46th AIAA Aerospace Sciences Meeting and Exhibit
7 - 10 January 2008, Reno, Nevada

AIAA 2008-479

Benchmarking the CGNS I/O performance

Thomas Hauser*

I. Introduction

Linux clusters can provide a viable and more cost effective alternative to conventional supercomputers for the purposes of computational fluid dynamics (CFD). In some cases, the Linux supercluster is replacing the conventional supercomputer as a large-scale, shared-use machine. In other cases, smaller clusters are providing dedicated platforms for CFD computations. One important, often overlooked, issue for large, three dimensional time-dependent simulations is the input and output performance of the CFD solver. The development of the CFD General Notation System (CGNS) (see[1–3]) has brought a standardized and robust data format to the CFD community, enabling the exchange of information between the various stages of numerical simulations. This paper presents detailed benchmarks of the serial and parallel I/O performance of the CGNS software.[4] The performance on serial and parallel file systems will be discussed in this paper.

II. The CGNS system

The specific purpose of the CFD General Notation System (CGNS) project is to provide a standard for recording and recovering computer data associated with the numerical solution of the equations of fluid dynamics. The intent is to facilitate the exchange of Computational Fluid Dynamics (CFD) data between sites, between applications codes, and across computing platforms, and to stabilize the archiving of CFD data.

The CGNS system consists of two parts: (1) the Standard Interface Data Structures, SIDS and (2) the ADF library. The "Standard Interface Data Structures" specification constitutes the essence of the CGNS system. While the other elements of the system deal with software implementation issues, the SIDS specification concerns itself with defining the substance of CGNS. It precisely defines the intellectual content of CFD-related data, including the organizational structure supporting such data and the conventions adopted to standardize the data exchange process. The SIDS are designed to support all types of information involved in CFD analysis. While the initial target was to establish a standard for 3D structured multi-block compressible Navier-Stokes analysis, the SIDS extensible framework now includes unstructured analysis, configurations, hybrid topology and geometry-to-mesh association. Although the SIDS specification is independent of the physical file formats, its design was targeted towards implementation using the ADF Core library, but one is able to define a mapping to any other data storage format. Currently, in CGNS 2.5 adf and hdf5 can be selected as the underlying data storage format. CGNS version 3.0, which is currently under testing, extends the underlying data formats to XML.

A. ADF data format

The "Advanced Data Format" (ADF) is a concept defining how the data is organized in the storage media. It is based on a single data structure called an ADF node, designed to store any type of data. Each ADF file is composed of at least one node called the "root". The ADF nodes follow a hierarchical arrangement from the root node down.

B. HDF5 data format

In the current version CGNS can also be configured to use HDF5 as the underlying format. The format of an HDF5 file on disk encompasses several key ideas of the HDF4 and AIO file formats as well as addressing

*Director, Center for High Performance Computing, Utah State University, Associate Fellow Member AIAA some shortcomings therein. The new format is more self-describing than the HDF4 format and is more uniformly applied to data objects in the file.

An HDF5 file appears to the user as a directed graph. The nodes of this graph are the higher-level HDF5 objects that are exposed by the HDF5 APIs. At the lowest level, as information is actually written to the disk, an HDF5 file is made up of the following objects:

- A super block
- B-tree nodes (containing either symbol nodes or raw data chunks)
- Object headers
- A global heap
- Local heaps
- Free space

The HDF5 library uses these low-level objects to represent the higher-level objects that are then presented to the user or to applications through the APIs. For instance, a group is an object header that contains a message that points to a local heap and to a B-tree which points to symbol nodes. A dataset is an object header that contains messages that describe datatype, space, layout, filters, external files, fill value, etc with the layout message pointing to either a raw data chunk or to a B-tree that points to raw data chunks.

In addition, the new upcoming 3.0 release of CGNS has the ability to also use XML as the underlying data format.

C. XML data format

The Extensible Markup Language (XML) [5] and [6] is already used in office applications, e.g. OpenOffice, and internet applications. The language has proven to be an effective method of transporting data from one location to another. XML is rapidly becoming the standard for exchanging data because it adds portability and open accessibility to traditional closed data formats.

XML has started as a subset of the Standard Generalized Markup Language (SGML), which was originally created as a universally interchangeable data format with rich information storage capabilities. XML has become the preferred method of data exchange because of its simplicity and extensibility. XML markup tags are entirely user-defined and therefore extensible. This made it fairly simply to implement the XML storage layer.

D. CGNS parallel I/O

To facilitate convenient and high-performance parallel access to netCDF files, we have defined a new parallel interface and provide a prototype implementation. Since a large number of existing users are running their applications over CGNS, our parallel CGNS design retains the original SIDS API and introduces extensions which are minimal changes from the original API. The parallel API is distinguished from the original serial API by prefixing the C function calls with "cgp_" instead of "cg_" as in the standard SIDS API.

Our parallel CGNS API is built on top of parallel HDF5, which enables the implementation to be simple. In parallel CGNS a file is opened, operated, and closed by the participating processes in a communication group. In order for these processes to operate on the same file space, especially on the structural information contained in the file header, a number of changes have been made to the original serial CGNS API.

For the function calls that create/open a CGNS file, an MPI communicator is added in the argument list to define the participating I/O processes within the file's open and close scope. By describing the collection of processes with a communicator, we provide the underlying implementation with information that can be used to ensure file consistency during parallel access. An MPI Info object is also added to pass user access hints to the implementation for further optimizations. Using hints is not mandatory (MPI_INFO_NULL can be passed in, indicating no hints). However, hints provide users the ability to deliver the high-level access information to HDF5 and MPI-IO libraries. Traditional MPI-IO hints tune the MPI-IO implementation to the specific platform and expected low-level access pattern, such as enabling or disabling certain algorithms or adjusting internal buffer sizes and policies. These are passed through the HDF5 layer to the MPI-IO implementation. Hints can be used to describe expected access patterns at the CGNS level of abstraction, in terms of variables and records. These hints can be interpreted by the CGNS implementation and either used internally or converted into appropriate MPI-IO hints. Parallel files may be stored and accessed in great many ways that depend on the operating system, particular devices used for storage and any middle ware that may live in between. In order to optimize file access the programmer may wish to provide additional information to MPI, in hope that MPI would know what to do with it. Such information is referred to as hints and there is a special MPI construct called the info object that is supposed to collect all the hints. Once constructed the info object can be passed to MPI_File_open, MPI_File_delete, MPI_File_set_view and MPI_File_set_info. It should be understood though that any hints you may wish to give MPI this way are only advisory and what MPI is going to do with them is implementation dependent.

The same syntax and semantics is maintained for the CGNS attribute functions, and inquiry functions as the original ones. These functions are also made collective to guarantee consistency of dataset structure among the participating processes in the same MPI communication group. For instance, all processes must call the write_zone functions with the same values to get consistent dataset definitions.

III. Performance Results

A. Serial Performance

1. Write Performance Increasing Number of Zones

Figure 1 shows the I/O times for different zone sizes and an increasing number of zones.

Figure 2 shows the I/O sizes for different zone sizes and an increasing number of zones.

2. Write performance Constant Number of Zones

Figure 3 show the timing of all implementations for 100 zones.

Figure 4 show the file sizes of all implementations for 100 zones.

B. Parallel Performance

1. Read Performance

The following scaling results were obtained on the Linux cluster FAUST. It is configured with a scratch file system spanning all compute nodes consisting of PVFS2. A single block data set has been chosen to perform the I/O test. Data sizes of 64^3 (8.2 MB), 128^3 (57 MB) and 256^3 (417 MB) were used to obtain the wall clock time and speedups. In Figures 5 and 6 the wall clock time for parallel CGNS I/O is presented. For one node performance measurements the serial CGNS I/O using the HDF5 layer is used. All data sets show a improvement in read performance except for 8 processors.

Figure 7 shows the speedup for reading several data sets. Initially the speedup is increasing as expected, but independent of the data set size a slow down in the neighborhood of 8 processors is observed. Increasing the number of processors beyond 16 gives again speedup for the I/O performance. The improvement using parallel CGNS and the PVFS on a Linux cluster is satisfactory and shows the potential performance improvements of using parallel I/O for large scale CFD applications.

2. Write Performance

The write performance is tested on PVFS2 on the Faust cluster. We measured three different times: total time of the output, time until end of the creation of the data sets, write time into the empty data sets. In this test case three different grid sizes were used: 50x50x50, 150x150x150 and 250x250x250. The problem size is also scaled with the number of processors. Each processor contains one zone with the above mentioned sizes. This means that the file size increases up to a maximum size on 16 processors of 23.0 MB, 618 MB and 2.8 GB. Note also that HDF5 is able to write files large than 2.0 GB on a 32 bit computing platform.

Figure 8 shows the improvement in performance of the parallel file system compared to the serial I/O. The total time shows some increase in time with the increase of the problem time. To look more detailed into the two phases of the output, the creation time and the write time into the empty data set are plotted in Figures 9 and 10 respectively.

As expected, since the creation of the data sets is done collectively by all processors, the creation time scales nearly linearly with the number of processors as shown in Figure 9. This is the phase of the writing which cannot be parallelized. This time is about 20% of the total I/O time.

In Figure 10 the time the parallel program uses to write into the empty, previously created, data sets. It shows not perfect scaling, which would be constant I/O time over the whole range of processors, but the increase in time is not very dramatic.

IV. Conclusion

Linux cluster computing appears to be the next-generation of supercomputing, offering options from large shared-use machines to small, dedicated, single application systems. However, optimal use of this systems for computational fluid dynamics will require tuning the software for the new hardware architectures.

Future work could include the completion of a production quality parallel CGNS API and making it freely available to the high-performance computing CFD community. Testing on different platforms and filesystem is also currently under way.

Acknowledgement

The author would like to thank Bruce Wedan for making the 3.0 implementation available, and his contribution to CGNS.

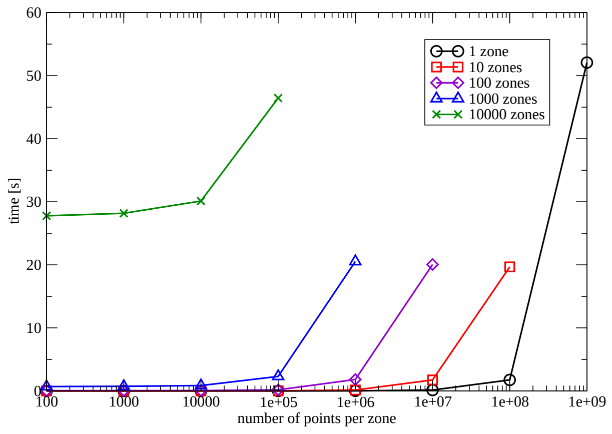
References

[1]Poirier, D., Allmaras, S. R., McCarthy, D. R., Smith, M. F., and Enomoto, F. Y., "The CGNS System," *AIAA Paper 98-3007*, 1998.

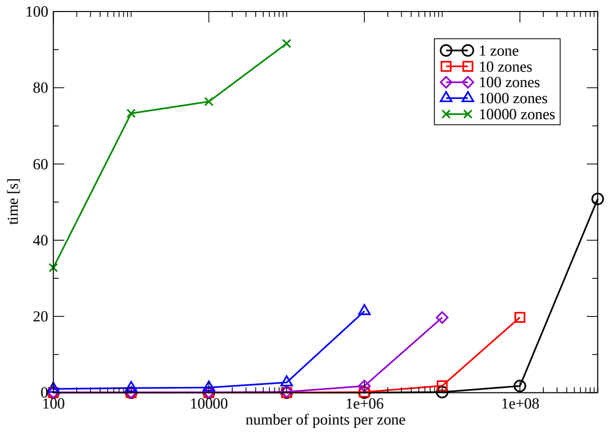
[2]Poirier, D. M. A., Bush, R. H., Cosner, R. R., Rumsey, C. L., and McCarthy, D. R., "Advances in the CGNS Database Standard for Aerodynamics and CFD," *AIAA Paper 2000-0681*, 2000.

[3]Legensky, S. M., Edwards, D. E., R. H. Bush, D. M. A. P., Rumsey, C. L., Cosner, R. R., and Towne, C. E., "CFD General Notation System (CGNS): Status and Future Directions," *AIAA Paper 2002-0752*, 2002.

[4]Hauser, T., "Parallel I/O for the CFD General Notation System," *Proceedings of the 42nd AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, January 05-08 2004.

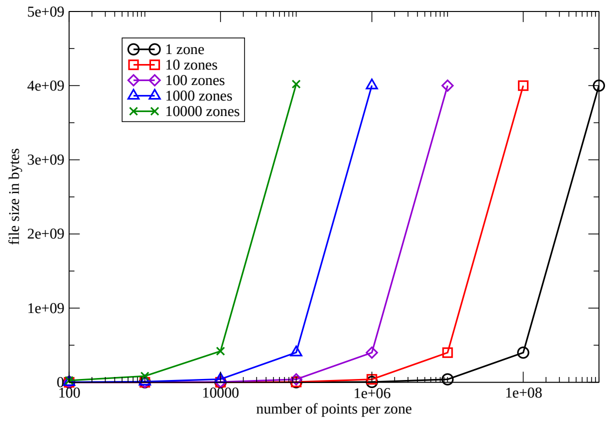


(a) ADF data format

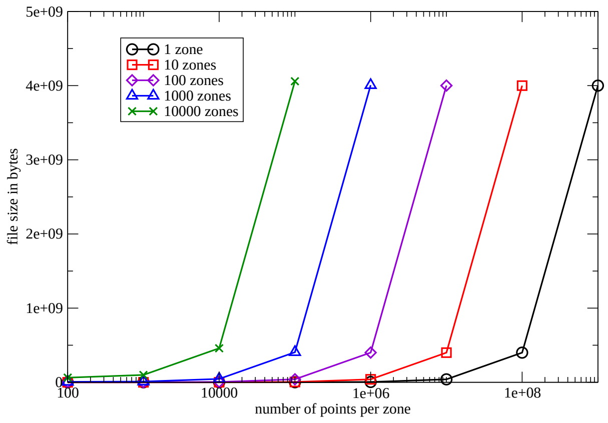


(b) HDF5 data format

Figure 1. I/O times for different number of zones for CGNS 2.5



(a) ADF data format



(b) HDF5 data format

Figure 2. I/O sizes for different number of zones for CGNS 2.5

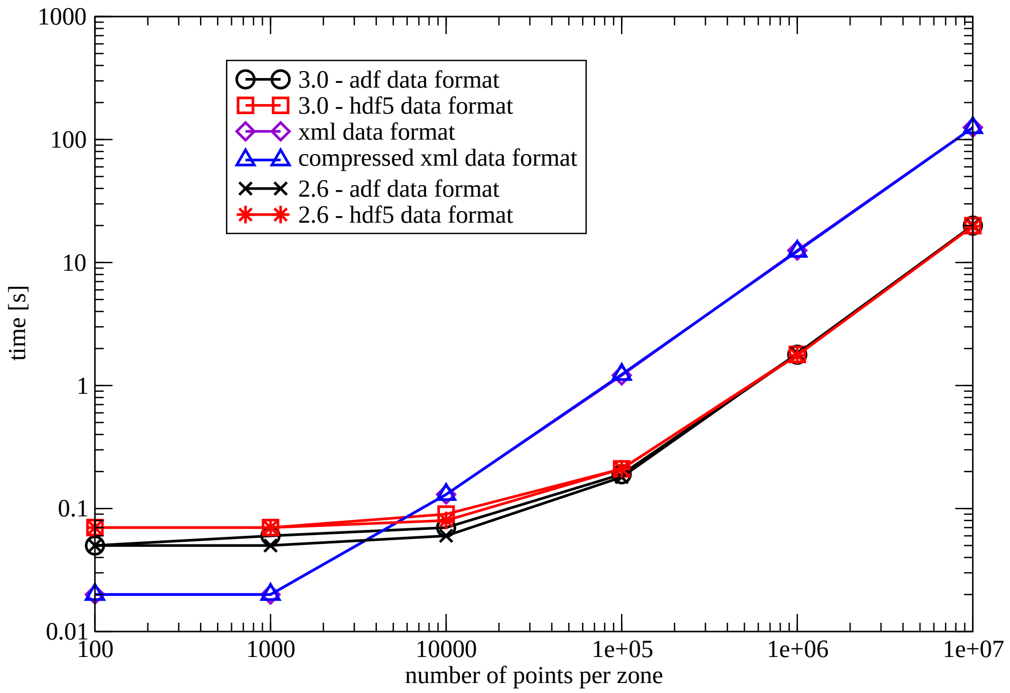


Figure 3. CGNS 3.0 and 2.5 write timings

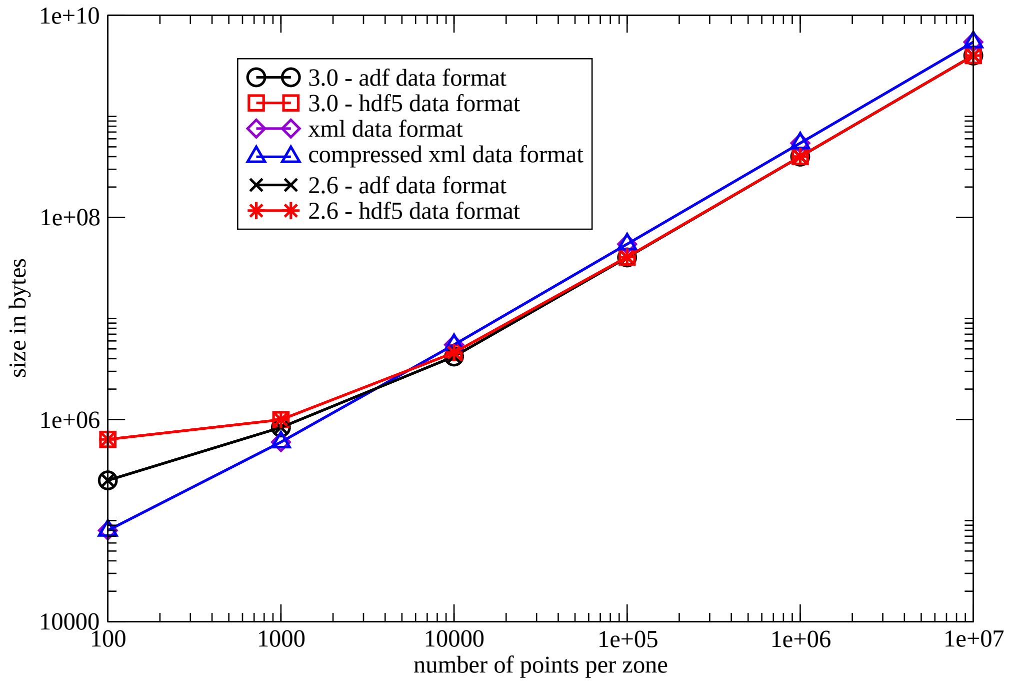


Figure 4. CGNS 3.0 and 2.5 write timings

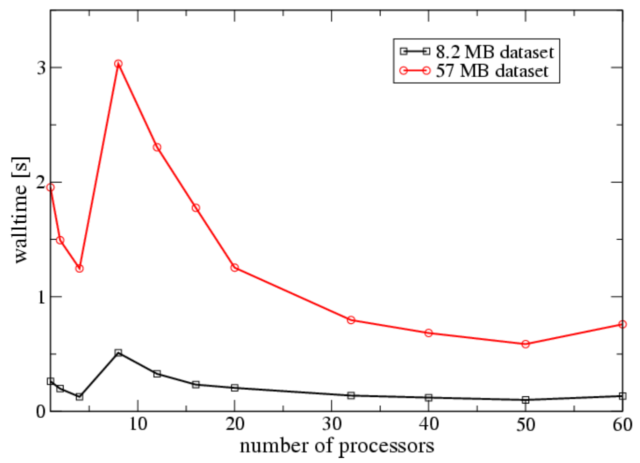


Figure 5. Wall time for parallel CGNS I/O using PVFS for the 8.2 and 57 MB data sets

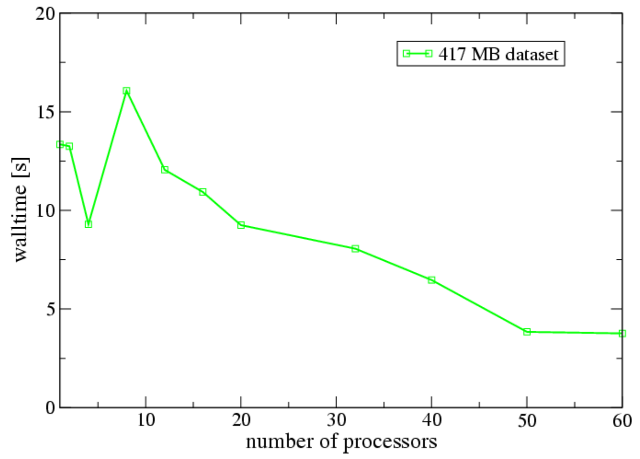


Figure 6. Wall time for parallel CGNS I/O using PVFS for 417 MB data set

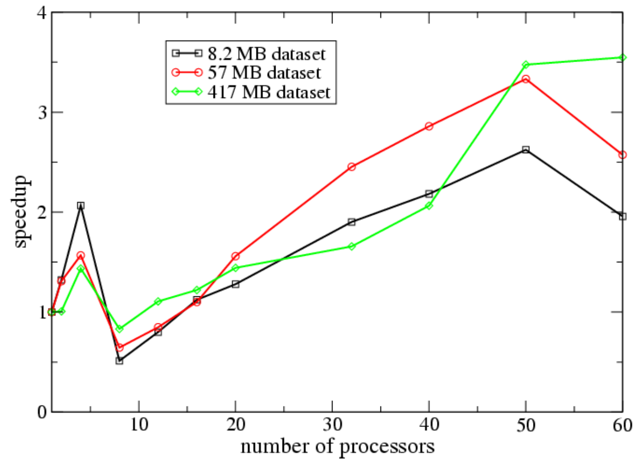


Figure 7. Speedup for parallel CGNS I/O using PVFS

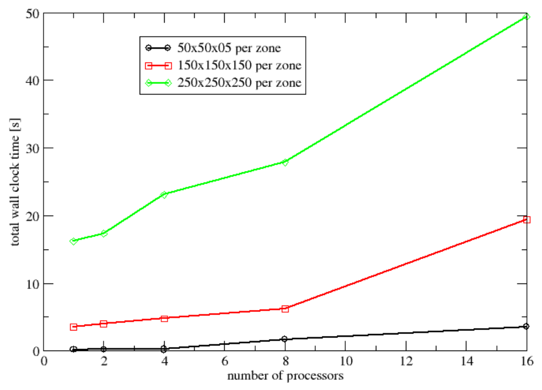


Figure 8. Total wall time for writing the data sets on PVFS2

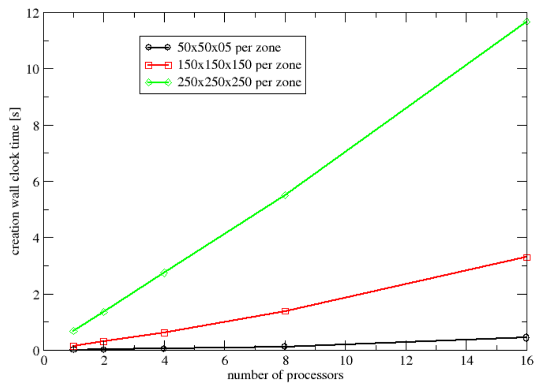


Figure 9. Creation time for creating the data sets on PVFS2

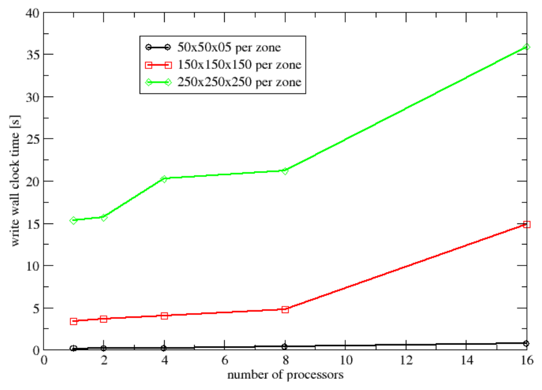


Figure 10. Total wall time for writing into the empty data sets on PVFS2