Final Project Tutorial, Part 1: Reading with MPI I/O

Instructions : Complete the tutorial using C.

To use C, edit only the file `C/main.cpp`.

If you get confused with GitHub/CMake logistics, refer back to Homework 0 here.

1 Initialization

In C: To use MPI, the first line of your main method (`tutorial_main(...)` should be the following:

```
MPI_Init(&argc, &argv);
```

2 Finalize

In C: When using MPI, you must finalize the MPI program. This should be the last line of your main method (immediately before your return statement).

```
MPI_Finalize();
```

3 Reading a Binary File

Edit the `readCols()` method. Upon compilation of the project, a random n x n matrix of doubles will be written to file titled `matrix.bin`, but the filename and dimension will be passed to this function as an argument. You will need to open this file with `MPI_File_open`. Then, define an MPI datatype that describes a regularly strided, non-contiguous memory layout. This will allow each column to be read to a contiguous buffer - **this tutorial requires that you read the matrix by columns.**
In C: Use an `MPI_Offset` and your previously defined `MPI_Datatype` to create a file view that limits the data accessible in the file to the process, allowing a single column to be read into a buffer.

```
MPI_Offset disp = (MPI_Offset) i * sizeof(double);
MPI_File_set_view(file, disp, MPI_DOUBLE, myDatatype, "native",
    MPI_INFO_NULL);
MPI_File_read_all(file, myBuffer, n, MPI_DOUBLE, &status);
```

At the end of your method, after transposing your matrix and reducing, make sure to free any allocated variables and MPI datatypes. You will also need to close the file with `MPI_File_close`.

4 Update Main Method

After writing the function, incorporate it into your main method (`tutorial_main`). This function should be called after initialization but before finalizing. You may assume that the user passes the filename and matrix dimensions as command line arguments or specify it yourself. Print the matrix returned by the function to ensure that the function is correctly reading the matrix.

5 Compile Code

Compile your code with:

```
sh compile.sh
```

6 Run Your Code

I have provided an example for you at `examples/example.cpp`, which will call your `tutorial_main` method. To run this example, first:

```
cd build/examples
mpirun -n <np> ./example matrix.bin 16
```

where `<np>` is replaced by the number of processes on which you want to run. The project is set to generate a matrix of size 16 within the examples directory. Verify that the matrix printed has dimension n.

```
mpirun -n 4 ./example matrix.bin 16
```

The transposed matrix should print to the screen. **Make sure to try multiple different, even process counts.** Depending on your MPI implementation, you may need to pass the oversubscribe flag if you surpass the number of available hardware threads on your system, e.g.

```
mpirun -n 16 --oversubscribe ./example matrix.bin 16
```

7 Check for Correctness

To check that your code is working, do the following:

1. Make sure all tests are passing locally

   ```
   sh compile.sh
   cd build
   make test ARGS="-L read"
   ```

2. Push all changes to GitHub

3. Check that your GitHub actions are passing

The End.