INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
Mid-Spring Semester 2022-23

Subject: HIGH PERFORMANCE PARALLEL PROGRAMMING (CS61064)
Solution

Section 1 - GPU/CUDA **[30]**

1. Short questions. **[7]**

(a) Which of the following statements is correct. **[1]**

A. Compared to CPU, GPUs have smaller register file, smaller L1/L2 cache with lower bandwidth.

B. Shared memory is private to SMs.

C. L1 Cache memory is unified for all SMs

Solution B

(b) Fill in the blanks. Consider a 1D CUDA kernel that computes on a 1D array A of size 2048 with launch parameters $<<<32, 64>>>$. The array is accessed using the following access expression.

```
...
i = blockIdx.x*blockDim.x + threadIdx.x;
s=32;
B[i]=A[(i+s)%2048];
...
```

The data point $A[401]$ will be loaded from global memory by a thread with $threadId.x =$ _____ and $blockIdx.x =$ _____. **[2]**

Solution Here every block has 64 threads, so each block process 64 elements. Here x=401 and blockDim.x=64 $(401 - 32)/64 = 5$ i.e first 5 blocks will process first 320 elements of the array. So the sixth block with blockIdx.x=**5** will process the next 64 array elements. The threadIdx.x of the thread which will process 401th data element in A is = 369-320= **49**.

(c) Consider a hypothetical GPU architecture where the warp size is 16 and a kernel program which is launched with a configuration where the total number of threads in a thread block is 32. The total number of warps launched per thread block is thus 2. Consider the following conditional statements in the kernel.

i. $if(threadIdx.x < 16)$
ii. $if(threadIdx.x\%2)$
iii. $if(threadIdx.x\%32)$
iv. $if(threadIdx.x < 8)$

Which of the above statements results in divergence? Give justification. **[2]**

Solution ii,iii,iv
Warp size =16 and block size = 32
Nos of warps per block = 32/16 = 2
Warp 0 will have threads with $threadIdx.x$ values ranging from 0 to 15, and Warp 1 will have threads with $threadIdx.x$ values ranging from 16 to 31.
The first statement results in a non-divergent execution pattern, whereas statements ii, iii, and iv will lead to divergent execution.

- $if(threadIdx.x < 16)$: All the threads in warp 0 will satisfy the condition hence no divergence All the threads in warp 1 will not satisfy the condition hence no divergence
- $if(threadIdx.x\%2)$ Threads with odd and even values of $threadIdx.x$ diverges within both warps.
- $if(threadIdx.x\%32)$ Warp 0 will diverge (thread with $threadIdx.x = 0$ and other threads) warp 1 will not diverge
- $if(threadIdx.x < 8)$ Warp 0 will diverge and warp 1 will not diverge.

(d) For a GPU architecture with warp size 16, the array access expression $A[16 + tid * 16]$ where $id = blockDim.x * blockIdx.x + threadIdx.x$ is coalesced. Is the statement correct? Explain with justification. **[2]**

Solution False
Warp size is 16
Assuming memory transaction width is 16.
The array access expression results in following access pattern
A[16+0] = A[16]
A[16+1*16]= A[32]
A[16+2*16]= A[48]
...
A[16+ 15*16] = A[256]
As for every array element access we are referring an element which is at offset 16 from previous memory access, the above access pattern results in non-coalesced memory access.

2. (a) Consider a kernel processing a 2D matrix of dimensions 1024x1024 where each thread is assigned to perform an operation on a single element of the matrix. The kernel is launched with grid configuration $(a, 64)$ and block configuration $(64, b)$. For a hypothetical GPU architecture where the maximum number of threads in a block is 512, what is the maximum value of b and the corresponding value of a?
Solution Maximum value of b is 8 as maximum number of threads per block is limited to 512 and accordingly the value of a is 32.

(b) Write the CUDA kernel of Matrix Multiplication with tiling.
Solution Refer to the slide for CUDA Module 3 on the course website.

(c) Assuming matrices of size $N \times N$, a tile of size $W \times W$ and warp size of w, discuss how the usage of shared memory optimizations in tiled Matrix Multiplication reduces the total number of global memory accesses.
Solution For matrix multiplication without shared memory, the number of memory accesses = $N^2 \times (N + N/w)$
In case of using shared memory in the tiled version of matrix multiplication, the total number of global memory access = the number of memory access for computing a tile in the output matrix = the number of memory access to load a single tile $\times$ the number of tiles to load the two input matrices.
Number of memory access per tile = $(W/w) \times W$.
Number of tiles to load the two input matrices = $2 \times (N/W)$.
Number of tiles = N^2/W^2
Number of global memory access = $W/w \times W \times 2 \times N/W \times N^2/W^2 = \frac{2 \times N^3}{W \times w}$

[2+6+3]

3. Assume there is a 1D input array **A** of **N** (where $N > 8$) integers, where each element is of the form A[i]=4*i. Consider the following kernel code snippet executing on a GPU architecture where the warp size is 8.

```
__global__ void divBranch(int* A, int* B, int M, int N, int k)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    if(A[tid]%16)
    {
        B[tid]+=sqrt(A[tid]);
        if(A[tid]%6==0)
        {
            B[tid]+=sqrt(A[tid]); B[tid]+=sqrt(A[tid]);
            B[tid]+=sqrt(A[tid]); B[tid]+=sqrt(A[tid]);
        }
        else
        {
            B[tid]+=sqrt(A[tid]); B[tid]+=sqrt(A[tid]);
        }
    }
}
```

The above code represents a kernel with nested levels of divergence. During the lifetime of the **first warp** (tid0-tid7), what are the total number of square root instructions that are executed. **[6]**

Solution The elements of the array A are 0,4,8,12,16,..... Let us consider the elements being accessed by warp '0'. The following table represents which threads are active depending upon the evaluation of the condition statements in each of the branches in the above kernel. The first row represents the **tid** values. The second row represents the values of A[tid]. The successive rows contain the values of A[tid] where A[tid] is marked bold if **tid** enters the branch. The number of bold values in each such row represents the number of active threads that satisfy the branch condition. The number of square root operations executed in each conditional block is equal to the number of active threads in a block multiplied by the number of square root operations specified in that block.

tid:	0	1	2	3	4	5	6	7	#active threads*#sqrt
A[i]	0	4	8	12	16	20	24	28	
A[i]%16	0	**4**	**8**	**12**	16	**20**	**24**	**28**	6*1
A[i]%6==0	0	4	8	**12**	16	20	**24**	28	2*4
A[i]%6!=0	0	**4**	**8**	12	16	**20**	24	**28**	4*2

Total square root operations: 6 + 8 + 8 = **22**

4. In a GPU system with memory transaction width of N, a global memory access can coalesce (bring in a single transaction) N consecutive floating point data values. Consider the following code snippet with `tid` being the global thread id.

```
__global__
void mem_access(float* A){
    int x=0;
        for(int i=1; i<=32; i=i*2)
                x+=A[tid*i];
}
```

Let the number of threads per block be 256 (tid = 0 to 255), the size of the array A be 8192 and the number of blocks in the grid be 8192/256 = 32. Assuming a warp size of 16, global memory read transaction width of 16, compute the total number of memory transactions made in the `for` loop by warp 1 (tid16-tid31). **[6]**

Solution For warp 1 (tid16-tid31), the number of memory transactions will depend on the current value of i i.e. the induction variable of the loop and the value of the transaction width. This is because the value of i dictates the range of array positions that are going to be accessed for every warp. Note, that each warp is going to execute for all values of i and can execute in an order which is specified by the GPU hardware scheduler. The total number of memory transactions for the program is the sum total of memory transactions for each warp and i value pair. In the following table, we illustrate the range of array positions accessed for the warp 1 and i pair. As observed from the

Table 1: Memory Accesses

Warp ID	i=1	i=2	i=4	i=8	i=16	i=32
1 (tid 16-31)	[16,17,...,31]	[32,34,..,62]	[64,68,...124]	[128,136,...248]	[256,272,...,496]	[512,544,...,992]
Memory Transactions	1	2	4	8	16	16

above table, the total number of memory transactions for warp size of 16 and transaction width 16 will therefore be (1+2+4+8+16+16)=**47**.