Science & Technology Facilities Council

Parallel IO in Code_Saturne

Charles MOULINEC

Vendel SZEREMI

STFC Daresbury Laboratory, UK

Acknowledgements to:

Yvan Fournier from EDF R&D, FR

CCP12, UKTC and The Hartree Centre

ARCHER/PRACE Training – 2-3 Sept 14

Contents

Code_Saturne

Main Features and Toolchain

Two Applications

Motivation

Code_Saturne IO Methods

On the Fly Mesh Generation: Mesh Multiplication

Test Architectures and Test Cases

Scalability at Scale

I/O using HECToR (Lustre)

Results - ARCHER (Lustre) vs Blue Joule (GPFS)

Conclusions - Perspectives

Science & Technology Facilities Council

Code_Saturne

- Code_Saturne is developed by EDF (France)
- Computational Fluid Dynamics
- open source
- Fortran, C, Python

- fully validated production versions with long-term support every two years (currently 3.0)
- development versions

- http://code-saturne.org

Code_Saturne's Features

Technology

-Co-located finite volume, arbitrary unstructured meshes, predictor-corrector
-350 000 lines of code, 37% Fortran, 50% C, 13% Python
-MPI for distributed-memory and some openMP for shared-memory machines

Physical modeling

-Laminar and turbulent flows: k-eps, k-omega, SST, v2f, RSM, LES models
-Radiative transfer (DOM, P-1)
-Coal, heavy-fuel and gas combustion
-Electric arcs and Joule effect
-Lagrangian module for particles tracking
-Atmospheric modeling (merging Mercure_Saturne)
-ALE method for deformable meshes
-Rotor / stator interaction for pump modeling, for marine turbines

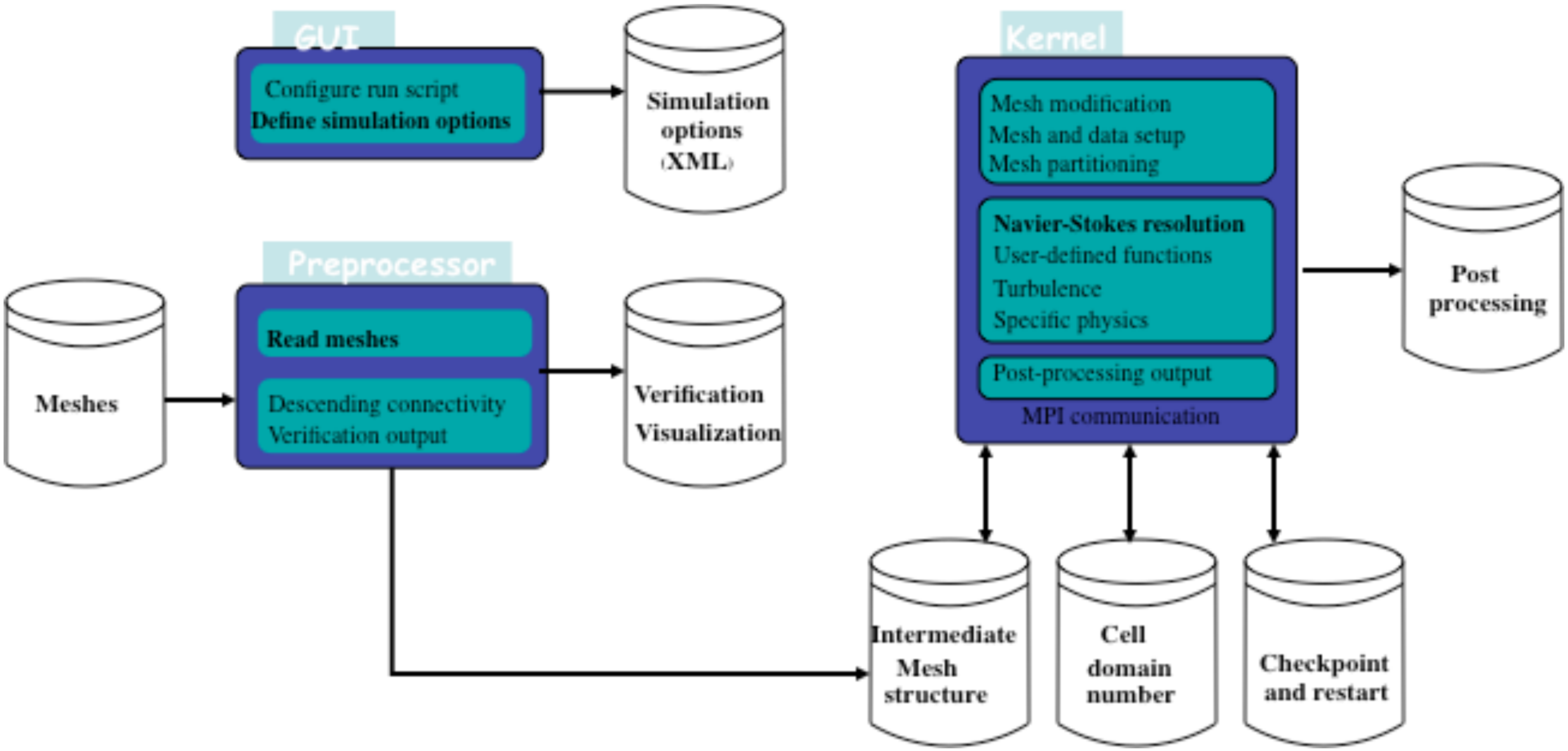
Flexibility

-Portability (Unix, Linux and MacOS X)
-Graphical User Interface with possible integration within the SALOME platform

Toolchain

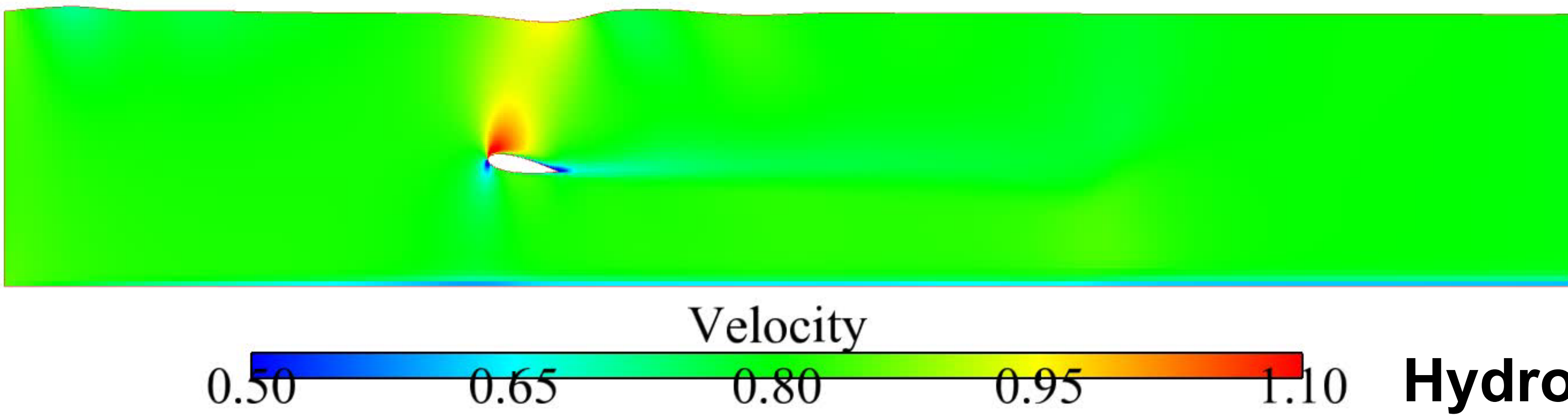
Reduced number of tools

- Each with rich functionality
- Natural separation between interactive and potentially long-running parts
- In-line (pdf) documentation



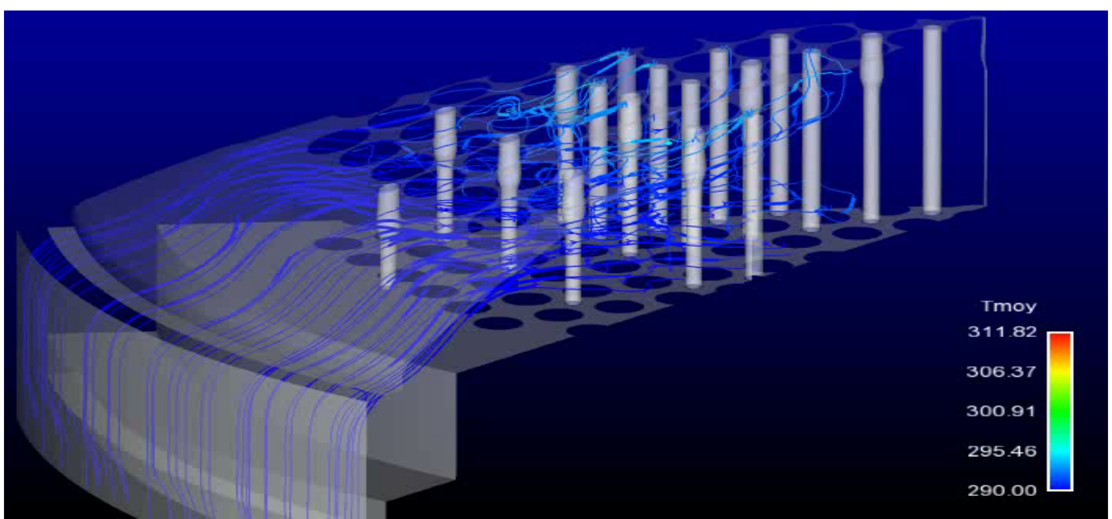
Example Applications

Free surface modelling (ALE)



Hydrofoil

Thermofluids study of the hot box dome AGR (EDF Energy)



- Complex flow due through the forest of tubes
- Calculation shows little mixing in the centre of the dome
- Temperatures at the dome highest where thermocouples are located

Code_Saturne I/O

Different types of file I/O

- read input
- write checkpoint data periodically
- read checkpoint if restarting a previous simulation
- write output

Different methods for I/O

- STD C IO
- MPI IO

Motivation (1)

High-End Machines offer hope for more multi-physics & multi-scale for engineering in ever more detailed configurations.

Huge effort has been dedicated to improve/optimise solvers (in our case Navier-Stokes equation solvers) for them to scale on the current existing petaflop machines, but arguably less time is dedicated by CFD developers to IOs.

Several types of IOs and some way around loading/writing huge data files have been identified:-

-INPUT: mesh, domain partition (if already known), restart file (if needed), input data

-OUTPUT: mesh (if changed, with added periodicity for instance), domain partition (if computed by the code), listing file, post-processing file, checkpoint, probes

Motivation (2)

Ways around exist to avoid loading full data set for:-

-INPUT:-

-mesh (mesh joining and mesh multiplication)

-domain partition (partition re-computed by the code)

-OUTPUT:-

-pre-processed mesh (not needed, because computed by the code)

-domain partition (not needed because computed by the code)

-post-processing (co-processing, for instance using Catalyst)

But not for:-

-INPUT:-

-restart file, as/if the whole flow field is needed

-OUTPUT:-

-checkpoint file, as/if the whole flow field is needed

I/O Methods in 3.3.1

I/O Method	
CS_FILE_STDIO_SERIAL	Serial standard C IO (funnelled through rank 0 in parallel)
CS_FILE_STDIO_PARALLEL	Per-process standard C IO
CS_FILE_MPI_INDEPENDENT	Non-collective MPI-IO with independent file open and close
CS_FILE_MPI_NON_COLLECTIVE	Non-collective MPI-IO with collective file open and close
CS_FILE_MPI_COLLECTIVE	Collective MPI-IO

I/O Methods in 3.3.1

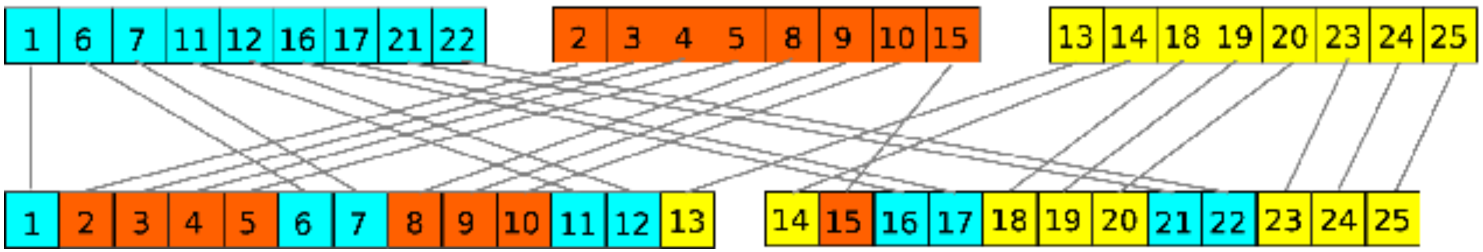
Selecting the I/O method

- **GUI and XML file**
    - -> "Calculation Management"
      -> "Performance Tuning"

- **Directly:**
    - Can be set in the cs_user_performance_tuning file in cs_user_parallel_io()
    - Can also provide MPI IO hints

Block-Based IO

Use global numbering
Redistribution on n blocks
- n blocks ≤ n cores
- Minimum block size may be set to avoid many small blocks (for some communication or usage schemes), or to force 1 block (for I/O with non-parallel libraries)
- Rank 0 is collecting info from the blocks



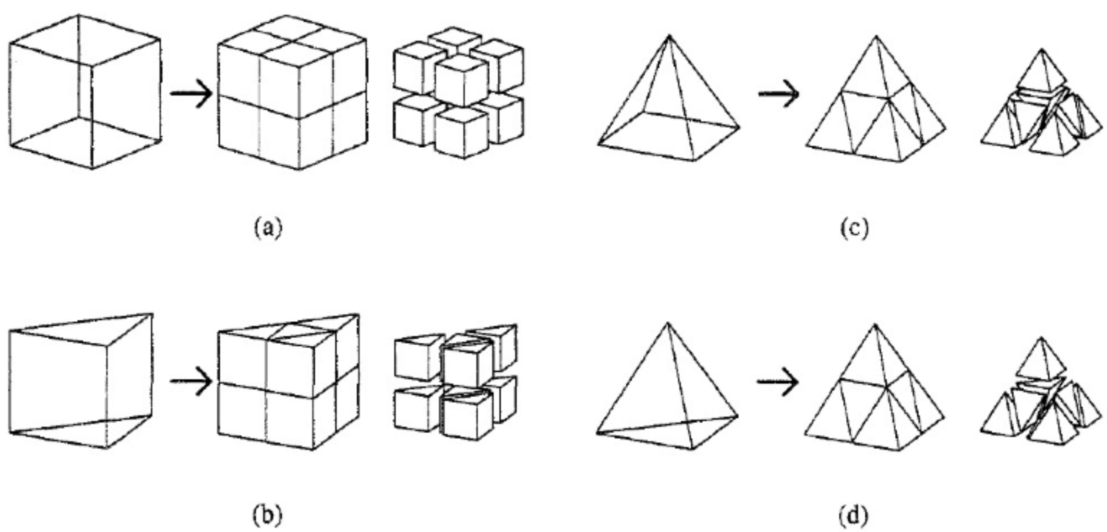
Mesh Multiplication

Most mesh generators are serial and thus memory-limited

A way around to generate extremely large meshes is to build meshes from existing coarse ones and globally refine each cell

This process might be repeated several times

Developed by Ales Ronovsky (VSB, PRACE)



Architectures

ARCHER – XC30 / Lustre

3008 Compute nodes: two 2.7 GHz, 12-core E5-2697 v2 (Ivy Bridge) series processors. Within the node, QuickPath Interconnect (QPI) links to connect the 2 processors

The Cray Aries interconnect links all compute nodes in a Dragonfly topology.

Compute nodes access the file system via IO nodes running the Cray Data Virtualization Service (DVS)

Blue Joule – BGQ / GPFS

6 racks, each rack containing 1,024 16-core, 64 bit, 1.60 GHz A2 PowerPC processors.

All the racks have 8 IO nodes which connect the BGQ racks to the shared GPFS storage over Infiniband.

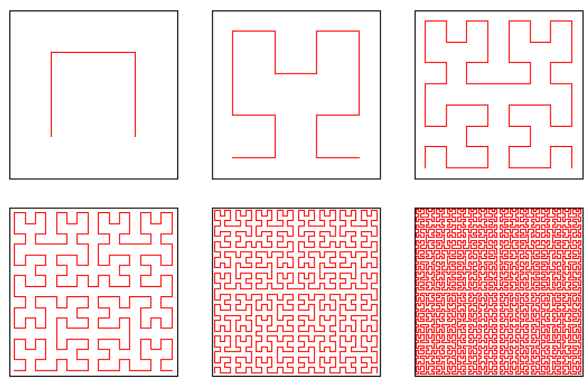
The minimum block size which can be booted for a job is therefore 1,024/8 nodes, or 128 nodes.

Test Case - Configuration

3D lid-driven cavity - fully unstructured mesh (tetras)

Size of the meshes:
MM Level 0 (13 million cells – Current production runs)
MM Level 1 (111 million cells – Current production runs)
MM Level 2 (890 million cells – Production runs in 2015)
MM Level 3 (7.2 billion cells – Production runs in 2016/2017)

Geometric partitioning using a Space-Filling Curve approach (Hilbert)



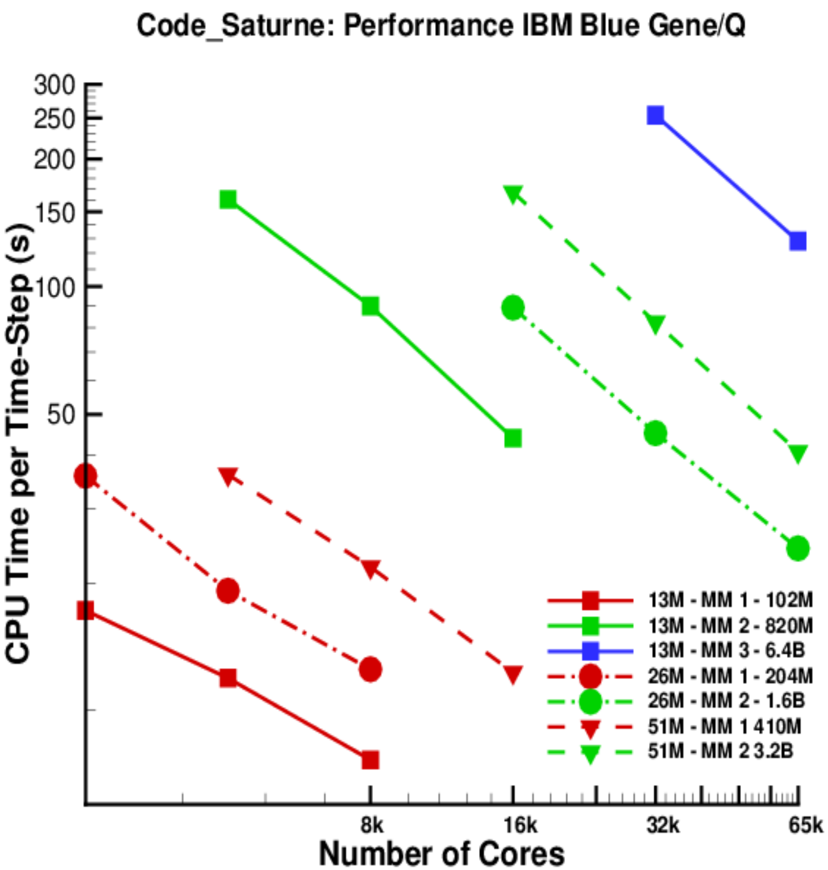
Note

IO tests are performed when the solver performance is still acceptable

If not stated, **machine default settings**. No striping for Lustre, for instance

Scalability at Scale (1)

Mesh generated by Mesh Multiplication



Hartree Centre
Science & Technology Facilities Council

105B Cell Mesh (MIRA, BGQ)

Cores	Time in Solver
262,144	652.59s
524,288	354.89s

13B Cell Mesh (MIRA, BGQ)

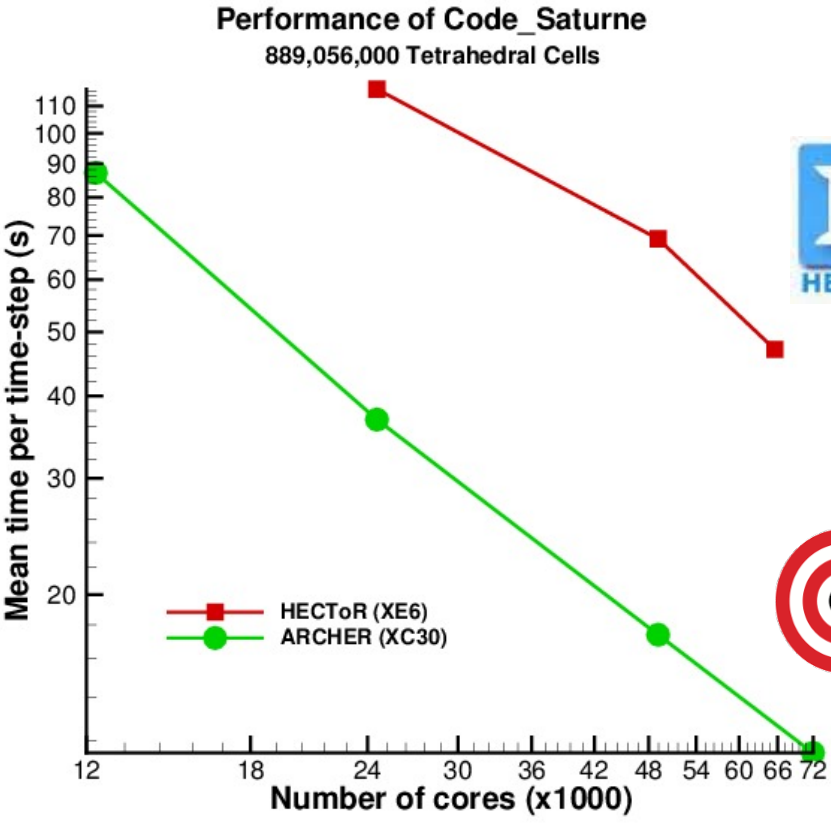
Nodes/Ranks	Time in Solver
16384/32	70.124s
32768/32	50.207s
49152/32	43.465s

Argonne NATIONAL LABORATORY

Scalability at Scale (2)

Comparison HECToR – ARCHER

Mesh generated by Mesh Multiplication
Cube meshed with tetra cells



IO HECToR (Lustre)

Comparison IO per Blocks (Ser-IO) and MPI-IO
Comparison Lustre (Cray) / GPFS (IBM BlueGene/Q)

	HECToR		Blue Joule	
Cores	MPI-IO	Ser-IO	MPI-IO	Ser-IO
2048	633	1203	-	-
4096	608	640	85	1279
8192	859	1147	86	1300
16384	732	747	67	1330
32768	-	-	59	1360

Tube Bundle

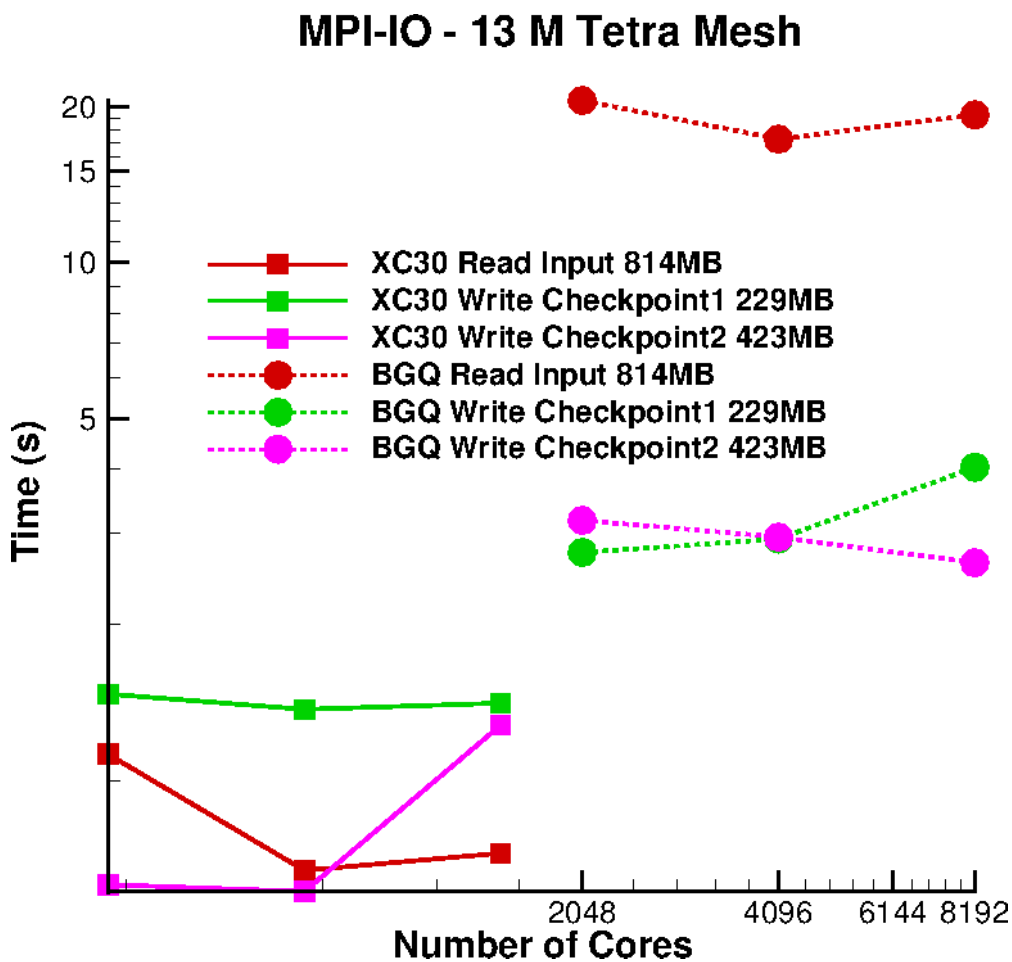
812M cells

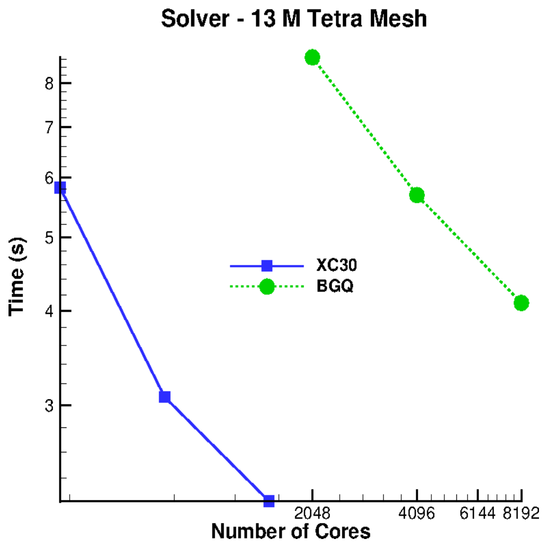
Block IO: ~same performance on Lustre and GPFS

MPI-IO: 8 to 10 times faster with GPFS

MM – Level 0

Writing Checkpoint Files

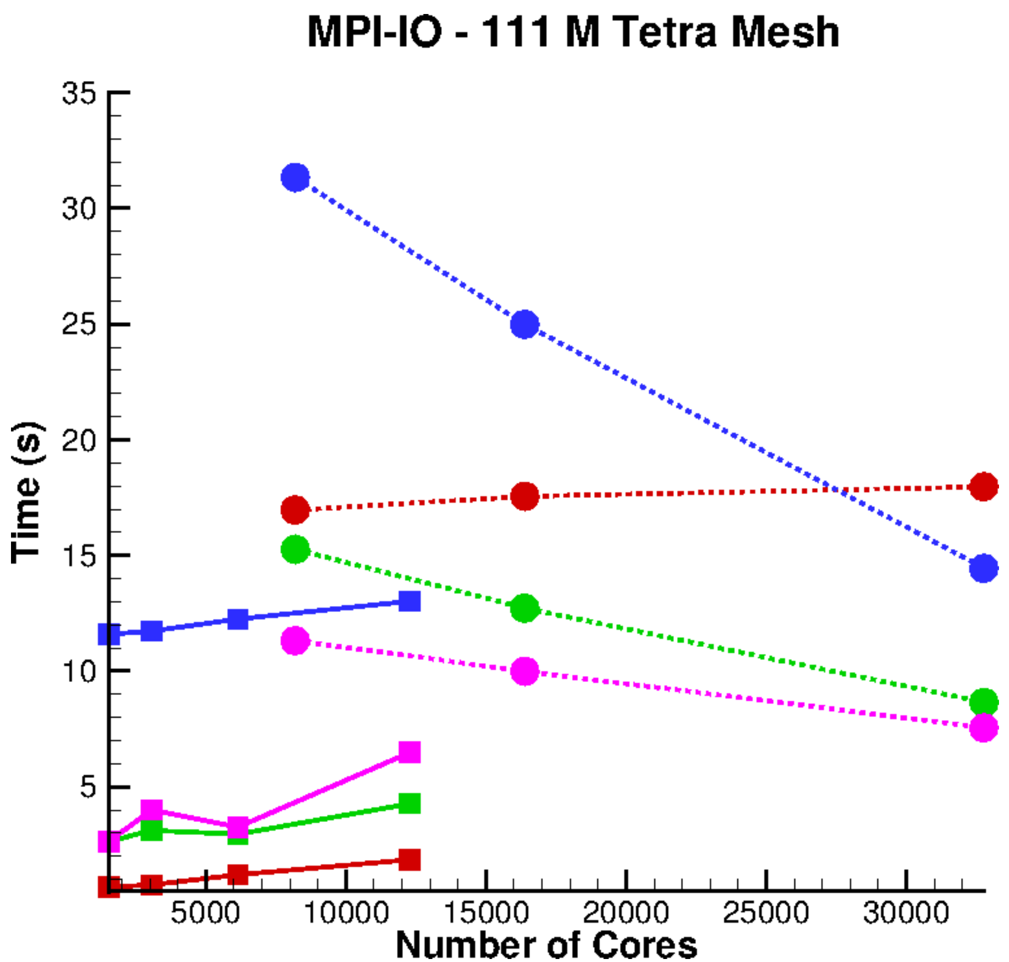


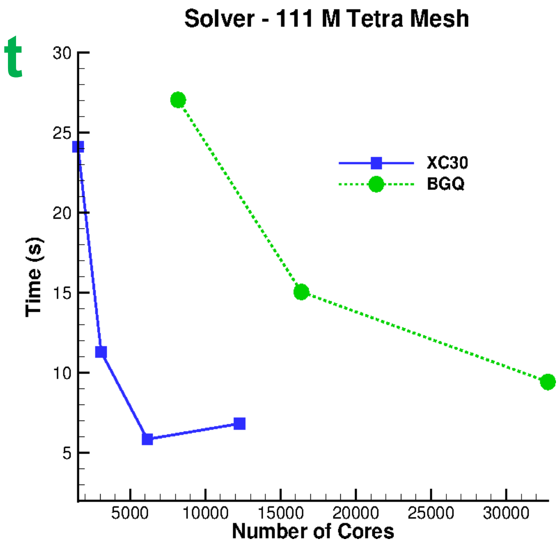


There is no mesh multiplication here

MM – Level 1

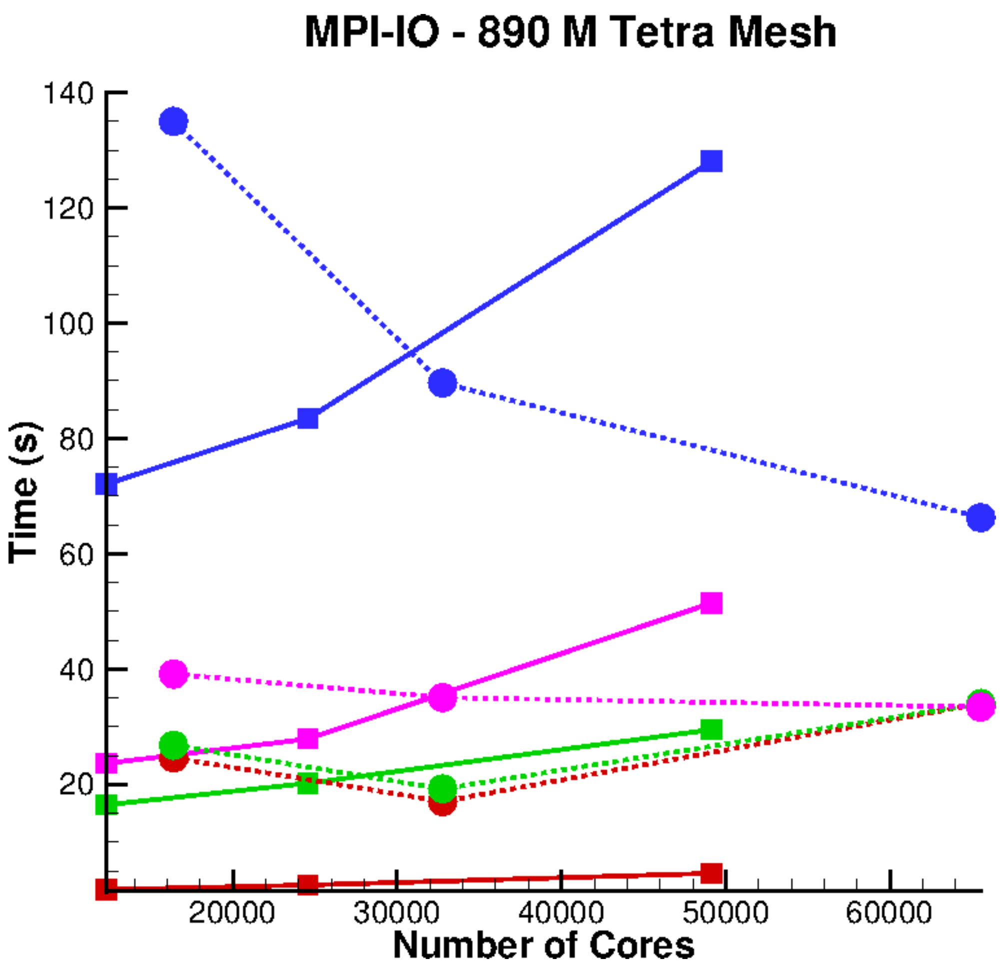
Writing Checkpoint Files – Mesh_Output

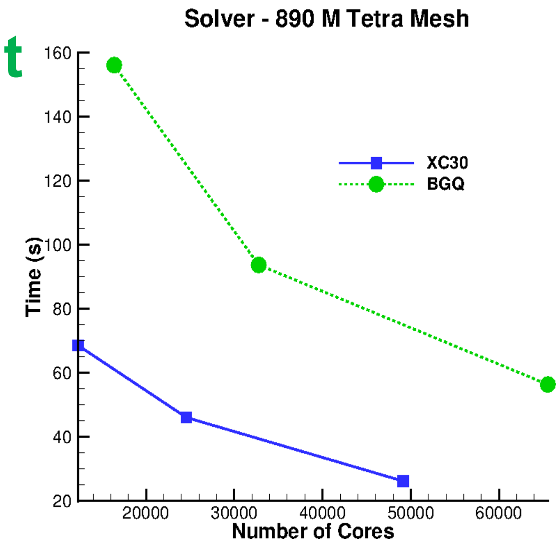




MM – Level 2

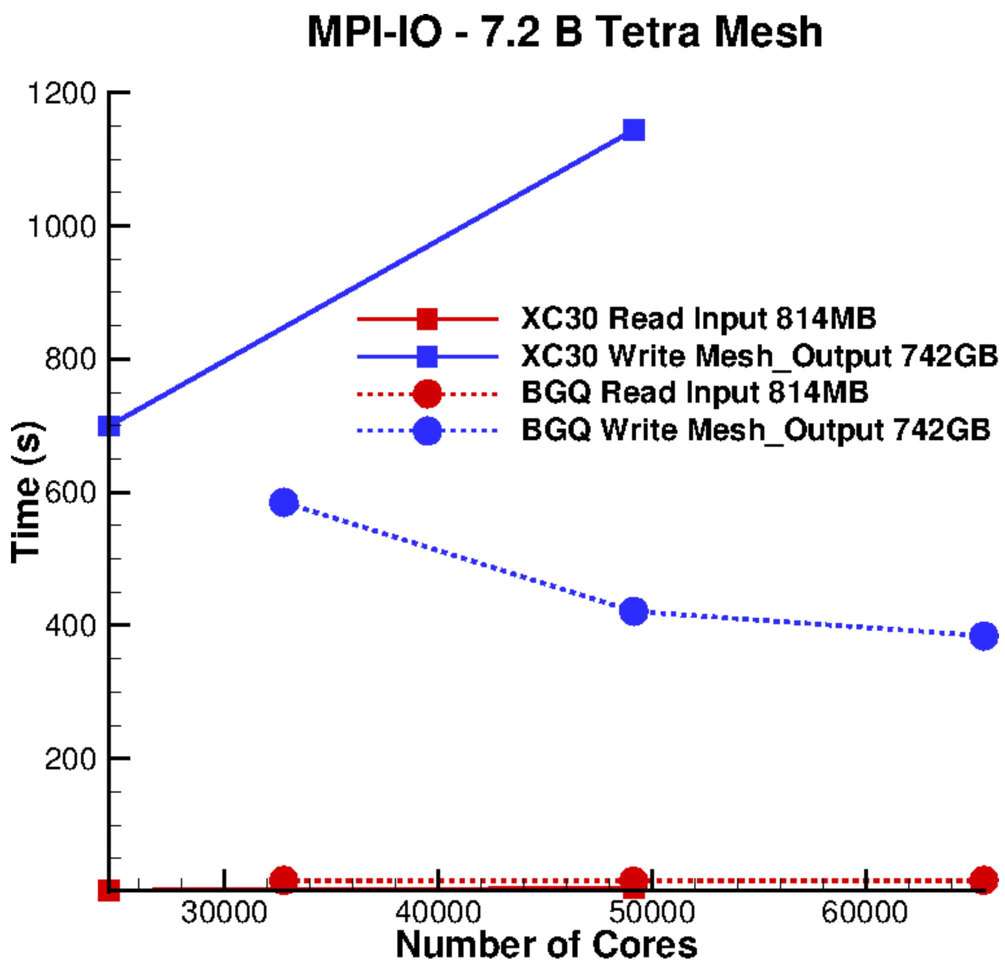
Writing Checkpoint Files – Mesh_Output

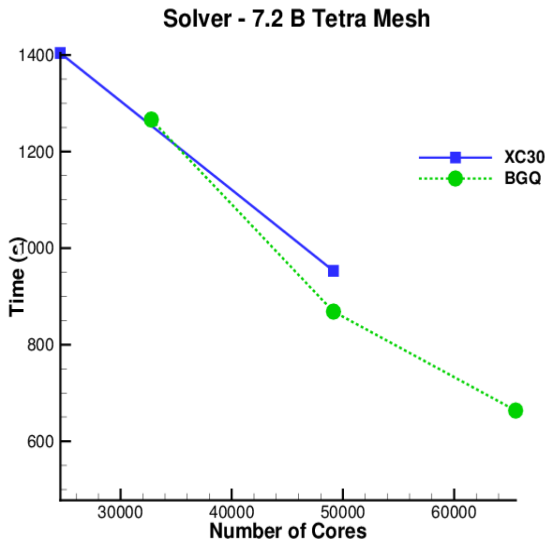




MM – Level 3

Writing Mesh_Output

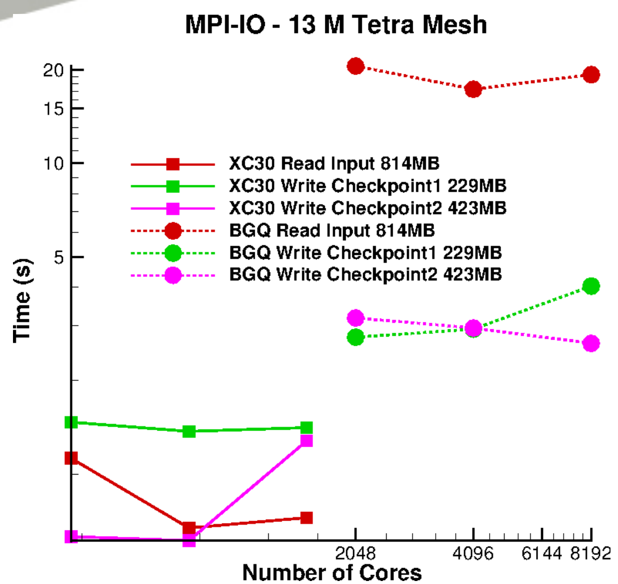


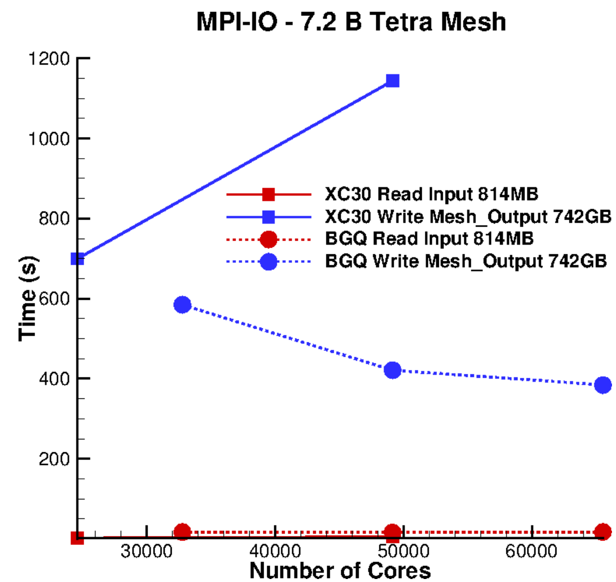


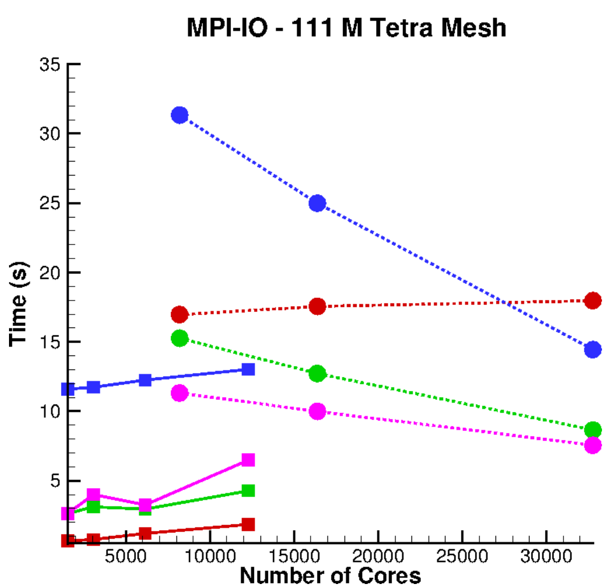
One time step only for the solver.

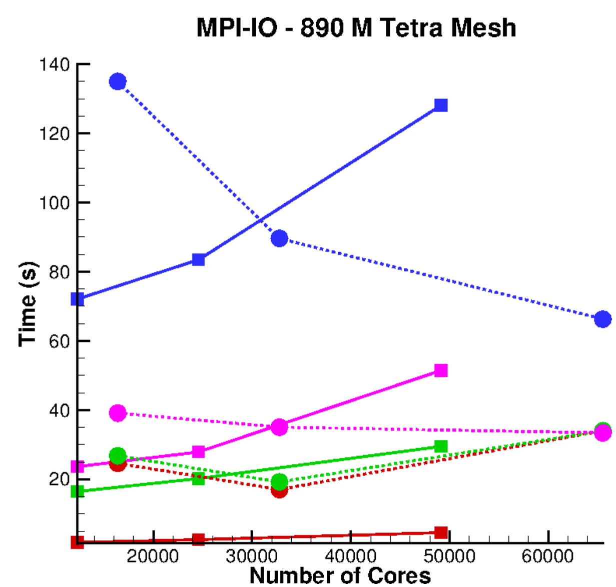
Timing also involves IOs

Quick Summary



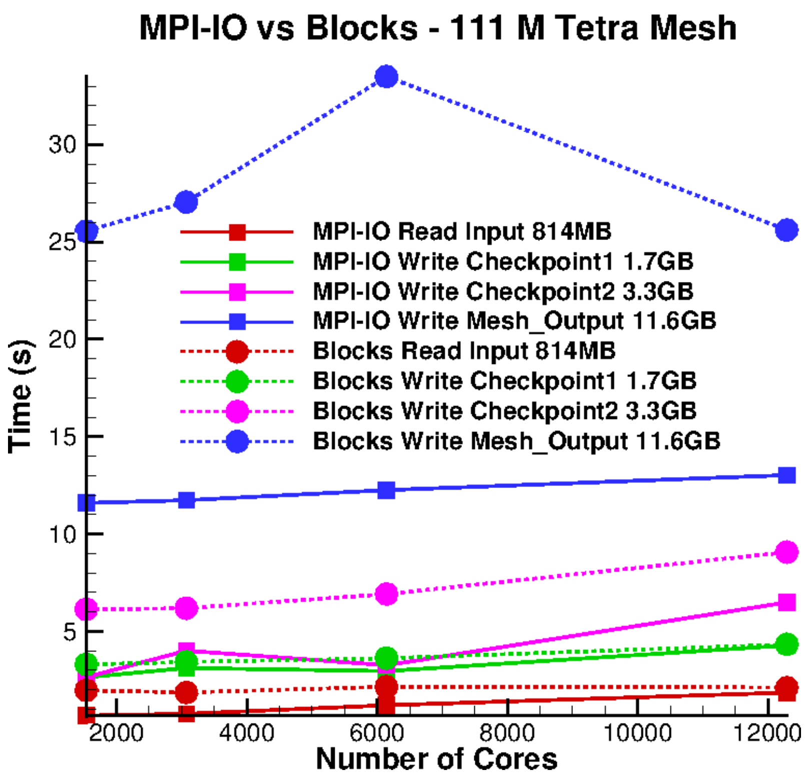


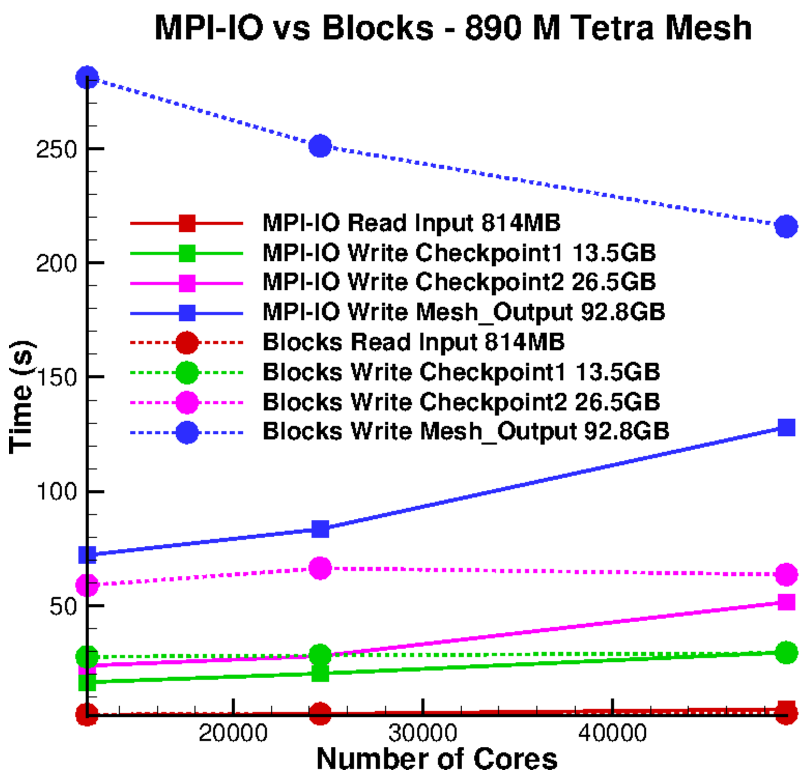




MPI – IO vs Block IO

Writing Checkpoint Files – Mesh_Output





Science & Technology Facilities Council

Conclusions

With the current machine/filesystem settings

MPI-IO

ARCHER (Lustre) better for small meshes than larger ones
BlueJoule (GPFS) better for large meshes than smaller ones

MPI-IO vs Block IO

If results on HECToR were comparable, much better obtained with MPI-IO on ARCHER

Science & Technology Facilities Council

LUSTRE Striping

Lustre and Striping

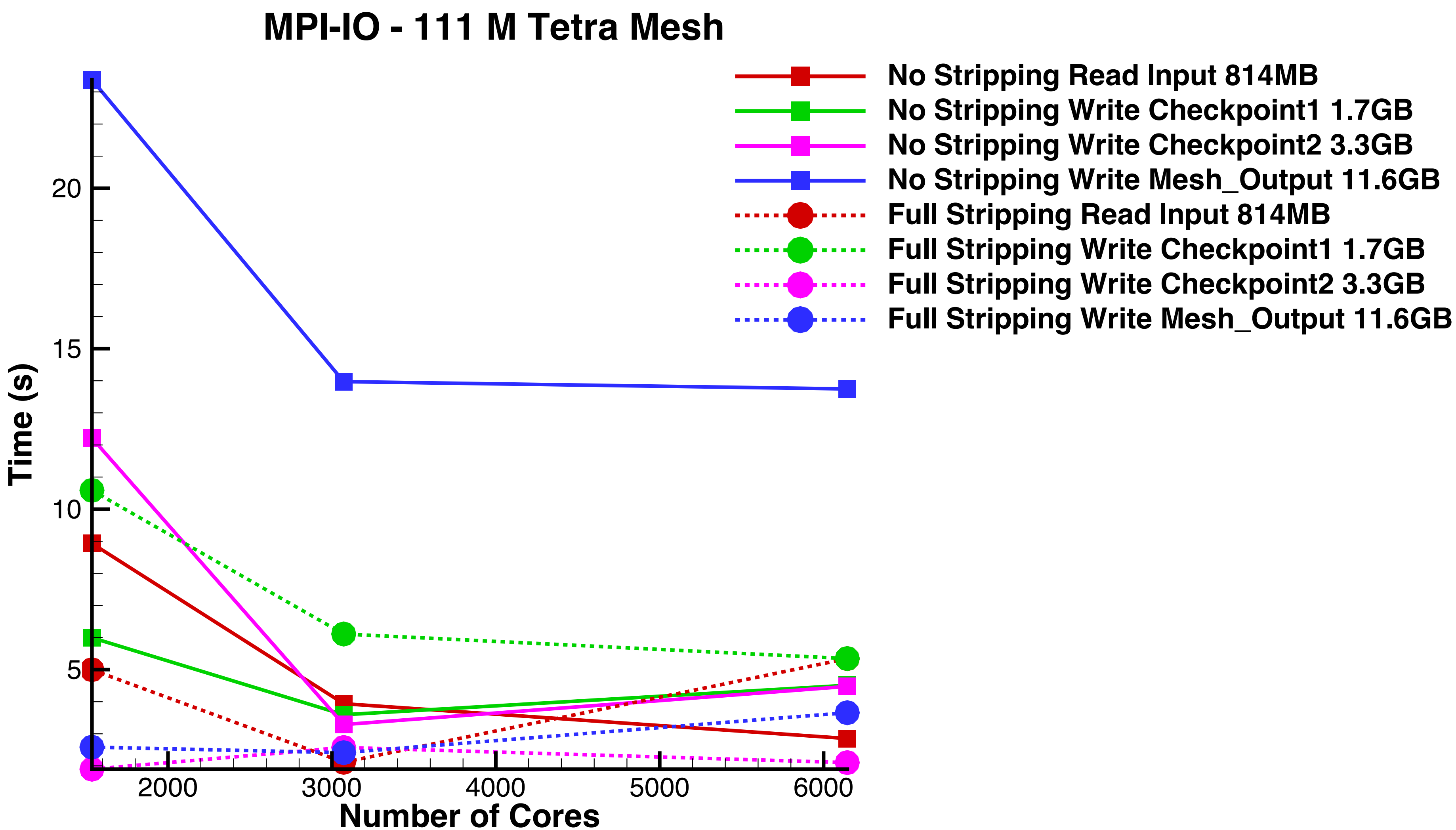
Previous ARCHER results used defaults for striping.

Use striping for better performance for large meshes?

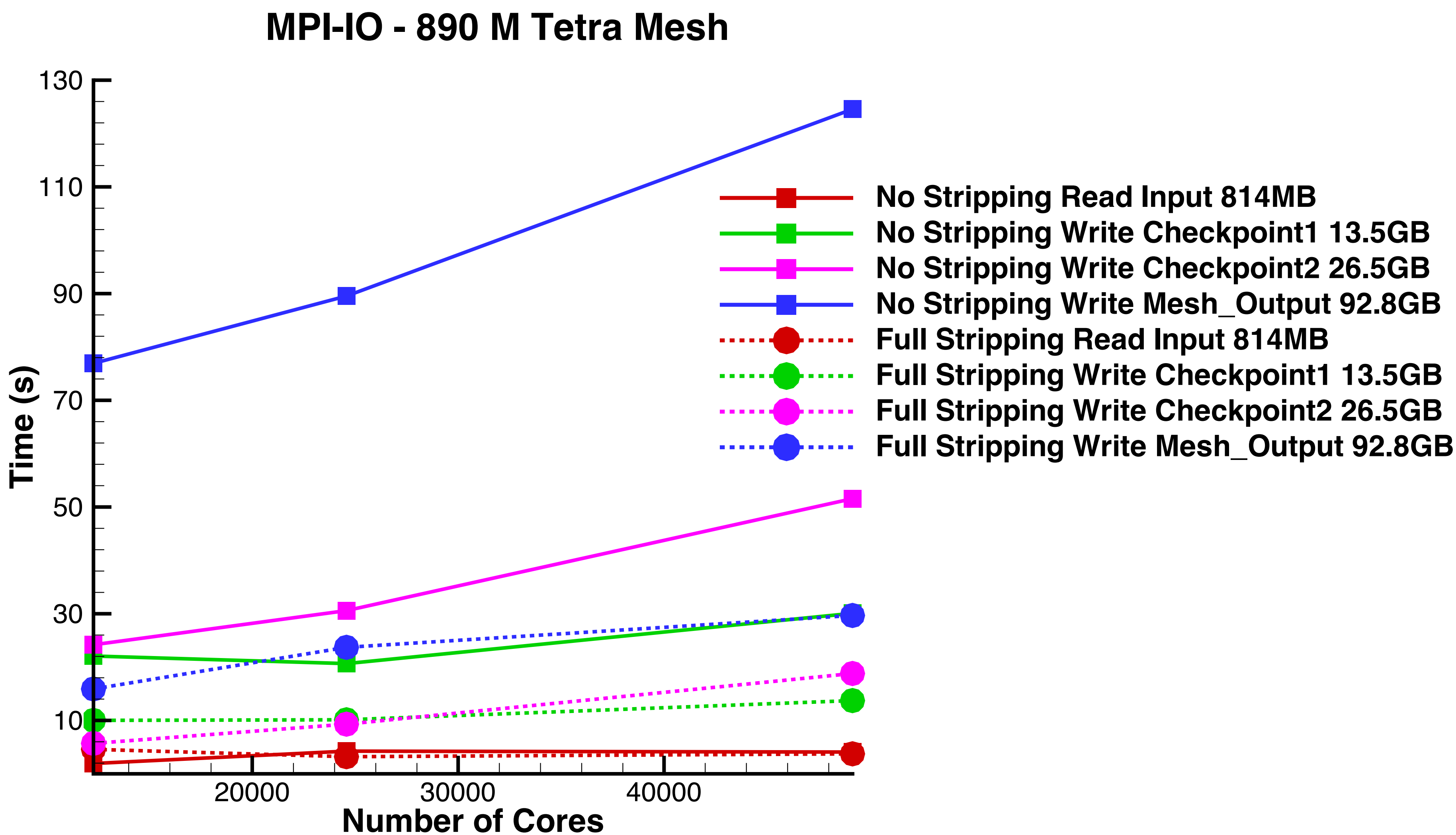
Stripe count for results directory set to all available OSTs with:

lfs setstripe

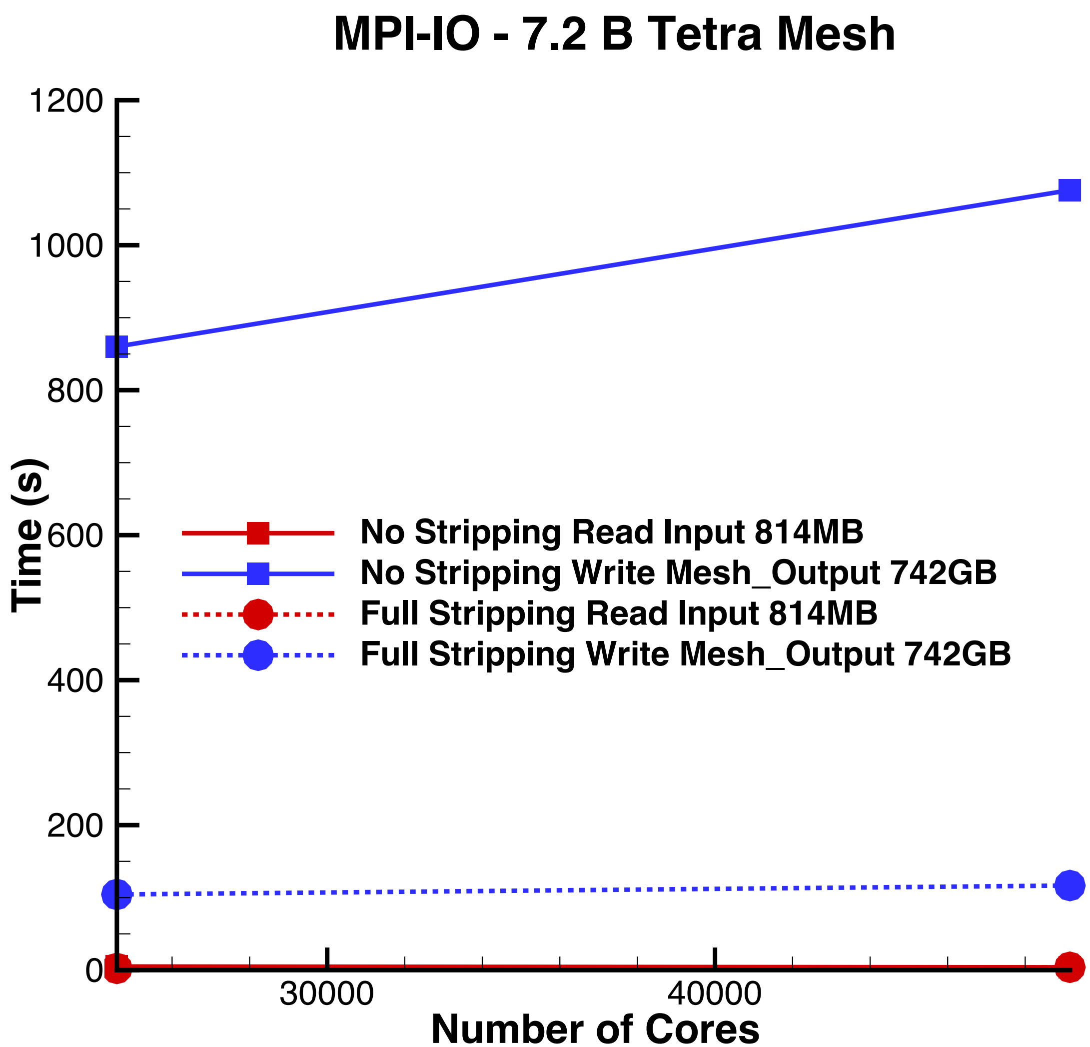
Striping – MM Level1



Striping – MM Level 2



Striping – MM Level 3



Perspectives

BGAS (Blue Gene Active Storage) System

The Active Storage Project is aimed at:-

-enabling close integration of emerging solid-state storage technologies with high performance networks and integrated processing capability

-exploring the application and middleware opportunities presented by such systems

-anticipating future scalable systems comprised of very dense Storage Class Memories (SCM) with fully integrated processing and network capability

Project to test Code_Saturne on the BGAS System

(Collaboration between STFC (the Hartree Centre) and IBM)

Science & Technology Facilities Council

THANK YOU FOR YOUR ATTENTION