PCAP (CSE 3252) 6th Sem Sessional-2 Scheme

Section - A(MCQ)

Answer all the questions. Section Duration: 20 mins

1) Choose the TRUE CUDA statement from the following (0.5)

cudaMalloc() can be used to allocate device shared memory	dereferencing device memory pointer in the host code can result in faster device memory access	cudaFree() can be used to deallocate host memory	dereferencing device memory pointer in host code can result in the runtime error

Correct option is: 4

2) In 1D convolution, for the given input array N = {1, 2, 3, 4, 5, 6, 7} and Mask array M = {3, 4, 5, 4 ,3}, the output array element, P [5] is _____. (0.5)

30	90	93	98

Correct option is: 2

3) The execution of each thread in CUDA is ___ (0.5)

only sequential	Only parallel	Either sequential or parallel	None of the above

Correct option is: 1

4) Choose the correct option with respect to CUDA kernel. (0.5)

i)The same kernel can be launched with different numbers of threadsat various parts of the host code.

ii)All the threads in a block share the same block index which can be accessed as the blockDim variable in a kernel.

i - T, ii - T	i - F, ii - T	i - T, ii - F	i - F, ii - F

Correct option is: 3

5) The time complexity of CUDA based Matrix-Matrix multiplication where each column in the output matrix is generated by a separate thread with first matrix size I x J and second matrix size J x K is (0.5)

theta(I x J) theta(I x K) theta(J x K) theta(I x J x K)

Correct option is: 1

6) Which one of the following statements is false? (0.5)

The configuration of thread structure in CUDA is usually based on the dimensionality of data

Total No. of threads in a block is always divisible by blockDim.y

The total number of threads configured in a grid will always be equal to that required for processing input data

CUDA allows 3-dimensional block structure

Correct option is: 3

7) Consider a grid with 1 block, whose configuration in x,y,z direction is (4,3,2). The number of threads occurring in the same block after the index (1,2,1) is _____ (0.5)

24 2 14 12

Correct option is: 2

8) When the sparse matrix shown below is represented in CSR format, the value of row_Ptr [2] is____ (0.5)

3 0 0 6 9

0 0 1 0 2

0 0 0 0 0

9 0 8 0 0

0 0 0 0 7

7 3 5 8

Correct option is: 3

9) Suppose we want to process an image of size 400*900 pixels. You would like to assign 1 thread to each pixel. You would like your thread blocks to be square (32*32). What is the total No. of threads created in the grid? (0.5)

360000 386048 1024 11250

Correct option is: 2

10) Suppose we want to process an image of size 400*900 pixels. You would like to assign 1 thread to each pixel. You would like your thread blocks to be square (32*32). How many inactive threads are there which are common in both inactive rows and inactive columns? (0.5)

14848 448 11648 None of

Correct option is: 2

Type: DES

Q11. Identify whether the following statements are True or False. If False, correct it and justify your answer. (2)

i) The pointers to variables in the constant memory are to be passed to kernel as parameters

Ans: False

The correct statement:

The pointers to variables in the constant memory do not need to be passed to the kernel as parameters. (0.5M)

Justification:

Kernel functions in CUDA access constant memory variables as global variables. Thus, their pointers do not need to be passed to the kernel as parameters. (0.5M)

Ii) We can keep both input array and mask array required for 1D convolution in the constant memory as both are read only.

Ans: False

The correct statement and reason:

Only mask array can be kept in Constant memory as it is read only an also the size is small.

Even though the input array is read only, its size is usually high, and we keep it in global memory as constant memory contents are cached during kernel execution and the cache size is small cannot hold big input array. 1M

2) There is a matrix A of size MxN where N is an odd integer. Write a CUDA kernel function to calculate the total number of elements lesser than the middle element on the left in the same row and the total number of elements which are greater than the middle element on the right in the same row. Each element of the output array must be generated by a separate thread and each thread resides in different block. Also, mention the kernel launch statement with configuration parameters. (2)

e.g. M=3, N=5

I/P matrix A:

1 2 **3** 4 5

3 9 **5** 6 2

6 5 **3** 9 2

Output Array:

4 2 1

Soln:

```
__global__ void find_LTGT_Middle(int *a, int N, int *out)
{
int M=gridDim.x;
int row=blockIdx.x;
int i,count=0;
for(i=0;i<N/2;i++)
if(a[row*N+i]<a[row*N+N/2])
count++;
for(i=N/2+1;i<N;i++)
if(a[row*N+i]>a[row*N+N/2])
count++;
out[row]=count;
}
```

Kernel launch Statement:

```
find_LTGT_Middle<<<M,1>>>(a, N, out);
```

Marks: kernel header, declaration of M,row: 0.5, calculating lesser: 0.5, calculating greater, assign to output array: 0.5, Kernel launch 0.5

3) A CUDA host program reads a matrix **A** of size **MxN** and finds the saddle points in matrix A. It produces an output matrix **S** of size **MxN** as follows: If the element of A is a saddle point, then it replaces the element by a value zero. Otherwise, it keeps the element as it is. It should also find out the total number of saddle points present in the matrix. It solves this problem in parallel by creating a grid of single block and 2D threads in this block. Kernel should accept only required arguments. Write a **CUDA kernel** which performs the above task. (3)

Note:

1. Saddle point is a value if it is maximum in the row and minimum in the column or vice versa
2. Write only the kernel invocation with configuration parameters in the host code.

Sample input:

Enter the value of M: 2 Enter the value of N: 3

Enter input matrix A:

7 4 8

3 2 1

The output matrix S:

7 0 8

0 2 1

Total saddle points: 2

Scheme:

```
dim3 numGrid(1,1,1);
dim numb(N,M,1);
saddle<<<numGrid,numb>>>(d_a,d_t,d_count);          0.5M
__global__ void saddle(int * A,int * S,int *count){
int cid = threadIdx.x;
int rid = threadIdx.y;
int N = blockDim.x;
int M = blockDim.y;                                 0.5M  // if all variables are correct

  int rmin=0,cmax=0,rmax=0,cmin=0;
  int ele = A[rid * N + cid];
  int i;
  int tot;
  for(i = 0;i<N;i++)
   { if(i==cid) continue;
  if(ele>A[rid*N+i])
  rmin++;
   else
  rmax++;
    }                                                       0.5M
  for(i = 0;i<M;i++)
   {
   if(i==rid) continue;
  if(ele<A[i*N+cid])    cmax++;
  else   | cmin++;
   }                                                    0.5M
```

```
if((rmin==0 && cmax==0) || (rmax==0 && cmin==0))
{
 atomicAdd(count,1);                                    0.5M
S[rid*N+cid]= 0;
}
else
S[rid*N+cid]=ele;
}                                                       0.5M   remaining logic
```

4) Write an efficient parallel CUDA kernel code which produces a two-dimensional character matrix RES of size NxN from a given input two-dimensional character matrix A of size NxN where N is an even number. The input matrix A must be sent to the kernel launched with (2, 2) grid and 2D block and generates the output matrix RES. Write the kernel code to incorporate the following conditions while producing every character of the output matrix RES in parallel. Also, mention the kernel launch statement with configuration parameters. (3)

1. All the border elements of the input A must be replaced with a special character '!'.

2. All the unfilled positions of every row with row index as a prime number are replaced with a special character '*'.

3. All the unfilled positions of every row with row index as a non- prime number are replaced with a special character '#'.

Note: Kernel should accept only required number of arguments

Sample Input :

Enter N (an even number): 6

Enter the input character matrix A:

I

WriTE

pCaP

SEm

LAB

eXAMS

Sample output RES:

! ! ! ! ! !

! r i t e !

! C a P * !

! E m * * !

! A B # # !

! ! ! ! ! ! .

```
/*  main() */
dim3 grid(2,2);
dim3 block(N/2,N/2);
CUDACount<<<grid,block>>>(d_A,N);   //  0.5M
__device__ int isPrime(int n)
{
        int flag=1;
        for (int i = 2; i <n; i++) {
        if (n % i == 0) {
                flag = 0;break;
        }
}
if (n <= 1)
        flag = 0;
return flag;
}                                                   //    0.5M
```

```
__global__ void CUDACount(char *A, int n)
{
char ch;
int row= blockIdx.y*blockDim.y+threadIdx.y;
int col= blockIdx.x*blockDim.x+threadIdx.x;
if ( col==n-1 || row==n-1 || row==0|| col==0)
        A[row*n+col]='!';
else{
if((!((A[row*n+col] >= 'a' && A[row*n+col] <= 'z') || (A[row*n+col]
>= 'A' && A[row*n+col] <= 'Z'))) && (isPrime(row)))
        A[row*n+col]='*';
else if ((!((A[row*n+col] >= 'a' && A[row*n+col] <= 'z') ||
(A[row*n+col] >= 'A' && A[row*n+col] <= 'Z'))) && !
(isPrime(row)))
        A[row*n+col]='#';
else{
        ch=A[row*n+col];
           }} }                                  //        (2M)
```