MINIO: An I/O Benchmark for Investigating High Level Parallel Libraries

James Dickson[1], Satheesh Maheswaran[2], Steven Wright[1], Andy Herdman[2], and Stephen Jarvis[1]

[1]Department of Computer Science, University of Warwick, UK
[2]High Performance Computing, UK Atomic Weapons Establishment, Aldermaston, UK

Introduction

Input/output (I/O) operations are amongst the biggest challenges facing scientific computing as it transitions to exascale. The traditional software stack – comprising of parallel file systems, middlewares and high level libraries – has evolved to enable applications to better cope with the demands of enormous datasets. This software stack makes high performance parallel I/O easily accessible to application engineers, however it is important to ensure best performance is not compromised through attempts to enrich these libraries. We present MINIO, a benchmark for the investigation of I/O behaviour focusing on understanding overheads and inefficiencies in high level library usage. MINIO uses HDF5 and TyphonIO, a library build on HDF5 with its own scientific data model, to explore I/O at scale using different application behavioural patterns. A case study is performed using MINIO to identify performance limiting characteristics present in the TyphonIO library as an example of performance discrepancies in the I/O stack.

Traditional Software Stack

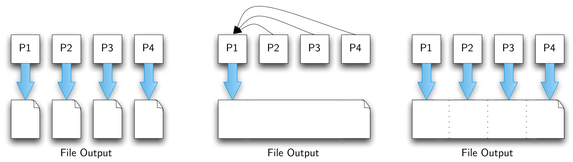
Application	
Data Model Libraries	**TyphonIO**, SILO, LibMesh
High Level Library	**HDF5**, PnetCDF, Adios
Middleware	MPI-IO
Parallel File System	Lustre, GPFS, HDFS
Storage Hardware	

Parallel I/O Paradigms

(a) The file-per-process approach achieves parallelism by simultaneously accessing multiple files. Often the number of simultaneous accesses must be limited to avoid overloading metadata systems. Distribution of data is handled by the application and is not easily portable between runs of different sizes.

(b) Gather to root aggregates data at a single process and writes to a single file. The greater workload is placed on the interprocess communication rather than the data storage subsystem, however there is no opportunity for any real I/O parallelism.

(c) Shared file parallel I/O uses the above software stack, which handles the distribution of data on behalf of the application. The management of contention and communication in shared file I/O however becomes an important focus for achieving best performance.

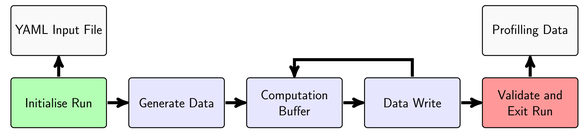


(a) File-Per-Process – N-N (b) Root Gather – N-Root (c) Shared File – N-1

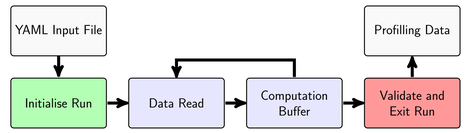
Design

- Our benchmark is partly derived from a closed source application called IOBench, which was designed for exercising file systems during system procurement. IOBench is limited in the flexibility of its execution pattern and only performs operations using the TyphonIO data model library, which in turn uses HDF5.
- A feature of our application is the computational buffer introduced between consecutive read or write operations. This buffer serves to produce a greater similarity to real applications, which consist of numerical computations between collections of I/O operations. The way computation is introduced to execution is via the input's *compute level* and *compute length*, which take an integer value of 1 to 3 and 1 to 5 respectively. Increasing the compute level value increases the complexity of operations to perform, while the length parameter influences the duration each buffer will take.

Application Characteristics



(a) Write Sequence



(b) Read Sequence

Parametrisation

- Control of the application parameters is handled by a self describing input file written in the YAML data interchange format.
- Structure of the dataset mimics the data model used by TyphonIO

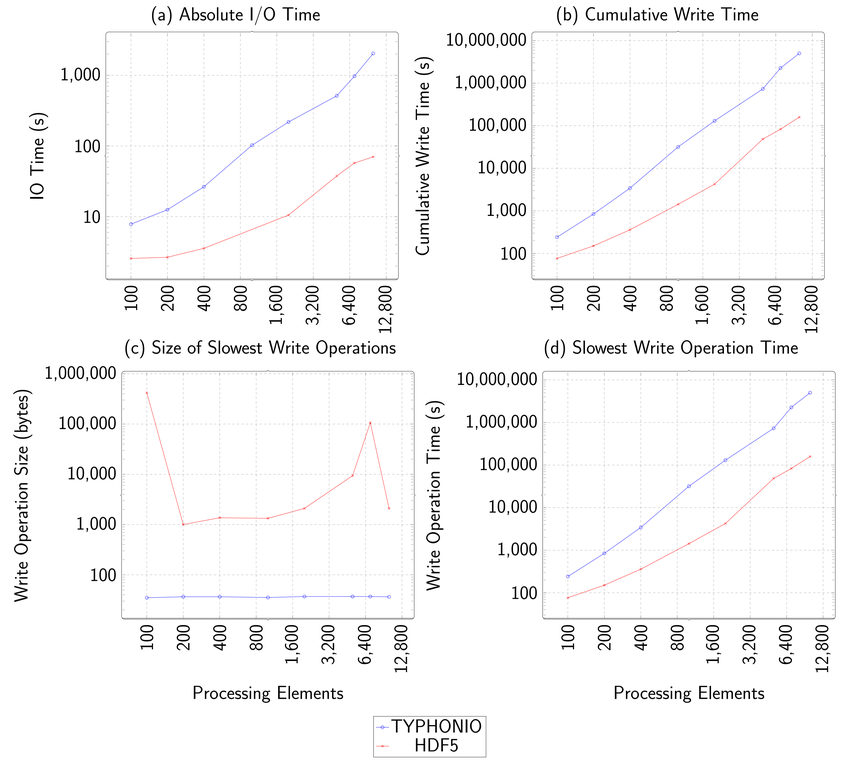
Parameter	Explanation
io_method	The method used for parallel I/O operations – either TyphonIO or HDF5
mesh	The structure of the mesh that will be written/read – quad, unstructured or point mesh
processors	Processes performing I/O operations
real	Real elements in the target mesh per process
mixed	Mixed material elements in the target mesh per process
ghost	Ghost elements in the target mesh per process (ghost layers for quad mesh types)
quantities	Data quantities (any mesh wide data other than materials)
variables	Data variables (additional data not attached to other objects or their metadata)
io_mode	I/O behaviour to exercise exercised – either write or read
steps	Number of iterations of the desired I/O behaviour
compute_level	Intensity of numerical computation performed between I/O kernel execution
compute_length	Duration of compute buffer

Case Study

Experimental Setup

	Archer
Processor	Intel Xeon E5-2697
CPU Speed	2.7 Ghz
Cores per node	24
Nodes	4920
Interconnect	Aries Dragonfly
File System	Lustre
Storage size	1.5PB

- A performance study was conducted using MINIO to identify how writing identical datasets differed between TyphonIO and its underlying library implementation HDF5.
- Weak scaling runs were performed from 100 to 10,000 processing elements using the ARCHER supercomputing platform.



Evaluation

- The results demonstrate that absolute I/O time for a run is much greater for TyphonIO than when performing the equivalent operations using HDF5 directly. As process count scales, dedicated I/O for the simulation increases only marginally for HDF5 operations. The HDF5 calls made by TyphonIO are very different and result in I/O time scaling linearly with process count. The trend of increasing write times is seen across all processes, resulting in a much greater cumulative time spent performing data writes for TyphonIO calls compared to HDF5 when scaling past 2,000 processes.
- Examination of the slowest write operations show that TyphonIO's slowest transfer size is consistently around 36 bytes. The slowest data transfers made by HDF5 are never less that 27× larger, with runs at 100 and 7,000 processes displaying sizes of 419 Kbytes and 104 Kbytes respectively. These values suggest that TyphonIO performs many smaller, less efficient write operations. Additionally, timing data also shows the slowest operation time is again consistently lower for HDF5, with the transfer in question taking 4 times longer at 100 processes. This operation upper bound increases at a rate slightly above that at which process count scales for TyphonIO, however the rate is decreasing for HDF5. Consequently, at 10,000 processes there is a 30× time difference with TyphonIO's slowest transfer taking 18 minutes compared to HDF5's 35 seconds.

Related Work

- IOR [1] is a benchmark derived from workload analysis of applications used at NERSC. This work has been successful in improving the representativeness of access patterns displayed by many benchmarks, a trait we also wish to incorporate into the design of MINIO.
- FLASH-IO [2], MADBench2 [3], Chombo I/O and S3D-IO derive I/O kernels from complete applications. This approach aims to bridge the gap between a standalone benchmark and the applications they attempt to model. These are useful for providing insight into I/O behaviours of single applications, but they are inevitably less useful for wider investigation.
- Skel [4] and APPrime [5] automatically generate I/O kernels based on input data and traces taken from target applications. While this approach can match real world applications, generating the required data may not always be feasible. Running full scale codes to produce I/O traces uses valuable computational resources and can often take a great deal of time given the complexity of many scientific applications.

References

[1] Shan, H., Antypas, K., Shalf, J.: Characterizing and Predicting the I/O Performance of HPC Applications using a Parameterized Synthetic Benchmark. In: Proceedings of the 20th ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'08), IEEE Press, (2008)

[2] FLASH-IO Benchmark on NERSC Platforms, `http://pdsi.nersc.gov/IOBenchmark/FLASH_IOBenchmark.pdf`

[3] Borrill, J., Oliker, L., Shalf, J., Shan, H.:Investigation of leading HPC I/O performance using a scientific-application derived benchmark. In: Proceedings of the 19th ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'07), IEEE Press, (2007)

[4] Logan, J., Klasky, S., Abbasi, H., Liu, Q., Ostrouchov, G., Parashar, M., Podhorszki, N., Tian, Y., Wolf, M.: Understanding I/O Performance Using I/O Skeletal Applications. In: Proceedings of the 18th International Conference on Parallel Processing (Euro-Par 2012), Springer Berlin, Heidelberg, (2012)

[5] Jin, Y., Liu, M., Ma, X., Liu, Q., Logan, J., Podhorszki, N., Choi, J.Y., Klasky, S.: Combining Phase Identification and Statistic Modelling for Automated Parallel Benchmark Generation. In: Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2015), ACM, New York, NY, USA, (2015)