EXAFLOW

ENABLING EXASCALE FLUID DYNAMICS SIMULATIONS

Michèle Weiland

m.weiland@epcc.ed.ac.uk

THE UNIVERSITY of EDINBURGH

epcc

ExaFLOW: key facts

- EU-funded project, 3 years, started October 2015
- www.exaflow-project.eu

→ Address current **algorithmic** bottlenecks for **Exascale** to enable the use of **accurate** CFD codes for problems of **practical engineering** interest

KTH VETENSKAP OCH KONST
ROYAL INSTITUTE OF TECHNOLOGY

Imperial College London

UNIVERSITY OF Southampton

Universität Stuttgart

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

McLaren

asc(s) Automotive Simulation Center Stuttgart

epcc

Project objectives

- **Adaptive mesh refinement** in complex computational domains
- **Solver efficiency**
  - e.g. mixed discontinuous/continuous Galerkin methods, appropriate optimised preconditioners
- Strategies to ensure **fault tolerance** and resilience
- **Heterogeneous modelling** to allow for different solution algorithms in different domain zones
- Improved **I/O** for large data volumes
- **Efficiency** ← EPCC's focus
  - Minimise energy/power consumption and time to solution
    → remove inefficiencies in implementations and system setup

epcc

Power and energy consumption of CFD Test Cases

Michael Bareford, Nick Johnson, and Michèle Weiland.
On the trade-offs between energy to solution and runtime for real-world CFD test-cases.
2016. DOI: http://dx.doi.org/10.1145/2938615.2938619

epcc

ARCHER

- Cray XC30 MPP, 4920 Compute Nodes
  - Dual Intel Xeon processors (Ivy Bridge), 24 cores, 64 GB
- Tests conducted on 2-cabinet *Test Development Server*
  - Private to EPCC, minimises resource contention

Cray XC30 Power Management Counters

Running Average Power Limit Counters	Power Management Counters
PACKAGE_ENERGY (nJ)	PM_POWER:NODE (W)
DRAM_ENERGY (nJ)	PM_ENERGY:NODE (J)
PP0_ENERGY (nJ)	PM_FRESHNESS

- PACKAGE = processor
  - two sets of RAPL counters per node
- Power **instantaneous**, energy **cumulative**

PAT MPI Library
(https://github.com/cresta-eu/pat_mpi_lib)

- Acts as a wrapper that simplifies monitoring a user-defined set of hardware performance counters during the execution of a MPI code running across multiple compute nodes
- Controls which MPI processes read counters (one per node for PM counters, two per node for RAPL).

```
CALL pat_mpi_open(out_fn)
DO i=1,nstep
  CALL pat_mpi_monitor(i,1)
  ...
  ! application code...
  ...
  CALL pat_mpi_monitor(i,2)
ENDDO
CALL pat_mpi_close()
```

- Only one MPI process (e.g., rank 0) collates the data, writing it to a single file.

epcc

Code 1: Nektar++

Nektar++ v4.2.0 (MPI)

http://www.nektar.info

Imperial College London

An open-source spectral element code that combines the accuracy of spectral methods with the geometric flexibility of finite elements, specifically, *hp*-version FEM

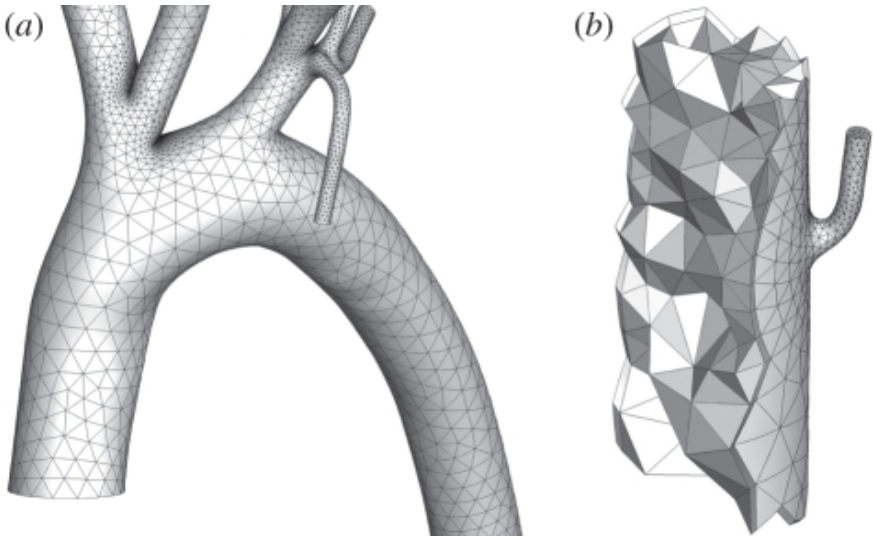
Supports several scalable solvers for many sets of partial differential equations, from (in)compressible Navier-Stokes to the bidomain model of cardiac electrophysiology

Nektar ++ test case

Aorta Blood Flow

Unsteady diffusion equations with a continuous Galerkin projection

Aortic arch mesh



Vincent, Plata, Hunt et al.,
J R Soc Interface. 2011

Resources

compiler: cray
nodes: 4
runs: 10

runtime: ~16 min
time steps: 4000

epcc

Code 2: SBLI

SBLI v4.2.0 (MPI)

http://www.southampton.ac.uk/engineering/research/projects/sbli_computer_code.page

SBLI is a high-order fully parallelised finite difference code that solves the full 3D compressible Navier-Stokes equations

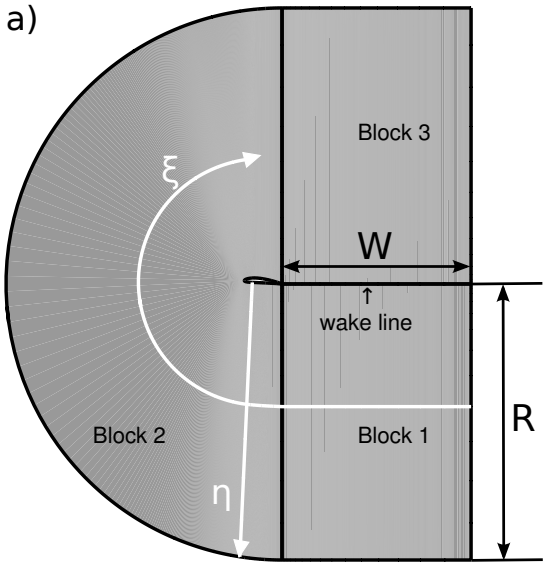
The code is designed for large eddy simulations of transitional and turbulent flow

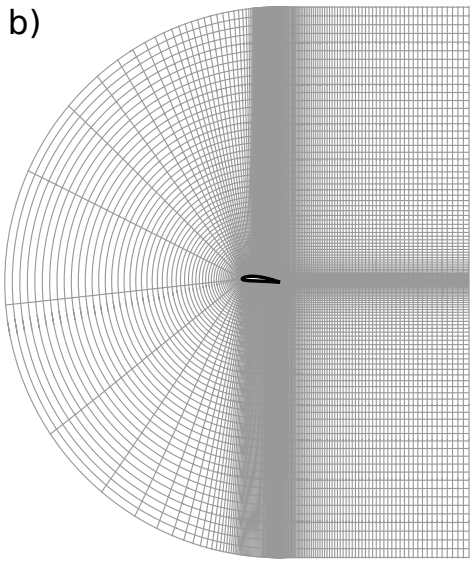
Actual code used is a *customisation* of v4.2.0 that includes a particular treatment for trailing airfoil edge

SBLI test case

NACA4412 Airfoil Simulation (compressible air flow)

http://library.propdesigner.co.uk/html/naca_4412.html



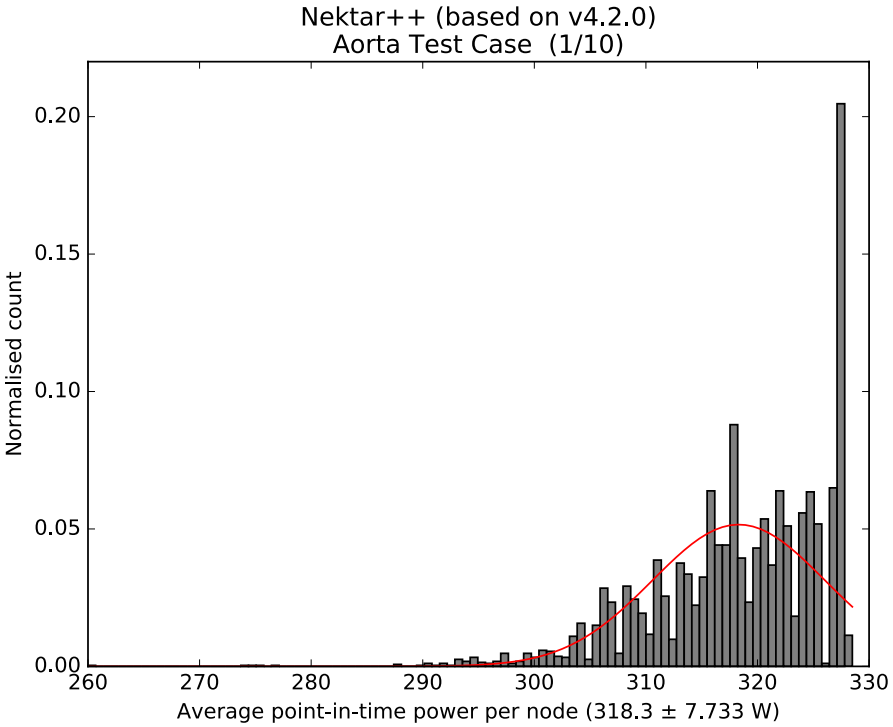


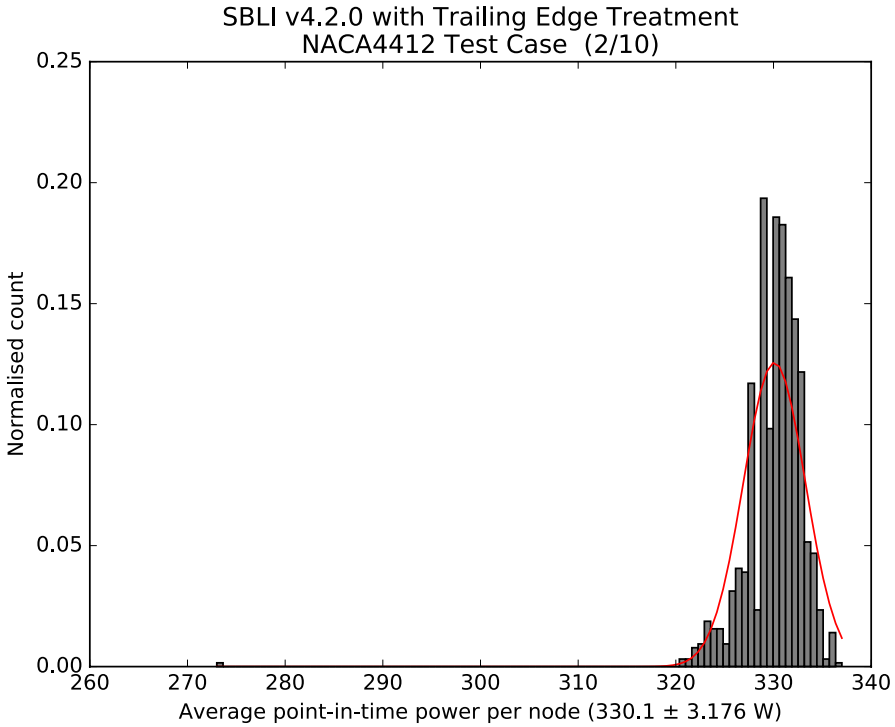
Resources

compiler: cray
nodes: 4
runs: 10

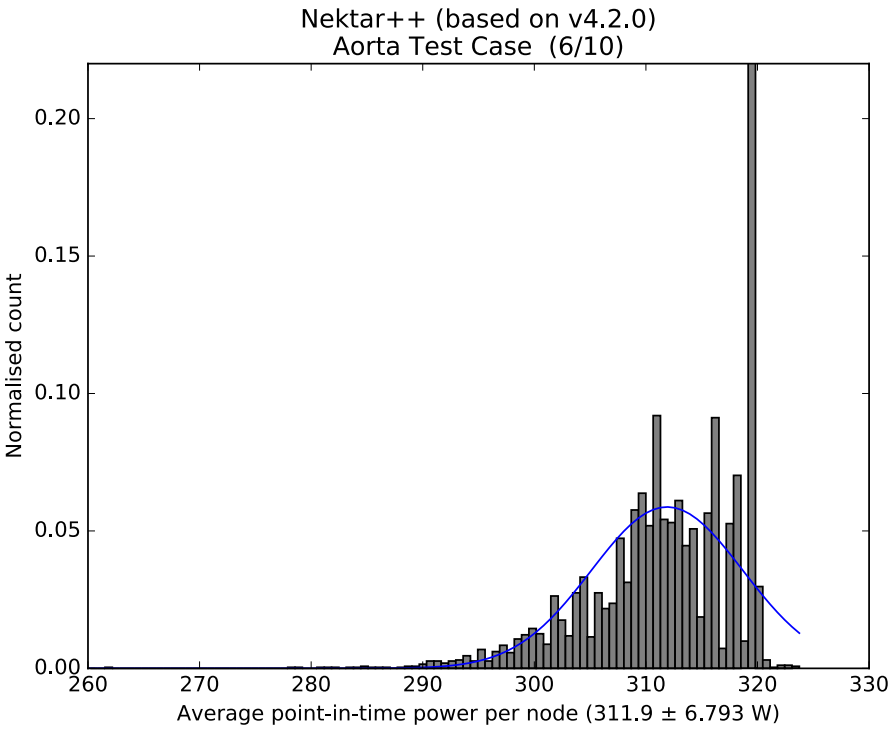
runtime: ~15 min
time steps: 1000

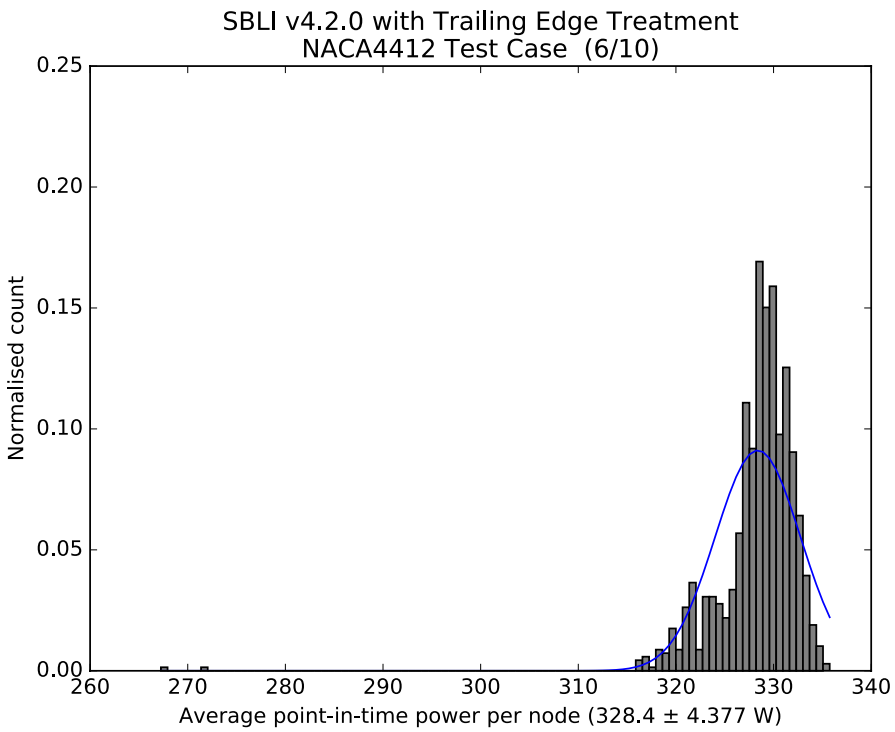
Average Power (per node)



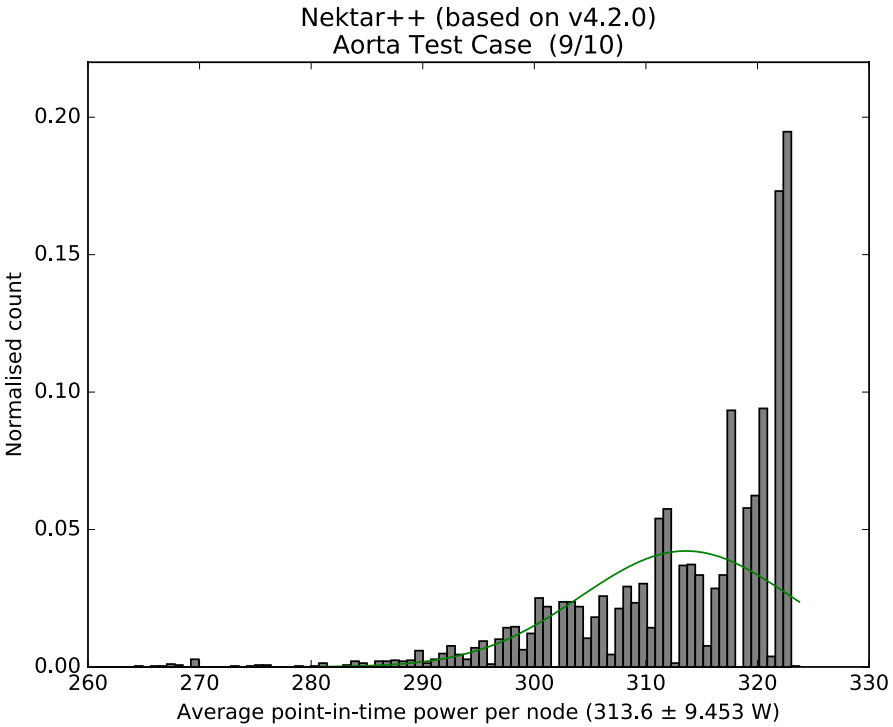


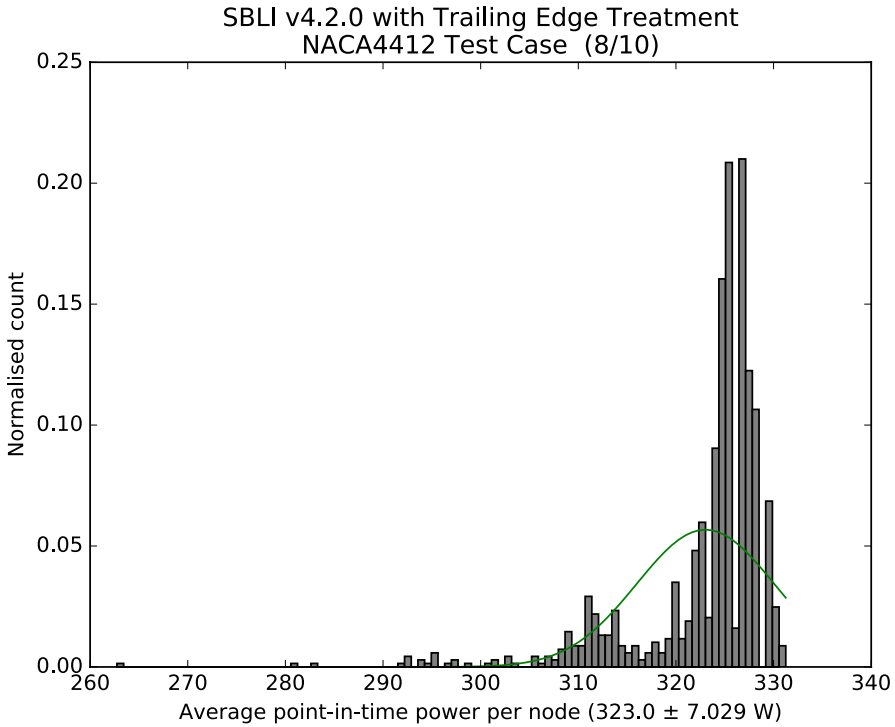
Average Power (per node)





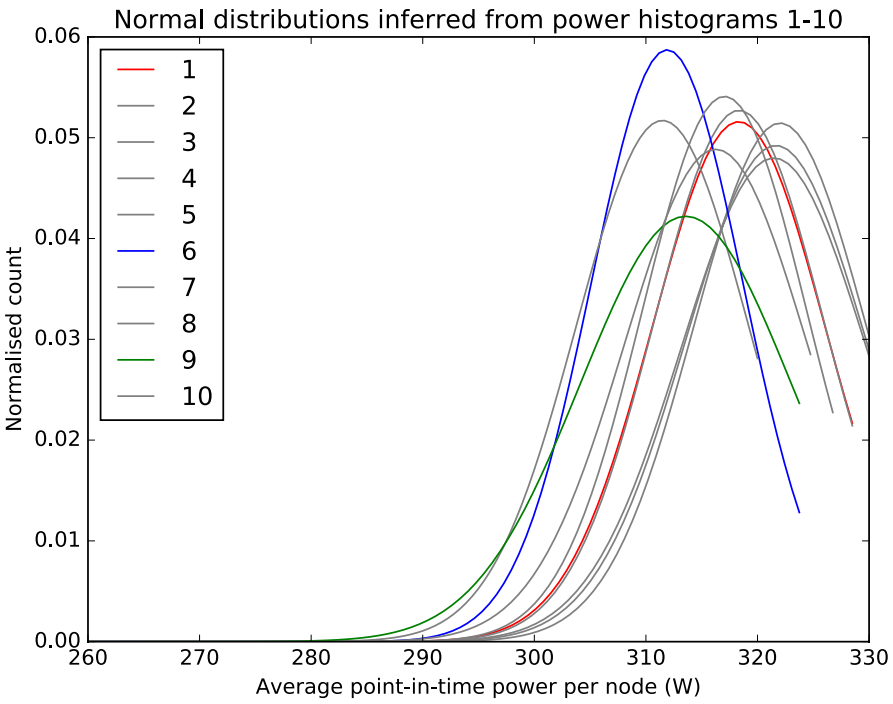
Average Power (per node)



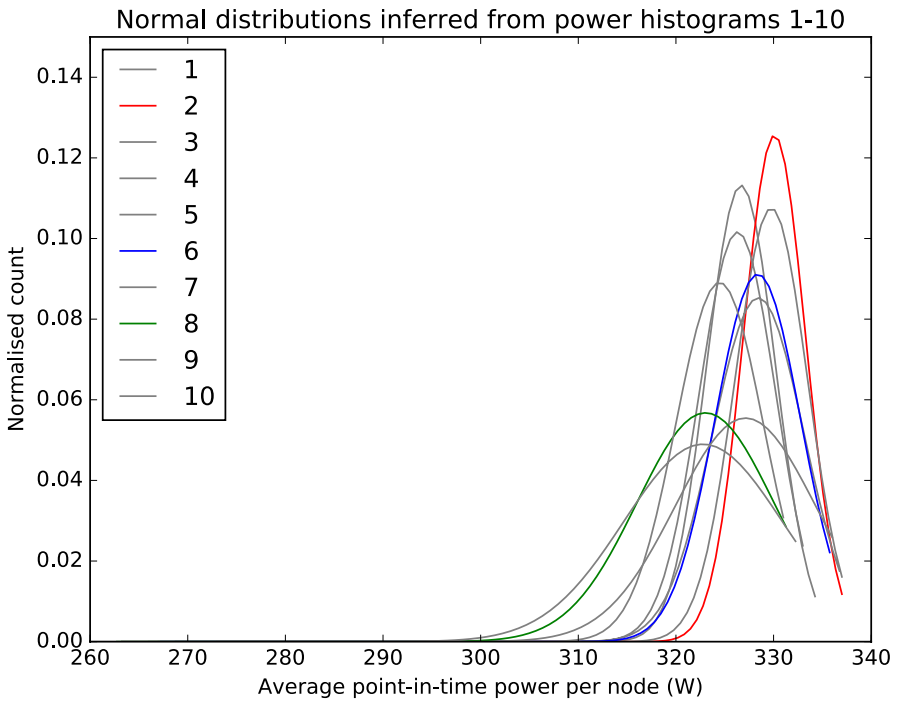


Average Power Distributions

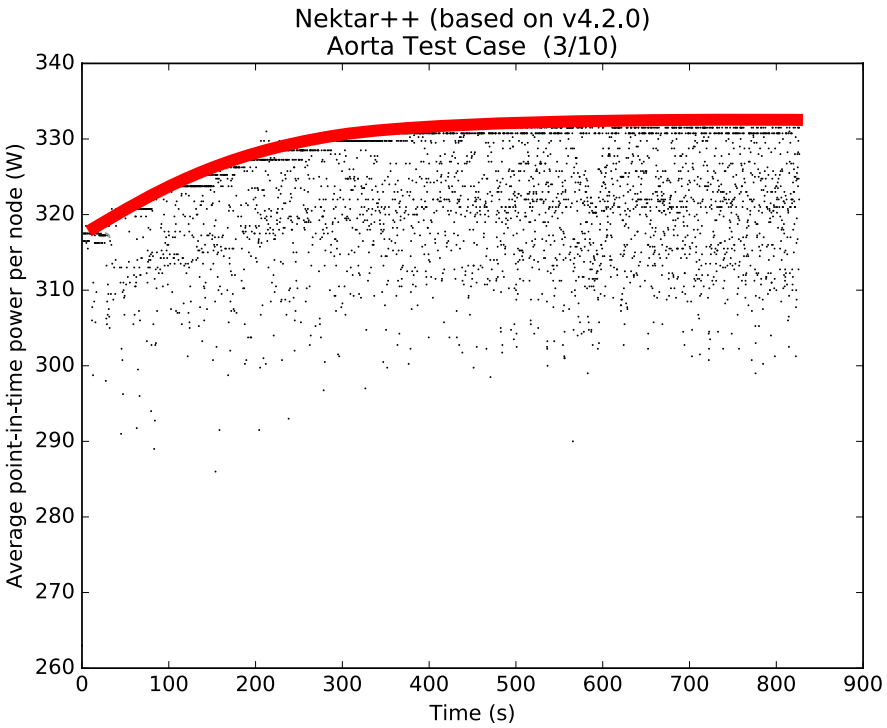
Nektar++

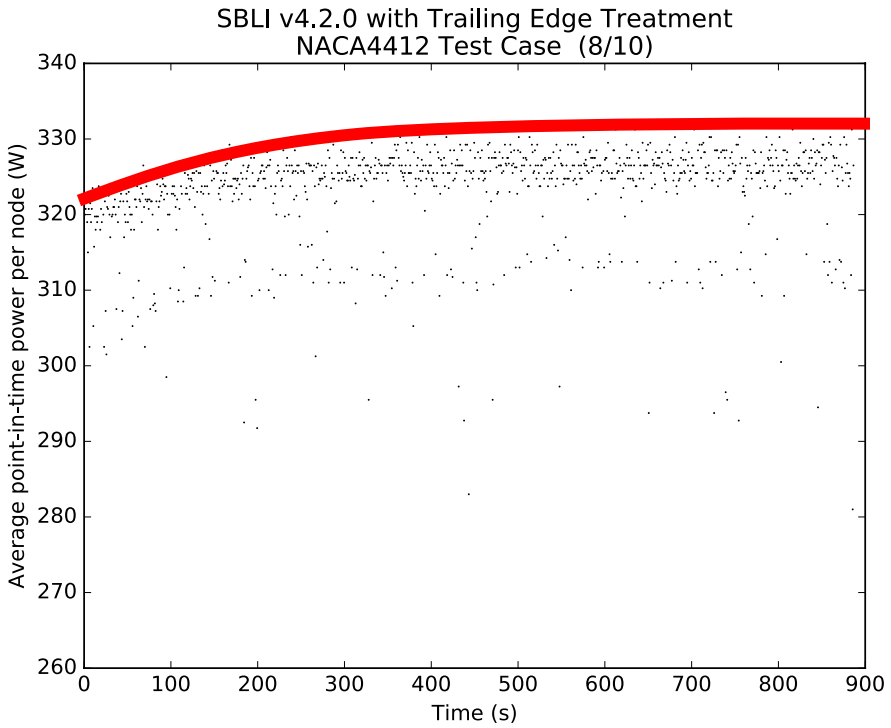


SBLI

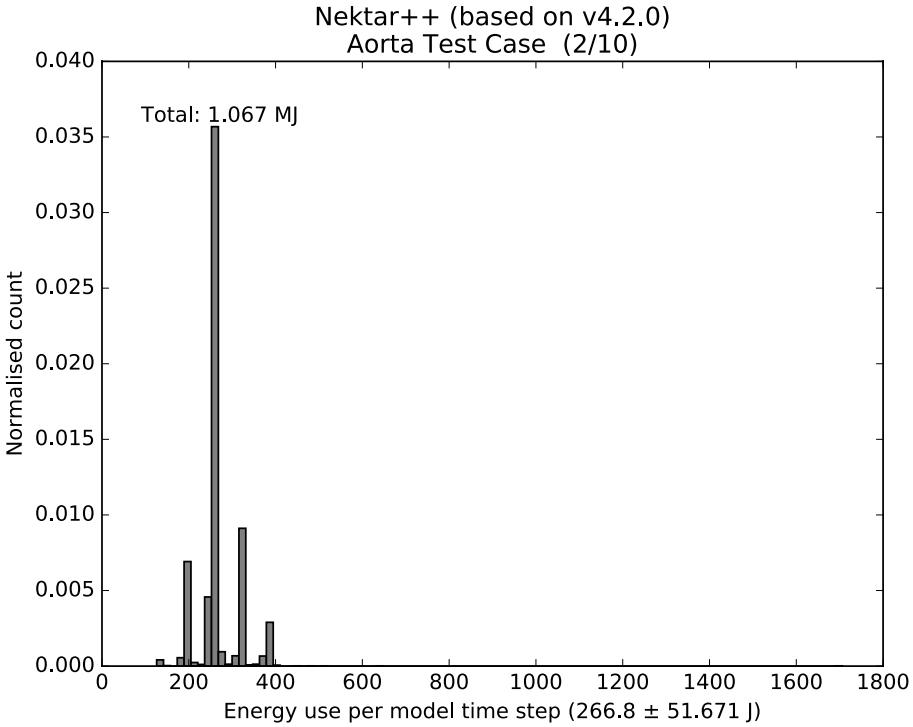


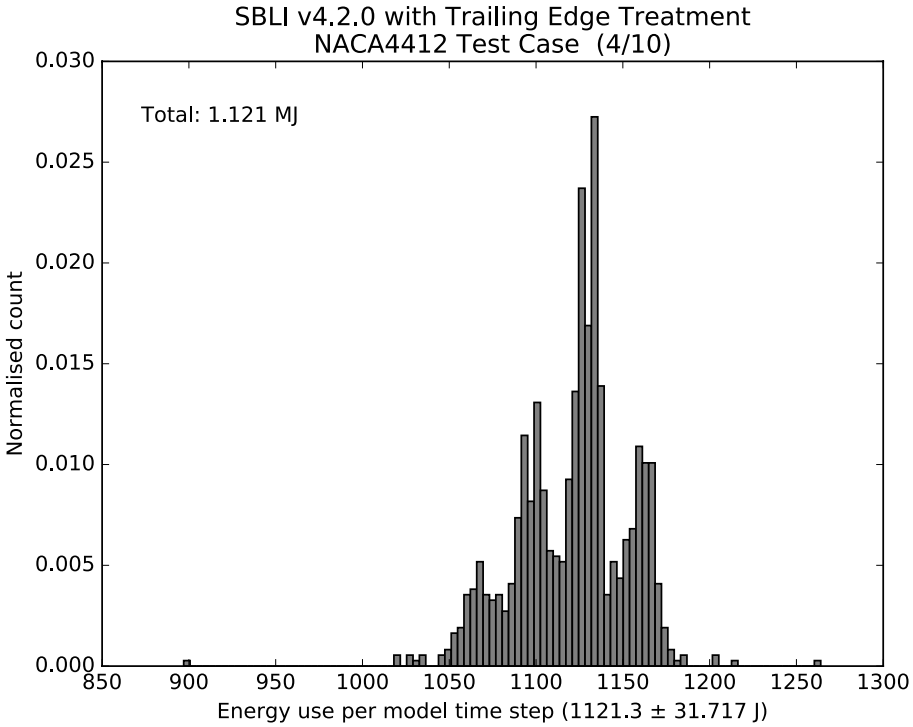
Average Power Over Time



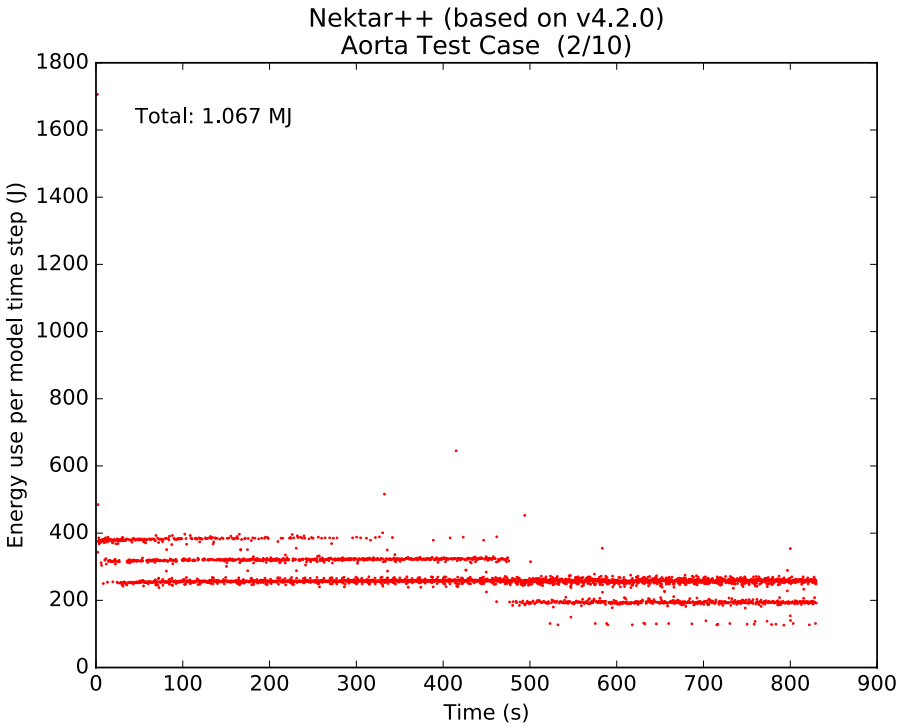


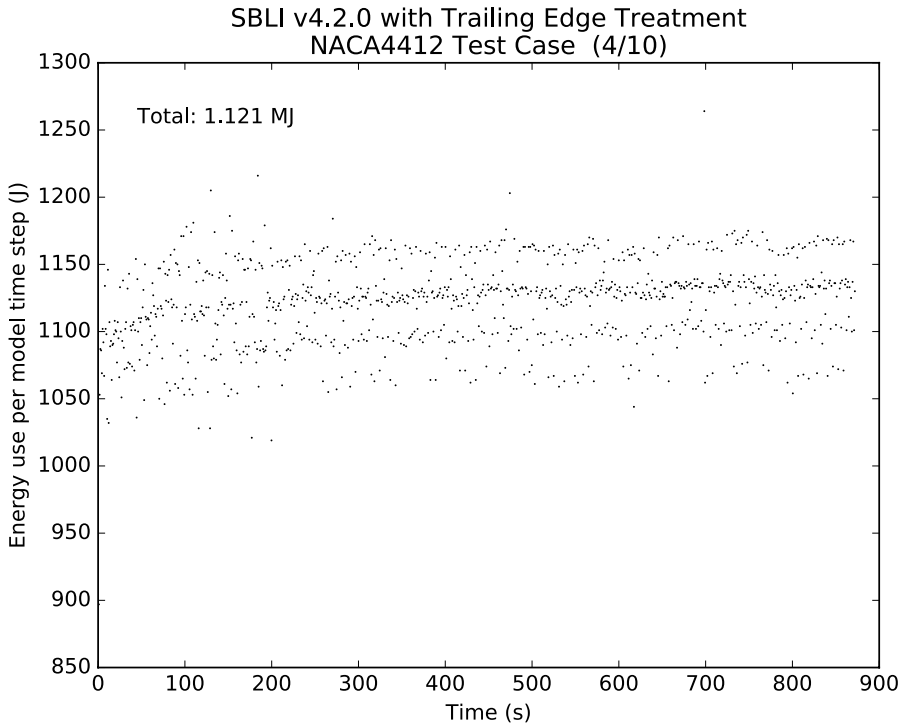
Energy Usage per time step



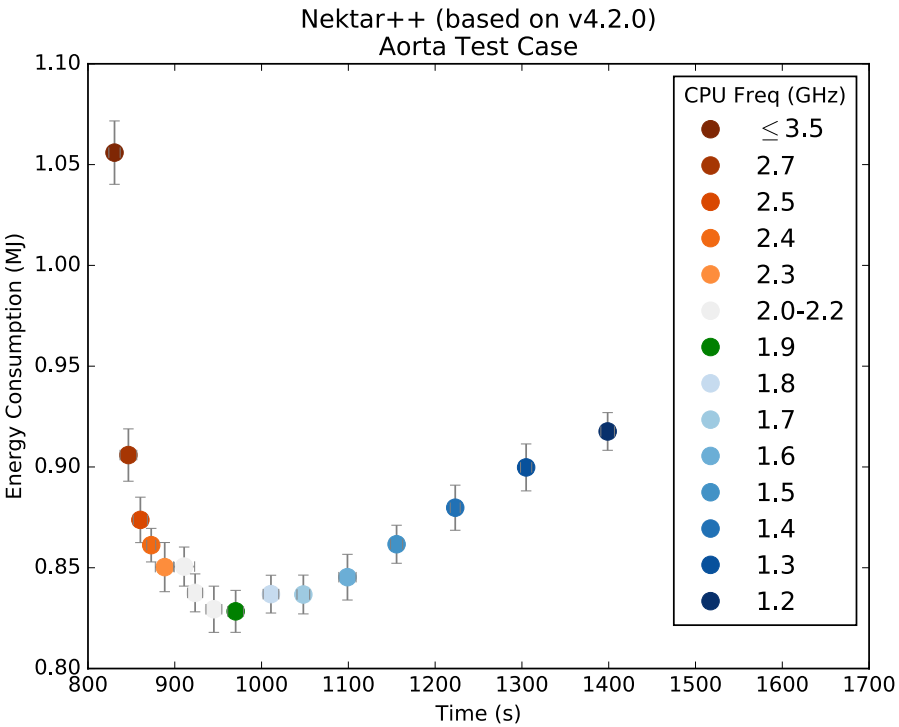


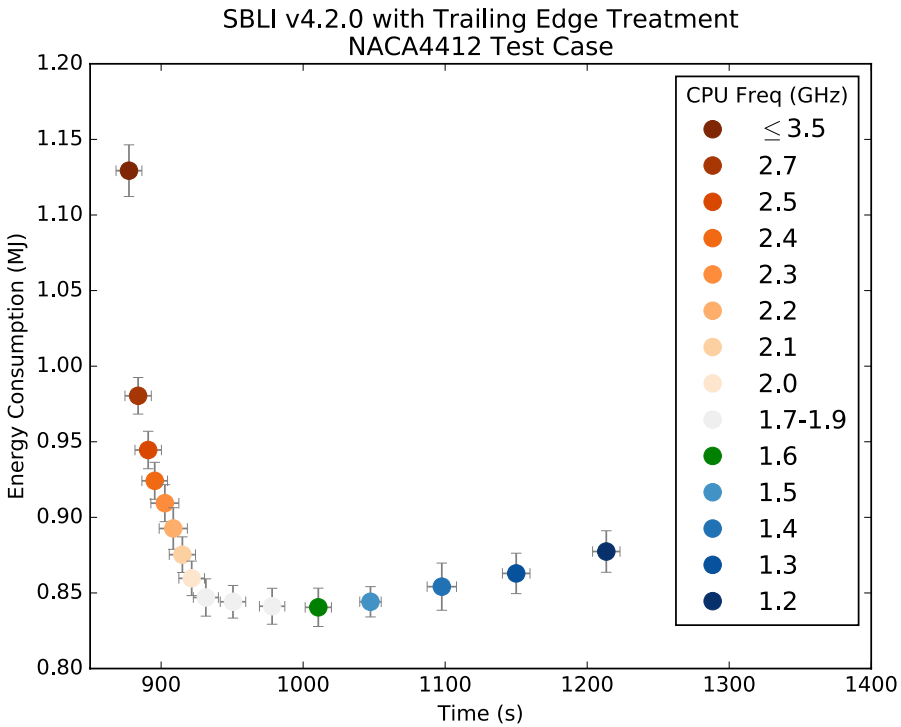
Energy Usage per time step





Energy Usage and CPU Frequencies



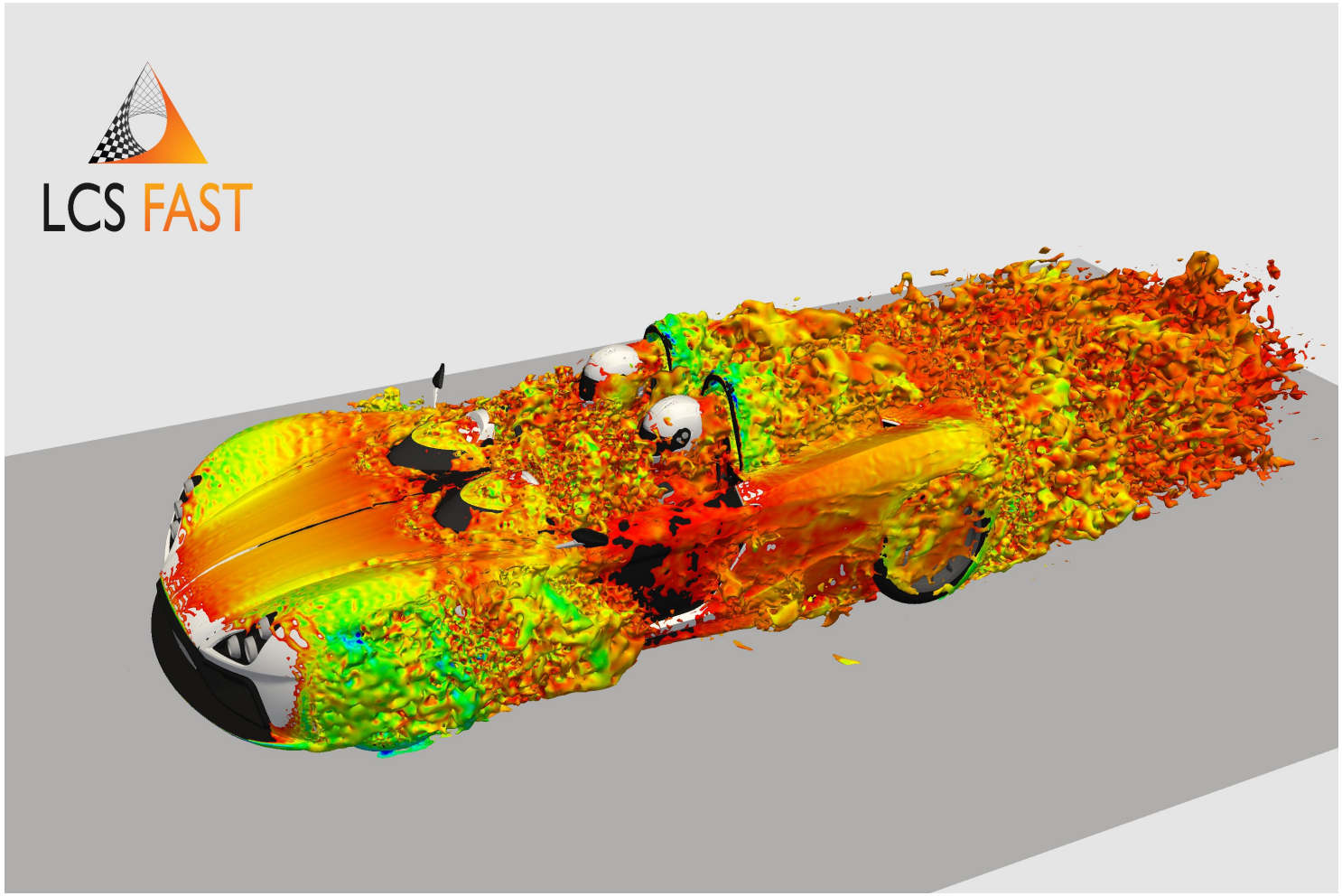


Summary Part 1

- Observed results show varying behaviour of different applications
- Performance, power, energy are all variable
  - Between runs and within a run
- If time to solution is not a concern, energy savings can be significant
- Detailed monitoring can give insight into efficiency to may inform optimisations

Moving Nektar++ towards industrial use

High-order simulation of whole car geometry



"Rp1" by Elemental (http://elementalcars.co.uk)

epcc

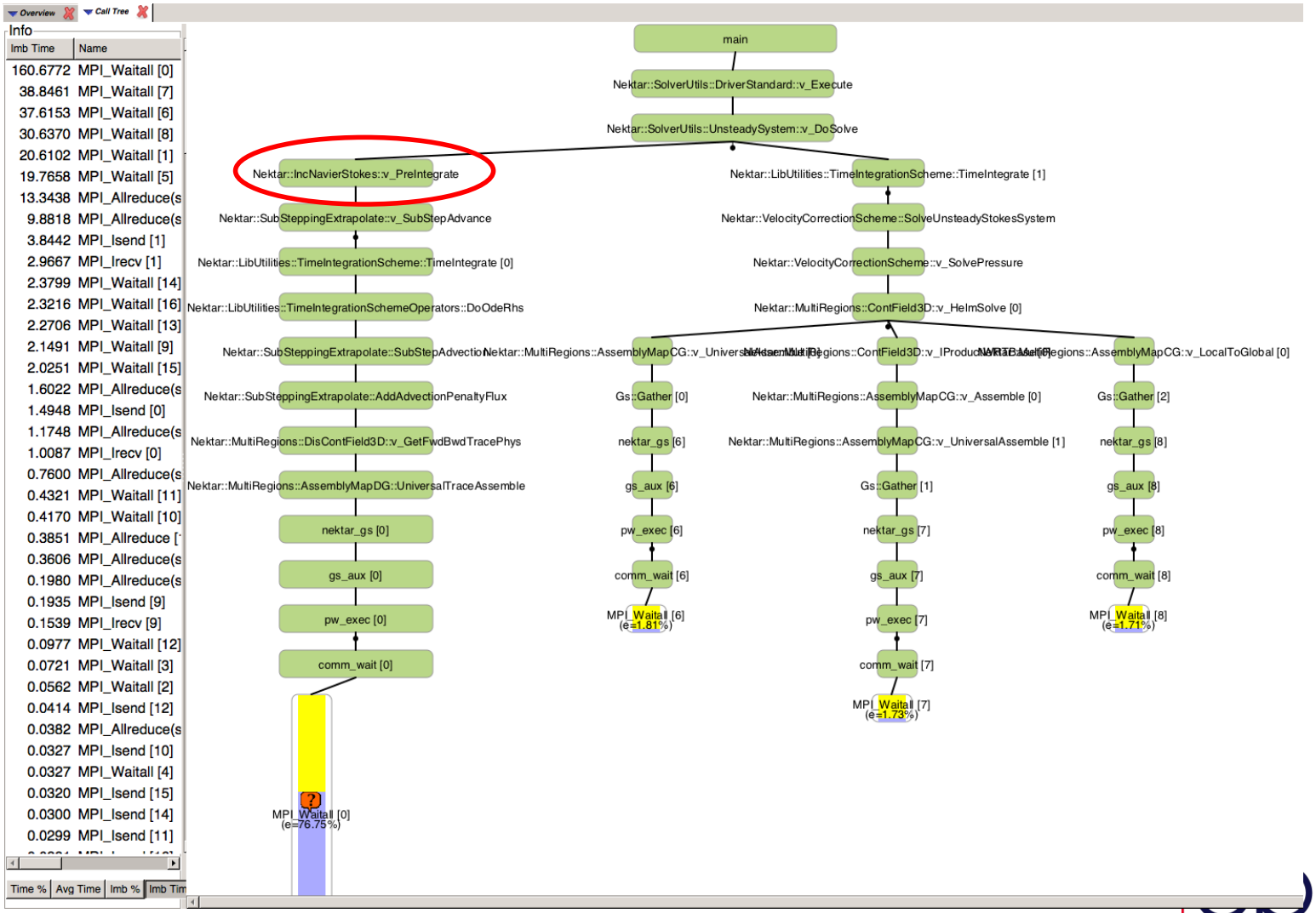
Task

- High-order simulations with Nektar++ are too slow to be usable in an industrial context
- EPCC tasked with understanding the bottlenecks and (hopefully) work out a solution
  - Focus on Nektar++'s incompressible Navier Stokes solver using the Rp1 test case
- Profiling on ARCHER Cray XC30
  - 84 compute nodes (2016 cores), total runtime ~50mins
  - Cray Performance Analysis Tools (CrayPAT)

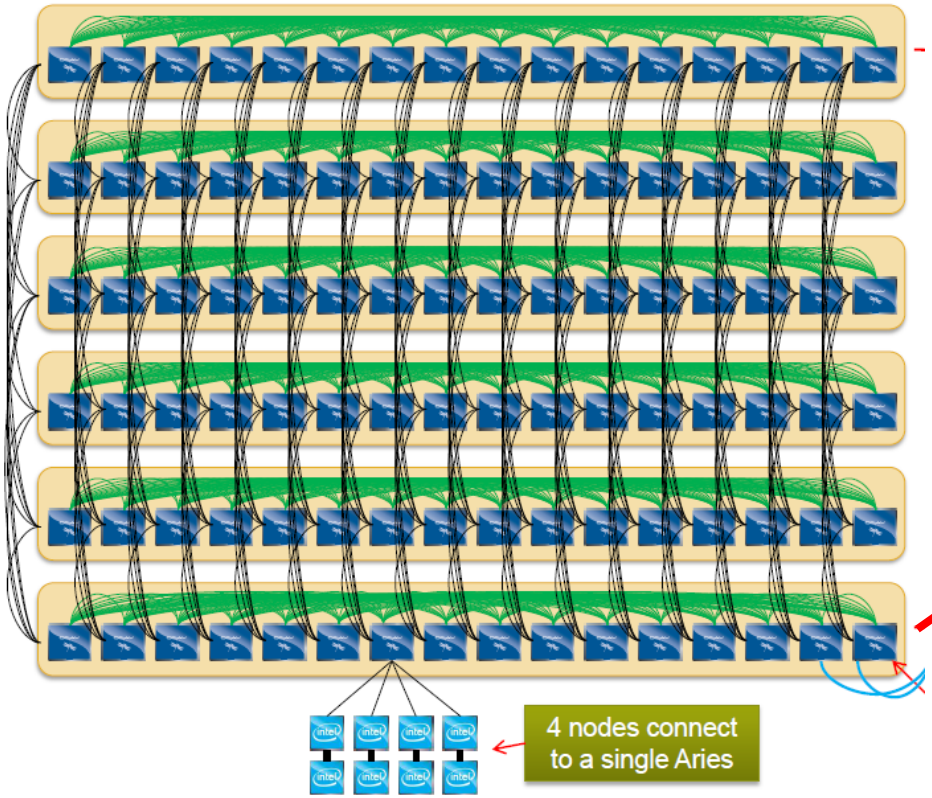
Initial profiling results

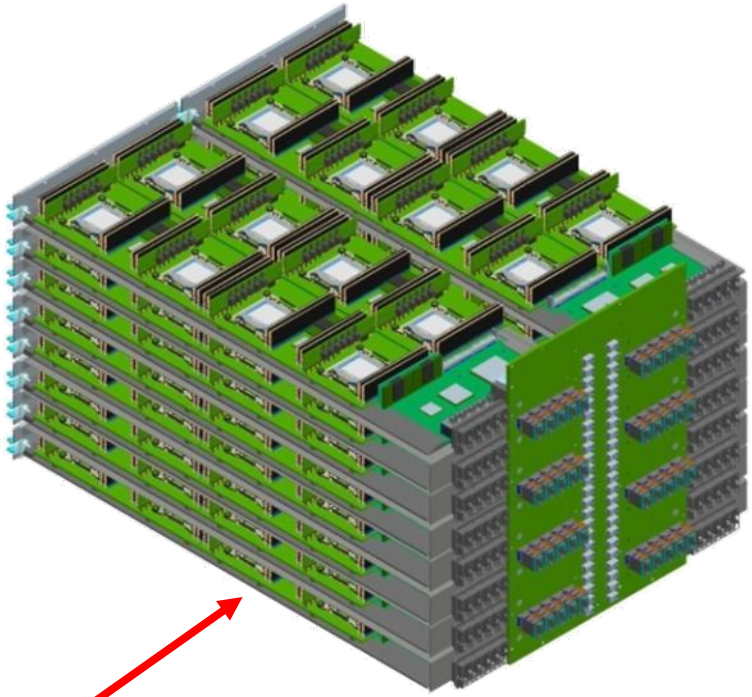
- Call Tree results showed significant `MPI_Waitall` imbalances originating from GSLib
  - GSLIB is a library for Gather/Scatter-type nearest neighbor data exchanges
- The most significant of these had an average to maximum time difference of 150 – 420 s
- ***Next step***: investigate communication patterns

Call tree visualisation of imbalance



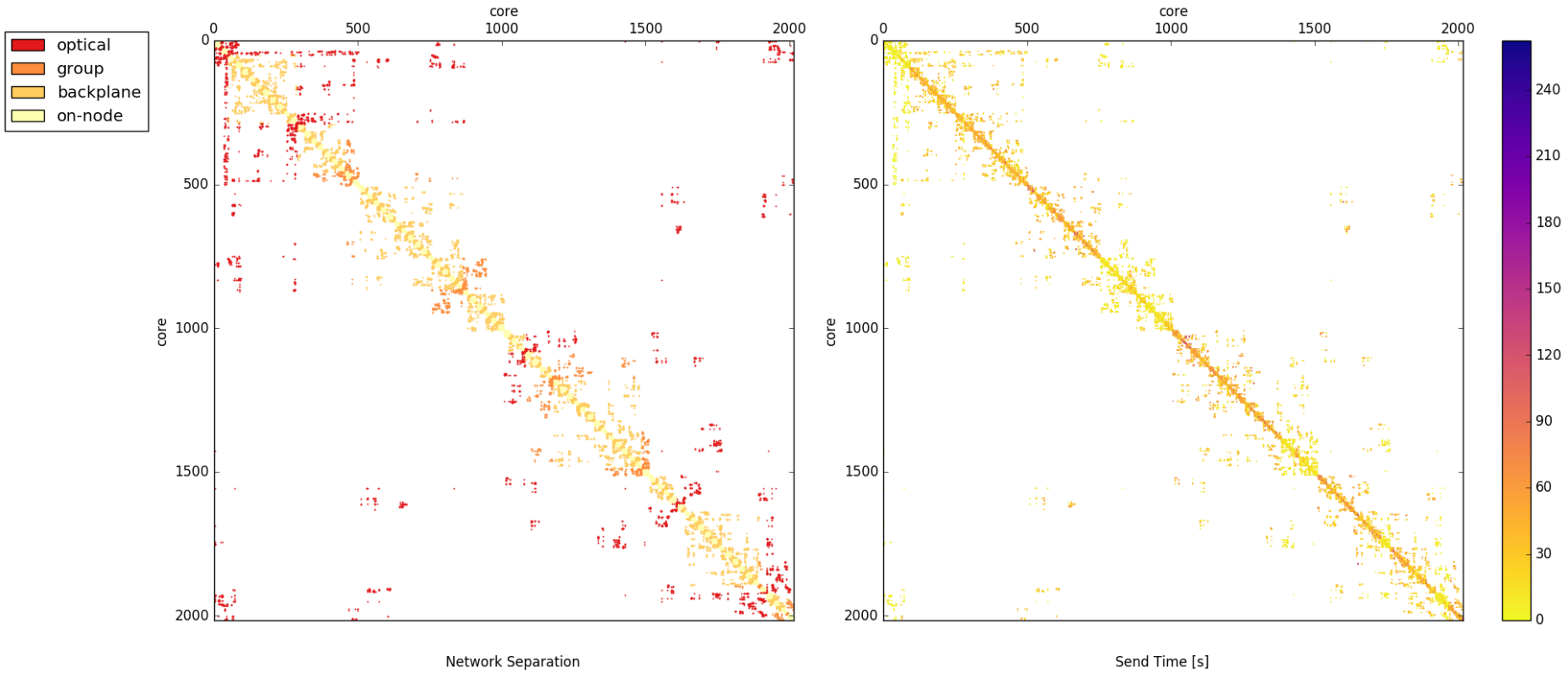
ARCHER network





- Chassis with 16 compute blades
- 128 Sockets
- Inter-Aries communication over backplane
- Per-Packet adaptive Routing

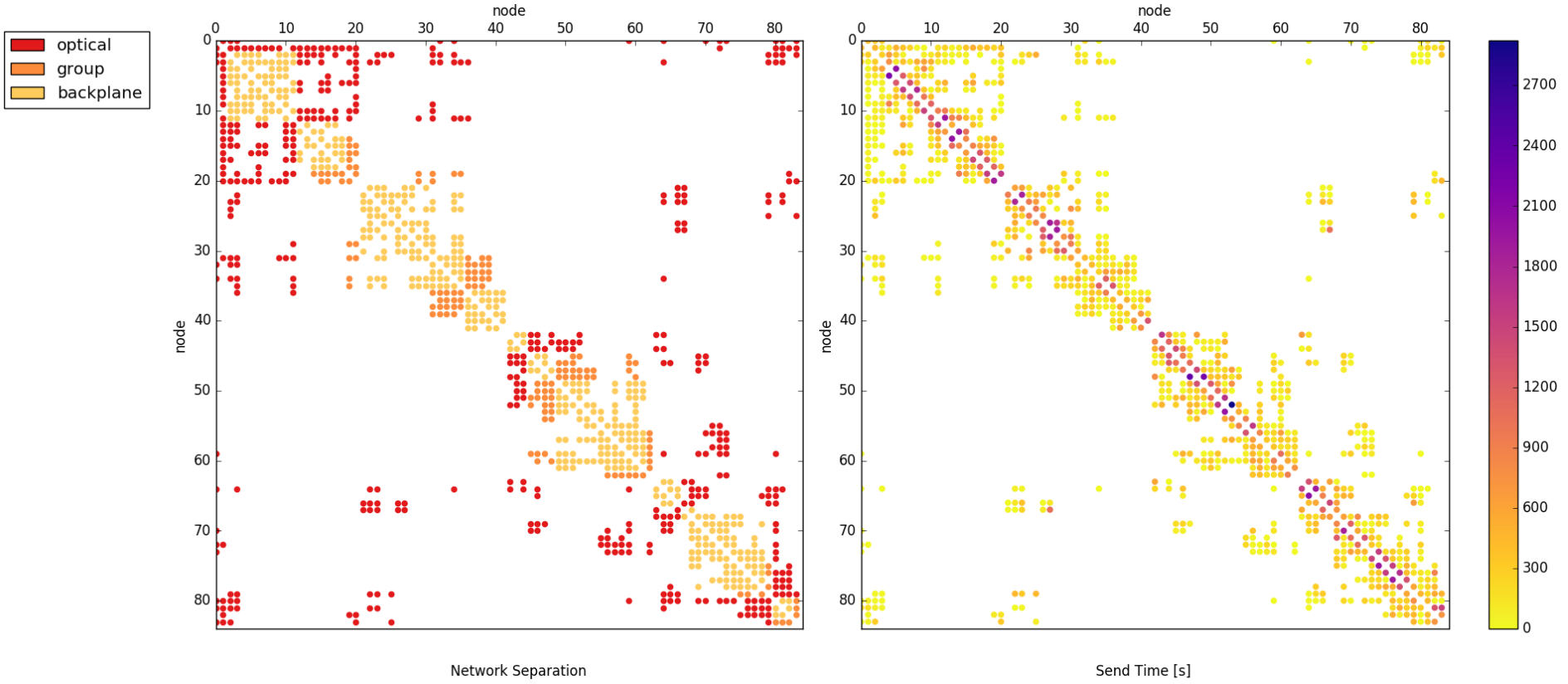
Visualisation of all communication



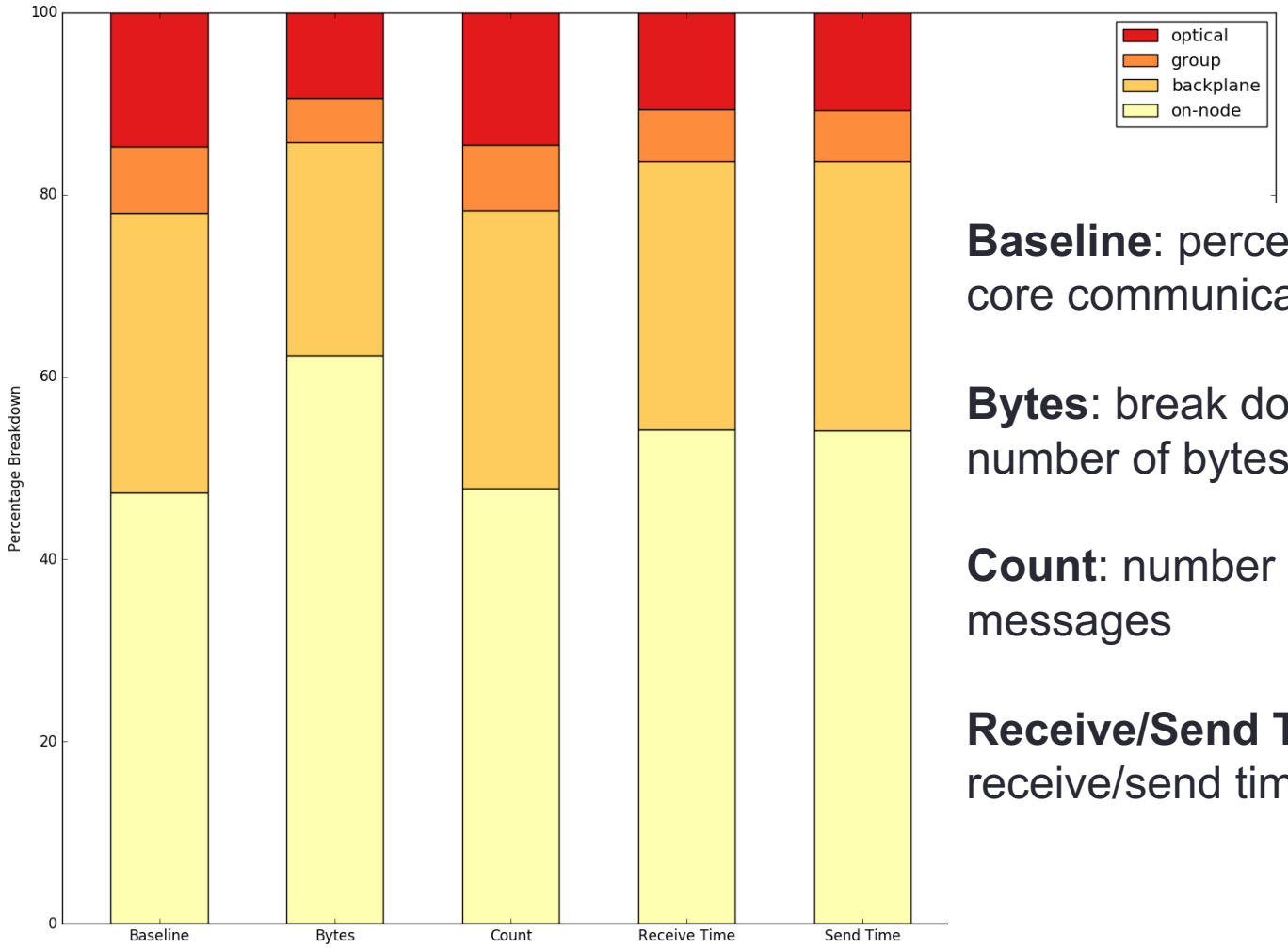
Developed python script to turn CrayPAT XML output into communication heatmap.

epcc

Visualisation of off-node communication only



Communications summary



Baseline: percentage of core communicating pairs

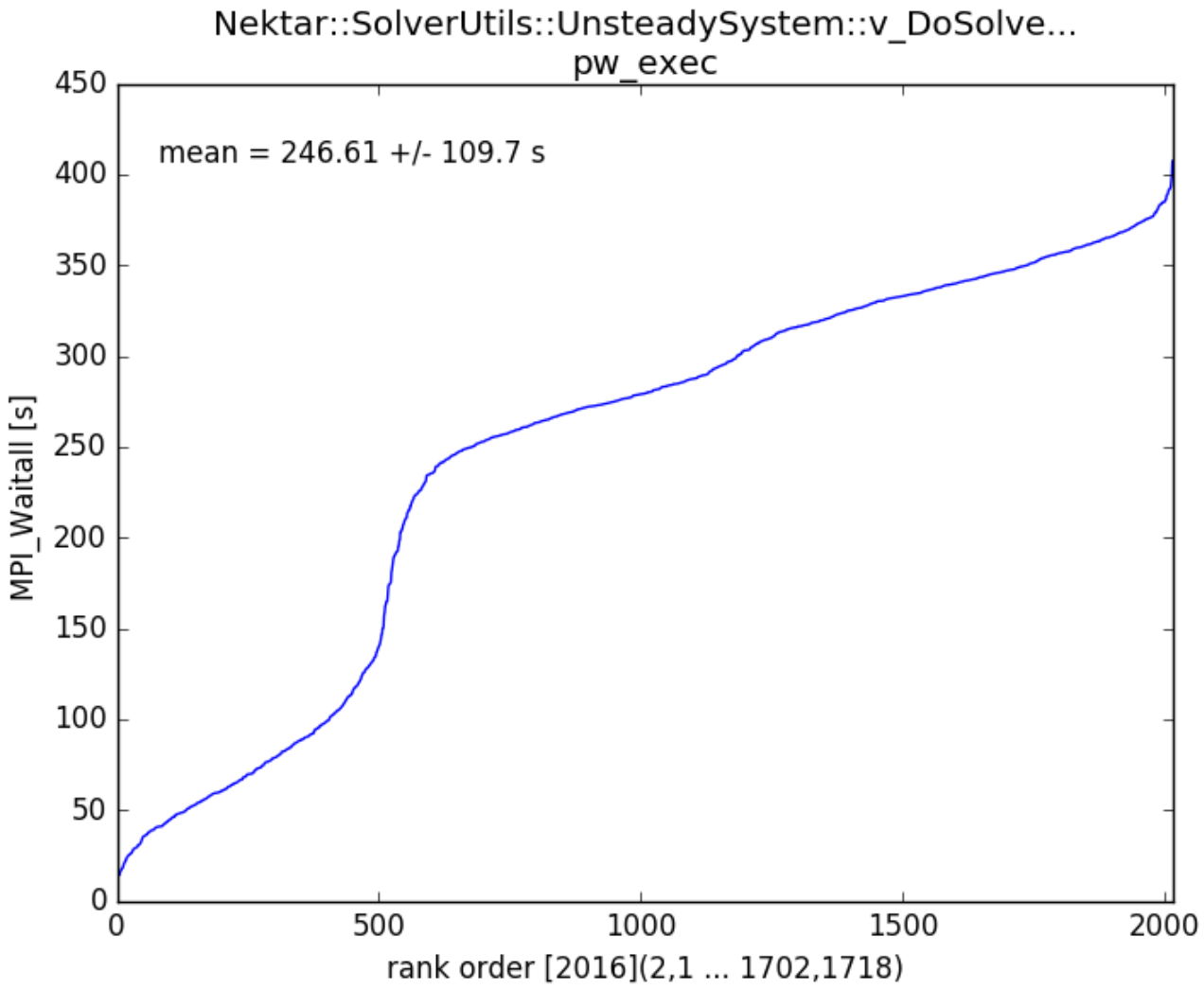
Bytes: break down by number of bytes

Count: number of messages

Receive/Send Time: total receive/send time

epcc

Time in MPI_Waitall across all MPI ranks



Ordered by waiting time per process rank, in ascending order

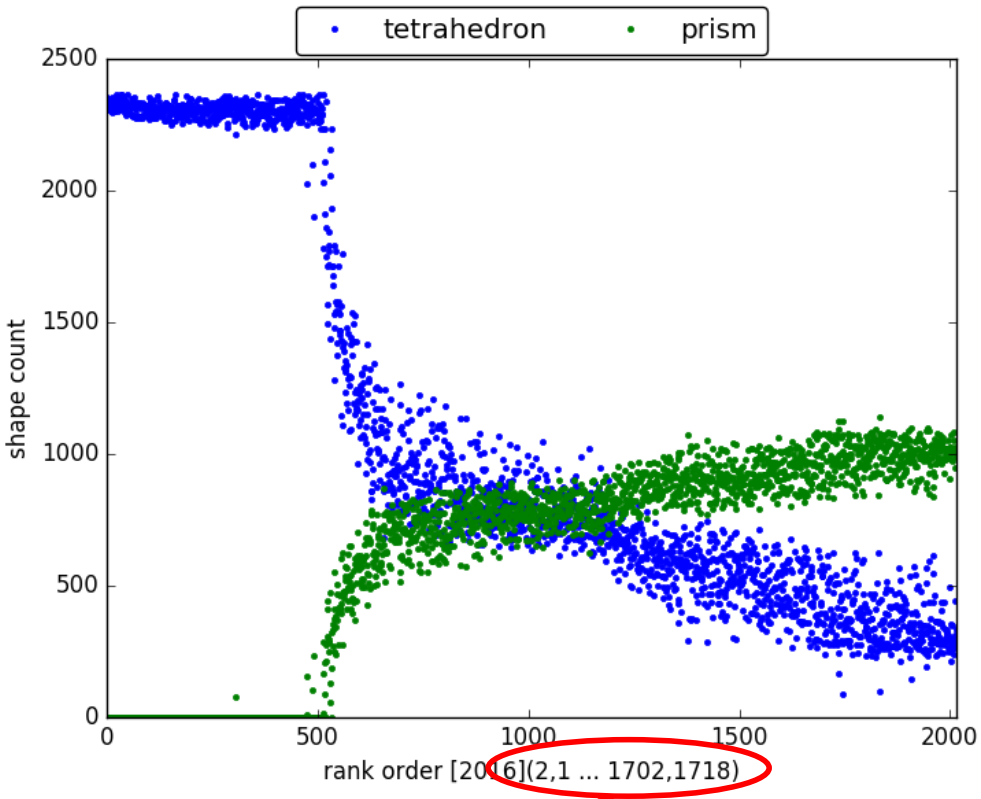
Ranks 2,1 wait the least amount of time

Rank 1702, 1718 wait the most amount of time

epcc

Impact of element shape per process on wait times

- Processes compute on both prism and tetrahedra elements
- Ranks 2,1: least time in wait ➔ lots of work
  - high number of tetrahedra
- Ranks 1702, 1718: most time in wait ➔ little work
  - few tetrahedra, more prisms



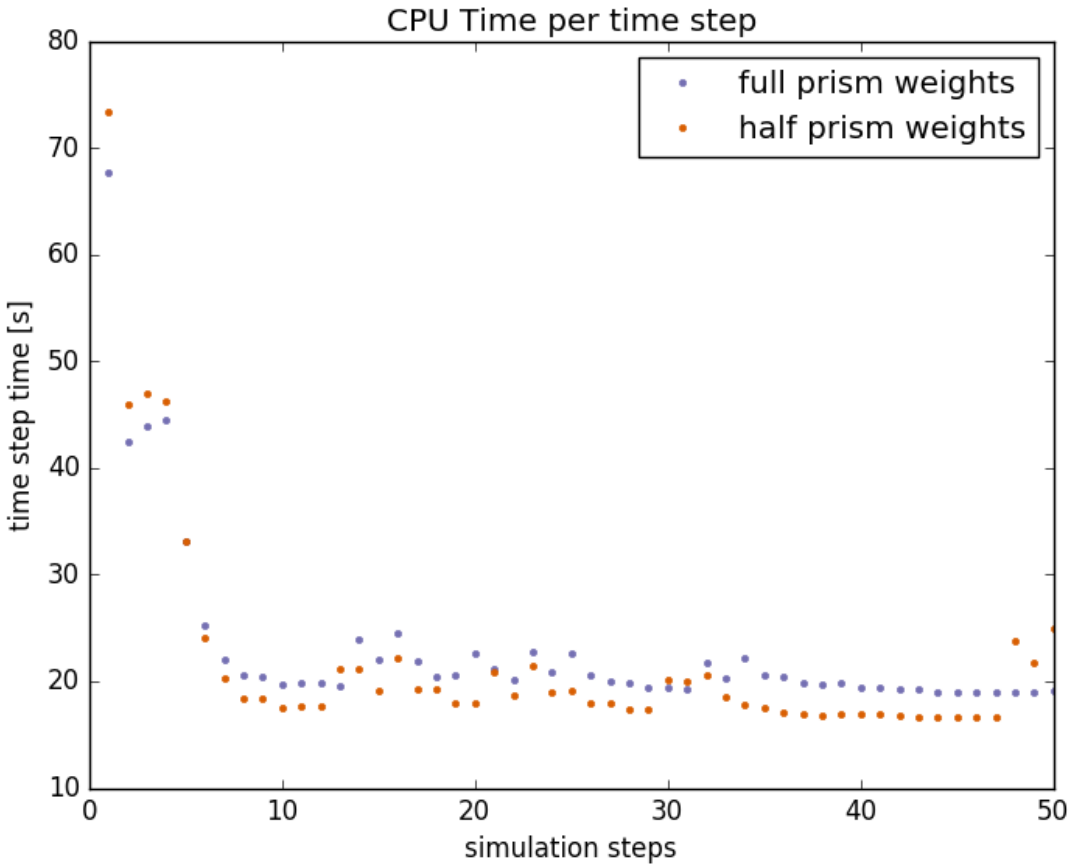
same rank order as previous slide

epcc

Change weight of prisms

- Implication from previous graph
  - Prisms take less time to compute than expected
  - Processes are given too few prisms and run out of work
- Element weights are calculated based on shape and boundary conditions
  - Graph partitioning takes into account the weights
  - Weights associated with prisms is too high, therefore partitioning leads to processes being allocated too few prisms
- What happens if prism weight is halved?

Changing the weight of prism elements



Good news: time per simulation step reduced on average, imbalance time roughly halved.

Bad news: not by enough, ~1s across all time steps.

Even worse news: new load imbalances have now appeared

➔ **Work in Progress!**

epcc

Summary Part 2

- Nektar++ has the potential to be an important application for high-order modelling
  - However in order to make code useable in industrial setting, we need to fix this performance bottleneck
  - Developed ways to visualise the profiling information to help us work out solutions
- Optimising load-balancing of complex meshes is very difficult
  - Fixing one problem can lead to new unexpected problems somewhere

Questions?