Parallel Computing with GPUs

CUDA Memory
Part 2 - Global and Constant Memory

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

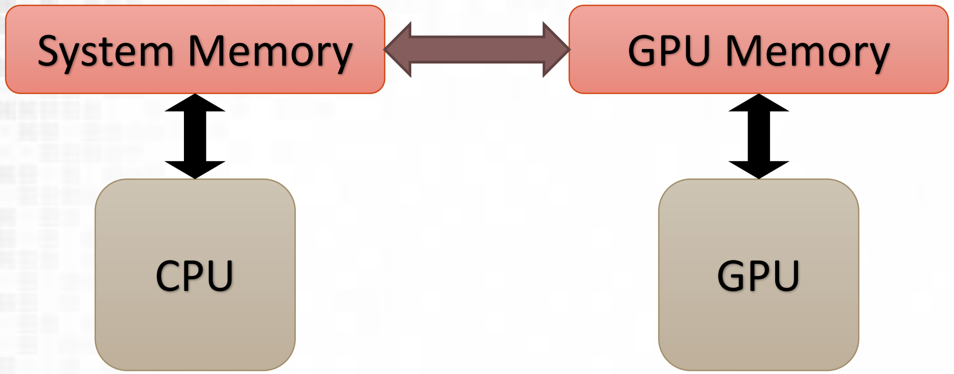
This Lecture (learning objectives)

- ❑Global and Constant Memory
  - ❑Compare and contrast manual memory movement with Unified Memory
  - ❑Identify the use cases for constant memory
  - ❑Demonstrate an appropriate use of constant memory

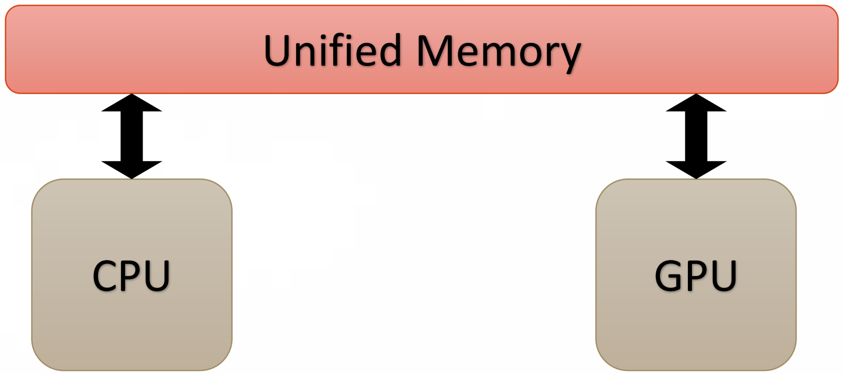
Dynamic vs Static Global Memory

- In the previous lab we dynamically defined GPU memory
  - Using `cudaMalloc()`
- You can also statically define (and allocate) GPU global memory
  - Using `__device__` qualifier
  - Requires memory copies are performed using `cudaMemcpyToSymbol` or `cudaMemcpyFromSymbol`
  - See example from last weeks lecture
- This is the equivalent of the following in C (host code)
  - `int my_static_array[1024];`
  - `int *my_dynamic_array = (int*) malloc(1024*sizeof(int));`

Unified Memory



CUDA 6.0+
Kepler+



- ❑So far the developer view is that GPU and CPU have separate memory
  - ❑Memory must be explicitly copied
  - ❑Deep copies required for complex data structures
- ❑Unified Memory changes that view

Unified Memory Example

C Code

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

CUDA (6.0+) Code

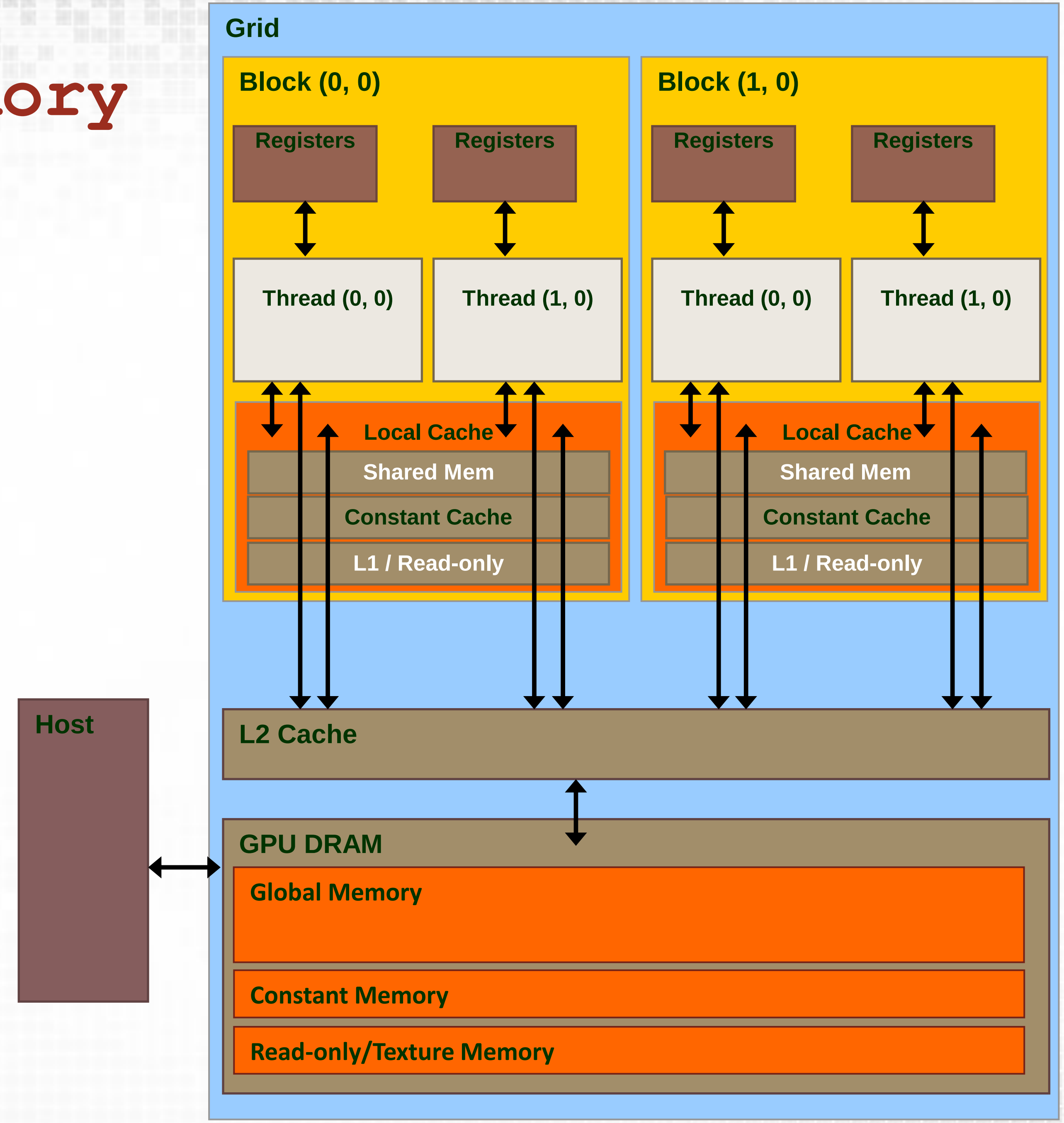
```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

  gpu_qsort(data, N, 1, compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);
}
```

Implications of CUDA Unified Managed Memory

- Simpler porting of code
- Memory is only *virtually* unified
  - GPU still has discrete memory
  - It still has to be transferred via PCIe (or NVLINK)
- Easier management of data to and from the device
  - Explicit memory movement is not required
  - Similar to the way the OS handles virtual memory
- Issues
  - Requires look ahead and paging to ensure memory is in the correct place (and synchronised)
  - It is not as fast as hand tuned code which has finer explicit control over transfers
- *We will manage memory movement ourselves!*

The University Of Sheffield.

Constant Memory



Constant Memory

- Constant Memory
  - Stored in the devices global memory
  - Read through the per SM constant cache
  - Set at runtime
  - When using correctly only 1/16 of the traffic compared to global loads
- When to use it?
  - When small amounts of data are **read only**
  - When values are **broadcast** to threads in a half warp (of 16 threads)
  - Very fast when cache hit
  - Very slow when no cache hit
- How to use
  - Must be **statically** (compile-time) defined as a symbol using `__co` qualifier
  - Value(s) must be copied using **cudaMemcpytoSymbol**.

Constant Memory Broadcast

- …. When values are **broadcast** to threads in a half warp (groups of 16 threads)

```
__constant__ int my_const[16];

__global__ void vectorAdd() {
int i = blockIdx.x;

int value = my_const[i % 16];
}
```

```
__constant__ int my_const[16];

__global__ void vectorAdd() {
int i = blockIdx.x * blockDim.x + threadIdx.x;

int value = my_const[i % 16];
}
```

Which is good use of constant memory?

Constant Memory Broadcast

```
__constant__ int my_const[16];

__global__ void constant_test() {
int i = blockIdx.x;

int value = my_const[i % 16];
}
```

```
__constant__ int my_const[16];

__global__ void constant_test() {
int i = blockIdx.x * blockDim.x + threadIdx.x;

int value = my_const[i % 16];
}
```

Which is good use of constant memory?

- ❑Best possible use of constant memory
- ❑Every thread in half warp reads the same
  - ❑ Index based on `blockIdx`
- ❑No serialisation
  - ❑ 1 read request for every thread!
- ❑Other threads in the block will also hit cache

- ❑Worst possible use of constant memory
- ❑Every thread in half warp reads different value
  - ❑ Index based on `threadIdx`
- ❑Each access will be serialised
  - ❑ 16 different read requests!
- ❑Other threads in the block will likely miss the cache

Constant Memory

- Question: Should I convert `#define` to constants?
  - E.g. `#define MY_CONST 1234`
- Answer: No
  - Leave alone
  - `#defines` are embed in the code by pre-processors
  - They don't take up registers as they are embed within the instruction space
    - i.e. are replaced with literals by the pre-processor
- Only replace constants that may change at runtime (but not during the GPU programs)

Summary

- Global and Constant Memory
  - Compare and contrast manual memory movement with Unified Memory
  - Identify the use cases for constant memory
  - Demonstrate an appropriate use of constant memory

- Next Lecture: Read Only and Texture Memory