Designing and building applications for extreme scale systems
CS598 SP2016
HW10: Parallel I/O with MPI

Goals

- Gain experience with MPI parallel I/O communication
- Explore collective I/O and optimizations for parallel I/O

Task

There are two forms for this homework, depending on how much you wish to learn. The grading will be the same.

The task is to run a program that writes a 2-D array, stored by rows (Fortran order), of size m x n, to a single file. The array is stored in a standard order, as shown in lecture 33, slides 12 and 13. The code on slide 13 "Case I: Local Array Contiguous in Memory," gives an example of one I/O approach, the fully collective I/O example.

The program should write out the file using two approaches:

1. Fully collective, using file views and a single collective "MPI_File_write_all" call.
2. Fully independent, using only a contiguous column of the array for each write, and using MPI_File_write_at

The program should record the time for the write and the MPI_File_close, and report both times for both cases, as well as the I/O rate in bytes/second for the write. For times, take the maximum over all processes.

The file should be opened with an MPI_Info value that specifies the striping_factor and cb_nodes. These should be set to the same value. The program should take three optional arguments: m, n, and k, where the array is of total size m x n and the striping_factor (and cb_nodes) is k (a value for k of 0 means "use the default", and not use MPI_Info to set the striping_factor). The file should be written to the /scratch file system; the directory should be /scratch/training/<your-user-name>, e.g., /scratch/training/tra666. The file name should be testfile-3.out for the collective I/O case and testfile-0.out for the independent I/O case.

Run the program (let's call it ioda) with

mpiexec -n 1024 -ppn 16 -maxtime 20:00 ./ioda 16384 16384 32
<delete the files that are created>
mpiexec -n 1024 -ppn 16 -maxtime 20:00 ./ioda 16384 16384 16
<delete the files that are created>
mpiexec -n 1024 -ppn 16 -maxtime 20:00 ./ioda 16384 16384 0
<delete the files that are created>

Make sure you delete the files from /scratch/training/<your-user-name> when you are done. Use lsf getstripe <filename> to check the file striping parameters on the created files.

Specific time limits have been given above. If the program times out for a particular choice, just note that as a "did not complete". You do not need to run those jobs to completion (some will take a long time with independent I/O).

The two choices: You can either write your own program or use ioda.c , which is also posted on the moodle. Or you can write your own and compare it to ioda.c . The ioda.c code provided has a few more features, including control of the stripe size and an independent-io-only hint.

You may want to either put these runs into a batch script for the batch system. Alternately, if you use a shell script, called, for example, iorun, you may want to run it with

```
nohup ./iorun >iorun.out 2>&1 </dev/null &
```

The reason for the nohup is that Blue Waters will often terminate your interactive shell; this normally terminates programs that are running, even if they are running in the background. Using nohup ("no hangup") to run the script should prevent this.

Submit

The results of running the program for the three cases given above. Discuss the results. How much difference is there between the two I/O styles (collective and independent)? How important is using striped I/O?

To think about (but not turn in)

1. How important is setting the striping factor? Try some different values. Do these match what you expect? That is, if you made a performance model where the I/O performance was proportional to the striping factor, does that fit what you observe?
2. Consider changing the striping unit (size of data in each stripe). What effect does that have?
3. What happens if you add a file header of 100 bytes? Does it change the performance?