CSCI-4320/636, Group Assignment #4: MPI Parallel I/O Benchmark

Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York U.S.A. 12180-3590
Email: `chrisc@cs.rpi.edu`

March 19, 2021

DUE DATE: Noon, Friday, April 2nd, 2021

1 Assignment Description

For this GROUP assignment (2 to 4 team members), you are to develop an MPI-based C program which performs the following parallel I/O benchmark computation using `AiMOS` and the `MPI_File_write_at` and `MPI_File_read_at` for a SINGLE FILE to:

- **scratch** sub-directory on the Parallel Disk Storage for one (and only one) project member's account.
- **NVMe** drive for one project member's account.

To use the NVMe storage in a node, request it along with the job specification: `--gres=nvme` (This can be combined with other requests, such as GPUs.) When the first job step starts, the system will initialize the storage and create the path `/mnt/nvme/uid_${SLURM_JOB_UID}/job_${SLURM_JOB`

- Processor Block size: 128K, 256K, 512K, 1M, 2M, 4M, 8M and 16M. Note, these are dummy blocks of data (block with all 1's) that was allocated using regular `malloc` or `calloc` memory allocation system calls.

- Blocks to write/read: 32 per test. Between I/O tests, make sure you close and re-open the file. Perform the write test first, then read the data written. Also, each rank reads or writes in order for each block. That is the file order is rank 0 block 0, rank 1, block0 ....rank 64 block 0, then rank 0 block 1, rank 1 block1...
- MPI ranks: 2, 4, 8, 16, 32, 64. **Note the max ranks per compute node should be no higher than 32.** Thus, you will run experiments using up-to 2 compute nodes with 4 POWER9 CPUs in total.
- Use the `taskset` command to bind MPI ranks to particular CPU cores to avoid using the "hyperthread" cores. For example, to do a 32 MPI rank run from the commandline (all on a single line), do:

  ```
  taskset --cpu-list 0,2,4,8,12,16,20,24,28,32,36,40,44,48,52,56,
  60,64,68,72,76,80,84,88,92,96,100,104,108,112,116,120,124 mpirun
  -np 32 ./mpi-io-benchmark
  ```
- Redo the above experiments using the NVMe storage device(s). Note that each compute node has it's own NVMe device. So, for the two compute node / 64 MPI rank case, 32 ranks on Node 0 will write to one file and 32 ranks on Node 1 will write to a different file that is local to that particular NVMe device.
- **MAKE SURE ALL DATA FILES ARE DELETED BETWEEN TESTS!! The maximum datafile will be 32 GB.**

Measure the execution time of each test using the hardware clock (see below) and output that performance data, convert to a bandwidth (e.g., Giga or Mega-bytes per second) and plot the performance results organized in the following 4 groups:

- write bandwidth for Parallel Disk Storage as a function of MPI ranks. Separate plot line for each block size.
- read bandwidth for Parallel Disk Storage as a function of MPI ranks. Separate plot line for each block size.
- write bandwidth for NVMe storage as a function of MPI ranks. Separate plot line for each block size.
- read bandwidth for NVMe storage as a function of MPI ranks. Separate plot line for each block size.

What trends do you see in this data? Does it matter which device you write to or read from? Do read and writes bandwidth differ? If so, why and by how much. What performance trend is observed as block size is increased for reads and for writes? These are the questions you'll want to address in your report.

NOTE: you can do this in teams of up-to 4 students per team. Ideally, try to use the same team as you will be using for your group project.

2 Hardware Clock

The function below returns the free running hardare clock time on the AiMOS/POWER9 system. This clock has a rate of 512,000,000 cycles per second or 512MHz.

```
unsigned long long aimos_clock_read(void)
{
  unsigned int tbl, tbu0, tbu1;

  do {
    __asm__ __volatile__ ("mftbu %0" : "=r"(tbu0));
    __asm__ __volatile__ ("mftb %0" : "=r"(tbl));
    __asm__ __volatile__ ("mftbu %0" : "=r"(tbu1));
  } while (tbu0 != tbu1);

  return (((unsigned long long)tbu0) << 32) | tbl;
}
```

Call the function at the start of a timed section of code and place in a variable called `start` and again at the end of a timed section of code using a variable called `end` and do `end - start` to obtain the time in cycles. Make sure that you cycle time variables are decleared to be `unsigned long long`.

3 HAND-IN INSTRUCTIONS and GRADING CRITERIA

Submit your code, your AiMOS job scripts with `mpirun` commands and project performance report to `submitty.cs.rpi.edu`.