Bilkent University

Department of Computer Engineer

CS426 Parallel Computing

Project 1 Report

Serhat Aras

21401636

1. Implementation and design choice

In this project, I am implementing message passing interface for various tasks. To achieve this, i need to install / include OpenMPI Framework to the gcc compiler to make the implementation run on the parallel manner. This way, ı was able to communicate in between the processes on the system.

For most of the part of the project, I used MPI_Send, MPI_Recv, MPI__Bcast of the openMPI framework. The coverage of this extends to 5 sub projects. These are namely:

- sum-serial.c
- sum-mpi-ppv1.c
- sum-mpi-ppv2.c
- matmult-serial.c
- matmult-mpi-1d.c

• sum-serial.c

•

I didn't use any openMPI function. In this program, we are basically calculating the overall sum of the given input, passed by argv[1], then computing the overall time consumption for further anaylsis.

```
neo@vulcan:~/Desktop/21401636/Part1$ ./serial 1-10
The number of data in the file is 100, total sum is 5050
---------------------------
Execution Time of the sum-serial.c ---> 0.000053
---------------------------
```

• sum-mpi-ppv1.c

This program is the MPI implementation of the first program. I tried to divide the work equally to all processes for any given process. Then each of the sub processes calculates their own respectful sub inputs. In this program, I used the MPI_Send and MPI_Recv functions in order to establish links between processes.

```
neo@vulcan:~/Desktop/21401636/Part1$ mpirun -oversubscribe -np 9
mpi1 1-10
Reading data from Master . . .
---> Data size is 100
Reading completed . . .
Workers:8, DataPerWorker:12, Data to Master:4
------
Distributing data to Workers . . .
Receiving calculations from Workers . .
[1] Sends partial sum : 78 . . .
[2] Sends partial sum : 222 . . .
[3] Sends partial sum : 366 . . .
[4] Sends partial sum : 510 . . .
[5] Sends partial sum : 654 . . .
[6] Sends partial sum : 798 . . .
[7] Sends partial sum : 942 . . .
[8] Sends partial sum : 1086 . . .
Sum: 5050
Execution Time of the sum-serial.c ---> 0.035427
Execution Time of the sum-serial.c ---> 0.031921
---------------------------
Execution Time of the sum-serial.c ---> 0.034770
---------------------------
Execution Time of the sum-serial.c ---> 0.029042
---------------------------
Execution Time of the sum-serial.c ---> 0.030534
---------------------------
Execution Time of the sum-serial.c ---> 0.031023
---------------------------
Execution Time of the sum-serial.c ---> 0.029854
---------------------------
Execution Time of the sum-serial.c ---> 0.034773
---------------------------
Execution Time of the sum-serial.c ---> 0.029083
---------------------------
```

- **sum-mpi-ppv2.c**

This program is very similar to the program described above. However, this program is featuring MPI__Bcast function and furthermore, this program does not calculates sub-summations, instead all processors calculates summation of the distributed data by the master process.

```
neo@vulcan:~/Desktop/21401636/Part1$ mpirun -oversubscribe -np 9
mpi1 1-10
Reading data from Master . . .
---> Data size is 100
Reading completed . . .
Process[2] calculates the overall sum: 5050
Process[3] calculates the overall sum: 5050
Process[7] calculates the overall sum: 5050
Process[8] calculates the overall sum: 5050
Process[0] calculates the overall sum: 5050
Process[5] calculates the overall sum: 5050
Process[6] calculates the overall sum: 5050
Process[4] calculates the overall sum: 5050
Process[1] calculates the overall sum: 5050
Execution Time of the sum-serial.c ---> 0.038426
Execution Time of the sum-serial.c ---> 0.038011
---------------------------
Execution Time of the sum-serial.c ---> 0.038307
---------------------------
Execution Time of the sum-serial.c ---> 0.031571
---------------------------
Execution Time of the sum-serial.c ---> 0.038322
---------------------------
Execution Time of the sum-serial.c ---> 0.036210
---------------------------
Execution Time of the sum-serial.c ---> 0.038225
---------------------------
Execution Time of the sum-serial.c ---> 0.034331
---------------------------
Execution Time of the sum-serial.c ---> 0.038309
```

- **matmult-serial.c**

In this program, i calculated the matrix multiplication of 2 matrices passed by the command line arguments argv[1], [argv[2] and writes the calculated result in to the given destination on argv[3].

Output of the program is0:

```
neo@vulcan:~/Desktop/21401636/Part2$ ./serial mat1 mat2 mat_res
- - - - ->Matrix 1
        Reading matrix completed . . .
          +++Transpose of Matrix Completed! . . .
- - - - ->Matrix 2
        Reading matrix completed . . .
          +++Transpose of Matrix Completed! . . .

+ + + ->Multiplication Complete!
+ + ->File Write Completed!
+ ->Execution Finished!
------------------------------------------------------
Execution Time of the sum-serial.c ---> 0.001343
------------------------------------------------------
```

- **matmult-mpi-1d.c**

In this program, i calculated the matrix multiplication of 2 matrices passed by the command line arguments argv[1], [argv[2] and writes the calculated result in to the given destination on argv[3]. However, this program must distributed the inputs to the processors. In the program, I used MPI_Send and MPI_Recv functions in order to establish links between processes. However, after collecting the multiplication datas from the processors with MPI_Recv() function, I cannot establish the access over the returned sub result matrices, I cannot finish the complete program. In this program, I can devide the work to master and the worker process equally and calculate the matrix multiplication of the passed sub matrices in the process. In this part I passed the dimentions and data of the sub matrices to the workers from master by using MPI_Send and MPI_Recv functions.

If the current process is (pid == 0) equal to master, I handled all of the preliminary preparations in order to distribute the sub matrices all over the processes. The output of the run can be found here.

```
neo@vulcan:~/Desktop/21401636/Part2$ mpirun -oversubscribe -np 9 mpi1 mat1 mat2 mat_mpi_res
- - - - ->Matrix 1
        Reading matrix completed . . .
          +++Transpose of Matrix Completed! . . .
- - - - ->Matrix 2
        Reading matrix completed . . .
          +++Transpose of Matrix Completed! . . .
Process[2] Divider 10, dimention 30
Process[3] Divider 10, dimention 30
Process[4] Divider 10, dimention 30
Process[1] Divider 10, dimention 30
Process[7] Divider 10, dimention 30
Process[8] Divider 10, dimention 30
Process[5] Divider 10, dimention 30
Process[6] Divider 10, dimention 30
+++Content of the matrix
[vulcan:16387] *** Process received signal ***
[vulcan:16387] Signal: Segmentation fault (11)
[vulcan:16387] Signal code:  (128)
[vulcan:16387] Failing at address: (nil)
[vulcan:16387] [ 0] /lib/x86_64-linux-gnu/libc.so.6(+0x41100)[0x7ff1d0b73100]
[vulcan:16387] [ 1] mpi1(+0x12ee)[0x563eb21c22ee]
[vulcan:16387] [ 2] mpi1(+0x208a)[0x563eb21c308a]
[vulcan:16387] [ 3] /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xeb)[0x7ff1d0b5609b]
[vulcan:16387] [ 4] mpi1(+0x11da)[0x563eb21c21da]
[vulcan:16387] *** End of error message ***
--------------------------------------------------------------
Primary job  terminated normally, but 1 process returned
a non-zero exit code. Per user-direction, the job has been aborted.
--------------------------------------------------------------
mpirun noticed that process rank 0 with PID 0 on node vulcan exited on signal 11 (Segmentation fault).
```

2. Performance Comparison & Observations

In this project, I cannot experience any performance improvements on the first part using MPI. Furthermore, the sequential program I more faster then the MPI versions for this problem. The difference of the programs are vary among them. Serial program finishes 0.000053 seconds, sum-mpi-ppv1.c finishes around 0.035427 seconds and sum-mpi-ppv2.c is finished around 0.038426 seconds. This shows that we are not gaining speed by seperating this problem in to many subproblems. Furthermore, in the second part, I expect a gain on execution time while using MPI implementation compared to serial one theoretically, I cannot managed to run the matmult-mpi-1d.c therefore, I cannot compare the experiment results. For the matmult-serial.c overall execution finished around 0.001343 seconds. I expect the MPI version will be little bit faster then the sequential one.

Hardware Info of the system:

Computer (MSI GT62 7RE)	
Processor	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Memory	16381MB
Graphic Card	NVidia Gtx1070 (8 GB)
Operating System	Ubuntu 18.10
Kernel Version	4.18.0-16-generic
Dev. Env	Visiual Studio Code
GCC Comp.	Using built-in specs.

Root directory contains 6 files used in the test of the program :

- 1-10
- rnd1
- rnd2

are used in the part 1 as:

```
-~neo@vulcan:~/Desktop/21401636/Part2$ ./serial 1-10
-~neo@vulcan:~/Desktop/21401636/Part2$ mpirun -oversubscribe -np 9 mpi 1-10
```

- mat1 (Input)
- mat2 (Input)
- mat3 (Output)

are used in the part 2 as:

```
-~neo@vulcan:~/Desktop/21401636/Part2$ mpirun -oversubscribe -np 9 mpi1 mat1 mat2
mat3
```