Fountainhead

CUDA Thread Model

~ Threading and Scheduling ~

Andrew Sheppard

Baruch College MFE “Big Data in Finance” Course

30th January 2014

Objectives

In this talk I will cover:

1. Differences between CPU and GPU threads.
2. CUDA threads and indexes.
3. CUDA kernel launch & thread scheduling.
4. CUDA thread synchronization.

Terminology

New terminology introduced in this talk:

1. A *warp* is a group of 32 thread. (Can anyone tell me why 32 threads?)
2. A block is a 1, 2 or 3-dimensional array of threads.
3. A grid is a 1 or 2-dimensional array of blocks.
4. dim2, dim3 are (x,y) and (x,y,z) data types, respectively.

Fountainhead

~ 1. CPU versus GPU ~

GPU threads are different from CPU threads:

• CPUs typically have 16 or less threads in-flight.

• GPUs have tens of thousands of threads in-flight.

• GPU threads are “lightweight” whereas CPU threads are “heavyweight”. CPU context switching is expensive. GPU context switching is very lightweight and fast.

Fountainhead

~ 2. Threads and Indexes ~

CUDA threads form a hierarchy:

• Threads are grouped into *blocks*.

• Blocks are grouped into a *grid*.

• Each thread has a unique index within it’s block.

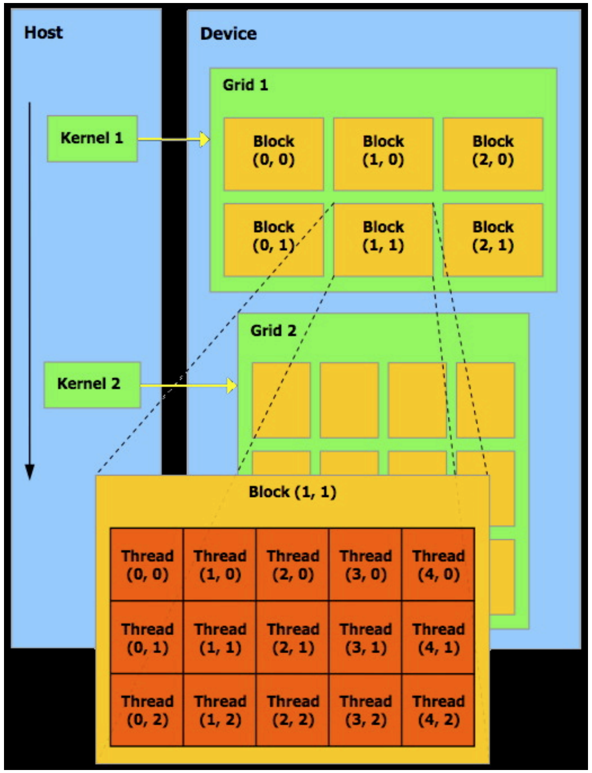
• Each block has a unique index within it’s grid.

• Note, there are many threads per block, and many blocks in a grid, but there is only *one* grid in a kernel.

Fountainhead

~ Threads and Indexes (cont.) ~

CUDA Thread Hierarchy



~ Threads and Indexes (cont.) ~

Threads in a single block:

- Run on a single multiprocessor.
- Share data held in shared memory.
- A warp will always be a subset of threads in a block.
- Threads in different blocks may be assigned to different multiprocessors concurrently, to the same multiprocessor concurrently (multithreading), or a mix at different times.

Fountainhead

~ Threads and Indexes (cont.) ~

Threads in a single block (cont.):

• Hard limit on a thread block: 512 threads (16 warps) on Tesla, 1024 threads (32 warps) on Fermi.

• Thread blocks always created in warp units, so don't bother defining a block that is not a multiple of 32 threads (number of threads in a warp).

• Thread blocks within a grid have the same size & shape.

Fountainhead

~ Threads and Indexes (cont.) ~

Threads:

• Tesla supports 32 active warps per multiprocessor. Fermi supports 48.

• Memory latency is hidden by swapping warps in and out, mixing computation with memory operations.

~ Threads and Indexes (cont.) ~

Threads running on a single multiprocessor:

• A Fermi GPU can have up to 1024 threads in a block, equivalent to 32 warps.

• However, the 1024 threads (32 warps) can be configured in a number of ways (# blocks, #warps): (2, 16), (3, 10), (4,8) … (8,4). Note: Max. of 8 blocks per multiprocessor.

• Fermi → 48 active warps → 1536 threads in-flight.

Fountainhead

~ Threads and Indexes (cont.) ~

Indexes are exposed through built-in variables:

- `dim3 threadIdx` – identifies a thread in a block.
- `dim3 blockIdx` – identifies a block in a grid.
- `dim3 blockDim` – block dimension.
- `dim3 gridDim` – grid dimension.
- `int warpSize` – warp size in threads.

Fountainhead

~ Threads and Indexes (cont.) ~

Grid dimensions:

```
1 <= gridDim.x <= 65,536

1 <= gridDim.y <= 65,536

     gridDim.z = 1
```

`gridDim` is specified at kernel launch.

Fountainhead

~ Threads and Indexes (cont.) ~

Block indexes:

```
0 <= blockIdx.x <= gridDim.x - 1

0 <= blockIdx.y <= gridDim.y - 1

     blockIdx.z = 0
```

Fountainhead

~ Threads and Indexes (cont.) ~

Thread indexes:

```
0 <= threadIdx.x <= blockDim.x - 1
0 <= threadIdx.y <= blockDim.y - 1
0 <= threadIdx.z <= blockDim.z - 1
```

`blockDim` is specified at kernel launch. Hard limit on total threads in a block: 512 for Tesla, 1024 for Fermi.

Fountainhead

~ Threads and Indexes (cont.) ~

Some recurring access patterns:

```
int x = blockIdx.x * blockDim.x +
        threadIdx.x;
int y = blockIdx.y * blockDim.y +
        threadIdx.y;
int idx = x + y * width; // 2-D data.
```

Fountainhead

~ 3. Kernel Launch ~

Kernel launch:

```
dim3 dimBlock(4, 2, 2);
dim3 dimGrid(2, 1, 1);
Kernel<<<dimGrid, dimBlock>>>( … );
```

~ Kernel Launch (cont.) ~

Scheduling unit is the warp:

- Zero overhead warp scheduling on multiprocessors.
- All threads in a warp execute the same instruction.
- A warp whose next instruction has its data ready is ready for execution.
- Warps eligible to execute are scheduled on a prioritized basis.

~ Kernel Launch (cont.) ~

Scheduling unit is the warp (cont.):

• 4 clock cycles needed to dispatch the same instructions for all threads in a block.

Fountainhead

~ 4. Thread Synchronization ~

Thread synchronization occurs at different levels:

• Threads within a block using ***shared memory***.

• Threads in different blocks and different grids must use ***global memory***.

• Implicit `__syncthreads()` (barrier) between kernels.

• Explicit `__syncthreads()` between threads in the same block. (All threads in a block must reach sync.)

~ Thread Synchronization (cont.) ~

Race conditions:

Race conditions arise when 2+ threads attempt to access the same memory location concurrently and at least one access is a write.

A word of advice:

It’s more efficient to design programs that require as little synchronization between threads as possible.

Fountainhead

~ Thread Synchronization (cont.) ~

Atomics:

- Atomic memory operations enforce atomic access to shared variables that can be accessed by multiple threads.
- You can synthesize various coordination objects and synchronization methods using atomics.