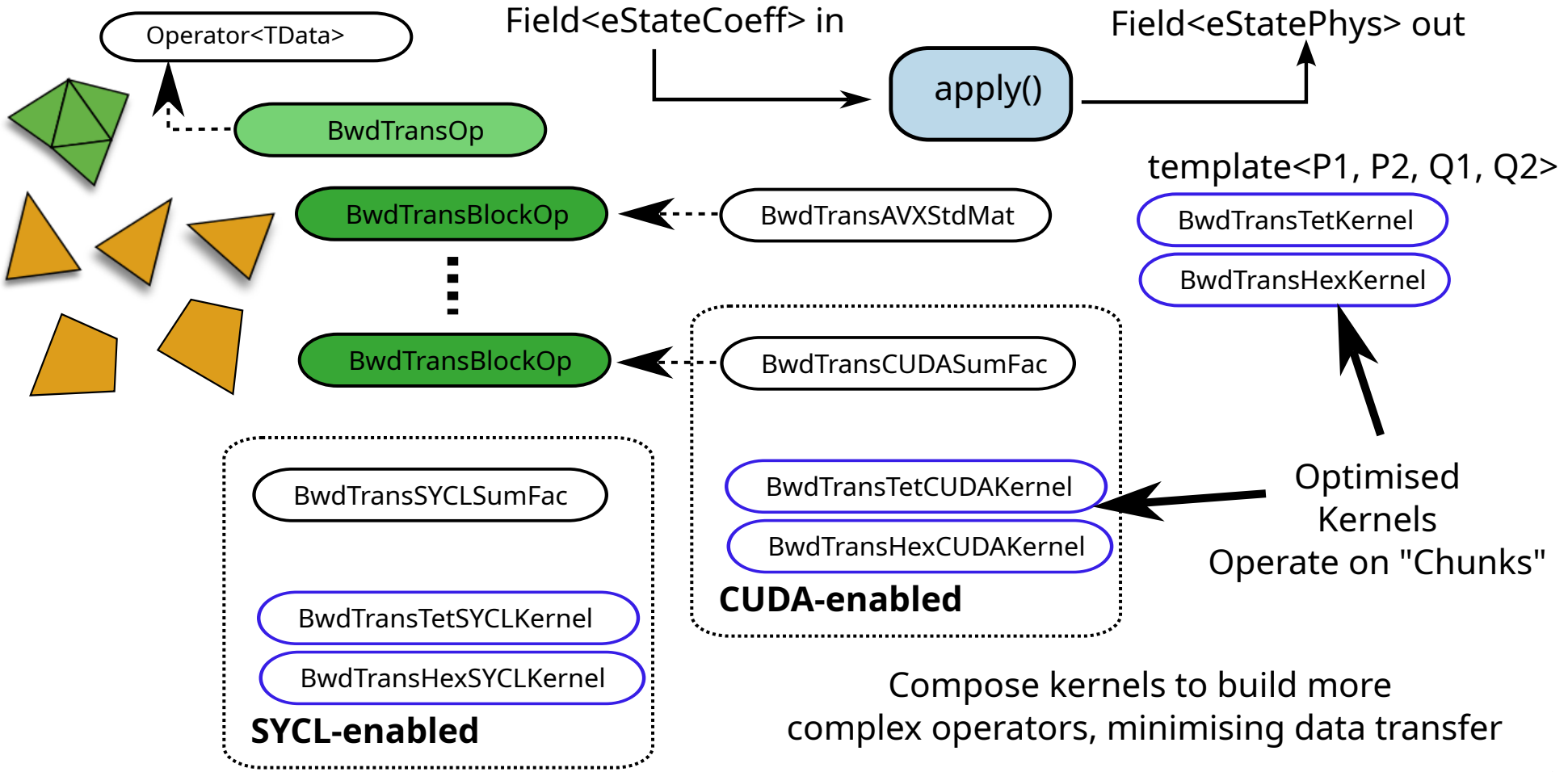
Code design: data management



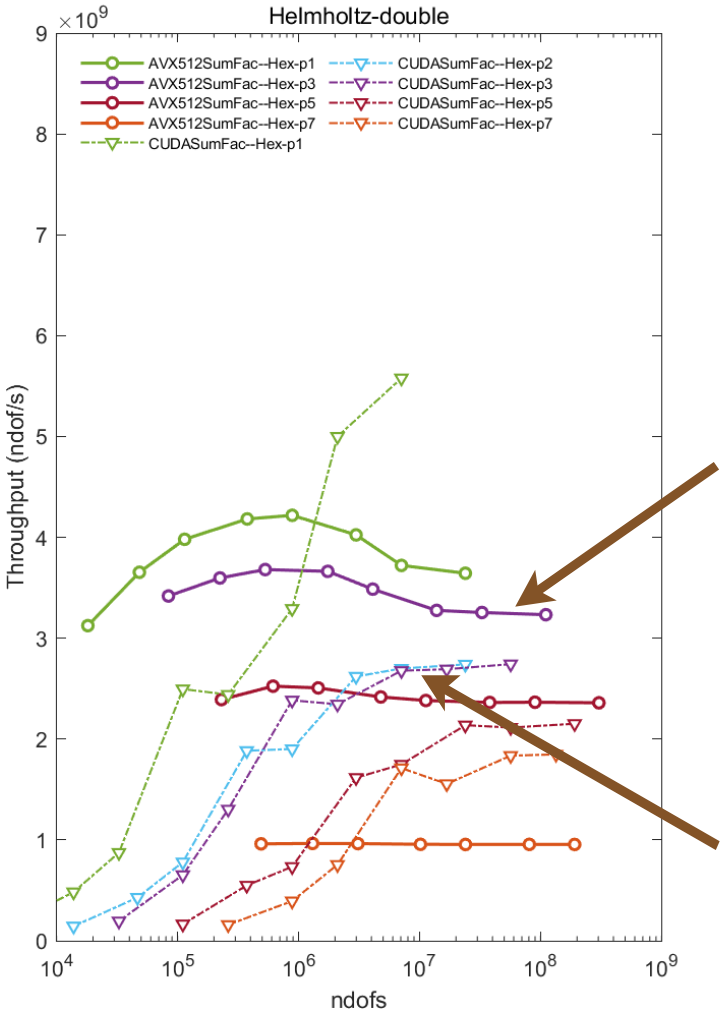
Code design: Operators and BlockOperators

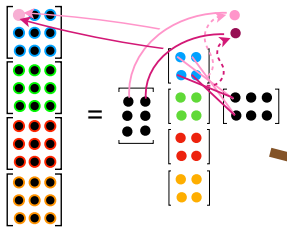
Represent algorithmic steps in reusable, composable and efficient forms

- **Abstraction**: solvers use base-class components - hiding specific implementation
- **Multiple implementations**: architecture-optimised, support programming models
- **High-performance kernels**: operations underpinned by compiler-optimised kernels

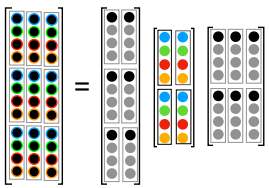


Performance

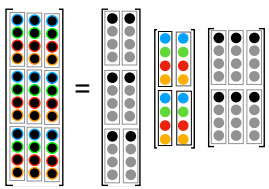




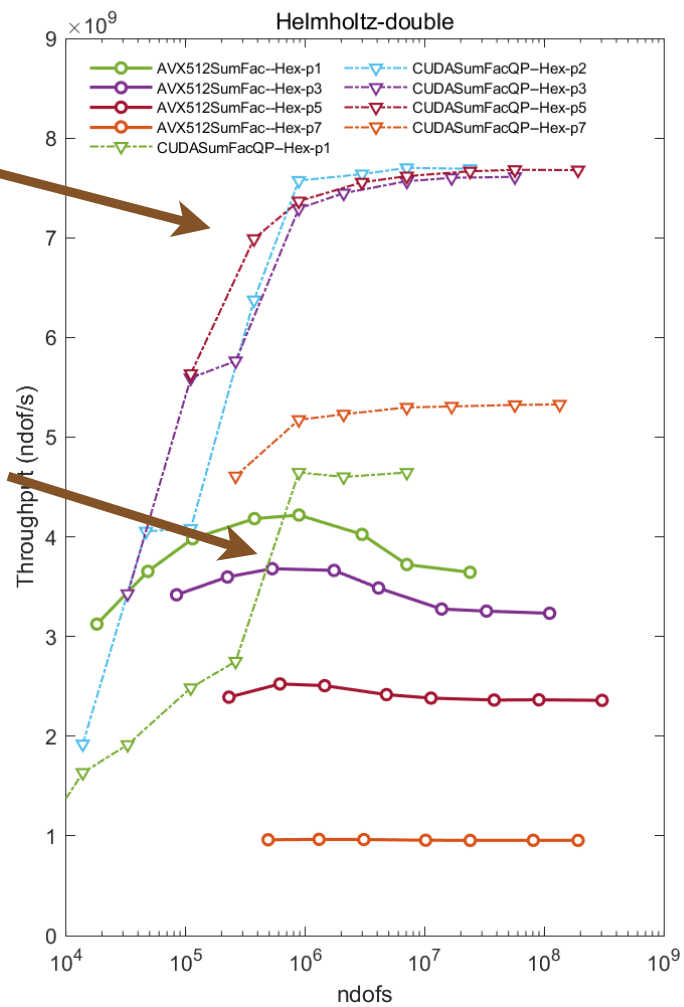
GPU A100
Entry by Entry



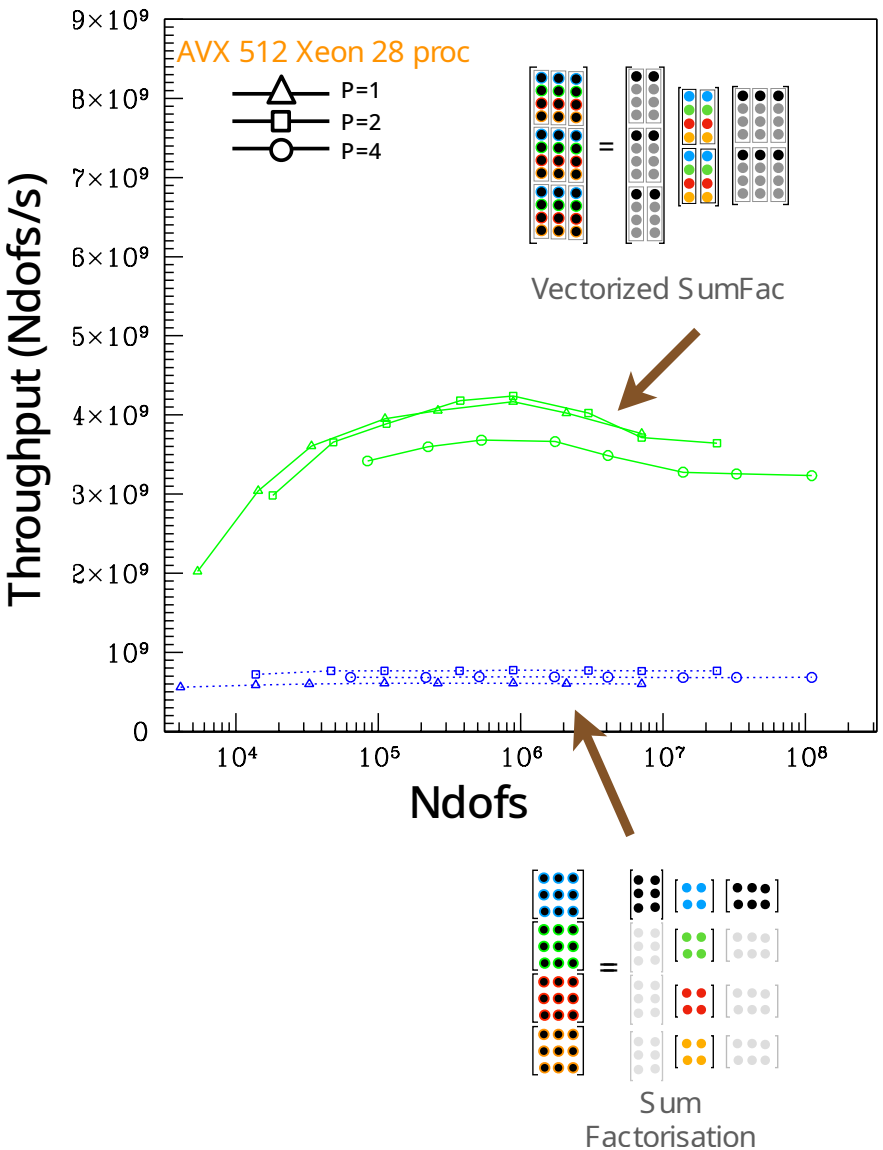
AVX 512 Xeon 28 proc
Vectorized SumFac

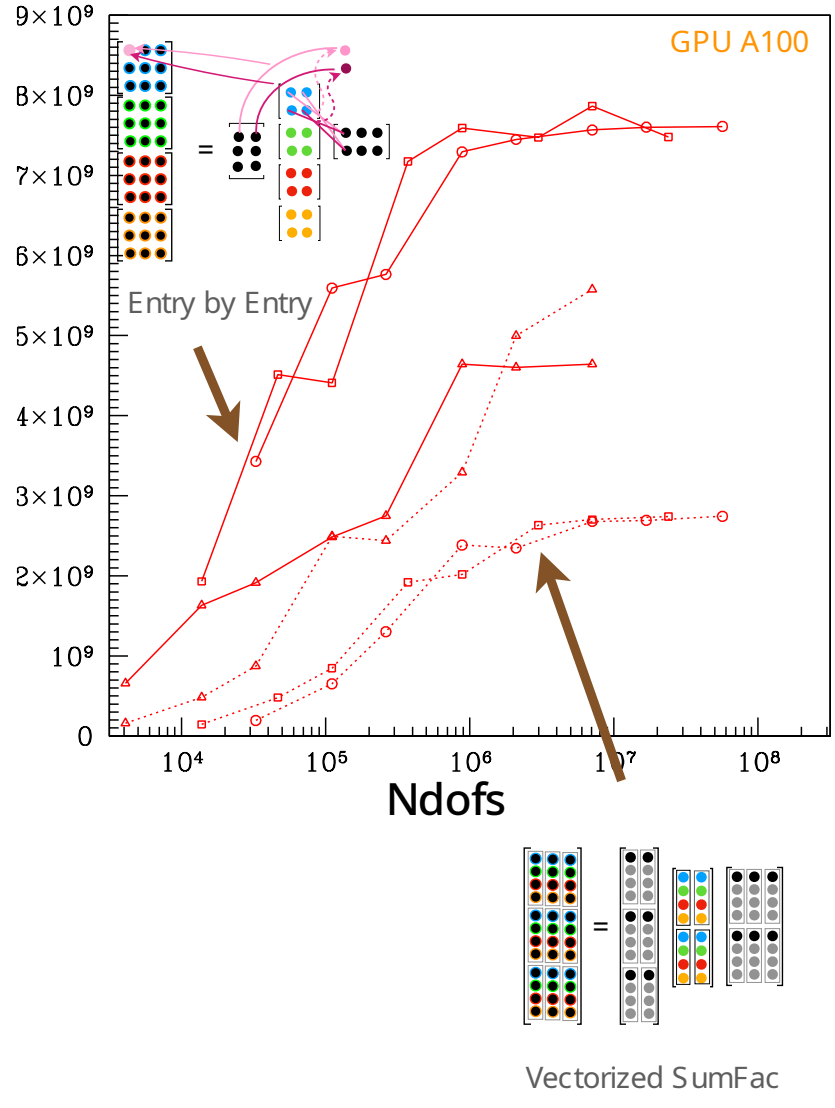


GPU A100
Vectorized SumFac



Performance





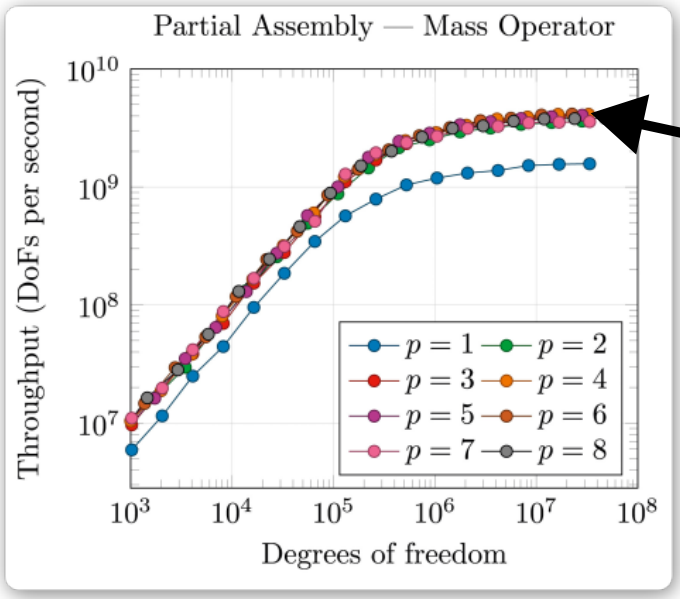
Comparable performance with other codes

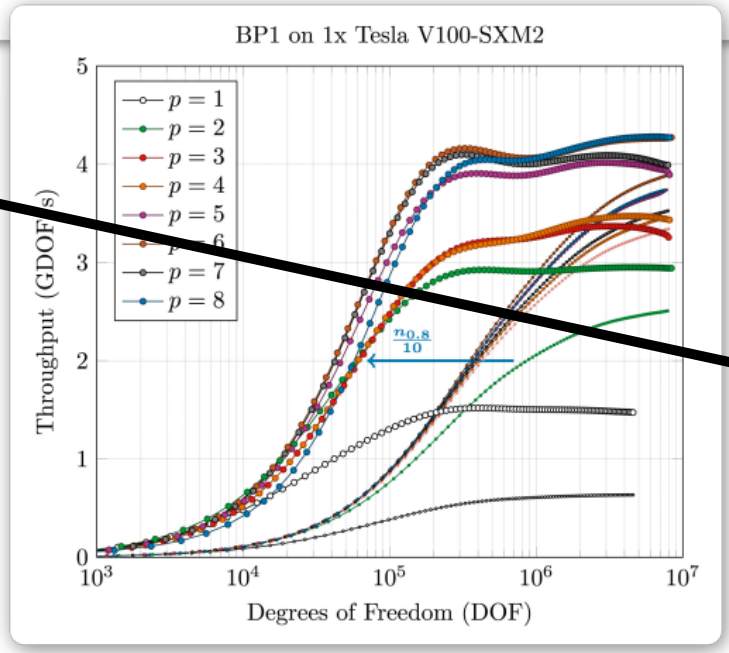
Available access | Research article | First published online June 14, 2024

High-performance finite elements with MFEM

Julian Andrej, Nabil Atallah, Jan-Phillip Bäcker, Jean-Sylvain Camier, Dylan Copeland, Veselin Dobrev, Yohann Dudouit, Tobias Duswald, Brendan Keith, Dohyun Kim, Tzanio Kolev, Boyan Lazarov, Ketan Mittal, Will Pazner, Socratis Petrides, Syun'ichi Shiraiwa, Mark Stowell, and Vladimir Tomov View all authors and affiliations

Volume 38, Issue 5 | https://doi.org/10.1177/10943420241261981





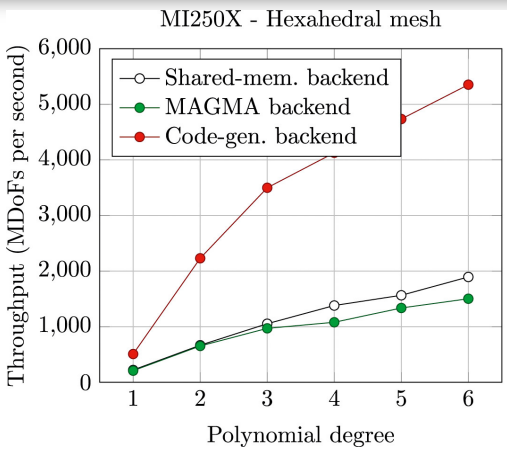
CUDA Kernels

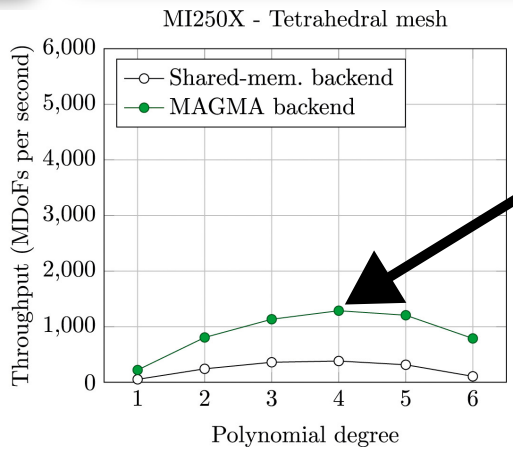
Tetrahedral elements

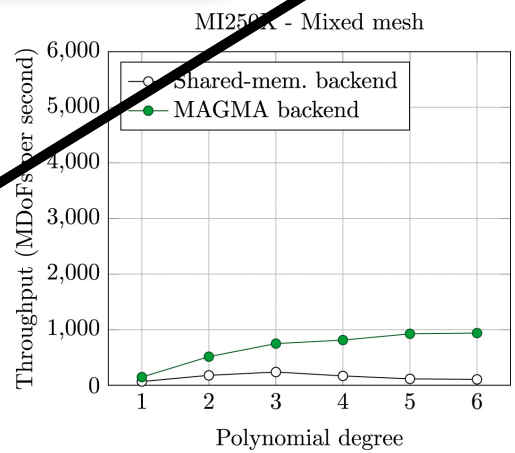
Nektar++ Helmholtz
~ 7x10^9 DOFs/s
NVIDIA A100 GPU

MFEM Mass
~ 3x10^9 DOFs/s
NVIDIA V100 GPU

MFEM Mass
~ 1.5x10^9 DOFs/s
AMD MI250 GPU







Summary

- Nektar++ and spectral/hp element methods enable high-fidelity modelling in complex geometries
- Successfully used at scale on large CPU systems (such as ARCHER2)
- On-going work to redesign code to efficiently leverage GPU systems
- Requirement to switch between algorithms to maximise performance
- Current GPU performance competitive with other leading packages
- Performance benefits from tensor-product bases on simplex elements

Acknowledgements

ARCHER2 eCSE GPU project GPU-eCSE01-48

Other support:

...and everyone who has contributed to the Nektar++ project!

Thank you for listening!