

PERFORMANCE OF PARALLEL IO ON LUSTRE AND GPFS

David Henty and Adrian Jackson
(EPCC, The University of Edinburgh)
Charles Moulinec and Vendel Szeremi
(STFC, Daresbury Laboratory)




Outline

- Parallel IO problem
- Common IO patterns
- Parallel filesystems
- MPI-IO Benchmark results
- Filesystem tuning
- MPI-IO Application results
- HDF5 and NetCDF
- Conclusions

Parallel IO problem



4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13



1	2	3	4
---	---	---	---

Process 4



1	2	3	4
---	---	---	---

Process 2



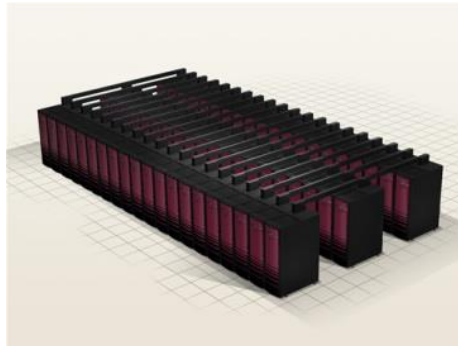
1	2	3	4
---	---	---	---



1	2	3	4
---	---	---	---

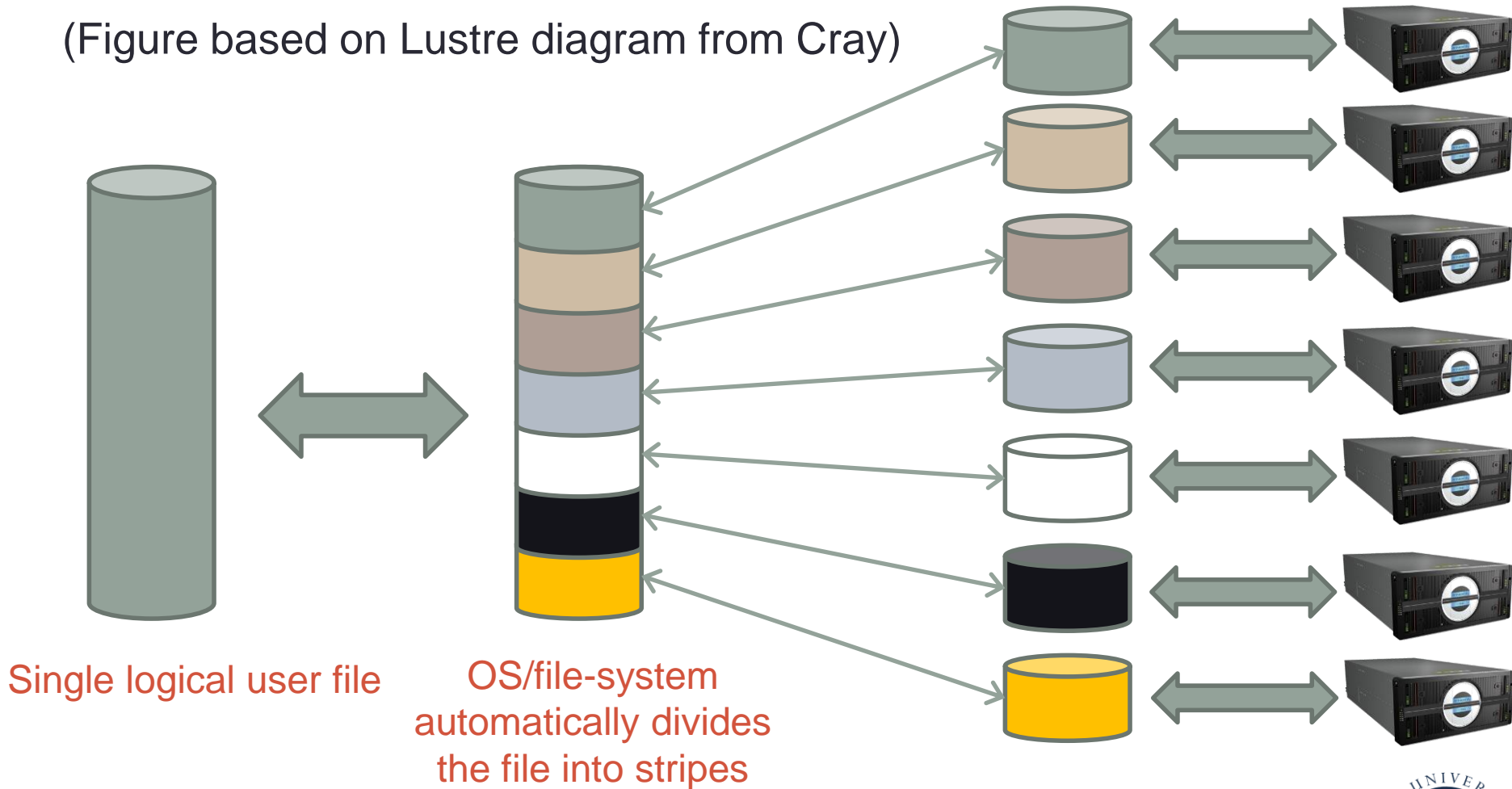
Process 1

Process 3



Parallel Filesystems

(Figure based on Lustre diagram from Cray)



Common IO patterns

- Multiple files, multiple writers
 - each process writes its own file
 - numerous usability and performance issues
- Single file, single writer (master IO)
 - high usability but poor performance
- Single file, multiple writers
 - all processes write to a single file; poor performance
- Single file, collective writers
 - aggregate data onto a subset of IO processes
 - hard to program and may require tuning
 - potential for scalable IO performance

Global description: MPI-IO

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

rank 1 (0,1)	rank 3 (1,1)
rank 0 (0,0)	rank 2 (1,0)

global file

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

rank 1 filetype

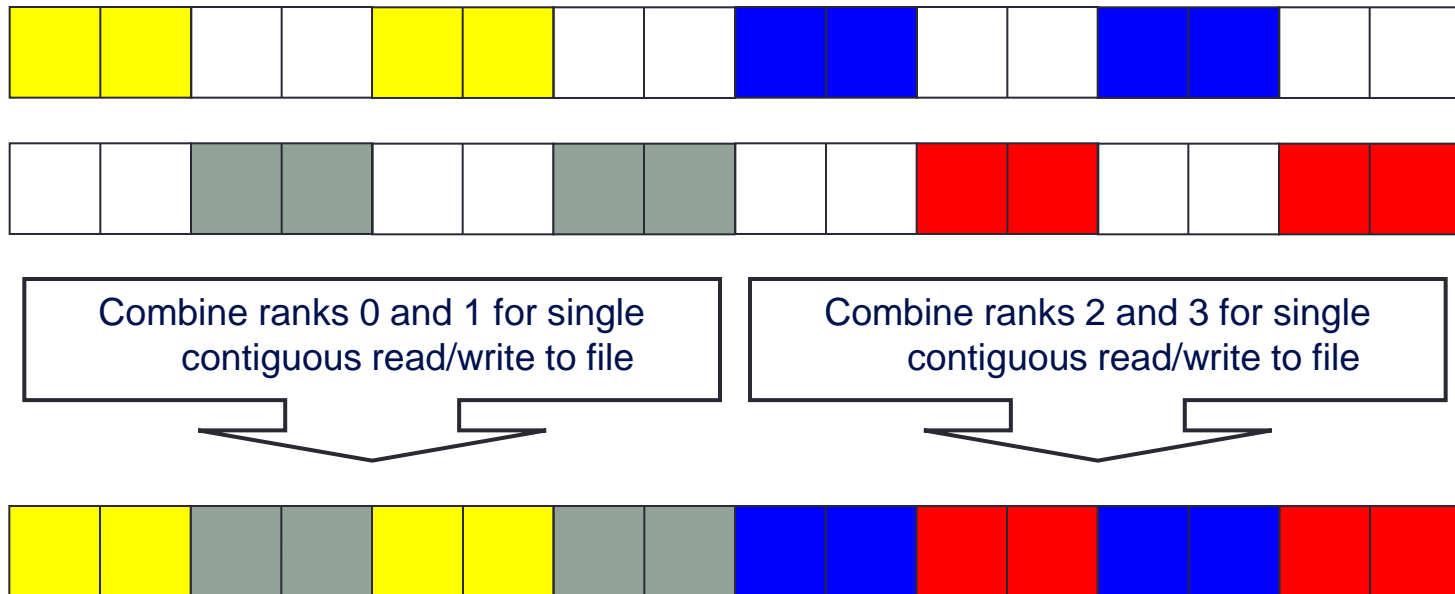
--	--	--	--	--	--	--	--

rank 1 view of file

3	4	7	8
---	---	---	---

Collective IO

- Enables numerous optimisations in principle
 - requires global description and participation of all processes
 - does this help in practice?



Cellular Automaton Model

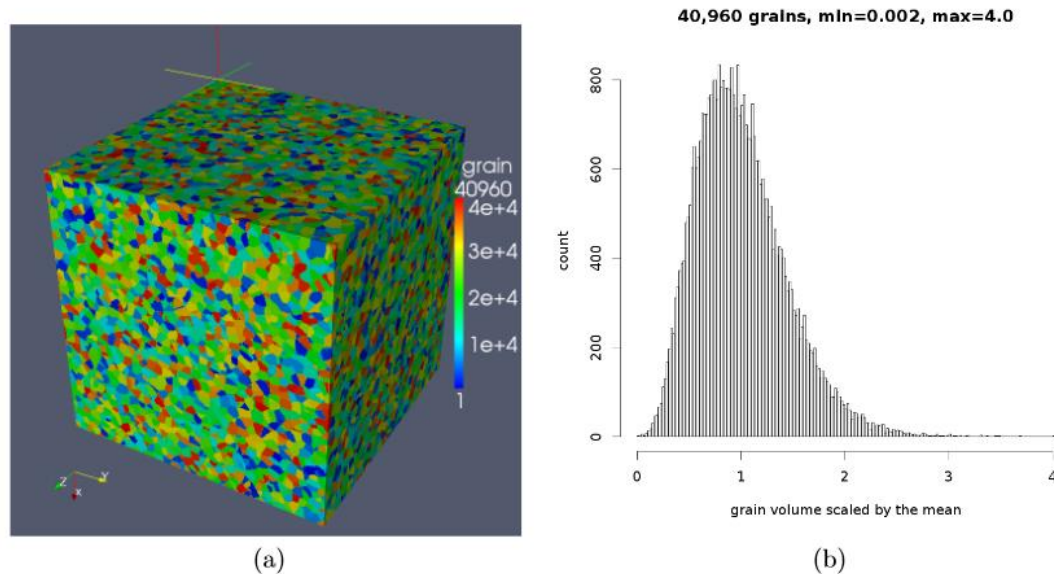


Figure 1: A 4.1×10^9 cell, 40,960 grain equiaxed microstructure model, showing (a) grain arrangement with colour denoting orientation; (b) grain size size (volume) histogram.

- *Fortran coarray library for 3D cellular automata microstructure simulation*, Anton Shterenlikht, proceedings of 7th International Conference on PGAS Programming Models, 3-4 October 2013, Edinburgh, UK.

Benchmark

- Distributed regular 3D dataset across 3D process grid
 - local data has halos of depth 1; set up for weak scaling
 - implemented in Fortran and MPI-IO

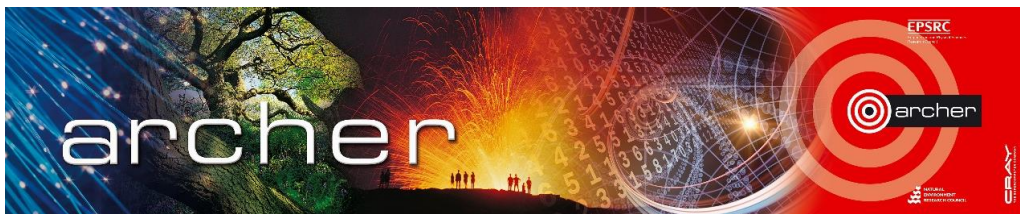
```
! Define datatype describing global location of local data
call MPI_Type_create_subarray(ndim, arraysize, arraysubsize, arraystart,
    MPI_ORDER_FORTRAN, MPI_DOUBLE_PRECISION, filetype, ierr)
```

```
! Define datatype describing where local data sits in local array
call MPI_Type_create_subarray(ndim, arraysize, arraysubsize, arraystart,
    MPI_ORDER_FORTRAN, MPI_DOUBLE_PRECISION, mpi_subarray, ierr)
```

```
! After opening file fh, define what portions of file this process owns
call MPI_File_set_view(fh, disp, MPI_DOUBLE_PRECISION, filetype,
    'native', MPI_INFO_NULL, ierr)
```

```
! Write data collectively
call MPI_File_write_all(fh, iodata, 1, mpi_subarray, status, ierr)
```

ARCHER XC30



Engineering and Physical Sciences
Research Council

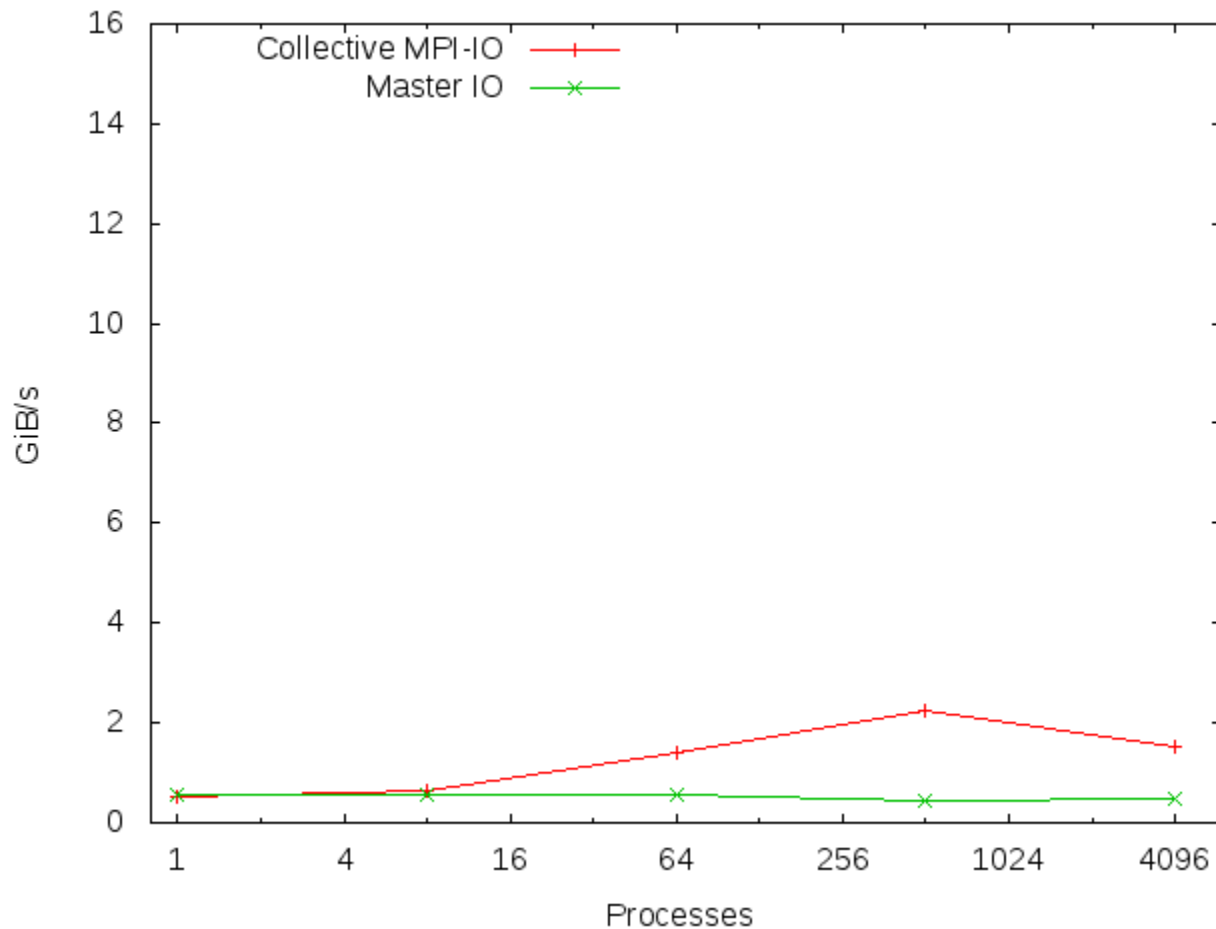


Single file, multiple writers

- Serial bandwidth on ARCHER around 400 to 500 MiB/s
- Use `MPI_File_write` not `MPI_File_write_all`
 - identical functionality
 - different performance

Processes	Bandwidth
1	49.5 MiB/s
8	5.9 MiB/s
64	2.4 MiB/s

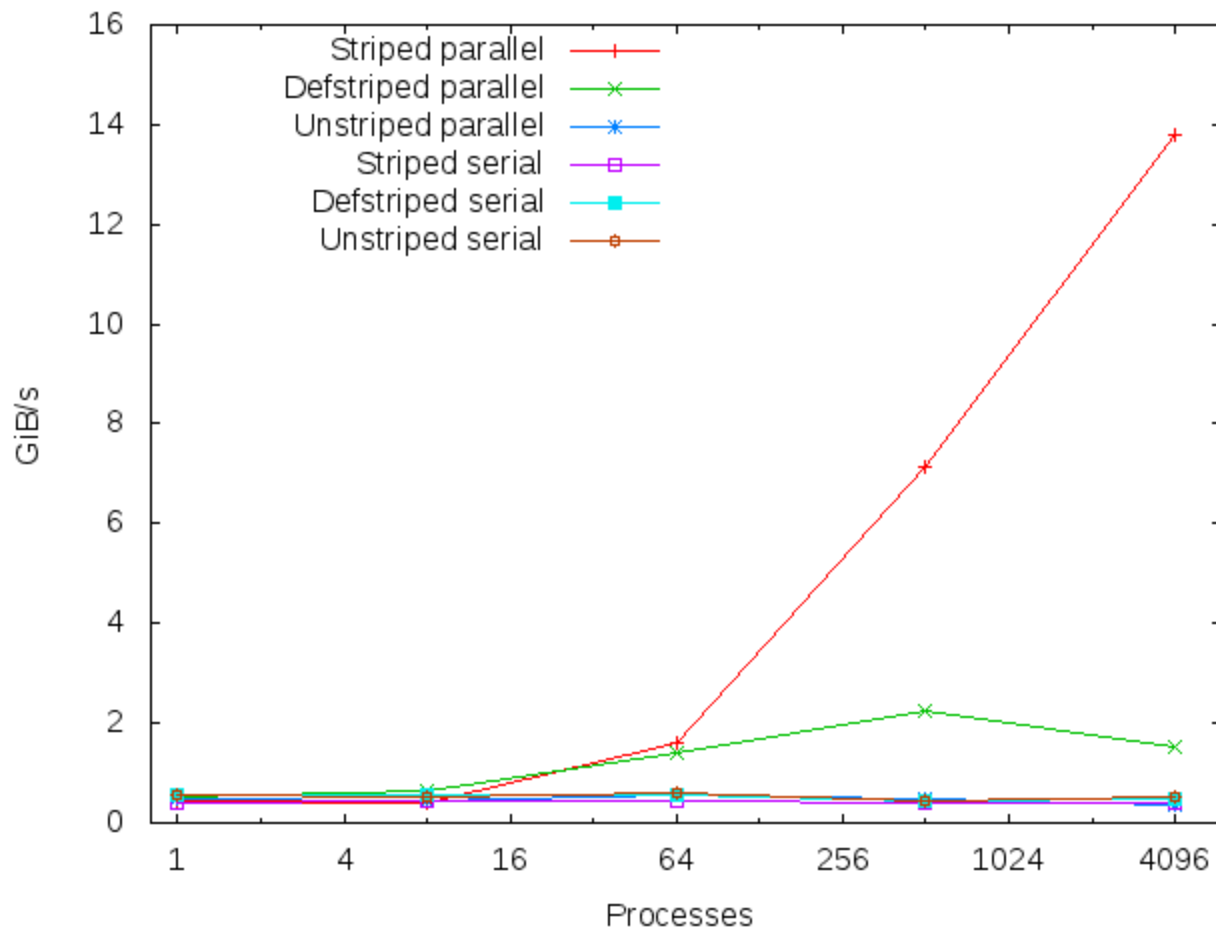
Single file, collective writers



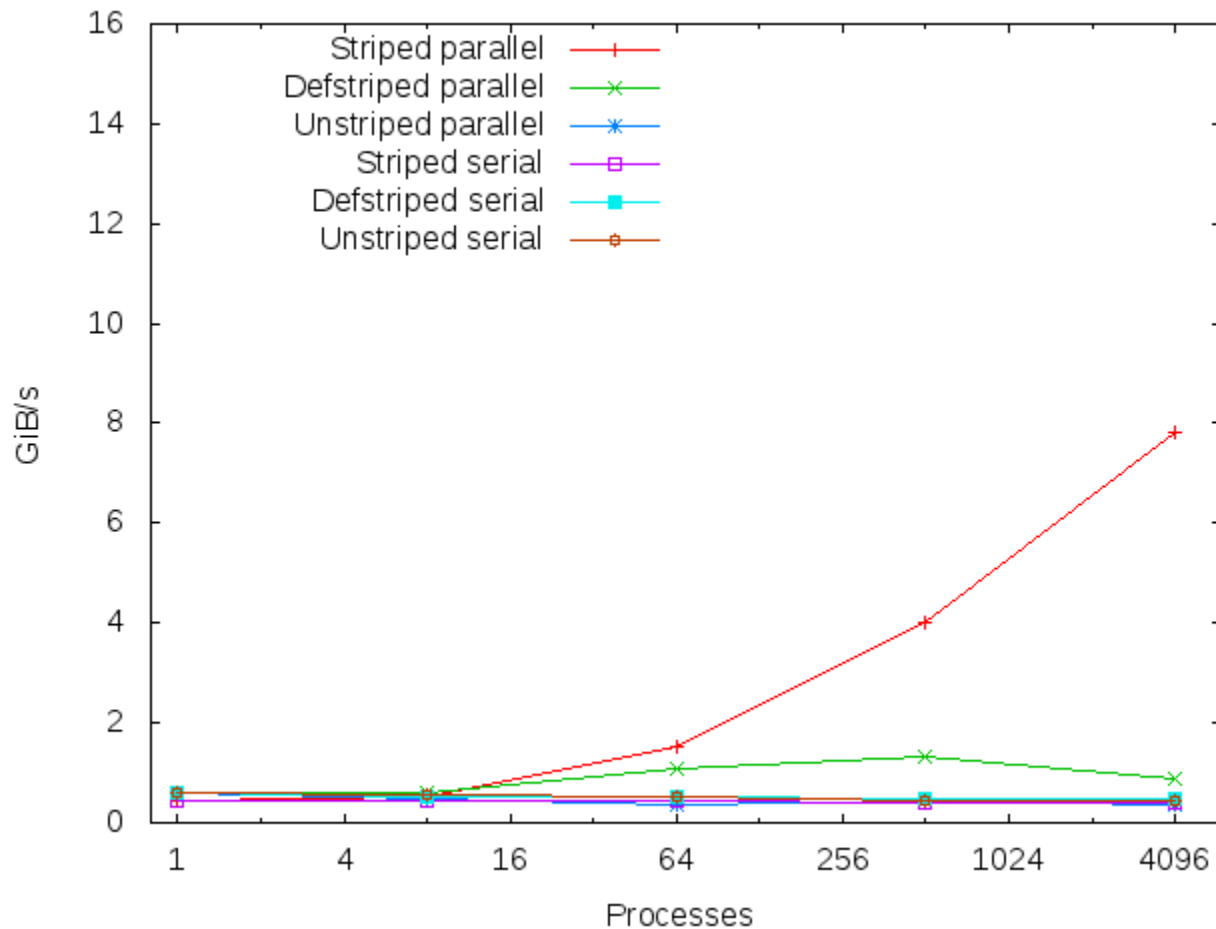
Lustre striping

- We've done a lot of work to enable (many) collective writers
 - learned MPI-IO and described data layout to MPI
 - enabled collective IO
 - MPI dynamically decided on number of writers
 - collected data and aggregates before writing
- ... for almost no benefit!
- Need many physical disks *as well as* many IO streams
 - in Lustre, controlled by the number of *stripes*
 - default number of stripes is 4; ARCHER has around 50 IO servers
- *User* needs to set striping count on a per-file/directory basis
 - `lfs setstripe -c -1 <directory> # use maximal striping`

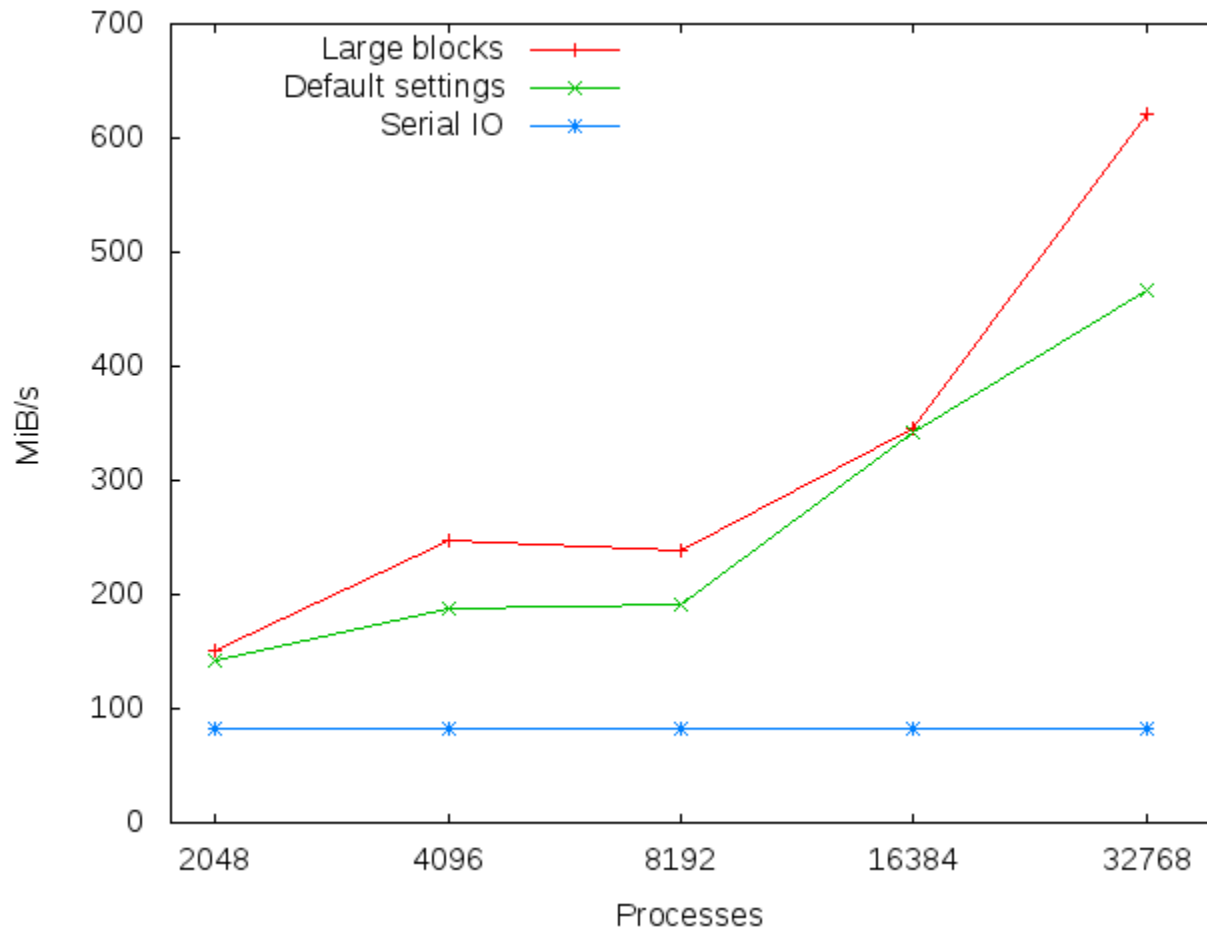
Cray XC30 with Lustre: 128^3 per proc



Cray XC30 with Lustre: 256^3 per proc



BG/Q: #IO servers scales with CPUs

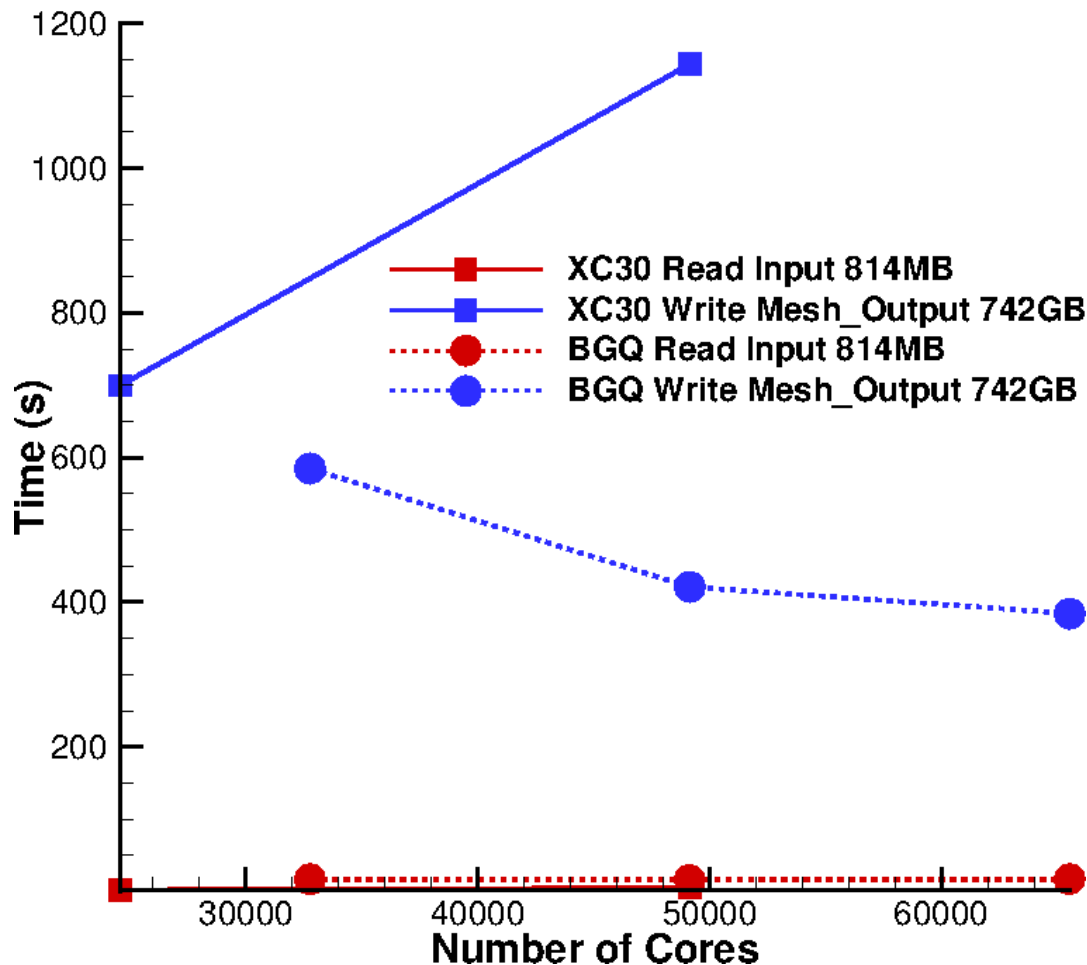


Code_Saturne <http://code-saturne.org>

- CFD code developed by EDF (France)
- Co-located finite volume, arbitrary unstructured meshes, predictor-corrector
- 350 000 lines of code
 - 50% C
 - 37% Fortran
 - 13% Python
- MPI for distributed-memory (some OpenMP for shared-memory) **including MPI-IO**
- Laminar and turbulent flows: k-eps, k-omega, SST, v2f, RSM, LES models, ...

Code_SATURNE: default settings

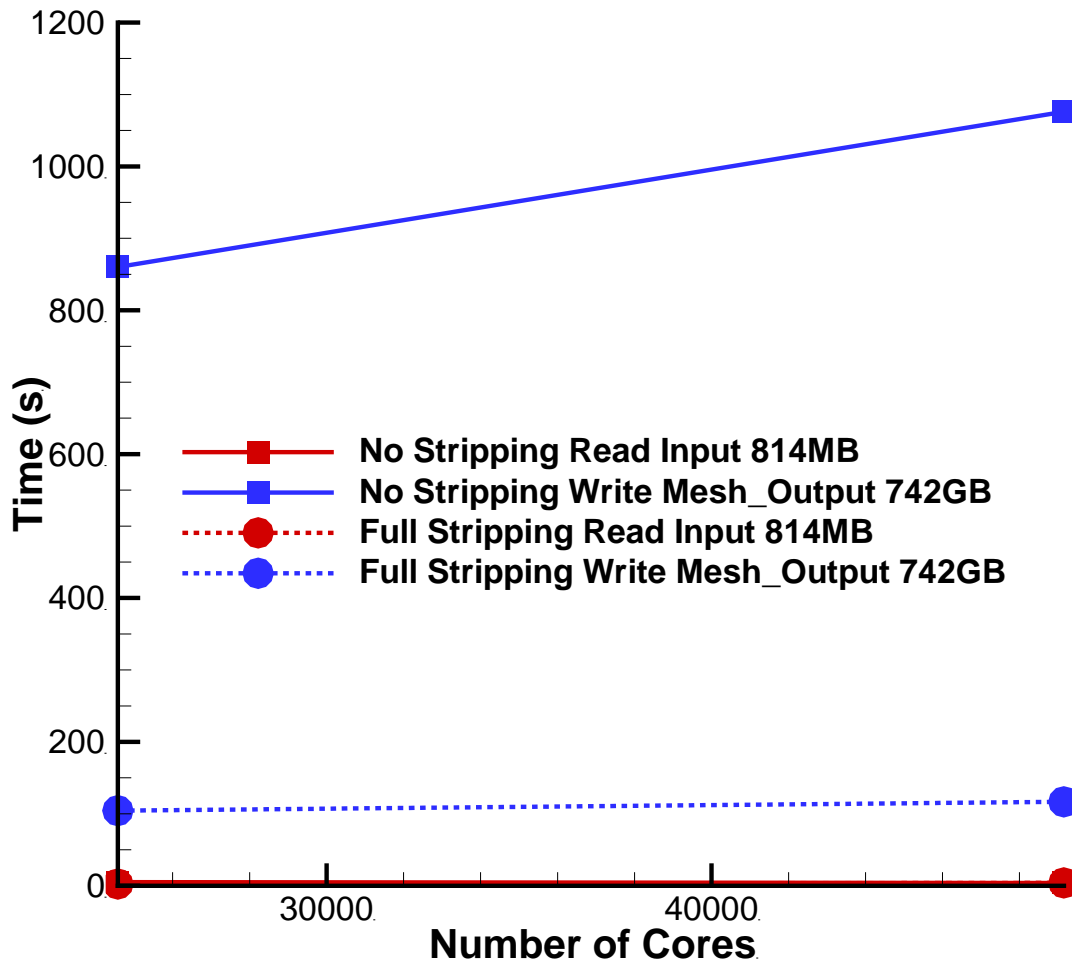
MPI-IO - 7.2 B Tetra Mesh



- Consistent with benchmark results
 - default striping Lustre similar to GPFS

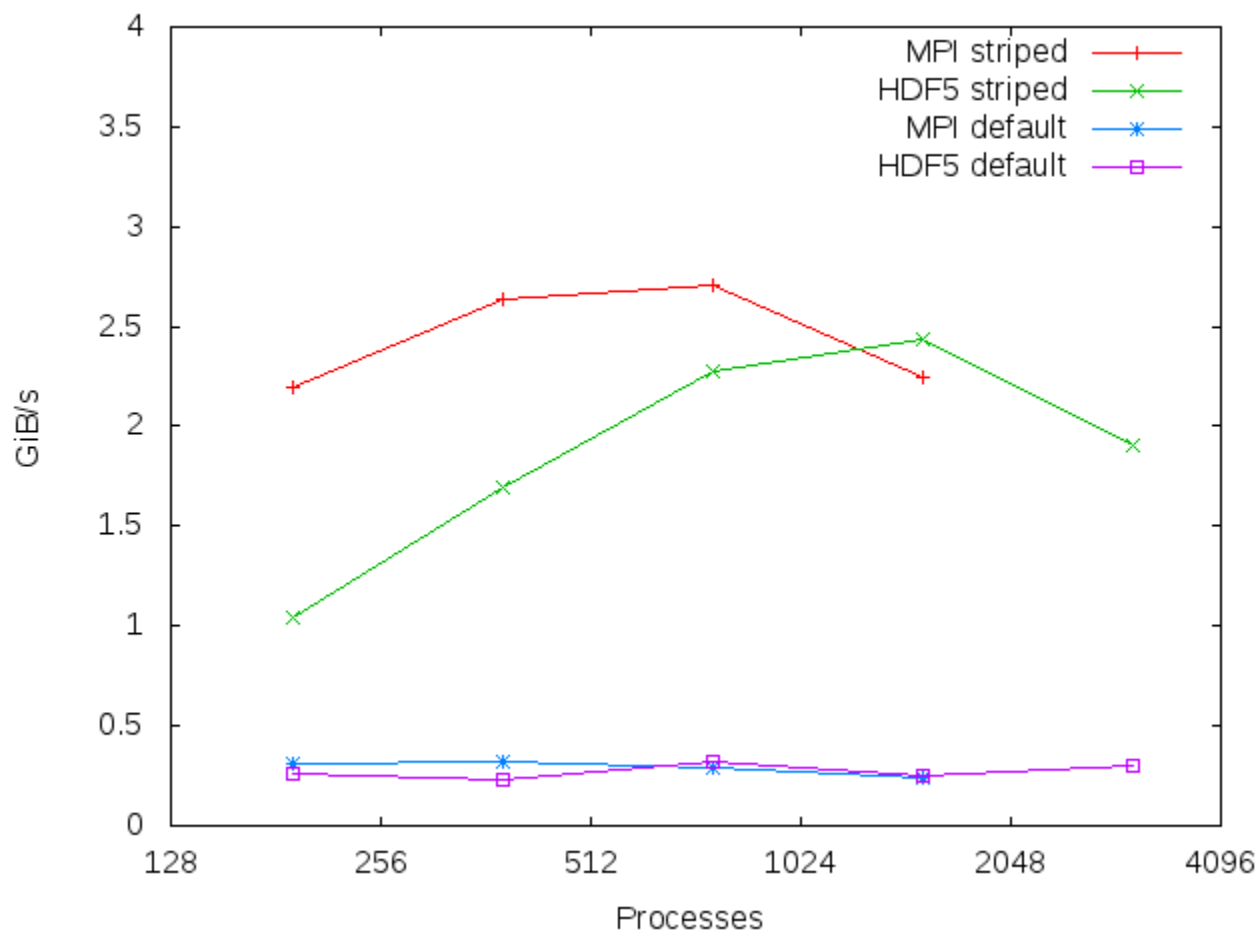
Code_Saturne: Lustre striping

MPI-IO - 7.2 B Tetra Mesh



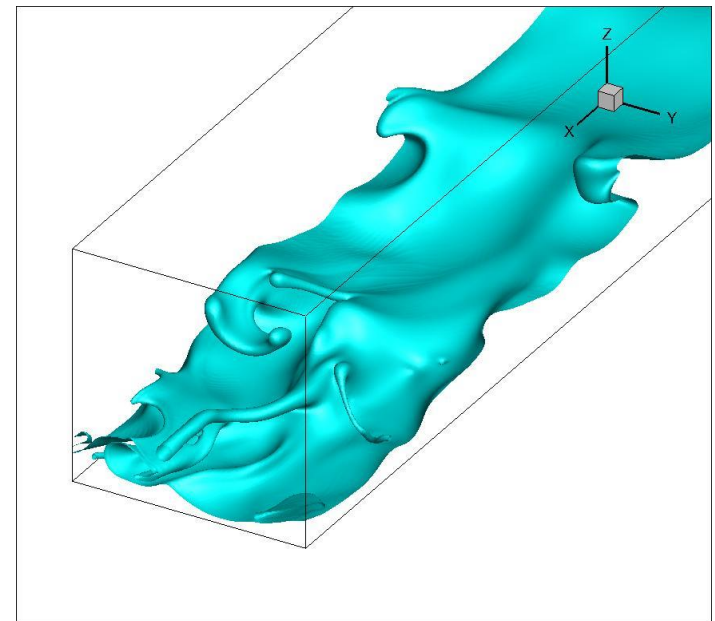
- Consistent with benchmark results
 - order of magnitude improvement from striping

Simple HDF5 benchmark: Lustre

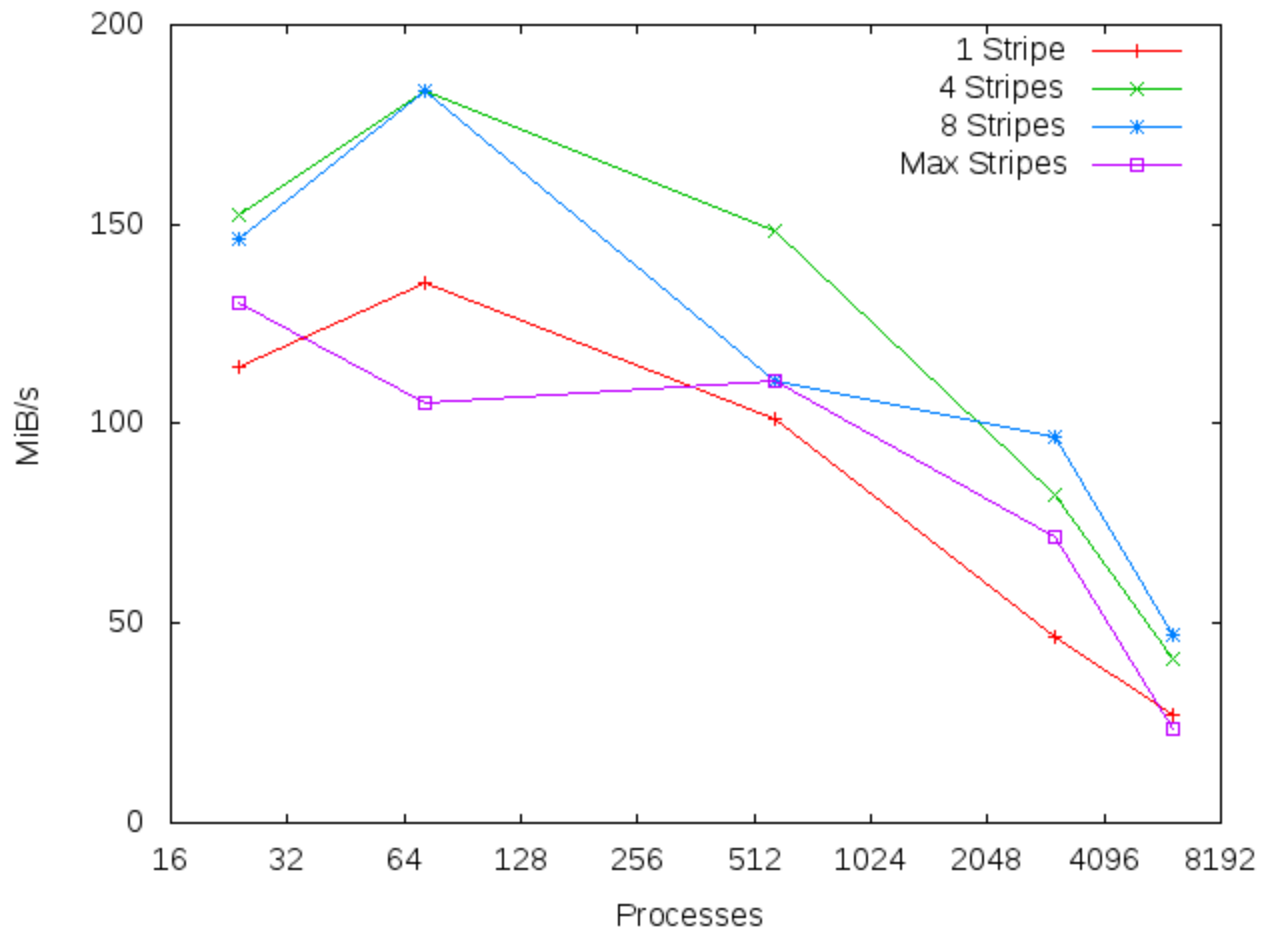


TPLS code

- Two-Phase Level Set: CFD code
 - simulates the interface between two fluid phases.
 - High resolution direct numerical simulation
- Applications
 - Evaporative cooling
 - Oil and gas hydrate transport
 - Cleaning processes
 - Distillation/absorption
- Fortran90 + MPI
 - IO improved by orders of magnitude
 - ASCII master IO -> binary NetCF
 - does striping help?



TPLS results



Further Work

- Non-blocking parallel IO could hide much of writing time
 - or use more restricted split-collective functions
 - extend benchmark to overlap comms with calculation
- I don't believe it is implemented in current MPI-IO libraries
 - blocking MPI collectives are used internally
- A subset of user MPI processes will be used by MPI-IO
 - would be nice to exclude them from calculation
 - extend `MPI_Comm_split_type()` to include something like `MPI_COMM_TYPE_IONODE` as well as `MPI_COMM_TYPE_SHARED` ?

Conclusions

- Efficient parallel IO requires **all** of the following
 - a global approach
 - coordination of multiple IO streams to the same file
 - collective writers
 - filesystem tuning
- MPI-IO Benchmark useful to inform real applications
 - NetCDF and HDF5 layered on top of MPI-IO
 - although real application IO behaviour is complicated
- Try a library before implementing bespoke solutions!
 - higher level view pays dividends