Exercises for GPU – CUDA

From book: Programming Massively Parallel Processors – Kirk, Hwu

3.5. If we need to use each thread to calculate one output element of a vector addition, what would be the expression for mapping the thread/block indices to data index:
(A) i = threadIdx.x + threadIdx.y
(B) i = blockIdx.x + threadIdx.x
(C) i = blockIdx.x*blockDim.x + threadIdx.x
(D) i = blockIdx.x*threadIdx.x

3.6 We want to use each thread to calculate two (adjacent) elements of a vector addition. Assume that a variable i should be the index for the first element to be processed by a thread. What would be the expression for mapping the thread/block indices to data index?
(A) i = blockIdx.x*blockDim.x + threadIdx.x + 2
(B) i = blockIdx.x*threadIdx.x*2
(C) i = (blockIdx.x*blockDim.x + threadIdx.x)*2
(D) i = blockIdx.x*threadIdx.x*2 + threadIdx.x

3.7. For a vector addition, assume that the vector length is 2000, each thread calculates one output element, and the thread block size is 512 threads. How many threads will be in the grid?
(A) 2000
(B) 2024
(C) 2048
(D) 2096

4.1. If a CUDA device's SM (streaming multiprocessor) can take up to 1536 threads and up to 4 thread blocks. Which of the following block configuration would result in the most number of threads in the SM?
(A) 128 threads per block
(B) 256 threads per block
(C) 512 threads per block
(D) 1024 threads per block

4.4. You need to write a kernel that operates on an image of size 400x900 pixels. You would like to assign one thread to each pixel. You would like your thread blocks to be square and to use the maximum number of threads per block possible on the device (your device has compute capability 3.0). How would you select the grid dimensions and block dimensions of your kernel?

4.5. For the previous question, how many idle threads do you expect to have?

4.7. Indicate which of the following assignments per multiprocessor is possible. In the case where it is not possible, indicate the limiting factor(s).

a) 8 blocks with 128 threads each on a device with compute capability 1.0

b) 8 blocks with 128 threads each on a device with compute capability 1.2

c) 8 blocks with 128 threads each on a device with compute capability 3.0

d) 16 blocks with 64 threads each on a device with compute capability 1.0

e) 16 blocks with 64 threads each on a device with compute capability 1.2

f) 16 blocks with 64 threads each on a device with compute capability 3.0

Technical specifications	Compute capability (version)									
	1.0	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.5**	**3.7**	**5.0**	**5.2**
Maximum dimensionality of grid of thread blocks	2				3					
Maximum x-dimension of a grid of thread blocks	65535					2^{31}-1				
Maximum y-, or z-dimension of a grid of thread blocks	65535									
Maximum dimensionality of thread block	3									
Maximum x- or y-dimension of a block	512				1024					
Maximum z-dimension of a block	64									
Maximum number of threads per block	512				1024					
Warp size	32									
Maximum number of resident blocks per multiprocessor	8					16			32	
Maximum number of resident warps per multiprocessor	24		32		48	64				
Maximum number of resident threads per multiprocessor	768		1024		1536	2048				
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K	64 K		128 K	64 K	
Maximum number of 32-bit registers per thread	128				63		255			
Maximum amount of shared memory per multiprocessor	16 KB				48 KB			112 KB	64 KB	96 KB
Number of shared memory banks	16				32					
Amount of local memory per thread	16 KB				512 KB					
Constant memory size	64 KB									
Technical specifications	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.5**	**3.7**	**5.0**	**5.2**
	Compute capability (version)									

CUDA Thread Indexing Cheatsheet

1D grid of 1D blocks

```
__device__ int getGlobalIdx_1D_1D()
{
        return blockIdx.x *blockDim.x + threadIdx.x;
}
```

1D grid of 2D blocks

```
__device__ int getGlobalIdx_1D_2D()
{
        return blockIdx.x * blockDim.x * blockDim.y + threadIdx.y * blockDim.x + threadIdx.x;
}
```

1D grid of 3D blocks

```
__device__ int getGlobalIdx_1D_3D()
{
        return blockIdx.x * blockDim.x * blockDim.y * blockDim.z
        + threadIdx.z * blockDim.y * blockDim.x + threadIdx.y * blockDim.x + threadIdx.x;
}
```

2D grid of 1D blocks

```
__device__ int getGlobalIdx_2D_1D()
{
        int blockId   = blockIdx.y * gridDim.x + blockIdx.x;
        int threadId = blockId * blockDim.x + threadIdx.x;
        return threadId;
}
```

2D grid of 2D blocks

```
 __device__ int getGlobalIdx_2D_2D()
{
        int blockId = blockIdx.x + blockIdx.y * gridDim.x;
        int threadId = blockId * (blockDim.x * blockDim.y) + (threadIdx.y * blockDim.x) +
threadIdx.x;
        return threadId;
}
```

2D grid of 3D blocks

```
__device__ int getGlobalIdx_2D_3D()
{
        int blockId = blockIdx.x
                         + blockIdx.y * gridDim.x;
        int threadId = blockId * (blockDim.x * blockDim.y * blockDim.z)
                          + (threadIdx.z * (blockDim.x * blockDim.y))
                          + (threadIdx.y * blockDim.x)
                          + threadIdx.x;
        return threadId;
}
```

3D grid of 1D blocks

```
__device__ int getGlobalIdx_3D_1D()
{
        int blockId = blockIdx.x
                         + blockIdx.y * gridDim.x
                         + gridDim.x * gridDim.y * blockIdx.z;
        int threadId = blockId * blockDim.x + threadIdx.x;
        return threadId;
}
```

3D grid of 2D blocks

```
__device__ int getGlobalIdx_3D_2D()
{
        int blockId = blockIdx.x
                      + blockIdx.y * gridDim.x
                         + gridDim.x * gridDim.y * blockIdx.z;
        int threadId = blockId * (blockDim.x * blockDim.y)
                          + (threadIdx.y * blockDim.x)
                          + threadIdx.x;
        return threadId;
}
```

3D grid of 3D blocks

```
__device__ int getGlobalIdx_3D_3D()
{
        int blockId = blockIdx.x
                         + blockIdx.y * gridDim.x
                         + gridDim.x * gridDim.y * blockIdx.z;
        int threadId = blockId * (blockDim.x * blockDim.y * blockDim.z)
                          + (threadIdx.z * (blockDim.x * blockDim.y))
                          + (threadIdx.y * blockDim.x)
                          + threadIdx.x;
        return threadId;
}
```