Using Advanced I/O on Garnet

⊙ From the Scientific Computing Resource Center
Information Technology Laboratory
U.S. Army Engineer Research and Development Center

⊙ This document was created as part of a larger effort to transition the usage model of the shared supercomputing resource Cray XE6, Garnet, toward capability class computing. That process involves changes in the management of the system and changes in the user view of this system's primary function as a scientific instrument.

In particular, this document addresses users who need more advanced I/O behavior from their application in order to scale to larger size jobs. In this context, more advanced means more parallel or more intelligent or both.

The most common reasons for using more advanced parallel I/O in applications on Garnet are:

1. Speed up I/O during execution. For example, a simulation which outputs data for visualization and analysis at regular intervals would benefit from speeding up the write phase.
2. Control filesystem access globally, instead of on a per-core basis. For example, some parallel I/O libraries allow a user to tune how many simultaneous file accesses occur, independent of number of processes.
3. Make code more maintainable. Using parallel I/O libraries instead of writing out I/O logic allows the library implementation to change without affecting the main code.

The Lustre File system, Parallel I/O, & ADIOS

The following sections present an introduction to parallel I/O concepts in the context of high performance computing. In particular, this document focuses on the Cray XE6 system, Garnet, which uses the Lustre filesystem to support fast parallel data access by user jobs.

Contents

1 Utilizing Lustre File system Capabilities

Lustre is heavily used in large-scale cluster computing because of its parallel distributed file system. Its scalability supports numerous compute clusters.

The architecture of the Lustre file system consists of three key components:

1. One or more metadata servers (MDS's)
2. Object storage servers (OSS's)
3. Clients.

These all communicate over a network interconnect. Each MDS contains at least one metadata target (MDT) per Luster file system. Each OSS stores file data on at least one object storage target (OST). Clients are able to access and use the data contained in OSS's, as Lustre enables concurrent read/write access to the files.

The technical specifications of the Lustre file system on Garnet consists of:

- 64 OSTs
- 112.9 TBytes of OST Capacity
- 7.1 Petabytes (PBytes) of Maximum Capacity

1.1 Lustre Concepts

File striping on the Lustre file system can significantly improve I/O performance. Striping allows read/write operations to access multiple OST's concurrently. Therefore, more than one OST can be associated with a file, causing that particular file's data to become striped across the objects.

The capacity and I/O bandwidth scale in relation to the number OST's used to stripe the file. The user has the ability to create differing striping patterns for each file, allowing for individual and custom

tailoring of performance and capacity. The lfs setstripe command is used to set the stripe parameters. If a user does not wish to use the default stripe configurations for the Lustre filesystem, the stripe_size, stripe_count and stripe_offset parameters may be defined. For the Garnet HPC, the default stripe count is 2, and the default stripe size is 1 Megabyte.

To support parallel file reads and writes, a distributed lock manager is used to maintain the integrity of the data contained within the file. The MDT manages the metadata locks while the file data locks are managed by the OST. Multiple reads are allowed on the same file, as well as multiple writes for particular regions of the file. Therefore, if multiple clients are engaged in reading or writing a part of a file, the lock manager allows the clients to view consistent results. The OST can perform multiple I/O operations such as locking, disk allocation, storage and retrieval.

1.2 Optimizing for Performance

In order to have striping capabilities placed on a directory, certain commands must be used. The directory must be set with the striping settings before files are created within it. The striping files that lie within the directory may have their striping settings changed though the cat, cp, sep or tar commands. The files created during program execution under the aforementioned commands will obtain the same striping settings as well. Commands such as mv will not change the original striping settings of the file.

1.3 Lustre File System Commands

The available commands for lfs are:

- setstripe
- getstripe
- find
- check
- catinfo
- join
- osts
- df
- quotachown
- quotacheck
- quotaon
- quotaoff
- setquota
- quota
- help
- exit
- quit

Listed below are the layouts on how to invoke these commands. If additional help is needed, type *lfs help* or *lfs help $command* where *$command* is one of the aforementioned commands.

- Viewing Striping Information:
  $ lfs getstripe <file-name — dir-name>
- Altering Striping Pattern:
  $ lfs setstripe [–size s] [–offset o] [–count c] [pool p] <dir—filename>
  where
  –size—-s *is the stripe size. Type '0' to use the default*
  –offset—-o *identifies the starting OST, Type '1' to use the default (round robin)*
  –count—-c *gives the stripe count, type '0' to use the default*
  -pool—-p *is the name of the OST pool*
- Viewing OST Storage:

$ lfs df [-i] [ih] [path]
reports the file system disk space or in odes usage of each MDS or OSD •
Display MDS/OST Status:
$lfs check <osts — mds — servers>
- Display Info for Specified Type Logs:
$lfs catinfo {keyword} [node name]
- Join two Lustre Files into One:
$join <filename_A> <filename_B>
- Change Files' Owner or Group:
$lfs quotachown [-i] <file system>
-i is the ignore error if the file does not exist •

Turn File system Quotas On/Off:
$lfs quotaoff [-ug] <file system>
$lfs quotaon[-ugf] <file system>

It is also recommended to not create files greater than 200 GBytes for the mass storage archival, as this greatly increases the chance of data corruption. This can result in slow file retrieval and could possibly prevent the storage system from creating a backup of your file. The tape filesystems are better suited for archiving a tar ball that contains many small files instead of many small individual files.

Note: It is not advised that users set an initial file stripe number with the lfs setstripe -i flag. When this flag is *not* used, then the initial stripe is randomized. This way, files will be distributed evenly across OST's. If users set the initial stripe number, then files will tend to cluster up on a particular OST, usually OST 0. Randomization will prevent this.

2 Implementing Parallel I/O

This section provides a survey of parallel I/O in the realm of HPC programming. First, section 2.1 presents several common patterns that applications use for parallel file access. Knowing these patterns will be useful for implementing parallel I/O calls or for understanding I/O options in some tool or library.

Next section 2.2 presents information on libraries and file formats that are commonly used on HPC platforms. Being aware of these implementations and their strengths and weaknesses will help in using them directly or in understanding I/O options in libraries built on top of them.

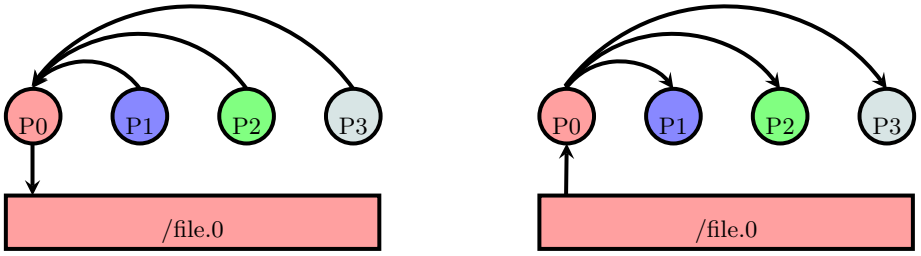
2.1 Parallel I/O Concepts: Per Process I/O, e.g. POSIX I/O

Supposing that some parallel application has been executing, then comes to point where it needs to input or output some set of data. It is possible to design a variety of parallel behaviors using basic I/O commands, such as traditional POSIX I/O library calls – `fopen`, `fseek`, `fwrite`, etc.

The following subsections describe several common I/O patterns used on Garnet and other HPC systems. Each can be implemented with serial I/O calls made from one or more processes in a parallel job. Each method has certain drawbacks, which are listed after a description of the method.

2.1.1 1 Writer, 1 File

The application could have all processes communicate through a single process, which writes all the data to a single file.



Write Read

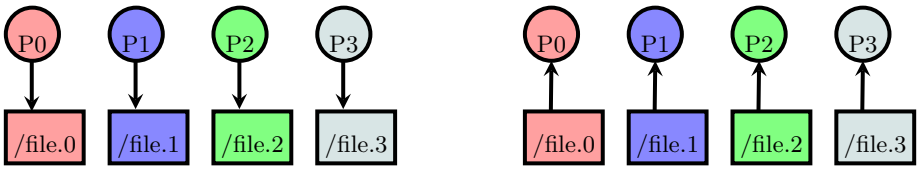
The drawbacks to this method are:

1. The more processes in the application, the more overloaded the network connection to process 0 becomes when all processes send or receive their data to be written or read.
2. Process 0 can only write out data at some maximum rate, limited by a single node's hardware and network connection. Allowing more processes to write simultaneously removes the bottleneck incurred by using only a single node to write.

Due to the fact that the application has a single point of file access, this method is typically the easiest to implement, but scales poorly.

2.1.2 N Writers, N Files

To avoid bottlenecking through a single process, each process could independently access a file which contains only data from that process.



Write Read

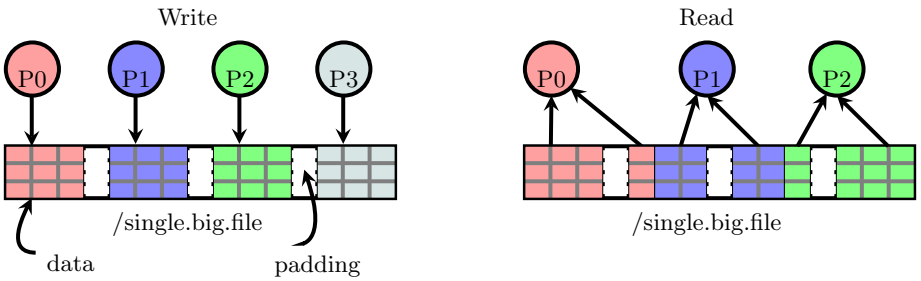
The drawbacks to this method are:

1. As the number of processes increases, the filesystem must handle larger numbers of simultaneous file accesses. This requires large amounts of meta-data and overhead for sharing the physical links to the disk subsystem. These overheads will cause slowdowns in I/O speed if the number of processes is large enough.
2. Managing large numbers of files on disk can be tedious in terms of data archiving and retrieval.
3. If a user varies the number of cores from one job run to the next, reading back data from some set of N files onto $M \neq N$ processes can be complicated, since data from multiple files may need to be stitched back together.

Of the I/O methods in this section, this is probably the most common, due to its simplicity and good performance up to several hundred processes. Above several thousand processes, however, more complex methods become more advantageous.

2.1.3 N Writers, 1 File

To avoid storing many separate files, all processes could read and write from a single file in multiple locations.



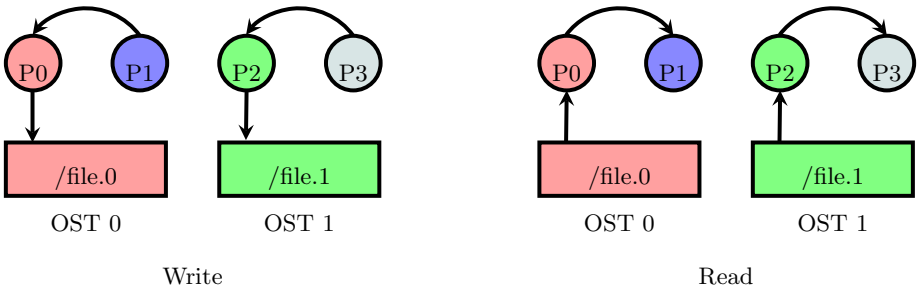
The drawbacks to this method are:

1. Each process must calculate offsets into the file, so that reads and writes from separate processes do not interfere with each other. This leads to more complicated code. Also, padding between files should line up with filesystem data striping, which further complicates offset calculation.
2. The filesystem still has to support many independent file accesses, with the associated problems of meta-data and physical link sharing bottlenecks.
3. If the filesystem does not allow independent locking of separate parts of a single file, then the accesses cannot occur in parallel. The parallel writes will serialize, reducing overall write speed to that of a single writer.

Because of file locking and serialization issues combined with complex logic, this method is typically a poor choice.

2.1.4 N Writers, M Files

To make the most of parallel filesystem capabilities, processes can arrange their data so that an optimal number of files are written simultaneously. In particular, Lustre file systems function well when the number of files is equal to the number of OSTs.



This is a very good method from the systems performance point of view. Unfortunately, from the applications point of view, this method requires complex implementation and upkeep in order to correctly map file accesses to hardware resources. For example, the application must query Lustre to obtain the number of OSTs. That requires using the Lustre API or some other system specific library.

Fortunately, there is a widespread need for this functionality in HPC applications. Over the past several years, libraries have emerged which provide a simple interface and automate the optimization of file access for a given system. Subsequent sections will provide more information.

2.2 Existing Parallel I/O Libraries

This sections lists several freely available I/O libraries that can be used on Garnet. Each subsection provides a short description of the library, along with links to the project home pages and documentation.

2.2.1 ADIOS

- The Adaptable IO System (ADIOS)
  http://www.olcf.ornl.gov/center-projects/adios/
- "The Adaptable IO System (ADIOS) provides a simple, flexible way for scientists to describe the data in their code that may need to be written, read, or processed outside of the running simulation."
  Current version: 1.5.0
  Garnet version: 1.5.0
- ADIOS User's manual
- http://users.nccs.gov/ pnorbert/ADIOS-UsersManual-1.5.0.pdf

ADIOS provides functionality built on top of other libraries and file formats described in this section. Section 3 describes how to use ADIOS. The information in section 2 will be helpful in understanding ADIOS options and configuration.

2.2.2 MPI I/O

MPI-2 added the standard I/O interface known as MPI-I/O. To learn the standard, any number of tutorials exist. The most widely used resource is the text *Using MPI-2: Advanced Features of the Message-Passing Interface*, by William Gropp, Ewing Lusk, and Rajeev Thakur. Examples from the text and errata can be found here: http://www.mcs.anl.gov/research/projects/mpi/usingmpi2/

As for implementations of the standard, Garnet uses the MPICH-2 implementation by default, with MPICH-1.2 and OpenMPI available as alternatives.

- MPICH Home
  http://www.mpich.org/
- "MPICH is a high performance and widely portable implementation of the Message Passing Interface (MPI) standard."
- 2012 Podcast Interview: Rajeev Thakur and Rob Latham, Developers of ROMIO, the MPICH I/O implementation
  http://www.rce-cast.com/Podcast/rce-66-romio-mpi-io.html
- OpenMPI Home
  http://www.open-mpi.org/
- "The Open MPI Project is an open source MPI-2 implementation that is developed and maintained by a consortium of academic, research, and industry partners."
- Video: Scalable and Modular Parallel I/O for Open MPI
  http://www.open-mpi.org/video/?category=internals&watch=Parallel_EdgarGabriel

2.3 Existing Parallel I/O File Formats

Previous sections have focused on the methods accessing files using parallel I/O. As seen in section 2.1, the format of a file or files can impact access methods. For example, recall the case of writing N files from N processes, then reading N files into $M \neq N$ processes.

This section covers several standard file formats that are often used in HPC applications. These file formats provide advanced storage and common functionality between applications.

2.3.1 HDF5

- HDF5 (Hierarchical Data Format)
- http://www.hdfgroup.org/HDF5/
- "HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data."
  Current version: 1.8.11

Garnet version: 1.8.8
- ⊙ Parallel Programming with HDF5
- ⊙ http://www.hdfgroup.org/HDF5/Tutor/pprog.html

2.3.2 XDMF

- ⊙ XDMF (eXtensible Data Model and Format)
- ⊙ http://www.xdmf.org
- ⊙ "The need for a standardized method to exchange scientific data between High Performance Computing codes and tools lead to the development of the eXtensible Data Model and Format (XDMF). Uses for XDMF range from a standard format used by HPC codes to take advantage of widely used visualization programs like ParaView, to a mechanism for performing coupled calculations using multiple, previously stand alone codes."
  Current version: N/A
  Garnet version: N/A See XDMF website for instructions on downloading and using their libraries.
- ⊙ Obtaining XDMF
- ⊙ http://www.xdmf.org/index.php/Get_Xdmf

2.3.3 NetCDF

- ⊙ NetCDF (Network Common Data Form)
- ⊙ http://www.unidata.ucar.edu/software/netcdf/
- ⊙ "NetCDF is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data."
  Current version: 4.3.0
  Garnet version: 4.2.0

3 ADIOS: The Adaptable IO System

ADIOS, short for the Adaptive I/O System, is middleware that can greatly improve parallel I/O performance, simplify parallel I/O coding, and improve code portability. Multiple languages are supported by ADIOS, including C, Fortran, Java, Python, and Matlab scripts.

3.1 Overview of Features

To use ADIOS, a programmer replaces low level I/O library calls with ADIOS calls in their programs. An external XML file is used to select I/O methods and specify which data is written to which files. This allows users to choose the best I/O method for their machine's architecture and change the content of their output files without having to rewrite code or support multiple low-level I/O libraries.

ADIOS makes calls to low level I/O libraries, as shown in Figure 1. The I/O methods supported by ADIOS include POSIX, MPI-IO, DataSpaces, EVPath, and the Lustre API. If the user is writing to a Lustre file system, ADIOS can automatically calculate ideal striping parameters. On systems with Infiniband and Cray Gemini networks, ADIOS can optimize network usage and reduce network contention.For large scale applications (over 10k cores), ADIOS can improve write performance by aggregating data from multiple processes before writing.

ADIOS reads and writes files in the BP file format, designed specifically to be resilient and support delayed consistency for better I/O performance. BP files are meta-data rich and can be converted using utilities that come with ADIOS. File conversion targets include the widely used HDF5 and NetCDF4 formats, as well as human readable ASCII files.

ADIOS I/O methods, optimizations, file formats, and language bindings are all discussed in greater detail in the ADIOS Users Manual [1].

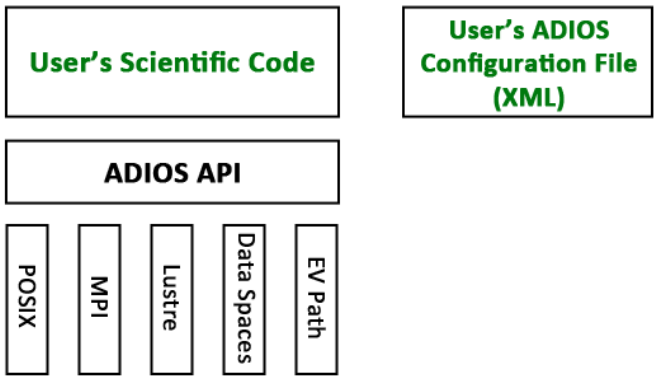


Figure 1: ADIOS is middleware that improves I/O performance and ease of use. It is configured through an XML file without the need for recompilation.

3.2 ADIOS Users and Applications

ADIOS is being actively developed by the Oak Ridge Leadership Computing Facility for the high performance computing community [2]. Users include scientists and engineers at national research centers and institutions of higher learnings such as Oak Ridge National Laboratory, Sandia National Laboratories, the Engineering Research and Development Center, the University of Tennessee, and Georgia Tech, and others.

The first version, ADIOS 1.0, became available in 2009. As users provided feedback, I/O libraries evolved, and new hardware came into use, ADIOS has been revised and updated [3][4]. The current version as of 2013 is ADIOS 1.5 and future versions are forthcoming [5].

Some of the applications that have benefited from ADIOS are:

- **S3D combustion code**
  The S3D code [6] has been used for direct numerical simulations of reacting jets in cross flow. The computation scaled to an entire Cray XT5, but the I/O portion, using MPI, did not. S3D switched to ADIOS 1.0 to improve maintainability [2]. With ADIOS 1.2 they achieved to a factor of 15 speedup in S3D runtime [7]. Ray Grout commented, "I appreciate the clean yet capable interface and that from my perspective, it just works... It would have been fairly difficult for us to get this work finished in time for our targeted paper deadline using our previous I/O solution, but now I'm confident that we'll make it thanks to ADIOS." [8]

- **QLG2Q**
  The OLG2O quantum lattice gas model [9] runs on Cray XT5 and on Cray XE6. It did not scale beyond 20k cores when using MPI and POSIX. After switching to ADIOS 1.3, the code scaled to 110k cores with write and read speeds of roughly 40 GB per second [10].

- **RAMGEN**
  The CFD solver by Numeca International, used by RAMGEN Power Systems at OLCF, was overhauled with many application updates leading to a factor 100 speedup. Testing was done on a two body test case with 500 million grid cells and 3840 processes. In particular, ADIOS improved code readability and reduced problems with initializations and restarts [11].

- **Particle Simulation**
  The LAMMPS atomic/molecular simulator uses an ADIOS XML configuration file to make specification of input and output simpler for users.

- **XGC Particle Simulator**
  The XGC particle simulator was used to model turbulent flow in plasma[12]. The XGC fortran

code originally used HDF5 I/O routines. It was an early adopter and testbed for ADIOS [2] to improve code maintainability. Currently, XGC can checkpoint 200K core computations using ADIOS in under 1 minute, while the original I/O methods take over 1 hour [13]

- **Fusion Simulations**
  GTC was changed eight times to optimize I/O performance. This code was an early adopter and test case for ADIOS due to the ease with which I/O methods could be changed, allowing one method to be used during debugging and a different method to be used in production [14]. Using ADIOS, several leading fusion codes, XGC-1, GTC, and GTS, all scaled to 140K cores on ORNL's Jaguar XE6 [8].
- **Chimera**
  Chimera is an astrophysics code used for supernova simulation [15]. The code changed to ADIOS 1.0 to allow easy switching between I/O methods [14].
- **Geoscience Simulators**
  The M8 earthquake simulator ran on a Cray XT5 using 223k cores. The application creators used ADIOS for ease of I/O configuration and to facilitate large volume data analysis [16]. Other HPC geoscience codes using ADIOS include AWP-ODC[17] and SPECFEM3D[18].
- **Chombo, Adaptive Mesh Refinement**
  Chombo [19] is an adaptive mesh refinement (AMR) code that uses ADIOS for its parallel I/O portion. ADIOS lets users interactively change which data is written to disk without recompiling their code. This is advantageous for AMR codes because their users often need alter what is written to disk [7].

3.3 Getting Started with ADIOS

On Garnet, ADIOS 1.5 is installed, and can be used by loading the appropriate module:

> module load adios

Garnet also contains sample codes in C, Fortran, Matlab, Python, and Java. These can be accessed at:

$SAMPLES_HOME/parallel-io/ADIOS

3.3.1 Modifying a C/MPI Program

The ADIOS User's Manual contains a chapter which shows all of the steps required to add ADIOS to an MPI application written in C.

⊙ ADIOS User's Manual
http://users.nccs.gov/ pnorbert/ADIOS-UsersManual-1.5.0.pdf
See Chapter 12: C Programming with ADIOS

A summary of steps required to add ADIOS to a C program on Garnet are:

1. Load your programming environment module, e.g. ProgEnv-pgi, then load the adios module.
2. Construct an XML file. The ADIOS User's Manual shows an example of syntax. This file will contain information about the programming language, I/O method, variable names, and MPI communicator used to coordinate processes.
3. Use the `gpp.py` utility (included in ADIOS) to generate a .ch file from the XML file. This .ch file includes C code that correlates to the XML file content. It will be included in the main body of the code between calls to `adios_open` and `adios_close`. See Chapter 10 of the ADIOS User's manual for more information on the content of this file.
4. Add ADIOS calls. Typically, `adios_init` and `adios_finalize` are inserted just after `MPI_Init` and just before `MPI_Finalize`, respectively. The init function will include the name of the XML configuration file and MPI communicator used by ADIOS. Then, to read or write data, add calls to `adios_open` and `adios_close` enclosing an `#include` directive referring to the .ch file produced in the previous step.

5. Compile the program and submit your job.

3.3.2 Modifying ADIOS XML Configuration File for Lustre

If you followed the example from the ADIOS User's Manual, you saw XML configurations for POSIX and MPI. To change the I/O method from MPI to MPI optimized for the Lustre file system, simply change the method field in the XML configuration file.

```
<!-- Use regular MPI IO to read/write files -->
<method group="temperature" method="MPI"/>
<method group="temperature" method="MPI"/>
```

As will be shown below, the "MPI" method actually performs and scales poorly. For better performance, the following changes will improve performance on a modest number of cores:

```
<!-- Use MPI optimized for the Lustre file system to read/write files -->
<method group="temperature" method="MPI_LUSTRE"/>
<method group="temperature" method="MPI_LUSTRE"/>
```

Above a few thousand cores, however, that method may show a slowdown. A more advanced method can be configured:

```
<!-- Use smarter MPI optimized for the Lustre file system to read/write files -->
<method group="temperature" method="MPI_AGGR"/>
<method group="temperature" method="MPI_AGGR"/>
```

In some cases, "MPI_AGGR" may show somewhat slower performance than other methods due to overheads for intelligent logic. However, the case study in Section 4 shows best performance at all scales with this method. As shown, it is relatively simple to switch methods to find the best match for your application. Note that in Section 4, the old name "MPI_AMR" is used. As of ADIOS 1.5, this was changed to "MPI_AGGR".

4 Case Study: ADIOS Performance On 10K Cores Of Garnet

A PETTT Pre-Planned Effort (PPE) is in process to run a simulation test case on the new Garnet system using more than 100,000 cores. In preparation for this simulation, several test cases were run using up to 10,000 cores on Garnet before it was taken down for the upgrade. The simulation runs George Vahala's Bose-Einstein Condensate (BEC) code using advanced qubit algorithms. The BEC code is run on a standard 3D Cartesian grid. The resolution of the grid increases with the number of cores, i.e. there are a fixed number of grid points per core. This leads to a constant volume of output data per core with total output data increasing linearly with core count. Essentially it is a weak scaling study for writing to disc.

4.1 Methodology

The initial tests on Garnet focused on identifying the most efficient I/O system for large scale implementation. Five different methods were analyzed, four of which were based on the ADIOS libraries.

1. Smart POSIX (non-ADIOS)
2. ADIOS POSIX
3. ADIOS MPI-IO
4. ADIOS MPI-Lustre
5. ADIOS MPI-AMR (renamed MPI-AGGR in ADIOS 1.5)

ADIOS has been successfully implemented on system such as the DoE's Jaguar for simulations up to 250,000 cores. Additionally, ADIOS allows the user to quickly change I/O methods without significant effort. This allowed the team to test different transport methods quickly.

4.1.1 Building and Installing On Garnet

In order to utilize the ADIOS libraries, they first needed to be installed on Garnet. Version 1.4.1 of the ADIOS system was the current version at the time of installation. It was configured using the directions in the ADIOS User Manual, however there were a few conditions unique to Garnet.

First, even with HDF5 and NetCDF modules loaded, the configure script needed to be manually pointed to the locations for all libraries and include files. Once the locations of the HDF5, NetCDF and Lustre libraries were known, compilation and installation were performed without any errors and took relatively little time.

Two versions of the ADIOS libraries were created, one for the PGI compiler environment and a second for both the Cray and GNU compiler environments. The newest version released in June 2013 of ADIOS is 1.5 and it utilizes the CMake system which may resolve the issues the PETTT team experienced building ADIOS. Once built, all features of ADIOS worked as expected.

4.1.2 Description of Five I/O Methods

Smart POSIX is a standalone library previously developed for use in the BEC code by a member of the PETTT PPE team. It is very similar to traditional POSIX where each MPI task writes its own file. There is some compression of data and better usage of MPI communicators compared to traditional POSIX I/O.

The four methods utilizing the ADIOS libraries were POSIX, MPI-IO, MPI-Lustre and MPI-AMR. ADIOS POSIX is essentially traditional POSIX. One file is generated per MPI task. There is some performance enhancement over traditional POSIX due to the ADIOS buffering system. Similarly, ADIOS MPI-IO is very similar to traditional MPI-IO where one file is generated for all tasks. Again, some performance increase can be expected due to ADIOS buffering.

ADIOS MPI-Lustre is the first method to truly take advantage of the ADIOS system. In this method, the ADOIS libraries interact with the Lustre API in order to take advantage of stripes and strides. It generates one file for all tasks but utilizes both the ADIOS buffering and Lustre striping systems.

The final method is the ADIOS MPI-AMR and is based on the MPI-Lustre system. The main difference here is that one file is written per Lustre OST. ADIOS utilizes a user defined number of cores to aggregate and schedule the data to be written. ADIOS includes metadata in the files so that the multiple files from all of the OSTs have an internal organization. This all occurs as a background task, so the aggregation cores can still be utilized for computation. The name of this method has been changed in the current version of ADIOS to MPI-AGGR to better reflect the advantages of aggregation.

4.2 Throughput Results

Figure 2 summarizes the mean throughput for each of the methods tested by the PETTT team. ADIOS MPI-AMR gives the highest mean throughput for all tests. It scales better than all other methods to 10240 cores. It also has the fastest performance at low core counts, although the ADIOS POSIX is close at 1728. If a code had a low core count and happened to be writing files that were close in size to the Lustre stripe size, it is conceivable that traditional POSIX could achieve similar results. Using the ADIOS libraries, however, the user is always be guaranteed to be using the Lustre file system in an optimum manner.

The benefits of automatic stripe size matching be seen when the ADIOS MPI-Lustre and the ADOIS MPI-IO methods are compared. Both methods use MPI-IO to create one file regardless of the number of MPI tasks. The difference is that the ADIOS MPI-Lustre format tunes writes to the Lustre file system,

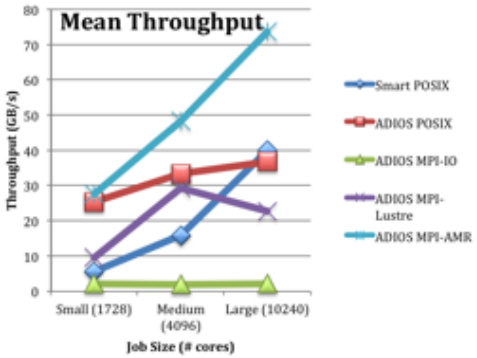


Figure 2: This is a figure.

whereas ADIOS MPI-IO is oblivious to Lustre configuration. ADIOS MPI-Lustre stops scaling after 4096 cores because the OSTs become overloaded trying to write to one large shared file.

4.3 Summary & Future Work

Overall, the PETTT team found that ADIOS gives an increase in I/O performance compared to previous in-house developed methods. As more advanced ADIOS features are used, performance increases. The most advanced method, ADIOS MPI-AMR (now called MPI-AGGR) demonstrates the best scalability to 10240 cores.

These tests will be extended on the new Garnet system to more than 100,000 cores, but its is expected that the MPI-AGGR (MPI-AMR) method will provide the best results.

5 Related External Links

5.1 DoD Open Source Software Policy Information

Free software is work already done. The DoD has had a history of misunderstandings about taking advantage of this vast resource. HPC development in particular does not need to fall victim to the creation of excess work due to misunderstanding.

⊙ DoD Open Source Software (OSS) FAQ

http://dodcio.defense.gov/OpenSourceSoftwareFAQ.aspx

⊙ Clarifying Guidance Regarding Open Source Software, Oct 16, 2009

http://dodcio.defense.gov/Portals/0/Documents/FOSS/2009OSS.pdf

5.2 General HPC Current Events

There is nothing slow about HPC. The news is still new here.

⊙ HPCWire

http://www.hpcwire.com/

Since 1986 - Covering the Fastest Computers in the World and the People Who Run Them

⊙ InsideHPC

http://insidehpc.com/

Founded on December 28, 2006, insideHPC is a blog that distills news and events in the world of HPC and presents them in bite-sized nuggets of helpfulness as a resource for supercomputing professionals.

⊙ RCE Podcast

http://www.rce-cast.com/

Research Computing and Engineering, targets topics relevant to the High Performance Computing (HPC) and Research Computing communities.

References

[1] N. Podhorszki, Q. Liu, J. Logan, H. Abbasi, J. Y. Choi, and S. Klasky. *ADIOS Users Manual 1.5*. Oak Ridge National Laboratory, June 2013.

[2] Chen Jin, Scott Klasky, Stephen Hodson, Weikuan Yu, Jay Lofstead, Hasan Abbasi, Karsten Schwan, Matthew Wolf, W Liao, Alok Choudhary, et al. Adaptive IO System (ADIOS). *Cray Users Group*, 2008.

[3] E. Gedenk. Say goodbye to bottlenecks - adios 1.3 is here. https://www.olcf.ornl.gov/2011/07/19/say-goodbye-to-bottlenecks-adios-1-3-is-here/, 2011. Accessed 2013-07-10.

[4] E. Gedenk. Adios middleware update sets the stage for exascale computing. https://www.olcf.ornl.gov/2012/07/26/adios-middleware-update-sets-the-stage-for-exascale-computing/, 2012. Accessed 2013-07-10.

[5] S. Klasky. Adios 2 why what when where. http://computing.ornl.gov/workshops/JICSGRS2012/presentations/s_klasky.pdf, 2012.

[6] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Snakaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science and Discovery*, 2(1), January 2009.

[7] Gregory Scott Jones. Say hello to the new adios. http://www.hpcwire.com/hpcwire/2010-09-12/say_hello_to_the_new_adios.html, September 12 2010. Accessed 2013-07-10.

[8] Gregory Scott Jones. Adios ignites combustion simulations. http://www.hpcwire.com/hpcwire/2009-10-29/adios_ignites_combustion_simulations.html, October 29 2009. Accessed 2013-07-10.

[9] G. Vahala, J. Yepez ad L. Vahala, M. Soe, B. Zhang, and S. Ziegeler. Poincare recurrence and spectral cascades in 3d quantum turbulence. *Physics Review*, 84(046713), 2011.

[10] Jeff Larkin. Adios. http://www.erdc.hpc.mil/docs/Tips/OptimizingIOPerformanceForLustre.pdf, 2011. Accessed 2013-07-10.

[11] Scott Jones. Ramgen simulates shocks waves, makes shock waves across energy spectrum. http://www.olcf.ornl.gov/2012/08/14/ramgen-simulates-shock-waves-makes-shock-waves-across-energy-spectrum/, 2012. Accessed 2013-07-10.

[12] S. Ku, C. S. Chang, M. Adams, J. Cummings, F. Hinton, D. Keyes, S. Klasky, W. Lee, Z. Lin, S. Parker, and the CPES team. Gyrokinetic particle simulation of neoclassical transport in the pedestal/scrape-off region of a tokamak plasma. *Journal of Physics*, 46(1), 2006.

[13] SDAV Team. SDAV technologies for the next generation fusion techniques. http://computing.ornl.gov/workshops/FallCreek12/posters/Podhorzski-sdav-fusion-poster-FCF.pdf, 2012. Smoky Mountains Computational Sciences and Engineering Conference.

[14] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Adaptable, metadata rich io methods for portable high performance io. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–10. IEEE, 2009.

[15] M. Chertkow, O. Messer, W. Hix, K. Yakunin, P. Marronetti, S. Bruenn, E. Lentz, J. Blondin, and A. Mezzacappa. Advancements in modeling self-consistent core collapse supernovae with chimera. *Journal of Physics*, 402, 2012.

[16] Y. Cui, K. B. Olsen, T. H. Jordan, K. Lee, J. Zhou, P. Small, D. Roten, G. Ely, D. K. Panda, A. Chourasia, J. Levesque, S. M. Day, and P. Maechling. Scalable earthquake simulation on petascale supercomputers. In *ACM/IEEE International Conference for High Performance Computing Networking Storage and Analysis*, 2010.

[17] Awp-odc code and related publications. http://hpgeoc.sdsc.edu/software.html. Accessed: 2013-07-10.

[18] Specfem3d code and related publications. http://www.geodynamics.org/cig/software/specfem3d. Accessed: 2013-07-10.

[19] Chombo code and related publications. https://commons.lbl.gov/display/chombo/. Accessed: 2013-07-10.