Maulana Azad National Institute of Technology

(An Institute of National Importance)

Bhopal – 462003 (India)

Department of Computer Science & Engineering

LAB ASSIGNMENT

HADOOP/CUDA (CSE-328)

Name: Jishan Shaikh (161112013)

Instructor: Rahul Singh

Date: March 7, 2019 (Mid Submission)

Index

CUDA:

HADOOP:

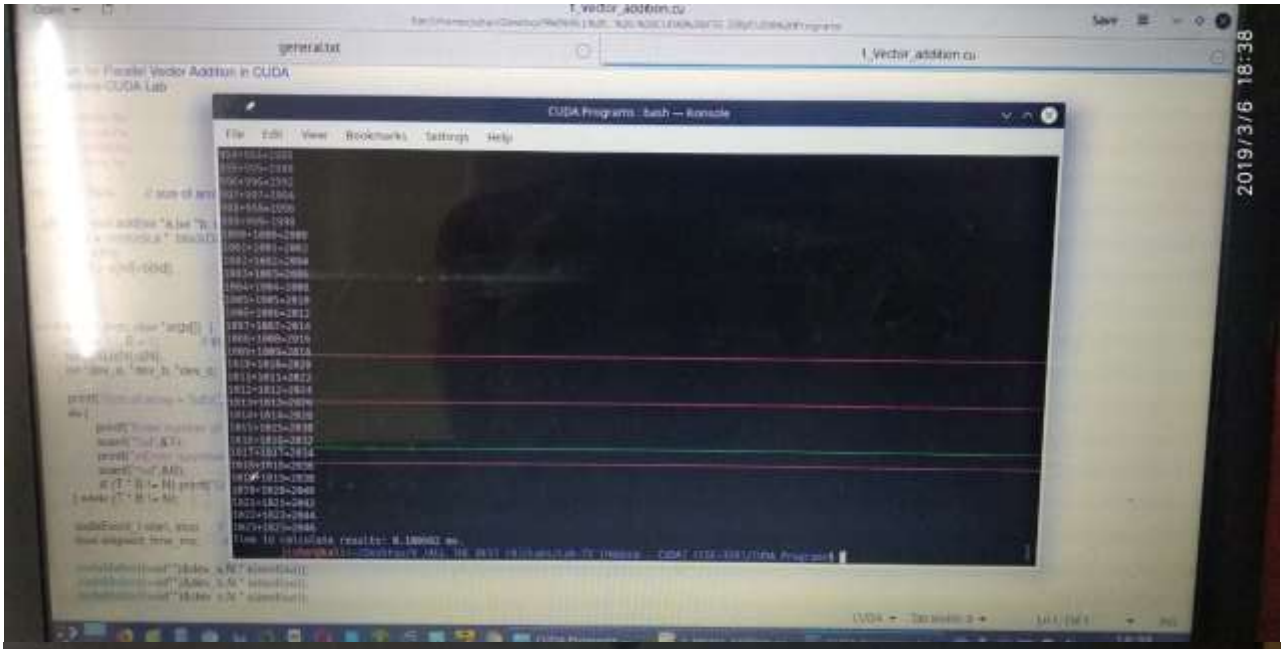
CUDA:

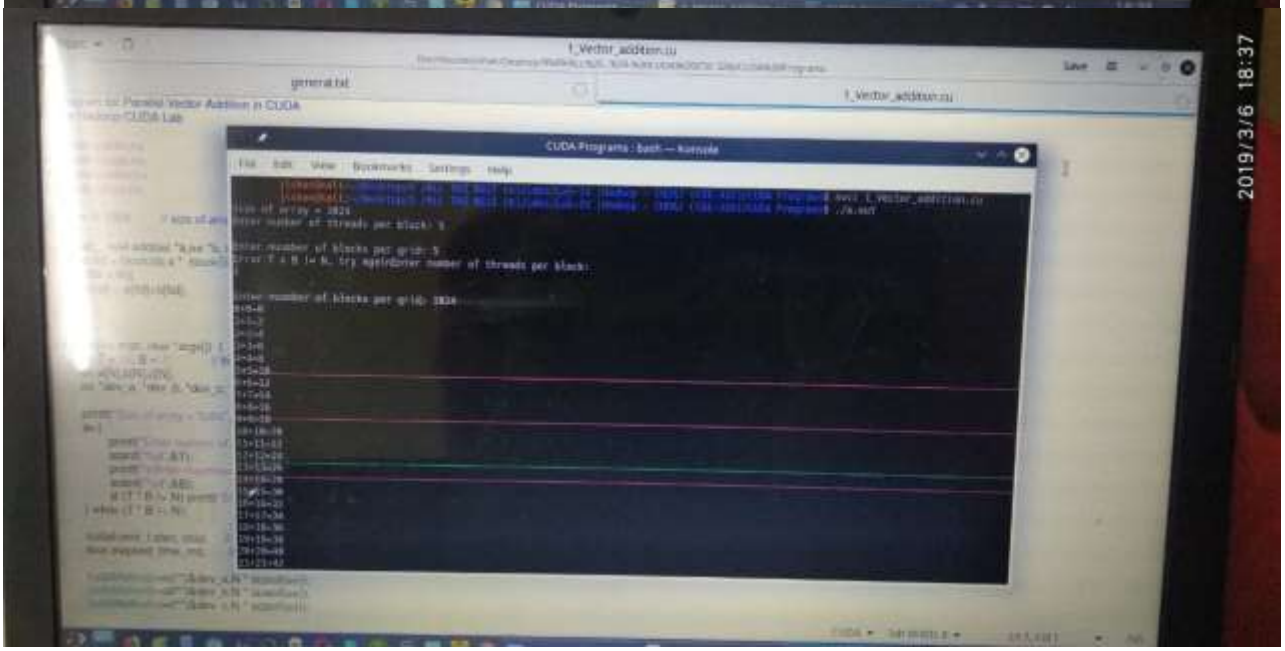
Write a CUDA Program for Parallel Vector addition.

Program:

```
// Program for Parallel Vector Addition in CUDA
// For Hadoop-CUDA Lab

#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>
#include <time.h>

#define N 1024        // size of array

_global_ void add(int *a,int *b, int *c) {
          int tid = blockIdx.x *  blockDim.x + threadIdx.x;
      if(tid < N){
        c[tid] = a[tid]+b[tid];
      }
}

int main(int argc, char *argv[])  {
          int T = 10, B = 1;          // threads per block and blocks per grid, taking default values
          int a[N],b[N],c[N];
          int *dev_a, *dev_b, *dev_c;

          printf("Size of array = %d\n", N);
          do {
                    printf("Enter number of threads per block: ");
                    scanf("%d",&T);
                    printf("\nEnter nuumber of blocks per grid: ");
                    scanf("%d",&B);
                    if (T * B != N) printf("Error T x B != N, try again");
          } while (T * B != N);

          cudaEvent_t start, stop;     // using cuda events to measure time
          float elapsed_time_ms;       // which is applicable for asynchronous code also

          cudaMalloc((void**)&dev_a,N * sizeof(int));
          cudaMalloc((void**)&dev_b,N * sizeof(int));
```

```
	cudaMalloc((void**)&dev_c,N * sizeof(int));

	for(int i=0;i<N;i++) {    // load arrays with some numbers
		a[i] = i;
		b[i] = i*1;
	}

	cudaMemcpy(dev_a, a , N*sizeof(int),cudaMemcpyHostToDevice);
	cudaMemcpy(dev_b, b , N*sizeof(int),cudaMemcpyHostToDevice);
	cudaMemcpy(dev_c, c , N*sizeof(int),cudaMemcpyHostToDevice);

	cudaEventCreate( &start );     // instrument code to measure start time
	cudaEventCreate( &stop );
	cudaEventRecord( start, 0 );

	add<<<B,T>>>(dev_a,dev_b,dev_c);

	cudaMemcpy(c,dev_c,N*sizeof(int),cudaMemcpyDeviceToHost);

	cudaEventRecord( stop, 0 );     // instrument code to measue end time
	cudaEventSynchronize( stop );
	cudaEventElapsedTime( &elapsed_time_ms, start, stop );

	for(int i=0;i<N;i++) {
		printf("%d+%d=%d\n",a[i],b[i],c[i]);
	}

	printf("Time to calculate results: %f ms.\n", elapsed_time_ms);  // print out execution
time

	// clean up
	cudaFree(dev_a);
	cudaFree(dev_b);
	cudaFree(dev_c);

	cudaEventDestroy(start);
	cudaEventDestroy(stop);

	return 0;
}
```





Write a CUDA Program for Parallel Matrix Multiplication.

Program:

```
// Program for Matrix Addition in CUDA
// For Hadoop-CUDA Lab

#include <stdio.h>
#include <cuda.h>
```

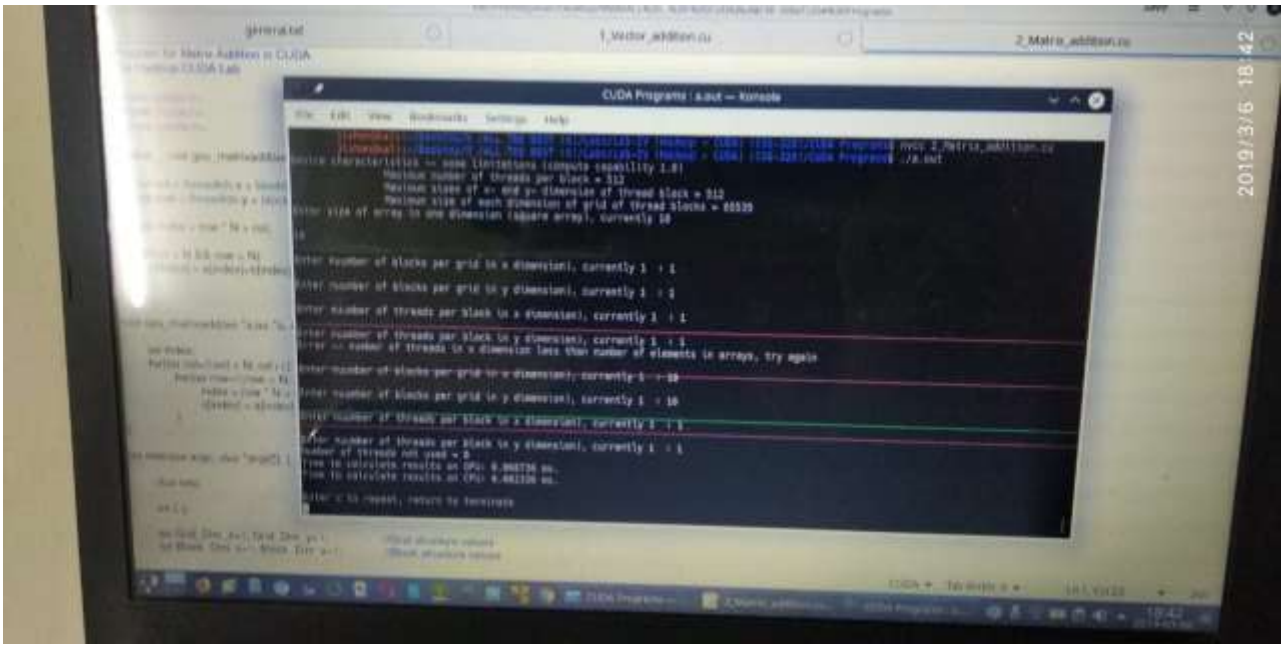
```
#include <stdlib.h>

_global_ void gpu_matrixadd(int *a,int *b, int *c, int N) {

        int col = threadIdx.x + blockDim.x * blockIdx.x;
        int row = threadIdx.y + blockDim.y * blockIdx.y;

        int index = row * N + col;

        if(col < N && row < N)
      c[index] = a[index]+b[index];

}

void cpu_matrixadd(int *a,int *b, int *c, int N) {

        int index;
        for(int col=0;col < N; col++)
                for(int row=0;row < N; row++) {
                        index = row * N + col;
                c[index] = a[index]+b[index];
                }
}

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

        char key;

        int i, j;                                       // loop counters

        int Grid_Dim_x=1, Grid_Dim_y=1;                         //Grid structure values
        int Block_Dim_x=1, Block_Dim_y=1;               //Block structure values

        int noThreads_x, noThreads_y;           // number of threads available in device, each
dimension
        int noThreads_block;                            // number of threads in a block

        int N = 10;                                     // size of array in each dimension
        int *a,*b,*c,*d;
        int *dev_a, *dev_b, *dev_c;
        int size;                                       // number of bytes in arrays
```

```
	cudaEvent_t start, stop;		// using cuda events to measure time
	float elapsed_time_ms;		// which is applicable for asynchronous code
also

/* --------------------ENTER INPUT PARAMETERS AND DATA ----------------------*/

do {  // loop to repeat complete program

	printf ("Device characteristics -- some limitations (compute capability 1.0)\n");
	printf ("		Maximum number of threads per block = 512\n");
	printf ("		Maximum sizes of x- and y- dimension of thread block = 512\n");
	printf ("		Maximum size of each dimension of grid of thread blocks = 65535\n");

	printf("Enter size of array in one dimension (square array), currently %d\n",N);
	scanf("%d",&N);

	do {
		printf("\nEnter nuumber of blocks per grid in x dimension), currently %d  :
",Grid_Dim_x);
		scanf("%d",&Grid_Dim_x);

		printf("\nEnter nuumber of blocks per grid in y dimension), currently %d  :
",Grid_Dim_y);
		scanf("%d",&Grid_Dim_y);

		printf("\nEnter nuumber of threads per block in x dimension), currently %d  :
",Block_Dim_x);
		scanf("%d",&Block_Dim_x);

		printf("\nEnter nuumber of threads per block in y dimension), currently %d  :
",Block_Dim_y);
		scanf("%d",&Block_Dim_y);

		noThreads_x = Grid_Dim_x * Block_Dim_x;		// number of threads in
x dimension
		noThreads_y = Grid_Dim_y * Block_Dim_y;		// number of threads in
y dimension

		noThreads_block = Block_Dim_x * Block_Dim_y;		// number of threads in
a block
```

```
		if (noThreads_x < N) printf("Error -- number of threads in x dimension less than number of elements in arrays, try again\n");
		else if (noThreads_y < N) printf("Error -- number of threads in y dimension less than number of elements in arrays, try again\n");
		else if (noThreads_block > 512) printf("Error -- too many threads in block, try again\n");
		else printf("Number of threads not used = %d\n", noThreads_x * noThreads_y - N * N);

	} while (noThreads_x < N || noThreads_y < N || noThreads_block > 512);

	dim3 Grid(Grid_Dim_x, Grid_Dim_x);		//Grid structure
	dim3 Block(Block_Dim_x,Block_Dim_y); //Block structure, threads/block limited by specific device

	size = N * N * sizeof(int);			// number of bytes in total in arrays

	a = (int*) malloc(size);		//this time use dynamically allocated memory for arrays on host
	b = (int*) malloc(size);
	c = (int*) malloc(size);		// results from GPU
	d = (int*) malloc(size);		// results from CPU

	for(i=0;i < N;i++)				// load arrays with some numbers
	for(j=0;j < N;j++) {
		a[i * N + j] = i;
		b[i * N + j] = i;
	}

/* ------------- COMPUTATION DONE ON GPU ----------------------------*/

	cudaMalloc((void**)&dev_a, size);		// allocate memory on device
	cudaMalloc((void**)&dev_b, size);
	cudaMalloc((void**)&dev_c, size);

	cudaMemcpy(dev_a, a , size ,cudaMemcpyHostToDevice);
	cudaMemcpy(dev_b, b , size ,cudaMemcpyHostToDevice);
	cudaMemcpy(dev_c, c , size ,cudaMemcpyHostToDevice);

	cudaEventCreate(&start);		// instrument code to measure start time
	cudaEventCreate(&stop);
```

```
	cudaEventRecord(start, 0);
//	cudaEventSynchronize(start);	// Needed?

	gpu_matrixadd<<<Grid,Block>>>(dev_a,dev_b,dev_c,N);

	cudaMemcpy(c,dev_c, size ,cudaMemcpyDeviceToHost);

	cudaEventRecord(stop, 0);	// instrument code to measue end time
	cudaEventSynchronize(stop);
	cudaEventElapsedTime(&elapsed_time_ms, start, stop );

//	for(i=0;i < N;i++)
//	for(j=0;j < N;j++)
//	  printf("%d+%d=%d\n",a[i * N + j],b[i * N + j],c[i * N + j]);

	printf("Time to calculate results on GPU: %f ms.\n", elapsed_time_ms);  // print out execution time

/* ------------ COMPUTATION DONE ON HOST CPU ----------------------------*/

	cudaEventRecord(start, 0);		// use same timing
//	cudaEventSynchronize(start);	// Needed?

	cpu_matrixadd(a,b,d,N);		// do calculation on host

	cudaEventRecord(stop, 0);	// instrument code to measue end time
	cudaEventSynchronize(stop);
	cudaEventElapsedTime(&elapsed_time_ms, start, stop );

	printf("Time to calculate results on CPU: %f ms.\n", elapsed_time_ms);  // print out execution time

/* ------------------- check device creates correct results -----------------*/

	for(i=0;i < N*N;i++) {
		if (c[i] != d[i]) printf("**** ERROR in results, CPU and GPU create different answers ***\n");
		break;
	}

	printf("\nEnter c to repeat, return to terminate\n");
	scanf("%c",&key);
```

```
        scanf("%c",&key);

} while (key == 'c'); // loop of complete program

/* -------------  clean up  ----------------------------------------*/
        free(a);
        free(b);
        free(c);
        cudaFree(dev_a);
        cudaFree(dev_b);
        cudaFree(dev_c);

        cudaEventDestroy(start);
        cudaEventDestroy(stop);

        return 0;
}
```



Write a CUDA Program for Parallel Binary Search.

Program: // Program for Parallel Binary Search in CUDA
// For Hadoop-CUDA Lab

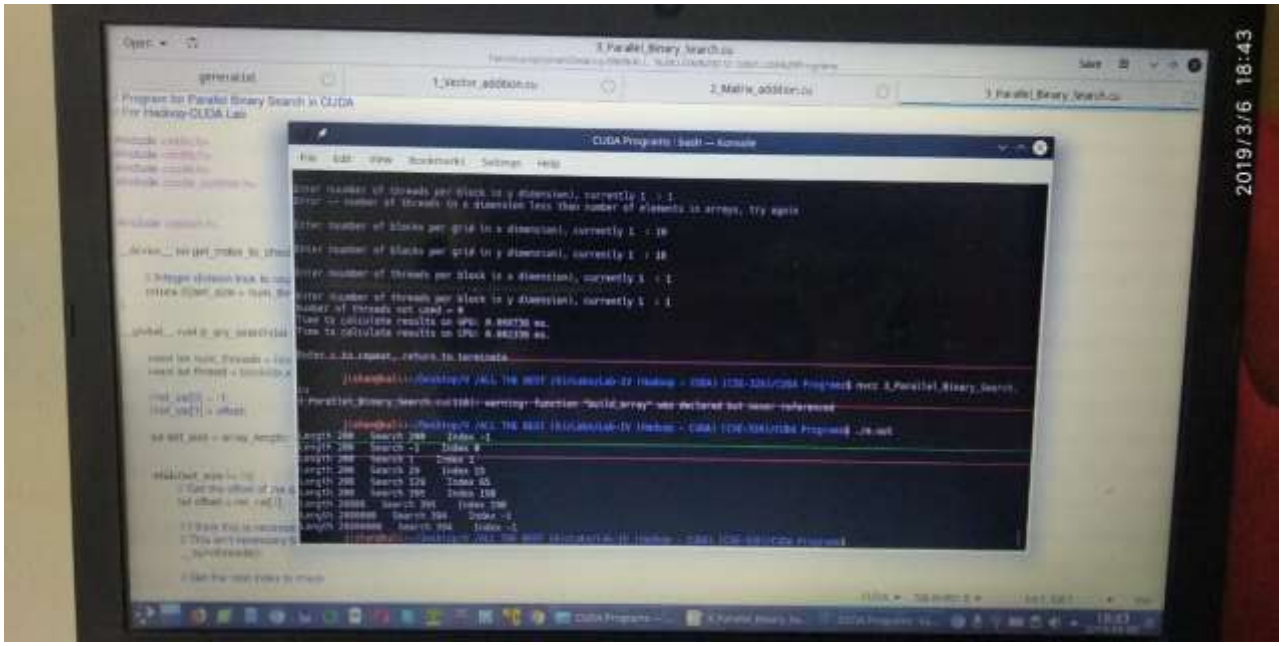
```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <cuda_runtime.h>
```

```
#include <assert.h>

_device_ int get_index_to_check(int thread, int num_threads, int set_size, int offset) {

	// Integer division trick to round up
	return (((set_size + num_threads) / num_threads) * thread) + offset;
}

_global_ void p_ary_search(int search, int array_length,  int *arr, int *ret_val ) {

	const int num_threads = blockDim.x * gridDim.x;
	const int thread = blockIdx.x * blockDim.x + threadIdx.x;

	//ret_val[0] = -1;
	//ret_val[1] = offset;

	int set_size = array_length;


	while(set_size != 0){
		// Get the offset of the array, initially set to 0
		int offset = ret_val[1];

		// I think this is necessary in case a thread gets ahead, and resets offset before
it's read
		// This isn't necessary for the unit tests to pass, but I still like it here
		__syncthreads();

		// Get the next index to check
		int index_to_check = get_index_to_check(thread, num_threads, set_size,
offset);

		// If the index is outside the bounds of the array then lets not check it
		if (index_to_check < array_length){

			// If the next index is outside the bounds of the array, then set it to
maximum array size
			int next_index_to_check = get_index_to_check(thread + 1,
num_threads, set_size, offset);
```

```
                        if (next_index_to_check >= array_length){
                                next_index_to_check = array_length - 1;
                        }

                        // If we're at the mid section of the array reset the offset to this index
                        if (search > arr[index_to_check] && (search <
arr[next_index_to_check])) {
                                ret_val[1] = index_to_check;
                        }
                        else if (search == arr[index_to_check]) {
                                // Set the return var if we hit it
                                ret_val[0] = index_to_check;
                        }
                }

                // Since this is a p-ary search divide by our total threads to get the next set size
                set_size = set_size / num_threads;

                // Sync up so no threads jump ahead and get a bad offset
                __syncthreads();
        }
}


int chop_position(int search, int *search_array, int array_length)
{
        // Get the size of the array for future use
        int array_size = array_length * sizeof(int);

        // Don't bother with small arrays
        if (array_size == 0) return -1;

        // Setup array to use on device
   int   *dev_arr;
        cudaMalloc((void**)&dev_arr, array_size);

        // Copy search array values
        cudaMemcpy(dev_arr, search_array, array_size, cudaMemcpyHostToDevice);

        // return values here and on device
        int             ret_val = (int)malloc(sizeof(int) * 2);
        ret_val[0] = -1; // return value
```

```
	ret_val[1] = 0; // offset
	array_length = array_length % 2 == 0 ? array_length : array_length - 1; // array size

	int 		*dev_ret_val;
	cudaMalloc((void**)&dev_ret_val, sizeof(int) * 2);

	// Send in some intialized values
	cudaMemcpy(dev_ret_val, ret_val, sizeof(int) * 2, cudaMemcpyHostToDevice);

	// Launch kernel
	// This seems to be the best combo for p-ary search
	// Optimized around 10-15 registers per thread
	p_ary_search<<<16, 64>>>(search, array_length, dev_arr, dev_ret_val);

	// Get results
	cudaMemcpy(ret_val, dev_ret_val, 2 * sizeof(int), cudaMemcpyDeviceToHost);

	int ret = ret_val[0];

	printf("Ret Val %i    Offset %i\n", ret, ret_val[1]);

	// Free memory on device
	cudaFree(dev_arr);
	cudaFree(dev_ret_val);

	free(ret_val);

	return ret;
}

// Test region
static int * build_array(int length) {

	int ret_val = (int)malloc(length * sizeof(int));

	for (int i = 0; i < length; i++)
	{
		ret_val[i] = i * 2 - 1;
	}

	return ret_val;
}
```

```
static void test_array(int length, int search, int index) {

	printf("Length %i   Search %i    Index %i\n", length, search, index);
	// assert(index == chop_position(search, build_array(length), length) &&
"test_small_array()");

}

static void test_arrays() {
	test_array(200, 200, -1);
	test_array(200, -1, 0);
	test_array(200, 1, 1);
	test_array(200, 29, 15);
	test_array(200, 129, 65);
	test_array(200, 395, 198);
	test_array(20000, 395, 198);
	test_array(2000000, 394, -1);
	test_array(20000000, 394, -1);
}

int main(){
	test_arrays();
}
```



HADOOP:

Write a Map Reduce program to calculate word frequency in given document

Program:

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordFrequency {
        public static void main(String [] args) throws Exception

        {

        Configuration conf=new Configuration();
        Job job=new Job(conf,"wordcount");
        job.setJarByClass(WordFrequency.class);
        job.setMapperClass(MapClass.class);
        job.setReducerClass(ReduceClass.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);

        }

        public static class MapClass extends Mapper<LongWritable, Text, Text, IntWritable>{

                public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
                {
                        String line = value.toString();
                        String[] words=line.split(" ");
```

```
                    for(String word: words )
                    {
                        context.write(new Text(word.trim()),  new IntWritable(1));
                    }
                }
        }

        public static class ReduceClass extends Reducer<Text, IntWritable, Text, IntWritable>
        {
                public void reduce(Text word, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException
                {
                     int sum = 0;
                     for(IntWritable value : values)
                     {
                        sum += value.get();
                     }

                    context.write(word, new IntWritable(sum));
                    }
        }
}
```

Calculate average marks of every student in the given dataset

Program:

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class AverageMarks {
        public static void main(String [] args) throws Exception

        {

        Configuration conf=new Configuration();
```

```
        Job job=new Job(conf,"Average marks ");
        job.setJarByClass(AverageMarks.class);
        job.setMapperClass(MapClass.class);
        job.setReducerClass(ReduceClass.class);
        job.setOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputValueClass(FloatWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);
        }

        public static class MapClass extends Mapper<LongWritable, Text, Text, IntWritable>{

                public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
                {
                        String line = value.toString();
                        String[] Words=line.split(",");
                        if(Words.length>1)
                        {
                        context.write(new Text(Words[0]),  new
IntWritable(Integer.parseInt(Words[1])));
                        }

                }
        }

        public static class ReduceClass extends Reducer<Text, IntWritable, Text, FloatWritable>
        {
                public void reduce(Text word, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException
                {
                         int total= 0;
                        int count=0;
                         for(IntWritable value : values)
                         {
                                total += value.get();
                                count+=1;
                         }

                        context.write(word, new FloatWritable(total/count));
```

```
                        }
        }
}
```

Write a program to illustrate use of partitioner in Map Reduce

Program:

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver{
        public static void main(String [] args)  throws Exception{

        Configuration conf =new Configuration();
        Job job=new Job(conf,"Implement Partitioner");
        job.setJarByClass(Driver.class);
        job.setMapperClass(MapClass.class);
        job.setPartitionerClass(PartitionClass.class);
        job.setReducerClass(ReduceClass.class);
        job.setNumReduceTasks(6);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);
        }

        public static class MapClass extends Mapper<LongWritable, Text, Text, Text>{

                public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
                {
                        String line = value.toString();
```

```
                String[] Words=line.split(",");
                if(Words.length>1)
                {
                        context.write(new Text(Words[0]),  new Text(Words[1]));
                }
        }
}

public static class PartitionClass extends Partitioner<Text,Text>{
        public int getPartition(Text key,Text value,int numReduceTask)
        {
        if(numReduceTask==0)
                return 0;
        if(key.equals(new Text("Cse1")))
                return 0;
        else if(key.equals(new Text("Mech1")))
                return 1%numReduceTask;
        else if(key.equals(new Text("Cse2")))
                return 2%numReduceTask;
        else if(key.equals(new Text("Mech2")))
                return 3%numReduceTask;
        else if (key.equals(new Text("Civil")))
                return 4%numReduceTask;
        else
                return 5%numReduceTask;
        }
}

public static class ReduceClass extends Reducer<Text,Text,Text,Text>{
        public void reduce(Text key,Iterable<Text> Values,Context context)throws
IOException, InterruptedException{
        int sum=0;
        for(Text val:Values)
        {
                context.write(key,val);
        }
        }
}
}
```