Grid Organization

- The simplest way to organize your threads is launch your kernel as follows:

```
mykernel<<<bpg, tpb>>>( ... );
```

  where **bpg** (blocks per grid) and **tpb** (threads per block) are integers. Note **tpb** must not exceed the total number of threads per block allowed by your graphics card.

- You can also organize your threads as a three dimensional array of blocks, with block itself being another 3D array.

```
dim3 bpg(num_blocks_x, num_blocks_y, num_blocks_z);
dim3 tpb(num_threads_x, num_threads_y, num_threads_z);
mykernel<<<bpg, tpb>>>( ... );
```

- If you do a **deviceQuery**, you can find out the maximum dimensions of a grid of blocks, and a block of threads.

- For the GTX470 cards in the lab, the maximum grid dimensions are $65535 \times 65535 \times 65535$.

- If you are using a newer card, your maximum grid dimensions may be $2147483647 \times 65535 \times 65535$.

- When specifying a block of threads, you must not exceed the maximum size per dimension AND the total number of threads in your block must not exceed the maximum number allowed. For example, the GTX470 card allows a maximum of 1024 threads per block with a maximum dimension size of a thread block as $1024 \times 1024 \times 64$. Therefore the following are valid:

```
dim3 tpb(1024, 1, 1);
dim3 tpb(512, 2, 1);
dim3 tpb(32, 16, 2);
```

  However, the following are not valid:

```
dim3 tpb(1024, 64, 1);
dim3 tpb(2048, 1, 1);
dim3 tpb(1, 1, 1024);
```

- By default the **nvcc** compiler will assume a CUDA compute capability of 2.0 (as is the case for the GTX470). Newer cards will have a higher compute capability (see the **deviceQuery** output). If you want to take advantage of this and the higher grid dimensions capability, you need to tell the compiler, eg

```
nvcc -arch=compute_35 -o myprog myprog.cu
```