Type: MCQ

Q1. Apply convolution filter [1, 2, 3, 2, 1] for element 1 for the array [1, 2, 3, 4, 5, 6,7]. Your answer is _____.(0.5)

1. **10
2. 20
3. 27
4. 9

Q2. Choose the full form of CUDA (0.5)

1. ** Compute Unified Device Architecture
2. Collective Unified Device Architecture
3. Compute Unified Data Architecture
4. Collective Unified Data Architecture

Q3. Compute the threadId of the thread belonging to block2 thread3 if the block dimension is 32 and the grid dimension is 16. Your answer is _____ (0.5)

1. 35
2. **67
3. 50
4. 98

Q4. Choose the correct answer

A)The convolution filter size is larger than the data size.
B)The number of grids is equivalent to the number of kernels. (0.5)

1. T F
2. F F
3. ** F T
4. T T

Q5. .Identify the correct statement with respect to MPI programming.

A)Collective communication routines must involve all processes within the scope of a communicator.
B)MPI_Barrier function will increase parallelism. (0.5)

1. F F
2. F T
3. ** T F
4. T T

Q6. Compared to shared-variable programs, MPI programs tend to exhibit ____ *cache hit rates* when executing on multiprocessors and d*ebugging them is* _____ than debugging shared-variable programs. (0.5)

1. ** high, simpler
2. low, simpler
3. high, difficult
4. low, difficult

Q7. Identify the correct statement with respect to parallel pipeline processing.
A)It is a combination of temporal and data parallelism.
B)Load balancing of work is possible in static assignment(0.5)

1. F T
2. ** T F
3. F F
4. T T

Q8. Values of integer array A is as shown below in 3 processes in which process 0 being the root:

In Process 0: 12 34 56 32 15 20
In Process 1: 89 90 1 23 6 57
In Process 2: 45 64 23 14 23 56

List the contents of the array B in the process with rank 2 after executing the following routine.

MPI_Allgather(A,2,MPI_INT,B,2,MPI_INT,MPI_COMM_WORLD); (0.5)

1. 15 20 6 57 23 56
2. **12 34 89 90 45 64
3. All the values of Process0, then Process 1 and Process 2
4. 12 89 45 34 90 64

Q9. Values of integer array A is as shown below in 3 processes in which process 0 being the root:

In Process 0: d e f
In Process 1: 1 2 3
In Process 2: a b c
List the contents of the array B in the process with rank 2 after executing the following routine.
MPI_Alltoall(A,1,MPI_INT,B,1,MPI_INT,MPI_COMM_WORLD); (0.5)

1. d 1 a e 2 b f 3 c
2. **f 3 c
3. d e f 1 2 3 a b c
4. d 1 a

Q10. Compute the time taken in temporal parallelism to correct 1000 papers having 4 questions, if 4 teachers are employed to evaluate one answer and each answer requires 5 minutes to evaluate. (0.5)

1. ** 5015
2. 5025
3. 5000
4. 20000

Type: DES

Q11. Develop a MPI program to read a word *Str* of length *wordsize* in the root process, where *wordsize* is evenly divisible by total *size* number of processes. Using point to point routines root sends equal data to all processes. Let each process (including root) check received characters and if there are digits extracts it. Store the sum of digits in variable *Sum* otherwise store a value 0. Print the extracted digits and partial sum in each process with rank. Using collective communication routine, find the total sum of all the digits extracted in each process and print total sum in root process.

Str: Pc3ap23mi12dexam Size : 4
Process 0: 3 Sum : 3 Process 1: 23 Sum : 5
Process 2: 12 Sum : 3 Process 3: Sum : 0
Total Sum in process 0 : 11 (4)

```
#include<stdio.h>
#include<mpi.h>
#include<string.h>
int main(int argc, char *argv[])
{
int i,chunk,len,r,size,totsum,sum=0;
char s[20],stemp[20];
MPI_Status status;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&r);
MPI_Comm_size(MPI_COMM_WORLD,&size);

if(r==0)
{
scanf("%s",s);
len=strlen(s);
chunk=len/size;                    0.5M


for(i=1;i<size;i++)
{
 MPI_Send(&chunk,1,MPI_INT,i,0,MPI_COMM_WORLD);
 MPI_Send(&s[i*chunk],chunk,MPI_CHAR,i,1,MPI_COMM_WORLD);   1M
}
}
```

```
else
{
MPI_Recv(&chunk,1,MPI_INT,0,0,MPI_COMM_WORLD,&status);
MPI_Recv(s,chunk,MPI_CHAR,0,1,MPI_COMM_WORLD,&status);       1M
}

for(i=0;i<chunk;i++)
{
if(s[i]>='0' && s[i]<='9')
 sum=sum+(s[i]-48);
 }  0.5M

 fprintf(stdout,"Sum by %d = %d", r,sum);
 fflush(stdout);

 MPI_Reduce(&sum,&totsum,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
 if(r==0)
 {
 fprintf(stdout,"Total Sum by %d = %d", r,totsum);
 fflush(stdout);
 }
 }  1M
```

Q12. Design a Cuda application for sorting N elements using odd-even transposition sort. Write the host and kernel code. (4)

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
#include<stdlib.h>
__global__ void odd(long int* A)
{
	int idx =  threadIdx.x;
	int size = blockDim.x;
	if ((idx % 2) != 0 && idx + 1 <= size - 1)
	{

		if (A[idx] > A[idx + 1])
		{
			int temp = A[idx];
			A[idx] = A[idx + 1];
			A[idx + 1] = temp;
		}

	}
}
__global__ void even(long int* A)
{
	int idx = blockIdx.x*blockDim.x+threadIdx.x;
	int size = blockDim.x*1024;
	if ((idx % 2) == 0 && idx <= size - 1)
	{
```

```
                if (A[idx] > A[idx + 1])
                {
                        int temp = A[idx];
                        A[idx] = A[idx + 1];
                        A[idx + 1] = temp;
                }
        }
}
int main()
{
        long int* dev_a = 0;
        long int* a;
        long int size;
        cudaError_t cudaStatus;
        printf("Enter the size of the array");
        scanf_s("%ld", &size);
        a = (long int*)malloc(sizeof(long int) * size);
        printf("Enter the array");
        for (long int i = 0; i < size; i++)
        {
                a[i] = rand();
        }
        cudaStatus = cudaMalloc((void**)&dev_a, size * sizeof(long int));
        cudaStatus = cudaMemcpy(dev_a, a, size * sizeof(long int),
cudaMemcpyHostToDevice);

        for(long int i=0;i<size/2;i++)
        {

                odd << < 1, size >> > (dev_a);
                even << <1, size>> > (dev_a);

        }
        cudaStatus = cudaMemcpy(a, dev_a, size * sizeof(long int),
cudaMemcpyDeviceToHost);
        printf("Result\n");
        for (long int w = 0; w < size; w++)
        {
                printf("%ld\t", a[w]);
        }
        cudaFree(dev_a);
        return 0;
}
```

Kernel→2M

Host Invocation→1M

Host Code→1M

Q13. Design a Cuda application that converts an RGB image to a Grayscale image. Write only the kernel code. (3)

```
__global__ void rgbToGray(unsigned char *in,unsigned char *out,int width,int
height)
{
        int Col = blockDim.x*blockIdx.x+threadIdx.x;
        int Row = blockDim.y * blockIdx.y + threadIdx.y;
        if (Col < width && Row < height)
```

```
	{
		int grayOffset = Row * width + Col;
		int rgbOffset = grayOffset * 3;
		unsigned char r = in[rgbOffset];
		unsigned char g = in[rgbOffset + 1];
		unsigned char b = in[rgbOffset + 2];
		out[grayOffset] =(unsigned char) (0.21f * r + 0.71f * g + 0.07f *
b);
	}
}
```

threadId Calculation→1M

offset calculation→1M

Computation→1M

Q14. Design a Cuda application to add two vectors. Write the host and kernel code. (3)

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
__global__ void vecAdd(int* A, int* B, int* C, int n)
{
	int idx = blockDim.x * blockIdx.x + threadIdx.x;
	if (idx < n)
	{
		C[idx] = A[idx] + B[idx];
	}

}
int main()
{
	int h_A[8] = { 1,1,1,1,1,1,1,1 };
	int h_B[8] = { 1,1,1,1,1,1,1,1 };
	int h_C[8];
	int* d_A, * d_B, * d_C;
	cudaMalloc(&d_A, sizeof(int) * 8);
	cudaMalloc(&d_B, sizeof(int) * 8);
	cudaMalloc(&d_C, sizeof(int) * 8);
	cudaMemcpy(d_A, h_A, sizeof(int) * 8, cudaMemcpyHostToDevice);
	cudaMemcpy(d_B, h_B, sizeof(int) * 8, cudaMemcpyHostToDevice);
	vecAdd << <1, 32 >> > ( d_A,d_B,d_C, 8);
	cudaMemcpy(h_C, d_C, sizeof(int) * 8, cudaMemcpyDeviceToHost);
	for (int i = 0; i < 8; i++)
	{
		printf("%d", h_C[i]);
	}
	cudaFree(d_A);
	cudaFree(d_B);
	cudaFree(d_C);
	return 0;
}
```

Kernel→1M

Host code→2M

Q15. Point out the Cuda qualifiers for kernels, clearly specifying where they are executed and where they are invoked from. (3)

	Executed on the:	Only callable from the:
`__device__ float DeviceFunc()`	device	device
`__global__ void KernelFunc()`	device	host
`__host__ float HostFunc()`	host	host

Each entry of the table carries 1M

Q16. Values of integer array A is as shown below in 4 processes in which process 0 being the root:

A value in Process 0: 0, 17, 34, 51
A value in Process 1: 5, 22, 39, 56
A value in Process 2: 10, 27, 44, 61
A value in Process 3: 15, 32, 49, 66

Let B and C are 1D integer arrays initialized with a value 0.

i)Write an MPI function call with appropriate parameters to display the following output using value of A as shown above.

B value in Process 0: 0, 5,10,15
B value in Process 1: 17, 22,27,32
B value in Process 2: 34, 39,44,49
B value in Process 3: 51, 56,61,66

ii) Write an MPI function call with appropriate parameters to display the following output using value of A as shown above.

C value in Process 0: 0, 17, 34,0
C value in Process 1: 5, 39, 73,0
C value in Process 2: 15, 66, 117,0
C value in Process 3: 30, 98, 166,0 (3)

1) MPI_Alltoall(&A, 1, MPI_INT, B, 1, MPI_INT, MPI_COMM_WORLD);
MPI_Scan (&A, &C, 3, MPI_INT,MPI_SUM , MPI_COMM_WORLD);
Each function 1.5M 1.5*2 =3M

Q17. Compare data and temporal parallel processing (any six). (3)

temporal parallel processing:

1. Job is divided into a set of independent tasks and tasks are assigned for processing
2. Tasks should take equal time. Pipeline stages thus be synchronized
3. Bubbles in job leads to idling of processors
4. Processors specialized to do specific tasks efficiently
5. Task assignment static

6. Not fault tolerant
7. Efficient with fine grained tasks
8. Scales well as long as number of jobs is much greater than number of tasks and communication cost from one processor to next is negligible

Data parallel processing:

1. Full jobs are assigned for processing
2. Jobs may take different times. No need to synchronize the beginning of the jobs
3. Bubbles do not cause idling of processors
4. Processors are general purpose
5. Job assignment may be static, dynamic and quasi dynamic
6. Tolerates processor faults
7. Efficient with coarse grained tasks and quasi dynamic scheduling
8. Scales well as long as number of jobs is much greater than number of processors and processing time is much higher than the time to distribute data to processors.
   Each difference : 0.5M 6*0.5 = 3M

Q18. Illustrate the main disadvantages of data parallelism with dynamic assignment. (2)

1. If many teachers correct answer book simultaneously then they have to wait in a queue to get answer book
2. The head examiner can become a bottleneck.
3. The head examiner is idle between handing out papers.
4. Not scalable

   Each 0.5*4 = 2M