Characterizing and Predicting the I/O Performance of HPC Applications Using a Parameterized Synthetic Benchmark

H. Shan, K. Antypas, J. Shalf
CRD/NERSC, Lawrence Berkeley National Laboratory Berkeley, CA 94720
{hshan, kantypas, jshalf@lbl.gov}

Abstract

The unprecedented parallelism of new supercomputing platforms poses tremendous challenges to achieving scalable performance for I/O intensive applications. Performance assessments using traditional I/O system and component benchmarks are difficult to relate back to application I/O requirements. However, the complexity of full applications motivates development of simpler synthetic I/O benchmarks as proxies to the full application. In this paper we examine the I/O requirements of a range of HPC applications and describe how the LLNL IOR synthetic benchmark was chosen as suitable proxy for the diverse workload. We show a procedure for selecting IOR parameters to match the I/O patterns of the selected applications and show it can accurately predict the I/O performance of the full applications. We conclude that IOR is an effective replacement for full-application I/O benchmarks and can bridge the gap of understanding that typically exists between stand-alone benchmarks and the full applications they intend to model.

1 Introduction

The advent of petascale computing is leading to High-End Computing platforms of unprecedented concurrencies. This daunting level of parallelism will pose enormous challenges for future I/O systems that must support efficient and scalable data movement between disks and distributed memories. In order to guide the design of new I/O systems, it is necessary to gain a better understanding of applications' I/O requirements. However, it can be impractical to run full applications for testing and evaluation of emerging I/O solutions, motivating the need for a compact synthetic benchmark capable of modeling the I/O access patterns of a diverse workload. Such a benchmark must not only be able to model application behavior, it must also be able to predict application performance to prove it is a suitable proxy for the full application codes it replaces.

Synthetic I/O benchmarks tend to offer performance metrics that relate directly to system or hardware components of the disk subsystem (eg. random vs. contiguous reads or writes), but are extraordinarily difficult to relate back to application requirements. By contract, full application benchmarks enable head-to-head comparisons of the effective performance delivered to applications, but offer very little insight or diagnostic capability because they do not isolate any specific system or hardware component of the underlying parallel filesystem. This creates a gap in understanding between the hardware-oriented synthetic benchmarks and the information that can be gathered from full applications.

There is a critical need to flesh out the relationship between the I/O requirements of full applications and the parameters of a flexible synthetic benchmark. Moreover, it is important to select an I/O benchmark that is sufficiently parameterized to model the behavior and predict the performance of a wide variety of applications.

This paper helps to close the gap between full application benchmarks and synthetic benchmarks by showing how the IOR benchmark can be used to reproduce I/O access patterns, and even *predict* the performance of key HPC applications. In doing so, we make the following contributions. First, benchmarks are only useful in that they represent a given application workload, so our analysis of I/O requirements is rooted in a survey of a diverse scientific workload from the National Energy Research Supercomputing Center (NERSC). This workload is reflective of the broader unclassified scientific workload in the US Department of Energy Office of Science. Second, we show a procedure for setting the parameters of the IOR synthetic benchmark to accurately reproduce the I/O access patterns of the applications involved in this study. Finally, this work subjects synthetic I/O benchmarks to a higher standard of modeling fidelity than is typically demonstrated by showing the selection of parameters not only matches the I/O behavior of the target applications, but can accurately predict application performance. A synthetic benchmark is often pitched as a faithful proxy to the full applications that it replaces, and therefore it should accurately model, and therefore predict their behavior. A model that does not offer predictive capability is not much of a model.

These topics are addressed in the following order. Section 2 discusses the process of analyzing the workload and describes three applications selected to model the diverse workload requirements. Section 3 describes an array of prior work in synthetic I/O benchmarks and shows how our evaluation criteria led us to focus on the IOR benchmark. After an overview of the parallel file systems architectures examined in this study in section 4, section 5 explains our methodology for choosing IOR parameters to match the I/O patterns of the three representative applications. Section 6 presents the performance results from the IOR to application comparisons showing the analysis of how IOR parameters can be selected to emulate the I/O patterns of the three representative applications. We demonstrate that with suitable parameterizations, IOR is capable of accurately predicting performance and behavior of the original applications on the target platforms enabling next-generation parallel I/O platforms to be evaluated using the much simpler IOR benchmark rather than the full application codes.

2 Workload

Benchmarks are only useful inside the context of the workload they are designed to model. Therefore, this section describes the methodology for identifying the target workload for our benchmarking efforts using data collected from the National Energy Research Supercomputing Center (NERSC) workload. NERSC conducts regular workload assessments to understand current practice and future requirements of I/O systems for the user community. Based on the project descriptions in yearly allocation requests, 50 I/O intensive projects were selected from a field of over

300 requests. For this subset of 50 projects, each Principal Investigator (PI) answered a detailed questionnaire regarding their current I/O practices and future requirements of their applications. Based on the HPC application survey results, we characterized the I/O requirements of our applications into the following parameters: access pattern (random/sequential read/write), the size of each read and write operation, file type (shared: all processes read/write one shared file, or one-file-per-processor: each process reads/writes its own separate file), and programming interface (POSIX, MPI-IO. HDF5, or Parallel-NetCDF).

The major results of this study include:

1. Random access is rare for HPC applications; the I/O access is dominated by sequential operations.
2. Application I/O is dominated by append-only writes.
3. I/O read and write sizes vary widely: from several kilobytes to hundreds of megabytes.
4. The majority of applications have adopted a one-file-per-processor approach to disk-I/O (POSIX I/O for Fortran unformatted I/O) where each process of a parallel application writes to its own separate file rather than using parallel/shared I/O APIs (such as MPI-IO) to write from all of the processors into a single file.
5. Most applications use their own custom file-formats rather than portable self-describing formats such as NetCDF or HDF5. However, interest in these formats is growing as the complexity of data management increases.

Three applications were selected that provide good coverage of the above set of I/O characteristics. They are described in the following section.

2.1 Application Benchmarks

Several I/O benchmarking efforts in the current HPC community focus on developing application specific benchmarks which have been derived directly from their corresponding scientific applications and therefore, reflect realistic I/O demands and communication patterns from those applications. Any optimization insight gained from studying these benchmarks can be fed directly into the original full application, thereby increasing scientific productivity. We have chosen to focus on three of these application specific I/O benchmarks, MADBench2, FLASH3-I/O, and VORPAL-I/O because their I/O patterns are representative of many I/O intensive codes in the NERSC workload and two of them represent the direction many PIs indicated they would like to implement in the future, higher level portable self describing APIs and shared file I/O. Using shared files instead of one-file-per-processor strategy is especially important on the future petaflop-scale platforms since managing the large number of files could pose a great challenge to the metadata management and archival storage regardless of the performance [1]. Using advanced self-describing APIs can help users overcome data portability problems across different architectures. The selected applications represent best practices in use of modern file formats and the preferred path for future application I/O implementations.

Application	Science Area	Algorithm	File Format	Data Type	Purpose
MADBench2	Cosmology	Dense Linear Algebra	Binary	Submatrix	Out-of-core
VORPAL-I/O	Fusion	Particle-In-Cell (PIC)	HDF5	Particle List/3D Grid	Checkpoint/Analysis
FLASH3-I/O	Astrophysics	Explicit Fluid Dynamics	pNetCDF, HDF5, Binary	3D Grid	Checkpoint/Analysis

Table 1. Overview of Three Application-Derived I/O Benchmarks.

2.1.1 MADBench2

MADBench2 [9] is derived directly from the analysis of massive Cosmic Microwave Background (CMB) datasets collected from satellite missions. The CMB is the earliest possible image of the Universe, as it was only 400,000 years after the Big Bang. Due to the large memory requirements of the computational domain, an out-of-core algorithm is used, which includes a number of large matrices to hold the data. A matrix is written to disk once it is calculated and read back when it is required in a demand-driven fashion. Therefore both read and write performances are important for this application.

2.1.2 VORPAL-I/O

VORPAL [16] is a versatile plasma simulation code developed by Tech-X Inc. that enables researchers to simulate complex physical phenomena in less time and at a much lower cost than empirically testing process changes for plasma and vapor deposition processes. The kinetic plasma model incorporated in it is based on the particle-in-cell algorithm both in the electromagnetic and electrostatic limit. The I/O activities are dominated by periodically checkpointing the variables, including three-dimensinal (3D) grids and particle lists where the demanding I/O pattern of the 3D grid storage generally presents the worst performance and is the focus of our study.

2.1.3 FLASH3-I/O

The FLASH [4] code is a modular adaptive mesh code used for simulating compressible reactive flows in astrophysical environments, primarily focused on the deflagration and detonation of type Ia supernovae. The FLASH3-I/O benchmark (which is very similar to a previous version of the benchmark studied extensively in a paper on Parallel-NetCDF [3, 5]), was extracted directly from the latest version of the FLASH code. While various physics applications may be run with the FLASH code, the I/O pattern for these applications is largely the same and consists of writing grid variables for checkpoint/restarting and smaller single precision plotfiles for visualization and analysis.

3 Synthetic Benchmark Selection

3.1 Related Work

In the course of our studies, we have investigated a number of synthetic benchmarks for studying I/O performance on HPC systems and have evaluated these benchmarks based on their ability to cover the parameter space of features demonstrated by application codes in our workload analysis. Most of the existing benchmarks are not reflective of the current I/O practice of HPC applications, either because the access pattern does not correspond to that of the HPC applications, because they only exercise POSIX APIs (eg. no MPI-IO, HDF5, NetCDF), or because they measure only serial performance.

IOZone [8] is a popular file system benchmark used to measure the performance of a variety of file operations. It is designed to understand the file system's I/O characteristics from the standpoint of manufacturer's I/O performance metrics, but is difficult to directly relate its results back with applications. Likewise, HPIO [6] is a synthetic benchmark designed to measure noncontiguous I/O performance, but does not target any specific application. FileBench [10] is a more flexible I/O system benchmark which provides a workload model language to enable the creation of diverse workload benchmarks, however, it lacks the parallel programming interfaces needed by scientific applications, such as MPI-IO, HDF5, or Parallel-NetCDF. Also many access patterns are targeted at transaction processing and online database applications, which are very different from the requriements of HPC applications.

The Effective I/O Benchmark [12] reports an I/O performance number by running a set of predefined configurations to derive a single performance number as its performance metric. However, the predefined composite metric does not cover the full parameter space of I/O patterns demonstrated by the HPC applications we investigated. MPI-Tile-IO [15] tests access to a two dimensional dense dataset. NAS-IO [17] includes the BT I/O benchmark to test the I/O performance of the Block Tridiagonal problem of the NPB. However, these benchmarks focus on a relatively narrow set of target applications. They also focus exclusively on the MPI-IO interface.

The SPIObench [14] and PIORAW [11] are parallel benchmarks that have been used successfully to assist in HPC system evaluations. Both of these benchmarks read and write to separate files in parallel fashion, but do not offer a way to assess performance for concurrent access to a single file. SPIOBench is used in the Department of Defense HPC Modernization Program procurements and includes a useful feature in that it allocates extra memory on each node as to avoid artificially high read/write performance from block buffer caching. We incorporate this idea into our benchmarking methodology as explained in more detail in section 5.1.2.

The common theme with the above synthetic benchmarks is that they are able to measure components of the I/O system and mimic some subset of applications, but have limited ability to model the wide range of application I/O behavior of an HPC workload. Although many of these benchmarks are able to model application behavior to some

degree, few such models are subjected to the more rigorous test of predicting application performance.

3.2 The IOR Synthetic Benchmark

The diversity of an HPC workload argues for a benchmark that is highly parameterized to cover the broad range of application behaviors. After examining a wide variety of publicly available and actively maintained I/O benchmarks we found that Lawrence Livermore National Laboratory's (LLNL) IOR [7] benchmark met most of our requirements for a parameterized benchmark that reflects HPC I/O requirements in the broader HPC workload.

IOR was used successfully to set performance targets for the ASCI Purple system procurement at LLNL. It focuses on measuring I/O performance under different read/write sizes, concurrencies, file formats, and file layout strategies. It supports file format APIs ranging from traditional POSIX interface to advanced parallel-I/O interfaces, including MPI-IO, HDF5, and Parallel-NetCDF. Importantly, it differentiates parallel I/O strategies between the shared file parallel I/O approach and a one-file-per-processor approach. These different options allow head-to-head comparisons of different file formats and storage strategies.

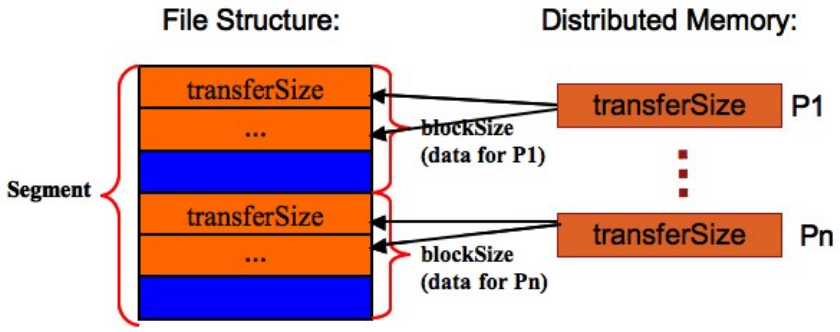


Figure 1. The design of the IOR benchmark for shared file type.

Figure 1 illustrates the relationship between the IOR file structure, processors and memories when writing to a shared file. An IOR file is organized as a sequence of segments that represent the application data for either one simulated time step of a single data variable (eg. pressure for time step1, 2, 3, etc.) or a sequence of data variables (eg. pressure, temperature, velocity). For high-level file formats such as HDF5 and Parallel-NetCDF, each segment directly corresponds to a dataset object in the nomenclature of these respective file formats. Each processor holds an evenly divided part of the segment called a block. The process with rank 0 gets the first block and the process with rank 1 gets the second block and so on. The parallel I/O layer re-assembles each processor's block into a single segment as viewed in an IOR file. Each block is further divided into transfer units which can be useful in emulating strided access patterns. The transfer unit chunks directly correspond to the I/O transaction size, which is the amount of data transferred from the processors memory to disk for each I/O function call (eg. the buffer size for a POSIX I/O call). For the one-file-per-processor case, the file structure is nearly identical to the diagram in Figure 1, except that

each process will write/read data to/from its own file (eg. each block is packed contiguously in separate files).

4 System Overview

Three HPC platforms located at NERSC were selected for comparison in our study. They are, Bassi, an IBM Power5/ Federation cluster running the GPFS file system, Jacquard, an Opteron/Infiniband cluster also running GPFS, and Franklin a Cray XT4 system running Compute Node Linux (CNL) with the Lustre file system. Given the relative size of these systems, it is important to compare the systems on the basis of performance characteristics rather than raw performance. Table 2 shows some of the highlights of these architectures.

Name Machine	Parallel File System	Proc Arch	Interconnect Compute to I/O nodes	Max Node BW to I/O	Measured Node BW	I/O Servers	I/O Disk BW	Total Disk (TB)
Bassi	GPFS	Power5	Federation	8.0	6.1	6	6.4	100
Jacquard	GPFS	Opteron	InfiniBand	2.0	1.2	16	6.4	30
Franklin	Lustre	Opteron	SeaStar-2	6.4	1.2	20	25	350

Table 2. Highlights of architectures and file systems for examined platforms. All bandwidths (BW) are in GB/s. The "Measured Node BW" uses MPI benchmarks to exercise the fabric between nodes.

4.1 Experimental Testbed

The demands of coordinating parallel access to file systems from massively parallel clusters has led to a more complex system architecture than the traditional client-server model of NFS. HPC-oriented Storage Area Networks (SANs) such as GPFS and Lustre, address some of the extreme requirements of modern HPC applications using a hierarchical multi-component architecture. Although the nomenclature for the components of these cluster file systems differs, the elements of each architecture are quite similar. In general, the requests for I/O from the compute nodes of the cluster system are aggregated by a smaller set of nodes that act as dedicated I/O servers. In terms of file system metadata (i.e. file and directory creation, destruction, open/close, status and permissions), Lustre and GPFS operations are segregated from the bulk I/O requests and handled by nodes that are assigned as dedicated metadata servers (MDS). The MDS processes file operations such as open/close can be distinct from the I/O servers that handle the bulk read/write requests.

The bulk I/O servers are referred to as Object Storage Servers (OSS) in Lustre nomenclature whereas they are Network Shared Disk (NSD) in GPFS nomenclature. The back-end storage devices for Lustre are referred to as Object Storage Targets (OSTs) whereas GPFS relies on either locally attached storage or a SAN for the back-end storage devices. The front-end of the I/O servers (both bulk I/O and metadata) are usually connected to the compute nodes via the same interconnection fabric that is used for message passing between the compute nodes. The back-end of the I/O

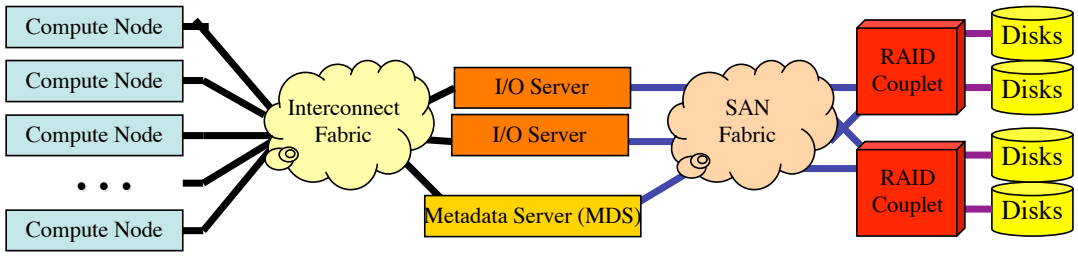


Figure 2. The general architecture of cluster file systems for Lustre and GPFS

servers connect to the disk subsystem via a different fabric, often FC, Ethernet or Infiniband, but occasionally locally attached storage that are contained within the I/O server.

4.2 Bassi: GPFS on Power5/Federation

The 122-node Power5-based Bassi employs GPFS as the global file system. Each node consists of 8-way 1.9 GHz Power5 processors, interconnected via the IBM HPS Federation switch at 4 GB/s peak (per node) bandwidth. The experiments conducted for our study were run under AIX 5.3 with GPFS v2.3.0.18.

Bassi has 6 GPFS I/O servers (Virtual Storage Devices: VSDs), each providing sixteen 2 Gb/s FC links. The disk subsystem consists of 24 IBM DS4300 storage systems, each with forty-two 146 GB drives configured as 8 RAID-5 (4data+1parity) arrays, with 2 hot spares per DS4300. Bassi's maximum theoretical I/O bandwidth is 6.4 GB/s.

4.3 Jacquard: GPFS on Opteron/Infiniband

The Jacquard system contains 320 dual-socket (single-core) Opteron nodes running Linux 2.6.5. Each node contains two 2.2 GHz Opteron processors interconnected via InfiniBand (IB) fabric in a 2-layer fat-tree configuration, where IB 4x and 12x are used for the leaves and spines (respectively) of the switch. Jacquard uses a GPFS-based parallel file system that employs 16 (out of a total of 20) I/O servers for data storage connected to the compute nodes via the IB fabric. Each I/O node has one dual-port 2 Gb/s QLogic FC card, which can deliver 400 MB/s, hooked to a S2A8500 DDN controller.

The DDNs have a theoretical peak aggregate performance of 12 GB/s, while the compute nodes have a theoretical peak bandwidth of 6.4 GB/s. However, peak bandwidth is limited since the IP over IB implementation sustains only $\sim$200-270 MB/s, thereby limiting the aggregate peak to $\sim$4.5 GB/s. The maximum measured aggregate I/O bandwidth on Jacquard is 4.2 GB/s (reads and writes).

4.4 Franklin: Lustre on Opteron/XT4

Franklin, a 9,660 compute node Cray XT4 supercomputer, uses the Lustre parallel file system. Each XT4 node contains a dual-core 2.6 GHz AMD Opteron processor, tightly-integrated to the XT4 interconnect via a Cray SeaStar-2 ASIC through a 6.4 GB/s bidirectional HyperTransport interface. All the SeaStar routing chips are interconnected in a 3D torus topology, where each node has a direct link to its six nearest neighbors. The XT4 uses Compute Node Linux (CNL) on compute processing elements (PEs) and SuSE SLES 9.0 on service PEs.

Franklin has 20 Object Storage Servers (OSSs), and one MDS, connected via the custom SeaStar-2 network in a toroidal configuration to the compute nodes. The OSSs implement a total of 80 OSTs that use Data Direct Networks (DDN) 9500 RAID couplets as backend block devices for the OSTs. With 80 OSTs (8 per DDN couplet) and ten approximately 2.5 GB/s (3 GB/s peak) DDN9500 couplets, providing a theoretical 20 GB/s aggregate *peak* I/O rate.

5 Data Collection and Analysis

5.1 Methodology

5.1.1 Selecting IOR parameters

In order to accurately model an application's I/O behavior a number of the synthetic benchmark parameters must be aligned with those of the application. At the highest level, we must select the file API, whether it be HDF5, Parallel-NetCDF, MPI-IO or raw binary. In IOR, this corresponds to the *API* parameter. The next parameter to consider is an application's parallel file strategy, that is whether an application uses one-file-per-processor or performs parallel I/O to a single shared file and this is controlled by the IOR parameter *filePerProc*. The direction of the I/O traffic is controlled by the IOR parameters *ReadFile* and *WriteFile* and the concurrency is selected using the *NumTasks* parameter. NumTasks needs to be chosen so as to ensure the back-end I/O subsystem is fully or close to saturation, which we measured empirically for each of the platforms.

Figure 3 shows the aggregate I/O performance on these three platforms under different concurrencies. We can find that the best performance is often achieved at a relatively low concurrency, so it is important not to assess parallel I/O scalability in terms of parallel speedup that exactly matches the speedup in FLOPs. In this case, we chose 64 processors because it is the minimum that enables the I/O subsystems to reach their full capability (or very close to full capability). *We have begun to collect data for a broader range of concurrencies, and can expand the paper accordingly if it is accepted.*

Moving to finer-grained IOR parameters, one of the most critical performance parameters is to control the amount of data each processor reads and writes in a single transaction. In IOR, the relevant parameter is called the *BlockSize* and represents the amount of data each processor will write/read from a global segment. In an application however, a block

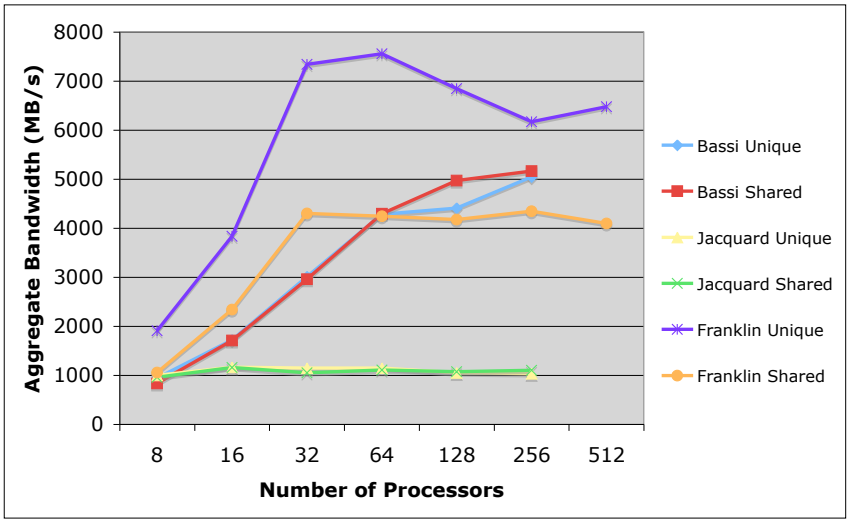


Figure 3. The I/O scalability of the evaluated platforms using both shared and one-file-per-processor data layout strategies.

of data to be written/read by a processor is often not transferred to disk in a single big chunk. Instead, smaller chunks may be transferred multiple times to copy the entire *BlockSize* to the data file. The corresponding IOR parameter is called *TransferSize* and is the I/O transaction size used to transfer data from memory to the file. Finally, an application can write/read more than one data segment to a file, for example, an application could write multiple grid variables separately. The corresponding IOR parameter is called the *SegmentCount* which controls the number of data segments in a file. Table 3 gives an overview of the important IOR parameters for our study. Combined together, we can tailor these parameters to create I/O patterns that closely match the data access patterns of our application benchmarks.

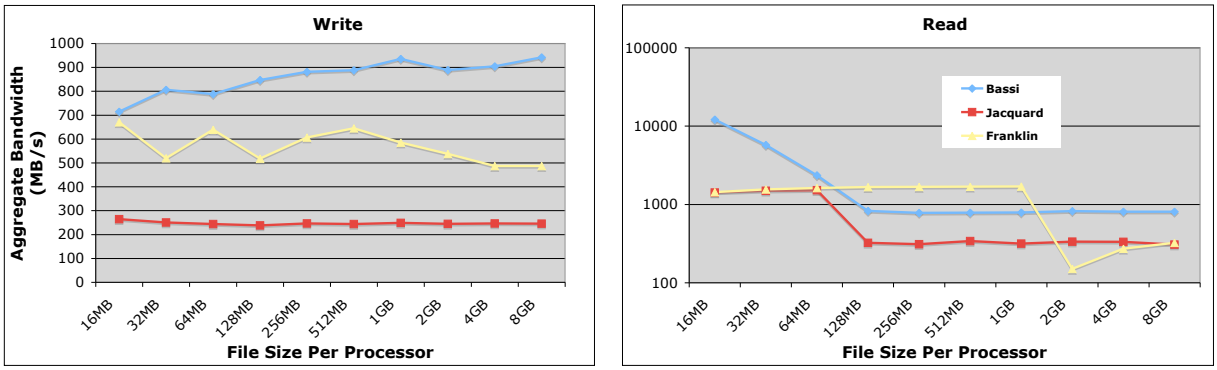
IOR Parameter	Description	Parameter Range
API	File format API	HDF5, pNetCDF, MPI-IO, POSIX
FilePerProc	One-file-per-proc or shared file	True or False
ReadFile	Read a file from disk	True or False
WriteFile	Write a file to disk	True or False
NumTasks	Num tasks participating in test	System limited
BlockSize	Bytes to write per task	Given in KB, MB, or GB
TransferSize	I/O transaction size per task	Divisible by BlockSize
SegmentCount	Number of datasets in file	File system limited

Table 3. Some important IOR parameters

5.1.2 Identifying and Eliminating Caching Effects

When measuring the I/O performance, file caching can result in anomalously high measured I/O rates if the data is being buffered in memory rather than hitting the disk. A typical cause of caching in Unix is the system's ability to

use any unoccupied memory on a compute node to buffer I/O transactions and flush them to disk gradually in order to improve apparent I/O performance. This is knowns as block-buffer caching. While many HPC applications will benefit from caching, seeing increased I/O performance, intensive I/O applications, those in which I/O is a potential bottleneck, typically operate above the caching threshold. Therefore an I/O benchmark must exhaust memory buffers to ensure that the performance of the underlying disk subsystem is actually being measured. To avoid the caching effect, I/O benchmarks can allocate large amounts of memory available on the node, depleting the available memory for caching or create files large enough to the point where caching becomes insignificant. We employ the later technique to determine the minimum file size per processor which must be used to eliminate the influence of caching effects.



(a) (b)

Figure 4. The I/O performance measured on a node using IOR for different file sizes under POSIX interface, one file per processor mode (8 processors on IBM Power5 and 2 processors on other systems).

On the IBM Power5, the memory size is 32GB/node, 4GB/processor. On the Franklin the memory size on a node is 4GB, 2GB per computational core and on Jacquard, the memory size per processor is 3GB. Fig. 4 shows the measured aggregate I/O bandwidth for a node on these three systems using one-file-per-processor strategy for different file sizes. When the file size is small, file caching has a considerable effect on on read performance and we can clearly see the two performance regions on all three platforms for read performance. For example, for the read case on Bassi, when the file size is 16MB, the data is clearly being buffered in memory where the read performance corresponds to the memory read performance, which is around12GB/s. With the increase of file size, the memory cache can no longer hold all the data and the read operation must get the data from the disks. The read performance degrades and gradually becomes stable when all data access is from disks. However, we see no clear caching effect on write.

The results from these three platforms show that caching effects are platform dependent and can have a significant effect on read performance. On Bassi and Jacquard, a file size of 256 MB per processors appears to be a suitable size to overcome the caching effect, on Franklin, it needs to be over 1GB.

In the following sections we will describe the logical and physical I/O patterns of the selected applications and how

the IOR parameter values are derived in our test. Table 4 shows an overview of each application's corresponding IOR parameters.

IOR Parameter	MADBench2	VORPAL-I/O	FLASH3-I/O
API	POSIX	HDF5	HDF5
FilePerProc	True/False	False	False
ReadFile	True	False	False
WriteFile	True	True	True
NumTasks	64	64	64
BlockSize	2400MB	160MB	64MB
TransferSize	300MB	160MB	1MB
SegmentCount	1	1	20

Table 4. Matching application IOR parameters

5.2 MADBench2

5.2.1 Logical and Physical I/O pattern

The analysis of CMB data requires enormous matrix algebra operations. Due to the large memory requirements of the computational domain in real calculations, all the required matrices generally do not fit in memory simultaneously. Thus, an out-of-core algorithm is used, with the individual component matrices being written to disk when they are first calculated, and re-read whenever they are required. MADBench2 includes four major steps:

- Recursively build a sequence of Legendre polynomial based CMB signal pixel-pixel correlation component matrices, writing each to disk (loop over {calculate, write});
- Form and invert the full CMB signal+noise correlation matrix (calculate/communicate);
- In turn, read each CMB signal correlation component matrix from disk, multiply it by the inverse CMB data correlation matrix, and write the resulting matrix to disk (loop over {read, calculate/communicate, write});
- In turn, read each pair of these result matrices from disk and calculate the trace of their product (loop over {read, calculate/communicate}).

The parameters that are closely related with I/O are:

1. The dimensions of a matrix, *nPixel*. All the matrices have the size of *nPixel*nPixel*. Each matrix element is a double float variable.

2. The number of matrices, *nBin*. There are total *nBin* matrices that are evenly divided among all the participated processors for our tests.

In code steps 1, 3 & 4 each processor will issue a sequence of *nBin* read and/or write calls, each of O(8* $nPixel^2$/P) bytes (P is the number of processors). At the top level each of these I/O calls is issued by every processor at the same

time. The main I/O characteristic of this code is that all the processes read/write their local subsection of the dataset using large I/O buffer concurrently.

5.2.2 IOR Parameters for MADBench2

MADBench2 is implemented using both the POSIX interface and the MPI-IO interface. Since the MPI-IO interface delivers very similar performance to the POSIX interface [2, 13], in our test, the IOR *API* parameter is set to POSIX. MADBench2 also allow users to select one file per processor approach or shared file one. In order to compare the results under both cases, the IOR parameter *FilePerProc* is correspondingly set to true and false. MADBench2 uses a weak-scaling strategy, therefore, the data set size on a processor keeps constant. For the test case, the *nPixel* is chosen as $6250*\sqrt{P}$ and *nBin* as 8, so the memory buffer size for a matrix on each processor is *nPixel* * *nPixel* * sizeof(double) / P, corresponding to the IOR parameter *TransferSize*=300MB, *BlockSize*= *TransferSize**nBin = 2400MB, and *SegmentCount*=1. Since MADBench2 uses an out-of-core algorithm, each matrix has to be written to disk when it is first calculated, and read back whenever it is required, both read and write operations are needed. Therefore, both *ReadFile* and *WriteFile* parameters are set as true.

5.3 VORPAL-I/O

5.3.1 Logical and Physical I/O pattern

The VORPAL-I/O stand-alone benchmark was derived directly from the I/O kernel employed by VORPAL to write out its 3D grids. For the convenience of data analysis and so that a simulation can be restarted on a different number of processors, the entire grid is written to the output file in a single-processor view. This means however, that the data belonging to an individual processor (a sub-3D grid stored continuous in memory) may not be contiguous in the output file and could be scattered in small chunks. Programming in POSIX interface is not convenient due to the complex offset computation and instead the code uses the HDF5 interface. However, because the data is not written in contiguous chunks using the default storage layout, the original performance of HDF5 was quite low. For example, using 64 processors on Bassi, the IBM Power5 platform, the total aggregate output bandwidth is only around 60MB/s, around 1% of the peak I/O bandwidth. Fortunately, the HDF5 API offers a number of tunable parameters through the H5property interface that enables exploration of alternative storage layouts. We optimized the VORPAL code to use a *chunked* storage layout where a dataset is partitioned into fixed-size multi-dimensional chunks and stored separately. Using the chunked layout, the performance of HDF5 is improved by more than ten times. Therefore, this storage layout has been incorporated into the application code and is the basis for our comparison on the evaluated platforms. Chunking is a popular approach to improving storage performance for a number of applications on HPC systems.

5.3.2 IOR Parameters for VORPAL-I/O

For a production run using 64 processors, a 3D computational grid is used with a total size of 1500*300*300 zones, which is partitioned by a 4*4*4 processor grid so that each processor receives an equal section of a sub-grid with 375*75*75 zones. Although the VORPAL-I/O benchmark can be configured to use a number of I/O strategies, we chose to evaluate the HDF5 file format to a shared file using the chunked I/O layout. Whereas the non-chunked layout would result in a strided access pattern as each processor reverses the domain-decomposition while it writes into the dataset, for chunked layout the subgrid from each processor is packed contiguously in neighboring sections of the file. Each grid point holds several variables, such as velocity and charge density, such that each grid point needs 10 double precision data elements to store its physical variables. Thus, the total dataset size is around 10.3GB, composed of 64 160MB domains. The *IOR BlockSize* from each processor is 160MB, defined by the subdomain size (375*75*75 * 10 double precision physical variables). Since we are using the chunked layout, the *IOR TransferSize* is the same as the block size since the chunks written by each processor are contiguous and can be written in a single transaction. The *IOR SegmentCount* is 1, but would be 10 were we to use a storage layout that stored the data arrays as separate HDF5 datasets. This application is dominated by the write performance, so we used the *IOR WriteFile* test for our performance evaulation.

5.4 FLASH3-I/O

5.4.1 Logical and Physical I/O pattern

Although the FLASH code can be used to study different science applications, the code's infrastructure is designed so that all applications share a common I/O pattern that involves outputting grid variables for checkpoint/restarting and smaller single precision plotfiles for visualization and anaylsis. The FLASH code is scalable to thousands of processors and is typically configured to use as much memory for a given node or processor as possible. In this sense, FLASH is typically used in a weak scaling manner such that the problem size increases proportionally with the number of processors. This is also true for the FLASH3-I/O. Although the specific I/O size for a given simulation will vary, a typical production run will output anywhere from 500MB-1.5GB of data per processor for a single checkpoint output. FLASH is an intensive I/O application not only because of the relatively large file sizes, but also due to the frequency of output. In a given simulation anywhere from 10-20% of wallclock time is spent doing I/O.

FLASH is capable of using HDF5, Parallel-NetCDF or a one file per processor binary fortran I/O. FLASH can operate in an adaptive mesh or uniform grid mode, but the I/O pattern for each is the same because in an adaptive mesh mode, smaller blocks of data are copied into a large contiguous buffer before writing, mimicing the uniform grid strategy for I/O. The FLASH3-I/O application comparison test was performed on 64 processors on a uniform grid of size 200x200x200 for each processor with 20 double precision variables making the output size 200*200*200 * 20

variables * 8 bytes = 1.28 GB per processor and 81.92 GB for all 64 processors.

5.4.2 IOR Parameters for FLASH3-I/O

For this specific application comparison to IOR the FLASH3-I/O test used the most common FLASH file format API, HDF5, to compare writing a checkpoint file to a shared file in double precision format. The corresponding *IOR API* parameter is HDF5 and the *FilePerProc* parameter is set to false. FLASH is I/O write intensive and does little reading and so the IOR parameter *ReadFile* is set to false and *WriteFile* is set to true. FLASH3-I/O uses 4 dimensional HDF5 datasets to store the grid variables, setup as x zones, y zones, z zones and number of blocks (which can be thought of in the uniform grid case as just the number of processors.) Each processor's grid is output to a contiguous section of an HDF5 dataset so that strided access is not necessary. Each variable has its own data set, so in this example, each processor writes out 64MB chunks of data 20 times corresponding to an *IOR BlockSize*=64MB and a *SegmentCount*=20. The *TransferSize* can be set within the HDF5 interface in the FLASH code and IOR and is set to 1MB. For the runs on Bassi, the file system hint IBM_largeblock_io was set for both FLASH and IOR.

6 Results and Discussion

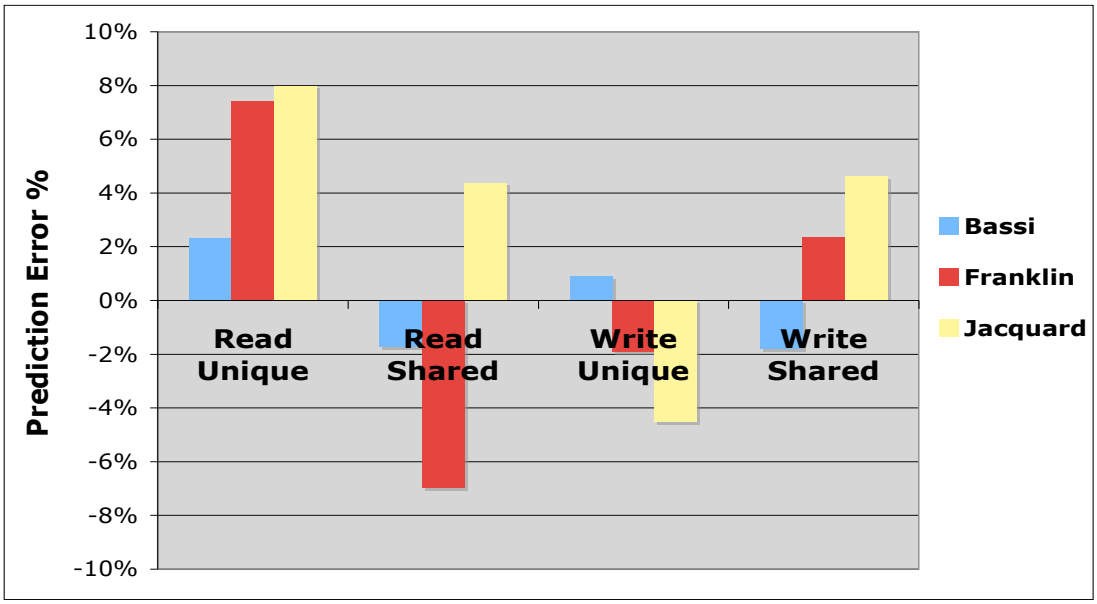


Figure 5. The performance ratio between IOR and MADBench2 on three platforms using 64 processors. (Positive values indicate IOR over-predicted MADBench2's performance while negative values indicate MADBench2 performed better than IOR predicted.)

All of the runs were performed during production mode (non-dedicated with other users on the system) which can increase variation in performance results. To correct for this effect, a minimum of five runs were performed for each

test with the maximum performance rate used for the application to IOR comparison. Fig. 5 shows the results of using IOR to predict read/write performance for MADBench2 on 64 processors using both one-file-per-processor (unique) approach and shared file approach. Overall, the prediction results match quite well within an error bound of 8%. On Bassi, the prediction error is within a 2% margin of error.

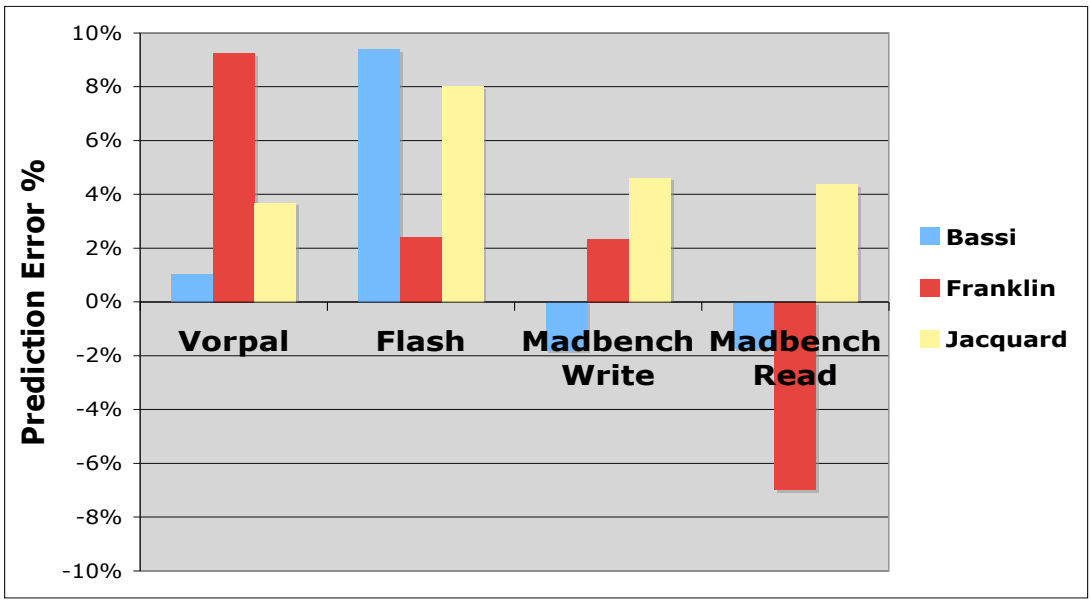


Figure 6. The performance comparison ratio of IOR to VORPAL, FLASH3-I/O, and MADBench2 (shared) on three platforms using 64 processors. (Positive values indicate IOR over-predicted an application's performance while negative values indicate an application performed better than IOR predicted.)

Fig. 6 displays the performance comparison results for all three applications on 64 processors and shows that in all cases IOR is accurate in predicting application performance to within 10% and in some cases much better.

The VORPAL-I/O results match quite accurately, particularly on Bassi where IOR and VORPAL-I/O deliver nearly identical aggregate bandwidth. Even in the worst case, the prediction error is within 9% on Franklin. We point out that the results on Franklin are collected with the file striped across all 80 OSTs. Using the default configuration (file striped only across 4 OSTs), the IOR and VORPAL-I/O results are much closer, with $< 1\%$ prediction error. This indicates that the performance difference is primarily caused by the striping implementation of the Lustre storage servers or contention among OSTs.

For the FLASH3-I/O runs, on all three machines FLASH3-I/O had lower performance than IOR predicted. The major difference in data layout between FLASH3-I/O and IOR is the dimensionality of the datasets written in HDF5. Although the same total amount of data is written, FLASH3-I/O uses 4 dimensional datasets whereas IOR only has the ability to write one dimensional datasets. We suspect this could be the cause of some of the performance difference, although with error rates of 10% and less, we are pleased the difference in dataset dimensionality did not cause greater

performance variation.

We consider the maximum 10% prediction error impressively accurate given the relative simplicity of IOR compared to the applications. However, we intend to examine the factors contributing to the prediction error to identify other performance parameters that are not currently included in our model. IOR could be modified to capture these newly identified effects to further improve the fidelity of its predictions.

7 Conclusions and Future Work

In this work, we analyzed a diverse array of applications to gain an understanding of typical I/O strategies employed by codes on HPC systems. The workload analysis led to the selection of IOR as our synthetic benchmark to represent the requirements of a typical HPC workload. We then attempted to configure IOR's many parameters to match three different applications' I/O patterns and then studied how closely IOR could represent and predict the I/O pattern and performance of the applications. We found for the three applications selected, IOR can be used to predict application behavior to within 10%. Although IOR parameters must be closely selected to match an application's I/O pattern, we believe our results make a strong case for using IOR as a synthetic I/O benchmark in system testing and procurements to predict the performance of full applications. The ability to predict application performance using appropriate selection of benchmark parameters is a much *stronger* test of benchmark fidelity than is typically required of I/O microbenchmarks.

In this paper we have primarily explored IOR's ability to predict applications with large block I/O patterns, which make up a substantial fraction of I/O intensive codes in the NERSC workload. However, there are a class of I/O intensive applications which write data in small bursts and we are interested in exploring IOR's ability to predict this type of application I/O behavior. Additionally, for simplicity we have focused the application comparison at a concurrency of 64, which was chosen to saturate back end I/O subsystems. We have collected preliminary data at higher concurrencies, and will expand this study in the final version of this paper to include a broader array of concurrencies. Future work will also study a broader range of applications.

8 Acknowledgments

The software used in this work was in part developed by the DOE-supported ASC Alliance Center for Astrophysical Thermonuclear Flashes at the University of Chicago. We also thank John Cary of TechX for providing access to the VORPAL source code for this study. All LBNL authors were supported by the Office of Advanced Scientific Computing Research in the Department of Energy Office of Science under contract number DE-AC02-05CH11231.

References

[1] K. Antypas, A.C.Calder, A. Dubey, R. Fisher, M.K. Ganapathy, J.B. Gallagher, L.B. Reid, K. Reid, K. Riley, D. Sheeler, and N. Taylor. Scientific applications on the massively parallel BG/L machines. http://ww1.ucmss.com/books/LFS/CSREA2006/PDP4125.pdf.

[2] J. Borrill, L. Oliker, J. Shalf, and H. Shan. Investigation of leading HPC I/O performance using a scientific-application derived benchmark. In *Proc. SC2007: High performance computing, networking, and storage conference*, 2007.

[3] A. Ching, A. Choudhary, W. Liao, R. Ross, and W. Gropp. Efficient structured data access in parallel file systems. In *Cluster 2003 Conference*, Dec 4, 2003.

[4] FLASH3 code. http://flash.uchicago.edu/.

[5] FLASH2 io-benchmark. http://www.astro.sunysb.edu/mzingale/software/.

[6] High-performance I/O. http://cholera.ece.northwestern.edu/~aching/research_webpage/hpio.html.

[7] The ASCI I/O stress benchmark. https://computing.llnl.gov/?set=code&page=sio_downloads.

[8] Iozone filesystem benchmark. http://www.iozone.org.

[9] MADBench2. https://crd.lbl.gov/~borrill/MADbench2/.

[10] R. McDougall and J. Mauro. FileBench. http://www.solarisinternals.com/si/tools/filebench.

[11] The PIORAW Test. http://www.nersc.gov/nusers/systems/bassi/code_profiles.php.

[12] R. Rabenseifner and A. E. Koniges. Effective file-I/O bandwidth benchmark. In *European Conference on Parallel Processing (Euro-Par)*, Aug 29–Sep 1, 2000.

[13] H. Shan and J. Shalf. Using IOR to analyze the I/O performance of HPC platforms. In *Cray Users Group Meeting (CUG) 2007*, Seattle, Washington, May 7-10, 2007.

[14] SPIOBENCH: Streaming Parallel I/O Benchmark. http://www.nsf.gov/pubs/2006/nsf0605/spiobench.tar.gz, 2005.

[15] MPI-Tile-I/O. http://www-unix.mcs.anl.gov/pio-benchmark/.

[16] VORPAL, versatile plasma simulation code. http://www.txcorp.com/products/VORPAL/.

[17] P. Wong and R. F. Wijngaart. NAS parallel benchmarks I/O version 2.4. In *Technical Report NAS-03-002, Computer Sciences Corporation*, NASA Arms Research Center, Moffett Field, CA 94035-1000, Jan, 2003.