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

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

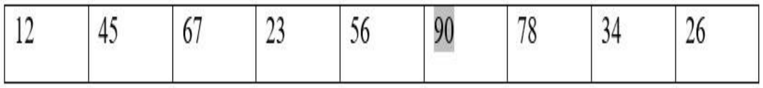
## SUBJECT: PARALLEL COMPUTER ARCHITECTURE AND PROGRAMMING (CSE-3252) (PCAP)

## MID-SEMESTER EXAMINATION MARCH 2024

## SCHEME OF EVALUATION

**Time :2hrs** **Max. Marks:30**

| Q. No . | Questions | M | CLO | AHEP LO | Blooms Taxonomy level |
|---|---|---|---|---|---|
| **1** | The platform model defines a device as an array of compute units which can function independently and are further divided into ___________.<br>**1. Processing elements**<br>2. kernels<br>3. memory units<br>4. cores | **0.5** | **3** | 1,2 | 2 |
| **2** | Creating buffer requires supplying the size of the buffer and a _________ in which the buffer will be allocated<br>1. Command queue<br>2. work item<br>**3. context**<br>4. kernel | **0.5** | **3** | 1,2 | 2 |
| **3** | Assume that a variable ‘x’ is declared inside a kernel function (OpenCL kernel) as int x. ‘x’ will be now allocated to ___________ memory.<br>**1. Private**<br>2. local<br>3. global<br>4. constant memory | **0.5** | **3** | 1,2 | 2 |
| **4** | Give the work item’s local id and global id which will be processing the shaded element of the 1D array in the following figure. Assume the work group size is 3. | **0.5** | **3** | 1,2 | 3 |

| | | | | | |
|---|---|---|---|---|---|
| |  12 45 67 23 56 90 78 34 26<br>1. 2,5<br>**2. 5, 2**<br>3. 5,1<br>4. 5,0 | | | | |
| **5** | Identify the CORRECT statements with respect to the CUDA programming?<br>i) A device function can be called from host function.<br>ii) The second execution configuration parameter in CUDA specifies the dimensions of each block in number of threads.<br>iii) All the threads in a block can access the block index using blockDim predefined variable in kernel.<br>1. i and ii are TRUE<br>**2. i and iii are FALSE**<br>3. ii and iii are TRUE<br>4. ii and iii are FALSE | **0.5** | **4** | 1,2 | 3 |
| **6** | The function cudaMemcpy() is used to copy data<br>a) From one location of device memory to another location of device memory<br>b) Between different GPUs in multi-GPU systems<br>c) From one location of device memory to another location of host memory<br>d) From one location of host memory to another location of device memory<br>**1. a, c and d only**<br>2. c and d only<br>3. b, c and d only<br>4. All of the above | **0.5** | **4** | 1,2 | 2 |
| **7** | For the given input array N = {10, 35, 52, 63, 7, 81, 90} and Mask array M={2, 3, 5}, the output array value P[6] calculated for the 1D convolution is ________<br>1. 188<br>2. 7719<br>**3. 432** | **0.5** | **4** | 1,2 | 3 |

| | | | | | |
|---|---|---|---|---|---|
| | 4. 707 | | | | |
| **8** | In MPI, what is the purpose of MPI_Barrier function?<br>1. To initialize MPI communication<br>**2. To synchronize processes within a communicator**<br>3. To broadcast data from one process to all others<br>4. To reduce data across all processes | **0.5** | **2** | 1 | 2 |
| **9** | The disadvantages in MPI_Bsend and MPI_Ssend are ___ and _____respectively.<br>**1. extra copying and extra waiting**<br>2. extra waiting and extra copying<br>3. extra scattering and extra gathering<br>4. extra gathering and extra scattering | **0.5** | **2** | 1 | 3 |
| **10** | In ________, all PEs receive the same instruction broadcast from the control unit but operate on different data sets from distinct data streams<br>1. SISD<br>**2. SIMD**<br>3. MIMD<br>4. MISD | **0.5** | **1** | 1 | 2 |
| **11** | Write an opencl program that uses kernel function to put integer data into the 1d array of 128 locations. The program prompts the user to enter any number between 2 to 10(2 and 10 are included). If valid number is entered by the user, then only kernel is enqueued and produces the desired output. If a valid number is not entered, then program should prompt the user to enter valid number.<br>Enter a number between 2 and 10.<br>Number entered by user is 2<br>The following o/p is desired o/p | **4** | **3** | 1 | 4 |

The 1d array is 1 3 5 7 9 11 13…………………………..251, 253, 255

Enter a number between 2 and 10.
Number entered by user is 3
The following o/p is desired o/p
The 1d array is 1 4 7 10 13 16………………………..376, 379, 382

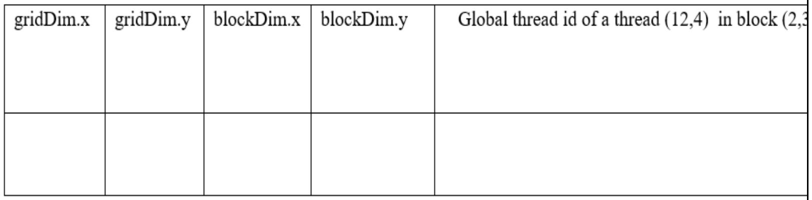
```
#include <stdio.h>
#include <stdlib.h>
// OpenCL includes
#include <CL/cl.h>
//#include<iostream>
#include<string.h>
#define MAX_SOURCE_SIZE 1000
//#include<conio.h>
// OpenCL kernel to perform an element-wise
// add of two arrays
int main() {
// This code executes on the OpenCL host
// Host data
            char *source_str;
   size_t source_size;
 char* programSource;
 char str[1000];
            FILE *fp;
            fp=fopen("E1.txt", "r");
   if(!fp) {
      fprintf(stderr, "Failed to load kernel.\n");
                        getchar();
      exit(1);
   }

   programSource = (char*)malloc(MAX_SOURCE_SIZE);
   source_size = fread(programSource, 1, MAX_SOURCE_SIZE, fp);
   fclose( fp );
int *A = NULL; // Input array
int n;
int datasize=sizeof(int)*128;
A=malloc(128*sizeof(int));
do
{
   printf("enter number between 2 and 10\n");
   scanf("%d",&n);

}while(n<2 || n>10);
cl_int status;

// STEP 1: Discover and initialize the platforms

cl_uint numPlatforms = 0;
```

```
cl_platform_id *platforms = NULL;
// Use clGetPlatformIDs() to retrieve the number of
// platforms
status = clGetPlatformIDs(0, NULL, &numPlatforms);
printf("%d platform success %d ",status,numPlatforms);
// Allocate enough space for each platform
platforms =
(cl_platform_id*)malloc(
numPlatforms*sizeof(cl_platform_id));
// Fill in platforms with clGetPlatformIDs()
status = clGetPlatformIDs(numPlatforms, platforms,
NULL);
char pform_vendor[40];
clGetPlatformInfo(platforms[0],           CL_PLATFORM_NAME,
sizeof(pform_vendor),
&pform_vendor, NULL);
printf(" the vendor %s",pform_vendor);

// STEP 2: Discover and initialize the devices

cl_uint numDevices = 0;
cl_device_id *devices = NULL;
// Use clGetDeviceIDs() to retrieve the number of
// devices present
status = clGetDeviceIDs(
platforms[0],
CL_DEVICE_TYPE_GPU,
0,
NULL,
&numDevices);
// Allocate enough space for each device
devices =
(cl_device_id*)malloc(
numDevices*sizeof(cl_device_id));
// Fill in devices with clGetDeviceIDs()
status = clGetDeviceIDs(
platforms[0],
CL_DEVICE_TYPE_GPU,
numDevices,
devices,
NULL);
printf("%d Device success %d ",status,numDevices);
char name_data[100];
int err = clGetDeviceInfo(devices[0], CL_DEVICE_NAME,
sizeof(name_data), name_data, NULL);
printf(" the device name %s",name_data);
// STEP 3: Create a context
cl_context context = NULL;
// Create a context using clCreateContext() and
// associate it with the devices
context = clCreateContext(
NULL,
numDevices,
devices,
NULL,
NULL,
&status);
printf("%d context success %d ",status,numDevices);
// STEP 4: Create a command queue
l_command_queue cmdQueue;
// Create a command queue using clCreateCommandQueue(),
// and associate it with the device you want to execute
// on
cmdQueue = clCreateCommandQueue(
context,
devices[0],
0,
&status);
printf("%d CQ success %d ",status,numDevices);
STEP 5: Create device buffers
cl_mem bufferA; // Input array on the device

cl_mem bufferB; // Output array on the device
// Use clCreateBuffer() to create a buffer object (d_A)
// that will contain the data from the host array A
bufferA = clCreateBuffer(
context,
CL_MEM_WRITE_ONLY,
datasize,
NULL,
&status);

// Use clCreateBuffer() to create a buffer object (d_C)
// with enough space to hold the output data
bufferB = clCreateBuffer(
context,
CL_MEM_READ_ONLY,
sizeof(cl_int),
NULL,
&status);
/ STEP 6: Write host data to device buffers
// Use clEnqueueWriteBuffer() to write input array A to
// the device buffer bufferA
status = clEnqueueWriteBuffer(
cmdQueue,
bufferB,
CL_FALSE,
0,
sizeof(cl_int),
&n,
0,
NULL,
NULL);
// STEP 7: Create and compile the program
// Create a program using clCreateProgramWithSource()
cl_program program = clCreateProgramWithSource(
context,
1,
(const char**)&programSource,
(const size_t *)&source_size,
&status);

// Build (compile) the program for the devices with
// clBuildProgram()
status = clBuildProgram(
program,
numDevices,
devices,
NULL,
NULL,
NULL);
printf("the build is %d\n",status);
// STEP 8: Create the kernel
cl_kernel kernel = NULL;
// Use clCreateKernel() to create a kernel from the
// vector addition function (named "vecadd")
kernel = clCreateKernel(program, "E1", &status);
// STEP 9: Set the kernel arguments
// Associate the input and output buffers with the
// kernel
// using clSetKernelArg()
status = clSetKernelArg(
kernel,
0,
sizeof(cl_mem),
&bufferA);

status |= clSetKernelArg(
kernel,
1,
sizeof(cl_mem),
&bufferB);
// STEP 10: Configure the work-item structure
// Define an index space (global work size) of work
// items for
// execution. A workgroup size (local work size) is not
// required,
// but can be used.
size_t globalWorkSize[1];
// There are ’ elements’  work-items
globalWorkSize[0] = 128;
// STEP 11: Enqueue the kernel for execution
// Execute the kernel by using
// clEnqueueNDRangeKernel().
// ' globalWorkSize'  is the 1D dimension of the
// work-items
status = clEnqueueNDRangeKernel(
cmdQueue,
kernel,
1,
NULL,
globalWorkSize,
NULL,
0,
NULL,
NULL);

// STEP 12: Read the output buffer back to the host

// Use clEnqueueReadBuffer() to read the OpenCL output
// buffer (bufferC)
// to the host output array (C)
clEnqueueReadBuffer(
cmdQueue,
bufferA,
CL_TRUE,
0,
datasize,
A,
0,
NULL,
NULL);
// Verify the output
for(int i=0;i<128;i++)
   printf("%d ",A[i]);

// STEP 13: Release OpenCL resources
// Free OpenCL resources
clReleaseKernel(kernel);
clReleaseProgram(program);
clReleaseCommandQueue(cmdQueue);
clReleaseMemObject(bufferA);
clReleaseMemObject(bufferB);
clReleaseContext(context);
// Free host resources
free(A);
//free(C);
free(platforms);
free(devices);

}
__kernel void E1(__global int* A,__global int* n)
{
            int idx=get_global_id(0);
```

```
 if(idx==0)
 A[idx]=1;
 else
 A[idx]=idx*(*n)+1;
}
```

Scheme:**Kernel 1M, steps 1to 4 1m, 5 to 8 1M , 9 to 12 1M**

| | | | | | |
|---|---|---|---|---|---|
| **2** | i)Assume that a grid has 128 blocks arranged in 2D and grid length (x direction) is 32. Threads in a block are arranged in 2D with block height(y direction) is 5. Each block contains 30 threads. Fill the following table with appropriate values. Show the calculations along with the required formulae for the last column.<br>Note: Global and local thread indexing and block indexing shown in the table starts with 0. For blocks and threads(x,y,z) notation is used. | **4** | **4** | 1,2 | 5 |



| gridDim.x | gridDim.y | blockDim.x | blockDim.y | Global thread id of a thread (12,4) in block (2,3 |
|---|---|---|---|---|
| | | | | |

ii) Write a CUDA kernel to multiply matrix A with size mxn and matrix B with size nXp and produce the resultant matrix C. In the host write the code snippet which launches the above kernel with a single block and required number of threads in each block. The thread should be created such that every column of the resultant matrix should be computed by one thread.

| gridDim.x | gridDim.y | blockDim.x | blockDim.y | Global thread id of thread (3,4) block (2,3) |
|---|---|---|---|---|
| 32 | 4 | 6 | 5 | 2967 |

Writing values in table – **0.5M**

| | | | | | |
|---|---|---|---|---|---|
| | BlockId = blockIdx.y * gridDim.x  + blockIdx.x<br>threadId = BlockId * blockDim.x * blockDim.y  + threadIdx.y * blockDim.x + threadIdx.x    **0.5M**<br>BlockId = 3 * 32 + 2            =   98       **0.5M**<br>threadId = 98 *  6 * 5 +  4 * 6 + 3   =  2940 + 24 + 3 = 2967   **0.5M**<br>multiplyKernel_b<<<1, p>>>(d_a, d_b, d_c, m,n);<br>__global__ void multiplyKernel_colwise(int * a, int * b, int * c, int ha, int wa)<br>{<br>        int cidB = threadIdx.x;<br>        int wb = blockDim.x;<br>        int sum, k;<br>          for(ridA = 0; ridA < ha; ridA++)     **1M**<br>          {<br>                  sum = 0;<br>                  for( k=0; k< wa; k++)<br>                  {<br>                          sum += (a[ridA * wa + k] * b[k * wb + cidB]);<br>                  }<br>        c[ridA * wb + cidB] =sum;   **1M**<br>          }<br>}<br>**Scheme:**<br>**i)   2m**<br>**ii)   2m** | | | | |
| **13** | 1) Write a note on MPI_Scatter, MPI_Gather and MPI_Allgather collective communications along with syntax. Compare two MPI operations that are used for collective computations while one perform the operations across all the tasks in the group and place the result in one task and the other computes and puts the partial operation results on each processor.<br>**Scheme: 3 collectives with syntax: each 0.5M *3=1.5M**<br>**Identifying Reduce and syntax=0.5M**<br>**Identifying Scan and Syntax=0.5M**<br>**Comparison of reduce and scan =0.5M**<br>**Total=3M (images displaying the operation explanation can be considered along with syntax instead of writeup)**<br>Solution:<br>MPI_Scatter (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf,<br>        int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)<br>• Sends individual messages from the root process to all other processes | **3** | **2** | 1 | 3 |

| | | | | | |
|---|---|---|---|---|---|
| | • Inverse to MPI_Gather<br>• sendbuf is ignored by all non-root processes<br><br>MPI_Gather (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf,<br>int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)<br>• One process (root) collects data from all the other processes in the same communicator (i.e each process in comm (including root itself) sends its sendbuf to root.)<br>• The root process receives the messages in recvbuf in rank order<br>Must be called by all the processes with the same arguments<br><br>MPI_Allgather (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf,<br>int recvcnt, MPI_Datatype recvtype, MPI_Comm comm)<br>• All the processes collects data from all the other processes in the same communicator (i.e similar to MPI_Gather except now all processes receive the result.)<br>• recvbuf is NOT ignored<br>• Must be called by all the processes with the same arguments<br><br>MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype,<br>MPI_Op op, int root, MPI_Comm comm)<br>• One process (root) collects data from all the other processes in the same communicator, and performs an operation on the data (i.e combines elements provided by input buffer of each process in the group using operation op.)<br>• Returns combined value in the output buffer of process with rank root<br>• MPI_Scan (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm )<br>• A reduction means all processors get the same value while scan returns the partial operation results on each processor<br>• For example:<br>o if you had 10 processors and you were taking the sum of their rank, MPI_Reduce would give you the scalar 45 (0+1+2+3+4+5+6+7+8+9) on the root process,<br>o while MPI_scan would give you the scalar of the reduction up to the rank of the processor on each processor. So processor 0 would get 0, processor 1 would get 1, processor 2 would get 3, and so on. Processor 9 would get 45 | | | | |
| **14** | 2) Write an MPI Write a program in MPI to get the following output.<br><br>Input string read by root: MIT TOP HUT<br>I am rank 0, my string is: TIM<br>I am rank 1, my string is: POT POT POT<br>I am rank 2, my stirng is: TUH TUH TUH TUH TUH<br>Output string displayed by root: TIM POT POT POT TUH TUH TUH TUH TUH TUH | **3** | **2** | 1 | 4 |

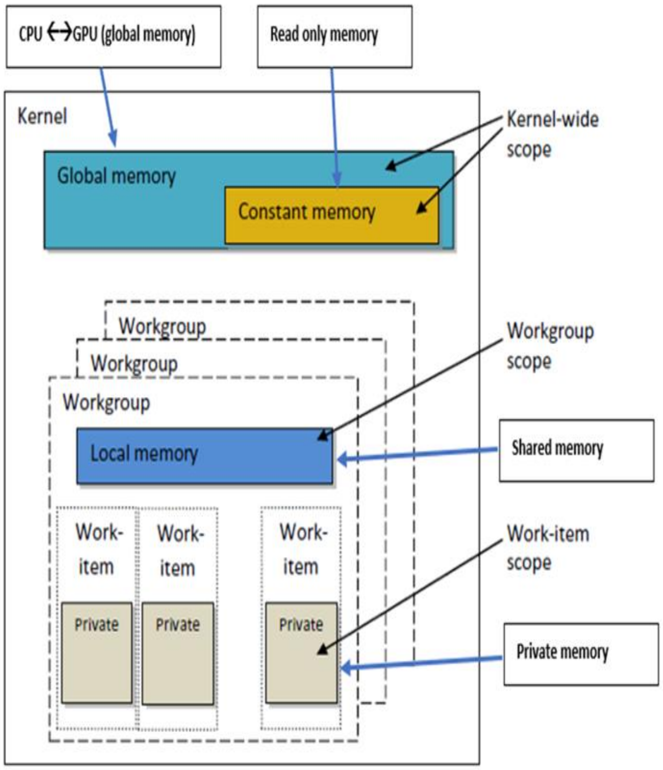
The root will distribute words in the string to the processes including itself and collects to print the final output. Each process is involved in reversing the word/partial string received and repeating the input word equal to (word length * rank ) times. If its rank 0, at least 1 time the reversed string needs to be printed. Use collective communication to distribute the words from the input string and use point to point communication to send the repeated string to root process. Each process should print its computed string as well.
Note: Take a string of equal word length to keep the problem simple.

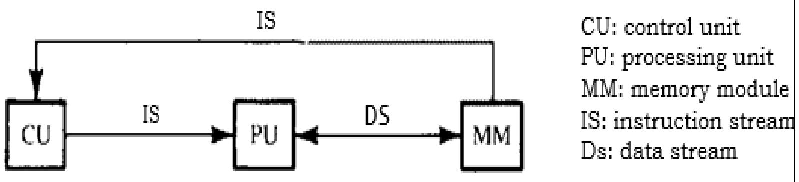
**Scheme: Reversing and computing new length= 1 M**
**Each process displaying its output =1M**
**Final output and total correctness=1M**

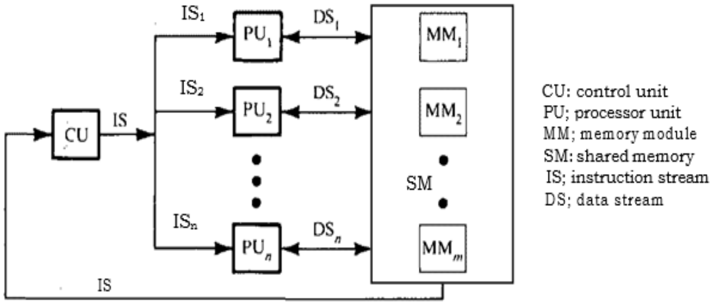
```
#include<string.h>
#include <stdio.h>
#define NPROCS 3
int main(int argc, char *argv[])
{
  int       rank, numtasks;
  char str[50],rstr[10],temp[50],t[50]=” “;
  char t1[20];
  int i,j,k,h,len,r;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    printf(" %d ",numtasks);
 if(rank==0)
 {
   strcpy(str, "MIT TOP HUT");
  for(i=0;str[i]!='\0';i++)
{
  if(str[i]==' ')
 {
  for(j=i;str[j]!='\0';j++)

   str[j]=str[j+1];
   str[j]='\0';
}
else
 continue;
}
Len =3;
 }

 MPI_Bcast(&len,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(str,len,MPI_CHAR,0,MPI_COMM_WORLD);
-  rstr[len]='\0';
printf("im %d process and sting i got is %s\n",rank,rstr);
for(i=0;i<=len/2;i++)
{
   char temp=rstr[i];
   rstr[i]=rstr[len-i-1];
   rstr[len-i-1]=temp;
}
 for(i=1;i<=rank+1;i++)
{
   strcat(t,str);
}
printf("im %d process and string modified is got is %s\n",rank,t);
int newlen;
if (rank == 0)
newlen=(rank+1)*len+1;
else newlen=rank*len +1;
if(rank!=0)
{
   MPI_Send(&newlen,1,MPI_INT,0,rank,MPI_COMM_WORLD);
  MPI_Send(t,newlen,MPI_CHAR,0,rank+1,MPI_COMM_WORLD);
}
MPI_Barrier(MPI_COMM_WORLD);
if(rank==0)
{

    strcpy(temp,t);
    printf("%s",temp);
for(i=1;i<=numtasks;i++)
{
  MPI_Recv(&newlen,1,MPI_INT,i,i,MPI_COMM_WORLD,&st);
   MPI_Recv(t1,newlen,MPI_CHAR,i,i+1,MPI_COMM_WORLD,&st);
  strcat(temp,t1);
  for(j=0;j<20;j++)
   t1[j]='\0';


}
printf("the new string is ");
puts(temp);
}
 MPI_Finalize();

  return 0;
}
```

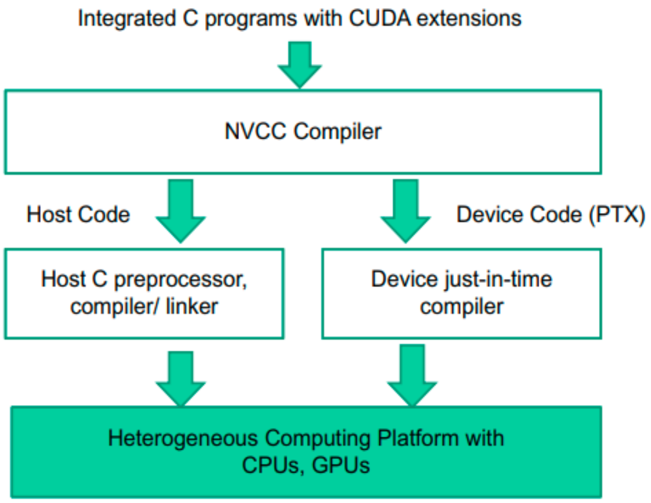
```
Input string read by root: MIT TOP HUT
I am rank 0, my string is: TIM
I am rank 1, my string is: POT POT POT
I am rank 2, my stirng is: TUH TUH TUH TUH TUH
Output string displayed by root: TIM POT POT POT TUH TUH TUH TUH TUH TUH
```

| | | | | | |
|---|---|---|---|---|---|
| **15** | | **3** | **3** | 1,2 | 4 |

| | | | | | |
|---|---|---|---|---|---|
| | Explain in brief why the devices in a context (opencl context) is limited to a specific platform. Write a neat diagram of the abstract memory model defined by OpenCL. (2M+1m)<br>Ans: i) Limiting the context to a given platform allows the programmer to provide context for multiple platforms and fully utilize s system comprising resources from a mixture of vendors.<br>ii)In addition, the application developer can target his application to different devices from different vendors and thus benefit from larger customer base.<br>**Scheme: Diagram 1M ,explanation each point 1M,1*2= 2M**<br><br> CPU ↔ GPU (global memory)<br>Read only memory<br>Kernel<br>Kernel-wide scope<br>Global memory<br>Constant memory<br>Workgroup<br>Workgroup<br>Workgroup<br>Workgroup scope<br>Local memory<br>Shared memory<br>Work-item<br>Work-item<br>Work-item<br>Work-item scope<br>Private<br>Private<br>Private<br>Private memory | | | | |
| **16** | Compare and contrast temporal parallelism and spatial parallelism. (3M)<br>**Scheme: each [1.5M]**<br>Temporal parallelism:<br>• Temporal parallelism or pipelining refers to the execution of a task as a 'cascade' of sub-tasks<br>• There exists one functional unit to carry out each sub-task<br>• All these successive units can work at the same time, in an overlapped fashion | **3** | **1** | 1 | 3 |

| | | | | | |
|---|---|---|---|---|---|
| | • As data are processed by a given unit Ui , they are sent to the next unit Ui+1 and the unit U i restarts its processing on new data, analogously to the flow of work in a car production line. Each functional unit can be seen as a "specialized" processor in the sense that it always execute the same sub-task<br><br>spatial parallelism:<br>• spatial parallelism refers to the simultaneous execution of tasks by several processing units<br>• At a given instant, these units can be executing the same task (or instruction) or different tasks. The<br>former case is called SIMD (Single Instruction stream, Multiple Data stream), whereas the latter is called MIMD (Multiple Instruction stream, Multiple Data stream) | | | | |
| **17** | **. Discuss the following with neat diagrams: SISD computer organization, MISD computer organization.**<br>**(3M)**<br><br>**Scheme: Diagrams [0.5M+0.5M]+ Explanation [1M+1M]**<br><br>SISD:<br><br>• This organization represents most serial computers available today<br><br>• Instructions are executed sequentially but may be overlapped in their execution stages (pipelining)<br><br>• An SISD computer may have more than one functional unit in It. All the functional units are under the<br><br>supervision of one control unit<br><br>• Applications: Whatever we do with our personal computers today<br><br>  IS<br>CU<br>IS<br>PU<br>DS<br>MM<br>CU: control unit<br>PU: processing unit<br>MM: memory module<br>IS: instruction stream<br>Ds: data stream<br><br>SIMD:<br><br>• In this organization, there are multiple processing elements supervised by the same control unit | **3** | **1** | 1 | 3 |

| | | | | | |
|---|---|---|---|---|---|
| | • All PEs receive the same instruction broadcast from the control unit but operate on different data sets<br>from distinct data streams<br>• The SIMD model of parallel computing consists of two parts:<br>▪ A front-end computer of the usual von Neumann style<br>▪ And a processor array<br>• Applications:<br>▪ Image processing<br>▪ Matrix manipulations<br>▪ Sorting<br> IS<br>CU<br>IS$_1$<br>IS$_2$<br>IS$_n$<br>PU$_1$<br>PU$_2$<br>PU$_n$<br>DS$_1$<br>DS$_2$<br>DS$_n$<br>MM$_1$<br>MM$_2$<br>MM$_m$<br>SM<br>IS<br>CU: control unit<br>PU; processor unit<br>MM; memory module<br>SM: shared memory<br>IS; instruction stream<br>DS; data stream | | | | |
| **18** | Illustrate the compilation process of a CUDA C program with the help of a neat diagram<br>• Each CUDA source file can have a mixture of **both host and device code**.<br>• By default, any traditional C program is a CUDA program that contains only host code. One can add device functions and data declarations into any C source file by marking them with **special CUDA keywords**.<br>• The **NVIDIA C Compiler (NVCC)** separates the host code and the device code during compilation process.<br>❑ The host code is compiled with the host's standard C compilers and runs as an ordinary CPU process.<br>❑ The device code(kernels) is compiled by the NVCC and executed on a GPU device | **2** | **4** | 1,2 | 3 |

| | | | | | |
|---|---|---|---|---|---|
| | <br> Integrated C programs with CUDA extensions<br>NVCC Compiler<br>Host Code<br>Device Code (PTX)<br>Host C preprocessor, compiler/ linker<br>Device just-in-time compiler<br>Heterogeneous Computing Platform with CPUs, GPUs<br>**Diagram 1M explanation – 1M** | | | | |