Advanced Message-Passing Programming

Efficient use of the Lustre Filesystem

Caution!

- IO benchmarking is very difficult and can be non-reproducible
  - you are sharing the ARCHER2 system with other users
  - someone else may also be writing to the same OSTs
  - someone else may be using the same network links
- Caching can give very high IO rates
  - especially with small files
- Ensure benchmarks run for a reasonable time
  - e.g. a few seconds (i.e. large amounts of data)
  - and repeat them several times

Serial IO

- You are not using parallel libraries
  - single file with controller IO (a single writer)
  - file-per-process (many independent writers)
- Little point in striping the file
  - single file: performance bottlenecks appear to be elsewhere
  - multiple files: already parallel as we are using many disks
- This is why a single stripe is the default
- Ballpark figure: single process can achieve about 1 GiB/s
  - in the absence of caching, i.e. writing very large files

MDS performance

- The MDS can become overloaded
  - e.g. opening and closing a file requires MDS access
  - this is therefore a serial operation
    - and you share the MDS with all users on the same filesystem
  - do not do multiple “open/seek/close” operations on the same file

- Tricks
  - try not to write too many files
    - a simple trick is file-per-node rather than file-per-process
    - or ensure only one process writes from each node at the same time

- If you must have lots of files, consider multiple directories
  - e.g. a directory per node
  - decreases lookup times for files

Serialising IO on each node

```
int noderank, nodesize, rankloop;
MPI_Comm nodecomm;
// Create communicator per node so only one process on a node is ever writing to file

MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, MPI_INFO_NULL, &nodecomm);

MPI_Comm_rank(nodecomm, &noderank);
MPI_Comm_size(nodecomm, &nodesize);

for (rankloop = 0; rankloop < nodesize; rankloop++)
  {
    // Do the writes one at a time
    if (rankloop == noderank)
      {
        // Do the IO here
      }
    // Wait your turn
    MPI_Barrier(nodecomm);
  }
MPI_Comm_free(&nodecomm);
```

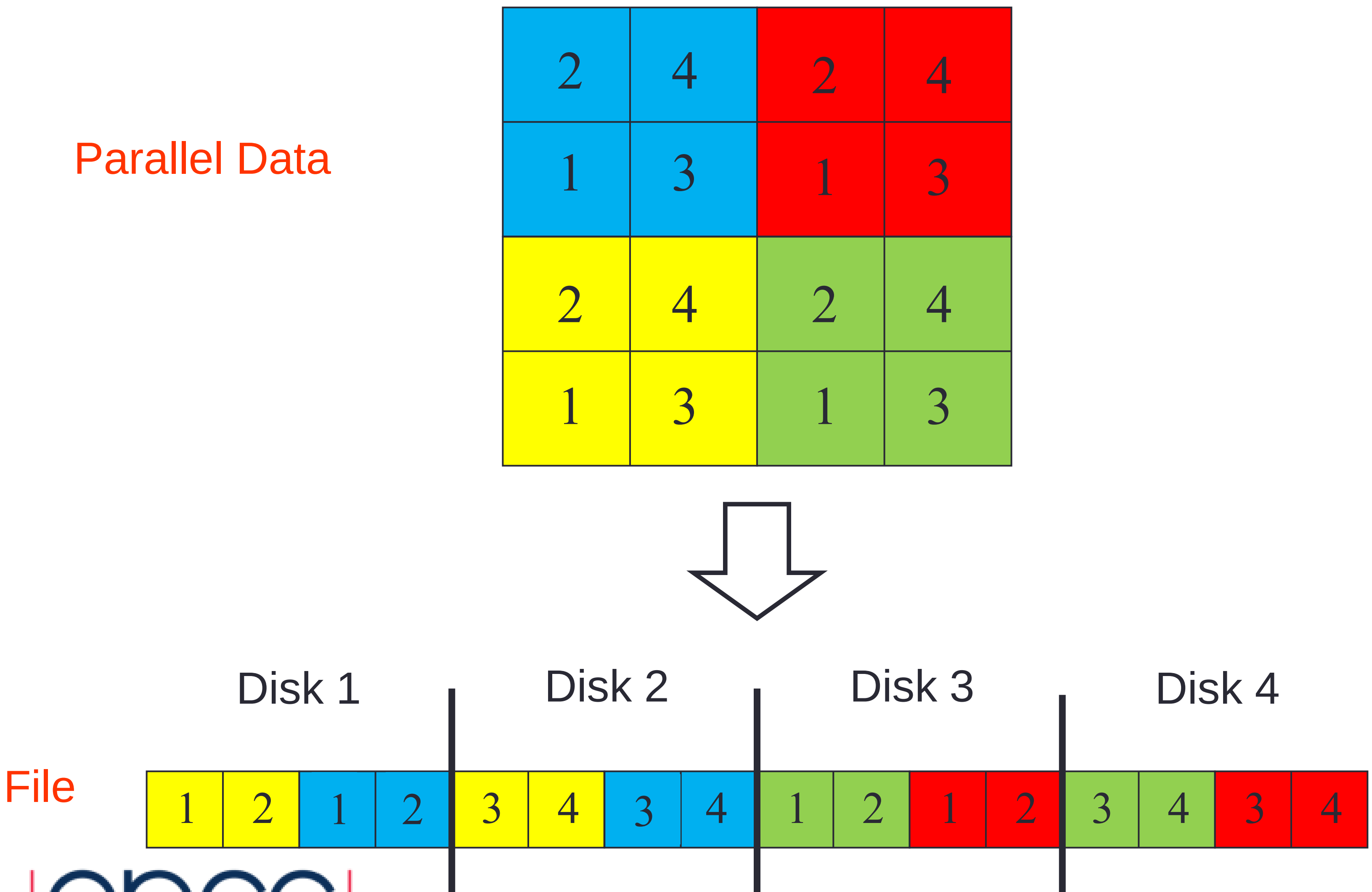
How does MPI-IO work?

- MPI-IO auto-configures to the Lustre settings
- Identifies a small number of aggregator processes
  - spread across nodes if possible
- Uses MPI collectives (e.g. scatter/gather) to aggregate data to these processes
  - then performs a small number of large write operations
  - minimises overheads of locking etc.
- Only possible with collective IO calls
- Can get useful runtime statistics by setting

  ```
  export MPICH_MPIIO_STATS=1
  ```

  - in your SLURM script
- Same applies to HDF5 and NetCDF which use MPI-IO

4x4 array on 2x2 Process Grid



Aggregators

- By default, Cray MPI uses one aggregator per stripe
  - does not seem optimal as single-process IO is slow
  - however it ensures only a single process is ever writing to an OST
    - no issues with file contention: no need to lock
- This default can be changed
  - e.g. to use four times as many aggregators (four per node)

```
export MPICH_MPIIO_HINTS=*:cray_cb_nodes_multiplier=4
```

- This causes multiple processes to write to the same OST
  - default locking approach is very inefficient
  - try using `*:cray_cb_write_lock_mode=2;*:cray_cb_nodes_multiplier=4`

ADIOS2

- A recent IO parallel library https://adios2.readthedocs.io/
  - can output using native MPI-IO or HDF5
  - also supports its own formats, e.g. BP5 (binary-pack v5)

- Same overall approach
  - each process defines what portion(s) of global data it owns
  - call read/write routines

- Much more configurable at runtime via XML file
  - e.g. no need to recompile to switch MPI-IO to BP5

ADIOS2 approach

- Unlike MPI-IO, do not produce exactly the same data on disk in parallel as in serial
  - MPI-IO writes a single shared file with no metadata
  - ADIOS2 writes many files to a directory with associated metadata

- Advantages
  - each aggregator writes its own file so can have many aggregators
  - default is one aggregator per node (not per stripe)
  - much less data rearrangement required

- Disadvantages
  - potential overhead when running on different numbers of nodes?

Results

- MSc in HPC student Petter Sandas ported benchio to C and extended it
  - https://github.com/Petter-Programs/benchio-c

- Also benchmarked IO read (though not in production code)

- Only used the default settings for ADIOS2

New Benchio Results

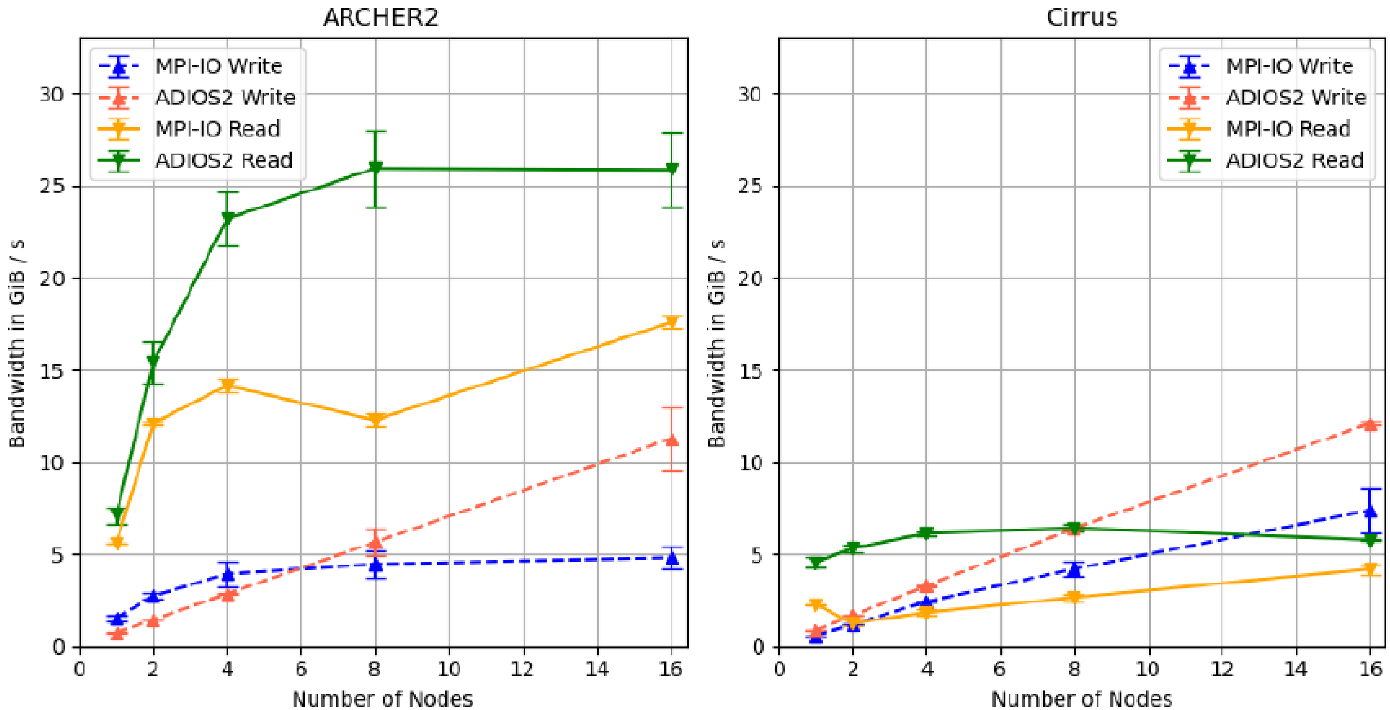


Figure 6.1: Read and write bandwidths on ARCHER2 and Cirrus using ADIOS2 and MPI-IO. Each marker in the graphs represents the maximum of 10 repeat runs and error bars show standard deviation.

From "Parallel IO Benchmarking: Extending the Functionality of benchio", Petter Sandas, MSc in HPC, The University of Edinburgh, 2024

Exercise

- Increase the number of aggregators for ADIOS2
  - how much does this increase performance?
- adios2.xml

```
<?xml version="1.0"?>
<adios-config>
  <io name="Output">
    <engine type="BP5">
      <parameter key="Verbose" value="0"/>
      <!-- Can optionally set the number of aggregators:
      <parameter key="NumAggregators" value="4"/>
      -->
    </engine>
   </io>
</adios-config>
```