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:

```
brew install openmpi
```

On Windows:

```
Launch the setup-x86_64.exe downloaded from previous lab session, make sure to select and
    install the following packages in the Select Packages dialog.
- openmpi,
- libopenmpi-devel,
- libopenmpi,
- libhwloc-devel,
- libevent-devel,
- zlib-devel,
- openssl
```

Verify the installation:

```
mpicc --version
```

Task 2 (2 marks)

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

Compile the source code by:

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

Run the program by:

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

Run it 3 times with different n, observe the outputs.

```
#include <stdio.h>
#include <string.h>   /* For strlen             */
#include <mpi.h>      /* For MPI functions, etc */

const int MAX_STRING = 100;

int main(void) {
   char       greeting[MAX_STRING];
   int        comm_sz;  /* Number of processes */
   int        my_rank;  /* My process rank     */

   MPI_Init(NULL, NULL);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

   if (my_rank != 0) {
      sprintf(greeting, "Greetings from process %d of %d!",
            my_rank, comm_sz);
      MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
            MPI_COMM_WORLD);
   } else {
      printf("Greetings from process %d of %d!\n", my_rank,
          comm_sz);
      for (int q = 1; q < comm_sz; q++) {
         MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
            0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
         printf("%s\n", greeting);
      }
   }

   MPI_Finalize();
   return 0;
}  /* 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.

```
/*Estimate PI*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <sys/time.h>

/* The argument now should be a double (not a pointer to a double) */
#define GET_TIME(now) { \
   struct timeval t; \
   gettimeofday(&t, NULL); \
   now = t.tv_sec + t.tv_usec/1000000.0; \
}

```

```c
double serial_pi(long long n) {
   double sum = 0.0;
   long long i;
   double factor = 1.0;

   for (i = 0; i < n; i++, factor = -factor) {
      sum += factor/(2*i+1);
   }
   return 4.0*sum;

}

int main(int argc, char** argv) {
   double start, finish;
   GET_TIME(start);
   double estimate_of_pi = serial_pi(1000000000);
   printf("\nEstimated of pi:  %1.10f.\n", estimate_of_pi);
   GET_TIME(finish);

   printf("\nAcutal value of pi:  %1.10f.\n\n", atan(1)*4);
   printf("The elapsed time is %e seconds\n", finish-start);
}
```

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.