I/O Performance Benchmarking and Investigation on Multiple HPC Architectures

B. Lawrencea, C. Maynardb, A. Turnerc, X. Guoc, D. Sloan-Murphyc

aUniversity of Reading, United Kingdom

bMet Office, United Kingdom

cEdinburgh Parallel Computing Centre, University of Edinburgh, United Kingdom

Abstract

I/O performance plays a key role in many scientific simulations and the bottleneck of I/O is an important challenge to solve towards Exascale computing. It is therefore necessary for CoEs and scientific communities with high I/O requirements to understand the usage pattern of existing HPC systems and applications to be suitably equipped to make informed plans for the future.

Theoretical performance numbers assume a clean formatted file system with no contention from other users. Obviously, when used in full production, this level of performance will not be attained.

The goal of this paper is to provide insight into the performance of the file systems in production. To answer questions such as: What is the maximum performance actually experienced? What variation in performance do users experience?

To that end, we detail here the I/O performance of multiple HPC architectures through testing a set of selected I/O benchmarks and representative scientific applications. Results are presented from the following systems:

**ARCHER**: the UK national supercomputing service, using a Cray Sonexion Lustre file system.

**COSMA**: a UK DiRAC resource, using a DDN implementation of the IBM GPFS file system.

**UK-RDF DAC**: the Data Analytic Cluster attached to the UK Research Data Facility, using DDN GPFS.

**JASMIN**: a data analysis cluster delivered by the STFC, using the Panasas file system.

In the Lustre case, a range of stripe counts and sizes are tested. GPFS figures are given under the default configuration as it provides less scope for user tuning.

We run *benchio*, a parallel benchmarking application which writes a three-dimensional distributed dataset to a single shared file. On all systems, we measure MPI-IO performance and, in select cases, compare this with HDF5 and NetCDF equivalent implementations.

We find a reasonable expectation is for approximately 50% of the theoretical system maximum bandwidth to be attainable in practice. Contention is shown to have a dramatic effect on performance. MPI-IO, HDF5 and NetCDF are found to scale similarly but the high level libraries introduce overhead.

Under Lustre, on a single shared file, maximum performance is found by maximising the stripe count, and when I/O operation scale matches the individual stripe size. HDF5 is discovered to scale poorly on Lustre due to an unfavourable interaction with the *H5Fclose()* routine.

# HPC Systems

## ARCHER

ARCHER is a Cray XC30-based system and the current UK National Supercomputing Service. The /work file systems on ARCHER use the Lustre technology in the form of Sonexion parallel file system appliances. The theoretical sustained performance (in terms of bandwidth) of Sonexion Lustre file systems is determined by the number of SSUs (Scalable Storage Units) that make up the file system. ARCHER has four Sonexion file systems:

* fs1: 1 SSU, theoretical sustained = 5 GB/s
* fs2: 6 SSU, theoretical sustained = 30 GB/s
* fs3: 6 SSU, theoretical sustained = 30 GB/s
* fs4: 7 SSU, theoretical sustained = 35 GB/s

## COSMA

The Durham-based Cosmology Machine (COSMA) is one of the five systems making up the UK DiRAC facility. Its disks use the IBM General Parallel File System (GPFS) implemented on two DDN SD12K storage controllers. The theoretical maximum performance is 20 GB/s.

## UK-RDF DAC

The UK Research Data Facility (UK-RDF) is a high volume file storage service collocated with ARCHER. Attached to it is the Data Analytic Cluster (DAC), a system for facilitating the analysis of data held at the RDF. The file system is also a DDN GPFS installation and is based on seven DDN 12K couplets. Separate metadata storage is on NetApp EF550/EF540 arrays populated with SSD drives. Three file systems are available to users:

* gpfs1: 6.4 PB storage, mounted as /nerc
* gpfs2: 4.4 PB storage, mounted as /epsrc
* gpfs3: 1.5 PB storage, mounted as /general

All DAC nodes have direct Infiniband connections to the RDF drives with a maximum theoretical performance of 56 Gbps, or 7 GB/s.

## JASMIN

The Joint Analysis System (JASMIN) is an STFC-delivered service providing computing infrastructure for big data analysis. It uses the Panasas parallel file system implemented via bladesets connected to compute nodes over a 10 Gbps, i.e. 1.25 GB/s, network.

# Parallel I/O benchmark: benchio

The parallel I/O performance of the HPC systems was evaluated by the *benchio* application developed at EPCC. This was chosen ahead of the popular IOR benchmark for a number of reasons:

* The parallel I/O decomposition can be varied to better model actual user applications. IOR uses an extremely simplistic 1D data decomposition (Figure 1) that does not model user codes and does not test the performance of MPI-IO collective operations that are key to real performance. This supported by previous work in *Parallel IO Benchmarking*[1] which found that the optimal MPI-IO write configuration for the IOR layout is to disable collective I/O, a feature essential for achieving speeds beyond that of a few kilobytes-per-second on realistic data layouts.
* The IOR code is very opaque, this makes it very difficult to draw useful conclusions as to what variations in performance are due to.
* benchio is also able to evaluate the performance of HDF5 and NetCDF, two libraries that support parallel I/O and are commonly used by user communities on many HPC services.

The benchio source code is Open Source and is available on GitHub[2].

Figure 1. IOR data layout: simple sequential



The benchio application measures write bandwidth to a single shared file for a given problem size per processor (weak scaling), i.e. the size of the output file scales with the number of processors. We chose to measure write bandwidth as it is the critical consideration of scientific application I/O performance, whereas read performance is traditionally not a factor beyond the initial “one-off” cost of reading input files.

The test data is a series of double precision floating point numbers held in a 3D array and shared over processes in a 3D block decomposition (see Figure 2 and Figure 3). Halos have been added to all dimensions of the local arrays to better approximate the layout of a “real-world” scientific application. By default these local arrays are of size 1283.

Figure 2. benchio data layout: 3D strided, P2 behind P0

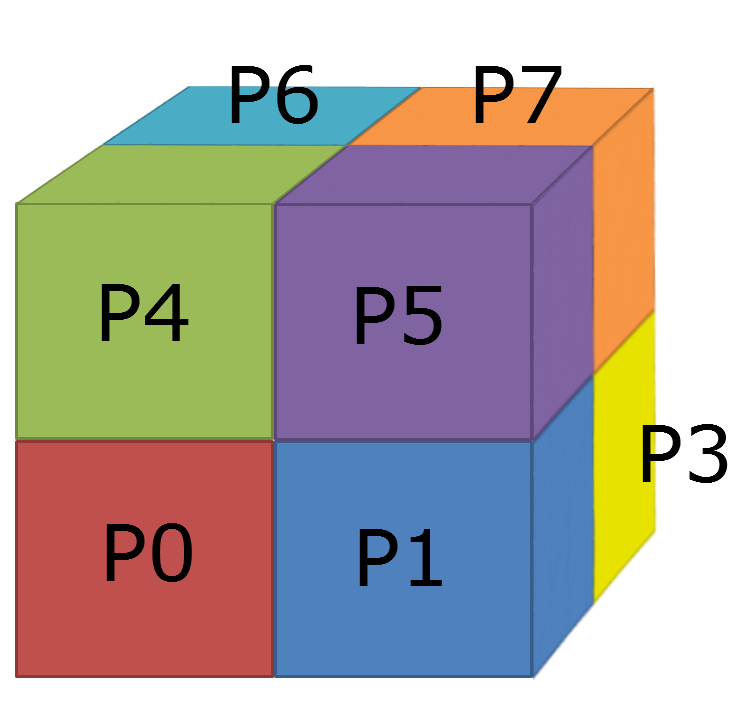
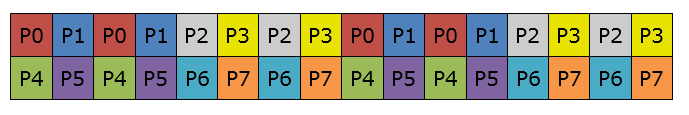


Figure 3. benchio data layout: example 2D decomposition, 2x2x2 grid per processor. Equivalent to layout of output file. Note: data is entirely contiguous and only split into two rows in this figure for legibility; data is not a 2x16 array



# Results

With benchio, each test is repeated a minimum of ten times and the maximum, minimum and average bandwidth reported. As I/O is a shared resource on all measured machines, and therefore subject to contention from other users, the maximum attained bandwidth is considered to be most representative of capabilities of a system. In our initial ARCHER results, we present the full range of values to demonstrate the high variance caused by user contention. However, in the results following, we present only the maximum unless otherwise indicated.

## ARCHER Performance

Using the default Lustre settings on ARCHER:

* Stripe size: 1 MiB
* Number of stripes: 4

we see the performance shown in Figure 4 and listed in Table 1.

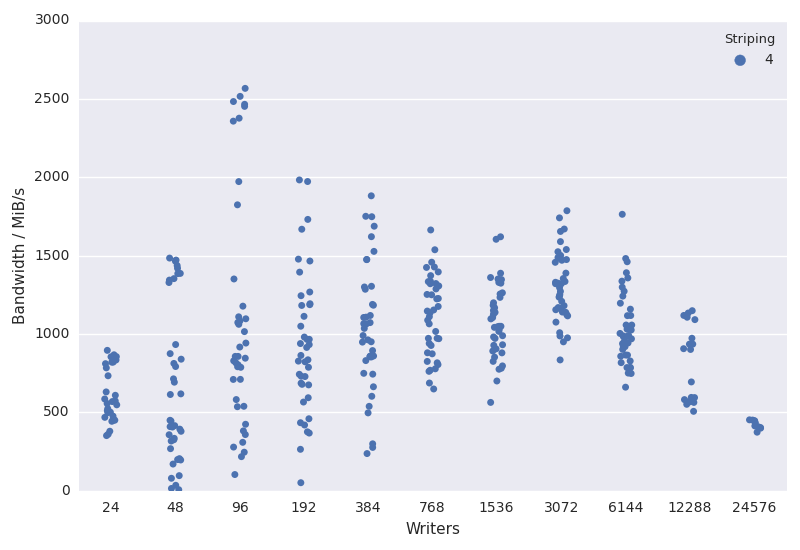


Figure 4. ARCHER MPI-IO default striping (4). A random jitter is applied to the x-axis to better illustrate clusters of similar performance.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **Write Bandwidth (MiB/s)** | | | |  |
| **Writers** | **Total MiB** | **Min.** | **Median** | **Max.** | **Mean** | **Count** |
| 24 | 384 | 352.327 | 563.193 | 896.015 | 607.84 | 30 |
| 48 | 768 | 7.326 | 447.742 | 1484.661 | 662.083 | 40 |
| 96 | 1536 | 103.956 | 857.516 | 2567.143 | 1096.012 | 40 |
| 192 | 3072 | 51.817 | 889.175 | 1982.988 | 938.818 | 40 |
| 384 | 6144 | 237.673 | 1048.918 | 1881.732 | 1042.079 | 40 |
| 768 | 12288 | 649.679 | 1141.212 | 1663.967 | 1116.774 | 40 |
| 1536 | 24576 | 564.05 | 1049.461 | 1620.391 | 1081.459 | 40 |
| 3072 | 49152 | 835.001 | 1308.683 | 1786.612 | 1307.008 | 40 |
| 6144 | 98304 | 661.195 | 985.927 | 1763.888 | 1040.65 | 40 |
| 12288 | 196608 | 507.069 | 798.218 | 1149.057 | 802.795 | 20 |
| 24576 | 393216 | 374.109 | 423.413 | 452.824 | 422.868 | 10 |

Table 1. ARCHER MPI-IO default striping (4) raw data.

Using the default stripe settings on ARCHER, the maximum write performance that can be achieved is just over 2,500 MiB/s, well short of the theoretical sustained performance of 30,000 MiB/s.

In the worst case, 48 writers give a speed of approximately 7 MiB/s, more than a factor of 200 slower than the maximum performance of near 1,500 MiB in that instance.

#### Lustre Tuning

As described in Parallel I/O Performance on ARCHER[3], to get the best parallel write performance for a single-shared file case we must use as many stripes as possible. This is achieved on Lustre by setting the striping to “-1” which stripes over all available OSTs. We repeated the benchmarks with:

* Stripe size: 1 MiB
* Number of stripes: -1 (corresponds to 48 on fs3)

The performance for this configuration is shown in Figure 5 and Table 2.

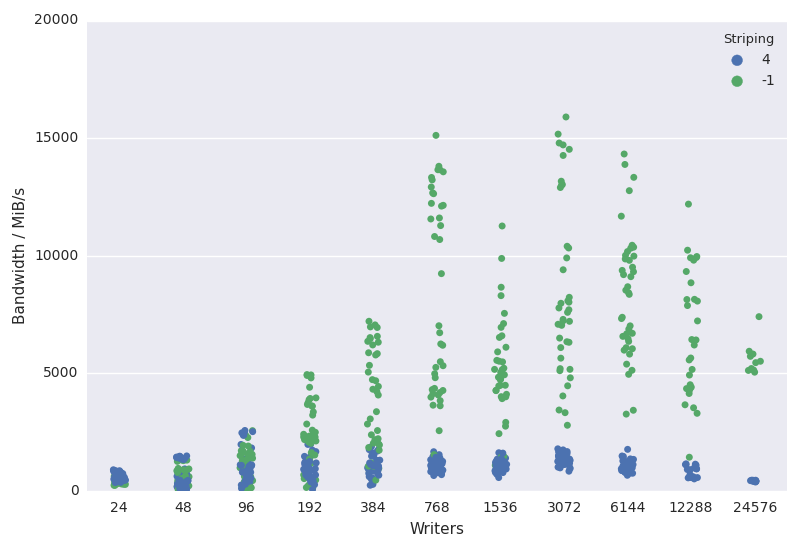


Figure 5. ARCHER MPI-IO maximum striping (-1). Default striping of 4 is plotted for comparison.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **Write Bandwidth (MiB/s)** | | | |  |
| **Writers** | **Total MiB** | **Min.** | **Median** | **Max.** | **Mean** | **Count** |
| 24 | 384 | 233.721 | 395.648 | 615.717 | 431.975 | 30 |
| 48 | 768 | 23.589 | 580.701 | 1355.754 | 694.411 | 40 |
| 96 | 1536 | 92.665 | 1289.047 | 2559.369 | 1232.707 | 40 |
| 192 | 3072 | 123.165 | 2316.967 | 4943.626 | 2546.546 | 40 |
| 384 | 6144 | 455.328 | 4145.48 | 7210.351 | 3890.188 | 40 |
| 768 | 12288 | 1540.754 | 6871.673 | 15115.907 | 8318.147 | 40 |
| 1536 | 24576 | 919.277 | 4883.091 | 11262.025 | 5050.061 | 40 |
| 3072 | 49152 | 2788.93 | 7644.637 | 15897.907 | 8547.329 | 40 |
| 6144 | 98304 | 3262.925 | 8476.518 | 14323.187 | 8371.443 | 40 |
| 12288 | 196608 | 1429.39 | 6308.175 | 12192.03 | 6597.808 | 30 |
| 24576 | 393216 | 5045.687 | 5480.116 | 7406.748 | 5633.765 | 10 |

Table 2. ARCHER MPI-IO maximum striping (-1) raw data.

When using the maximum number of stripes, we see much improved performance (compared to the default stripe count of 4) with a maximum write bandwidth of slightly under 16,000 MiB/s with 3072 cores (128 nodes) writing simultaneously. This is still not much more than 50% of the advertised sustained bandwidth of 30,000 GiB/s for this file system.

The experiments were then repeated, adjusting the size of each Lustre stripe:

* Stripe sizes: 4 MiB and 8 MiB
* Number of stripes: -1 and 4

Maximum measured performance is given in Figure 6 and Figure 7 with the data from the default 1 MiB configuration plotted for comparison.

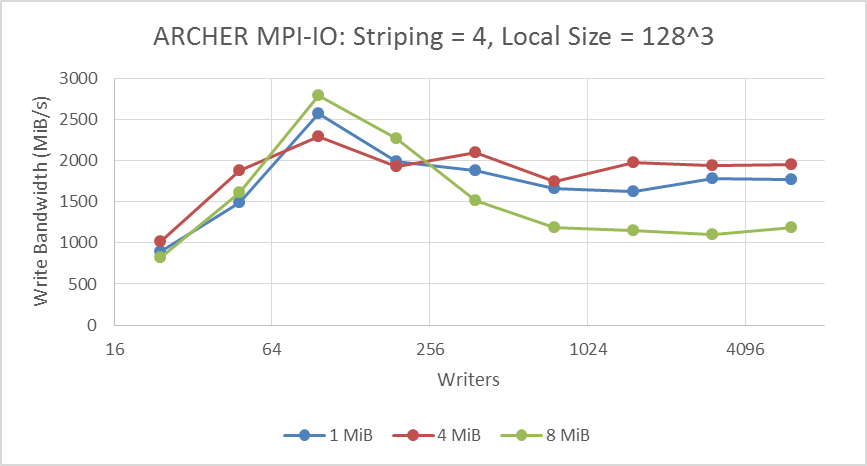


Figure 6. ARCHER stripe size performance, default stripe count

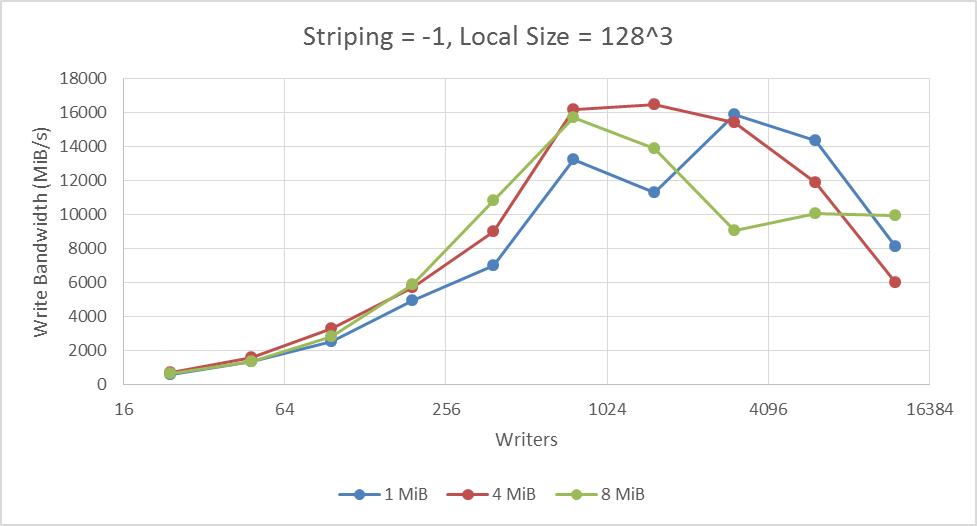


Figure 7. ARCHER stripe size performance, maximum stripe count

Stripe size was found to have a limited effect on the write performance, with the peak for all three sizes being approximately 16,000 MiB/s as before and the measured differences being in-line with the expected variance caused by file system contention. However, the largest 8 MiB setting is shown to be detrimental as core counts increase beyond this performance peak. This is attributed to increased file locking times and OST contention.

#### Data Size

All prior experiments were performed with the default local data array of 1283 double precision values/16 MiB of data per process. We expected that the benefits of larger stripe sizes would be made apparent with greater volumes of data so repeated the above tests with an increased array size of 2563 values/128 MiB per process. Results are given in Figure 8 and Figure 9.

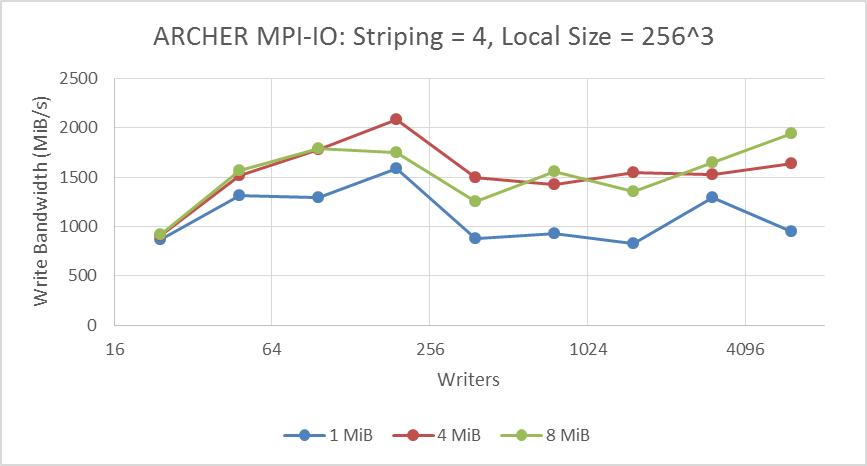


Figure 8. ARCHER large local arrays bandwidth, default stripe count

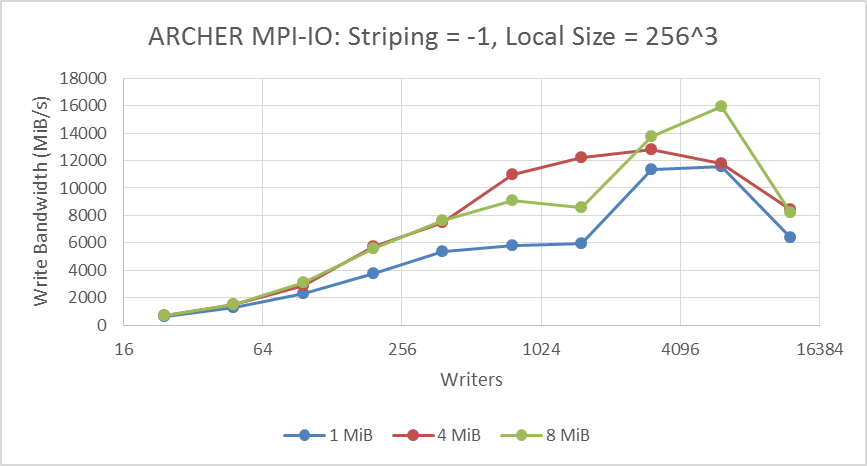


Figure 9. ARCHER large local arrays bandwidth, maximum stripe count

The larger stripe sizes give consistently better performance at both 4 and -1 stripe counts. Indeed 8 MiB at 6144 cores is the only configuration to achieve the apparent 16,000 MiB/s limit on ARCHER I/O while the default 1 MiB reaches less than 12,000 MiB/s.

It is apparent that stripe size configuration must be considered in conjunction with I/O operation size to attain maximum performance. In general they must match; lower volume operations should be given smaller stripe sizes, while larger operations require larger stripes.

#### NetCDF Performance

Optimised installations of NetCDF, backed by parallel HDF5, are provided by Cray as part of the operating system on ARCHER. At time of writing, the default version of this cray-netcdf-hdf5parallel module is 4.3.3.1. However, it was found to give poor performance, failing to demonstrate scalability and instead reaching a peak bandwidth of approximately 1 GiB/s regardless of number of writers or Lustre configuration. We therefore used the more recent NetCDF version 4.4.0, which scales as expected, for all benchmarks and would recommend 4.3.3.1 and below be avoided by users for performance reasons.

Results for version 4.4.0, repeating the stripe and array size experiments performed for MPI-IO, are plotted in Figure 10 to Figure 13.

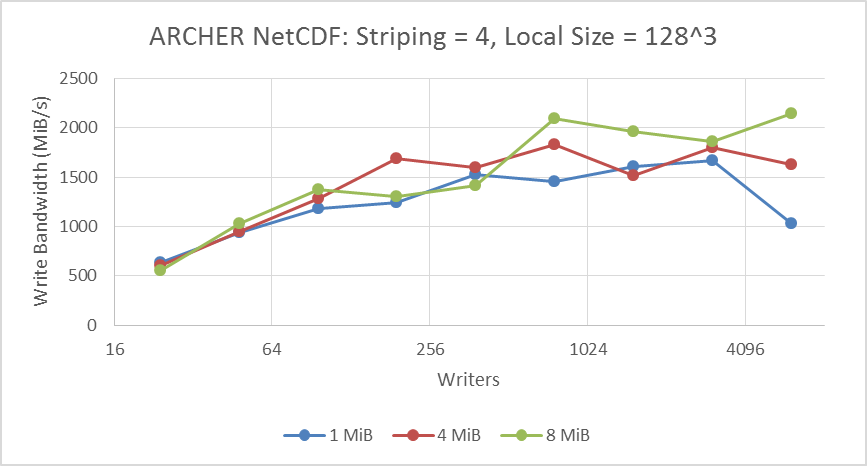


Figure 10. ARCHER NetCDF v4.4.0 performance, default striping, default array sizes

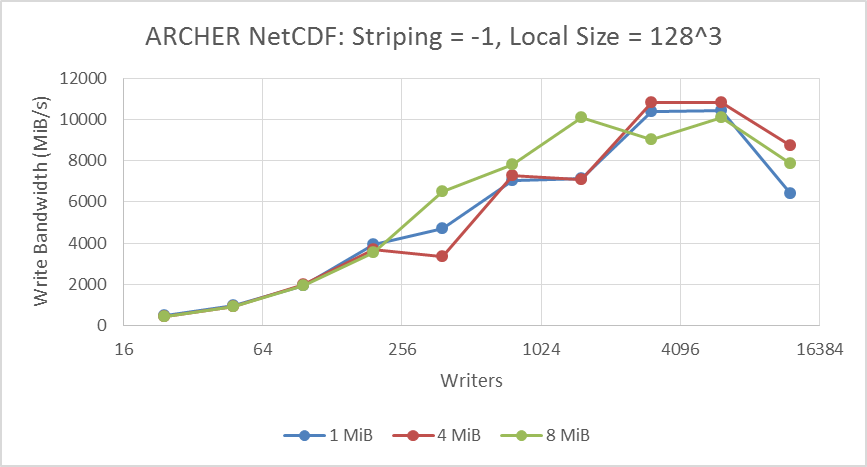


Figure 11. ARCHER NetCDF v4.4.0 performance, maximum striping, default array sizes

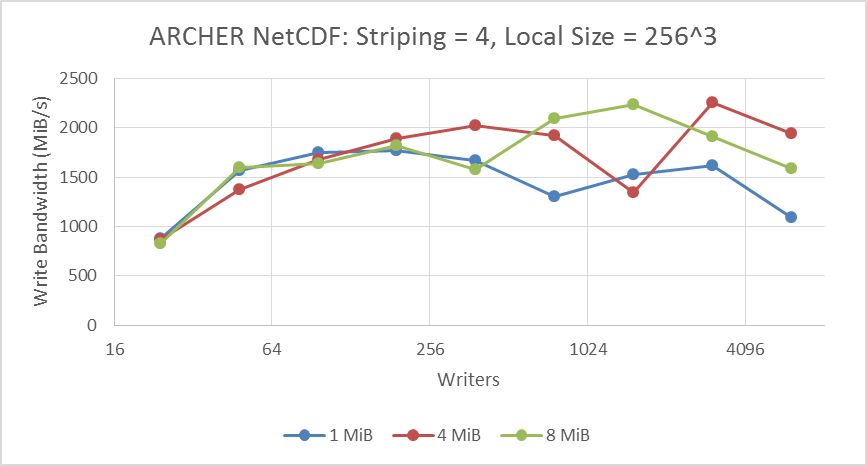


Figure 12. ARCHER NetCDF v4.4.0 performance, default striping, large arrays

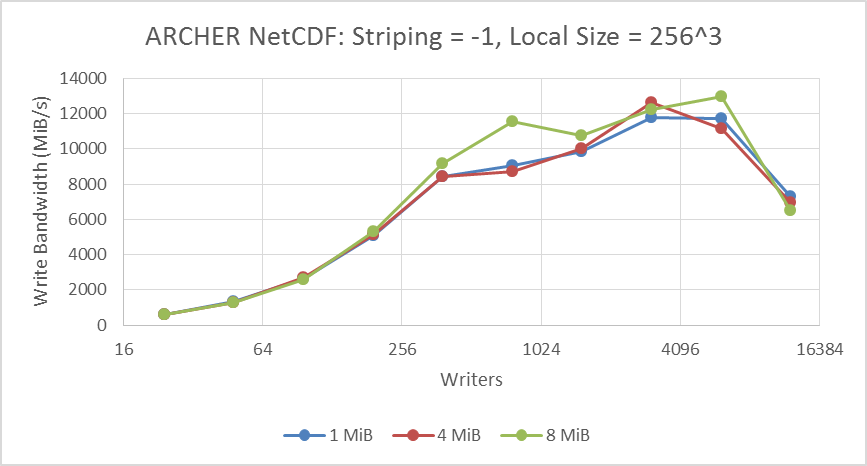


Figure 13. ARCHER NetCDF v4.4.0 performance, maximum striping, large arrays

NetCDF performance characteristics were found to be entirely similar to MPI-IO, with variations in stripe count, stripe size and local array size producing the same general trend. This is in line with expectations as NetCDF interfaces to HDF5 for its parallel implementation, which is itself based on MPI-IO.

Peak bandwidth was measured at 13,000 MiB/s, down from the 16,000 MiB/s seen with MPI-IO. This is attributed to the overhead of the NetCDF/HDF5/MPI-IO stack and the additional structuring applied to NetCDF files.

#### HDF5 Performance

As with NetCDF, Cray provide the HDF5 parallel libraries on ARCHER. Similar performance limitations to NetCDF 4.3.3.1 were observed with HDF5 tests however they persisted with all system-installed versions of the library, from the default 1.8.14 to the most current 1.10.0. Given the hierarchical nature of the libraries, we theorise that the NetCDF 4.3.3.1 limitations are in reality a manifestation of this HDF5 bug, and that NetCDF 4.4.0 circumvents the issue by following an alternate code path around the problematic library calls.

Application profiling of benchio with the HDF5 backend found the majority of compute time is spent in function *MPI\_File\_set\_size()*, called within the HDF5 library from the user-level *H5Fclose()* routine. Discussions with Cray revealed this to be a known bug specific to the combination of HDF5 with Lustre file systems.

An *MPI\_File\_set\_size()* operation, on a Linux platform like ARCHER, eventually calls the POSIX function: *ftruncate()*. This has an unfavourable interaction with the locking for the series of metadata communications the HDF5 library makes during a file close. In practice, this leads to relatively long close times of tens of seconds and hence the lack of scalability observed.

The HDF5 developers have noted this behaviour in the past where it manifested in H5Fflush(), the function for flushing write buffers associated with a file to disk: “when operating in a parallel application, this operation resulted in a call to MPI\_File\_set\_size, which currently has very poor performance characteristics on Lustre file systems. Because an HDF5 file’s size is not required to be accurately set until the file is closed, this operation was removed from H5Fflush and added to the code for closing a file”[4] hence leading to the behaviour currently observed in *H5Fclose().*

Cray discussions on this bug are on-going and, at present, no known work-around or mitigation is provided for end users. The recommendation for CoEs is to be aware of this interaction and inform scientific communities as the issue is observed.

#### Impact of System Load

To better understand the impact of file system contention, we simulated degrees of load by running multiple instances of the benchio MPI-IO test in parallel. Figure 14 shows the aggregate mean performance of one, two and four benchio instances writing concurrently to independent files.

Note that mean performance is presented in this instance as, on occasion, test timing was such that a single benchio instance would be performing I/O while the others were in a setup phase or otherwise stalled. The maximum bandwidth is therefore equivalent to the single instance case and not a representative value for this test.

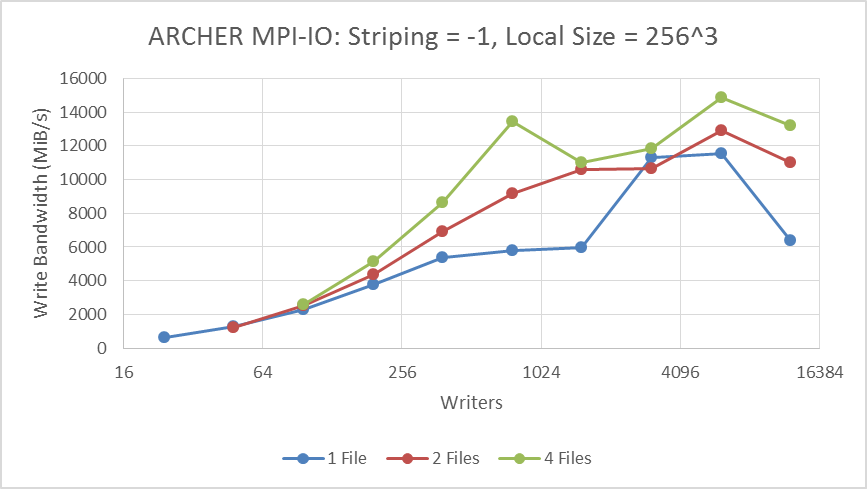


Figure 14. Effect of I/O load on ARCHER

From the similarities in the data trends, we see that bandwidth is on average divided equally between writers. e.g. the aggregate bandwidth of two benchio instances, each with 24 writers putting data to independent files, is roughly equivalent to the bandwidth of a single instance with 48 writers.

## COSMA Performance

The GPFS file system employed by the DiRAC COSMA service does not facilitate user tuning like Lustre. GPFS settings are fixed at installation and cannot be adjusted at run time. We therefore ran a single set of benchmarks to determine the peak bandwidth of the system, presented in Figure 15.

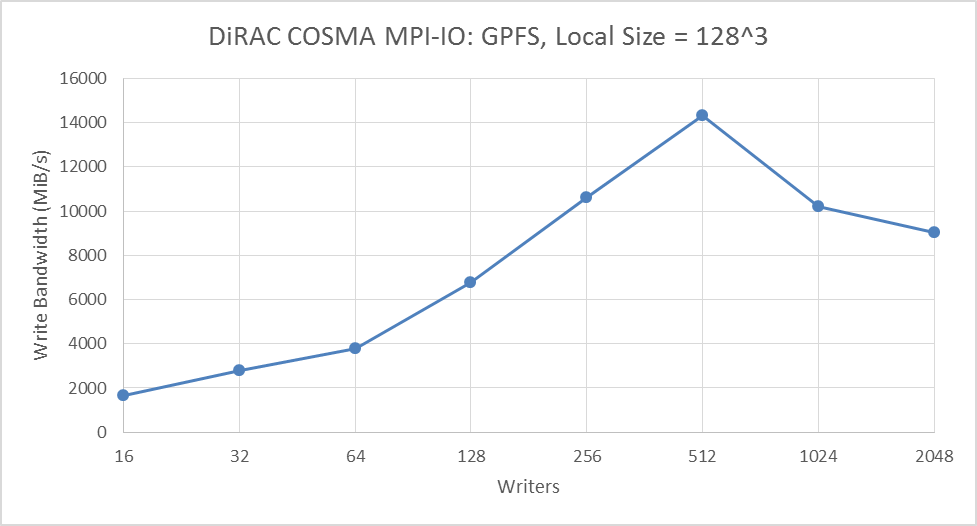


Figure 15. MPI-IO bandwidth for DiRAC COSMA

Best performance is seen at 512 writers, which attain marginally more than 14000 MiB/s or approximately 68% of the rated maximum, before parallel efficiency drops.

## UK-RDF DAC Performance

The UK-RDF DAC supports only shared memory parallelism; jobs cannot span nodes. All tests were therefore run on a single, standard compute node offering 40 CPU cores.

We benchmarked two of the three GPFS file systems and examined the performance of each of the benchio parallel backends. Comparisons are given in Figure 16 and Figure 17.

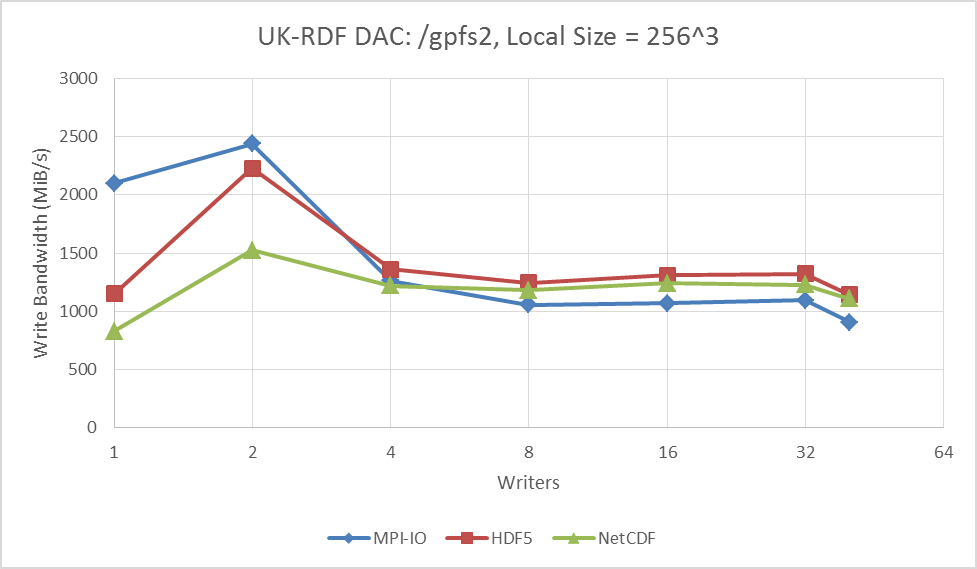


Figure 16. All backends bandwidth for UK-RDF DAC. File system: 4.4PB /gpfs2 mounted as /epsrc.

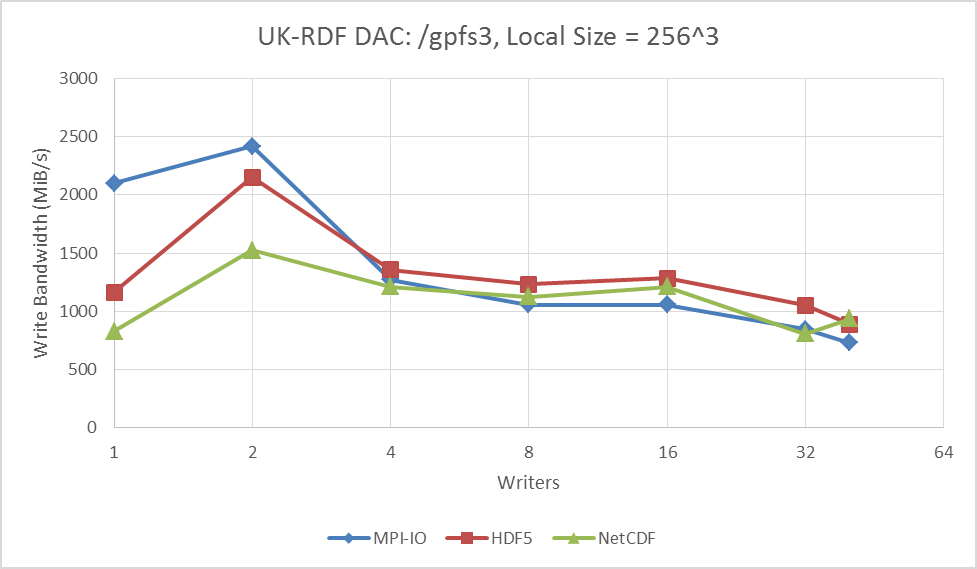


Figure 17. All backends bandwidth for UK-RDF DAC. File system: 1.5 PB /gpfs3 mounted as /general.

No difference in performance was measured between the /gpfs2 and /gpfs3 file systems. Both achieved the same peak performance of approximately 2500 MiB/s, or less than 40% of the theoretical maximum of 7 GB/s. Hence file system storage capacity was found to have no bearing on overall write speed in this instance.

Scope for parallelisation is limited on this system with performance dropping significantly at 4 writers and above. Previous work in *Investigating Read Performance of Python and NetCDF when using HPC Parallel Filesystems*[5] on the RDF DAC supports these findings, showing sequential serial read performance to peak at roughly 1400 MiB/s, i.e. the same performance level seen from 4 to 40 cores in Figure 16 and Figure 17. Further work is needed to precisely identify the bottleneck limiting this scalability.

## JASMIN Performance

As with the RDF DAC, JASMIN tests were performed on a single compute node. Results were gathered from 1 to 32 writers and are presented in Figure 18.

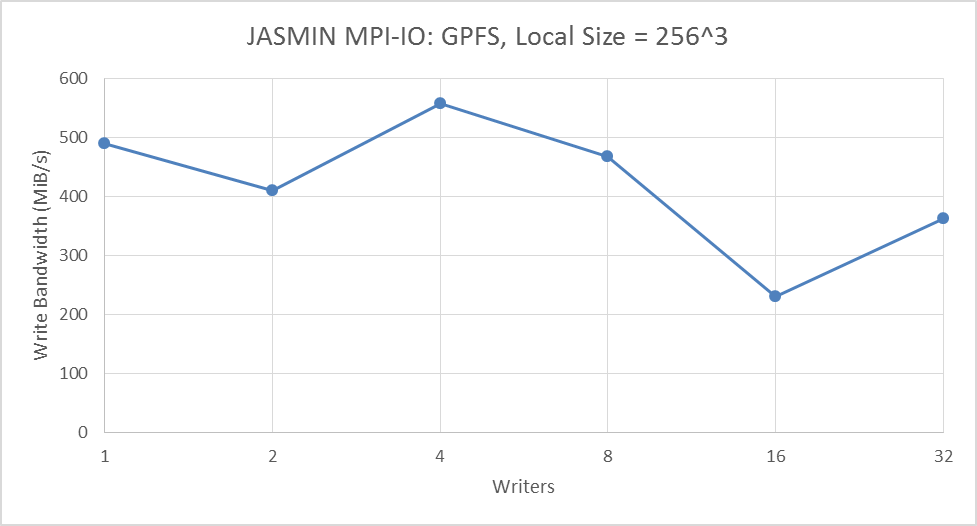


Figure 18. MPI-IO bandwidth for JASMIN

With further reference to *Investigating Read Performance of Python and NetCDF when using HPC Parallel Filesystems* [5], sequential serial performance on JASMIN has been measured at approximately 500 MiB/s, the same level of performance observed in these parallel I/O tests. From this, we conclude that there is no scope for improvement with parallelisation on this system under the default configuration. However, at time of writing, additional work is underway from Jones et al. to expand their investigation to include multi-threaded performance and examine parallelism on JASMIN in greater detail. Results are expected to be published at a later date.

## Comparative System Performance

Figure 19 gives an overview of all four benchmark systems and compares their overall performance.

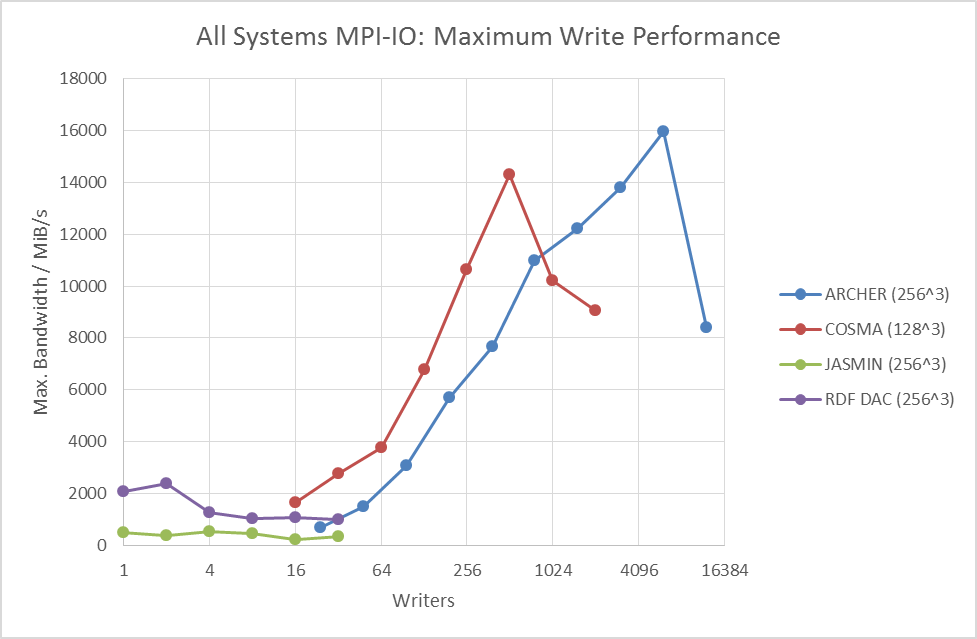


Figure 19. Comparison of maximum write performance between benchmark systems

The two systems intended for high-performance parallel simulations, ARCHER and COSMA, are broadly comparable, as are the two data analysis systems. The scope for parallelism is simply lower on JASMIN and the RDF DAC and users should not expect compute and analysis platforms to have similar performance.

# Conclusions

Our findings can be summarised as follows:

* Approximately 50% of the theoretical maximum on a system is expected
* Variance in I/O performance due to user contention is dramatic – factor of 200 difference in worst case
* Under load, write bandwidth is on average divided equally between writers
* MPI-IO, HDF5 and NetCDF share the same performance characteristics but the higher level libraries introduce overhead
* Systems designed for parallel simulations offer much higher performance than data analysis platforms
* Specific to Lustre file systems:
  + Maximum striping should be used for a single shared output file
  + I/O operation and stripe sizes must be in accordance
  + Users should be aware of the HDF5 performance issue
  + Versions of NetCDF below 4.4.0 should be avoided as they affected by the HDF5 issue
* With GPFS, file system capacity was found to have no bearing on performance

References

1. Jia-Ying Wu, Parallel IO Benchmarking, <https://static.ph.ed.ac.uk/dissertations/hpc-msc/2015-2016/Jia-ying_Wu-MSc-dissertation-Parallel_IO_Benchmarking.pdf>, retrieved 22 Nov 2016
2. EPCCed/benchio: EPCC I/O benchmarking applications, <https://github.com/EPCCed/benchio>, retrieved 01 Nov 2016
3. David Henty, Adrian Jackson, Charles Moulinec, Vendel Szeremi: Performance of Parallel IO on ARCHER Version 1.1, <http://www.archer.ac.uk/documentation/white-papers/parallelIO/ARCHER_wp_parallelIO.pdf>, retrieved 01 Nov 2016
4. Mark Howison, Quincey Koziol, David Knaak, John Mainzer, John Shalf: Tuning HDF5 for Lustre File Systems, https://support.hdfgroup.org/pubs/papers/howison\_hdf5\_lustre\_iasds2010.pdf , retrieved 03 Nov 2016
5. Matthew Jones, Jon Blower, Bryan Lawrence, Annette Osprey: Investigating Read Performance of Python and NetCDF When Using HPC Parallel Filesystems, http://link.springer.com/chapter/10.1007%2F978-3-319-46079-6\_12, retrieved 24 Nov 2016

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EU’s Horizon 2020 research and innovation programme (2014-2020) under grant agreement 653838.

The authors would like to thank Harvey Richardson of Cray Inc. for his invaluable advice on the ARCHER file systems and software.