MANIPAL INSTITUTE OF TECHNOLOGY
Manipal – 576 104

DEPARTMENT OF COMPUTER SCIENCE & ENGG.

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

CERTIFICATE

This is to certify that Ms./Mr. ……...…… Reg. No. …..…… Section: …… Roll No: ……... has satisfactorily completed the lab exercises prescribed for Parallel Programming Lab [CSE 3263] of Third Year B. Tech. Degree at MIT, Manipal, in the academic year 2023-2024.

Date: …….....

Signature	Signature
Faculty in Charge	Head of the Department

CONTENTS

Course Objectives

- Learn different APIs used in MPI for point to point, collective communications and error handling
- Learn how to write host and kernel code for device neutral architecture using OpenCL
- Learn how to write host and kernel code in CUDA for nVIDIA GPU card
- To develop the skills of design and implement parallel algorithms using different parallel programming environment

Course Outcomes

At the end of this course, students will be able to

- Write MPI programs using point-to-point and collective communication primitives.
- Solve and test OpenCL programs using GPU architecture
- Develop CUDA programs for parallel applications

Evaluation plan

- Internal Assessment Marks : 60%
    - Continuous Evaluation : 60%

      Continuous evaluation component (for each evaluation):10 marks

      The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce.

- End semester assessment of 2 hours duration: 40 %

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session.
2. Be in time and follow the institution dress code.
3. Must Sign in the log register provided.
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum.
6. Students must come prepared for the lab in advance.

In- Lab Session Instructions

- Follow the instructions on the allotted exercises.
- Show the program and results to the instructors on completion of experiments.
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required.

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- Observation book should be complete with program, proper input output clearly showing the parallel execution in each process. Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
  - Solved example
  - Lab exercises - to be completed during lab hours
  - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed during the repetition class with the permission of the faculty concerned but credit will be given only to one day's experiment(s).
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

Lab No 1: **Date:**

Introduction to execution environment of MPI

Objectives:

In this lab, student will be able to

1. Understand the execution environment of MPI programs
2. Learn the various concept of parallel programming
3. Learn and use the Basics API available in MPI

I. Introduction

In order to reduce the execution time work is carried out in parallel. Two types of parallel programming are:

- Explicit parallel programming
- Implicit parallel programming

Explicit parallel programming – These are languages where the user has full control and has to explicitly provide all the details. Compiler effort is minimal.

Implicit parallel programming – These are sequential languages where the compiler has full responsibility for extracting the parallelism in the program.

Parallel Programming Models:

- Message Passing Programming
- Shared Memory Programming

Message Passing Programming:

- In message passing programming, programmers view their programs (Applications) as a collection of co-operating processes with private (local) variables.
- The only way for an application to share data among processors is for programmer to explicitly code commands to move data from one processor to another.

Message Passing Libraries: There are two message passing libraries available. They are:

- PVM – Parallel Virtual Machine
- MPI – Message Passing Interface. It is a set of parallel APIs which can be used with languages such as C and FORTRAN.

Communicators and Groups:

- MPI assumes static processes.
- All the processes are created when the program is loaded.
- No process can be created or terminated in the middle of program execution.
- There is a default process group consisting of all such processes identified by **MPI_COMM_WORLD**.

III. MPI Environment Management Routines:

MPI_Init: Initializes the MPI execution environment. This function must be called in every MPI program, must be called before any other MPI functions and must be called only once in an MPI program.

```
MPI_Init (&argc,&argv);
```

MPI_Comm_size: Returns the total number of MPI processes to the variable size in the specified communicator, such as MPI_COMM_WORLD.

```
MPI_Comm_size(Comm,&size);
```

MPI_Comm_rank: Returns the rank of the calling MPI process within the specified communicator. Each process will be assigned a unique integer rank between 0 and size - 1 within the communicator MPI_COMM_WORLD. This rank is often referred to as a process ID.

```
MPI_Comm_rank Comm,&rank);
```

MPI_Finalize: Terminates the MPI execution environment. This function should be the last MPI routine called in every MPI program. No other MPI routines may be called after it.

```
MPI_Finalize ();
```

Solved Example:

Write a program in MPI to print total number of process and rank of each process.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
        int rank,size;

        MPI_Init(&argc,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        printf("My rank is %d in total %d processes", rank, size);
        MPI_Finalize();
        return 0;
}
```

Steps to execute a MPI program is provided in the form of video which is available in individual systems. However, the basic installation steps are given as follows.

// Follow the following steps to Install, Compile and Run MPI programs in Ubuntu O.S

// To install MPI in Ubuntu, execute the following command in command line

$sudo apt-get update; sudo apt-get install mpich

// To edit MPI program, use any text editor such as vim or gedit and create a file with .c extention.

//To Compile MPI program, execute the following command in command line

$mpicc filename.c -o filename.out

//To Run MPI program, execute the following command in command line

$mpirun -np 4 filename.out

Lab Exercises:

1. Write a simple MPI program to find out pow (x, rank) for all the processes where ‘x’ is the integer constant and ‘rank’ is the rank of the process.
2. Write a program in MPI where even ranked process prints “Hello” and odd ranked process prints “World”.
3. Write a program in MPI to simulate simple calculator. Perform each operation using different process in parallel.

4. Write a program in MPI to toggle the character of a given string indexed by the rank of the process. Hint: Suppose the string is HELLO and there are 5 processes, then process 0 toggle ‘H’ to ‘h’, process 1 toggle ‘E’ to ‘e’ and so on.

Additional Exercises:

1. Write a program in MPI to reverse the digits of the following integer array of size 9 with 9 processes. Initialize the array to the following values.

   Input array : 18, 523, 301, 1234, 2, 14, 108, 150, 1928

   Output array: 81, 325, 103, 4321, 2, 41, 801, 51, 8291

2. Write a MPI program to find the prime numbers between 1 and 100 using two processes.