

DTS202TC Foundation of Parallel Computing

Lab 2: MPI

This lab weight 30 marks of the A3 lab report, please save all screenshots of activities, source code for future reference.

Task 1 (8 marks)

OpenMPI is one of the Message Passing Interface (MPI) implementations. First of all, install **OpenMPI** on your computer.

Similar to Lab 1, Use **Homebrew** on Mac or **Cygwin** on Windows.

On Mac:

```
1 brew install openmpi
```

On Windows:

```
1 Launch the setup-x86_64.exe downloaded from previous lab session, make sure to select and  
install openmpi-4.* in the Select Packages dialog.
```

Verify the installation:

```
1 mpicc --version
```

Task 2 (2 marks)

Type the following source code in a `mpi_hello.c` file.

Compile the source code by:

```
1 mpicc -g -Wall -o mpi_hello mpi_hello.c
```

Run the program by:

```
1 mpiexec -n 4 ./mpi_hello
```

Run it 3 times, observe the outputs.

```

1  #include <stdio.h>
2  #include <string.h>  /* For strlen */
3  #include <mpi.h>     /* For MPI functions, etc */
4
5  const int MAX_STRING = 100;
6
7  int main(void) {
8      char    greeting[MAX_STRING];
9      int     comm_sz; /* Number of processes */
10     int     my_rank; /* My process rank */
11
12     MPI_Init(NULL, NULL);
13     MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
14     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
15
16     if (my_rank != 0) {
17         sprintf(greeting, "Greetings from process %d of %d!",
18             my_rank, comm_sz);
19         MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
20             MPI_COMM_WORLD);
21     } else {
22         printf("Greetings from process %d of %d!\n", my_rank,
23             comm_sz);
24         for (int q = 1; q < comm_sz; q++) {
25             MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
26                 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
27             printf("%s\n", greeting);
28         }
29     }
30     MPI_Finalize();
31     return 0;
32 } /* main */

```

Figure 1: For task 2

Task 3 (20 marks)

The following code estimates the value of the mathematical constant PI. Write an MPI version of `estimate_pi` that uses all available processes to do the work. Use `MPI_Send` and `MPI_Recv` to communicate between processes.

```

1  /*Estimate PI*/
2
3  #include <stdio.h>
4  #include <math.h>
5  #include <stdlib.h>
6  #include <sys/time.h>
7
8  /* The argument now should be a double (not a pointer to a double) */
9  #define GET_TIME(now) { \
10     struct timeval t; \
11     gettimeofday(&t, NULL); \
12     now = t.tv_sec + t.tv_usec/1000000.0; \
13 }
14

```

```

15 double serial_pi(long long n) {
16     double sum = 0.0;
17     long long i;
18     double factor = 1.0;
19
20     for (i = 0; i < n; i++, factor = -factor) {
21         sum += factor/(2*i+1);
22     }
23     return 4.0*sum;
24 }
25
26
27 int main(int argc, char** argv) {
28     double start, finish;
29     GET_TIME(start);
30     double estimate_of_pi = serial_pi(1000000000);
31     printf("\nEstimated of pi:  %1.10f.\n", estimate_of_pi);
32     GET_TIME(finish);
33
34     printf("\nActual value of pi:  %1.10f.\n\n", atan(1)*4);
35     printf("The elapsed time is %e seconds\n", finish-start);
36 }

```

Analysis the performance of the serial and MPI versions by simply print out the total execution time. Also compare the performance with different number of processes.