Prof. Jingke Li (lij@pdx.edu); Classes/Labs: TR 1400-1550 @FAB 150, 88-10; Office Hours: TR 1300-1400 @FAB 120-06

Lab 8: MPI File IO

1 File IO Samples

1. Read and understand the MPI program `file-in.c`. It opens a file; reads two integers from the file; and prints out their values. The input file name is provided as a command-line argument. Note that the input file is byte-encoded. To view its content, use the `od` command:

   ```
   linux> od -i data.txt
   ```

   Compile and run this program with different number of processes:

   ```
   linux> mpirun -n 4 file-in data.txt
   linux> mpirun -n 8 file-in data.txt
   ```

   Modify the buffer size and have each process read in four integers.

2. Read and understand the program `file-out.c`. This program is similar to the previous one, except that it is for writing to files. Compile and run this program with any number of processes:

   ```
   linux> mpirun -n 4 file-out output
   ```

   What do you observe? How many output files are created? What are the contents?

   Change this program so that there is only one output file, `output.all`; and have all processes write to this file. Compile and run. What do you see in this file? Can you explain?

3. Read and understand the program `file-view.c`. Pay attention to the `MPI_File_set_view()` line. Compile and run this program. Change the `offset` parameter and see the effect.

2 Two New File-IO Programs

Write two new programs, `file-io1.c` and `file-io2.c`. They have the same interface,

```
linux> mpirun -n P file-io[12] <infile> <outfile>
```

i.e., they take two file names as command-line arguments. Both programs read data from a file, perform some operations, and write results to another file.

Input Data The input file contains a set of four-byte (32-bits) integer values. The total number of these value, `N`, can be derived through the following file-IO call:

```
MPI_Offset fsize;
MPI_File_get_size(fin, &fsize);     // fin is the infile handle
int N = (int) fsize / sizeof(int);
```

The provided program `datagen.c` can be used to generate new input data sets. It takes a command-line argument, `N`, and generates `N` random integers (with values in the range `[0, 8191]`):

```
linux> ./datagen 1024 > data1k
```

The range constraint on the values is non-essential; it is for easy visualization – these values are all expressible by 4 digits.

Your Tasks

- The first program, `file-io1.c`, implements the following actions:
  1. All processes open the input file, and figure out the total number of integer values in the file, `N`. They check that `N` evenly divides `P`. If not, process 0 prints out an informative message, and all processes gracefully terminate (i.e. by making a call to `MPI_Finalize()`).
  2. Each process allocates an array of size `N/P`, and reads a unique section of data of that size from the input file. A proper file-view offset needs to be setup for that.
  3. The processes collectively compute the total sum of all `N` integer values, and process 0 prints out the result. For this step, you need to figure out what local computation and what communication are needed.
  4. Each process doubles each integer's value in its array, and write the resulting data (still `N/P` integer values) to a proper section in the output file.
- The second program, `file-io2.c`, follows the same actions as above, except that it removes the "`N` evenly divides `P`" constraint. It still distributes work among processes as evenly as possible (i.e. section sizes differ by at most 1).

Here are some sample runs of these two programs:

```
linux> mpirun -n 4 file-io1 in16 out16
P0: psum = 18846
P1: psum = 15990
P2: psum = 18253
P3: psum = 16409
The result sum is 69498
linux> od -i in16
0000000        7653        1338        3008        6847
0000020        5919        3580        6404          87
0000040        1376        4687        6916        5274
0000060        3091        4948        3203        5167
linux> od -i out16
0000000       15306        2676        6016       13694
0000020       11838        7160       12808         174
0000040        2752        9374       13832       10548
0000060        6182        9896        6406       10334
0000100
linux> mpirun -n 4 file-io2 in50 out50
P0: block=13, items=0, psum = 62864
P1: block=13, items=13, psum = 57047
P3: block=12, items=38, psum = 54413
P2: block=12, items=26, psum = 43833
```

Submission

Write a short report summarizing your work. Submit it with your two new programs through the "Lab 8" folder on Canvas. The submission deadline is the end of tomorrow (Friday).