Assignment 4

1 Synopsis

Please read the document, ***General Requirements Programming Assignments***, posted on the course website and make sure you follow them when you do this assignment. This assignment is to write your third MPI program, which is to solve the ***room assignment problem***. The room assignment problem is described in Chapter 8 of the course Lecture Notes, but it is restated here.

We are given a set of $n/2$ rooms, each of which has a two-person occupancy, and a set of n college freshmen. n must be an even number. Each student has completed a survey of their personal preferences and an artificial intelligence algorithm was then run to create a table of incompatibilities between pairs of students. This is codified into an $n \times n$ symmetric ***incompatibility matrix*** C such that $C_{i,j}$ is a real number expressing the extent to which students i and j will be ***incompatible*** as roommates. The objective is to assign students to rooms so that every student is assigned to a room and all rooms have two students, while minimizing the total incompatibility. A solution is therefore a finite function $a : [0, n-1] \to [0, n/2-1]$ that maps each student into one of the $n/2$ rooms. We can represent the function a as an array. We let

$$f_C(a,i,j) = \begin{cases} C_{i,j} & \text{if } a(i) = a(j) \\ 0 & \text{otherwise} \end{cases}$$

We parameterize f by C to indicate that its value depends upon the incompatibility matrix C. The objective function $F_C(a)$ is

$$F_C(a) = \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j\neq i} f_C(a,i,j)$$

Each different assignment a leads to a different cost, and the problem amounts to finding an array a that minimizes the value of this cost function. You assignment is to write a parallel program using MPI that, given the incompatibility matrix, finds an assignment that minimizes the cost, using the ***simulated annealing method*** described in the notes. The notes also contain a pseudocode description of a sequential algorithm to solve this problem.

2 Program Invocation, Input, and Output

2.1 Usage

The program should be named `assign_rooms`. It expects a single command line argument, which is the pathname of a file containing the incompatibility matrix. The correct usage is therefore

```
assign_rooms <0|1>  <compatibility_file>
```

where

- if the first argument is `0`, the program's behavior is not repeatable, and if the first argument is `1`, it is repeatable.
- the matrix file uses the same format as is used in other assignments that work with matrices: the file is a ***binary file*** and it starts with two integers specifying the number of rows followed by the number of columns of the matrix, after which the elements of the matrix are stored in row-major order as ***double-precision floating point numbers*** (C type `double`). The matrix may be of any size.

2.2 Error Handling

The program must detect the following errors, report them as indicated, and terminate:

- Report a missing command line argument on the standard output stream.
- If the first argument is something other than 0 or 1 it is also an error that should be reported on the standard output stream.
- Report if the file cannot be opened on the standard output stream.
- Report any matrix that is not square on the standard output stream.
- Report any matrix that is not symmetric on the standard output stream.
- Although the diagonal elements should not have any effect on the program's behavior, it should also reject any square matrix that has a non-zero diagonal element and report it on the standard output stream.

If during processing the program results in an error such as being unable to allocate memory or other programmatic failures, it should print a message on ***standard error*** that it failed and it should clean up all MPI processes and exit. Notice that usage errors go to standard output, but programming errors go to standard error.

2.3 Output

With a valid matrix argument, the program should write to standard output the solution that it found to minimize the objective function and the value of that function, both as plain text. The objective function value is printed on the first line in fixed decimal format, with 5 digits of precision. The assignment is printed on the next line, as an array of integers separated by white space, terminated by a newline. Room numbers can be zero-based for simplicity. For example, with hypothetical numbers from a file with 10 students a run should look like

```
$ assign_rooms  1 compatfile
2.54901
4 0 4 3 1 1 2 0 3 2
```

3 Program Design

The room assignment problem is a good candidate for parallelization because simulated annealing may not find the optimal solution. It is possible for the algorithm, when run multiple times, to converge to multiple, different solutions, not necessarily optimal. Therefore, it makes sense to run many instances of the same algorithm with different seeds for the random number generator. With MPI it is possible to write a highly parallel program that has a much higher probability of finding the optimal solution than a sequential one in the same amount of time. Your program should have each process compute using the same algorithm, the same method of random number generation, but with each process being seeded independently. Consult the notes for how to do this.

The program should check the value of the first command line argument. If it is 1, then all seeds given to processes should be created from an initial seed that is hard-coded into the program. If it is zero, the seeds should come from an initial seed that is obtained by calling `time(NULL)`, which requires including `<time.h>`. The root process should do this and then send distinct seeds to the other processes, which can use these to initialize their lag tables.

See the demo program `gen_exponential.c` for an example of how to use the `time()` function.

4 Implementation Requirements

You must design and implement a parallel program to obtain an optimal solution using the simulating annealing method. The random number generator should use a function based on a lagged Fibonacci method, such as the C library's `random()` function. Other requirements are:

1. If the program is run with the repeatability argument equal to 1, then the random number generation should be repeatable from one run to another and the sequences of numbers that each process gets should be independent, uncorrelated, and repeatable as well. This means that the seed(s) are fixed for each run. If the repeatability argument is equal to 0, then then the random number generation should not be repeatable from one run to another. This means that the seeds can be chosen at run-time dynamically, such as by using the time or the process id assigned to the process, etc.
2. If the program is run with the repeatability argument equal to 1, the program must produce the same results when run with the same number of processes. The output will be different as the number of processes is increased because a larger number of processes might result in a better solution being found.
3. Only the root process can perform input, output, and error handling. It is an error if any other process does so.
4. The program must produce correct results regardless of the size of the matrix file. There is no upper bound on the size of the matrix.
5. The program must work correctly regardless of how many tasks are run.
6. The program must be documented and written to comply with the requirements stated in the ***General Requirements Programming Assignments*** referred to above.

5 Testing

You should create some small compatibility matrices that have obvious optimal solutions. For example, with this matrix

```
0    1    5   10
1    0    10   5
5    10   0    1
10   5    1    0
```

it is clear that students 0 and 1 should be roommates and 2 and 3 should be roommates. The program should be able to find this. You can create larger matrices that have this same pattern.

6 Program Grading Rubric

The program will be graded based on the following rubric out of 100 points:

- The program must compile and run on any `cslab` host. If it does not compile and link on any `cslab` host, it loses 80 points.
- **Correctness** (60 points)
  - The program should do exactly what is described above. Incorrect output, incorrectly formatted output, missing output, or output containing other characters are all errors.
  - It should handle errors correctly.

- **Performance** (16 points)

  When run with successively greater numbers of processes, the accuracy should increase (the objective function value should be smaller). Check that this behavior occurs on average.

- **Design and clarity** (8 points)

  Is the program organized clearly? Are functions and variables designed in an appropriate way? Are the tasks divided appropriately? If not, the program will lose points.

- **Compliance with the Programming Rules** (16 points)

  Are all of the rules stated in that document observed? Programs that violate them will lose points accordingly.

7 Submitting the Homework

This project is due by the end of the day (i.e. 11:59PM, EST) on December 2, 2019. Follow these instructions precisely to submit it.

Your program should be a single source code file, written in C. It must have a `.c` suffix. It does not matter what base name you give it because the `submithwk` program will change its base name anyway to the form `username_hwk4.c` (or more accurately, `username_hwk4.`*XXX*, where *XXX* is the suffix you gave it.)

Assuming this file is in your current working directory and is named `myhwk4.c` you submit by entering the command

```
submithwk_cs49365 -t  4  myhwk4.c
```

The `-t` tells the command it is a plain text file. The program will copy your file into the `hwk4` subdirectory

```
/data/biocs/b/student.accounts/cs493.65/hwks/hwk4/
```

and if it is successful, it will display the message, "`File username_hwk4.c successfully submitted.`"

You will not be able to read this file, nor will anyone else except for me. But you can double-check that the command succeeded by typing the command

```
ls -l /data/biocs/b/student.accounts/cs493.65/hwks/hwk4/
```

and making sure you see a non-empty file there.

If you put a solution there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.