Project 2: Parallel implementation of search based on MPI and OpenMP

The description of this project is identical to that of the first project. The only difference is that the implementation should use both OpenMP and MPI. So, essentially your OpenMP implementation will remain the same, and you have to implement MPI on top of that. Please contact me if you have not done the first project, and do not want to implement the OpenMP part.

Note:

- You should read the project description from Project 1.
- The total mark for this project is 25. The due date is October 20, 2023, 11:59 pm.
- The project can be done either in a group of two, or individually. It is your responsibility to find a partner if you want to do the project in a group.
- Only one group member should submit the project.
- The submission should have two parts, a code file and a report in pdf format. You should zip these two files and submit the zip file.
- All submission is through cssubmit. I will create a submission folder soon.

1 Project deliverables

- The first deliverable is an experiment in communication using MPI. You should generate the fish data in only the master process, and distribute it equally among the workers. I have been successful in requesting for 4 nodes in Setonix. The master node should distribute the fish data to all four worker nodes (including itself). Then the workers should send back the data to the master (the master should also send back its part of the data to itself). You should output this data to two files.

First, immediately after the master generates the data. And then after master gets back all the data. Note that, there is no need to do any computation. I just want to make sure that you have understood message passing correctly in MPI. I will check these two files by applying the unix `diff` command, that outputs nothing if the files are identical. I want to see that the two files you write are identical. Since the coordinates of the fish(es) are double numbers, you should use the `%f` format to write to files. Occassionally the least significant digits of double precision or floating point numbers may change during message transmission. You need not worry about that. You can read about the output of the `diff` command for better understanding. Please see the code sample below on how to open files and write to files in C.

- The second deliverable is simulating the fish school behaviour using MPI and OpenMP. You have to write MPI code and the OpenMP code will run underneath the MPI code. You should do experiments as you did for OpenMP code. You can vary the number of MPI modes only 2,3 or 4. Also, you should experiment with the number of threads under each MPI node. It is up to you to decide how you will vary the number of threads, as you have already good experience from the first project.

2 A sample C code for file writing

The following is a simple C code where there are two MPI processes and two threads under each process. Thread 0 of process 0 writes the two files. I think you can also do the same for the first task above.

```
#include<stdio.h>
#include<omp.h>
#include <mpi.h>

int main(int argc, char* argv[])
{
FILE *fp1, *fp2, *fopen();
int process_id, number_of_processes;
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&process_id);
MPI_Comm_size(MPI_COMM_WORLD,&number_of_processes);
#pragma omp parallel
 {
printf("Hello world, I am process %d among %d processes
  and thread_id %d among %d threads\n",process_id,number_of_processes,
  omp_get_thread_num(),omp_get_num_threads());

if ((process_id==0)&&(omp_get_thread_num()==0))
{
  fp1 = fopen("out1.txt","w+");
  fp2 = fopen("out2.txt","w+");
  fprintf(fp1,"I am thread %d of process %d\n",
                 omp_get_thread_num(),process_id);
  fprintf(fp2,"I am thread %d of process %d\n",
                 omp_get_thread_num(),process_id);
}
}
MPI_Finalize();
}
```