CERFACS
CENTRE EUROPÉEN DE RECHERCHE ET DE FORMATION AVANCÉE EN CALCUL SCIENTIFIQUE

HOPPS

Performance portability for high-fidelity CFD

Alexandre Dutka
CFD & HPC software engineer

@dutkalex
adutka@cerfacs.fr

adutka@cerfacs.fr
@dutkalex

About me

Engineer specialized in:
- Fluid dynamics
- Applied mathematics
- HPC

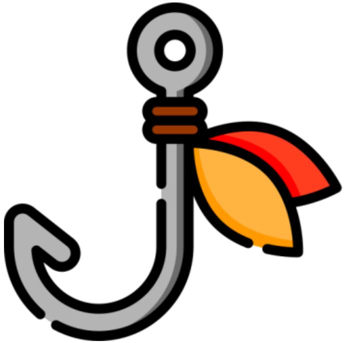
Finishing a PhD @ CERFACS:
- Compressible CFD
- High-order numerics
- Tree-based AMR

Day to day work:
- Lots of C++, Kokkos & MPI
- A fair amount of Python, t8code, CMake & HDF5 too

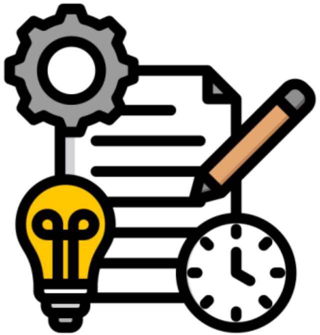
Also:
- Software design nerd
- OSS advocate & contributor
- CMake guru by accident

Disclaimer: Views expressed are my own and do not reflect my employer's

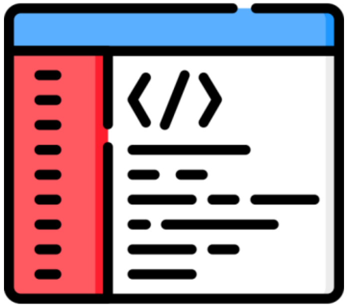
Outline



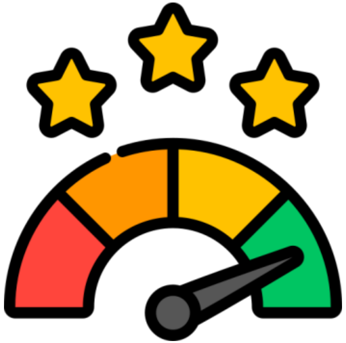
Quick intro



Numerical methodology



Implementation



Performance

High-fidelity CFD is expensive!

Compressible Navier-stokes equations

- Turbulence: 3D unsteady multi-scale phenomenon
- Acoustics: 100x less energetic than turbulence

Reynolds-Averaged Navier-Stokes:

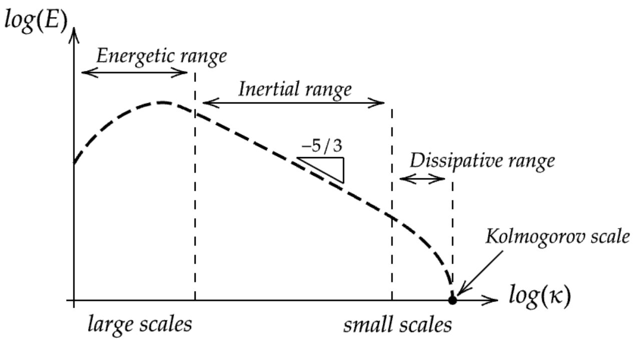
- Cheap because you only resolve the mean flow
- Low-fidelity approach

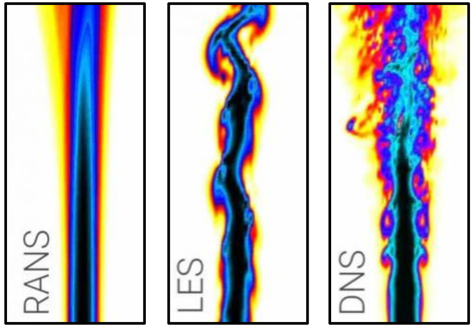
Large Eddy Simulation:

- Let's resolve what we can afford, and model what remains
- Most of the physics for a heavy but manageable computational cost

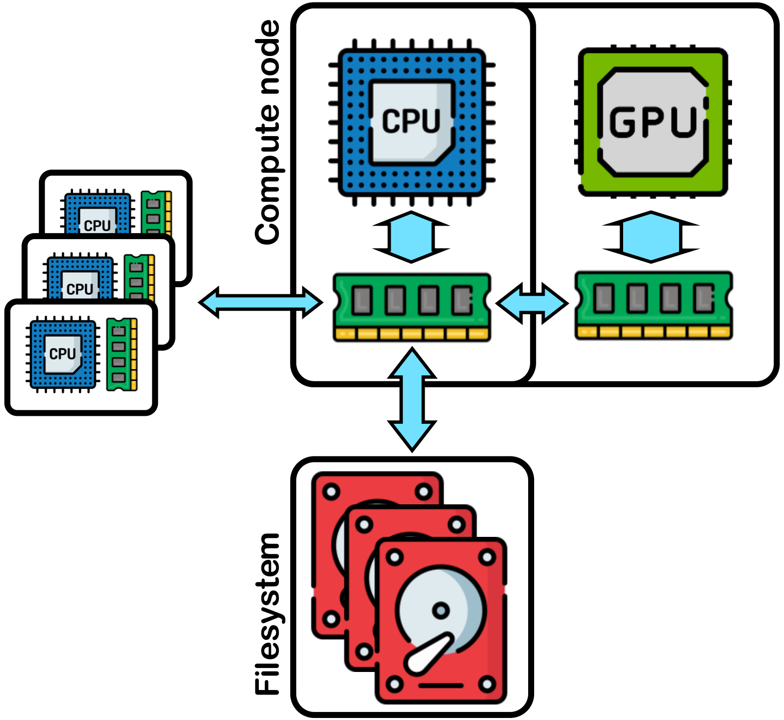
Direct Numerical Simulation:

- Great if you're patient and you have unlimited memory
- Only relevant for small academic configurations





What dictates performance?

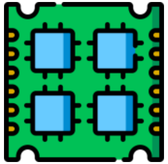


Bottlenecks
- Raw compute power
- Data transfers

Bandwidth

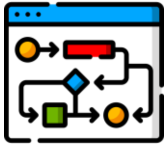
Latency

How to improve performance?



Adapt algorithms to new hardware

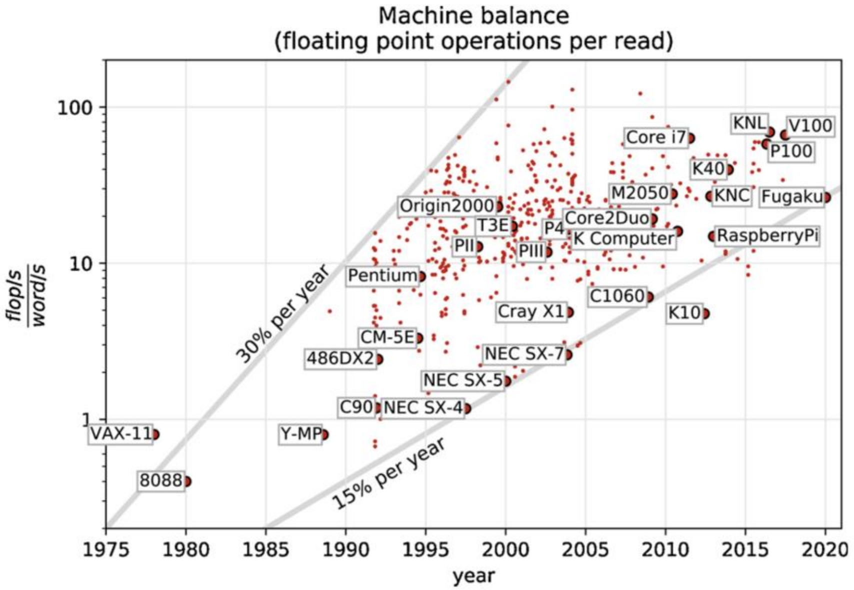
- Hardware diversity ⟹ need for portability across systems
- Fast evolutions ⟹ need for high-level abstractions
- Zero friction with existing workflows and practices



Use more efficient algorithms

- Improve accuracy and convergence ⟹ high-order methods
- Eliminate unnecessary dofs ⟹ adaptive mesh refinement
- Requires user adoption and evolution of practices

High-order methods: why?



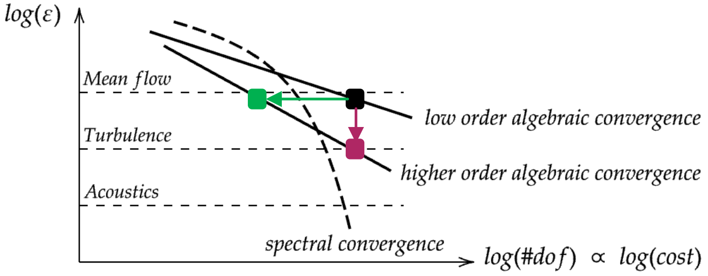
Source: Translational process: Mathematical software perspective, J. Dongarra *et al.*, 2020

Compute improves faster than memory

- Compute ↗ ↗ ↗
- Memory bandwidth ↗
- Memory latency →

Higher orders of accuracy either means

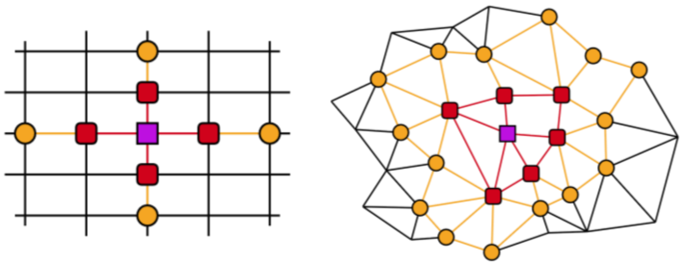
- More physics for the same mesh
- The same physics with a coarser mesh



High-order methods: how?

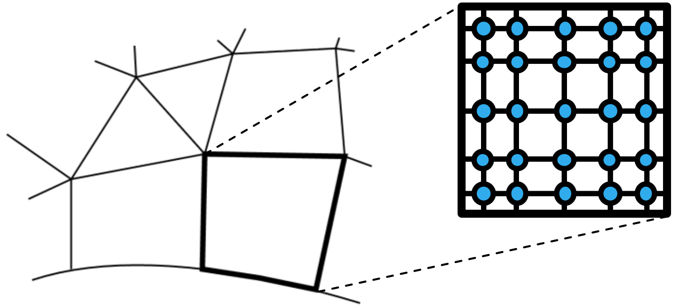
FV, FD, FE extensions

- ✓ Mature methodologies
- ⚠ Huge stencils on unstructured grids
- ⚠ Boundary conditions are tricky
- ✗ Comm. cost increase with the order

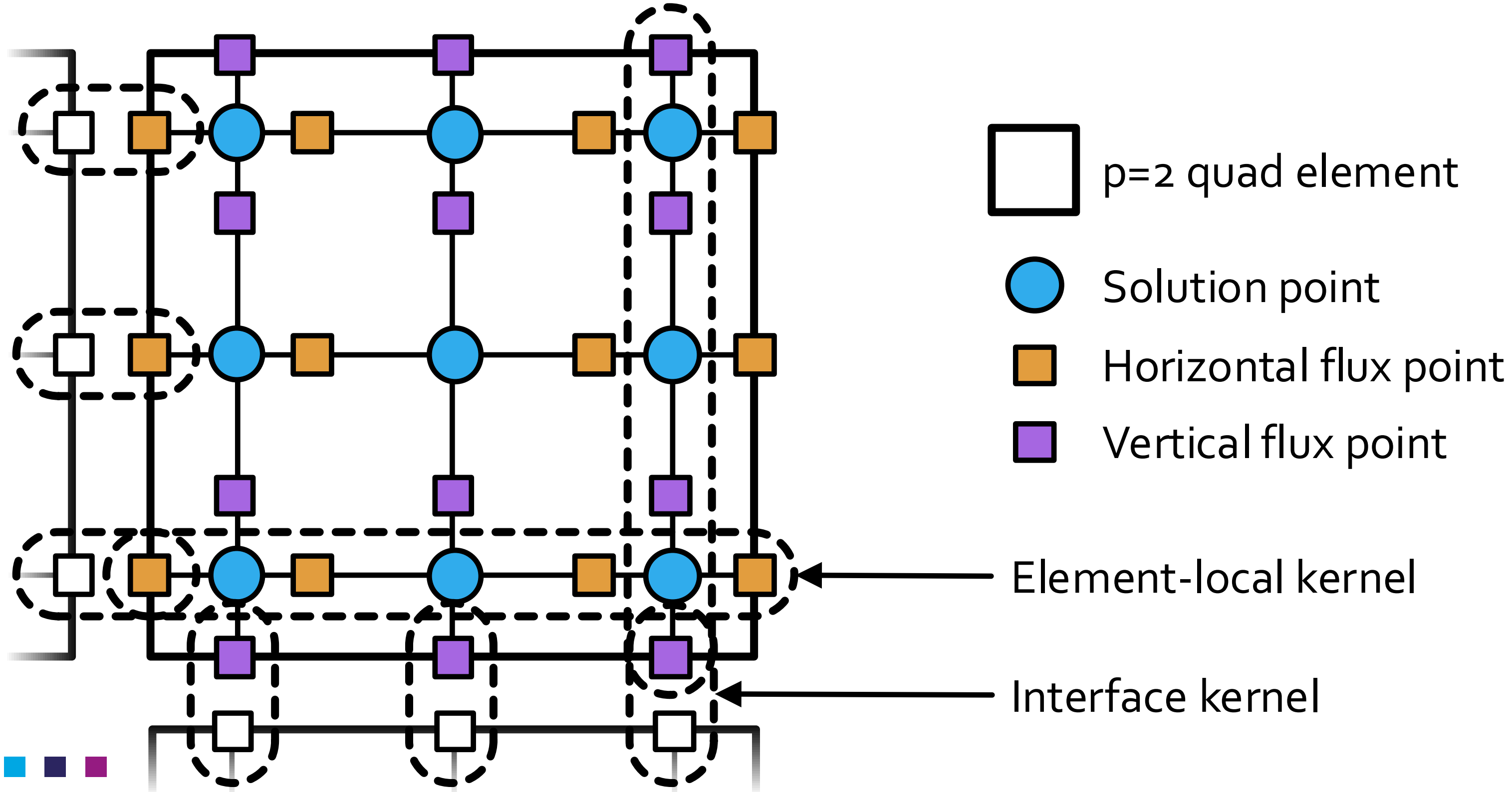


Discontinuous approaches (DG, SD, FR)

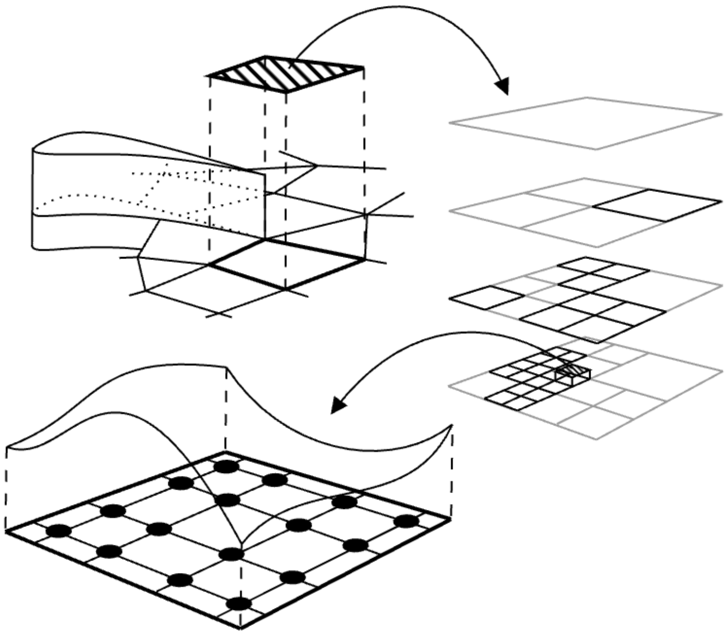
- ✓ Arbitrary orders
- ✓ Built-in hp-adaptation
- ✓ Comm. cost independent of the order
- ⚠ Relative lack of maturity



Spectral Difference method



HOPPS: overview



Rewrite of the JAGUAR solver

- Legacy Fortran + MPI code
- Compressible NS equations
- Spectral Difference method

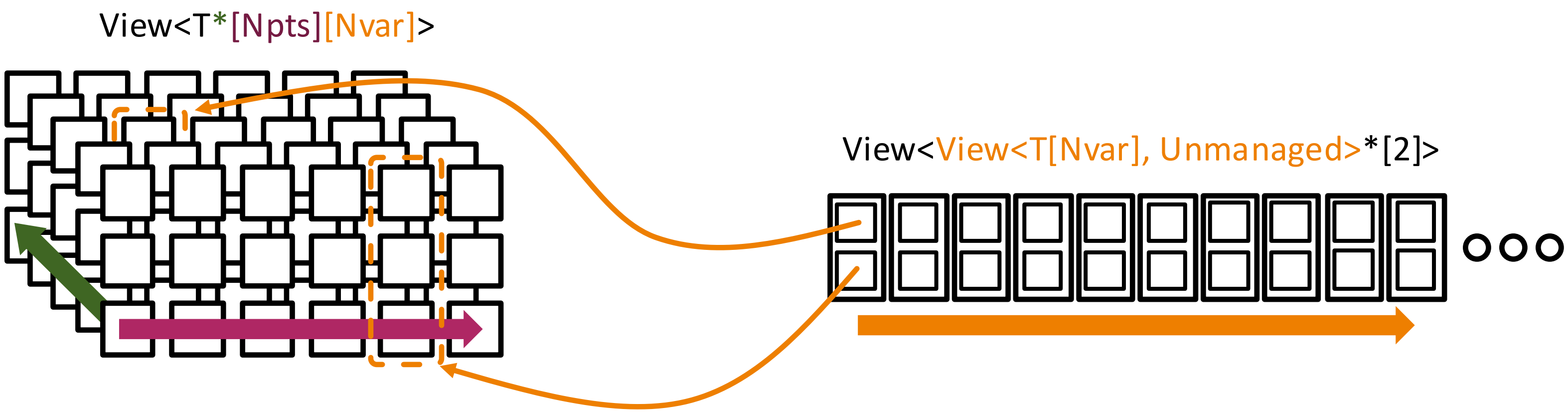
Objectives:

- Performance portability
- Hierarchical grids ⟹ AMR

First key successes:

- New data structures ⟹ ~1.5x faster than JAGUAR on CPUs + runs on GPUs
- Significantly more compact (~15K LoC)

HOPPS: implementation



Element-local kernels

- Process elements in parallel
- Extra parallelism within each element
- Structured memory access patterns

Interface kernels

- Process pairs of flux points in parallel
- Indirection table (main array + MPI buffers)
- Irregular memory accesses

Performance metrics

Reduced computational time:

$$R = T_{solve} \times \frac{N_{cu}}{N_{dof} N_{it}}$$

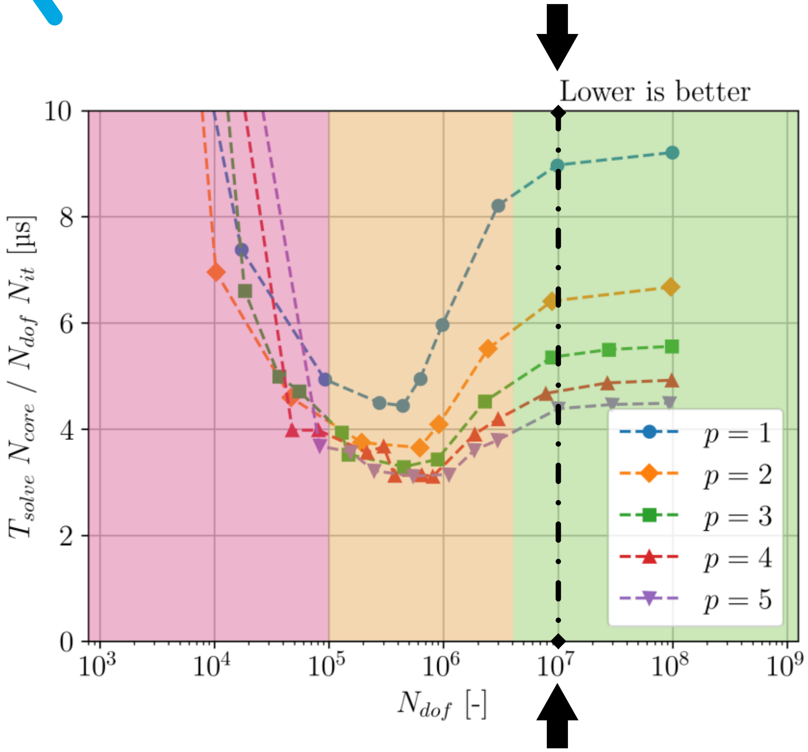
- N_{cu} is the number of compute units (CPU cores or GPUs)
- User oriented metric: "Cost per dof per iteration" [s]
- Depends on the temporal integration scheme (RK3SSP here)

Effective bandwidth:

$$B = 8 \times 10^{-9} \frac{M N_{dof} N_{it}}{T_{solve}}$$

- M is the number of memory accesses per dof per iteration
- Implementer oriented metric: "Bandwidth effectively leveraged by the solver" [GB/s]
- Directly comparable with HW spec

AMD Genoa CPU: RCT



Single node: bi-socket 2x96 cores

GCC 11.2 + OpenMPI 4.1.1 + Kokkos 4.3

Hybrid MPI+OpenMP setup: 1 MPI = 1 L3 cache

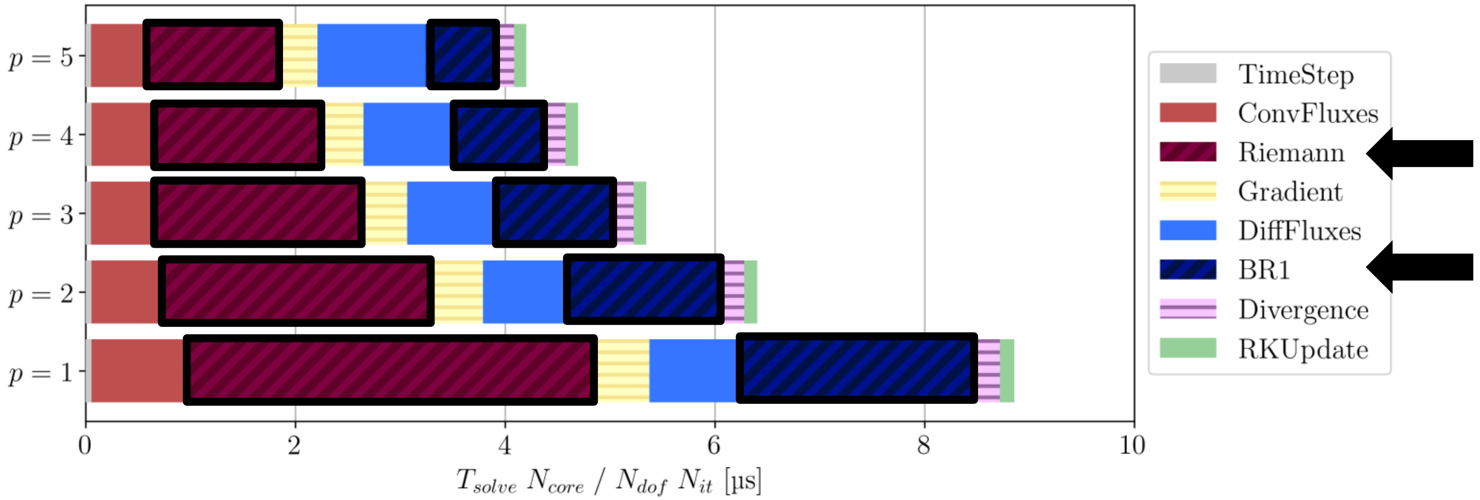
Higher p $\Longrightarrow$ better RCT

Small meshes: not enough data

Big meshes: flat curves as expected

In-between: everything fits in cache

AMD Genoa CPU: RCT@$N_{dof} = 10^7$

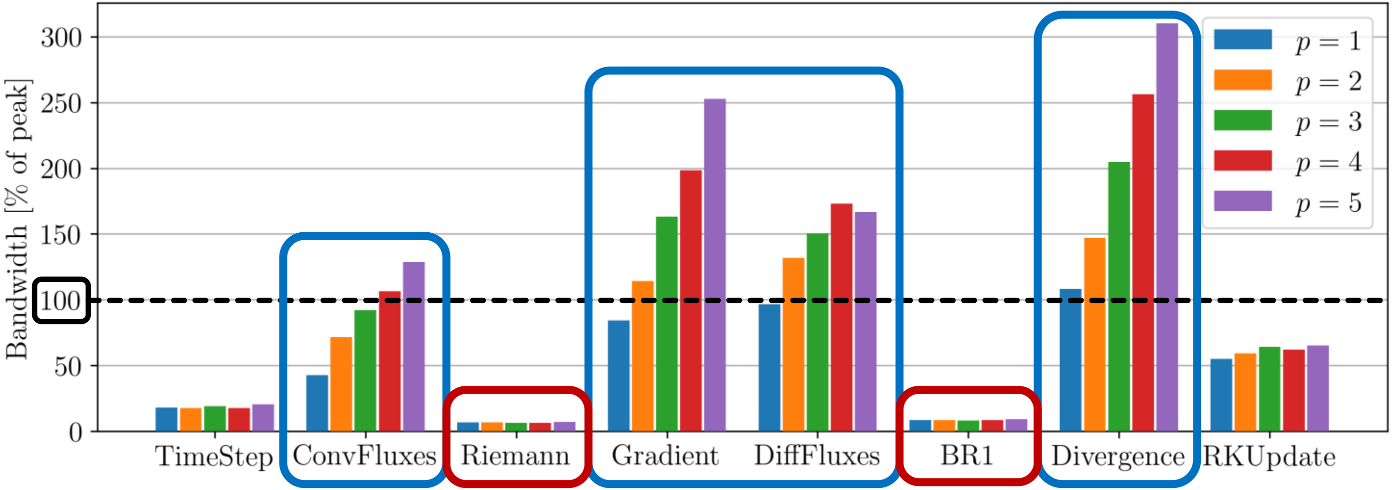


Interface kernels drive the RCT improvements

Turnaround time = $f(RCT, \Delta t)$

p	1	2	3	4	5
CFL limit	1.19	0.966	0.808	0.7	0.618
Fourier limit	1.12	0.63	0.4	0.25	0.18

AMD Genoa CPU: BW@$N_{dof} = 10^7$



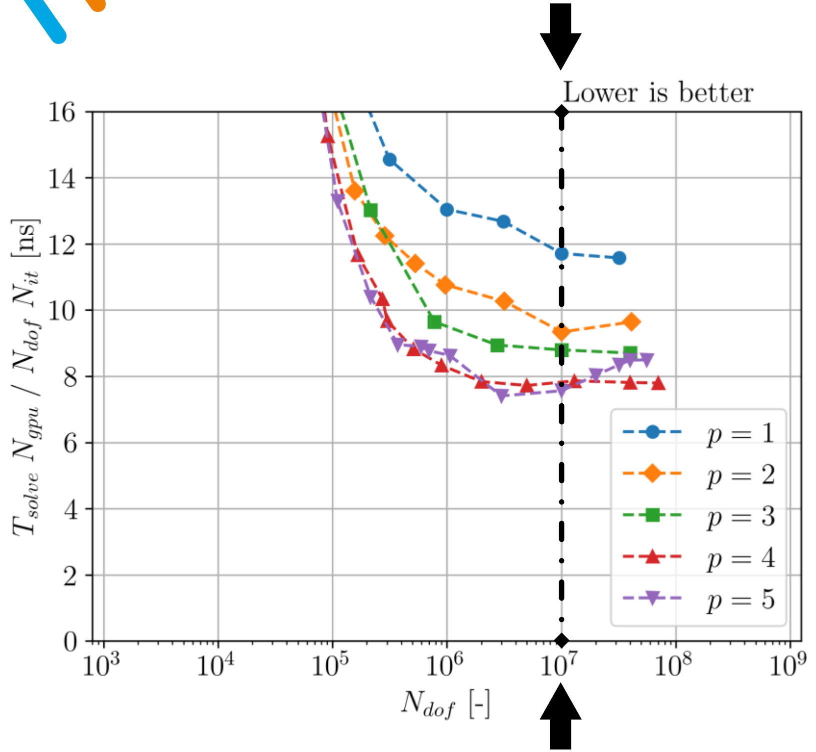
Stencil kernels

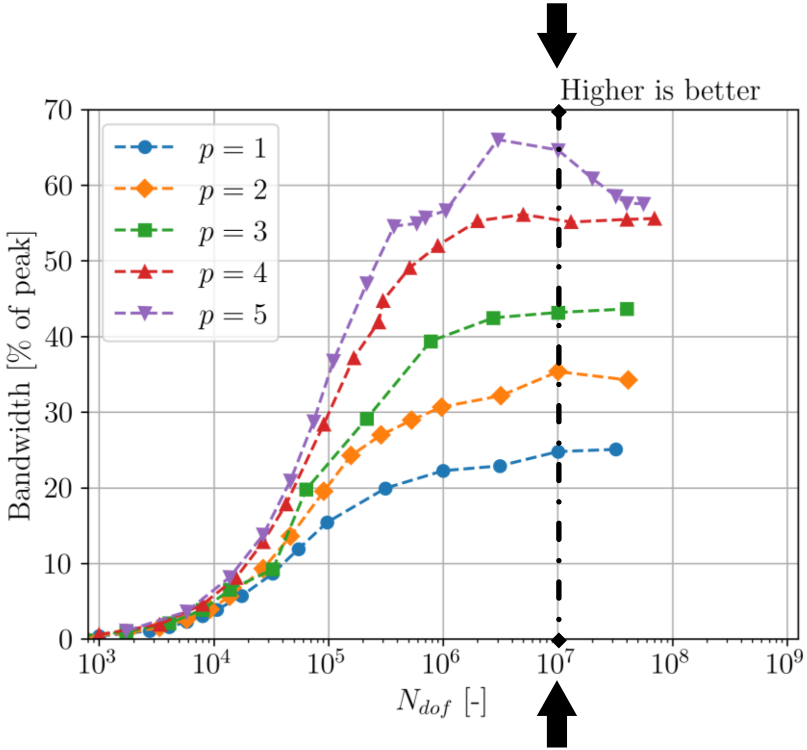
- More efficient at higher orders
- Increased cache hit rates

Interface kernels

- Always inefficient
- Latency-bound

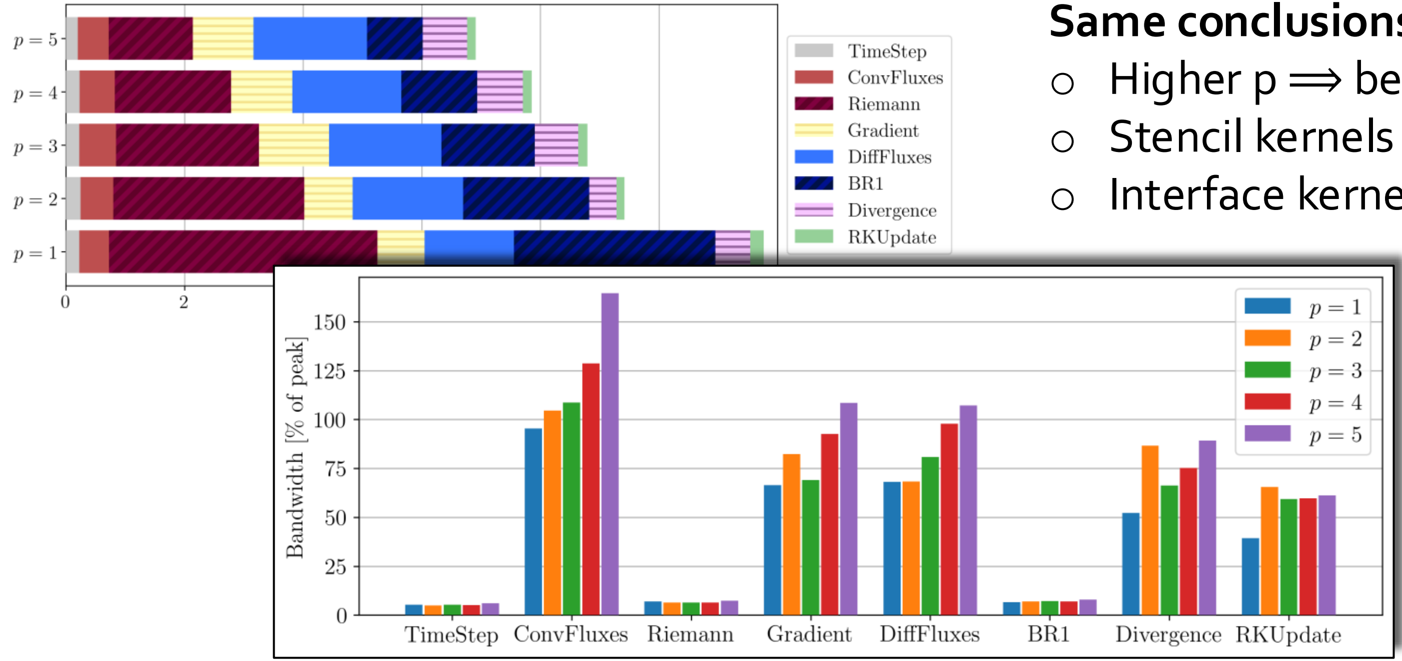
Nvidia GH200 GPU: RCT & BW





Single node GH200 superchip -- NVHPC 24.3 + Kokkos 4.3

Nvidia GH200 GPU: RCT & BW@$N_{dof} = 10^7$



Same conclusions on GPU:

- Higher p $\Longrightarrow$ better RCT
- Stencil kernels are efficient
- Interface kernels are not

Published paper

More HPC hardware

- 10 different archs (5 CPU + 5 GPU)
- All major vendors
- Different hardware generations

Strong & weak scalability analysis

- Tested up to 512 H100 GPUs
- Good scalability even with a naïve MPI implementation

International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Research Paper

The International Journal of High Performance Computing Applications
2025, Vol. 0(0) 1–22

DOI: 10.1177/10943420251384688
journals.sagepub.com/home/hpc
Sage

HOPPS: A performance portable spectral difference solver for high-fidelity computational fluid dynamics

Alexandre Dutka[1], Guillaume Daviller[1], Pierre Kestener[2,3] and Gabriel Staffelbach[4]

Abstract

High-fidelity computational fluid dynamics (CFD) enables the study of complex and subtle fluid dynamics phenomena, but remains to this day very computationally expensive. Therefore, being able to take advantage of all the raw compute power provided by high-performance computing (HPC) hardware evolutions such as the rise of GPU computing is key to making high-fidelity CFD more affordable. However, considering the diverse and fast-evolving HPC hardware landscape, long-term sustainability and software maintainability can rapidly be compromised. The use of adequate numerical methods is also key to reduce the computational cost, and discontinuous high-order methods which combine geometric flexibility and efficient hardware use in an increasingly bandwidth-bound HPC landscape, are very promising in this regard. This work reports the implementation of such a high-order CFD solver using the open source library Kokkos to address the performance portability and sustainability issues. Performance is investigated over a broad range of CPU and GPU architectures, demonstrating the relevance of the approach. This work also highlights the fitness of the chosen numerical method to achieve high orders of accuracy without compromising performance nor scalability.

Keywords

high-order methods, high-performance computing, GPU computing, performance portability, sustainable programming

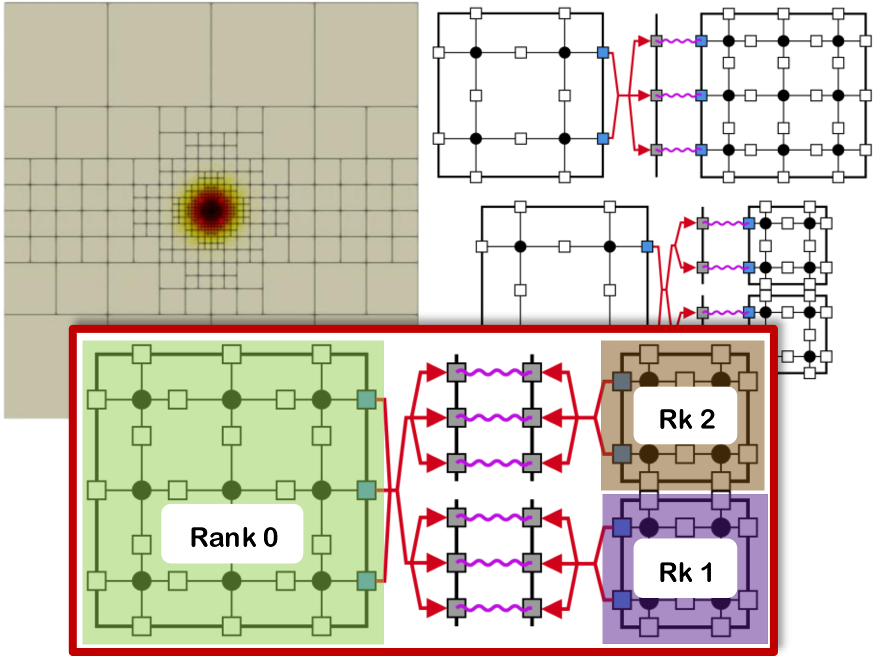
Ongoing work: hp-refinement

p-adaptation implemented

- Adjust the order of each element
- Requires some development effort

h-adaptation is a WIP

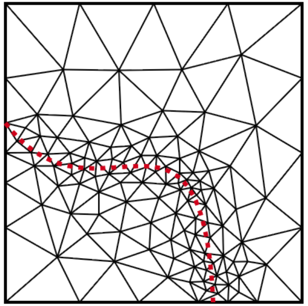
- Refine (recursively) each element
- Significantly more challenging
- MPI interop. is hard to get right
- Heavily relies on the t8code library



Adaptive Mesh Refinement

Conformal approach

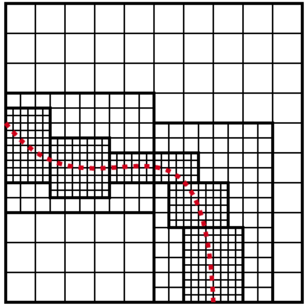
Unstructured



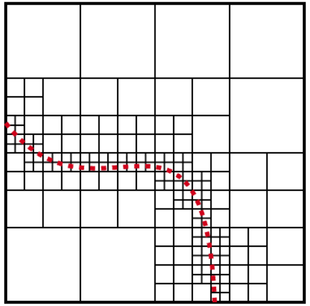
- Simplex grid solvers
- Expensive remeshings

Non-conformal approach

Patch-based



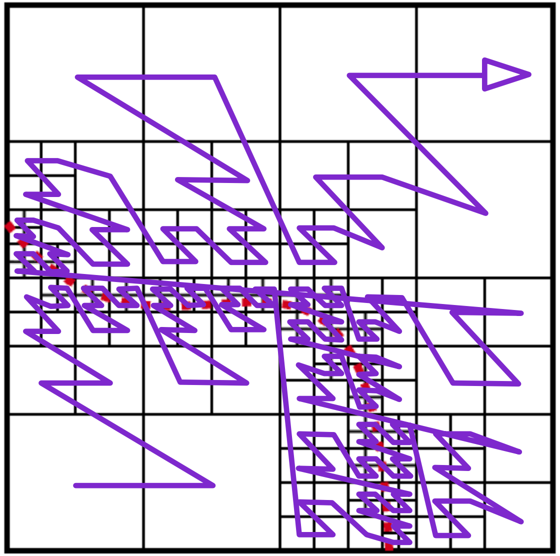
Tree-based



- Popular for in cartesian frameworks
- Hierarchical discretization approach

$$\text{Compression ratio} = \frac{\text{dof count}}{\text{dof count with uniform grid}}$$

Tree-based AMR



Efficient & scalable algorithms

- Elements stored linearly based on a space-filling curve
- Partitioning is trivial $\Longrightarrow$ just cut the SFC
- Fast deterministic partition-independent adaptation
- Better compression ratios than with patch-based AMR

Works for cartesian & unstructured grids

- Forest approach enables body-fitted frameworks
- Not limited to a specific element type

Increased solver complexity

- Non-conformal interfaces are challenging
- Hanging-node elimination is a research topic

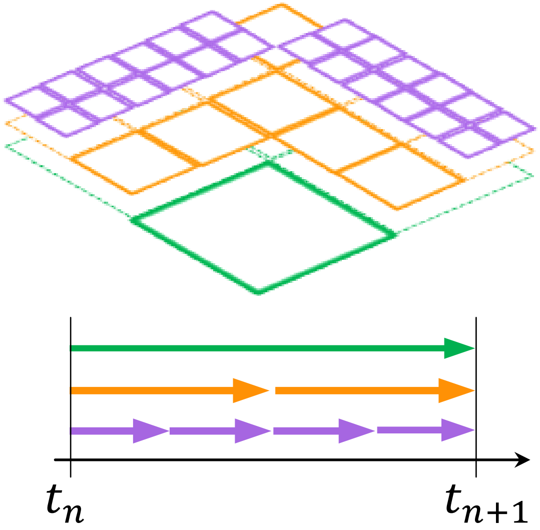
Perspectives

Use compact diffusive schemes (BR2 & co.)

- Reduced stencil $\Longrightarrow$ improved stability
- Kernel fusion opportunities

Investigate hierarchical time-stepping

- A.k.a. sub-cycling, a.k.a. local-time stepping
- Advance each refinement level at its own pace
- More complexity at the interfaces
- Massive speedup potential for some applications



Final thoughts

Great performance portability results across various archs & toolchains

Kokkos rewrites are significantly more compact (esp. compared to legacy Fortran)

⟹ Maintainability & sustainability

KokkosComm for interop. with MPI

HDF5 interop. is also a need
⟹ KokkosIO side-project in the future?

`Kokkos::View<T*[N]> array{…}` ⟹ the spelling of arrays is confusing for newcomers

"`array[a:b,:]`" is spelled `Kokkos::subview(array, std::pair<int, int>{a, b}, Kokkos::ALL)` (verbose syntax) + Slicing bumps the ref. count ⟹ need for an easier way to create unmanaged views

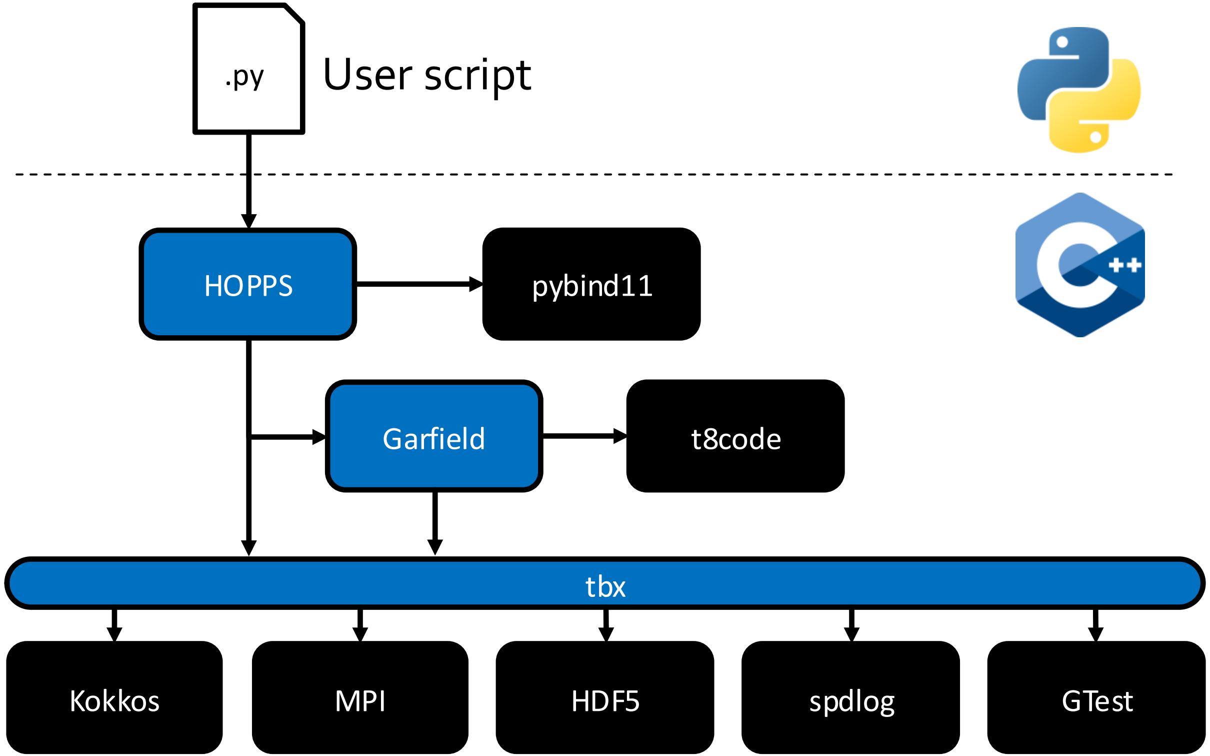
The `View< View<T*, Unmanaged>* >` pattern is super-useful ⟹ maybe it should be a built-in vocabulary type?

CENTRE EUROPÉEN DE RECHERCHE ET DE FORMATION AVANCÉE EN CALCUL SCIENTIFIQUE

Questions?

@dutkalex
adutka@cerfacs.fr

HOPPS: structure



Solver library

Mesh library

Core utils library

Strong scaling H100 GPUs p=5

Naive MPI implementation: no compute/comm overlapping