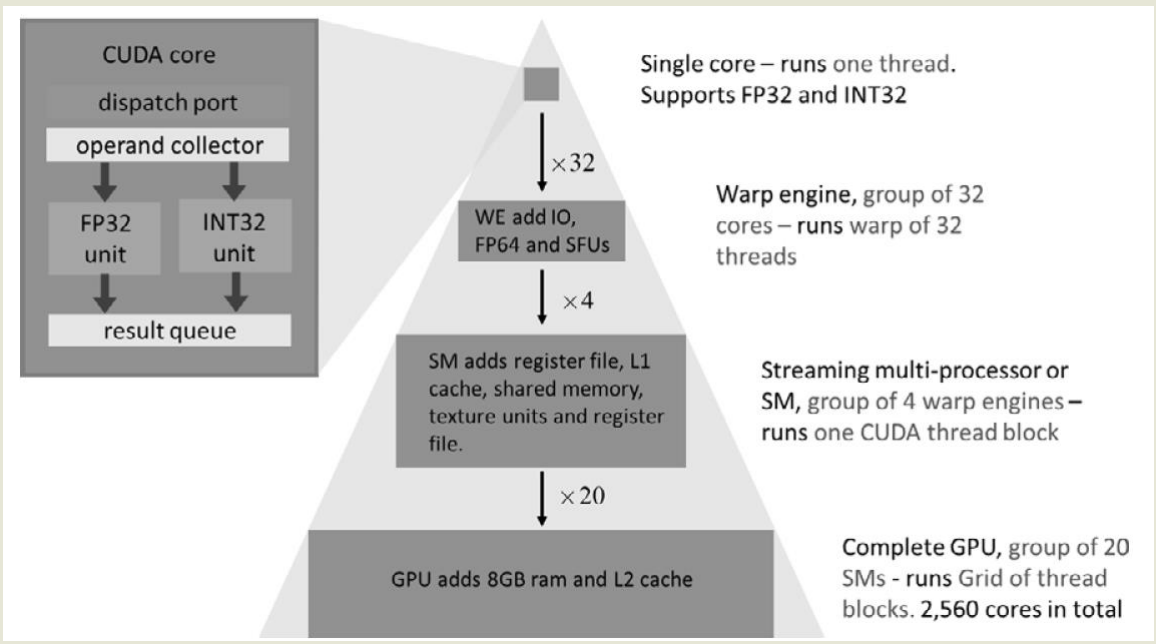
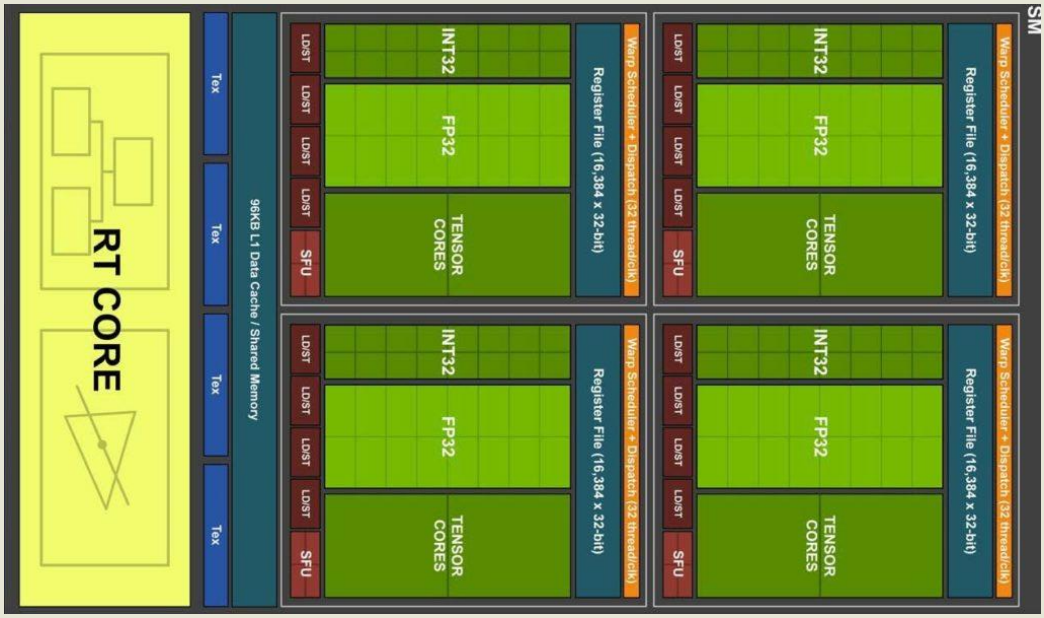
# CUDA Intro

1




GTX 1080 Architecture

2




3

The GPU architecture is reflected in the way a CUDA kernel is designed and launched by host software.

Designing good kernels to match particular problems requires skill and experience.

4

- In CUDA the thread block is a key concept;
- it is a group of threads that are batched together and run on the same SM.

5

- The size of the thread block should be a multiple of the warp size (currently 32 for all NVIDIA GPUs) up to the hardware maximum size of 1024.
- In kernel code, threads within the same thread block can communicate with each other using shared or global device memory and can synchronize with each other where necessary.
- Threads in different thread blocks cannot communicate during kernel execution and the system cannot synchronize threads in different thread blocks.

6

# Understanding Launch parameters

7

The configuration parameters are given between the

“<<<” and “>>> ”

before the traditional C function arguments.

8

These parameters sets the it sets the **grid** and **thread** block dimensions

9

Configuration Parameters

- The first configuration parameter gives the number of blocks in the grid.
- The second specifies the number of threads in each block.

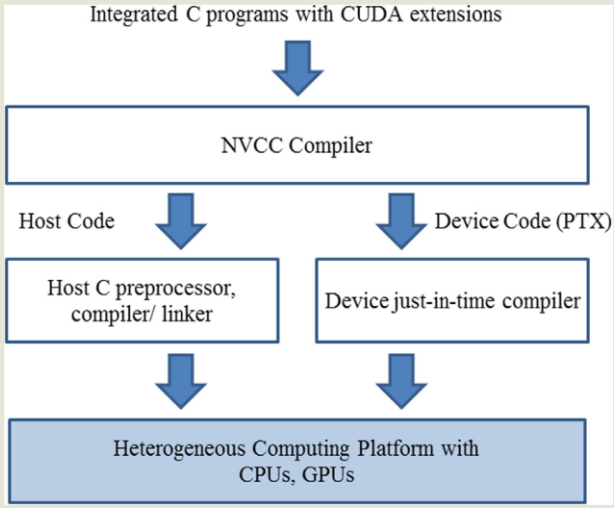
10

## Thread block execution

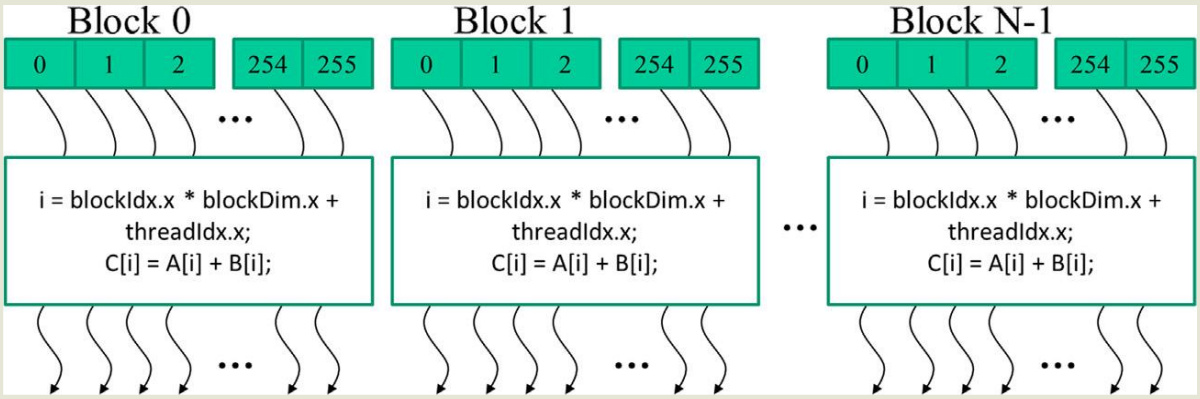
- Note that all the thread blocks operate on different parts of the vectors.
- They can be executed in any arbitrary order. The programmer must not make any assumptions regarding execution order.

11

## Cuda Compilation




12




13

- A struct with three unsigned integer fields (x, y, and z)
- Organizes the threads into a 1D, 2D, or 3D array.

blockDim

- Gives each thread a unique coordinate within a block

threadIdx

- Gives all threads in a block a common block coordinate.

blockIdx

14

a unique global index i is calculated as

```
i = blockIdx.x * blockDim.x + threadIdx.x
```

15

Value of i can exceed the size of Array.

What can we do?

16

```
1.  // Compute vector sum C = A + B
2.  // Each thread performs one pair-wise addition
3.  __global__
4.  void vecAddKernel(float* A, float* B, float* C, int n) {
5.    int i = threadIdx.x + blockDim.x * blockIdx.x;
6.    if (i < n) {
7.     C[i] = A[i] + B[i];
8.    }
9.  }
```

17

## Why *if(i<n)* statement

- This is because not all vector lengths can be expressed as multiples of the block size.
- For example, let's assume that the vector length is 100.
- The smallest efficient thread block dimension is 32.
- Assume that we picked 32 as block size. One would need to launch four thread blocks to process all the 100 vector elements. However, the four thread blocks would have 128 threads.
- We need to disable the last 28 threads in thread block 3 from doing work not expected by the original program.
- Since all threads are to execute the same code, all will test their i values against n, which is 100.
- With the ***if (i<n)*** statement, the first 100 threads will perform the addition, whereas the last 28 will not.
- This allows the kernel to be called to process vectors of arbitrary lengths.

18

# Cuda Qualifiers

19

| Qualifier Keyword | Callable From | Executed On | Executed By |
|---|---|---|---|
| __host__ (default) | Host | Host | Caller host thread |
| __global__ | Host (or Device) | Device | New grid of device threads |
| __device__ | Device | Device | Caller device thread |

20

**9.** Consider the following CUDA kernel and the corresponding host function that calls it:

```
01    __global__ void foo_kernel(float* a, float* b, unsigned int
N){
02        unsigned int i=blockIdx.x*blockDim.x + threadIdx.
x;
03        if(i < N) {
04            b[i]=2.7f*a[i] - 4.3f;
05        }
06    }
07    void foo(float* a_d, float* b_d) {
08        unsigned int N=200000;
09        foo_kernel <<< (N + 128–1)/128, 128 >>> (a_d,
b_d, N);
10    }
```

**a.** What is the number of threads per block?
**b.** What is the number of threads in the grid?
**c.** What is the number of blocks in the grid?
**d.** What is the number of threads that execute the code on line 02?
**e.** What is the number of threads that execute the code on line 04?

21

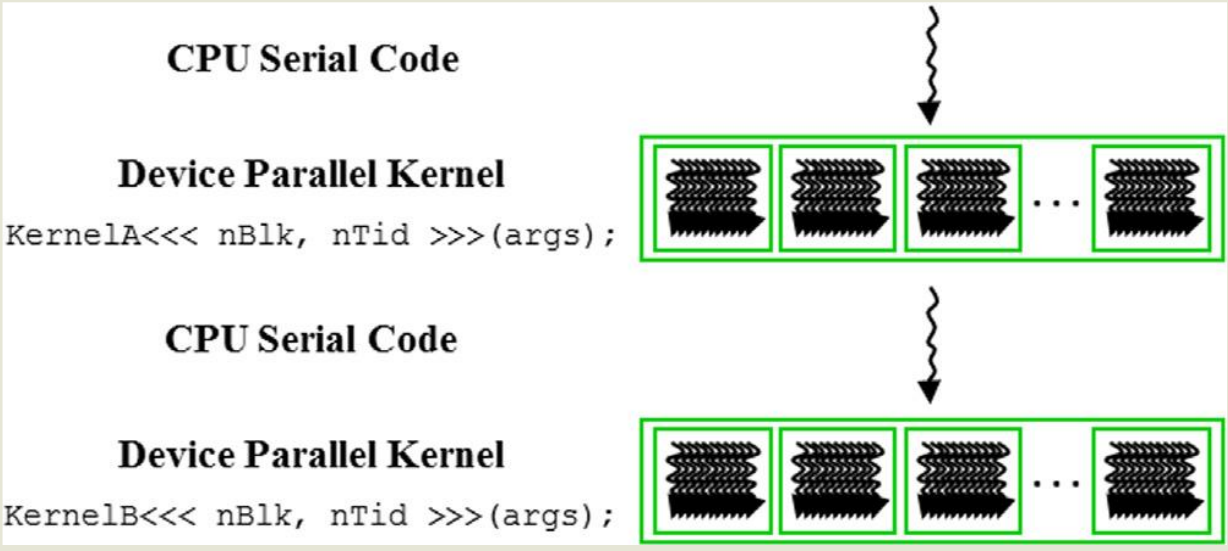
# Multi-dimensional Grid Organization

22

What is a grid

- When a kernel function is called, a large number of threads are launched on a device to execute the kernel.
- All the threads that are launched by a kernel call are collectively called a grid.
- These threads are the primary vehicle of parallel execution in a CUDA platform.

23




24

When all threads of a grid have completed their execution, the grid terminates, and the execution continues on the host until another grid is launched.

25

Grids

- A grid of threads is organized into a two-level hierarchy.
  - Array of thread blocks (blocks)
  - Blocks contains array of threads
- All blocks of a grid are of the same size; each block can contain up to 1024 threads on current systems

26

- all threads in a grid execute the same kernel function,
- and they rely on coordinates, (thread indices),
  - to distinguish themselves from each other and to
  - identify the appropriate portion of the data to process.

27

- A grid consists of one or more blocks, and each block consists of one or more threads.
- All threads in a block share the same block index

28

The execution configuration parameters in a kernel call statement specify the ***dimensions of the grid*** and the ***dimensions of each block***

29

The first execution configuration parameter specifies the ***dimensions of the grid*** in number of blocks.

The second specifies the ***dimensions of each block*** in number of threads.

30

Dim3

- Is an integer vector type of three elements x, y, and z.
- These three elements specify the sizes of the three dimensions.
- The programmer can use fewer than three dimensions by setting the size of the unused dimensions to 1.

31

gridDim

- In CUDA C the allowed values of gridDim.x range from 1 to $2^{31}$ - 1, and
- those of gridDim.y and gridDim.z range from 1 to $2^{16}$ - 1 (65,535).
- All threads in a block share the same blockIdx.x, blockIdx.y, and blockIdx.z values.
- Among blocks,
  - The blockIdx.x value ranges from 0 to gridDim.x-1,
  - The blockIdx.y value ranges from 0 to gridDim.y-1, and
  - The blockIdx.z value ranges from 0 to gridDim.z-1.

32

# Block Configuration

33

All blocks in a grid have the same dimensions and sizes.

34

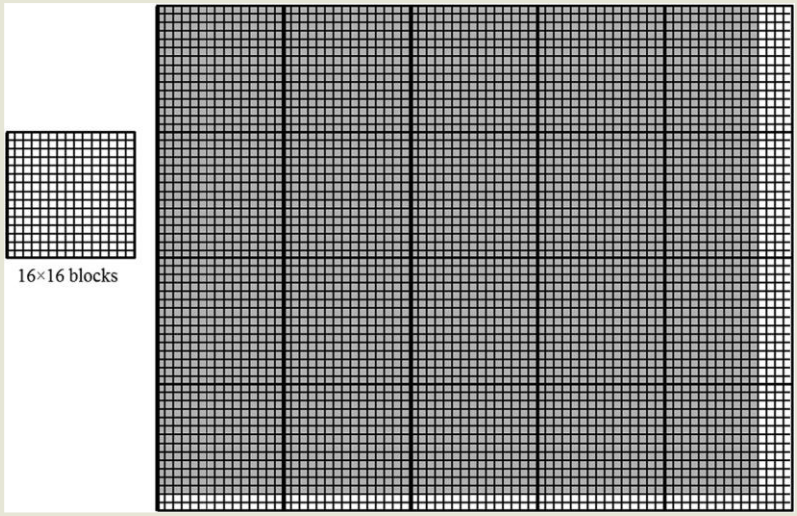
The total size of a block in current CUDA systems is limited to 1024 threads.

These threads can be distributed across the three dimensions in any way as long as the total number of threads does not exceed 1024.

35

62x76 picture

36




37