Heterogeneous Computing with a Homogeneous Codebase

F.D. Witherden, **P.E. Vincent**

Department of Aeronautics
Imperial College London

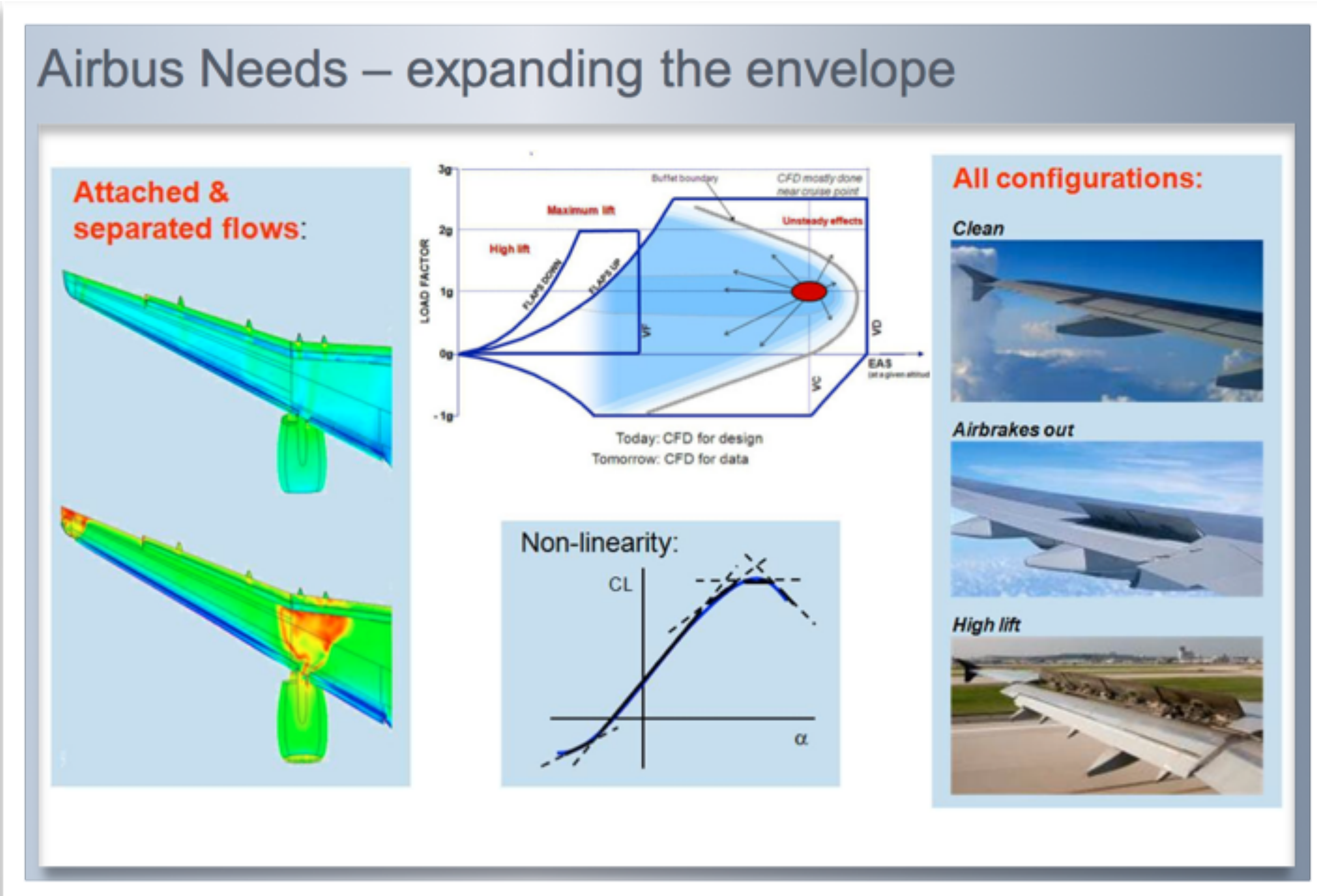
About Me

- Final year PhD candidate under the supervision of **P. Vincent** and **S. Sherwin**.
- Interested in the efficient implementation and application of **flux reconstruction schemes** to the **compressible Navier-Stokes** equations.
- Lead developer of **PyFR**.

Overview

- Motivation.
- Heterogeneous computing;
    - “what?”, “why?”, and “how?”.
- PyFR.
- Summary.

Motivation



[1] Murray Cross, Airbus, Technology Product Leader - Future Simulations (2012)

Motivation

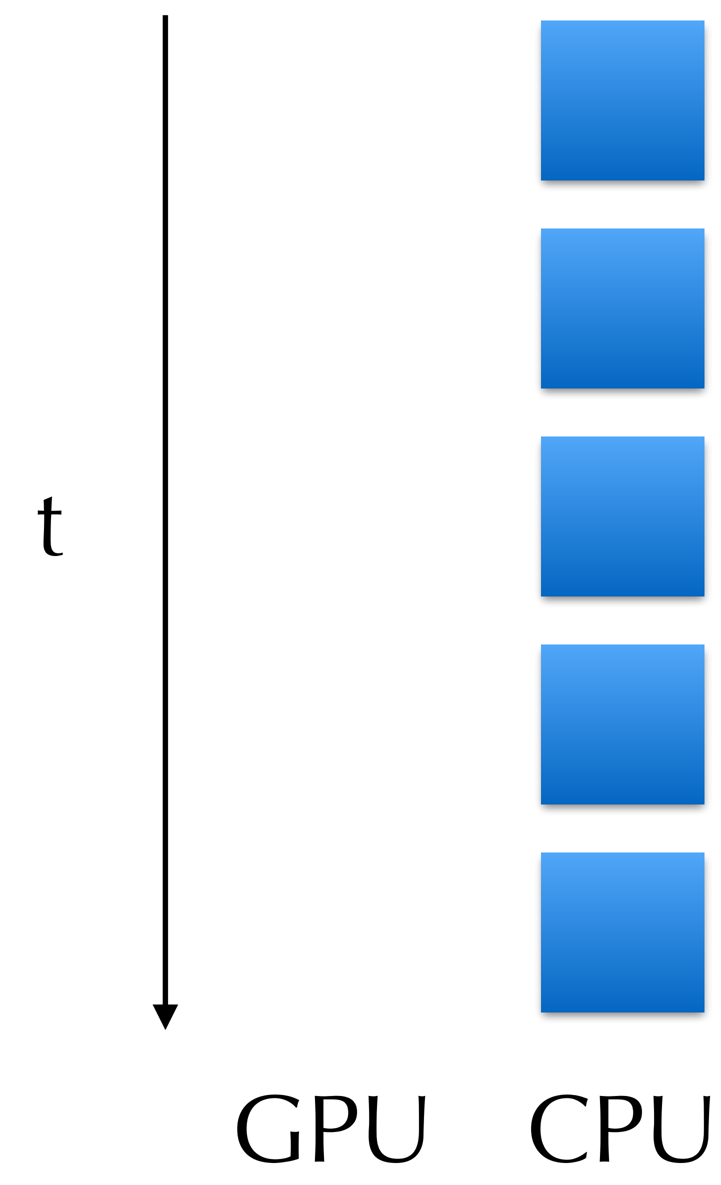
- Objective is to advance **industrial CFD** capabilities from their current **RANS plateau**.
- But…
  - unsteady simulations are **more expensive**;
  - so **every FLOP counts**!

Heterogeneous Computing

- Lots of **overly broad** definitions.
- Let us therefore start by defining what we mean in the context of **scientific computing**.
- We shall do this by **code classification**.

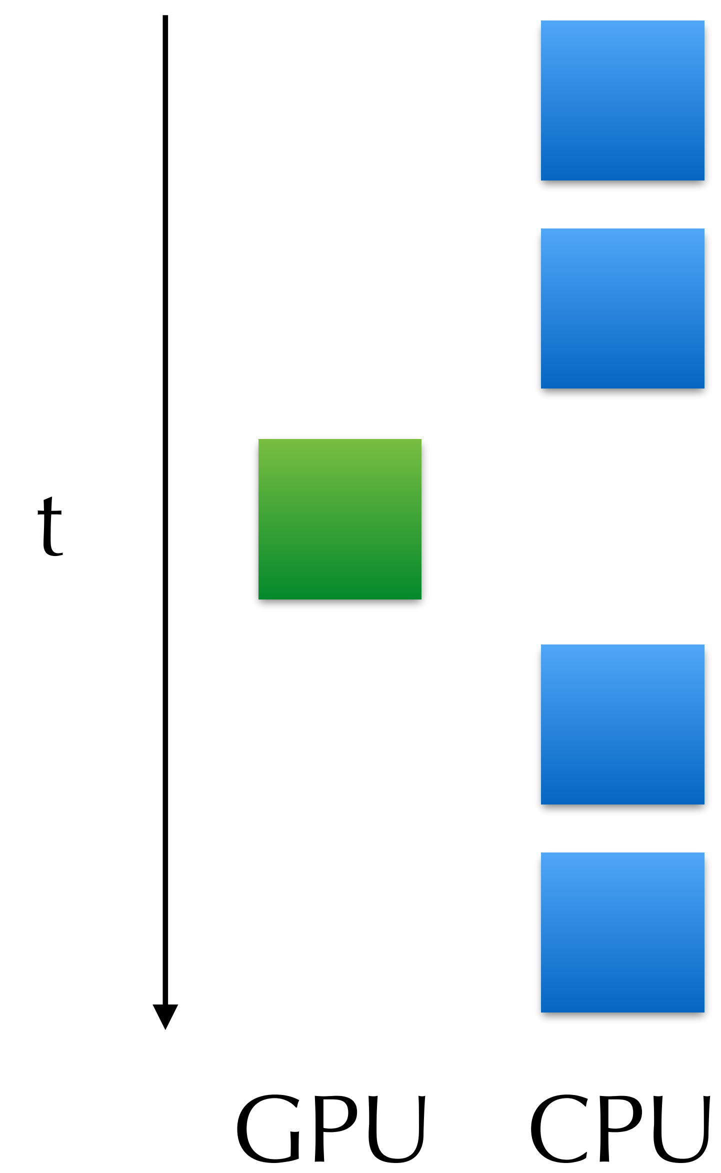
Code Classifications I

- Traditional code.
- Parallelised with **MPI**.
- May be **hybrid** using both MPI and OpenMP.



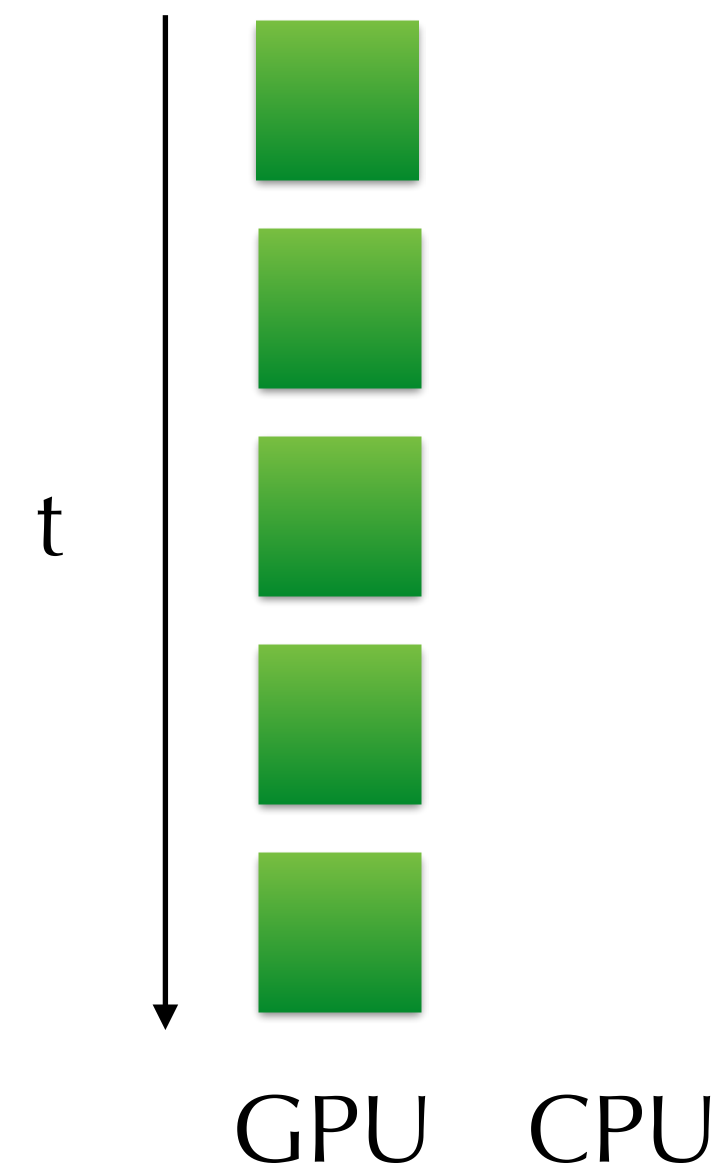
Code Classifications II

- Partially accelerated code.
- Some regions **offloaded**.
- Data movement overheads can be significant.
- Speedup limited by **Amdahl's law**.



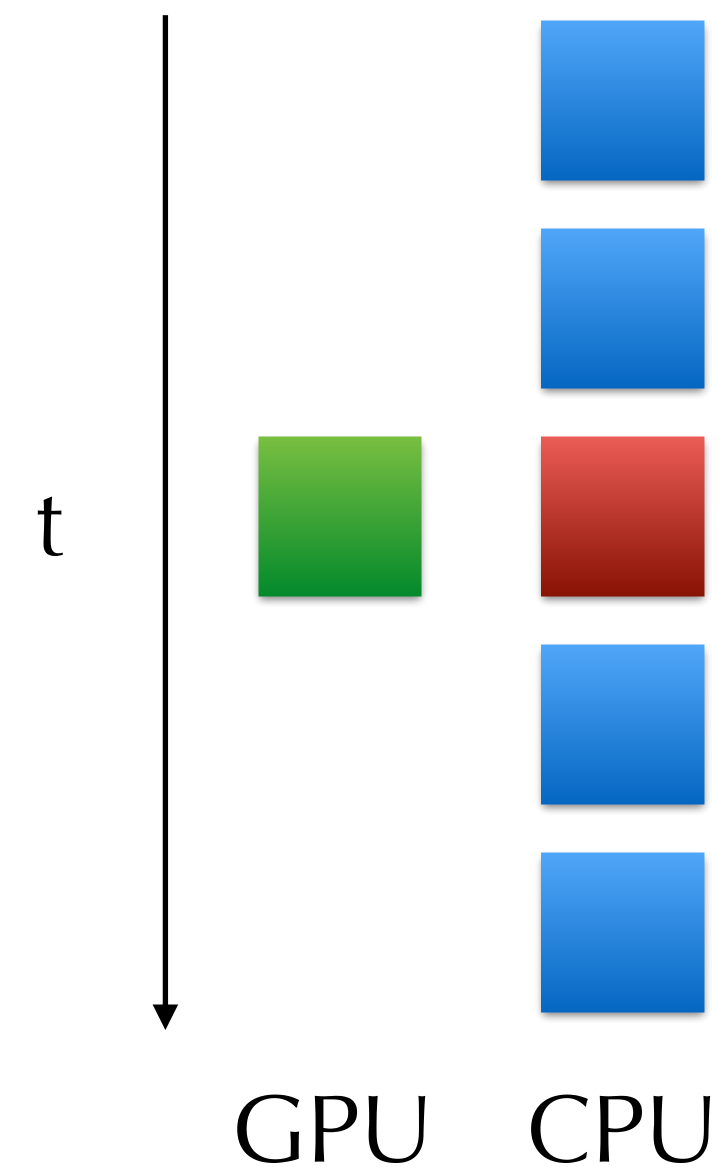
Code Classifications III

- Fully accelerated code.
- All kernels **on-device**.
- Minimal overheads.
- CPU cores mostly idle.



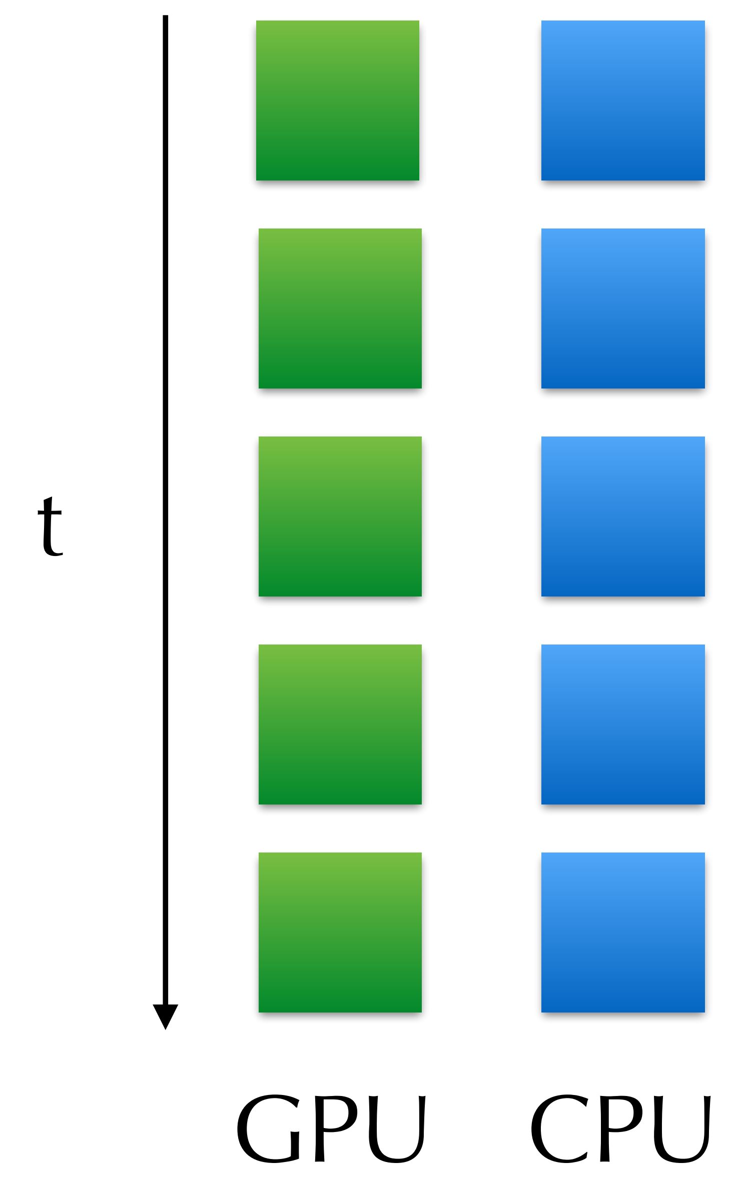
Code Classifications IV

- Heterogeneous code.
- CPU and GPU perform **different operations**.
- Improved utilisation.
- Sensitive to cluster configuration.



Code Classifications V

- Fully heterogeneous code.
- CPU and GPU perform **identical operations**.
- Domain decomposed with appropriate weighting factors.

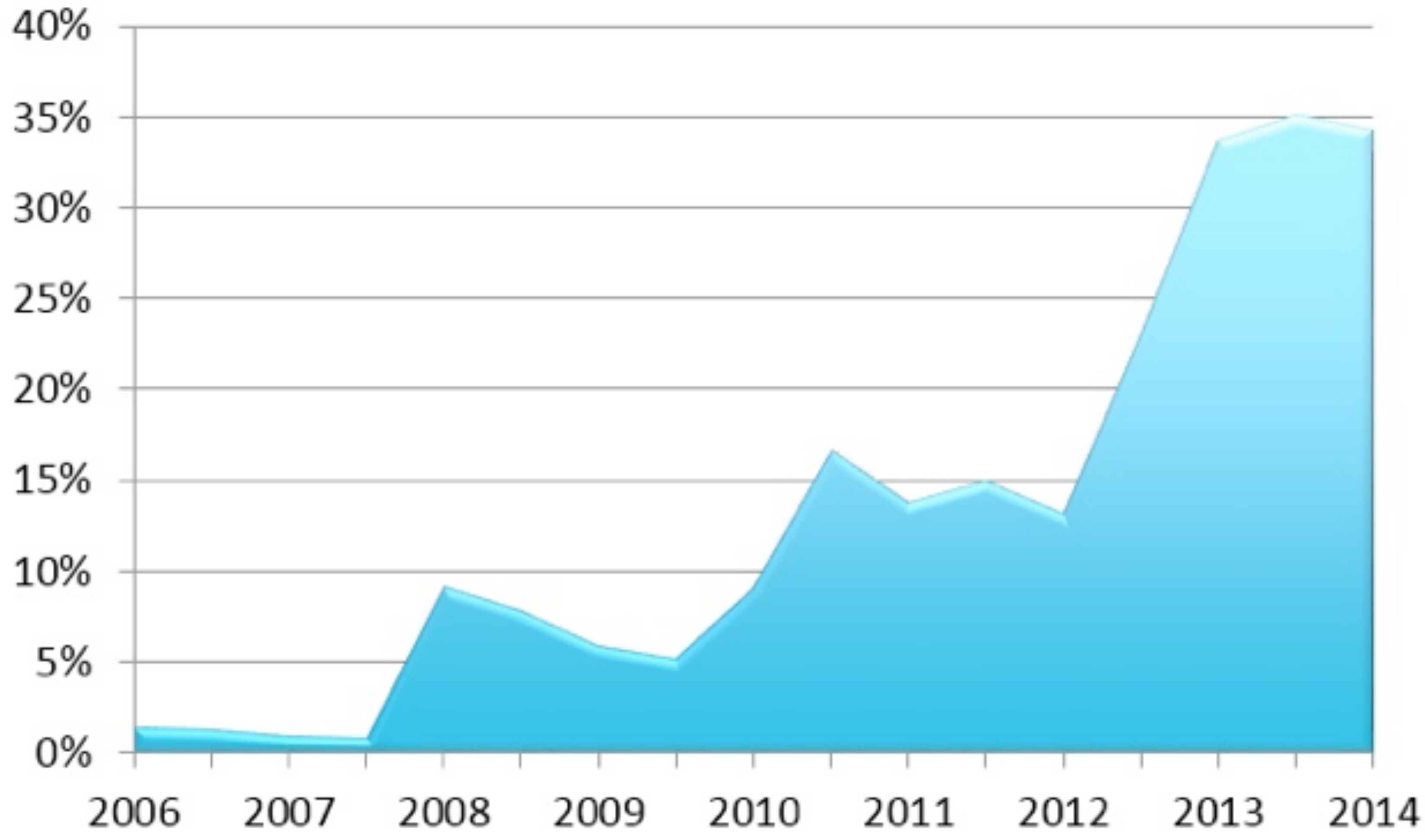


Our Focus

- We are interested in **category V** codes.
- However, they require **time to develop and expertise to maintain**.
- So…"**Why should this take priority over adding new physics?**"

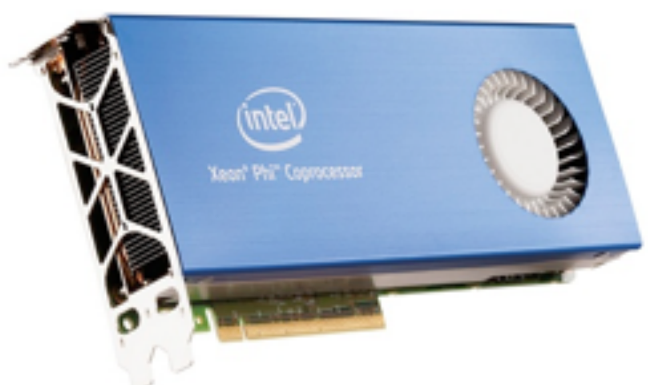
Accelerator Adoption

- FLOPS contributed by accelerators to the **TOP500**. [HPCwire]



Accelerator Adoption

- Within the **top ten**:

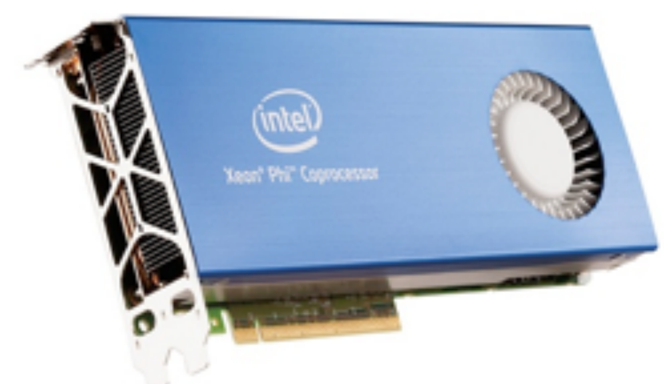


Intel Xeon Phi
2 of 10

NVIDIA Tesla
3 of 10

Increasing Heterogeneity

- Consider **Stampede** at **TACC**.
- Currently **#7** on the TOP500 list.

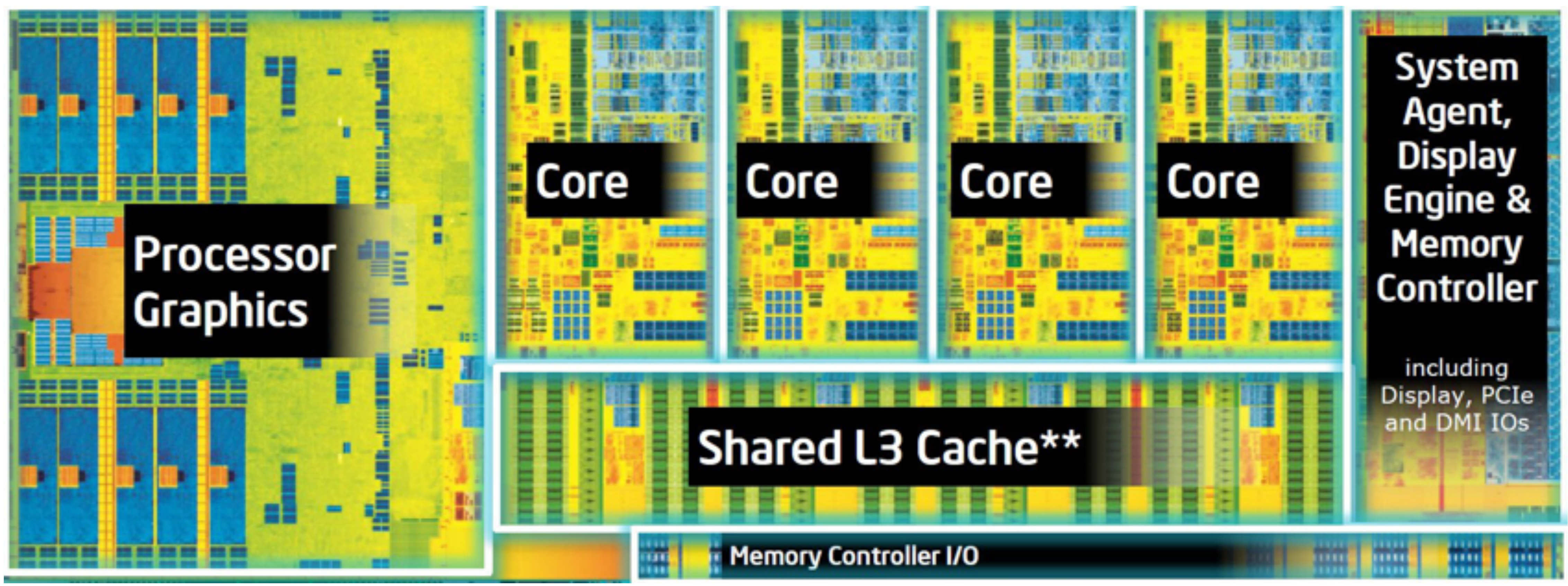


Intel Xeon CPUs
2.2 PFLOP/S

Intel Xeon Phis
7.4 PFLOP/S

Increasing Heterogeneity

- …and it is not just the high end.

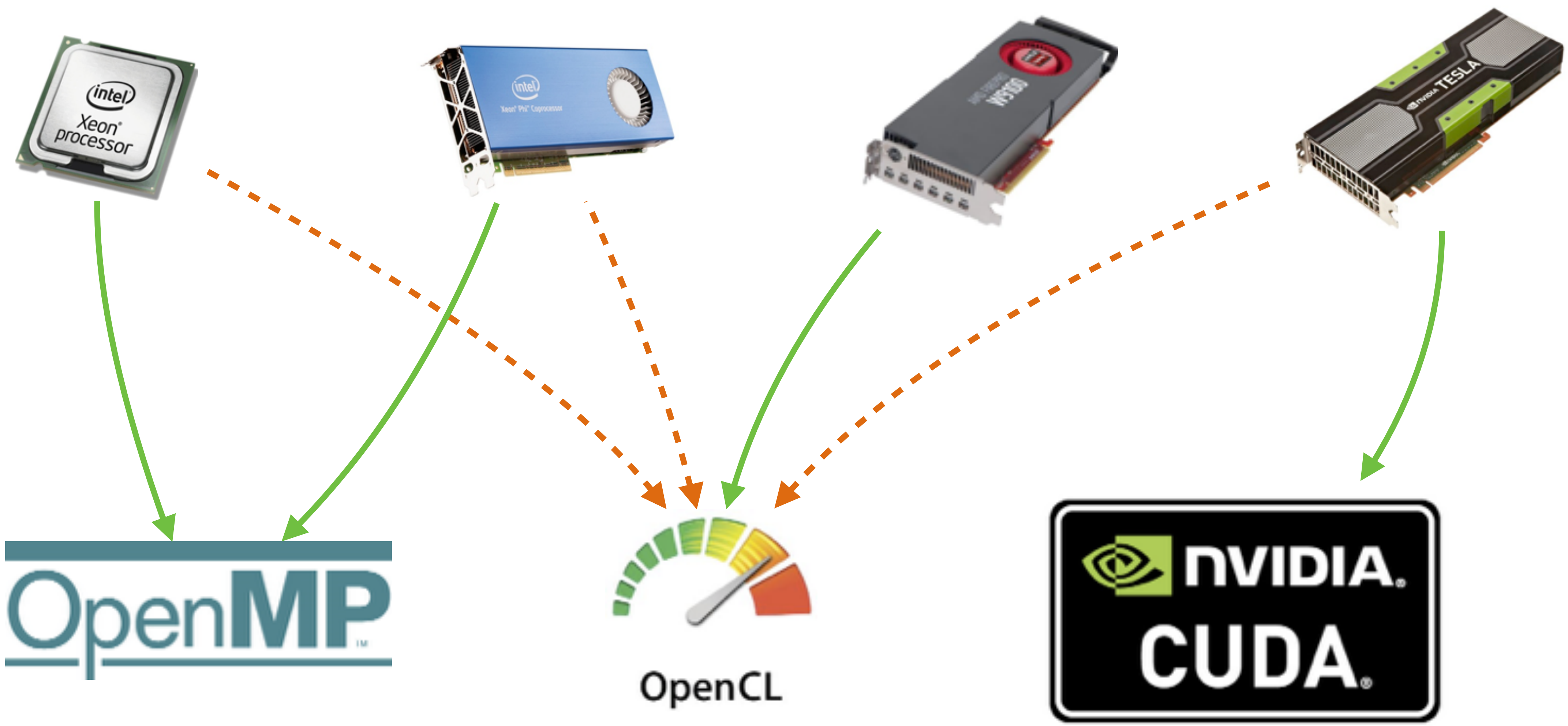


Writing A Category V Code

1. Make your code **performance portable**.
2. Use a **consistent MPI exchange format**.
3. There is no step three.

Performance Portability

- It is a challenging environment…

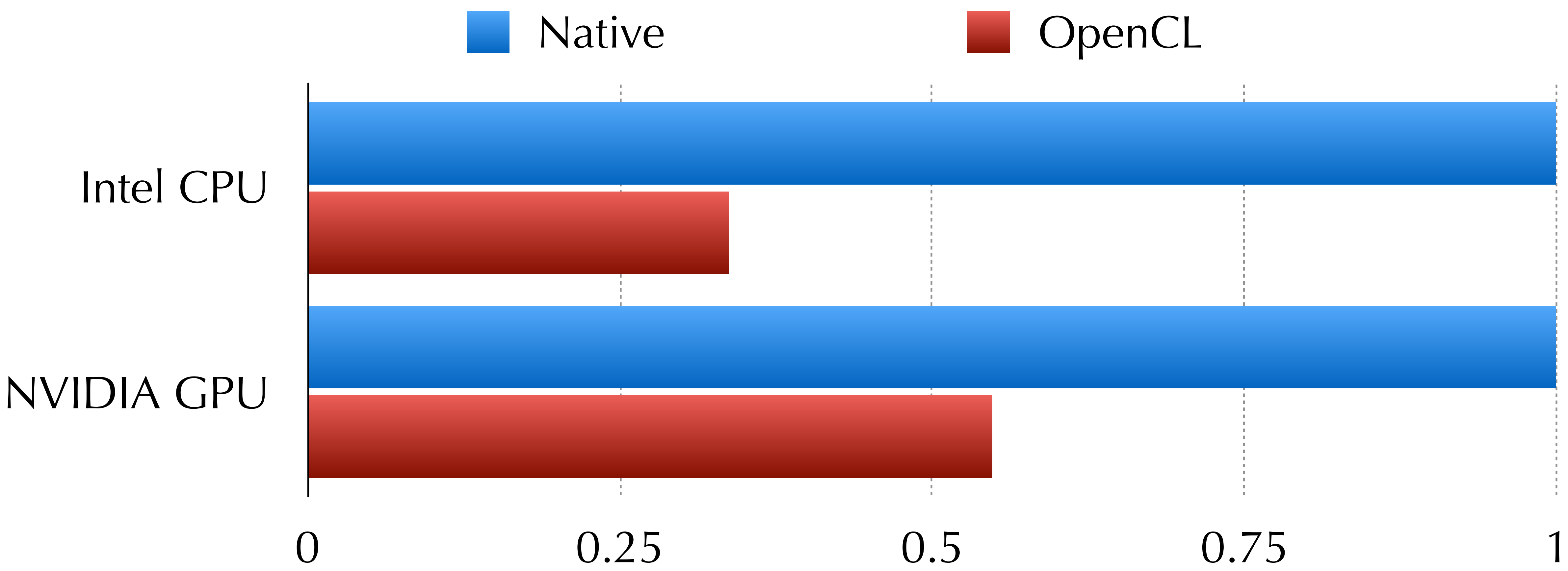


Performance Portability

- …so what about OpenCL?
- Supported **everywhere**; but optimal almost **nowhere**.
- Sacrifice the ability to call native libraries;
    - cuBLAS, MKL, ….

Performance Portability

- OpenCL vs Native for a S/DGEMM dominated code.



PyFR

- Our solution **PyFR**.
- Written in **Python**.
- Uses flux reconstruction to solve the **Navier-Stokes** equations on mixed unstructured grids in **2D/3D**.
- **Performance portable** across a variety of hardware platforms.

PyFR

- Python outer layer.

Python Outer Layer
(**Hardware Independent**)

- Setup
- Distributed memory parallelism
- Outer ‘for’ loop and calls to **Hardware Specific Kernels**

PyFR

- Need to generate **hardware specific kernels**.

Python Outer Layer
(**Hardware Independent**)

- Setup
- Distributed memory parallelism
- Outer 'for' loop and calls to **Hardware Specific Kernels**

PyFR

- In FR **two types** of kernel are required.

Python Outer Layer
(**Hardware Independent**)

- Setup
- Distributed memory parallelism
- Outer ‘for’ loop and calls to **Hardware Specific Kernels**

Matrix Multiply Kernels

- Data interpolation/ extrapolation etc.

Point-Wise Nonlinear Kernels

- Flux functions, Riemann solvers etc.

PyFR

- Matrix multiplications are quite simple.

Python Outer Layer
(**Hardware Independent**)

- Setup
- Distributed memory parallelism
- Outer ‘for’ loop and calls to **Hardware Specific Kernels**

Matrix Multiply Kernels

- Data interpolation/ extrapolation etc.

Use GEMM from vendor supplied BLAS

Point-Wise Nonlinear Kernels

- Flux functions, Riemann solvers etc.

PyFR

- Harder for point-wise nonlinear kernels.

Python Outer Layer
(**Hardware Independent**)

- Setup
- Distributed memory parallelism
- Outer 'for' loop and calls to **Hardware Specific Kernels**

Matrix Multiply Kernels

- Data interpolation/ extrapolation etc.

Use GEMM from vendor supplied BLAS

Point-Wise Nonlinear Kernels

- Flux functions, Riemann solvers etc.

Pass templates through Mako derived templating engine

PyFR

- These can now be called.

Python Outer Layer
(**Hardware Independent**)

- Setup
- Distributed memory parallelism
- Outer ‘for’ loop and calls to **Hardware Specific Kernels**

Matrix Multiply Kernels

- Data interpolation/ extrapolation etc.

Point-Wise Nonlinear Kernels

- Flux functions, Riemann solvers etc.

C/OpenMP
Hardware Specific Kernels

CUDA
Hardware Specific Kernels

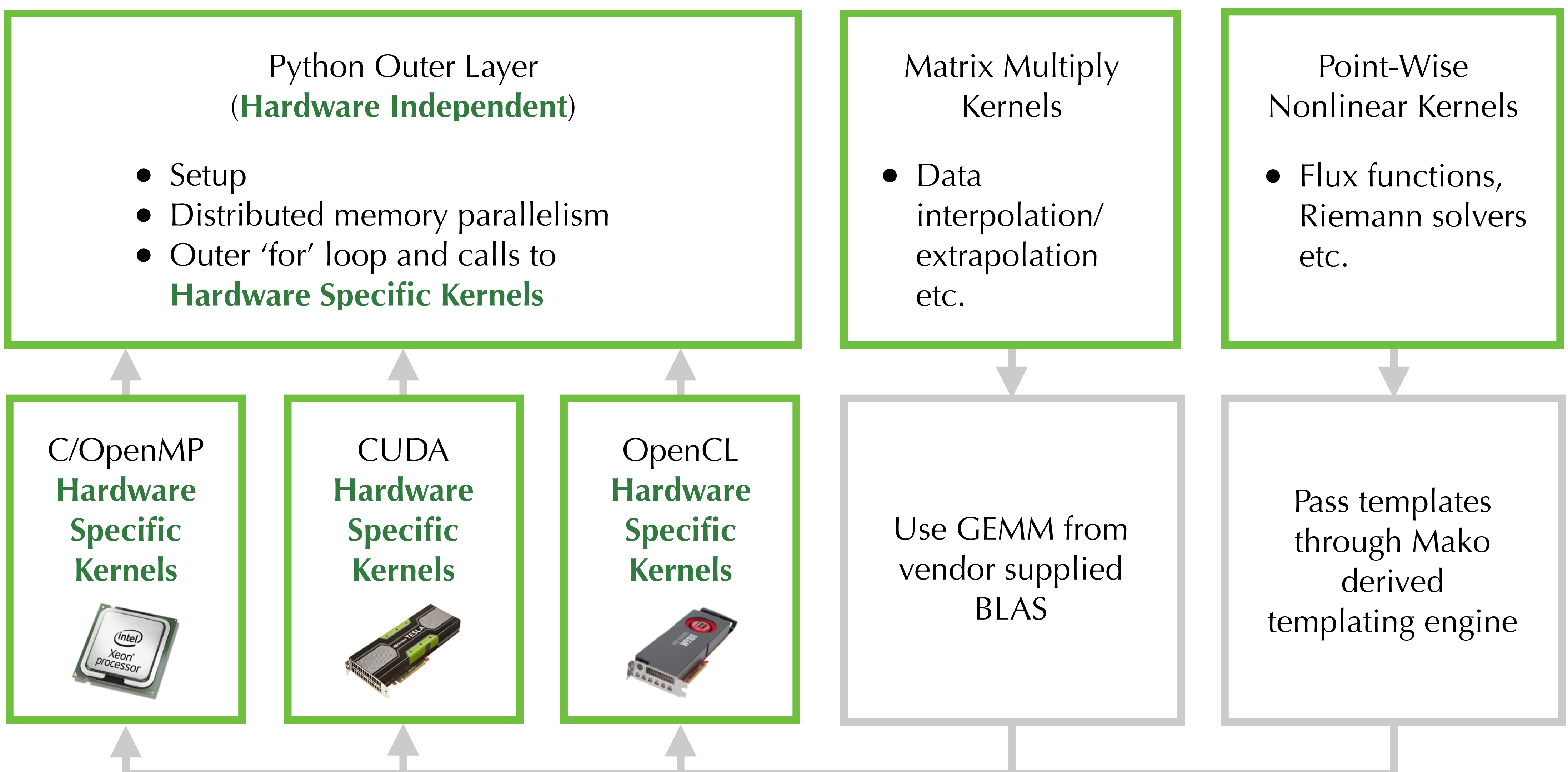
OpenCL
Hardware Specific Kernels

Use GEMM from vendor supplied BLAS

Pass templates through Mako derived templating engine

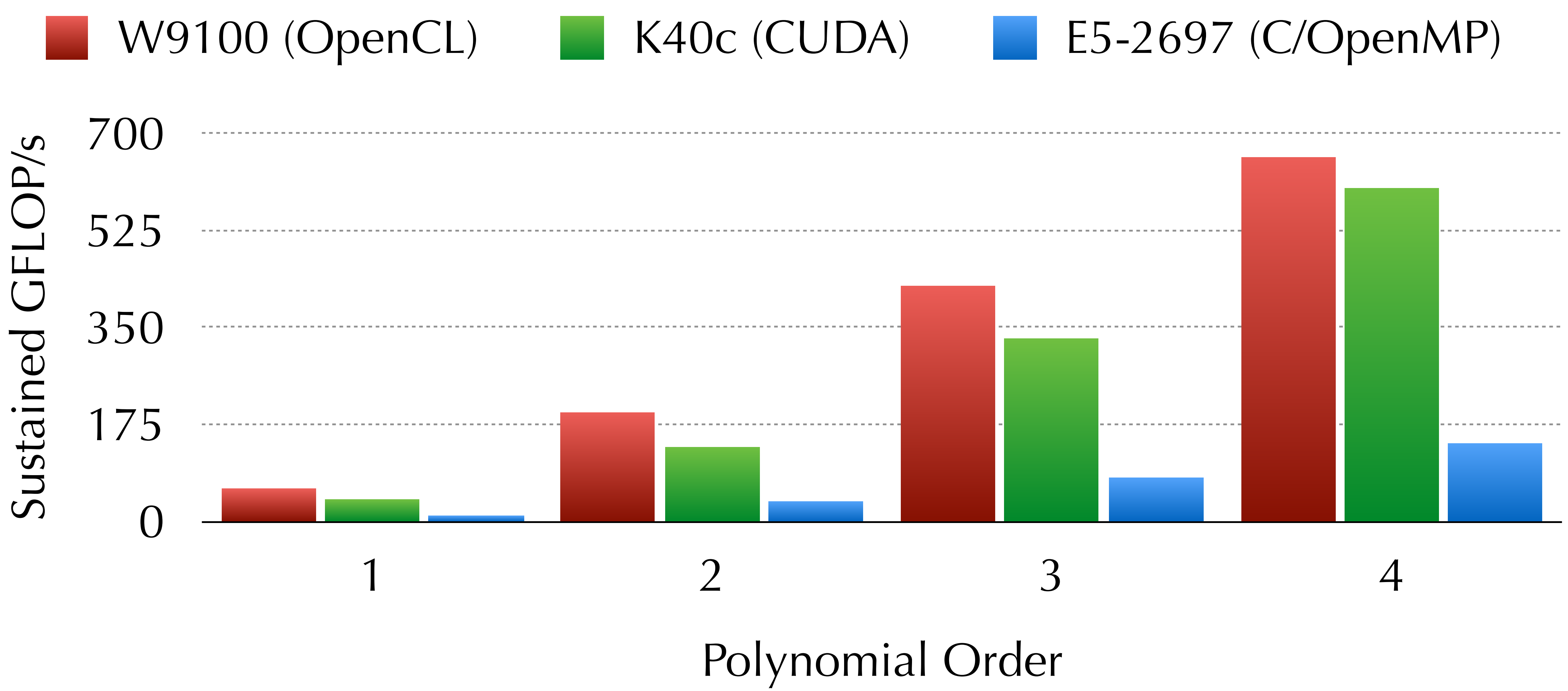
PyFR

- These can now be called.



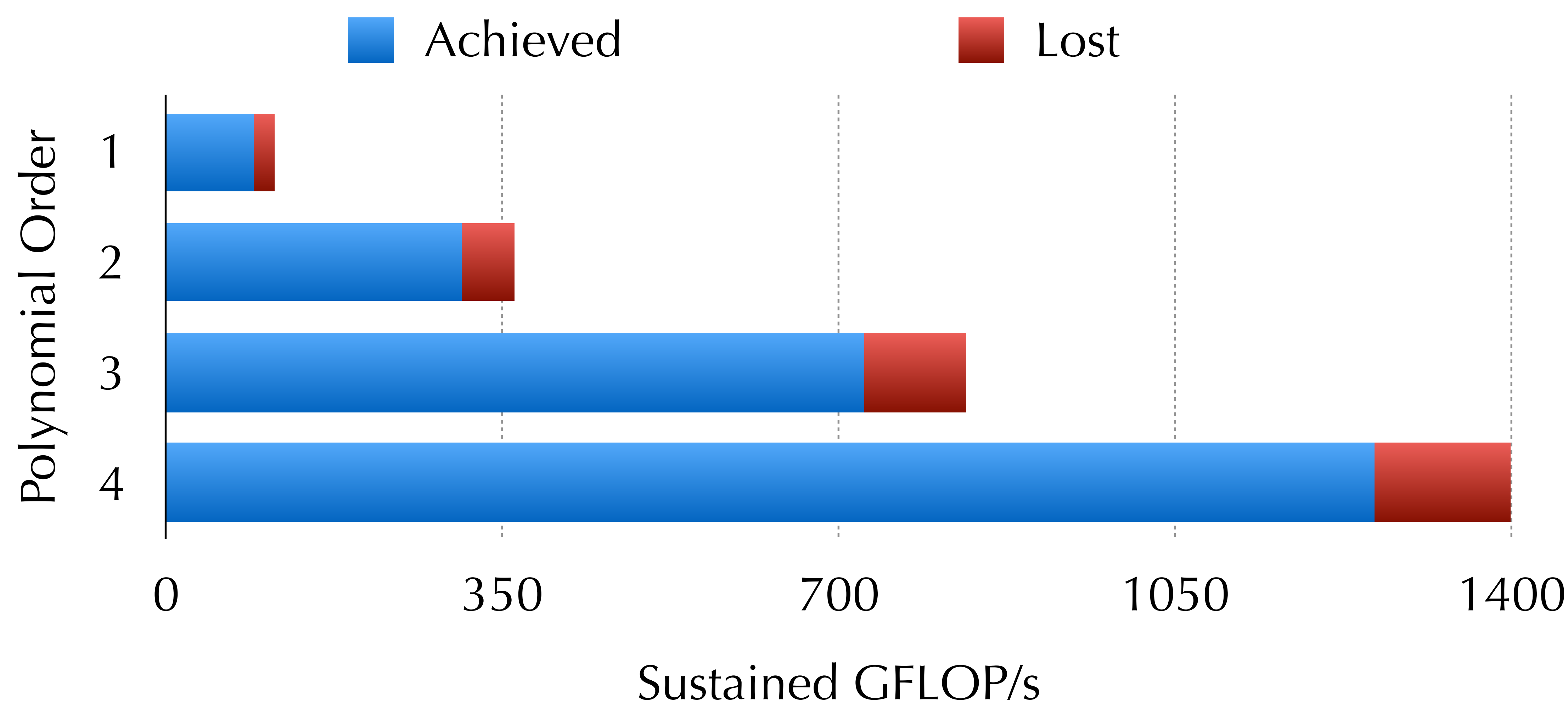
PyFR Results I

- **Single node** performance on a prism/tet mesh.



PyFR Results II

- **Multi-node heterogeneous** performance on the same mesh.



Summary

- Funded and supported by

EPSRC Engineering and Physical Sciences Research Council

NVIDIA

intel

AMD

- Any questions?
- E-mail: **freddie.witherden08@imperial.ac.uk**