High Performance Parallel Programming

Week 2	7 August		
	8 August	Tutorial 1(Divergence)	
Week 3	21 August		
	22 August	Tutorial 2(Reduction)	Assignment 1 (Shared Memory)
Week 4	28 August		
	29 August	Tutorial 3(Fusion Coarsening)	Assignment 2 (Reduction)
Week 5	4 September		Assignment 3(Convolution Operation)
Week 6	11 September		
	12 September		

Tutorial 1

1) Correct statement for allocation of memory chunk for 1024 floating point data element in GPU device is (Assume mem_chunk is device pointer):

A. cudaMalloc((void **)&mem_chunk, 1024*sizeof(float))
B. cudaMalloc((void **)&mem_chunk, 1024)
C. cudaMalloc((float **)&mem_chunk, 1024*sizeof(float))
D. cudaMalloc((float **)&mem_chunk, 1024)

Ans: A

2) Consider the following declaration of variable in CUDA kernel:

```
__device__ int myvariable
```

Which of the following statement is TRUE:

A. The lifetime of myvariable is limited to kernel
B. The variable will be stored in global memory of GPU
C. The variable will be stored in shared memory of GPU
D. The scope of myvariable is limited to block

Answer: B

3) Consider a vector addition kernel launch vectorAdd<<<dim3(16, 8), dim3(32, 16)>>>(d_A, d_B, d_C, n);

What does the following CUDA kernel launch configuration imply:
A. 128 threads per block and 128 blocks in the grid.
B. 256 threads per block and 128 blocks in the grid.
C. 512 threads per block and 2048 blocks in the grid.
D. 512 threads per block and 128 blocks in the grid.

Ans: D
Solution:
Total number of blocks in the grid :16 * 8 = 128
Total number of threads in each block:32 * 16 =512

4) Consider the following kernel snippet.

```
__global__ void kernel1(float *A)
{
        int tid = threadIdx.x;
        int gid = blockIdx.x * blockDim.x + threadIdx.x;
        int loop_bound = threadIdx.x / WARP_SIZE;
        if(tid/WARP_SIZE)
        {
                for(i=0;i<loopbound;i++)
                {
                      A[gid]++;
                }
        }
}
```

Consider three different GPU architectures - A1 where WARP_SIZE is 8; A2 where WARP_SIZE is 16 and A3 where WARP_SIZE is 32. Match and pair the following columns of statements.

i. A1	
ii A2	a. kernel exhibits divergent behaviour
iii A3	b. kernel does not exhibit divergent behaviour

Select the correct option from the following.

A. i -> a, ii ->b, iii->b
B. i -> b, ii ->a, iii->b
C. i -> b, ii ->b, iii->b
D. i -> a, ii ->b, iii->a

Correct Answer: C

Detailed answer: The range of values for tid is [0...31] irrespective of the warp size of the architecture. No matter what the warp size is, it is evident from the conditional statement that threads in a warp will evaluate **tid/WARP_SIZE** to either a zero or non-zero value. For A1, there are 4 warps for a thread block of size 32, where the 1st warp containing thread ids [0,..,7] will evaluate the conditional statement to a zero value and the remaining warps will evaluate the statement to a non-zero value. Similar behaviour may be observed for the remaining architectures as well.

5) 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 following options is correct?

A. All conditional branches (i)-(iv) are divergent for all warps
B. Conditional branch (i) is not divergent for warp 0
C. Conditional branch (ii) is divergent for only warps 0
D. Conditional branch (iv) is divergent for both warp 0 and warp 1

Correct Answer: B

Solution

thread ids in warp 0: 0-15
thread ids in warp 1: 16-31

One can observe that all threads in warp 0 can satisfy threadIdx.x < 16

6) 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 the following grid and block configurations: <a,32,2>

blocks of <32,b,2>. For a hypothetical GPU architecture where the maximum number of threads in a block is 512, what are the values of a and b? Select the correct option from below.

A. a= 16, b=8
B. a=4, b =32
C. a=32, b=8
D. None of these

Correct Answer: C

Solution

For b=4, number of threads in a block is 32*8*2=512 which satisfies the constraint in the given question. Setting a=32, we ensure that a total of (32*32*2) * 512 = 1024*1024 threads are launched for processing the 1024x1024 matrix.

7) Consider the following kernel processing **A** which is a 1D array of 2048 elements on a GPU architecture where the warp size is 16. The kernel is launched with a configuration of <64,1,1> blocks of <32,1,1> threads. Assume each element of A is initialized to 0.

```
__global__ void kernel1( float *A)
{
        int tid = threadIdx.x;
        int gid = blockIdx.x * blockDim.x + threadIdx.x;
        int loop_bound = threadIdx.x / 8;
        if(tid%2)
        {
                for(i=0;i<loopbound;i++)
                {
                        A[gid]++;
                }
        }
}
```

After the execution of the above program, what would be the value of A[1035]?

A. 0
B. 1
C. 2
D. 4

Answer: B

For gid = 1035, the value of tid is 1035%32=11. The kernel thus enters the body of the conditional statement. The value of loopbound is 11 / 8 = 1 and thus A[1035] is incremented to 1.

8) Consider the following piece of divergent code:

```
__global__ void kernel(int *a, int *b, int *x, int *y){
        int i = threadIdx . x + blockDim . x * blockIdx . x ;
        if (a[i]==b[i])
                ++x;
        else
                ++y;
}
```

The above operations can be rewritten to avoid branch divergence as:

A.

```
__global__ void kernel(int *a, int *b, int *x, int *y){
        int i = threadIdx . x + blockDim . x * blockIdx . x ;
        if (a[i]==b[i])
                ++x;
         if (a[i] != b[i])
                 ++y;
}
```

B.

```
__global__ void kernel(int *a, int *b, int *x, int *y){
        int i = threadIdx . x + blockDim . x * blockIdx . x ;
        if (a[i]==b[i])
                ++x;
        else if (a[i] != *b[i])
        ++y;
}
```

C.

```
__global__ void kernel(int *a, int *b, int *x, int *y){
    int i = threadIdx . x + blockDim . x * blockIdx . x ;
    x += (a[i] ==b[i]);
    y += (a[i] !=b[i]);
}
```

D.

```
__global__ void kernel(int *a, int *b, int *x, int *y){
    int i = threadIdx . x + blockDim . x * blockIdx . x ;
    x += (a[i] !=b[i]);
    y += (a[i] ==b[i]);
}
```

Ans: C