CMPE 300
ANALYSIS OF ALGORITHMS

MPI PROGRAMMING PROJECT

ZUHAL DİDEM AYTAÇ - 2018400045
NURİYE ÖZLEM ÖZCAN ŞİMŞEK
26 January 2021

INTRODUCTION

The project implements parallel programming using the MPI library. The aim of the program is to implement feature selection. The program takes an input file as command line argument and performs desired operations, printing the output to the console.

The library and compiler versions are as below:

```
Zuhal-MacBook-Pro:desktop didemaytac$ g++ --version
Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include-dir=/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/c++/4.2.1
Apple clang version 12.0.0 (clang-1200.0.32.2)
Target: x86_64-apple-darwin19.6.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
Zuhal-MacBook-Pro:desktop didemaytac$ mpirun --version
mpirun (Open MPI) 4.0.5
```

PROGRAM EXECUTION

The code is compiled with mpic++ as:

mpic++ -o cmpe300_mpi_2018400045 ./cmpe300_mpi_2018400045.cpp

The code is run as:

mpirun --oversubscribe -np <P> cmpe300_mpi_2018400045 <inputfile>

The details of the input is as specified in the project description. The input path and number of processes (P) is given as a command line argument.

The output is as defined in the project description and is printed to the console.

PROGRAM STRUCTURE

The program implements openMPI, where the process with rank 0 is considered to be the root. The task of the root process is to scatter the *N* input lines among the *P-1* other processes (excluding itself) equally and in the order of increasing process rank.

The root opens and reads the input file and assigns necessary values to the arguments. Then, the root sends the input file's variables (*P,N,A,M* and *T* namely) to other processes via MPI_Bcast.

Then, necessary variables are initialized. The input is read by the root to an array named *input*. Then, it is copied into another array named *scatter_input* with dumb values appended to the beginning of the array. This is done because the root process divides the input among processes using MPI_Scatter and MPI_Scatter also scatters to the root itself.

After the input is scattered as requested, the given algorithm is implemented for the processes except the root. By each process, *M* iterations are performed. In each iteration, the process selects a target instance. The target is selected from the scattered input of that process, starting from the first, up to *M*th, in increasing order of instance indexes.

In each iteration, the aim is to identify the nearest hit and miss to the instance that is currently the target. For that purpose, Manhattan distance between all other instances and the current target is calculated into an array named *Manhattan* by traversing all instances. In each step of the traversal, another loop traverses all features of the instance being traversed and adds the absolute difference between that feature's value among current instance and the target instance. Also, at this step, the minimum and maximum values for any feature is being searched.

After the iteration of features is completed, the final distance has been calculated. Then it is checked if the current instance can be the nearest hit or nearest miss for the current target (that is whether the instance is the one with smallest Manhattan distance and has the same -or opposite for miss- class variable as the target).

Now that the nearest hit and miss instances are identified and minimum and maximum values for each feature is obtained, the weights of the features is updated with the given specific formula.

After the explained process is completed *M* times, the final weights for all features has been calculated. The slaves choose *T* features with highest weights and prints them to the console in increasing order of feature numbers. All slaves send the array named *res* to the root process using MPI_Gather.

When root receives top *T* features from all the slaves, it takes the unification. Then, the master prints its output which is the unification of all slaves outputs. The program ends with MPI_Finalize.

DIFFICULTIES ENCOUNTERED

I was using MPI for the first time but the PS and online contents were clarifying enough. As I have dealt with multiprocess programs in CmpE322 - Operating Systems course, it was easy to get used to the parallel programming concept.

However, I am using a MacOS PC and CLion IDE and both had some issues. MPI gave the following error : "A system call failed during shared memory initialization that should not have. It is likely that your MPI job will now either abort or experience performance degradation."

Later I found out that it is a MacOS related issue and can be solved when the code is compiled as

mpirun --mca shmem posix --oversubscribe -np <P> cmpe300_mpi_2018400045 <inputfile>

CONCLUSION

The code I've implemented successfully manages parallel programming and mpi tasks. The outputs for the provided test cases are correct. The code is compiling and working. A compile-run-output example is provided below.

```
Zuhal-MacBook-Pro:desktop didemaytac$ mpic++ -o cmpe300_mpi_2018400045 ./cmpe300_mpi_2018400045.cpp
Zuhal-MacBook-Pro:desktop didemaytac$ mpirun --mca shmem posix --oversubscribe -np 6 cmpe300_mpi_2018400045 mpi_project_dev0.tsv
Slave P1 : 0 5
Slave P2 : 0 2
Slave P3 : 5 7
Slave P4 : 5 7
Slave P5 : 0 3
Master P0 : 0 2 3 5 7
Zuhal-MacBook-Pro:desktop didemaytac$ mpirun --mca shmem posix --oversubscribe -np 6 cmpe300_mpi_2018400045 mpi_project_dev1.tsv
Slave P2 : 3 5 7 8 18
Slave P1 : 4 5 8 10 18
Slave P3 : 0 3 12 13 16
Slave P4 : 0 3 4 5 16
Slave P5 : 0 5 6 11 18
Master P0 : 0 3 4 5 6 7 8 10 11 12 13 16 18
Zuhal-MacBook-Pro:desktop didemaytac$ mpirun --mca shmem posix --oversubscribe -np 11 cmpe300_mpi_2018400045 mpi_project_dev2.tsv
Slave P1 : 4 5 8 11 18 21 30 32 44 49
Slave P2 : 0 3 4 8 11 13 21 26 32 39
Slave P3 : 0 3 4 5 11 18 21 26 46 47
Slave P4 : 0 3 11 14 21 28 30 32 39 40
Slave P5 : 0 3 5 11 16 18 21 26 35 47
Slave P6 : 0 3 5 8 16 21 26 30 35 47
Slave P7 : 0 2 3 4 5 16 18 21 32 45
Slave P8 : 0 3 11 18 21 24 26 39 44 46
Slave P9 : 0 3 5 11 18 21 26 30 35 47
Slave P10 : 0 5 8 11 16 18 20 21 30 46
Master P0 : 0 2 3 4 5 8 11 13 14 16 18 20 21 24 26 28 30 32 35 39 40 44 45 46 47 49
```

As explained in Difficulties Encountered section, I used " - - mca shmem posix" option to solve the MPI-MacOS issue.