Parallel Processing (PP) Final LAB
FALL (2023)

Ghulam Ishaq Khan Institute of Engineering Sciences and Technology

Time: 90 minutes CS-417L FINAL Exam FALL 2023 Marks: 50

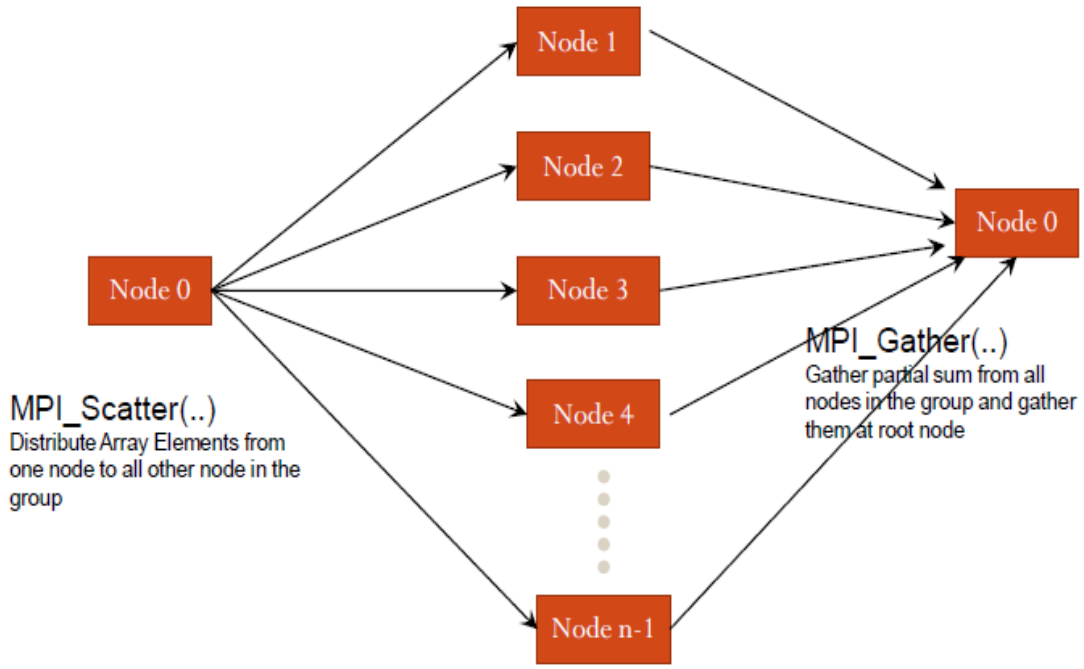
Course: CS-417L	**Subject: PP**
Instructor: MUNEEB BAIG	
NAME:	**REG:**

VETTED BY	
Signature	

Question 01: MPI

Task Description:

1. **Break up a long vector into sub-vectors of equal length. Distribute sub-vectors to processes. Let the processes to compute the partial sums. Collect the partial sums from the processes and add them at root node using collective computation operations.**



2. **The given code represents a simple application involving 2 MPI processes, where one process sends a message, and the other process receives it. However, the code contains an error that could potentially lead to a crash. Identify the incorrect statement in the code and propose a correction. Keep in mind that while the code might not crash in some MPI implementations, it is considered incorrect. What is the incorrect statement, and how would you fix it to ensure proper functionality?**

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

/**
 * @brief This exercise is to find the bug in a simple send-receive.
 **/
int main(int argc, char* argv[])
{
  MPI_Init(&argc, &argv);

  int comm_size;
  MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
  if(comm_size != 2)
  {
    printf("This application must be run with 2 MPI processes.\n");
    MPI_Abort(MPI_COMM_WORLD, -1);
  }

  int my_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  if(my_rank == 0)
  {
    int value_sent = 12345;
    int buffer_size = sizeof(int);
    char* buffer = (char*)malloc(buffer_size);
    MPI_Buffer_attach(buffer, buffer_size);

    printf("[MPI process %d] I send value %d.\n", my_rank, value_sent);
    MPI_Bsend(&value_sent, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);

    MPI_Buffer_detach(&buffer, &buffer_size);
    free(buffer);
  }
  else
  {
    int value_received;
    MPI_Recv(&value_received, 1, MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("[MPI process %d] I received value %d.\n", my_rank, value_received);
  }

  MPI_Finalize();

  return 0;
}
```

Question 02: Cuda

Vector Addition and Multiplication

In this exercise you will add two vectors of the same size and store the result in a third vector. The operation should be done in parallel. Start from the skeleton code in vector_addition_start.cu. Every thread should do the addition of one element of each input vector and store it in the corresponding element of the output vector. Try different vector sizes and different numbers of threads per block.

Note: When done do the exact thing in terms of vector multiplication.

```
#include <stdio.h>
#include <iostream>
#include <chrono>

#include "helpers.h"

using namespace std;

__constant__ int vec_size_d;

/* Kernel doing vector addition with one thread per vector entry  */
__global__ void vector_addition_kernel( int *a, int *b, int *c) {
  /* to do: calculate the index correctly from the blockIdx.x and threadIdx.x
     const int index = ...
   */

  if ( index < vec_size_d ) {
    c[ index ] = a[ index ] + b[ index ];
  }

}


int main(int argc, char *argv[] ) {

  if ( argc != 4 ) {
    cout << "Need three arguments: size of vector, number of threads / block and device to use" << endl;
    return -1;
  }

  const int vec_size_h  = atoi(argv[argc-3]);
  const int n_threads = atoi(argv[argc-2]);
```

```
  const int device_id = atoi(argv[argc-1]);

  /* Chose device to use */
  CUDA_ASSERT( cudaSetDevice(device_id) );

  cout << "Adding vectors of size " <<  vec_size_h << " with " << n_threads << " threads" << endl;

  /* Host memory for the two input vectors a and b and the output vector c */
  int *a_h = new int[vec_size_h];
  int *b_h = new int[vec_size_h];
  int *c_h = new int[vec_size_h];

  for ( int i = 0; i < vec_size_h; i++ ) {
    a_h[i] = i;
    b_h[i] = i;
    c_h[i] = 0;
  }

  /* Device pointers for the three vectors a, b, c */
  int *a_d, *b_d, *c_d;

  /* to do: Allocate memory on the device for b_d and c_d, following the example for a_d */
  CUDA_ASSERT( cudaMalloc( (void**)&a_d, vec_size_h * sizeof(int) ) );


  /* to do: copy the second input vector to the device, following the example of a_d */
  CUDA_ASSERT( cudaMemcpy( a_d, a_h, vec_size_h * sizeof(int), cudaMemcpyHostToDevice ) );

  /* copy vector size to device
     note: vec_size_d is declared as constant memory,
     therefore the syntax is different than when copying to global memory
  */
```

```
    CUDA_ASSERT( cudaMemcpyToSymbol( vec_size_d, &vec_size_h, sizeof(int) ) );

    /* Define grid dimensions
       to do: discuss why the block number is set to this value!
     */
    int n_blocks  = vec_size_h / n_threads + (vec_size_h % n_threads != 0);
    dim3 blocks( n_blocks );
    dim3 threads(n_threads);

    std::chrono::time_point<std::chrono::system_clock> start, end;
    start = std::chrono::system_clock::now();

    /* to do:
       1) finish kernel definition (see above __global__ void vector_addition_kernel(...) )
       2) call kernel
    */




    /* to do: copy back the result vector */


    cudaDeviceSynchronize();

    end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end-start;

    cout << "Kernel duration: " << elapsed_seconds.count() << " s " << endl;
    cout << "Time per kenel: " << elapsed_seconds.count() / vec_size_h << endl;
```

```
  for ( int i = 0; i < vec_size_h; i++ ) {
    cout << a_h[i] << " + " << b_h[i] << " = " << c_h[i] << endl;
  }

  /* to do: free the remainig device memory */
  CUDA_ASSERT( cudaFree( a_d ) );

  /* free host memory */
  delete [] a_h;
  delete [] b_h;
  delete [] c_h;

  return 0;
}
```

Question 03: OpenMP

Task Description:

Write an OpenMP program to parallelize a sorting algorithm (e.g., quicksort or mergesort). Discuss the challenges of parallelizing sorting algorithms and how you addressed them. (comment your answer)

Best of luck 😊