Selecting a Finite Element Analysis Backend for Exascale Fusion Reactor Simulations

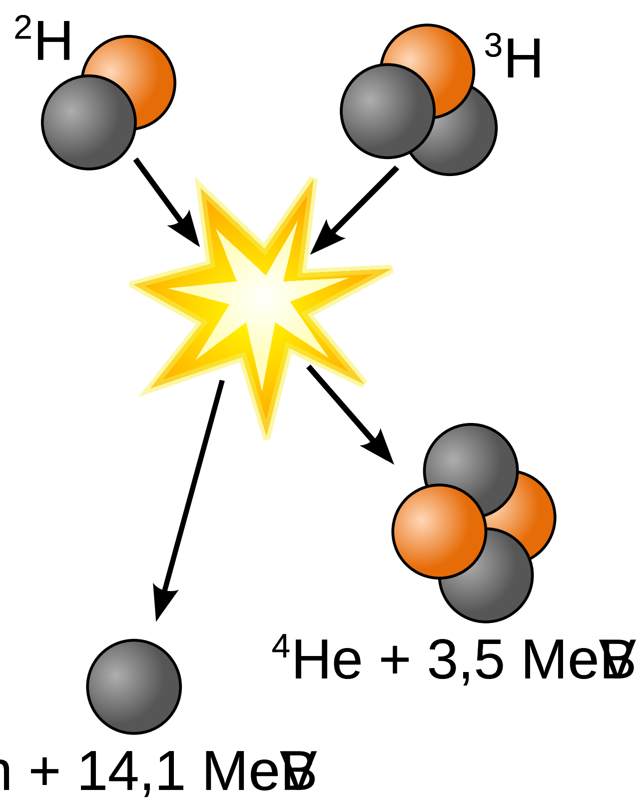
UK Atomic Energy Authority

CCFE
CULHAM CENTRE FOR FUSION ENERGY

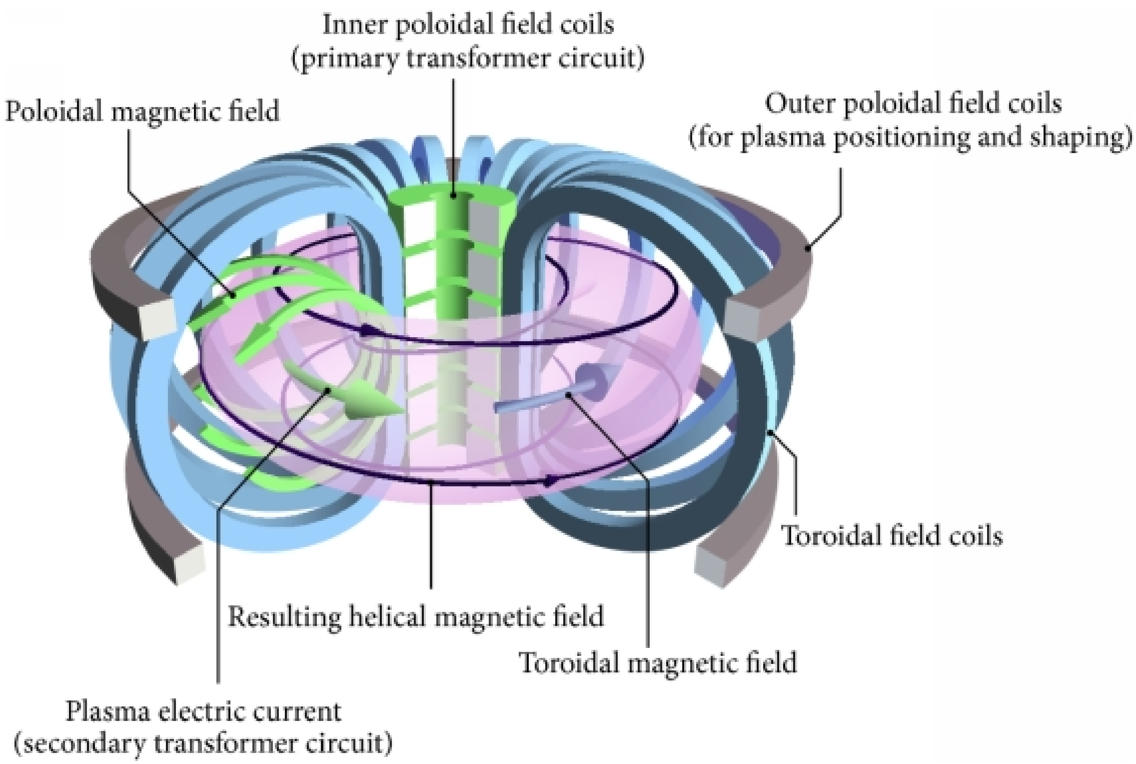
A Brief Introduction to Fusion

Producing Energy with Magnetically Confined Plasma

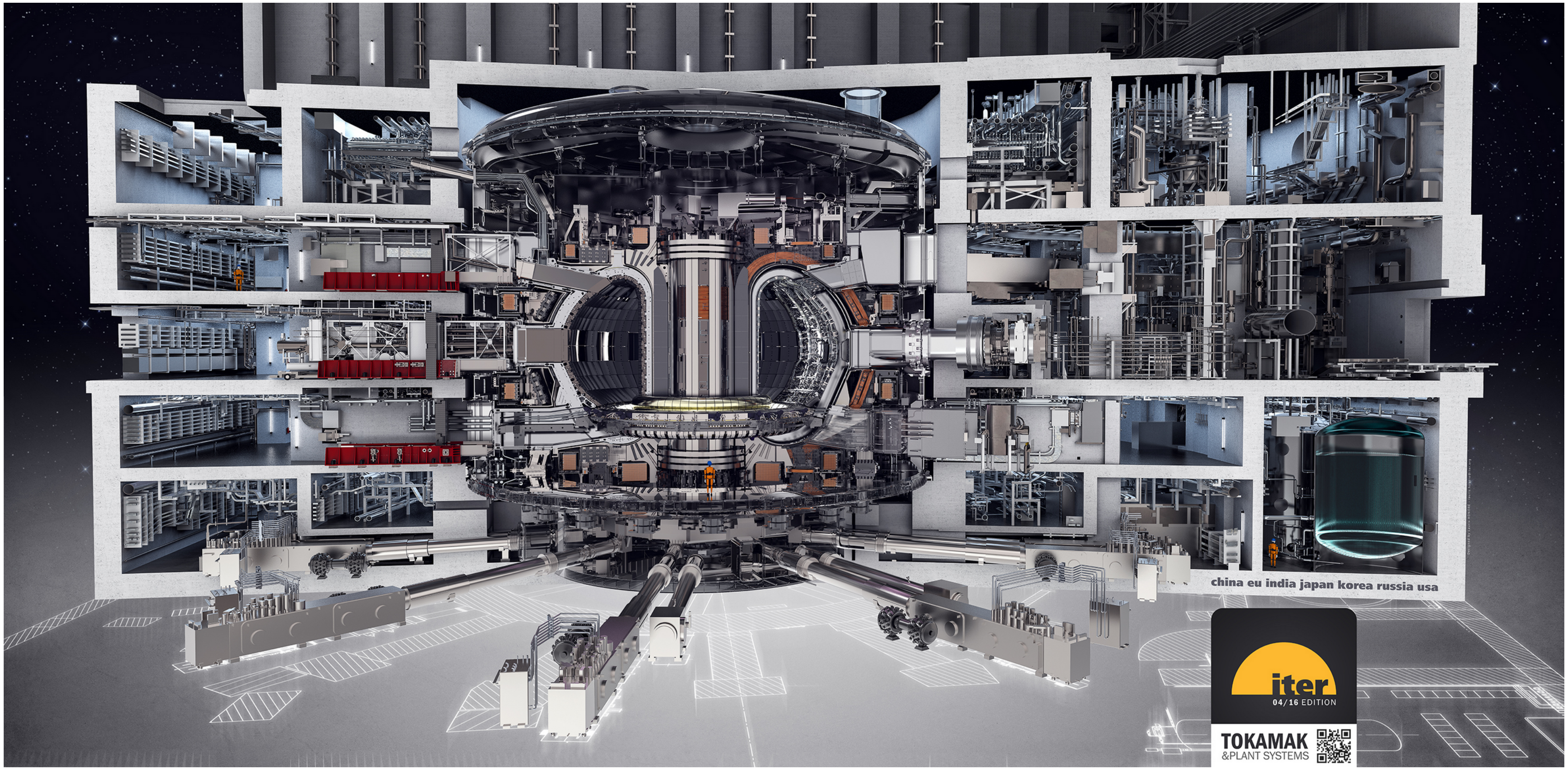
The Physics Behind Fusion



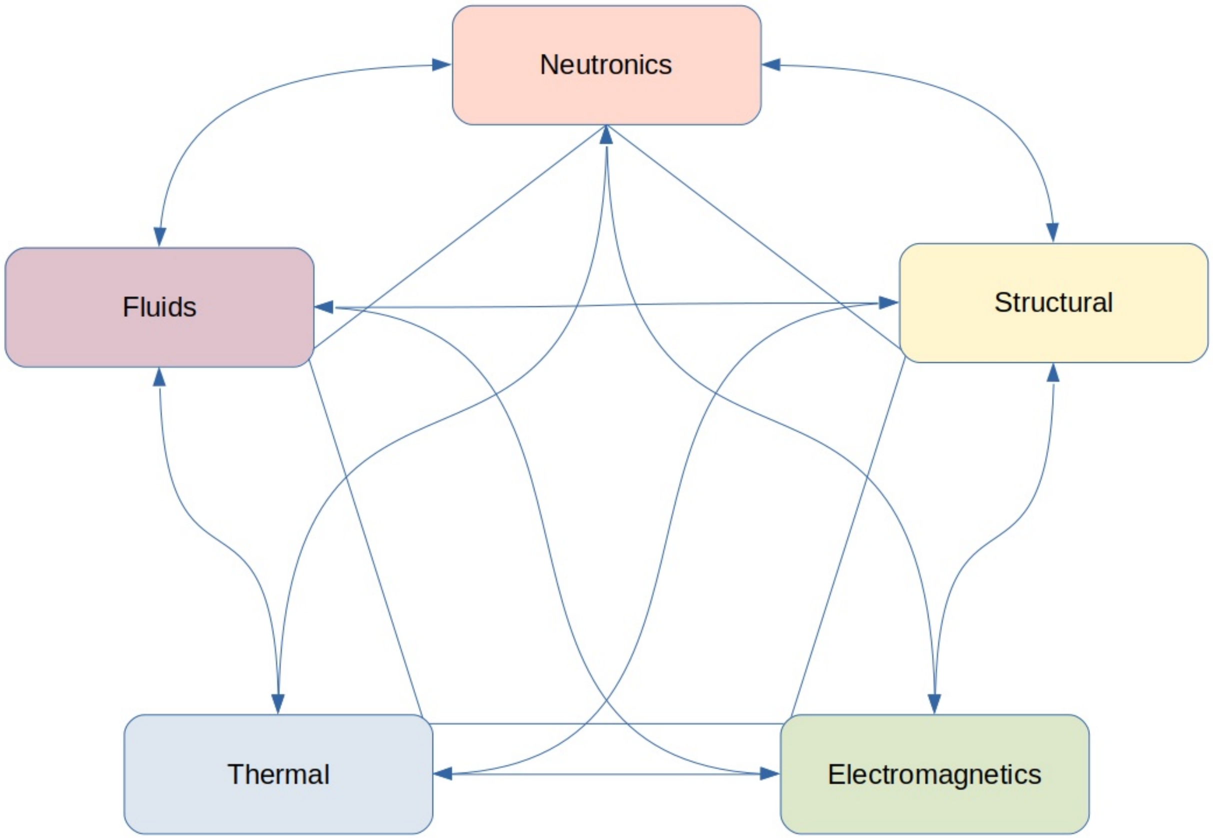
Magnetic Confinement - The Tokamak



ITER


china eu india japan korea russia usa
iter
04/16 EDITION
TOKAMAK
&PLANT SYSTEMS

Engineering Analysis Challenges



Two Approaches

Single tightly coupled simulation

- One single program
- Solve a large linear system for all the physics involved
- Ensures capture of strongly coupled physical phenomena
- Solution may be numerically 'stiff'

Many loosely coupled simulations

- Use best in class for each domain
- Couple together with a third party library and iterate
- Temporal accuracy may suffer
- Easy to decouple irrelevant physics

Selecting a Finite Element Library

Criterion One – Parallel First

Exascale simulation

Designed as a parallel code from the outset

Optimised for HPC environment

Criterion Two – Permissively Licensed

UK Atomic Energy Authority

Any location, including w/o internet

Any number of processes

Extension and modification permitted

Open Source?

CCFE CULHAM CENTRE FOR FUSION ENERGY

Criterion Three - Portable

What does the exascale look like?

Vectorised? Mixed-mode? GPU?

Criterion Four - Extensible

Open to external contribution

Good software engineering practices

Criterion Five - Supported

User community – forums, mailing lists, IRC, workshops and tutorials

Documentation – for both user and developer

Implicit Criterion – Compiled Language

Interpreted languages incur an overhead

Example: FEniCS versus DOLFIN

When scaled up, every little helps

Implicit Criteria – Stable API, Actively Developed

A reliable library must have a stable API, thus not in ‘alpha’ or ‘beta’ development

To be actively supported, it must actively developed

Initial Survey and Elimination

Initial survey found 35 potential candidates

Eliminated those that were:
Not parallel-first / HPC
In early development
Poorly supported
Inextensible
Abandoned

Shortlist

• deal.ii	www.dealii.org
• DUNE	www.dune-project.org
• DOLFIN	fenicsproject.org
• libMesh	libmesh.github.io
• MFEM	mfem.org
• MOOSE	mooseframework.org
• Nektar++	www.nektar.info

Performance Measurement – Problem Definition

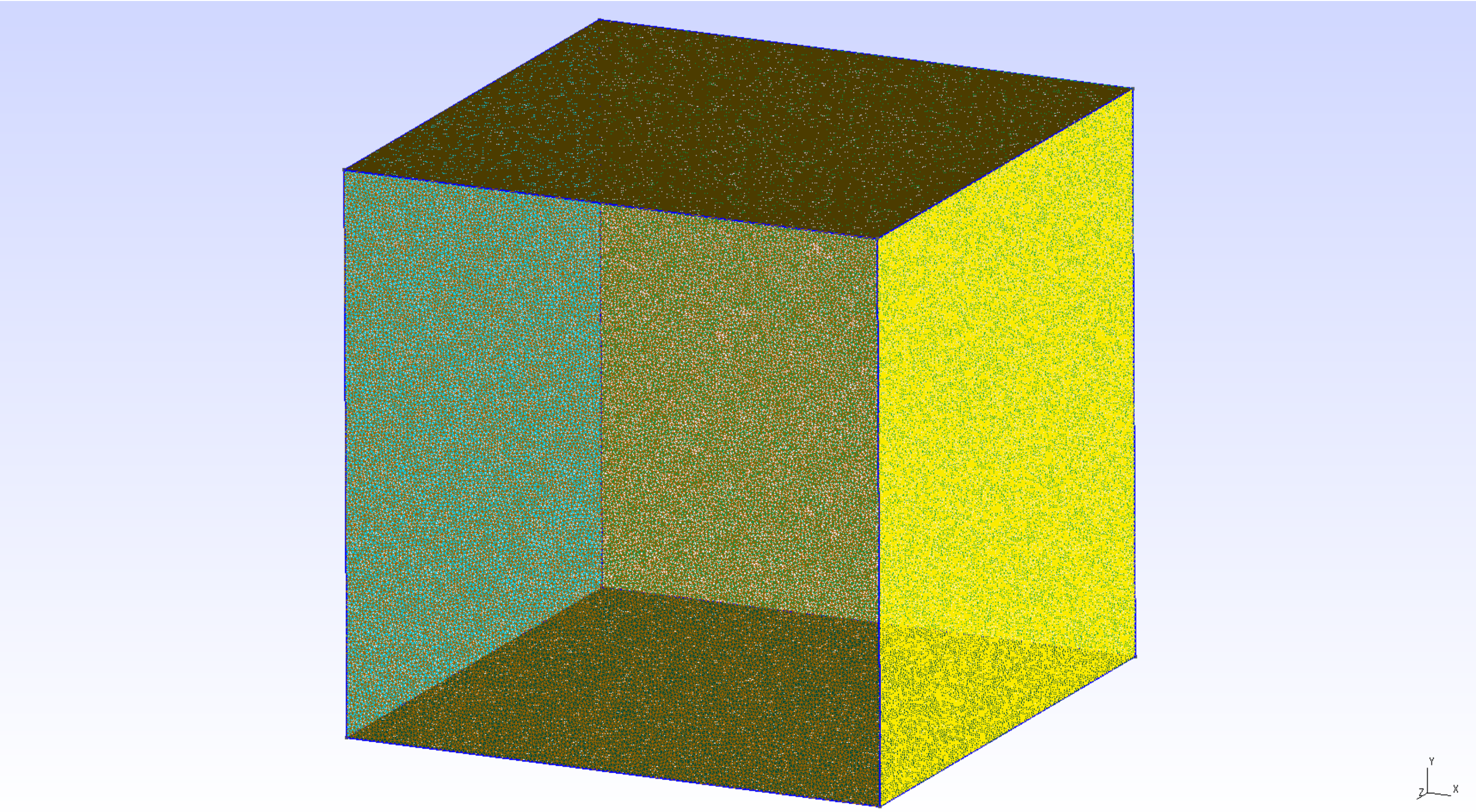
- Steady State: Poisson Equation

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

- Time Dependent: Heat Equation

$$\frac{\partial u}{\partial t} - \nabla^2 u = f$$

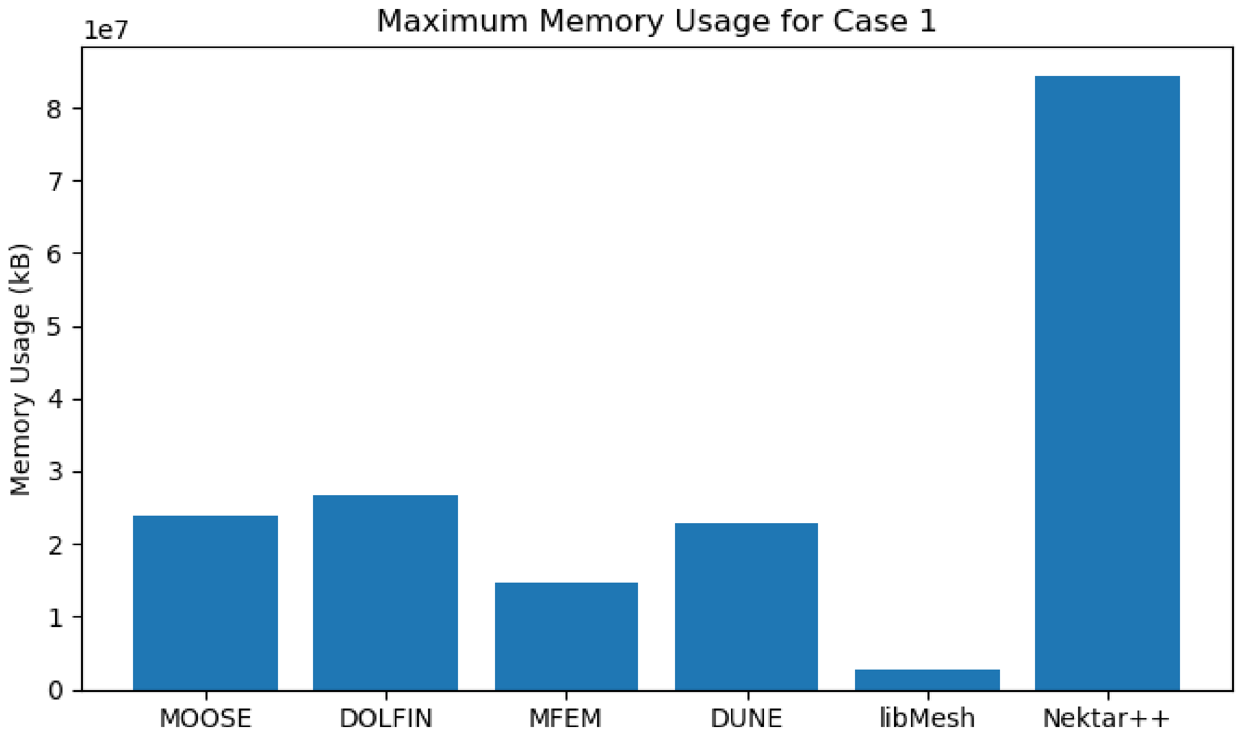
Performance Measurement – Mesh Definition



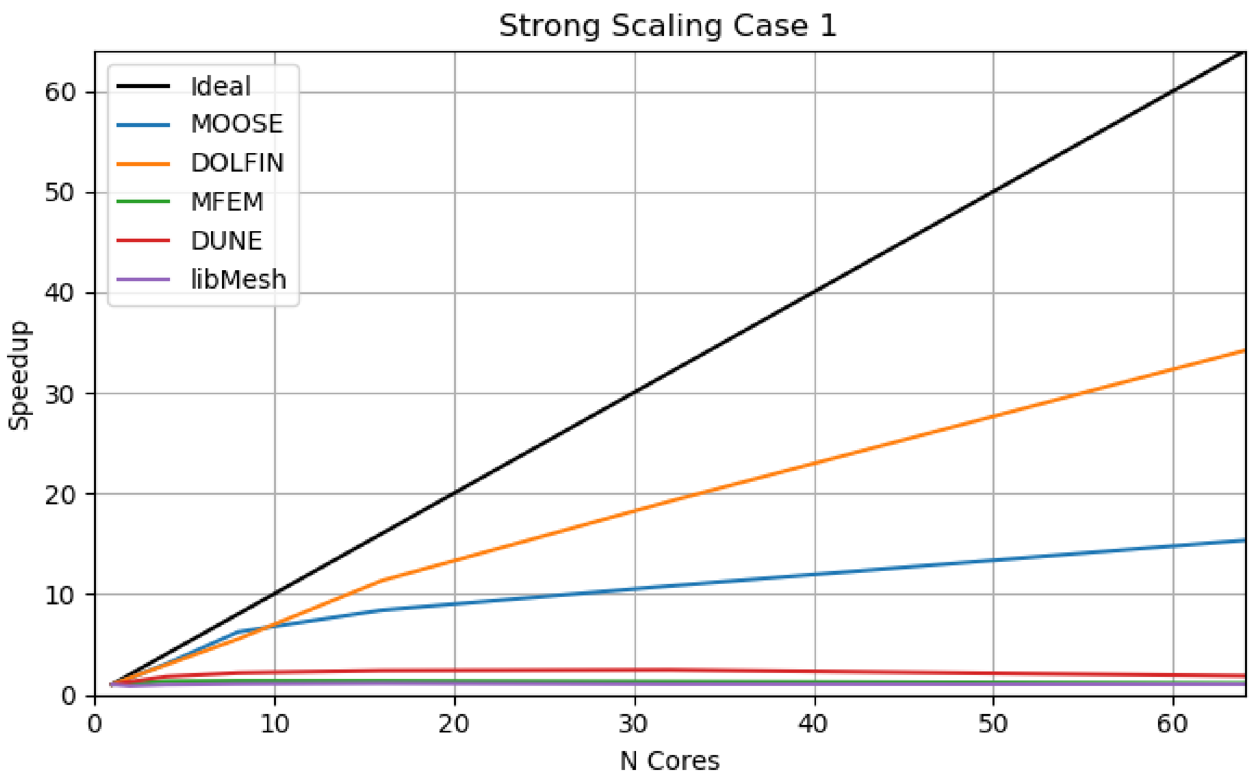
http://gmsh.info

Dealbreaker

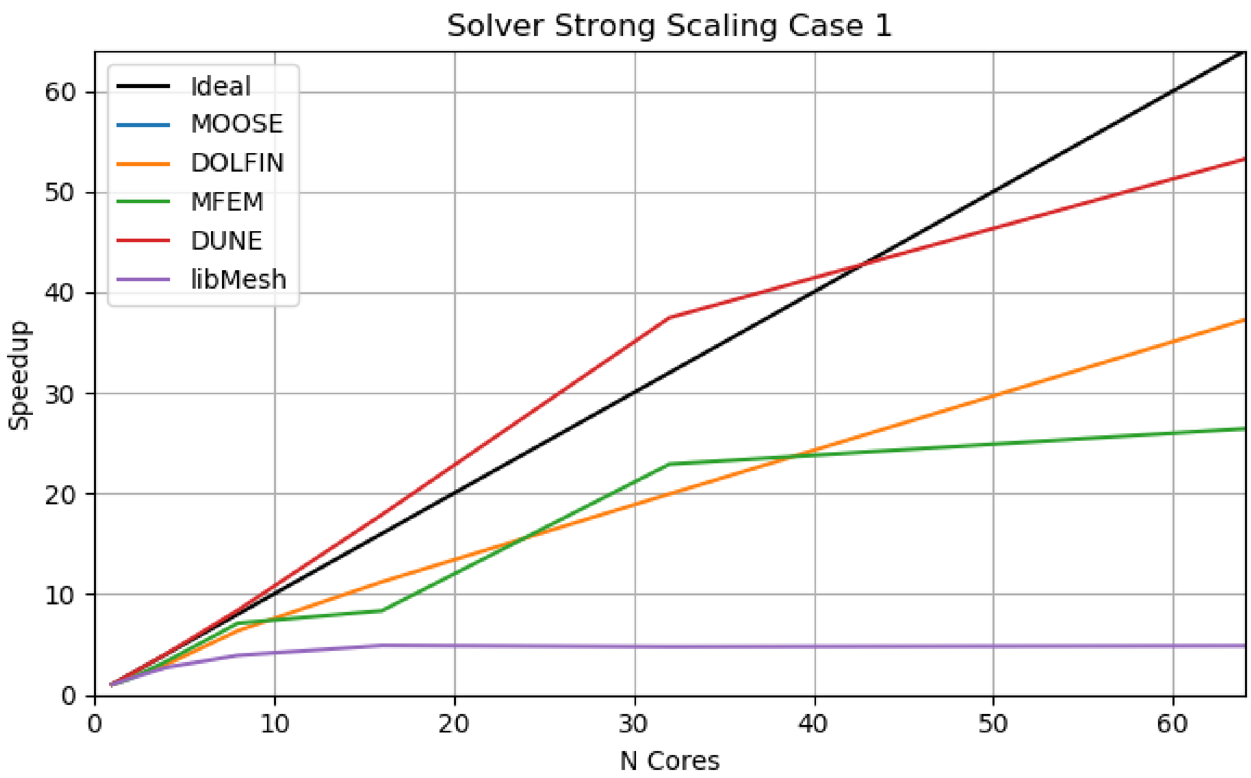
Results – Memory Usage



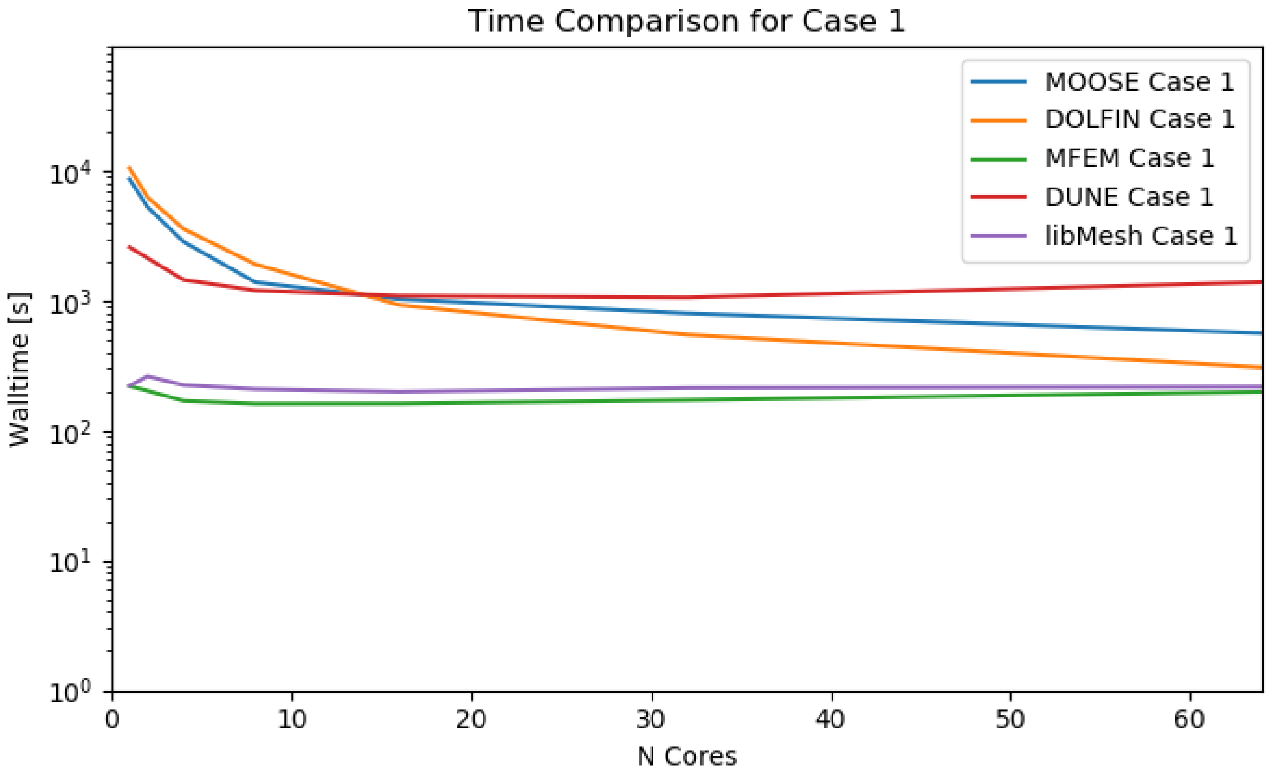
Results – Scaling (Total Time)



Results – Scaling (Solver Time)



Results – Wall Time



Results – Honourable Mentions

MFEM – Highly portable, few dependencies, clear and simple build process

MOOSE – Multiphysics coupling design

Conclusions

All things considered, there is no clear winner

Picking A Winner

$$s_{\text{final}} = w_p r_p + w_q r_q$$

$$w = \text{weight}, r = \text{rank}, x_p = \text{performance}, x_q = \text{quality}$$

$$s_q = w_i s_i + w_u s_u + w_d s_d$$

$$s = \text{score}, x_i = \text{installation}, x_u = \text{usability}, x_d = \text{documentation}$$

Summary – Important Aspects of HPC Software

- In HPC, performance and scalability are essential
- A well documented, easy to use and portable build process
- User interaction is still important, consider how data will go in and out - support common, open formats
- Good documentation:
  - Tutorials
  - Examples
  - Source Comments

Thank You For Listening

Any Questions?