# IOR Benchmark Instructions

## Introduction

IOR is designed to measure parallel I/O performance at both the POSIX and MPI-IO level. It will be run on a small range of nodes (file system clients), as described below and in the README files included with the benchmark source code. All of the general run rules for NERSC-8/Trinity benchmarking apply.

Observed benchmark performance shall be obtained from a file system configured as closely as possible to the proposed global file system. If the proposed solution includes multiple file access protocols (e.g., pNFS or NFS as on certain platforms), benchmark results for IOR should be provided for each protocol.

Performance projections are permissible if they are derived from a similar system that is considered an earlier generation system. Projections shall be rigorously derived, easily understood and thoroughly documented.

## Submission Guidelines

Benchmark results (or projections including original results) for the proposed system shall be recorded in the spreadsheet provided. Note that in the supplied spreadsheet, the “Proposed System Node Count” entry refers to the value for the full, proposed system, whether benchmarked or projected. A paper version of all completed tables shall be submitted as part of the RFP response. Additionally, the Offeror shall submit electronically all completed tables, benchmark codes and output files and documentation on any code optimizations or configuration changes on a CD or similar medium. The submitted source shall be in a form that can be readily compiled on the proposed system. Do not include object and executable files, core dump files or large binary data files in the electronic submission. An audit trail showing any changes made to the benchmark codes must be supplied and it must be sufficient for the University to determine that the changes made conform to the spirit of the benchmark and do not violate any specific restrictions on the various benchmark codes.

If performance projections are used, this must be clearly indicated. The output files on which the projections are based, and a description of the projection method must be included. In addition, each system used for benchmark projections must be described. Enter the corresponding letter of the “System” column of Table 3 into the “System” column of the benchmark result tables. For solutions that use multiple phases of technology to offer the University the best value, the Offeror should duplicate the tables and provide the performance information for each phase.

## Modifications

Modifications to the benchmark are only permissible to enable correct execution on the target platform. Any modifications must be fully documented and reported back to NERSC/ACES. Changes related to optimization and tuning must be practical for production utilization of the filesystem. For example, optimizations that are tuning hints that can be controlled by users on the system are permissible, but optimizations that would require superuser privilege (but are not part of the steady-state configuration of the filesystem) are not permissible.

## Run Rules

The benchmark performance shall only be derived from runs that exhibit correct execution. Specific instructions to compile, execute and report results for each benchmark are included in the source distribution.

The intent of these benchmarks is to measure system performance for file read and write operations that access disk. Although optimizations may be possible that enable significant caching or buffering of the transferred data within system memory, and although such optimizations may be beneficial in a production environment, you are not permitted to engage in such optimizations for the benchmark runs. Consequently, the run rules are designed to defeat memory caching effects so as to measure the throughput of the underlying disk subsystem. Procedures for adjusting IOR benchmark script parameters to reduce memory-caching effects will be provided later in the document.

The benchmarks should be executed in a *fully-packed* manner on computational nodes. For MPI applications, this simply implies that the number of MPI tasks running on a node should equal the number of physical processors available on the node. In architectures with multiple cores, the number of MPI tasks per node, or number of instances, should equal the number of cores available on the node.

You must provide with your results an end-to-end description of the environment in which each benchmark was run. This will include:

* Client and server system configurations, including node and processor counts, processor models, memory size and speed, and OS (names and versions)
* Storage used for global file system and storage configurations.
* Network fabric used to connect servers, clients, and storage, including network configuration settings.
* Client and server configuration settings. These should include:
  + Client and server sysctl settings
  + Driver options
  + Network interface options
  + File system configuration options
* Compiler name and version, compiler options, and libraries used to build benchmarks.

## Benchmark Description

### IOR

IOR (MPI Coordinated Test of Parallel I/O) is a parallel file system test code developed by the SIOP (Scalable I/O Project) at Lawrence Livermore National Laboratory. This parallel program performs writes and reads to/from files under several sets of conditions and reports the resulting throughput rates. IOR stands for "**I**nterleaved **O**r **R**andom.”

#### Coding

MPI and MPI-IO are required in order to build and run the code. The source code used for NERSC-8/Trinity is version 2.10.3 and it is included here. More information about IOR is available on <http://sourceforge.net/projects/ior-sio> .

#### Building the Code

After extracting the archive, cd to the IOR-2.10.3 directory and type “make mpiio” which will build both the MPI-IO and the POSIX version of the code into the resulting executable.

Compiler options, names, etc., are in the file src/C/Makefile.config.

The resulting executable is src/C/IOR and it will need to be moved up to the “run” subdirectory.

You can also use “make clean” in the highest level directory to remove object (.o) files.

#### Required Runs

For all runs it is required to report the maximum write and maximum read bandwidth. Write bandwidth is the most important, so when in doubt report the parameters where maximum write bandwidth was achieved.

IOR will execute both read and write tests for each run, doing two repetitions of each and calculating the maximum values. The bandwidth numbers to be reported are the results listed as “Max Write” and “Max Read” measured in “MB/s”.

For more information on IOR, please read the User Guide included with the source distribution.

The vendor shall determine the file size to be used in the tests. See the description of transferSize in the **Running the Code** section.

The following tests will be run:

* MPI/IO file per process (e.g. N-N)
* MPI/IO shared file (e.g. N-1)
* POSIX I/O file per process (e.g. N-N)
* POSIX I/O shared file (e.g. N-1)

Each of the tests previously listed will be run for the following processor/node count conditions:

1. The number of processes on one node that yields the peak results for a single node.
2. Process count for each node equals the number of cores on a node. Find the number of nodes that yields the peak results for the test system.
3. Process count for each node equals the number of cores on a node. Run on the total number of nodes that exist on the test system.
4. Process count for each node equals the number of cores on a node. Estimate the bandwidth when the test is run on all nodes that will exist on the delivered system (a projection based on data gathered).

#### Running the Code

Definitions:

* transferSize: size (in bytes) of a single data buffer to be transferred in a single I/O call. Find the transfer size that produces the highest bandwidth results and report this transfer size. blockSize will always be equal to transferSize.
* File access: either a single file per MPI process is read/written (called “file-per-proc”) or a single shared file for all processes is read/written (called “shared file”).
* One of the tests will just measure throughput for a packed node. Since that concurrency is system dependent, we refer to this concurrency as *“nodesize”* in italics.

These runs must be set up via a combination of input configuration files and command line options, i.e., both must be used for any given run. One input configuration file is provided (in the “run” subdirectory” that implements these tests. It is called:

N8TrinityScript

Changes to this file are permitted only for the following four parameters:

testFile **(**path to data file)

hintsFileName **(**MPI-IO hints file)

collective(MPI-IO collective vs. independent operation mode)

segmentCount(number of segments (steps, datasets) in a file)

YOU MUST adjust the segmentCount parameter to ensure that the I/O benchmark files are not cached in DRAM (the block-buffer cache). You must set segmentCount so that the total amount of data written is greater than 1.5 times the available memory on the compute clients involved in the test. The total filesize is given by

filesize = segmentCount \* blocksize \* concurrency

For example, the baseline filesize for the supplied script is 6.4GB for a segmentCount of 100, and 1,000,000 byte blocksize and concurrency of 64. The segment count should be scaled so that the total filesize written is 1.5 times the sum of DRAM available on the client nodes to defeat the client-side buffer caches. Server buffer caches may still be used, but they must be configured as they would be for the delivered NERSC-8 and Trinity systems.

The exact execution line cannot be provided here, as it depends on the particular MPI implementation. A generic execution line might be:

mpirun -np 64 ./IOR -f ./ N8TrinityScript

This command will execute IOR on 64 clients with the supplied input configuration file called “N8TrinityScript”.

Six sample batch submission scripts (run.*platform*.*n*, where *platform* refers to the job scheduler used, and *n* refers to the number of MPI processes to be used with this file) are provided that contain the execution lines for all required runs.

You may provide your own MPI-IO “hints” file for the MPI-IO runs, which the program will look for in a file called “hintfile” (or in a file specified by the hintsFileName keyword in the input file). If this file is not present the program will issue warning messages that may be ignored.

For each run the program does eight repetitions of the write/read process. The maximum values are calculated and these values are to be reported as the benchmark result.

#### Timing Issues

The default timing function for the code is the standard C-library function gettimeofday(). This is used to measure the elapsed time of reading/writing to determine throughput.