

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)

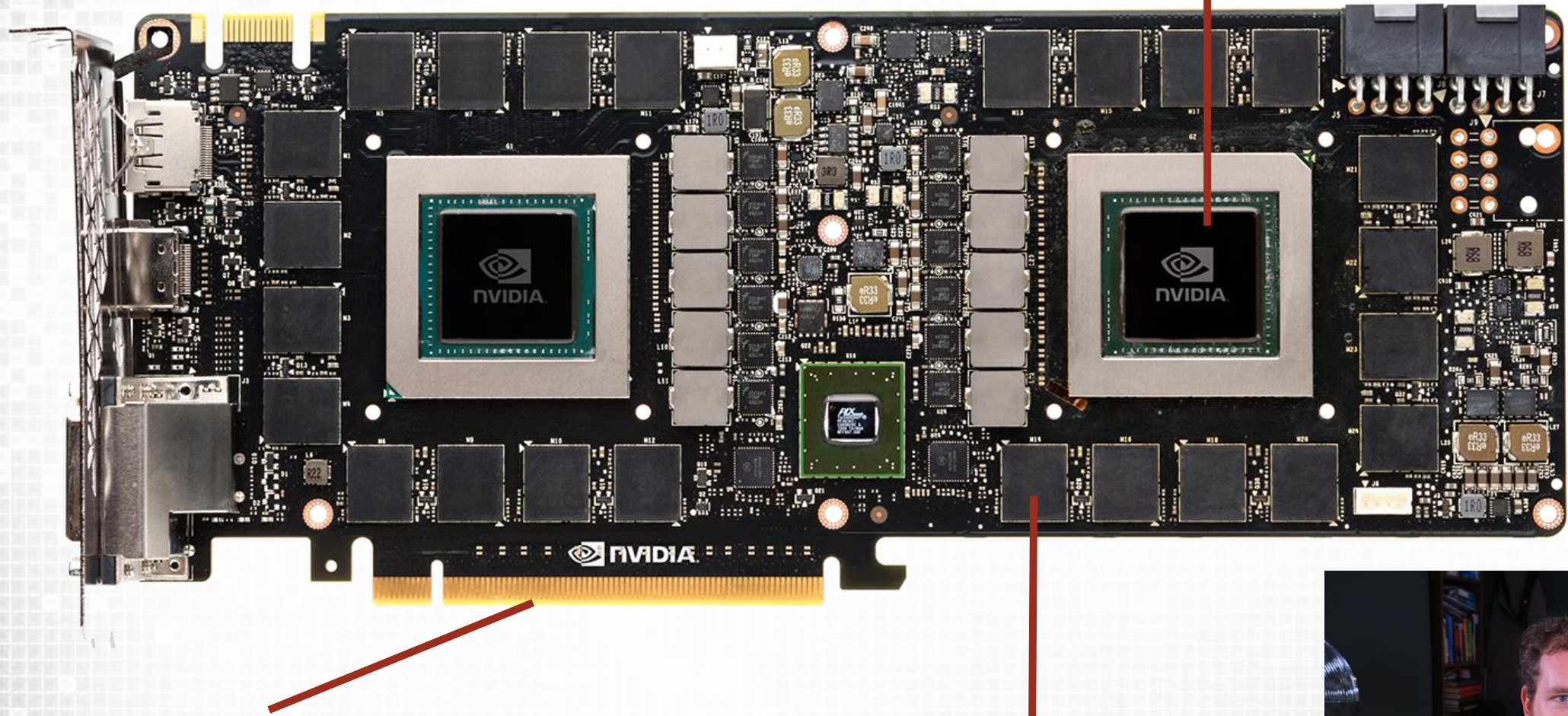
❑ CUDA Memory Overview

- ❑ Present the GPU's 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)

Shared Memory, cache and registers



Host Memory (via PCIe)

GPU DRAM Memory



Simple Memory View

❑ Threads have access to;

❑ **Registers**

❑ Read/Write **per thread**

❑ **Local memory**

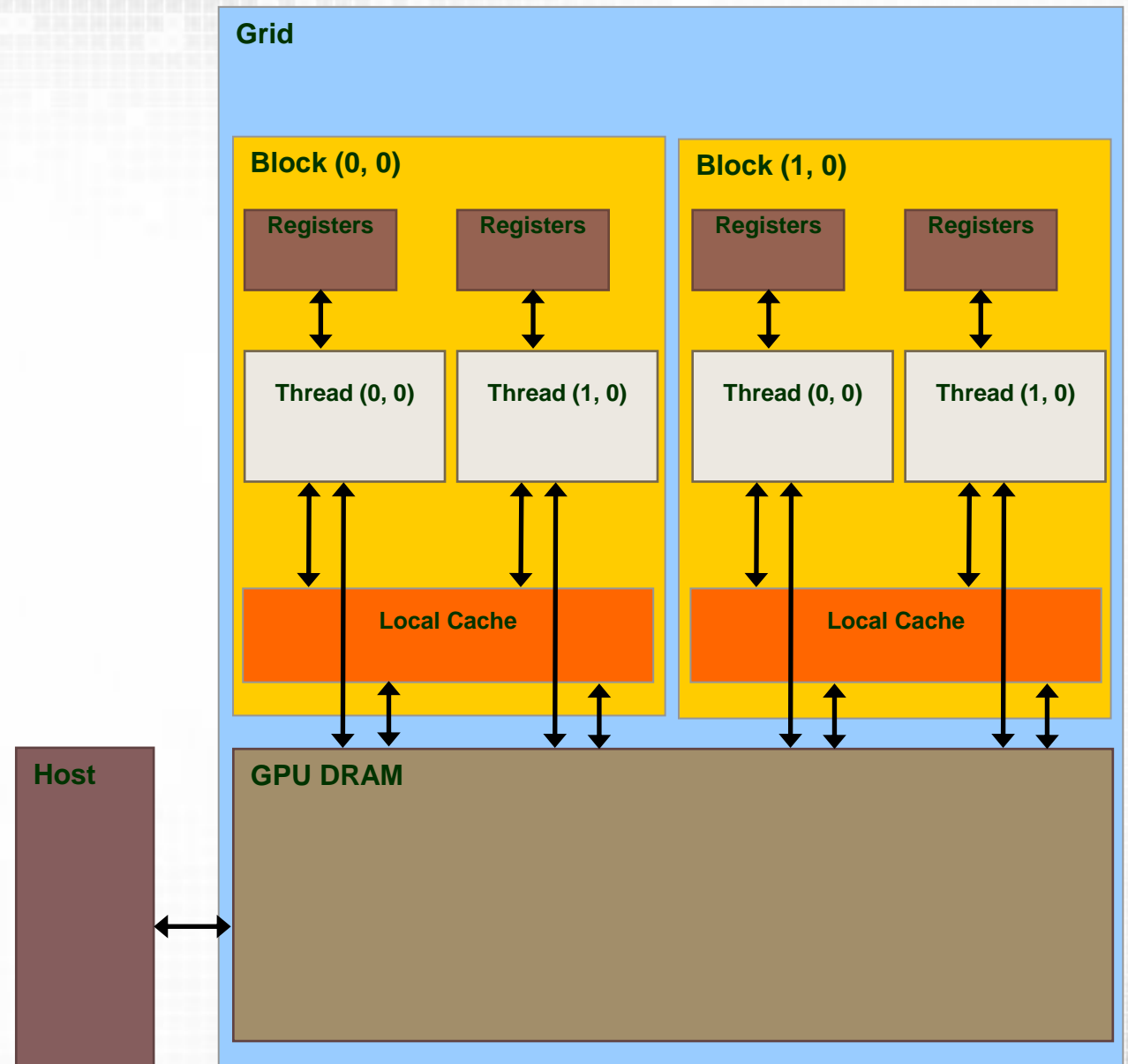
❑ Read/Write **per thread**

❑ **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



Memory Latencies

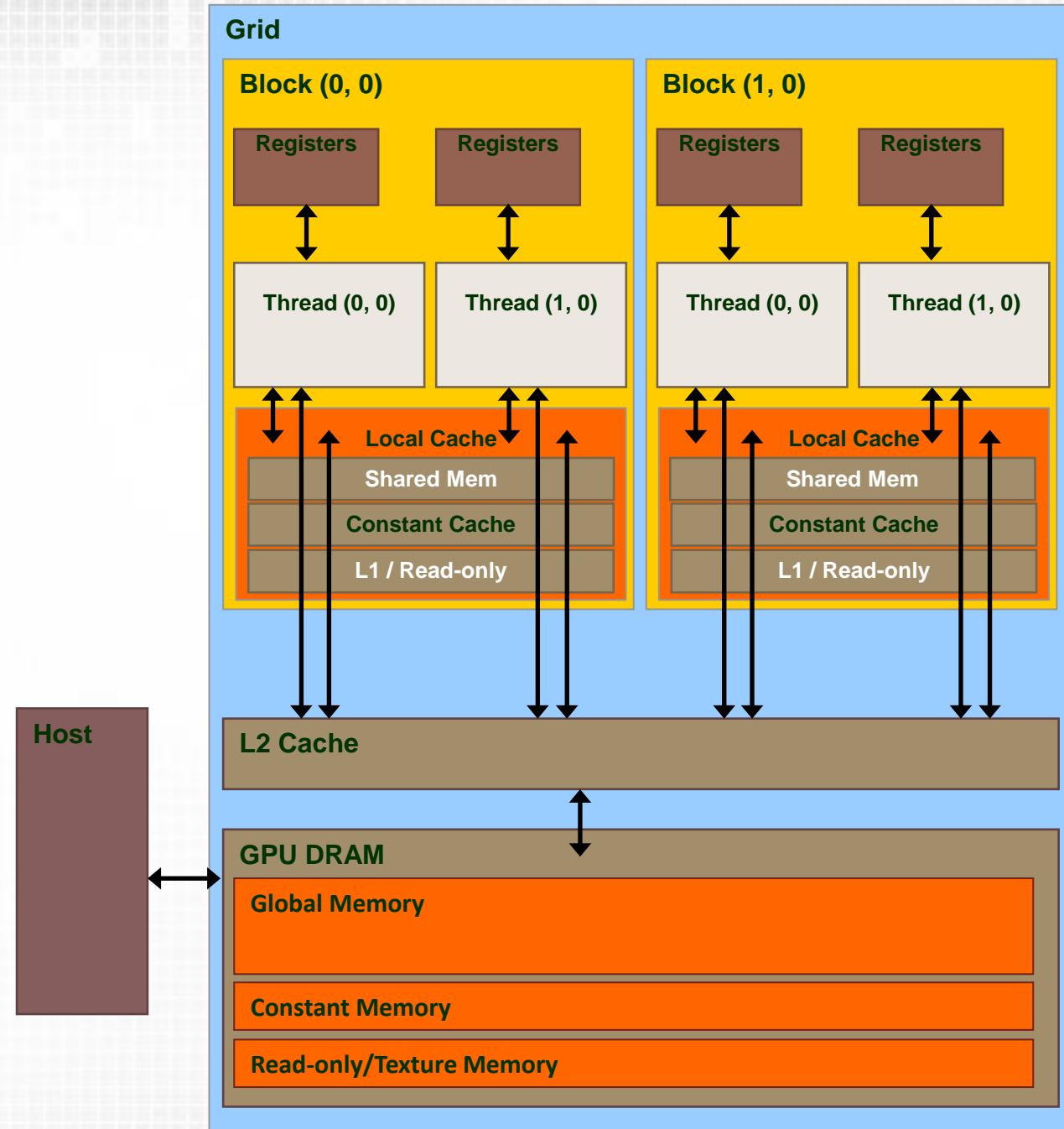
- ❑ What is the cost of accessing each area of memory?
 - ❑ On chip caches are MUCH lower latency

	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)



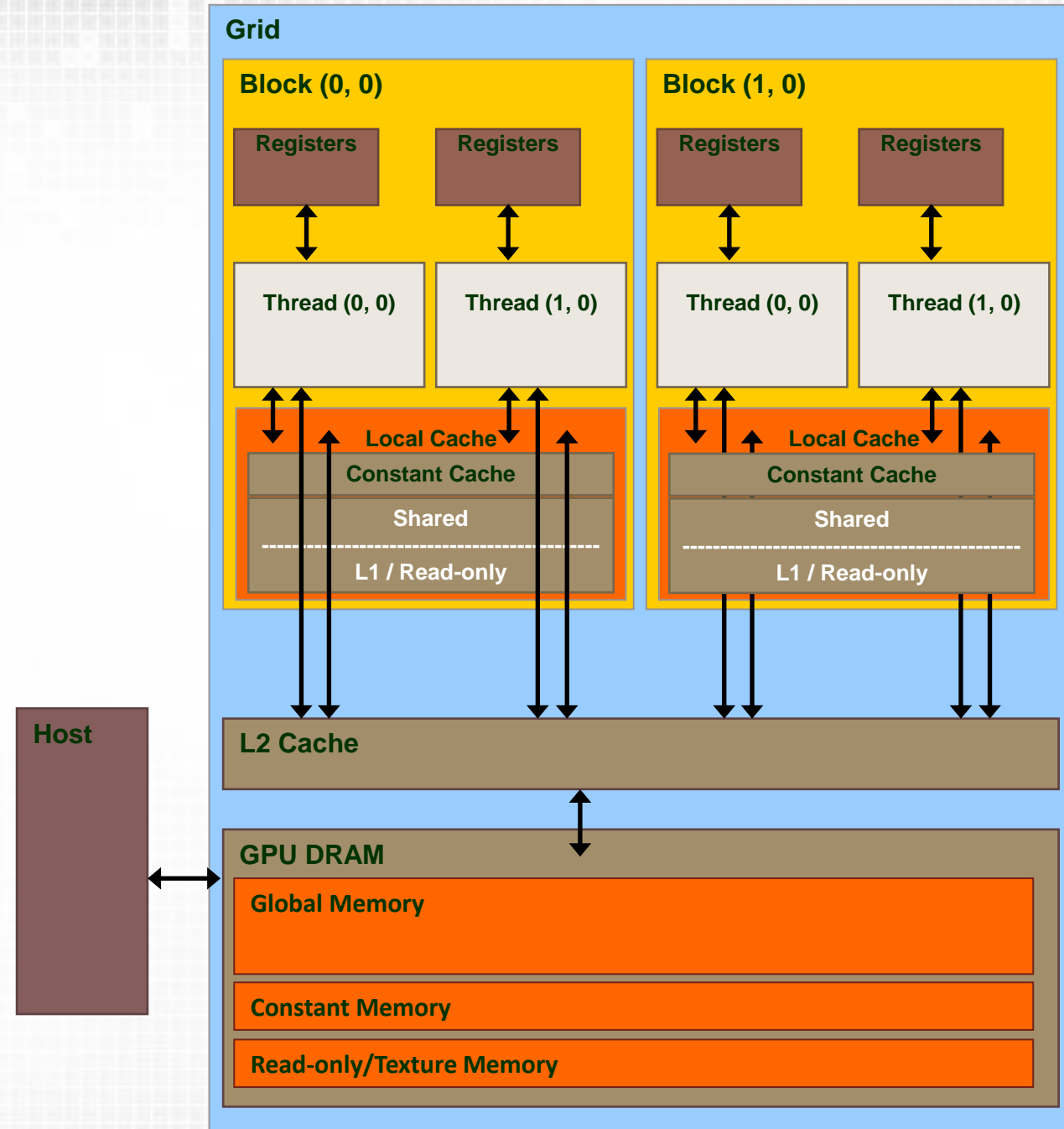
Maxwell/Pascal

- ❑ 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

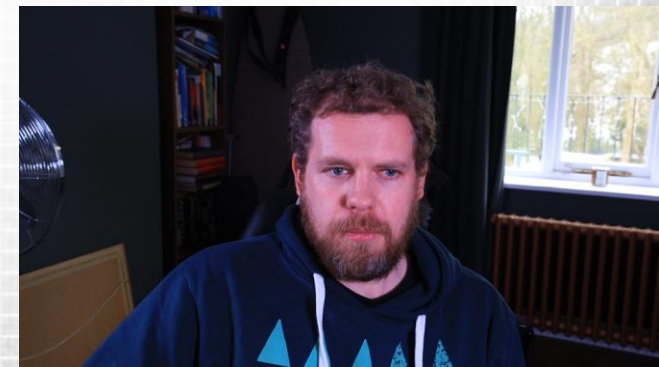
- ❑ Each Thread has access to
 - ❑ Registers
 - ❑ Local memory
 - ❑ Main DRAM Memory via L2 cache
 - ❑ Global Memory
 - ❑ Can be read via Unified Data Cache
 - ❑ Constant Memory
 - ❑ Via L2 cache and per block Constant cache
- ❑ **Unified Shared/L1/Texture Cache**
 - ❑ Same cache used for read only or 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);
    ...
}
```

```
Administrator: C:\Windows\system32\cmd.exe

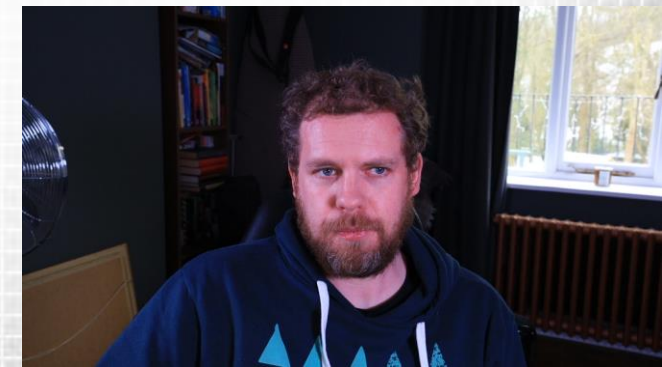
CUDA Device Query (Runtime API) version (CUDA static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 980"
  CUDA Driver Version / Runtime Version          7.0 / 7.0
  CUDA Capability Major/Minor version number:    5.2
  Total amount of global memory:                 4096 MBytes (4294967296 bytes)
  (16) Multiprocessors, (128) CUDA Cores/MP:     2048 CUDA Cores
  GPU Max Clock rate:                            1291 MHz (1.29 GHz)
  Memory Clock rate:                             3505 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536),
  3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z):  (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z):     (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                      Yes
  Integrated GPU sharing Host Memory:             No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  CUDA Device Driver Mode (TCC or WDDM):         WDDM (Windows Display Driver Mo
del)
  Device supports Unified Addressing (UVA):       Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 2 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simu
ltaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.0, CUDA Runtime Versi
on = 7.0, NumDevs = 1, Device0 = GeForce GTX 980
Result = PASS

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v7.0\bin\win64\Debug>
```



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 GPU's 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

