Introduction to CUDA

CAP 5705

Fall 2012

Tushar Athawale

Resources

- CUDA Programming Guide
- Programming Massively Parallel Processors: A Hands-on Approach
  - David Kirk

Motivation

- Process independent tasks in parallel for a given application.
- How can we modify 'line drawing routine'?

a)Divide line into parts and assign each part to each processor.

b)What if we assign a processor per scanline?

(Each processor knows its own y coordinate)

- Ray tracing?

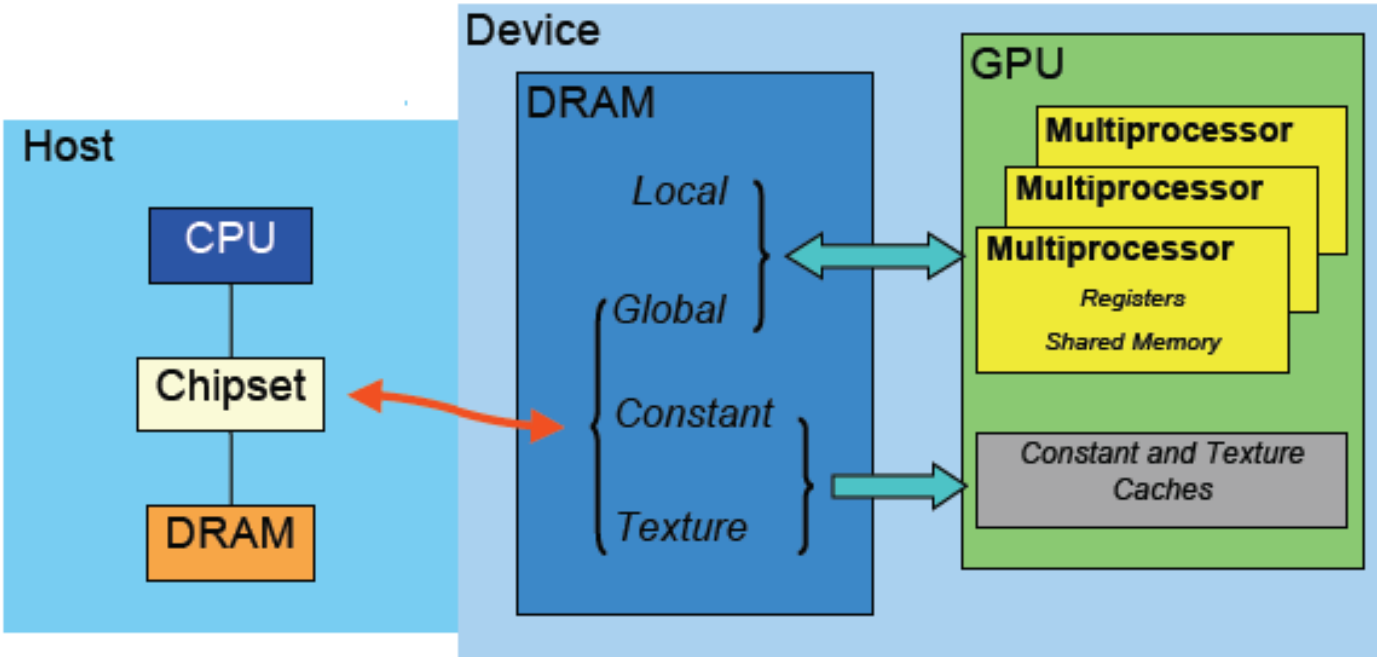
Memory Model

- Host (CPU) and device (GPU) have separate memory spaces
- Host manages memory on device
  - Use functions to allocate/set/copy/free memory on device
  - Similar to C functions

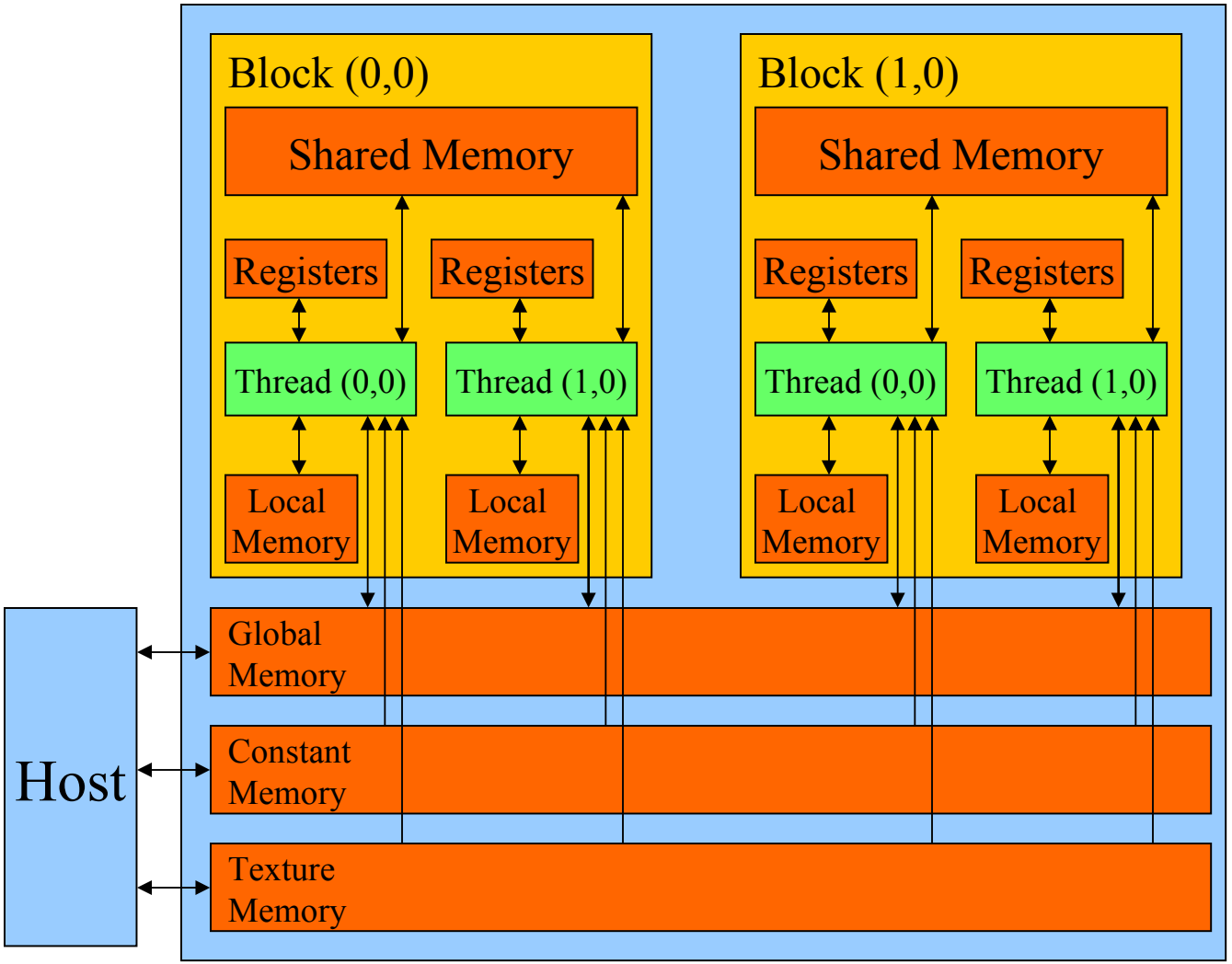
Memory Model

- Types of device memory
  - Registers – read/write per-thread
  - Local Memory – read/write per-thread
  - Shared Memory – read/write per-block
  - Global Memory – read/write across grids
  - Constant Memory – read across grids
  - Texture Memory – read across grids

Memory Model



Memory Model



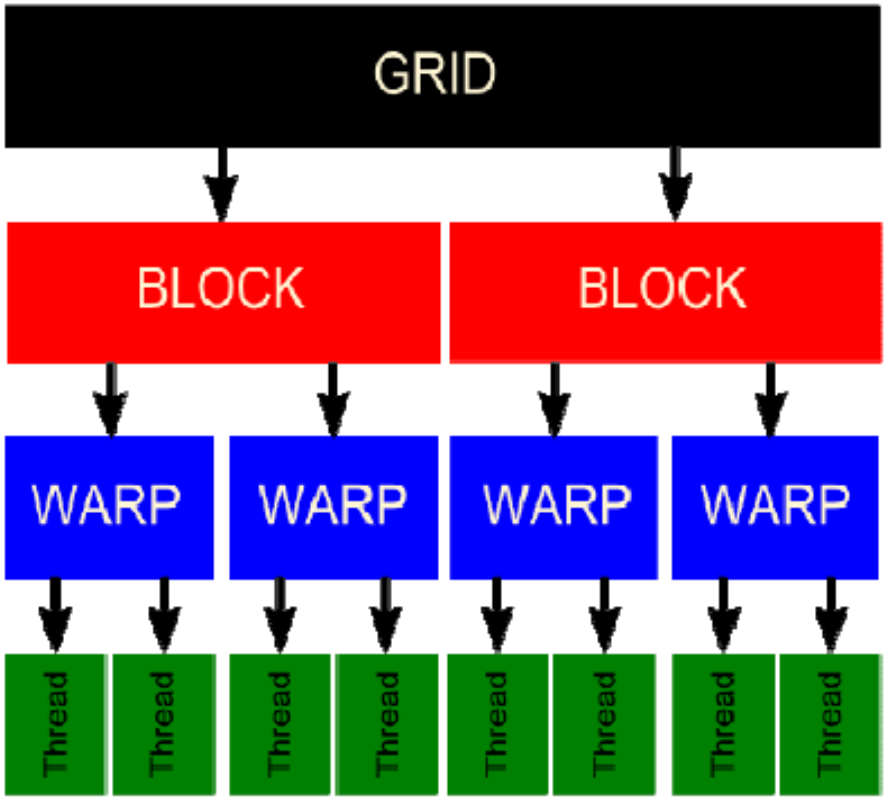
Programming Model

- SIMT (Single Instruction Multiple Threads)
- Threads run in groups of 32 called warps
- Every thread in a warp executes the same instruction at a time

Programming Model

- A single kernel executed by several threads
- Threads are grouped into ‘blocks’
- Kernel launches a ‘grid’ of thread blocks

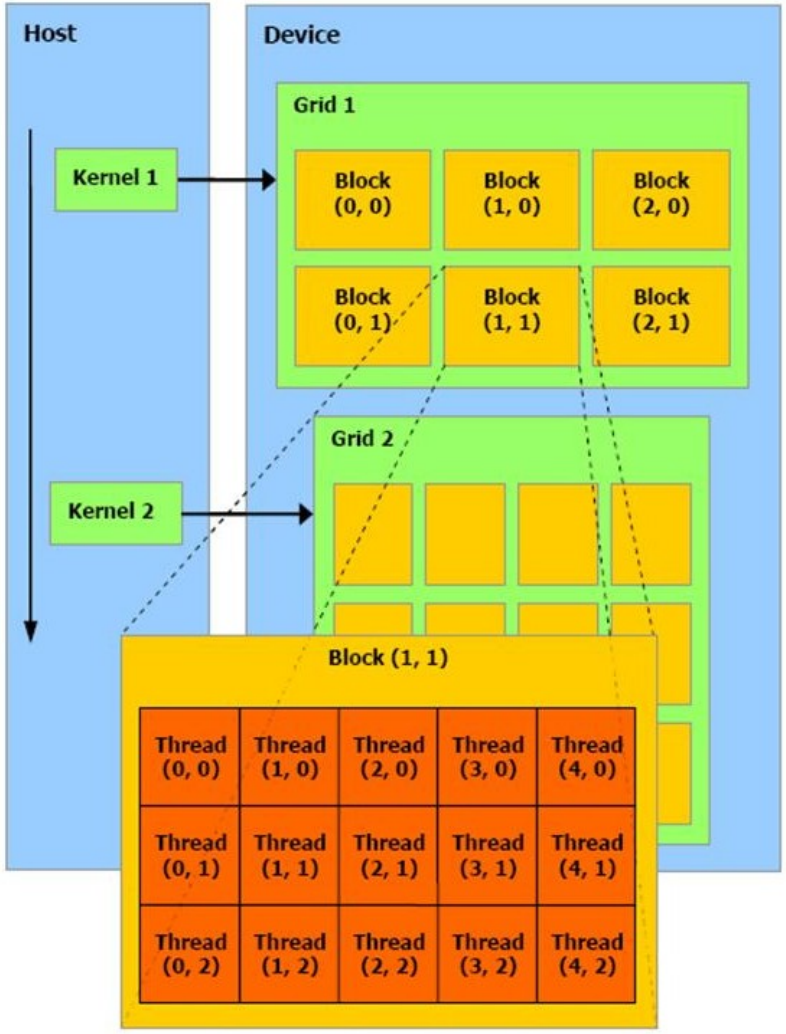
Programming Model



Programming Model

- All threads within a block can
  - Share data through ‘Shared Memory’
  - Synchronize using ‘_syncthreads()’
- Threads and Blocks have unique IDs
  - Available through special variables

Programming Model



Squaring Array Elements

Consider Array of 8 elements

Array sitting in the host(CPU) memory

```
Void   host_square(float* h_A)
{
   For(I =0; I < 8; I ++ )
         h_A[I] = h_A[I] * h_A[I]
}
```

Squaring Array Elements

- Array sitting in the Device(GPU) memory

(Also called as Global Memory)

1)Spawn the threads

2)Each thread automatically gets a number

3)Programmer controls how much work each thread will do. e.g(in our current example)

4 threads - each thread squares 2 elements

8 threads – each thread squares 1 element.

Squaring Array Elements

- Consider 8 threads are spawned by programmer, where each thread squares 1 element. Consider threads are generated within 1 block.
- Each thread automatically gets a number

  (0,0,0) (1,0,0) (2,0,0) … (7,0,0) in registers corresponding to each thread.

  These built in registers are called ThreadIdx.x ,ThreadIdx.y, Threadx.z

Squaring Array Elements

- Write a program for only 1 thread
- Following program will be executed for each thread in parallel

```
_global_ void  device_square(float* d_A)
{
        myid =  ThreadIdx.x;
       d_A[myid] = d_A[myid] * d_A[myid];
}
```

Squaring Array Elements

- Block Level Parallelism?

  Suppose programmer decides to visualize array of 8 elements as 2 blocks

  BlockId's are stored in built in BlockIdx.x, BlockIdx.y, BlockIdx.z

BlockID	(0,0,0)	(1,0,0)
ThreadID	(0,0,0) .. (3,0,0)	(0,0,0)..(3,0,0)

Squaring Array Elements

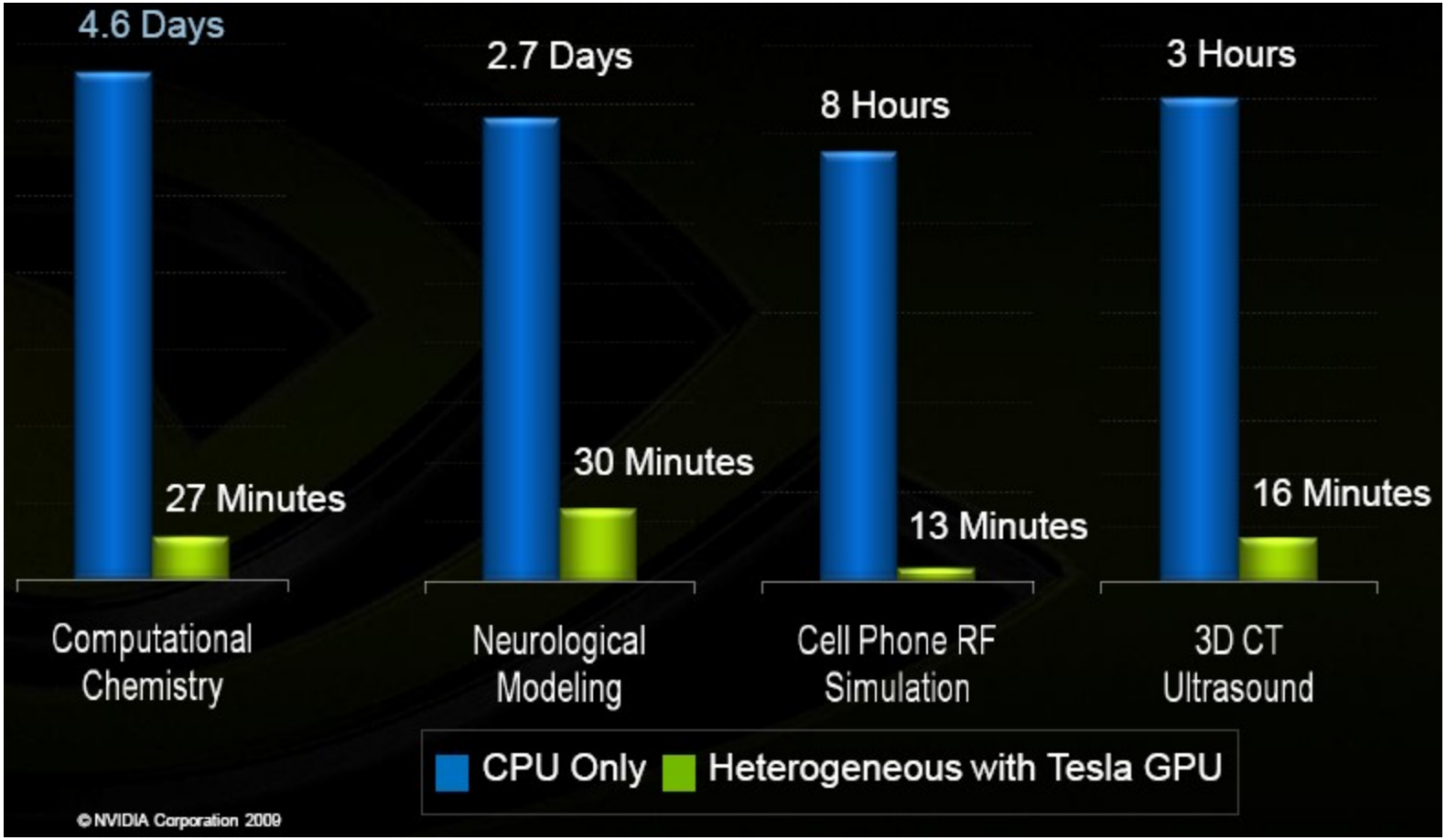
Here each thread knows its own blockID and ThreadId

```
Int BLOCK_SIZE = 4;
_global_ void  device_square(float* d_A)
{
    myid =  BlockIdx.x*BLOCK_SIZE
        +ThreadIdx.x;
    d_A[myid] = d_A[myid] * d_A[myid];
}
```

General flow of .cu file

- Allocate Array in host memory (malloc) e.g h_A
- Initialize that Array
- Allocate Array in device memory(cudaMalloc) e.g d_A
- Transfer data from host to device memory(cudaMemCpy)
- Specify kernel execution Configuaration (This is very important. Depending upon it blocks and threads automatically get assigned numbers)
- Call Kernel
- Transfer result from device to host memory(cudaMemCpy)
- Deallocate host(free) and device(cudaFree) memories

CPU v/s GPU



Compiling

- Use nvcc to compile .cu files

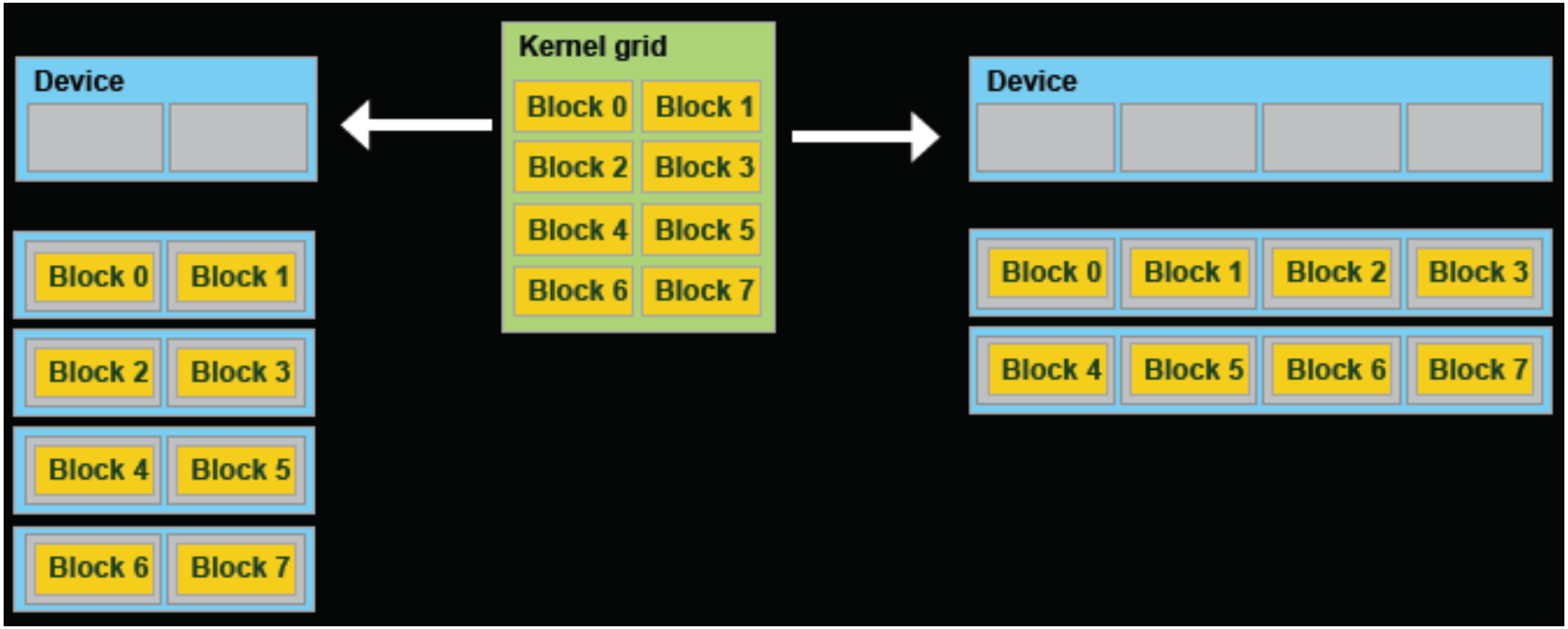
```
nvcc –o runme kernel.cu
```

- Use –c option to generate .obj files

```
nvcc –c kernel.cu
g++ –c main.cpp
g++  –o runme *.o
```

Transparent Scalability

- Hardware is free to schedule thread blocks on any processor



Thank you

- http://developer.download.nvidia.com/presentatic