

Student ID number:
--------------------

Student Last Name:
--------------------

**Guidelines.** The quiz consists of two exercises. All answers should be written in the *answer boxes*. No justifications for the answers are needed, unless explicitly required. You are expected to do this quiz on your own without assistance from anyone else in the class. If possible, please avoid pencils and use pens with dark ink. Thank you.

**CUDA Cheat Sheet .** This cheat sheet contains two examples of kernel code seen in class. The first one has also the code for launching the kernel.

#### CPU program

```
void increment_cpu(float *a, float b, int N)
{
    for (int idx = 0; idx < N; idx++)
        a[idx] = a[idx] + b;
}
```

```
void main()
{
    ....
    increment_cpu(a, b, N);
}
```

#### CUDA program

```
__global__ void increment_gpu(float *a, float b, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N)
        a[idx] = a[idx] + b;
}
```

```
void main()
{
    ....
    dim3 dimBlock (blocksize);
    dim3 dimGrid( ceil( N / (float)blocksize) );
    increment_gpu<<<dimGrid, dimBlock>>>(a, b, N);
}
```

```

__global__ void sum_kernel(int *g_input, int *g_output)
{
    extern __shared__ int s_data[ ]; // allocated during kernel launch

    // read input into shared memory
    unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;
    s_data[ threadIdx.x ] = g_input[ idx ];
    __syncthreads( );

    // compute sum for the threadblock
    for ( int dist = blockDim.x/2; dist > 0; dist /= 2 )
    {
        if ( threadIdx.x < dist )
            s_data[ threadIdx.x ] += s_data[ threadIdx.x + dist ];
        __syncthreads( );
    }

    // write the block's sum to global memory
    if ( threadIdx.x == 0 )
        g_output[ blockIdx.x ] = s_data[0];
    }
}

```

**Exercise 1.** Consider the three kernels below.

```

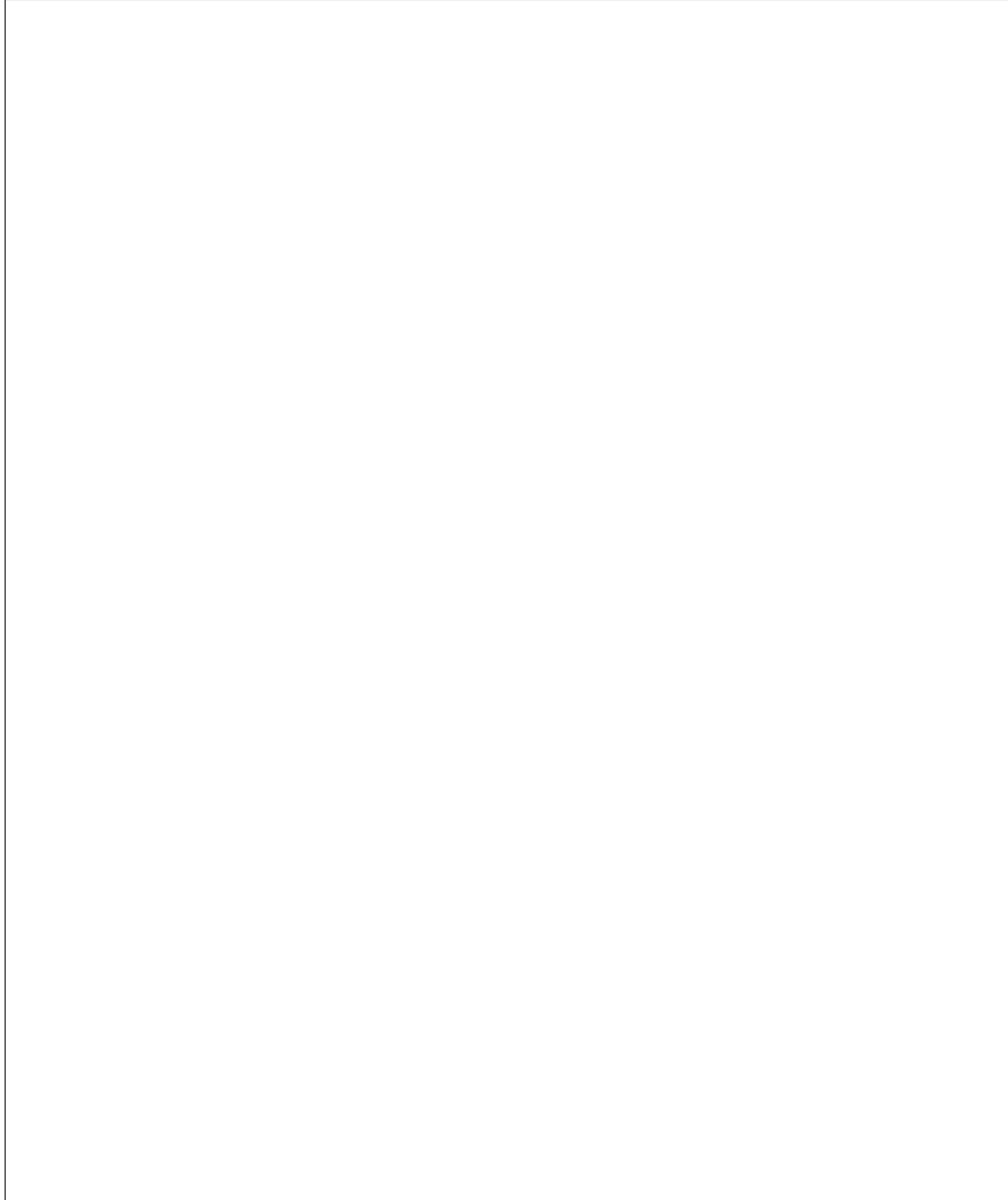
__global__ void kernel( int *a )
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    a[idx] = 7;
}

__global__ void kernel( int *a )
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    a[idx] = blockIdx.x;
}

__global__ void kernel( int *a )
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    a[idx] = threadIdx.x;
}

```

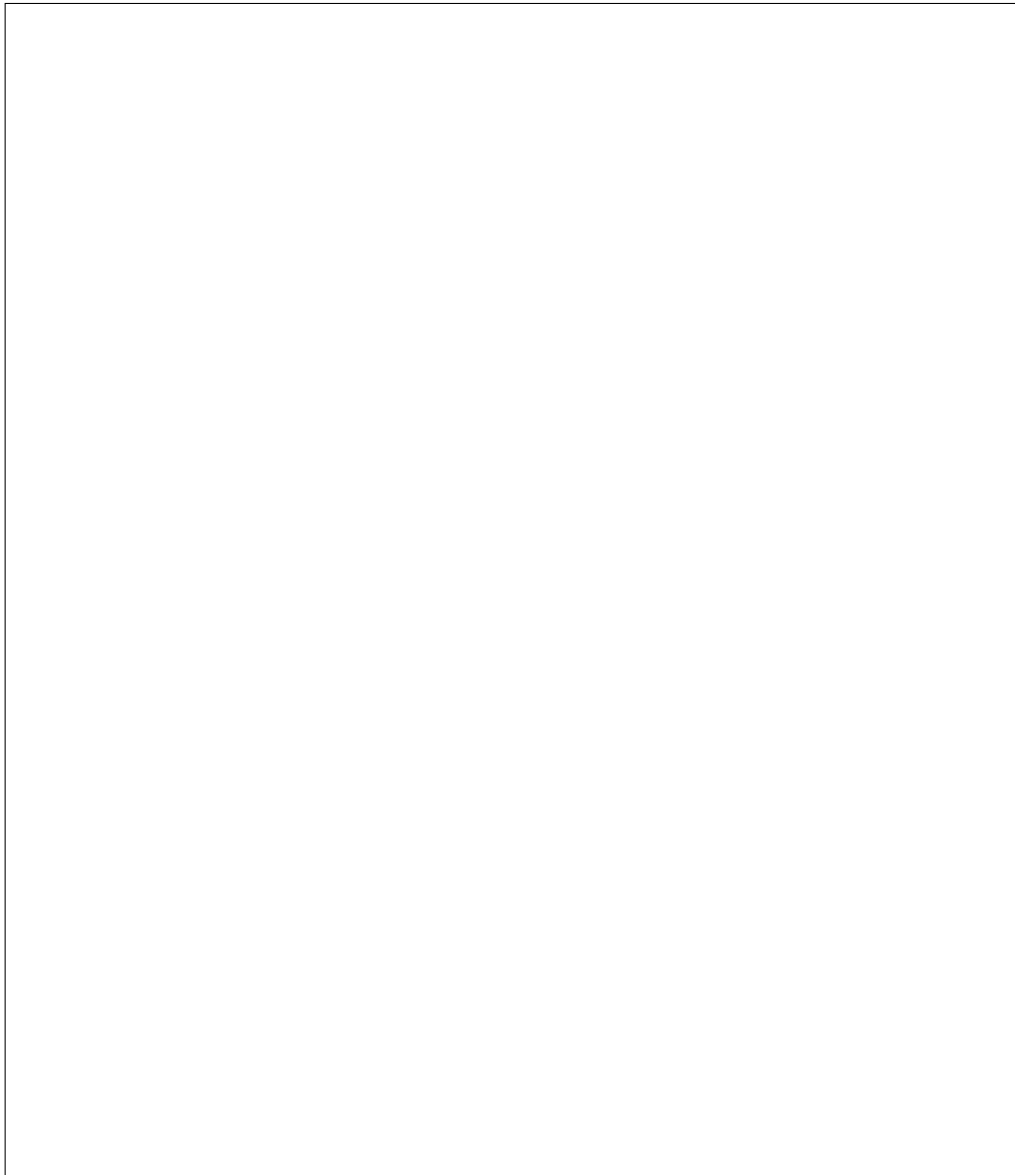
In each case, write the value of the array a

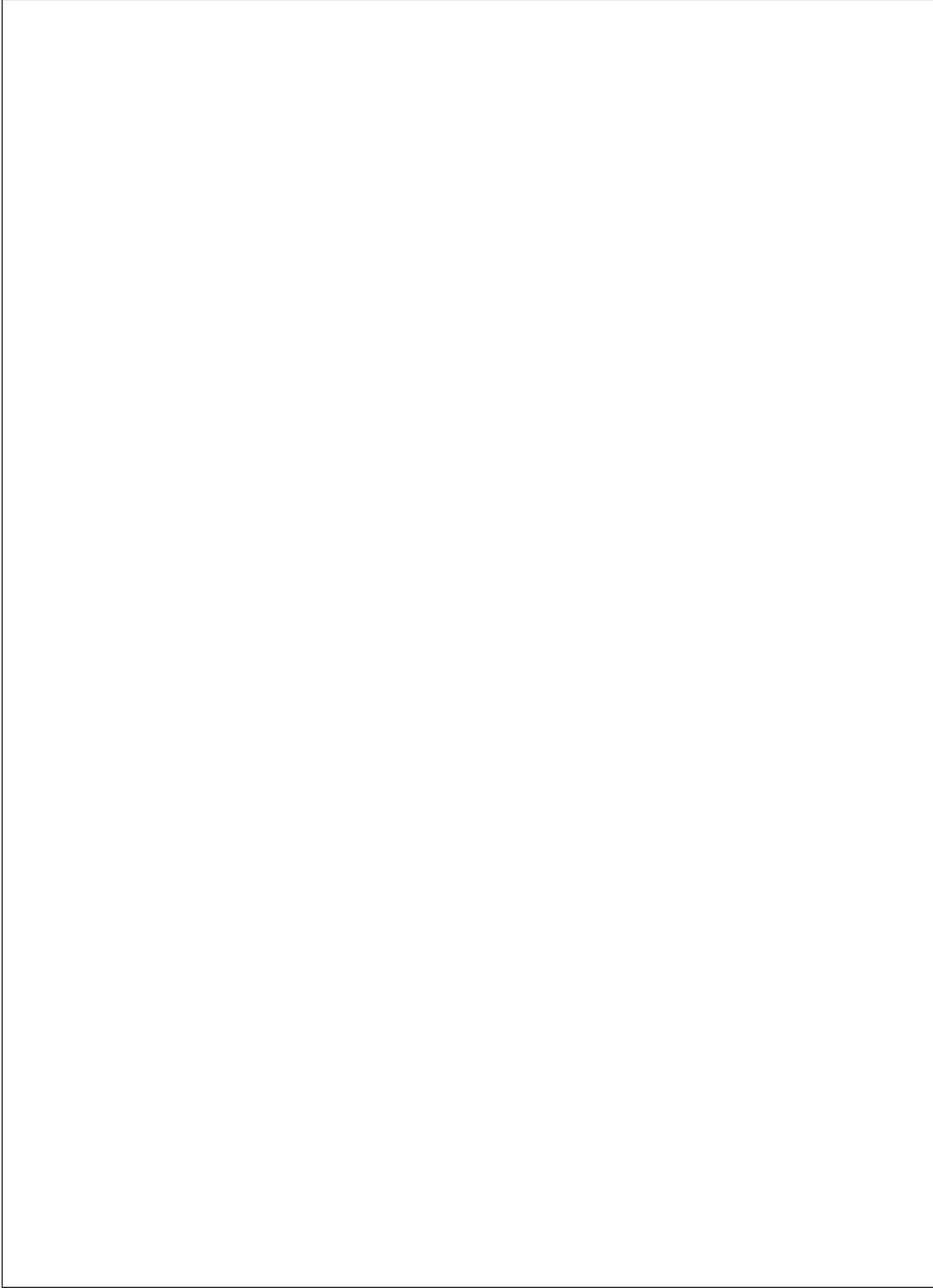


**Solution is on Slide 35 of [http://www.csd.uwo.ca/~moreno/HPC-Slides/Many\\_core\\_computing\\_with\\_CUDA.pdf](http://www.csd.uwo.ca/~moreno/HPC-Slides/Many_core_computing_with_CUDA.pdf)**

**Exercise 2.** Write a CUDA kernel (and the launching code) implementing the reversal of an input integer array  $A$  of size  $n$ . This reversing process will be out-of-place. You are asked to proceed in two steps.

- (1) First write a “naive” kernel which does not use shared memory.
- (2) Then, write a kernel using shared memory.





Solution is on Slide 110 of [http://www.csd.uwo.ca/~moreno/HPC-Slides/Optimizing\\_CUDA\\_Code.pdf](http://www.csd.uwo.ca/~moreno/HPC-Slides/Optimizing_CUDA_Code.pdf)