Parallel Computing with GPUs

CUDA Memory
Part 1 – Memory Overview

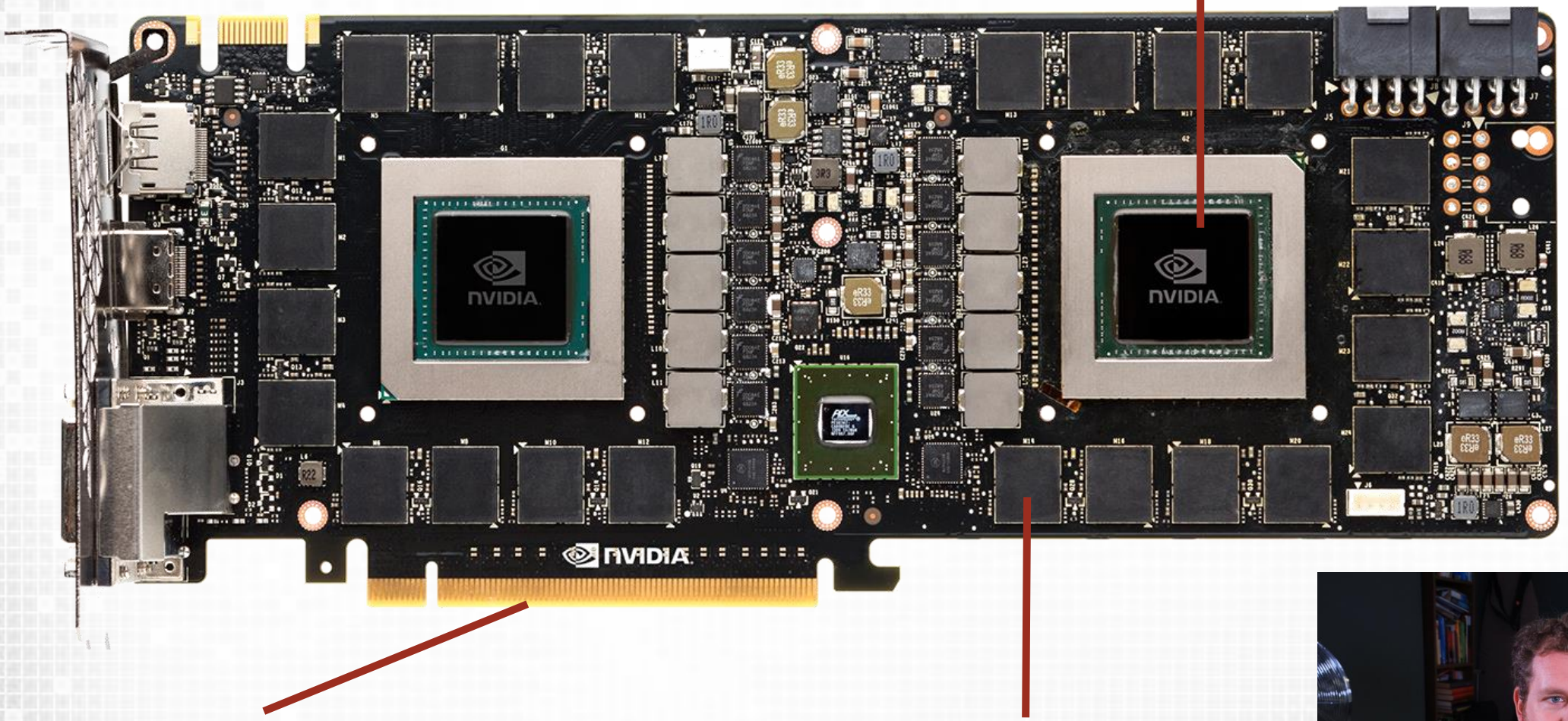
The University Of Sheffield.

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

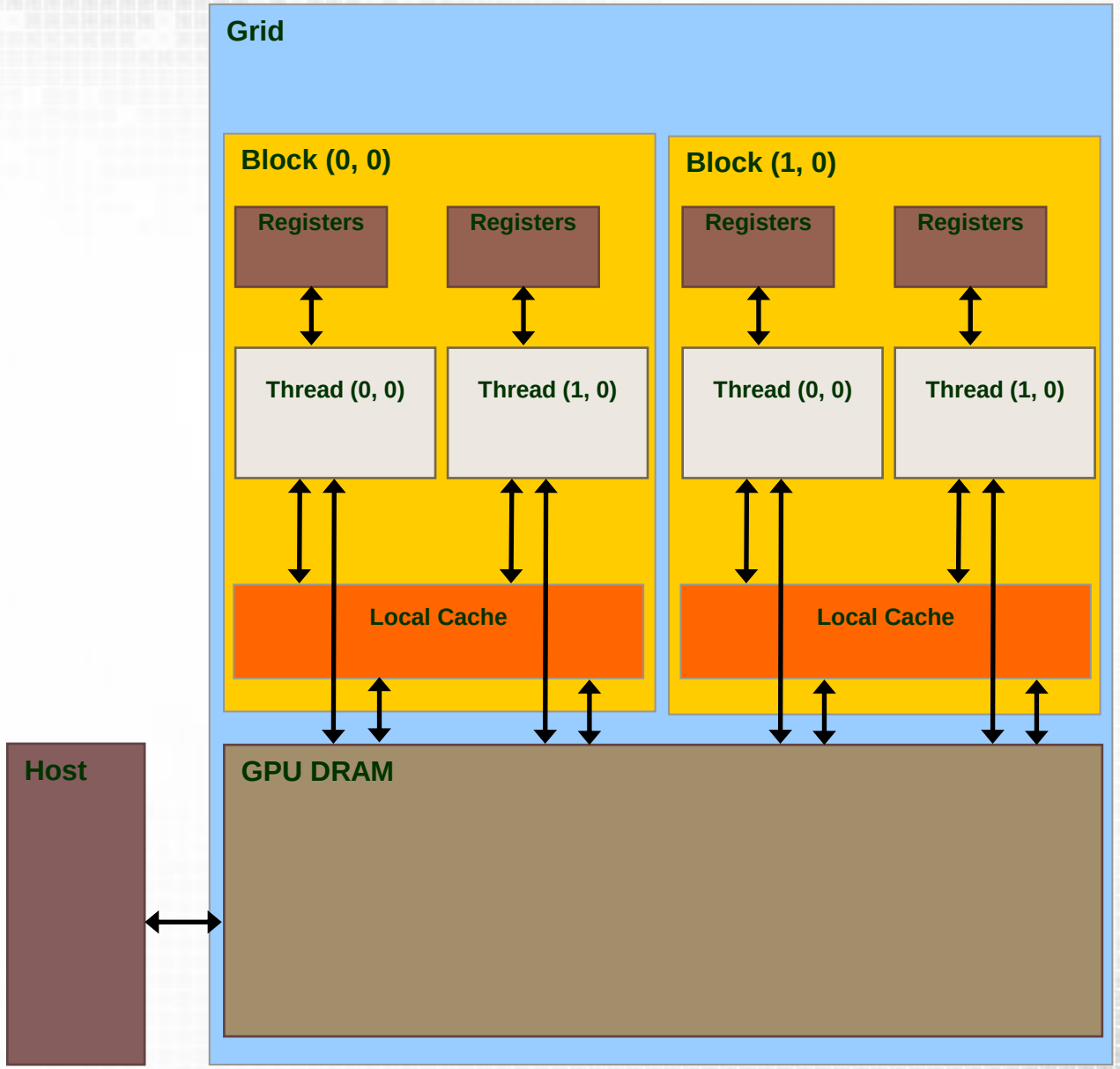
- CUDA Memory Overview
  - Present the GPUs memory hierarchy and how this differs between hardware versions
  - Identify where latencies exist memory operations
  - Give an example of how to benchmark a CUDA program

GPU Memory (GTX Titan Z)



Simple Memory View

- Threads have access to;
  - **Registers**
    - Read/Write **per thread**
  - **Local memory**
    - Read/Write **per thread**
  - **Local Cache**
    - Read/Write **per block**
  - **Main DRAM Memory**
    - Read/Write **per grid**



Local Memory

- **Local memory (Thread-Local Global Memory)**
  - Read/Write per thread
  - Local memory does not physically exist
    - Mapped to reserved area in global memory
    - Usually uses an are of local cache (e.g. L1)
  - Used for variables if you exceed the number of registers available
    - Very bad for performance!
  - Arrays always go in local memory if they are indexed with non constants

```
__global__ void localMemoryExample
(int * input)
{

    int a;
    int b;
    int index;

    int myArray1[4];
    int myArray2[4];
    int myArray3[100];

    index = input[threadIdx.x];
    a = myArray1[0];
    b = myArray2[index];

}
```

non constant index

https://stackoverflow.com/questions/10297067/in-a-cuda-kernel-how-do-i-store-an-array-in-local-thread-memory

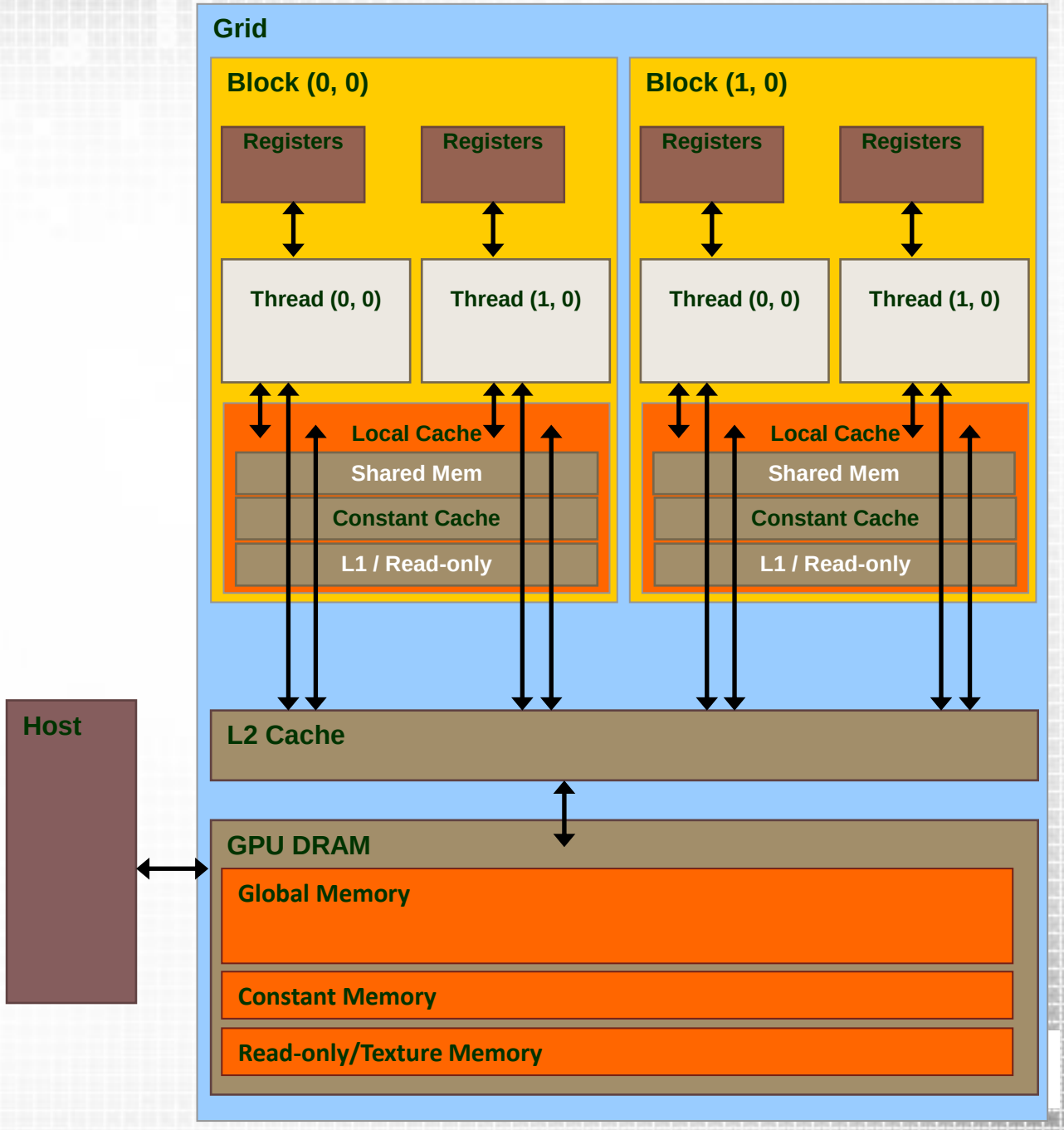
Memory Latencies

- What is the cost of accessing each area of memory?
  - On chip caches are MUCH lower latency

	Cost (cycles)
Register	1
Global	200-800
Shared memory	~1
L1	1
Constant	~1 (if cached)
Read-only (tex)	1 if cached (same as global if not)

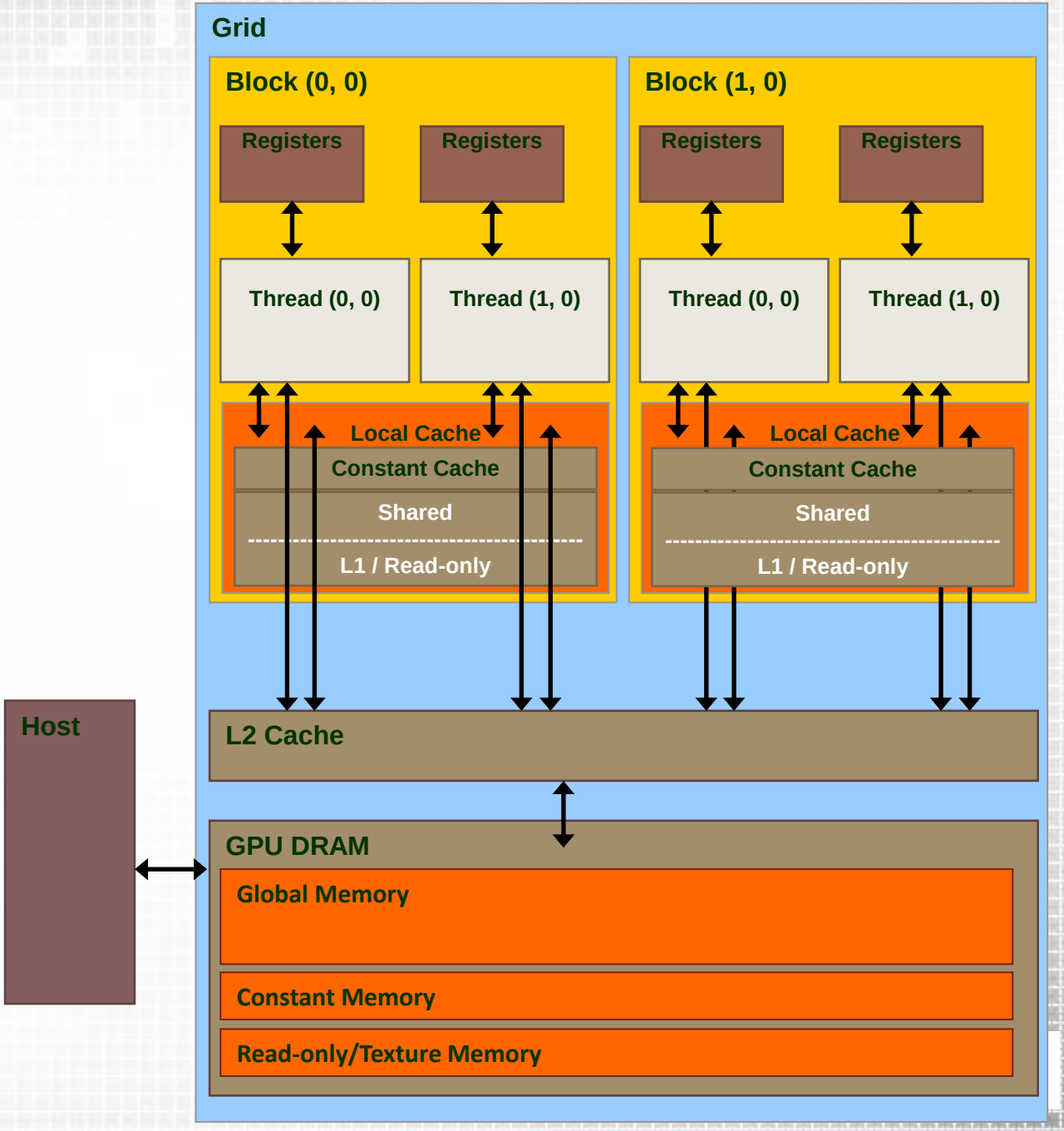
Maxwell/Pascal

- Each Thread has access to
  - **Registers**
  - **Local memory**
  - Main DRAM Memory via L2 cache
    - Global Memory
      - Can be read via Unified Data Cache
    - Constant Memory
      - Via **L2 cache** and per **block Constant cache**
    - Unified L1/Texture Cache
      - Same cache used for read only or texture reads
    - Dedicated Shared Memory
      - User configurable cache



Volta

- Each Thread has access to
  - Registers
  - Local memory
  - Main DRAM Memory via L2 cache
    - Global Memory
      - Can be read via Unified Data Cache
    - Constant Memory
      - Via L2 cache and per block Constant cache
    - **Unified Shared/L1/Texture Cache**
      - Same cache used for read only or texture reads
      - Amount of shared memory configurable at runtime



Cache and Memory Sizes

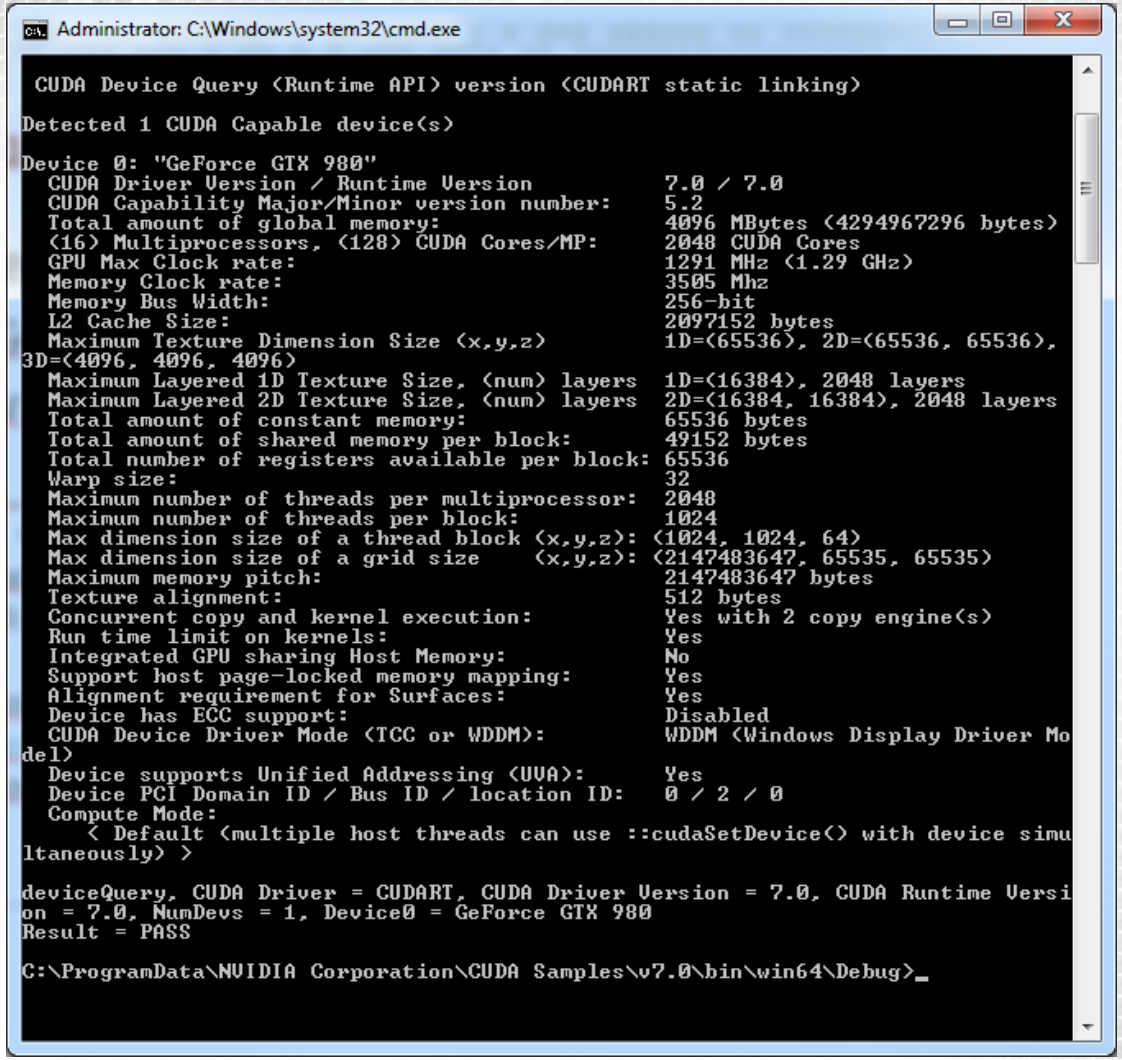
	Pascal (P100) GP100	Volta (V100) GV100
Register File Size	*256KB per SM*	*256KB per SM*
Shared Memory	*64KB Dedicated*	*Configurable up to 96KB*
Constant Memory	*64KB DRAM 8KB Cache per SM*	*64KB DRAM 8KB Cache per SM*
L1/Read Only Memory	*24KB per SM Dedciated*	*Configurable up to 128KB per SM*
L2 Cache Size	*4096KB*	*6144KB*
Device Memory	*16GB*	*16GB*
DRAM Interface	*4096-bit HBM2*	*4096-bit HBM2*

https://devblogs.nvidia.com/inside-volta/

Device Query

- What are the specifics of my GPU?
  - Use `cudaGetDeviceProperties`
  - E.g.
    - `deviceProp.sharedMemPerBlock`
  - CUDA SDK deviceQry example

```
int deviceCount = 0;
cudaGetDeviceCount(&deviceCount);
for (int dev = 0; dev < deviceCount; ++dev)
{
    cudaSetDevice(dev);
    cudaDeviceProp deviceProp;
    cudaGetDeviceProperties(&deviceProp, dev);
    …
}
```



Performance Measurements

- How can we benchmark our CUDA code?
- Kernel Calls are asynchronous
  - If we use a standard CPU timer it will measure only launch time not execution time
  - We could call `cudaDeviceSynchronise()` but this will stall the entire GPU pipeline
- Alternative: CUDA Events
  - Events are created with `cudaEventCreate()`
  - Timestamps can be set using `cudaEventRecord()`
  - `cudaEventElapsedTime()` sets the time in *ms* between the two events.

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start);
my_kernel <<<(N /TPB), TPB >>>();
cudaEventRecord(stop);

cudaEventSynchronize(stop);
float milliseconds = 0;
cudaEventElapsedTime(&milliseconds,
                     start, stop);

cudaEventDestroy(start);
cudaEventDestroy(stop);
```

CUDA qualifiers summary

- Where can a variable be accessed?
  - Is declared inside the kernel?
    - Then the host can not access it
    - Lifespan ends after kernel execution
  - Is declared outside the kernel
    - Then the host can access it (via `cudaMemcpyToSymbol`)

```
__device__ int my_global;


__global__ void my_kernel() {
  int my_local;

  int *ptr1 = &my_global;
  int *ptr2 = &my_local;
}
```

- What about pointers?
  - They can point to anything
  - BUT are not typed on memory space
  - Be careful not to confuse the compiler

```
if (something)
  ptr1 = &my_global;
else
  ptr1 = &my_local;
```

Summary

- CUDA Memory Overview
  - Present the GPUs memory hierarchy and how this differs between hardware versions
  - Identify where latencies exist memory operations
  - Give an example of how to benchmark a CUDA program

- Next Lecture: Global and Constant Memory