IOR USER GUIDE

Index:

1. DESCRIPTION

IOR version 2 is a complete rewrite of the original IOR (Interleaved-Or-Random) version 1 code. IOR can be used for testing performance of parallel file systems using various interfaces and access patterns. IOR uses MPI for process synchronization.

2. BUILDING IOR

Build Instructions:

Type 'gmake [posix|mpiio|hdf5|ncmpi|all]' from the IOR/ directory. In IOR/src/C, the file Makefile.config currently has settings for AIX, Linux, OSF1 (TRU64), and IRIX64 to model on. Note that MPI must be present for building/running IOR, and that MPI I/O must be available for MPI I/O, HDF5, and Parallel netCDF builds. As well, HDF5 and Parallel netCDF libraries are necessary for those builds. All IOR builds include the POSIX interface.

3. RUNNING IOR

Two ways to run IOR:

* Interactive command line with arguments -- executable followed by command line options.

```
    E.g., to execute:  IOR -w -r -o filename
    This performs a write and a read to the file 'filename'.

  * Interactive command line with scripts -- any arguments on the command line
    will establish the default for the test run, but a script may be used in
    conjunction with this for varying specific tests during an execution of the
    code.

    E.g., to execute:  IOR -W -f script
    This defaults all tests in 'script' to use write data checking.


**************
* 4. OPTIONS *
**************
These options are to be used on the command line. E.g., 'IOR -a POSIX -b 4K'.
  -a S  api --  API for I/O [POSIX|MPIIO|HDF5|NCMPI]
  -b N  blockSize -- contiguous bytes to write per task (e.g.: 8, 4k, 2m, 1g)
  -B    useO_DIRECT -- uses O_DIRECT for POSIX, bypassing I/O buffers
  -c    collective -- collective I/O
  -C    reorderTasks -- changes task ordering to n+1 ordering for readback
  -d N  interTestDelay -- delay between reps in seconds
  -D N  deadlineForStonewalling -- seconds before stopping write or read phase
  -e    fsync -- perform fsync upon POSIX write close
  -E    useExistingTestFile -- do not remove test file before write access
  -f S  scriptFile -- test script name
  -F    filePerProc -- file-per-process
  -g    intraTestBarriers -- use barriers between open, write/read, and close
  -G N  setTimeStampSignature -- set value for time stamp signature
  -h    showHelp -- displays options and help
  -H    showHints -- show hints
  -i N  repetitions -- number of repetitions of test
  -I    individualDataSets -- datasets not shared by all procs [not working]
  -j N  outlierThreshold -- warn on outlier N seconds from mean
  -J N  setAlignment -- HDF5 alignment in bytes (e.g.: 8, 4k, 2m, 1g)
  -k    keepFile -- don't remove the test file(s) on program exit
  -K    keepFileWithError  -- keep error-filled file(s) after data-checking
  -l    storeFileOffset -- use file offset as stored signature
  -m    multiFile -- use number of reps (-i) for multiple file count
  -n    noFill -- no fill in HDF5 file creation
  -N N  numTasks -- number of tasks that should participate in the test
  -o S  testFile -- full name for test
  -O S  string of IOR directives (-O checkRead=1,lustreStripeCount=32)
  -p    preallocate -- preallocate file size
  -P    useSharedFilePointer -- use shared file pointer [not working]
  -q    quitOnError -- during file error-checking, abort on error
  -r    readFile -- read existing file
  -R    checkRead -- check read after read
  -s N  segmentCount -- number of segments
  -S    useStridedDatatype -- put strided access into datatype [not working]
  -t N  transferSize -- size of transfer in bytes (e.g.: 8, 4k, 2m, 1g)
  -T N  maxTimeDuration -- max time in minutes to run tests
  -u    uniqueDir -- use unique directory name for each file-per-process
  -U S  hintsFileName -- full name for hints file
  -v    verbose -- output information (repeating flag increases level)
  -V    useFileView -- use MPI_File_set_view
```

```
  -w    writeFile -- write file
  -W    checkWrite -- check read after write
  -x    singleXferAttempt -- do not retry transfer if incomplete
  -z    randomOffset -- access is to random, not sequential, offsets

NOTES: * S is a string, N is an integer number.
       * For transfer and block sizes, the case-insensitive K, M, and G
         suffices are recognized.  I.e., '4k' or '4K' is accepted as 4096.


**********************
* 5. OPTION DETAILS *
**********************
For each of the general settings, note the default is shown in brackets.

GENERAL:
========
  * api                 - must be set to one of POSIX, MPIIO, HDF5, or NCMPI
                          depending on test [POSIX]

  * testFile            - name of the output file [testFile]
                          NOTE: with filePerProc set, the tasks can round
                                robin across multiple file names '-o S@S@S'

  * hintsFileName       - name of the hints file []

  * repetitions         - number of times to run each test [1]

  * multiFile           - creates multiple files for single-shared-file or
                          file-per-process modes; i.e. each iteration creates
                          a new file [0]

  * reorderTasks        - reorders tasks for writing/reading neighbor's
                          data from different nodes [0]

  * quitOnError         - upon error encountered on checkWrite or checkRead,
                          display current error and then stop execution;
                          if not set, count errors and continue [0]

  * numTasks            - number of tasks that should participate in the test
                          [0]
                          NOTE: 0 denotes all tasks

  * interTestDelay      - this is the time in seconds to delay before
                          beginning a write or read in a series of tests [0]
                          NOTE: it does not delay before a check write or
                                check read

  * outlierThreshold    - gives warning if any task is more than this number
                          of seconds from the mean of all participating tasks.
                          If so, the task is identified, its time (start,
                          elapsed create, elapsed transfer, elapsed close, or
                          end) is reported, as is the mean and standard
                          deviation for all tasks.  The default for this is 0,
                          which turns it off.  If set to a positive value, for
                          example 3, any task not within 3 seconds of the mean
                          displays its times. [0]
```

```
* intraTestBarriers    - use barrier between open, write/read, and close [0]

* uniqueDir            - create and use unique directory for each
                         file-per-process [0]

* writeFile            - writes file(s), first deleting any existing file [1]
                         NOTE: the defaults for writeFile and readFile are
                               set such that if there is not at least one of
                               the following -w, -r, -W, or -R, it is assumed
                               that -w and -r are expected and are
                               consequently used -- this is only true with
                               the command line, and may be overridden in
                               a script

* readFile             - reads existing file(s) (from current or previous
                         run) [1]
                         NOTE: see writeFile notes

* filePerProc          - accesses a single file for each processor; default
                         is a single file accessed by all processors [0]

* checkWrite           - read data back and check for errors against known
                         pattern; can be used independently of writeFile [0]
                         NOTES: * data checking is not timed and does not
                                  affect other performance timings
                                * all errors tallied and returned as program
                                  exit code, unless quitOnError set

* checkRead            - reread data and check for errors between reads; can
                         be used independently of readFile [0]
                         NOTE: see checkWrite notes

* keepFile             - stops removal of test file(s) on program exit [0]

* keepFileWithError    - ensures that with any error found in data-checking,
                         the error-filled file(s) will not be deleted [0]

* useExistingTestFile  - do not remove test file before write access [0]

* segmentCount         - number of segments in file [1]
                         NOTES: * a segment is a contiguous chunk of data
                                  accessed by multiple clients each writing/
                                  reading their own contiguous data;
                                  comprised of blocks accessed by multiple
                                  clients
                                * with HDF5 this repeats the pattern of an
                                  entire shared dataset

* blockSize            - size (in bytes) of a contiguous chunk of data
                         accessed by a single client; it is comprised of one
                         or more transfers [1048576]

* transferSize         - size (in bytes) of a single data buffer to be
                         transferred in a single I/O call [262144]

* verbose              - output information [0]
```

```
                              NOTE: this can be set to levels 0-5 on the command
                                    line; repeating the -v flag will increase
                                    verbosity level

  * setTimeStampSignature - set value for time stamp signature [0]
                             NOTE: used to rerun tests with the exact data
                                   pattern by setting data signature to contain
                                   positive integer value as timestamp to be
                                   written in data file; if set to 0, is
                                   disabled

  * showHelp              - display options and help [0]

  * storeFileOffset       - use file offset as stored signature when writing
                            file [0]
                            NOTE: this will affect performance measurements

  * maxTimeDuration       - max time in minutes to run tests [0]
                            NOTES: * setting this to zero (0) unsets this option
                                   * this option allows the current read/write
                                     to complete without interruption

  * deadlineForStonewalling - seconds before stopping write or read phase [0]
                              NOTES: * used for measuring the amount of data moved
                                       in a fixed time.  After the barrier, each
                                       task starts its own timer, begins moving
                                       data, and the stops moving data at a pre-
                                       arranged time.  Instead of measuring the
                                       amount of time to move a fixed amount of
                                       data, this option measures the amount of
                                       data moved in a fixed amount of time.  The
                                       objective is to prevent tasks slow to
                                       complete from skewing the performance.
                                     * setting this to zero (0) unsets this option
                                     * this option is incompatible w/data checking

  * randomOffset          - access is to random, not sequential, offsets [0]
                            NOTES: * this option is currently incompatible with:
                                     -checkRead
                                     -storeFileOffset
                                     -MPIIO collective or useFileView
                                     -HDF5 or NCMPI


POSIX-ONLY:
===========
  * useO_DIRECT           - use O_DIRECT for POSIX, bypassing I/O buffers [0]

  * singleXferAttempt     - will not continue to retry transfer entire buffer
                            until it is transferred [0]
                            NOTE: when performing a write() or read() in POSIX,
                                  there is no guarantee that the entire
                                  requested size of the buffer will be
                                  transferred; this flag keeps the retrying a
                                  single transfer until it completes or returns
                                  an error
```

```
  * fsync                - perform fsync after POSIX write close [0]

MPIIO-ONLY:
===========
  * preallocate          - preallocate the entire file before writing [0]

  * useFileView          - use an MPI datatype for setting the file view option
                           to use individual file pointer [0]
                           NOTE: default IOR uses explicit file pointers

  * useSharedFilePointer - use a shared file pointer [0] (not working)
                           NOTE: default IOR uses explicit file pointers

  * useStridedDatatype   - create a datatype (max=2GB) for strided access; akin
                           to MULTIBLOCK_REGION_SIZE [0] (not working)

HDF5-ONLY:
==========
  * individualDataSets   - within a single file each task will access its own
                           dataset [0] (not working)
                           NOTE: default IOR creates a dataset the size of
                                 numTasks * blockSize to be accessed by all
                                 tasks

  * noFill               - no pre-filling of data in HDF5 file creation [0]

  * setAlignment         - HDF5 alignment in bytes (e.g.: 8, 4k, 2m, 1g) [1]

MPIIO-, HDF5-, AND NCMPI-ONLY:
==============================
  * collective           - uses collective operations for access [0]

  * showHints            - show hint/value pairs attached to open file [0]
                           NOTE: not available in NCMPI

LUSTRE-SPECIFIC:
================
  * lustreStripeCount    - set the lustre stripe count for the test file(s) [0]

  * lustreStripeSize     - set the lustre stripe size for the test file(s) [0]

  * lustreStartOST       - set the starting OST for the test file(s) [-1]

  * lustreIgnoreLocks    - disable lustre range locking [0]


**********************
* 6. VERBOSITY LEVELS *
**********************
The verbosity of output for IOR can be set with -v.  Increasing the number of
-v instances on a command line sets the verbosity higher.

Here is an overview of the information shown for different verbosity levels:
  0 - default; only bare essentials shown
  1 - max clock deviation, participating tasks, free space, access pattern,
      commence/verify access notification w/time
  2 - rank/hostname, machine name, timer used, individual repetition
```

```
      performance results, timestamp used for data signature
  3 - full test details, transfer block/offset compared, individual data
      checking errors, environment variables, task writing/reading file name,
      all test operation times
  4 - task id and offset for each transfer
  5 - each 8-byte data signature comparison (WARNING: more data to STDOUT
      than stored in file, use carefully)


********************
* 7. USING SCRIPTS *
********************

IOR can use a script with the command line.  Any options on the command line
will be considered the default settings for running the script.  (I.e.,
'IOR -W -f script' will have all tests in the script run with the -W option as
default.)  The script itself can override these settings and may be set to run
run many different tests of IOR under a single execution.  The command line is:

  IOR/bin/IOR -f script

In IOR/scripts, there are scripts of test cases for simulating I/O behavior of
various application codes.  Details are included in each script as necessary.

An example of a script:
===============> start script <===============
IOR START
  api=[POSIX|MPIIO|HDF5|NCMPI]
  testFile=testFile
  hintsFileName=hintsFile
  repetitions=8
  multiFile=0
  interTestDelay=5
  readFile=1
  writeFile=1
  filePerProc=0
  checkWrite=0
  checkRead=0
  keepFile=1
  quitOnError=0
  segmentCount=1
  blockSize=32k
  outlierThreshold=0
  setAlignment=1
  transferSize=32
  singleXferAttempt=0
  individualDataSets=0
  verbose=0
  numTasks=32
  collective=1
  preallocate=0
  useFileView=0
  keepFileWithError=0
  setTimeStampSignature=0
  useSharedFilePointer=0
  useStridedDatatype=0
  uniqueDir=0
```

```
  fsync=0
  storeFileOffset=0
  maxTimeDuration=60
  deadlineForStonewalling=0
  useExistingTestFile=0
  useO_DIRECT=0
  showHints=0
  showHelp=0
RUN
  # additional tests are optional
  <snip>
RUN
  <snip>
RUN
IOR STOP
===============> stop script <===============

NOTES: * Not all test parameters need be set.  The defaults can be viewed in
         IOR/src/C/defaults.h.
       * White space is ignored in script, as are comments starting with '#'.


**************************************
* 8. COMPATIBILITY WITH OLDER VERSIONS *
**************************************
1)  IOR version 1 (c. 1996-2002) and IOR version 2 (c. 2003-present) are
    incompatible.  Input decks from one will not work on the other.  As version
    1 is not included in this release, this shouldn't be case for concern.  All
    subsequent compatibility issues are for IOR version 2.

2)  IOR versions prior to release 2.8 provided data size and rates in powers
    of two.  E.g., 1 MB/sec referred to 1,048,576 bytes per second.  With the
    IOR release 2.8 and later versions, MB is now defined as 1,000,000 bytes
    and MiB is 1,048,576 bytes.

3)  In IOR versions 2.5.3 to 2.8.7, IOR could be run without any command line
    options.  This assumed that if both write and read options (-w -r) were
    omitted, the run with them both set as default.  Later, it became clear
    that in certain cases (data checking, e.g.) this caused difficulties.  In
    IOR versions 2.8.8 and later, if not one of the -w -r -W or -R options is
    set, then -w and -r are set implicitly.


*********************************
* 9. FREQUENTLY ASKED QUESTIONS *
*********************************
HOW DO I PERFORM MULTIPLE DATA CHECKS ON AN EXISTING FILE?

  Use this command line:  IOR -k -E -W -i 5 -o file

  -k keeps the file after the access rather than deleting it
  -E uses the existing file rather than truncating it first
  -W performs the writecheck
  -i number of iterations of checking
  -o filename

  On versions of IOR prior to 2.8.8, you need the -r flag also, otherwise
```

you'll first overwrite the existing file. (In earlier versions, omitting -w and -r implied using both. This semantic has been subsequently altered to be omitting -w, -r, -W, and -R implied using both -w and -r.)

If you're running new tests to create a file and want repeat data checking on this file multiple times, there is an undocumented option for this. It's -O multiReRead=1, and you'd need to have an IOR version compiled with the USE_UNDOC_OPT=1 (in iordef.h). The command line would look like this:

```
IOR -k -E -w -W -i 5 -o file -O multiReRead=1
```

For the first iteration, the file would be written (w/o data checking). Then for any additional iterations (four, in this example) the file would be reread for whatever data checking option is used.

HOW DOES IOR CALCULATE PERFORMANCE?

IOR performs get a time stamp START, then has all participating tasks open a shared or independent file, transfer data, close the file(s), and then get a STOP time. A stat() or MPI_File_get_size() is performed on the file(s) and compared against the aggregate amount of data transferred. If this value does not match, a warning is issued and the amount of data transferred as calculated from write(), e.g., return codes is used. The calculated bandwidth is the amount of data transferred divided by the elapsed STOP-minus-START time.

IOR also gets time stamps to report the open, transfer, and close times. Each of these times is based on the earliest start time for any task and the latest stop time for any task. Without using barriers between these operations (-g), the sum of the open, transfer, and close times may not equal the elapsed time from the first open to the last close.

HOW DO I ACCESS MULTIPLE FILE SYSTEMS IN IOR?

It is possible when using the filePerProc option to have tasks round-robin across multiple file names. Rather than use a single file name '-o file', additional names '-o file1@file2@file3' may be used. In this case, a file per process would have three different file names (which may be full path names) to access. The '@' delimiter is arbitrary, and may be set in the FILENAME_DELIMITER definition in iordef.h.

Note that this option of multiple filenames only works with the filePerProc -F option. This will not work for shared files.

HOW DO I BALANCE LOAD ACROSS MULTIPLE FILE SYSTEMS?

As for the balancing of files per file system where different file systems offer different performance, additional instances of the same destination path can generally achieve good balance.

For example, with FS1 getting 50% better performance than FS2, set the '-o' flag such that there are additional instances of the FS1 directory. In this case, '-o FS1/file@FS1/file@FS1/file@FS2/file@FS2/file' should adjust for the performance difference and balance accordingly.

HOW DO I USE STONEWALLING?

To use stonewalling (-D), it's generally best to separate write testing from read testing. Start with writing a file with '-D 0' (stonewalling disabled) to determine how long the file takes to be written. If it takes 10 seconds for the data transfer, run again with a shorter duration, '-D 7' e.g., to stop before the file would be completed without stonewalling. For reading, it's best to create a full file (not an incompletely written file from a stonewalling run) and then run with stonewalling set on this preexisting file. If a write and read test are performed in the same run with stonewalling, it's likely that the read will encounter an error upon hitting the EOF. Separating the runs can correct for this. E.g.,

```
IOR -w -k -o file -D 10  # write and keep file, stonewall after 10 seconds
IOR -r -E -o file -D 7   # read existing file, stonewall after 7 seconds
```

Also, when running multiple iterations of a read-only stonewall test, it may be necessary to set the -D value high enough so that each iteration is not reading from cache. Otherwise, in some cases, the first iteration may show 100 MB/s, the next 200 MB/s, the third 300 MB/s. Each of these tests is actually reading the same amount from disk in the allotted time, but they are also reading the cached data from the previous test each time to get the increased performance. Setting -D high enough so that the cache is overfilled will prevent this.

HOW DO I BYPASS CACHING WHEN READING BACK A FILE I'VE JUST WRITTEN?

One issue with testing file systems is handling cached data. When a file is written, that data may be stored locally on the node writing the file. When the same node attempts to read the data back from the file system either for performance or data integrity checking, it may be reading from its own cache rather from the file system.

The reorderTasks '-C' option attempts to address this by having a different node read back data than wrote it. For example, node N writes the data to file, node N+1 reads back the data for read performance, node N+2 reads back the data for write data checking, and node N+3 reads the data for read data checking, comparing this with the reread data from node N+4. The objective is to make sure on file access that the data is not being read from cached data.

```
Node 0: writes data
Node 1: reads data
Node 2: reads written data for write checking
Node 3: reads written data for read checking
Node 4: reads written data for read checking, comparing with Node 3
```

The algorithm for skipping from N to N+1, e.g., expects consecutive task numbers on nodes (block assignment), not those assigned round robin (cyclic assignment). For example, a test running 6 tasks on 3 nodes would expect tasks 0,1 on node 0; tasks 2,3 on node 1; and tasks 4,5 on node 2. Were the assignment for tasks-to-node in round robin fashion, there would be tasks 0,3 on node 0; tasks 1,4 on node 1; and tasks 2,5 on node 2. In this case, there would be no expectation that a task would not be reading from data cached on

a node.

HOW DO I USE HINTS?

It is possible to pass hints to the I/O library or file system layers following this form:

'setenv IOR_HINT__<layer>__<hint> <value>'

For example:

'setenv IOR_HINT__MPI__IBM_largeblock_io true'

'setenv IOR_HINT__GPFS__important_hint true'

or, in a file in the form:

'IOR_HINT__<layer>__<hint>=<value>'

Note that hints to MPI from the HDF5 or NCMPI layers are of the form:

'setenv IOR_HINT__MPI__<hint> <value>'

HOW DO I EXPLICITY SET THE FILE DATA SIGNATURE?

The data signature for a transfer contains the MPI task number, transfer-buffer offset, and also timestamp for the start of iteration. As IOR works with 8-byte long long ints, the even-numbered long longs written contain a 32-bit MPI task number and a 32-bit timestamp. The odd-numbered long longs contain a 64-bit transferbuffer offset (or file offset if the '-l' storeFileOffset option is used). To set the timestamp value, use '-G' or setTimeStampSignature.

HOW DO I EASILY CHECK OR CHANGE A BYTE IN AN OUTPUT DATA FILE?

There is a simple utility IOR/src/C/cbif/cbif.c that may be built. This is a stand-alone, serial application called cbif (Change Byte In File). The utility allows a file offset to be checked, returning the data at that location in IOR's data check format. It also allows a byte at that location to be changed.

HOW DO I CORRECT FOR CLOCK SKEW BETWEEN NODES IN A CLUSTER?

To correct for clock skew between nodes, IOR compares times between nodes, then broadcasts the root node's timestamp so all nodes can adjust by the difference. To see an egregious outlier, use the '-j' option. Be sure to set this value high enough to only show a node outside a certain time from the mean.

WHAT HAPPENED TO THE GUI?

In versions of IOR earlier than 2.9.x, there was a GUI available. Over time it became clear that it wasn't find enough use to warrant maintenance. It was retired in IOR-2.10.x.