43rd AIAA Aerospace Sciences Meeting and Exhibit, 10 - 13 Jan 2005, Reno, Nevada

Benchmarking Parallel I/O Performance for Computational Fluid Dynamics Applications

Parimala D. Pakalapati* Thomas Hauser† *Utah State University, Logan, Utah, 84322, USA*

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 super cluster 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[1,2,3]) 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 benchmarks results of the parallel I/O performance of a proposed parallel CGNS extension.[4]

II. The parallel 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. 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.

In our implementation we used the new HDF5 layer This made the implementation of parallel I/O much easier and still maintains the portability with the CGNS standard since the information about the CFD data is described by the SIDS structure.

Our approach introduces a new API with parallel access semantics and optimized parallel I/O implementation such that all processes perform I/O operations cooperatively or collectively through the parallel

*Graduate Student, Department of Computer Science, Utah State University, 4205 Old Main Hill, Logan, UT, 84322

†Assistant Professor, Mechanical & Aerospace Engineering, Utah State University, 4130 Old Main Hill, Logan, UT, 84322, AIAA member.

CGNS library to access a single CGNS file. This approach, as shown in Figure 1, both frees the users from dealing with details of parallel I/O and provides more opportunities for employing various parallel I/O optimizations in order to obtain higher performance.

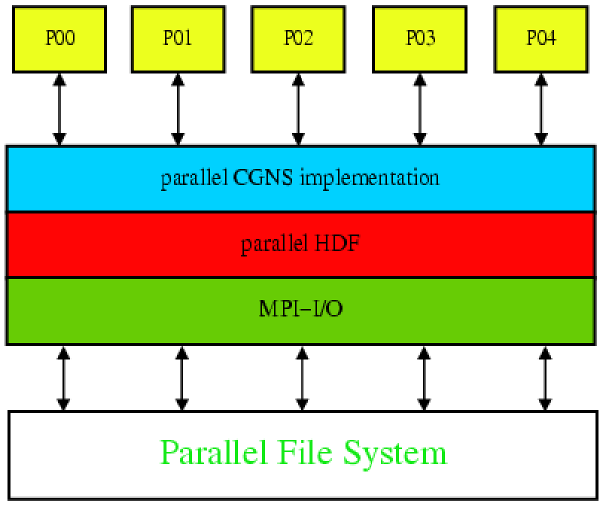

Figure 1. Data access using parallel CGNS API based on HDF5 and MPI-I/O

To facilitate convenient and high-performance parallel access to CGNS files, we define 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.

A. Parallel HDF5

In parallel HDF5 a parallel file is opened with a communicator argument and an access template set for MPI parallel access. It returns a file handle which can be used for future access to the file. All processes of the communicator are required to participate in the collective HDF5 API. Here is a list of the collective HDF5 operations:

- File open H5FCreate or H5Fopen)
- Dataset open (H5Dcreate or H5Dopen)
- Dataset close (H5Dclose)
- File close (H5Fclose)
- Changes to dataset attributes

Data set access can be collective or independent. In our CGNS implementation we use the independent access mode to a collectively opened data sets. This allows the grid blocks on different processor to write their data sets independently in parallel to the file.

B. Parallel Implementation

Several calls were added for the parallel implementation. The most important change is the breakup of the writing of data arrays into two steps. In the serial implementation "GridCoordinates" are written in one function all "cg_coord_write". Since creating structural nodes in HDF5 and therefor CGNS require collective operations, an additional step "cgp_coord_create" which has the same syntax as the serial write but does not write any data and returns a coordinate index number. The only difference is that an empty data set is created collectively. This data set is then filled with the actual data using another call "cgp_coord_write" using the coordinate index number from the creation subroutine call. This write is done independently for each processor and therefor allows for independent parallel I/O.

III. Performance Benchmarks

The number and variety of Linux clusters in use for computational science and engineering is increasing daily, such that it is now difficult to cover the complete design field in a single paper. However, we are uniquely positioned to run simulations on both shared-user superclusters and on a sizable variety of smaller-scale cluster designs. Since we are mainly interested in I/O performance we focus on two different file systems: NFS and PVFS2.

A. FAUST - a low cost Beowulf Cluster

Fluid Athlon Utah State Testbed (FAUST) is a cluster of 64 (plus 2 "hot spare") AMD 2200+ Athlon XP (figure 2). Each PC contains 256MB of main memory and four 100Mb/s Fast Ethernet interfaces. Nine (plus one spare) 32-way Ethernet switches are used in a Flat Neighborhood topology[5,6] to interconnect the machines with low latency and high bandwidth. In addition a fat tree network consisting of one layer of MBit switches and a top level GBit switch is used for administrative and I/O network traffic

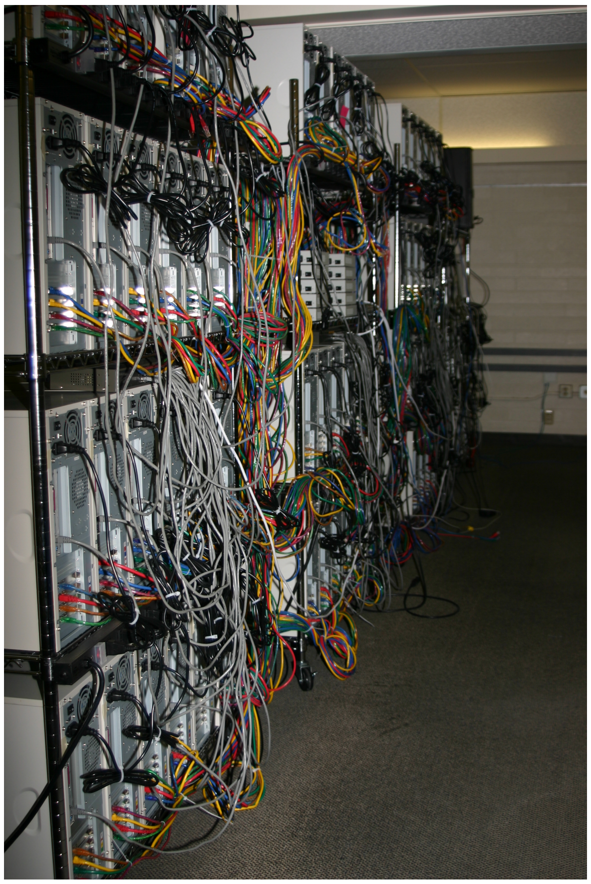

Figure 2. Fluid Athlon Utah State Testbed (FAUST)

B. Network File System

NFS works on a client-server model. One computer is the server and offers file systems to other systems. The clients can mount server exports in a manner almost identical to that used to mount local file systems.

NFS can be tuned using different NSF data transfer buffer sizes. In our case we used the standard Linux Mandrake 10.1 settings.

C. Parallel Virtual File System 2

One area in which commercial parallel machines have always maintained great advantage, however, is that of parallel file systems. A production-quality high-performance parallel file system has not been available for Linux clusters, and without such a file system, Linux clusters cannot be used for large I/O-intensive parallel applications. The Parallel Virtual File System (PVFS),[7] is a parallel file system that can provide high-performance I/O for Linux clusters.

As a parallel file system, the primary goal of PVFS is to provide high-speed access to file data for parallel applications. In addition, PVFS provides a cluster wide consistent name space, enables user-controlled striping of data across disks on different I/O nodes, and allows existing binaries to operate on PVFS files without the need for recompiling.

PVFS2[8] is the second generation file system from the Parallel Virtual File System project team. It improves on the design of the original PVFS[7] to provide parallel and aggregated I/O performance for Linux clusters. It employs a client/server architecture, both the server and client side libraries can reside completely in user space. A file is striped across a number of file servers and the clients communicate with the servers for access to file data. In our configuration meta data accesses is done through the main cluster server. The individual file servers on the nodes use the native file system of the node. Our nodes are configured to be disk less, but since they contain disk space it is used for parallel I/O and scratch storage. More information about PVFS2 can be found in.[8]

On FAUST all compute nodes are I/O servers and clients. The master node is running the manager daemon and is also a client of the parallel file system. The PVFS is used as a temporary file system for applications with parallel I/O enabled and large I/O needs.

IV. Performance Results

A. 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 3 and 4 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.

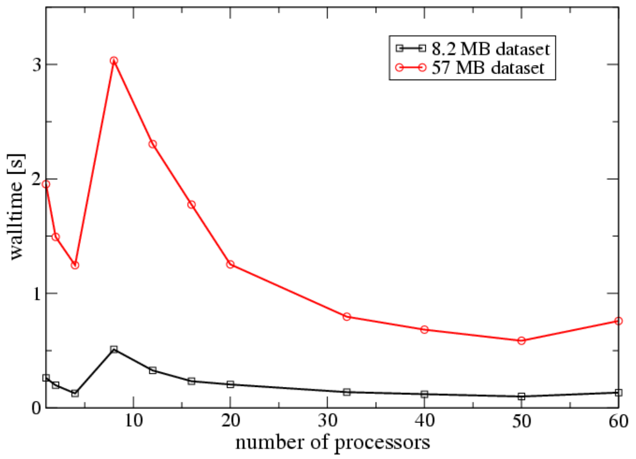

Figure 3. Wall time for parallel CGNS I/O using PVFS for the 8.2 and 57 MB data sets

Figure 5 shows the speedup for reading several data sets. Initially the speedup is increasing as expected,

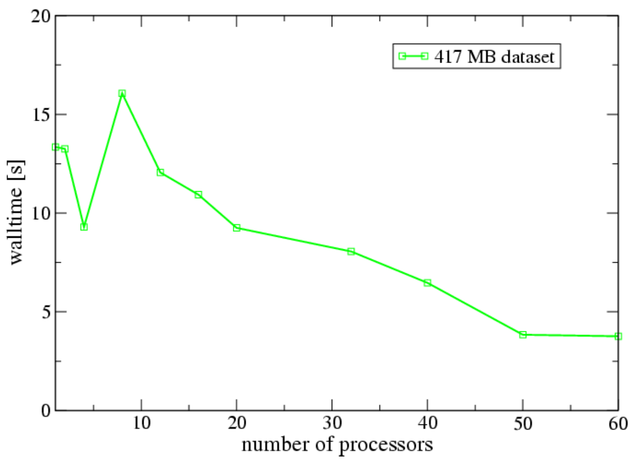

Figure 4. Wall time for parallel CGNS I/O using PVFS for 417 MB data set

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.

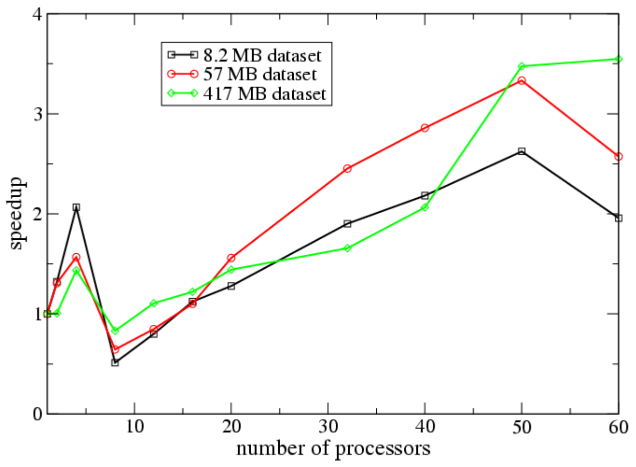

Figure 5. Speedup for parallel CGNS I/O using PVFS

B. Write Performance

The write performance is tested on NSF and 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.

We stopped our benchmarks using 8 processors, because I/O times using NFS were to large. In Figure 6 all three grid sizes show similar behavior. The total I/O time increases with the problem size. This was expected behavior because all processors compete for the disk bandwidth on the FAUST master server.

Comparing Figure 6 with Figure 7 shows the large difference in performance of the parallel file system

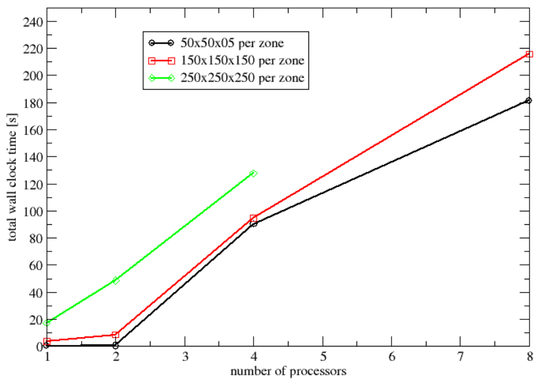

Figure 6. Total wall time for writing the data sets on NFS

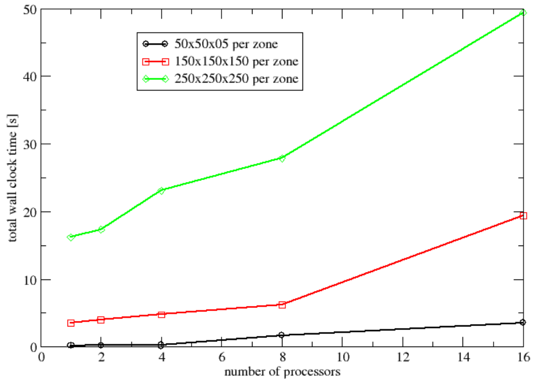

Figure 7. Total wall time for writing the data sets on PVFS2

compared to NFS. The PVFS2 file system performs 35 times faster on the 150x150x150 grid case. 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 8 and 9 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 8. This is the phase of the writing which cannot be parallelized. This time is about 20% of the total I/O time.

In Figure 9 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.

V. 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. In

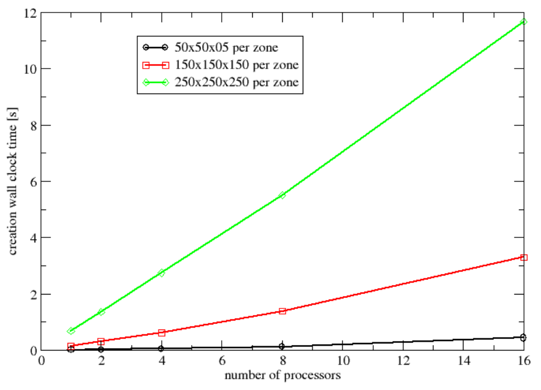

Figure 8. Creation time for creating the data sets on PVFS2

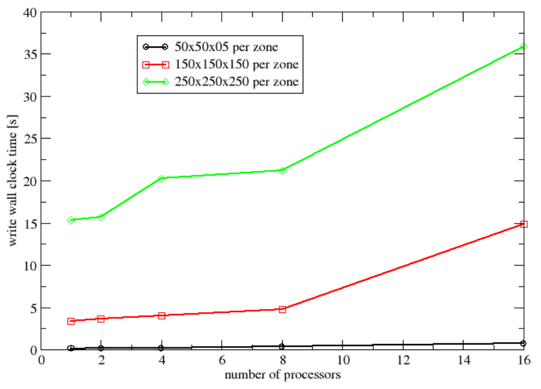

Figure 9. Total wall time for writing into the empty data sets on PVFS2

this work the serial CGNS API was extended with a prototype implementation of parallel I/O for the CGNS system. By building on top of HDF5 the implementation took advantage of the already existing parallel HDF5 implementation on top of MPI-IO. Input and output time was measured and the results demonstrate the superior performance of the parallelized CGNS system.

VI. Future Work

More detailed benchmarking is necessary to identify bottle necks and optimize HDF5 and the underlying parallel file system. With the new upcoming release of CGNS with integrated HDF5 support, we will port and finalize the parallel API. A alpha release is expected by end of April. Support for parallel I/O of unstructured grids needs to be added.

VII. Acknowledgment

This work is supported in part by National Science Foundation grant CTS-0321170.

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.

[5]Dietz, H. G. and Mattox, T. I., "Compiler Techniques for Flat Neighborhood Networks," *to appear in Conference Record of the International Workshop on Programming Languages and Compilers for Parallel Computing, New York*, August 2000.

[6]Dietz, H. G. and Mattox, T. I., "KLAT2's Flat Neighborhood Network," *the Extreme Linux track in the 4th Annual Linux Showcase, Atlanta, GA*, October 2000.

[7]III, W. L. and Ross, R., "PVFS: Parallel Virtual File System," *Beowulf Cluster Computing with Linux*, edited by T. Sterling, MIT Press, November 2001, pp. 391–430.

[8]"The Parallel Virtual File System, Version 2," http://www.pvfs.org/pvfs2.