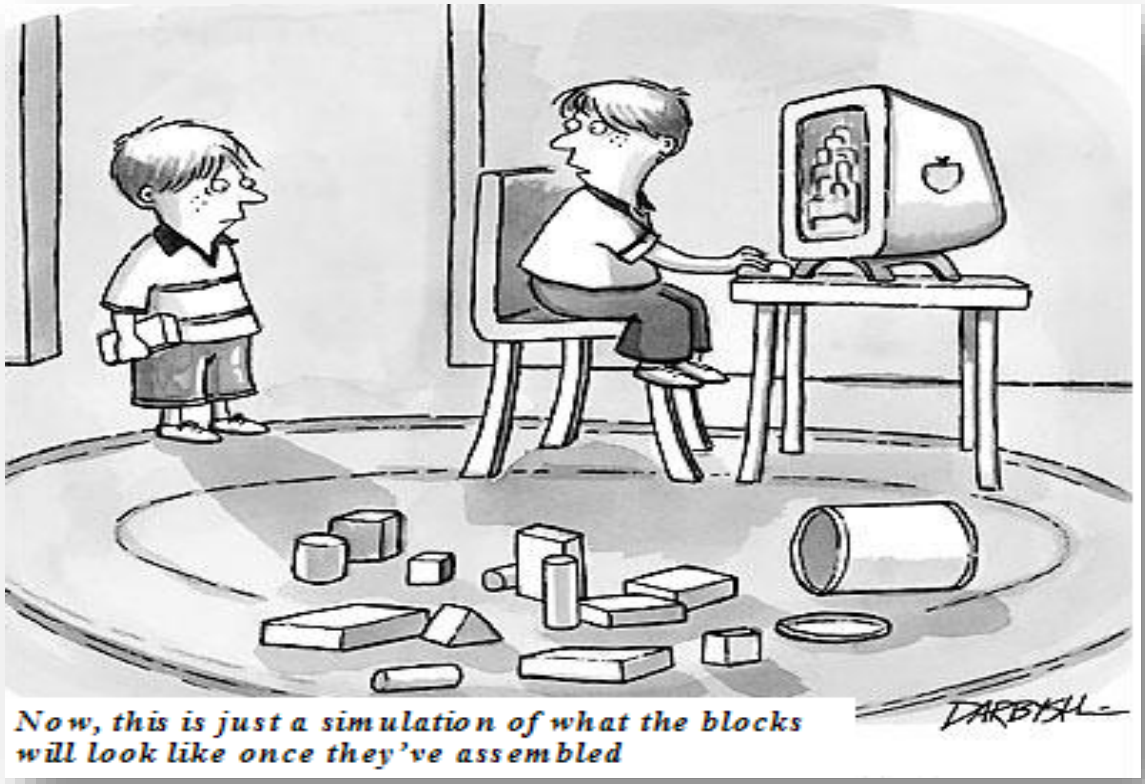
ECE569
Module 15



- CUDA Programming model

Utilizing hardware execution resources

- **Assigning thread blocks to execution resources**
  - Mapping data to threads
- **Capacity constrains of execution resources**
  - Limitations on number of threads, blocks,
  - Memory (local, shared, global)
- **Zero-overhead thread scheduling**
  - Scheduling of threads
  - No waiting, let a thread start execution as soon as its resources are available

Block Size Options vs Thread and Thread Block Utilization

- For an image blur kernel, which thread block configuration allows best thread utilization for a GPU with the following capabilities: **1536 threads/SM, 1024 threads/block and 8 resident blocks/SM?**

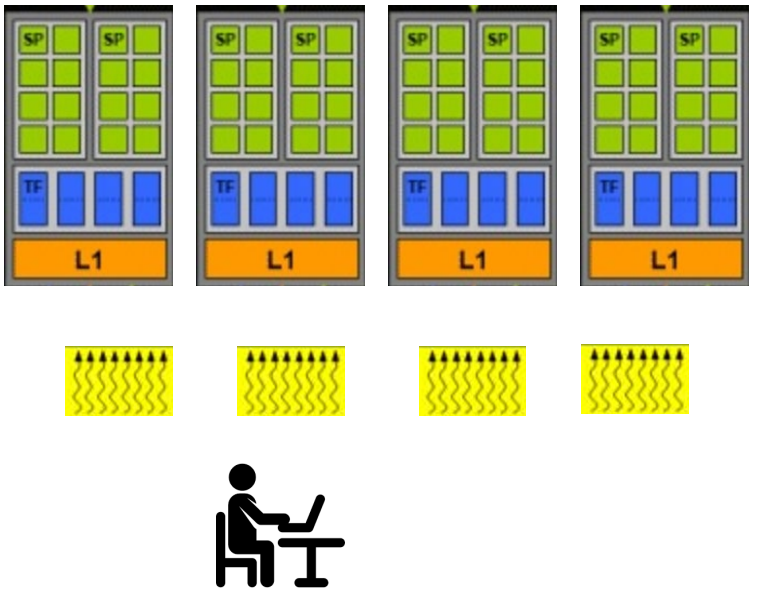
- ❑ 4x4
- ❑ 8X8
- ❑ 16X16
- ❑ 32X32

Programming Model

- **Which of the following statements are correct?**
  - ❑ A thread block contains many threads
  - ❑ An SM may run more then one block
  - ❑ A block may run on more then one SM
  - ❑ All threads in a thread block may cooperate to solve a sub problem
  - ❑ All the threads that run on a given SM may cooperate to solve a problem

Programming Model

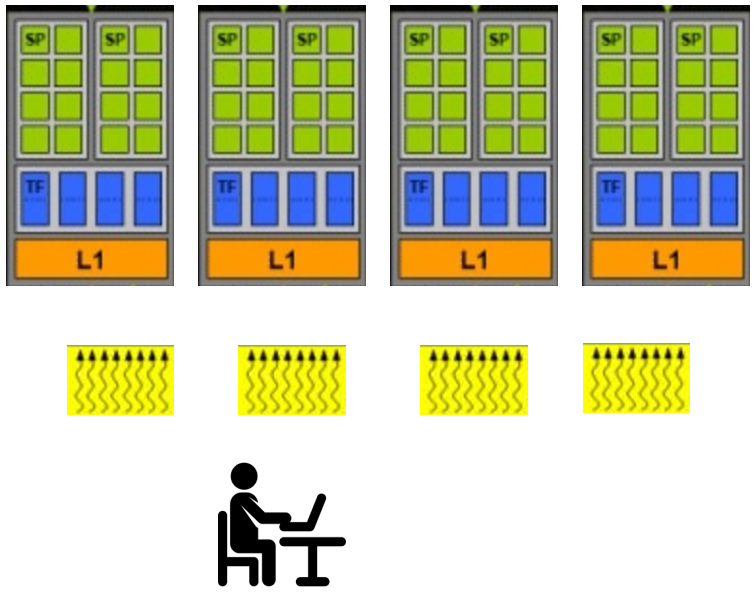
- **The _____ is responsible for defining thread blocks in software**
  - ❑Programmer
  - ❑GPU
- **The _____ is responsible for allocating thread blocks to hardware SMs**
  - ❑Programmer
  - ❑GPU



Programming Model

- **Given a single kernel that is launched on many thread blocks including X and Y, the programmer can specify:**
  - ❑ That Block X will run at the same time as block Y
  - ❑ That Block X will run after Block Y
  - ❑ That Block X will run on a specific SM.

  Choose all that apply!



Programming Model

- **CUDA makes few guarantees**
  - Threads in a block run on the same SM at the same time
  - All blocks in a kernel finish before any blocks from the next kernel run
  - No waiting on thread blocks
    - If a thread block is waiting, scheduler will issue other thread blocks from the same kernel
    - A thread in a block must complete for that block to complete
  - Scalability
    - Same code will run on resource limited/resource rich GPU without code modification.
  - No guarantees on when and where thread blocks run
    - Consequence: no communication between blocks! Programmer can not guarantee certain order
      - Block X waiting for Block Y, but Y already exited (deadlock)

Programming Model

```
#include <stdio.h>
#define NUM_BLOCKS 16
#define BLOCK_WIDTH 1

__global__void hello() {
printf("Hello world, I am thread in block %d\n", blockIdx.x);}

int main(int argc, char** argv) {
    hello<<<NUM_BLOCKS, BLOCK_WIDTH>>>();
    cudaDeviceSynchronize();
    printf("Done\n");
    return 0; }
```

- **How many different outputs can different run of this program produce?**
  - ❑1
  - ❑16
  - ❑65,536
  - ❑21 trillion

Image Blur Kernel

```
// Get the average of the surrounding 2xBLUR_SIZE x 2xBLUR_SIZE box
        for(int blurRow = -BLUR_SIZE; blurRow < BLUR_SIZE+1; ++blurRow) {
          for(int blurCol = -BLUR_SIZE; blurCol < BLUR_SIZE+1; ++blurCol) {

            int curRow = Row + blurRow;
            int curCol = Col + blurCol;
            // Verify we have a valid image pixel
            if(curRow > -1 && curRow < h && curCol > -1 && curCol < w) {
              pixVal += in[curRow * w + curCol];
              pixels++; // Keep track of number of pixels in the accumulated total
            }
          }
        }

        // Write our new pixel value out
        out[Row * w + Col] = (unsigned char)(pixVal / pixels);
```

Global Memory Only: Time Spent On Memory

- **Assume that all threads access global memory**
  - One memory access (4 bytes) per floating-point add
  - Peak single precision floating-point rate ~4000 GFLOPS
    - **What will be the memory demand at this peak rate in B/s?**

Global Memory Only: Time Spent On Memory

- **Assume that all threads access global memory**
  - One memory access (4 bytes) per floating-point add
  - Peak single precision floating-point rate ~4000 GFLOPS
    - **What will be the memory demand at this peak rate in B/s?**
    - 4*4,000 = **16,000 GB/s** required for peak FLOPS rating
  - with 250 GB/s DRAM bandwidth

Global Memory Only: Time Spent On Memory

- **Assume that all threads access global memory**
  - One memory access (4 bytes) per floating-point add
  - Peak single precision floating-point rate ~4000 GFLOPS
    - **What will be the memory demand at this peak rate in B/s?**
    - 4*4,000 = **16,000 GB/s** required for peak FLOPS rating
  - with 250 GB/s DRAM bandwidth
    - 250/16,000 = 1.56%
    - 250/4 = 62.5GFLOPS achieved execution throughput
    - 62.5/4000 = 1.56%
  - Need to drastically cut down memory accesses to get close to 4000 GFLOPS

Global Memory Only: Time Spent On Memory

- **Does a new generation GPU resolve this problem?**
- Assume device offers 720GB/s DRAM bandwidth and peak throughput of 9300 GFLOPS
  - With 1 memory (4B) access per floating point addition

Global Memory Only: Time Spent On Memory

- **Does a new generation GPU resolve this problem?**
- Assume device offers 720GB/s DRAM bandwidth and peak throughput of 9300 GFLOPS
  - With 1 memory (4B) access per floating point addition
    - 720/4 = 180 GFLOPS (almost 3 times better than previous GPU)
    - DRAM is still bottleneck
    - Peak performance achieved is 180/9300 = 1.94% utilization.
  - Programmer needs to utilize fast memory!!

Writing efficient programs

- **Maximize arithmetic intensity**
  - Math per amount of Memory access
    - Maximize work per thread
      - Maximize compute operations per thread
    - Minimize memory per thread
      - Minimize time spent on memory per thread
- **Minimize time spent on memory**
  - Computer Organization
    - CPU: Registers, Cache

Next

- **Introduce fast memory available on the GPU**
  - Memory types