Performance Analysis of Filesystem I/O using HDF5 and ADIOS on a Cray XC30

Ruonan Wang, Andreas Wicenec
ICRAR
The University of Western Australia
Perth, Australia
Email: jason.wang@icrar.org, andreas.wicenec@icrar.org

Christopher Harris
iVEC
Perth, Australia
Email: chris.harris@ivec.org

***Abstract*—The Square Kilometer Array telescope will be one of the worlds largest scientific instruments, and will provide an unprecedented view of the radio universe. However, to achieve its goals the Square Kilometer Array telescope will need to process massive amounts of data through a number of signal and imaging processing stages. For example, for the correlation stage the SKA-Low Phase 1 will produce terabytes of data per second and significantly more for the second phase. The use of shares filesystems, such as Lustre, between these stages provides the potential to simplify these workflows. This paper investigates writing correlator output to the Lustre filesystem of a Cray XC30 using the HDF5 and ADIOS high performance I/O APIs. The results compare the performance of the two APIs, and identify key parameter optimisations for the application, APIs and the Lustre configuration.**

***Keywords*-ADIOS; HDF5; Lustre; Radio astronomy;**

I. Introduction

Modern radio astronomy is becoming one of the most challenging big data applications. Being one of the largest scientific instruments in the world, the Square Kilometer Array (SKA) will produce terabytes of visibility data per second, for Phase 1 of the low frequency component alone. The amount of data throughput required will potentially grow by orders of magnitude by the time the full SKA is operational, and will potentially be the limiting factor of the entire system.

While the signal processing work flow was traditionally implemented on dedicated hardware, a recent trend is to process the data using general purpose supercomputers or clusters. This provides considerably better flexibility to the work flow design as improvements can be made by upgrading software. One advantage of this flexibility is that various data models can be applied in a single system and switched dynamically. For instance, visibilities as intermediate data are often discarded after the posterior processing stage is accomplished, due to the limitation of the storage system. In a traditional hardware based system, once it is designed and fixed in this way, then there is no low-cost workarounds to obtain the visibility data. With the flexibility of a software system, this can be solved by simply introducing another module or plug-in into the pipeline software, and thus easily enables certain science cases that require such non-standard data input.

This paper investigates writing visibility data to the Lustre filesystem using HDF5 and ADIOS I/O APIs. Testing is across a number of schemes and configurations, varying the number of compute nodes, the Lustre stripe size, the number of input data streams, the number of frequency channels and the number of time slices per file. To illustrate the scalability of the Lustre filesystem, results from local storage machines are also given for reference. By analyzing these results, the key parameter optimizations are identified, in terms of the application, I/O APIs and the Lustre configuration.

In the following section, we will first introduce some background knowledge on signal correlation, Lustre filesystem, HDF5 and ADIOS. The data pattern and implementation strategy we used in this work is then described in the Method section. This is followed by an introduction to our hardware and software testing environment, the details on the testing parameters, and the testing results. Finally, in the Discussion section, we will identify the optimal parameter range and discuss issues we noticed during the work.

II. Background

This section will give a brief introduction to radio astronomy signal correlation, Lustre, HDF5, and Adios, to outline these technologies and define terms used in subsequent sections.

A. Radio Astronomy Signal Processing

The raw signal data from the receivers in a radio interferometer is converted to images of the radio sky via a signal and imaging processing pipeline. This process is referred to as aperture synthesis. This is a highly involved process, with a large number of customized processing stages specific to a particular telescope array. For the purposes of this section, this is abstracted into a number of consecutive stages.

After any initial preprocessing, signals undergo correlation, in which streams from each telescope are transformed to the Fourier domain and conjugate multiplied with every non-redundant pairing. This produces data known as visibilities for the each telescope baseline. The visibility data then undergoes calibration and imaging, which first accounts for instrumental and atmospheric factors and then grids and performs an inverse Fourier transform spatially to form an initial image. Subsequent deconvolution techniques are then applied to achieve the final image.

For more detailed information on correlation and related signal processing for radio astronomy, the reader is directed to the standard references [1], [2].

B. Lustre

Lustre is a high-performance, scalable, open-source filesystem [3], which is widely used by cluster and supercomputing systems. Nodes of such systems are Lustre clients, and by mounting and accessing the filesystem they interface with a number of Lustre components. Metadata Servers (MDS) provides access to file metadata stored in one or more Metadata Targets (MDT), and Object Storage Servers (OSS) provide file and network handling for file data stored across one or more Object Storage Targets (OST). The interaction of these components is handled by a Management Server (MGS).

C. HDF5

The Hierarchical Data Format 5 (HDF5) is becoming an industrial standard as a flexible data format and IO library for storing and accessing large and complex hierarchical data objects. It supports a variety set of IO operations for both serial and parallel applications. In particular, while working with global filesystems such as Lustre, the HDF5 library provides the ability to handle synchronous IO via the MPI-IO interface, as well as asynchronous IO via hyper-slab operations. The latter was chosen in this work for the HDF5 interface implementation.

D. ADIOS

The Adaptive IO System (ADIOS) [6] is an IO system/library aiming to improve parallel IO throughput for extremely large scientific datasets. ADIOS provides two sets of application interfaces, one through XML configuration files and the other based on function calls.

With the XML interface for collective parallel IO, ADIOS is also a powerful middleware providing the ability to change the output file format and transport method without touching a single line of the application code, but rather simply changing the parameters in XML configuration files. This feature is highly helpful for comparing different file formats and transport methods to find the optimum one.

On the other hand, through the non-XML interface, ADIOS can be used more flexibly compared to the XML interface in terms of enabling non-collective IO operations. This work will use the non-XML interface and non-collective IO of ADIOS for testing, as will be mentioned in later sections.

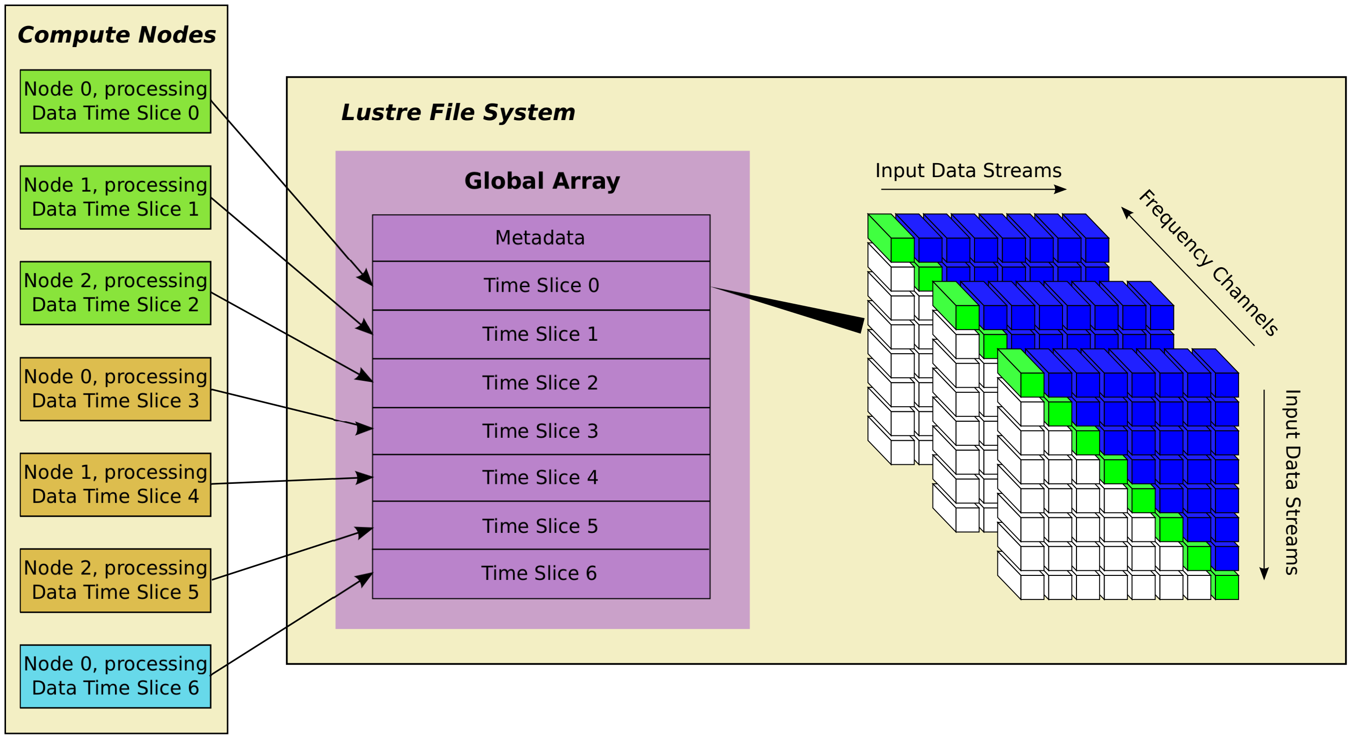

Compute Nodes
Node 0, processing Data Time Slice 0
Node 1, processing Data Time Slice 1
Node 2, processing Data Time Slice 2
Node 0, processing Data Time Slice 3
Node 1, processing Data Time Slice 4
Node 2, processing Data Time Slice 5
Node 0, processing Data Time Slice 6
Lustre File System
Global Array
Metadata
Time Slice 0
Time Slice 1
Time Slice 2
Time Slice 3
Time Slice 4
Time Slice 5
Time Slice 6
Input Data Streams
Frequency Channels
Input Data Streams

Figure 1. Shown is the output data flow and pattern of the time-division multiplex radio astronomy signal correlator used in this work.

III. Method

The correlation code adopted in this work is a GPU cluster based prototype correlator [4]. It implements two models to subdivide and distribute correlation tasks, based on time-division and baseline-division respectively. The time-division correlation model is chosen for all testing in this work, because a global filesystem is more beneficial in this case, in terms of managing the output data in a more usable format while achieving good scalability.

More specifically, output data of a baseline-division or frequency-division correlator can be arbitrarily appended over time, and one compute node therefore can write output data for an arbitrary amount of time independently of others. On the other hand, in a time-division correlator, output data of different time slices may not be produced in exactly corresponding sequences, and therefore cannot be easily appended. Workarounds to this problem could be either to introduce a management node to re-arrange the output data into correct sequences before writing to disks, such as what is done in DiFX [5], or to create a file for every single time slice. However, having all output data collected by a single computing node means that all disk output operations concentrate on a single node, which would inevitably cause a bottleneck when the problem size is sufficiently large. Creating a file for every time slice is not a good practice either due to the fact that for a certain telescope configuration, the data size of a time slice is usually a fixed number, which could be highly unoptimal for the file/IO system to create files for.

This work is based on a more scalable solution. We took the correlator code in [4] and replaced the output module with a switchable implementation between HDF5 and ADIOS, which writes output data directly from correlation nodes to global filesystems. As shown in Figure I, the output data is ensured to be in the correct order by creating global arrays with pre-defined structures, and then asynchronously filling in data from multiple compute nodes through IO libraries while correlation is being processed.

A. ADIOS

For the ADIOS subroutine, the non-XML interface is used for asynchronous output. This is because that in a typical cluster based correlator, some nodes need to be allocated for non-correlation functions, such as data streaming and work flow management. These non-correlation nodes do not write any output data, and this results in an unbalanced data output pattern from each compute node, which is very difficult to be handled by the XML interface of ADIOS or other collective IO functions.

Moreover, the correlator works in a time-division pattern, which means each correlation node processes a time chunk of data across the entire problem domain, while the total number of time chunks is arbitrary. Therefore, even for the correlation nodes alone, the amount of data each node is writing may not be balanced. As shown in I for instance, Rank 1 needs to write three times while Rank 2 and 3 only twice. Collective functions such as the MPI-IO or the XML interface of ADIOS then become less applicable in this case. However, ADIOS has the ability to handle such unbalanced output pattern through the non-XML interface.

In terms of the file format, this work uses the ADIOS standard configuration with the POSIX transport method. This results in a set of files with the extension of .bp for each global array.

B. HDF5

The HDF5 subroutine is implemented in a similar way. A HDF5 dataset is created with pre-defined size and structure. Compute nodes then write time slices as HDF5 hyper-slabs into the HDF5 dataset. Due to the asynchronous nature of the time-division correlator, the collective MPI interface of HDF5 library is not applicable. Therefore, this subroutine is implemented using the serial HDF5 library.

IV. Testing

This section will introduce the testing environment, in particular, the hardware and software specificaitons of Magnus and Fornax supercomputers that have been used in this work. Testing results are then presented with a variety of figures.

A. Testing Environment

Testing was carried out using iVEC's **Magnus** system, a Cray XC30 supercomputer. Magnus is the first phase of a petascale system, consisting of 208 compute nodes. Each node contains two Intel Xeon E5-2670 CPUs, which have an eight core Sandy Bridge architecture, and 64 GB of random access memory. The nodes are interconnected by an Aries interconnect in a dragonfly topology, capable of 72Gbps of bandwidth per node. In addition to the compute nodes, at the time of testing Magnus had 24 service nodes, which route traffic between the Aries interconnect and an Infiniband network. The latter provides access to the Lustre version 2.2.0 filesystem, provided by a Cray Sonexion 1600. This has two petabytes of storage via nine Scalable Storage Units (SSUs). These SSUs have 8 OSTs, each using a 8+2 RAID 6 configuration. The specification of each SSU has a 5 GB per second bandwidth from the IOR benchmark, and thus the expected peak bandwidth is 45 GB per second.

In terms of software on Magnus, the GCC version 4.7 compiler was used, along with the Cray MPICH version 6.1 library, HDF5 version 1.8.12 and ADIOS version 1.4.1.

Another iVEC supercomputer, **Fornax**, was used in this work as a reference system. Fornax was designed for data intensive research, especially radio astronomy related data processing. It consists of 96 nodes, each containing two Intel Xeon X5650 CPUs, an NVIDIA Tesla C2075 GPU and 72 gigabytes of system memory. The Intel 5520 Chipset is used in the computing node architecture, which enables the

NVIDIA Tesla C2075 GPU to work on an x16 PCI-E slot and two QLogic Infiniband IBA 7322 QDR cards to run on two x8 PCI-E slots.

The back-end of Fornax's Lustre system is a SGI Infinite S16k, which is a re-badged DDN SFA 10k, consisting of 8 Object Storage Servers (OSSs) and 44 Object Storage Targets (OSTs), of which 32 are assigned to the scratch file system used in this testing. Each of the OSSs has dual 4x QDR Infiniband connections to the switch connecting compute nodes, and the OSTs are connected to the OSSs via 8 4x QDR Infiniband connections. Each OST consists of 10 Hitachi Deskstar 7K2000 hard drives arranged into a 8+2 RAID 6 configuration. Operational testing using the ost_survey Lustre benchmark achieved a mean bandwidth of 343 MB per second, and thus the expected bandwidth is approximately 11 GB per second.

The software used on Fornax included the GCC version 4.4 compiler, OpenMPI version 1.6.3, HDF5 version 1.8.12 and ADIOS version 1.4.1.

B. Testing Parameters

Testing was across a wide range of parameters, as shown in Table I. The number of frequency channels, noted as f, which is also the length of a visibility in the correlator output, varies from 128 to 1024. The number of input data streams, noted as n, varies from 100 to 400, and this number has a quadratic impact on the output data size. The ranges of these two parameters taken in the testing are normally seen in real telescopes. The data size per time slice in bytes, noted as s_t, is then given in Equation 1.

$$s_t = 4fn(n+1) \tag{1}$$

The number of time slices, noted as t, varies from 100 to 400, covering the optimal data sizes that were identified in preliminary testing. The global array size in bytes, noted as s, is then given in Equation 2

$$s = ts_t \tag{2}$$

Testing was conducted using from 20 to 90 compute nodes on both Fornax and Magnus, as 20 is where different configurations start to behave distinctly, and 90 is approaching the maximum number of nodes available on Fornax. The Lustre stripe sizes used in the testing varies from 1 to 8. This is because Fornax has 8 object storage nodes in total, while Magnus is less sensitive to this configuration as learned in preliminary testing.

C. Testing Results

Testing was conducted exhaustively within the parameter ranges. However, due to the space limitation, four testing schemes that were compiled using some of the most representative results are to be presented.

Table I
TESTING PARAMETERS

Parameters	Range	Stepping
Number of Frequency Channels	128 - 1024	x2
Number of Input Data Streams	100 - 400	+100
Number of Time Slices	100 - 400	+300
Compute Nodes	20 - 90	+10
Lustre Stripe Size	1 - 8	x2

1) Small frequency channels, comparing ADIOS and HDF5, with varying input streams: This testing scheme mainly intends to illustrate the performance difference between ADIOS and HDF5, with a relatively small number of frequency channels, 256, and a large number of time slices, 400. Testing results for this scheme are shown in Figure 2.

2) Large frequency channels, comparing ADIOS and HDF5, with varying input streams: In addition to the last one, this testing scheme changes to a relatively large number of frequency channels, which is 1024. Other parameter ranges remain the same. Testing results for this scheme are shown in Figure 3.

3) ADIOS only, comparing large and small time slices, with varying input streams: This testing scheme mainly intends to demonstrate how performance is affected by the number of time slices in an ADIOS global array, with a medium number of frequency channels, 512. Testing results for this scheme are shown in Figure 4.

4) Stability testing, comparing ADIOS and HDF5, with varying Lustre stripe size: This testing scheme mainly intends to illustrate the performance stability of ADIOS and HDF5 by plotting a number of testing result samples and then conclude to an average curve. A medium number of frequency channels, 512, and a medium number of input streams, 200, were chosen for the scheme. Testing results are shown in Figure 4.

V. DISCUSSION

This section will firstly explain the significance of the unusual asynchronous data output method used in this work, and then interpret the testing results presented in the last section to highlight some of the most valuable information.

A. Asynchronous Data Output

The asynchronous data output method used in this work is not applied as commonly as synchronous methods such as algorithms using MPI-IO. In particular, neither HDF5 nor ADIOS evidently recommends users to implement their applications in this way. As a result, some lower-level interface tricks need to be played to do this, particularly, the non-XML interface of ADIOS. However, for algorithms that have a non-synchronous nature, for instance, most of the time-division multiplex systems, asynchronous data operations on global filesystems still provide a practical and efficient way to resolve problems.

B. Global Array Size

One interesting result we encountered on Fornax is that the peak result is higher than expected. We suspect this is due to caching, and thus we have investigated how the global array size affects performance, as shown in Figure 6. The performance for Fornax decreases significantly around the global array size of 32 GB, matching the cache size of the SGI Infinite S16K. There is a similar effect for Magnus, while the performance boost is not as significant, the effect exist for larger global array sizes

C. ADIOS & HDF5

As seen in the Testing section, ADIOS shows absolute advantages over HDF5 for this type of asynchronous data writing to Lustre filesystems. For the largest global array size, which is least affected by caching, the bandwidth achieved on Magnus by ADIOS was 11 GB/s, and HDF5 achieved 5.5 GB/s. For global array sizes that fit in cache, ADIOS performs an order of magnitude faster than HDF5. This is likely due to the advanced buffering system built-into ADIOS, and dedicated optimizations for parallel IO.

D. Magnus & Fornax

As seen in Figure 2 and 3, Fornax generally performs better on smaller datasets that fit in cache, while for larger global array sizes, Magnus significantly outperforms Fornax.

There is another factor affecting the performance on Fornax for a large number of compute nodes, for instance, in Figure 5(g). Fornax has 96 nodes in total, and when the testing environment approaches this number, it means testing is occupying the entirety of Fornax, and this removes the impact of other users on the system. On the other hand, Magnus consists of 300 compute nodes and occupying 90 is not sufficient to see similar performance benefits.

E. Lustre Stripe Size & Number of Compute Nodes

In ADIOS testing, Fornax is more sensitive than Magnus in terms of the Lustre stripe size. A small Lustre stripe size would limit Fornax's scalability with compute nodes, while Magnus does not seem to be limited by this. More specifically, as shown in Figure 5(a) and (c), when a Lustre stripe size 1 or 2 was chosen, the performance of Fornax is hardly scalable with the number of compute nodes, whereas Magnus still shows decent scalability.

For HDF5, however, neither scales well. Even a decreasing trend is seen on Fornax, as shown in Figure 5(d) and (f).

VI. Conclusion

In this paper, we re-implemented the data output module of a GPU cluster based radio astronomy signal correlator [4] in a more flexible and scalable way using the HDF5 and ADIOS libraries, and then carried out a series of testing on iVEC's Magnus and Fornax supercomputers. Testing results showed that under most circumstances ADIOS achieved an order of magnitude higher throughput than HDF5, therefore, it came into a conclusion that ADIOS is more suitable for such multi-node asynchronous data output applications. The performance is also affected by a number of other factors such as the global array size, the inner dimension sizes of the global array, and the Lustre stripe size. In addition, the two supercomputers used for testing behaved differently. More specifically, Fornax achieved superior throughputs with small data scales and large numbers of compute nodes, whereas the Cray XC30 machine, Magnus, dominated for larger data scales.

A. Future Work

There are a number of avenues for further research. Larger global array sizes could be investigated to provide more information of the domain free from caching effects. Additionally, benchmarks using the entirety of Magnus could be carried out, to see if there is performance peak similar to that of Fornax when using the whole system. Finally, a similar investigation on local filesystems of Fornax would provide an interesting comparison.

Acknowledgment

The work was supported by iVEC through the use of advanced computing resources located at iVEC@CSIRO and iVEC@UWA, and through the provision of funding for travel and accommodation to attend the Cray User Group 2014 meeting. We thank David Schibeci and Ashley Chew from the iVEC operations team for providing storage system specifications of the Magnus and Fornax supercomputers.

References

[1] Taylor, Greg B., Chris Luke Carilli, and Richard A. Perley, *Synthesis Imaging in Radio Astronomy II*, vol. 180, 1999.

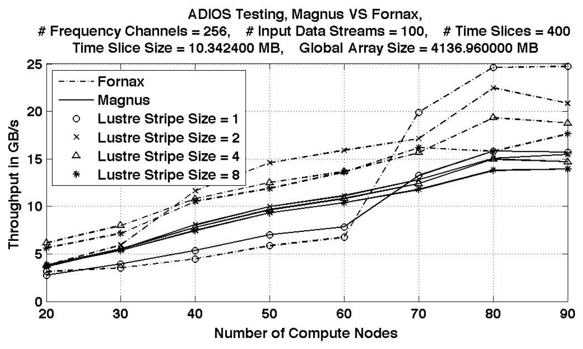
[2] Thompson, A. Richard, James M. Moran, and George W. Swenson Jr., *Interferometry and synthesis in radio astronomy*, John Wiley and Sons, 2008.

[3] Peter J. Braam and Michael J. Callahan, *Lustre: A SAN File System for Linux*, http://www.lustre.org/docs/luswhite.pdf, Stelias Computing Incorporated, 1999.

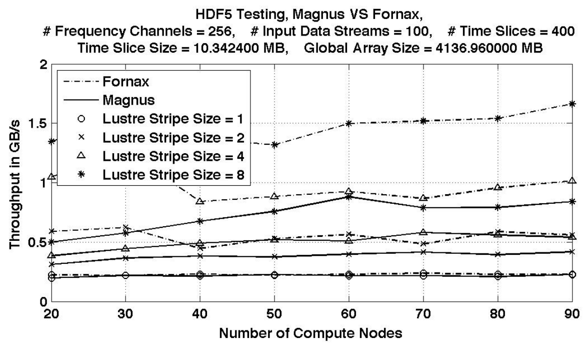
[4] R. Wang and C. Harris, *Scaling radio astronomy signal correlation on heterogeneous supercomputers using variousdata distribution methodologies*, Experimental Astronomy, Vol. 36, pp. 433-449, 2013.

[5] A. T. Deller, S. J. Tingay, M. Bailes, and C. West, *Difx: A software correlator for very long baseline interferometry using multiprocessor computing environments*, Publications of The Astronomical Society of The Pacific, Vol. 119, pp. 318336, March 2007.

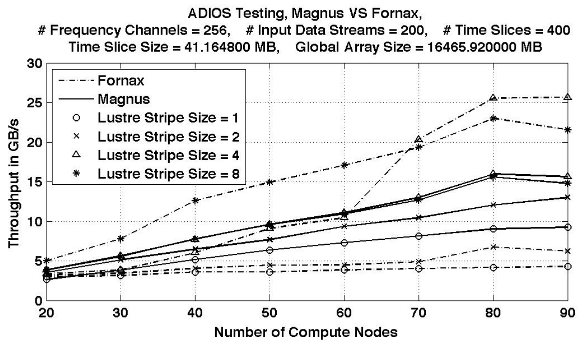
[6] S. Klasky et al., *Adaptive IO System*, in Proceedings of the Cray User Group meeting 2008, 2008.



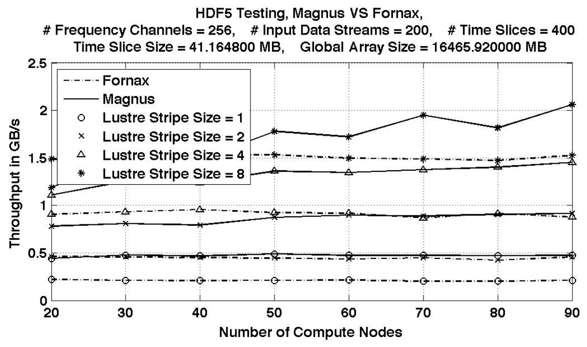
(a) ADIOS: Input Data Streams = 100



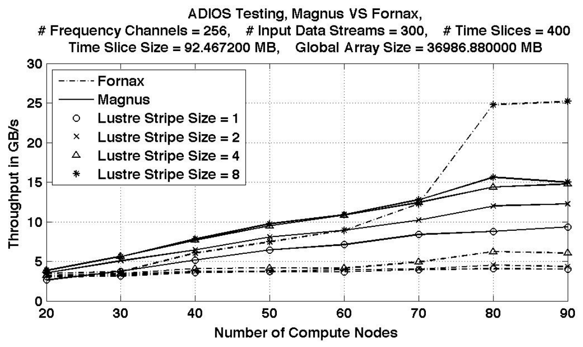
(b) HDF5: Input Data Streams = 100



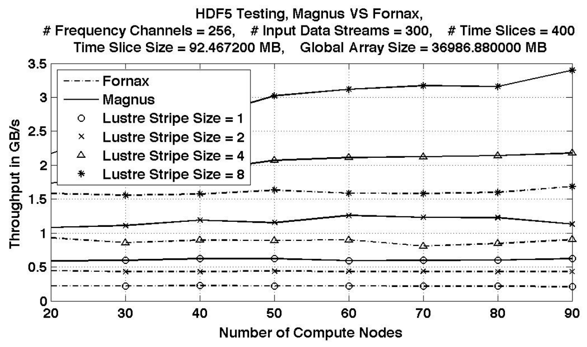
(c) ADIOS: Input Data Streams = 200



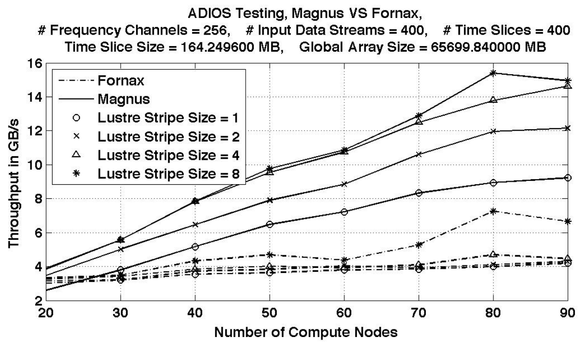
(d) HDF5: Input Data Streams = 200



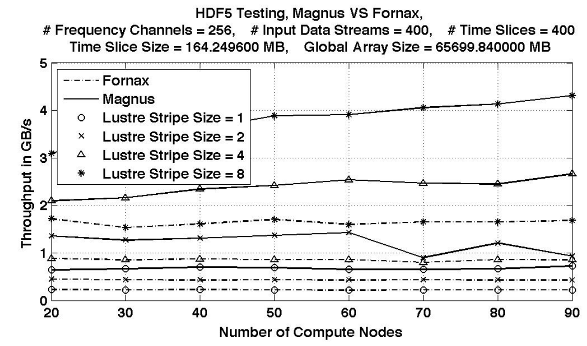
(e) ADIOS: Input Data Streams = 300



(f) HDF5: Input Data Streams = 300

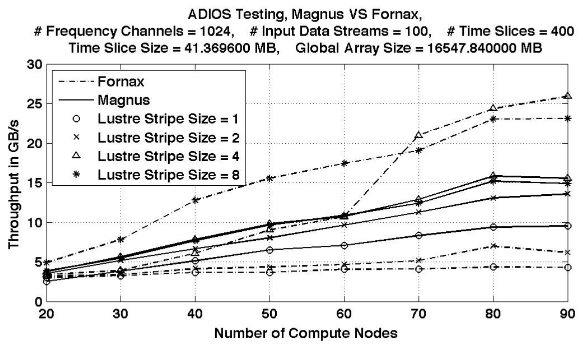


(g) ADIOS: Input Data Streams = 400

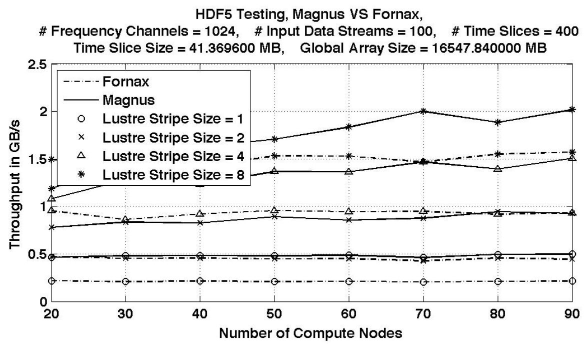


(h) HDF5: Input Data Streams = 400

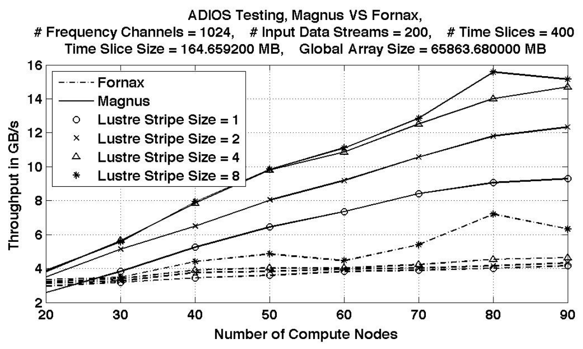
Figure 2. Shown is the testing results for a relatively small data scale, with 256 frequency channels and 400 time slices. Testing varies the number of input data streams from 100 to 400, and compares the performance between HDF5 and ADIOS.



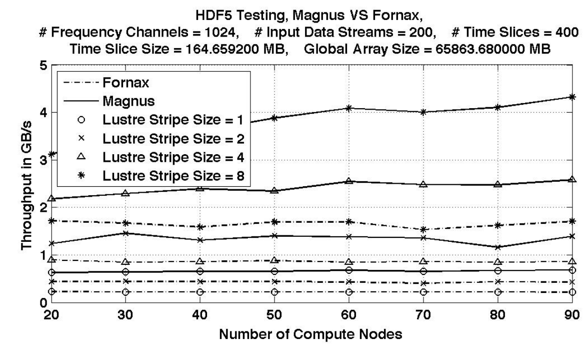
(a) ADIOS: Input Data Streams = 100



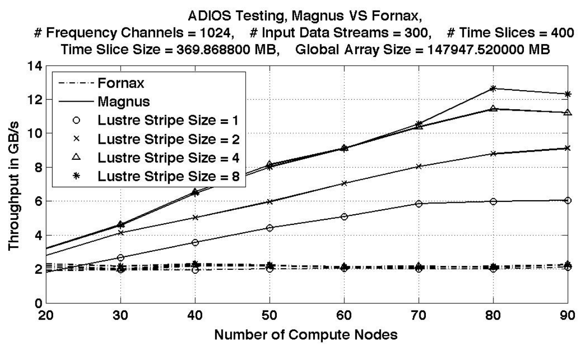
(b) HDF5: Input Data Streams = 100



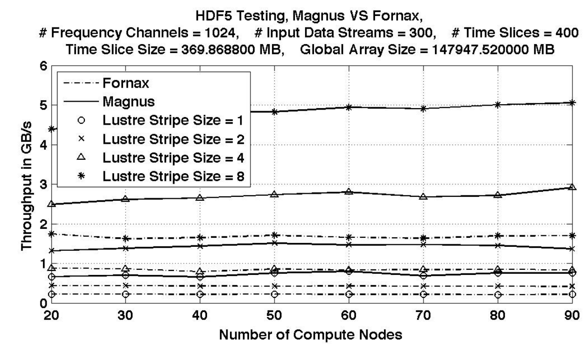
(c) ADIOS: Input Data Streams = 200



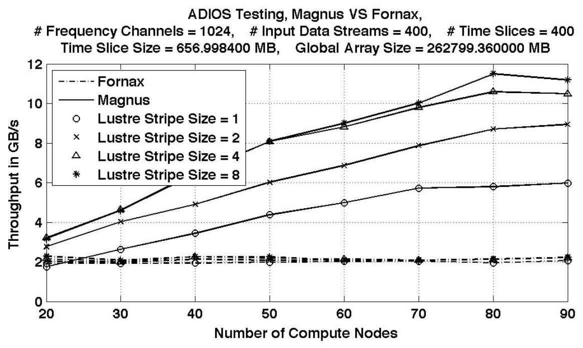
(d) HDF5: Input Data Streams = 200



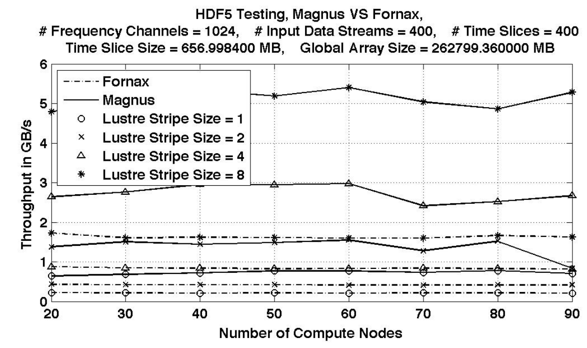
(e) ADIOS: Input Data Streams = 300



(f) HDF5: Input Data Streams = 300

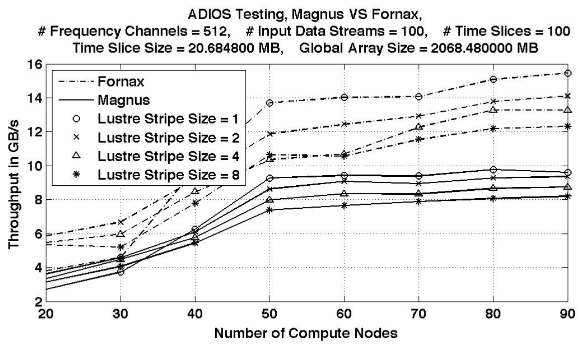


(g) ADIOS: Input Data Streams = 400

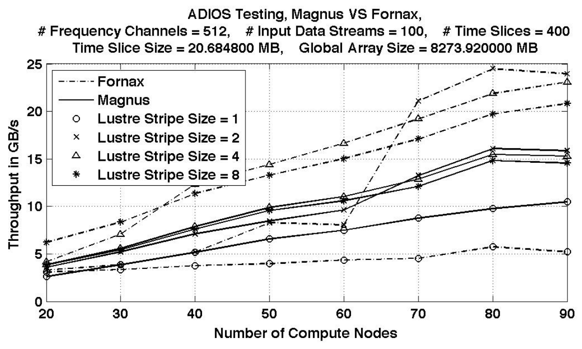


(h) HDF5: Input Data Streams = 400

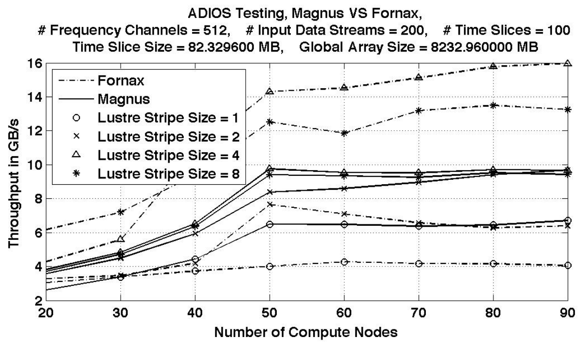
Figure 3. Shown is the testing results for a relatively large data scale, with 1024 frequency channels and 400 time slices. Testing varies the number of input data streams from 100 to 400, and compares the performance between HDF5 and ADIOS.



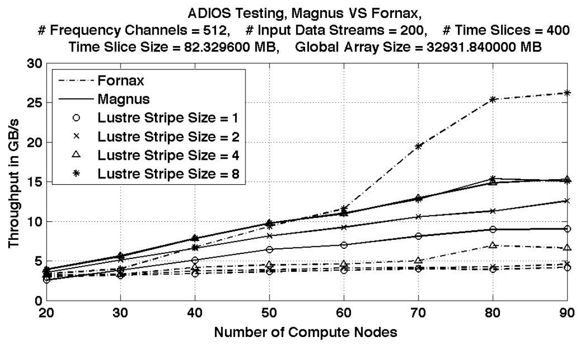
(a) 100 time slices, 100 input data streams



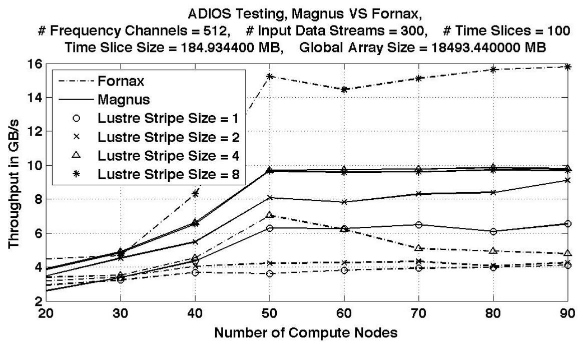
(b) 400 time slices, 100 input data streams



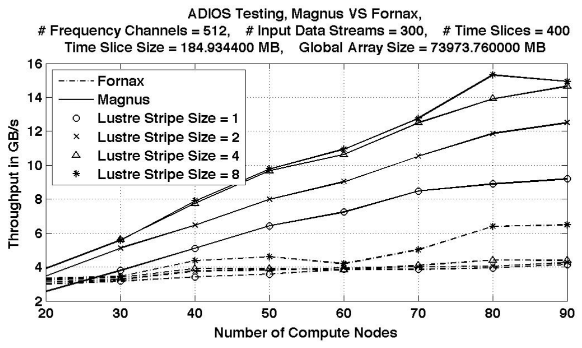
(c) 100 time slices, 200 input data streams



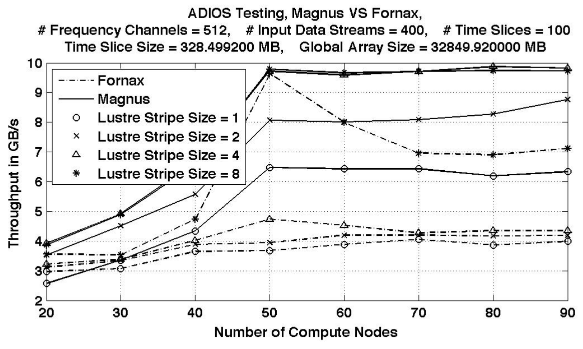
(d) 400 time slices, 200 input data streams



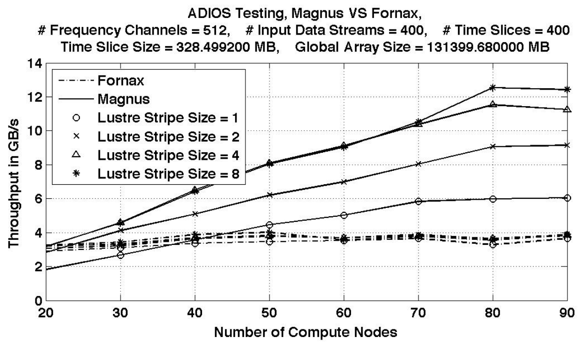
(e) 100 time slices, 300 input data streams



(f) 400 time slices, 300 input data streams

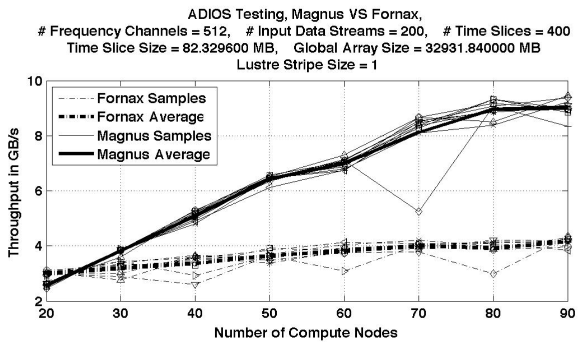


(g) 100 time slices, 400 input data streams

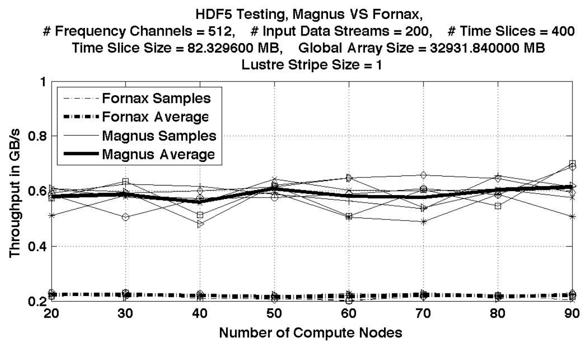


(h) 400 time slices, 400 input data streams

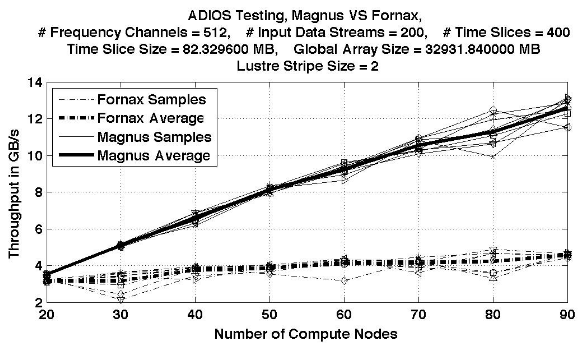
Figure 4. Shown is the ADIOS testing results for a medium data scale, with 512 frequency channels. Testing varies the number of input data streams from 100 to 400, and compares the performance between 100 and 400 time slices.



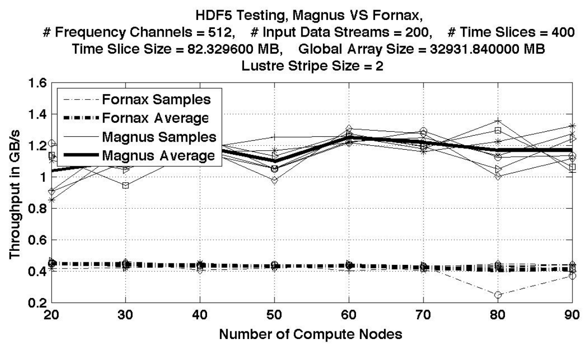
(a) ADIOS, Lustre stripe size = 1



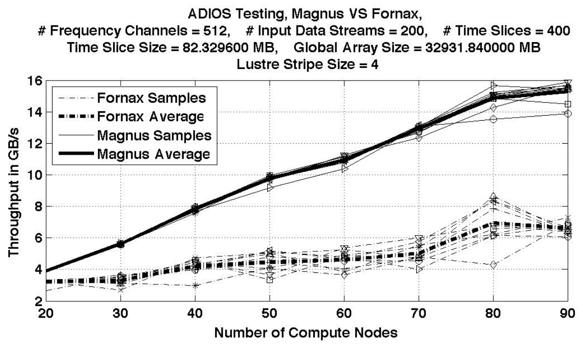
(b) HDF5, Lustre stripe size = 1



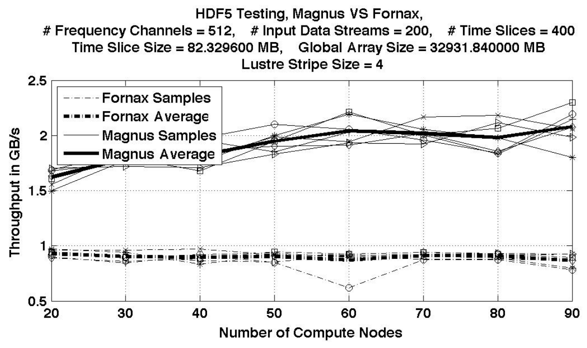
(c) ADIOS, Lustre stripe size = 2



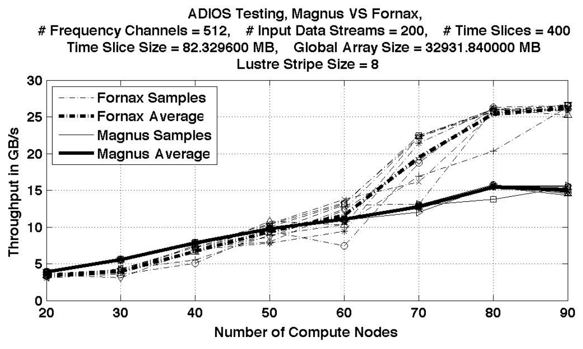
(d) HDF5, Lustre stripe size = 2



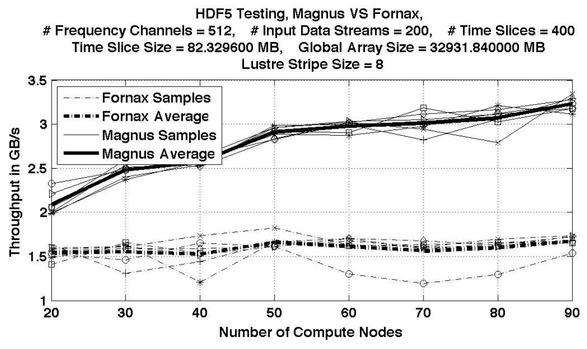
(e) ADIOS, Lustre stripe size = 4



(f) HDF5, Lustre stripe size = 4



(g) ADIOS, Lustre stripe size = 8



(h) HDF5, Lustre stripe size = 8

Figure 5. Shown is the results of stability testing, with 512 frequency channels, 400 time slices. Testing varies the Lustre stripe size from 1 to 8, and compares the performance between HDF5 and ADIOS.

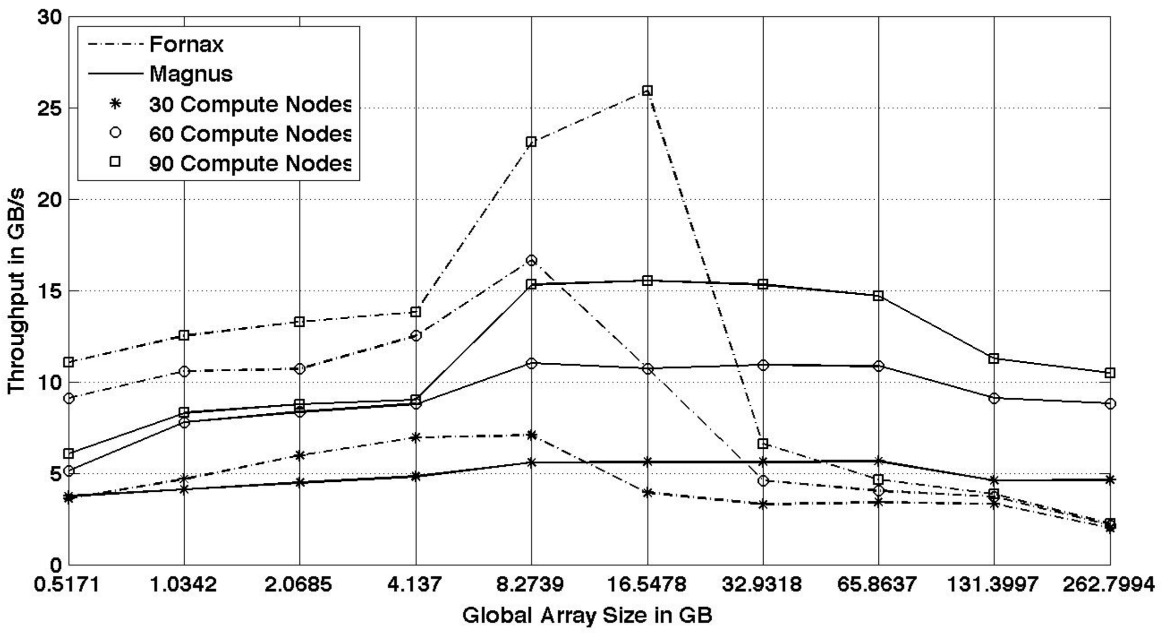


Figure 6. Shown is the data rate achieved for a range of global array sizes, using ADIOS on both Magnus and Fornax. A Lustre stripe size of 4 was used. The impact of caching can be seen up to 32 GB on Fornax, and 131 GB on Magnus.