

# ECMWF training course

## January 23-27 2017

### I/O practicals darshan – ccb

#### NOTES:

1. Remember to login to the HPC: ccb
2. See slides, man pages or online documentation.
3. Some job examples are available under:  
<https://software.ecmwf.int/wiki/display/UDOC/Batch+environment%3A++PBS>
4. Create a subdirectory for this practical session, e.g.

```
% cd $SCRATCH
% tar xzf ~trx/io-darshan/io-darshan-practicals.tar.gz
% cd io-darshan
```

#### BENCHMARK description

IOR can be used for testing performance of parallel file systems using various interfaces and access patterns. IOR uses MPI for process synchronization.

```
*****
```

```
* 3. RUNNING IOR *
```

```
*****
```

```
Two ways to run IOR:
```

```
* 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'.
```

```
* 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 N testNum -- test number for reference in some output
```

```
-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
```

```
-Q N taskPerNodeOffset for read tests use with -C & -Z options (-C constant N, -Z
at least N) [!HDF5]
```

```
-Z reorderTasksRandom -- changes task ordering to random ordering for readback
```

```
-X N reorderTasksRandomSeed -- random seed for -Z option
```

```
-d N interTestDelay -- delay between reps in seconds
```

```
-D N deadlineForStonewalling -- seconds before stopping write or read phase
```

```
-Y fsyncPerWrite -- perform fsync after each POSIX write
```

```
-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 (e.g. -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 within a file

```

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.

## EXERCISE 0

To compile IOR, you have to follow these steps:

```

cd src/IOR
module unload atp
#make sure that PrgEnv-cray/5.2.82 is loaded
make mpiio
cp src/C/IOR ../../bin/

```

## EXERCISE 1

In this exercise we are profiling the I/O of some POSIX ways to read/write a single file or several files with Darshan.

### **Comparison between 144 tasks writing one single file vs. 144 tasks writing 144 different files**

This exercise will help to check the difference between write/read a single file and write/read 1 file per task.

1. Go to **\$SCRATCH/io-darshan/run/single-multiple** folder.
2. You have to complete the **job-posix.pbs** script with the correct values (search for #TODO and "...").
  - a. Unload the **atp** module and load **darshan** module
  - b. Export **DARSHAN\_LOG\_DIR** to some location in \$SCRATCH
  - c. Replace **##arguments##** by the proper IOR arguments:

## single-shared-file benchmark

Command line used: `${EXE} -C -t 2m -b 500m -i 1 -a POSIX -w -r`  
Summary:

```
api                = POSIX
test filename      = testFile
access             = single-shared-file
clients            = 144 (72 per node)
repetitions        = 1
xfersize           = 2 MiB
blocksize          = 500 MiB
aggregate filesize = 70.31 GiB
```

## file-per-process benchmark

Command line used: `${EXE} -F -C -t 2m -b 500m -i 1 -a POSIX -w -r`

Summary:

```
api                = POSIX
test filename      = testFile
access             = file-per-process
clients            = 144 (72 per node)
repetitions        = 1
xfersize           = 2 MiB
blocksize          = 500 MiB
aggregate filesize = 70.31 GiB
```

Now submit the job using **qsub**.

(We have created the two darshan logs in **darshan-logs** directory to prevent waiting in the queue and the execution. Once running the job lasts about 10 minutes).

HINT: To compare both summaries, we suggest you to use **xxdiff** command.

Generate two different text files to compare redirecting stdout:

```
module load darshan
darshansummary user_XXXX_t2b500_IOR_XXX.darshan.gz > single-
shared
darshansummary user_XXXX_t2b500F_IOR_XXX.darshan.gz > file-
per-process
xxdiff single-shared file-per-process
```

Fill in the table:

	<b>single-shared-file</b>	<b>file-per-process</b>
Read time per task		
Write Time per task		
Number of different files		

What is the best way to achieve the best performance? Why?

## **EXERCISE 2**

### **Comparison of 144 tasks writing a single file using MPI-IO with and without stripe**

In this exercise you are writing a single file of 70.31 GB in a folder that does not have stripe and then in a folder with stripe. Both tests will use MPI-IO to write the file using aligned chunks.

1. Go to **\$SCRATCH/io-darshan/run/mpiio** folder.
2. You have to complete **job-mpiio.pbs** (again #TODO and ##arguments##)

- a. First you have to create two different folders called:

- i. MPIIO

```
1. mkdir MPIIO
```

- ii. MPIIO\_stripe

```
1. mkdir MPIIO_stripe
```

- b. Then set the 2M stripe to MPIIO-stripe. Use this command:

```
lfs setstripe -S 2097152 -c 4 MPIIO_stripe
```

This will set a stripe of 2MB per OST with a count of 4 OSTs per file.

Allowing MPI-IO to enhance the read/write

- c. Both arguments should be:

```
-C -t 2m -b 500m -I 1 -a MPIIO -w -r
```

Then the job will submit two *aprun* commands, one in the MPIIO directory and the other on MPIIO-stripe. Both will use MPI-IO to write a single-shared-file of 70.31GiB in chunks of 500Mb, one per process. Then you can compare the effect of the stripe and MPI-IO.

This job takes around 15 minutes. You can use the logs in **darshan-logs** directory.

	<b>No-stripe</b>	<b>stripe</b>
Read time per task		
Write Time per task		
Meta Time per task		

Now try with different stripe sizes (4MB, 8MB) and different transfersize (-t) parameters?