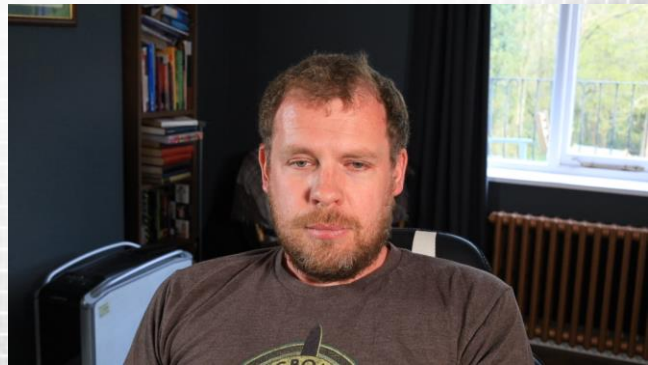
Parallel Computing with GPUs

CUDA Streams
Part 4 – Multi GPU Programming

The University Of Sheffield.

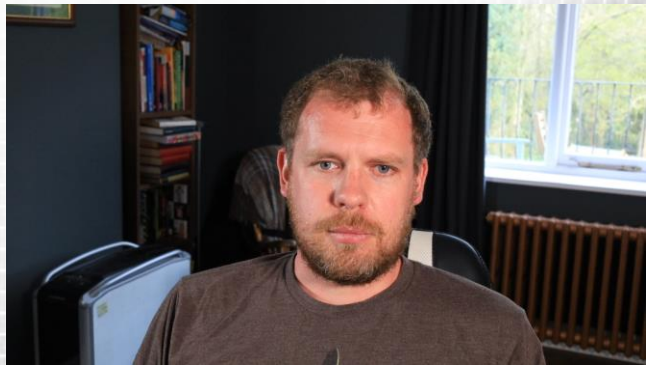
Dr Paul Richmond

http://paulrichmond.shef.ac.uk/teaching/COM4521/



This Lecture (learning objectives)

- Multi GPU Programming
  - Demonstrate how to change the GPU device
  - Explain how devices can be concurrently operated and synchronised using streams
  - Demonstrate mechanisms for device to device asynchronous memory copying

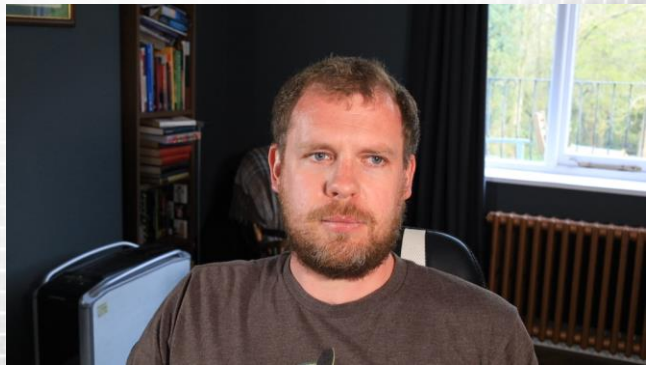


Multi GPU Programming

- By default CUDA uses the first device in the system
  - Not necessarily the fastest device!
- Device can be changed using `cudaSetDevice(int)`
  - Device capabilities can be queried using device properties API

```
int deviceCount = 0;
cudaGetDeviceCount(&deviceCount);

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



Multi GPU Devices and Streams

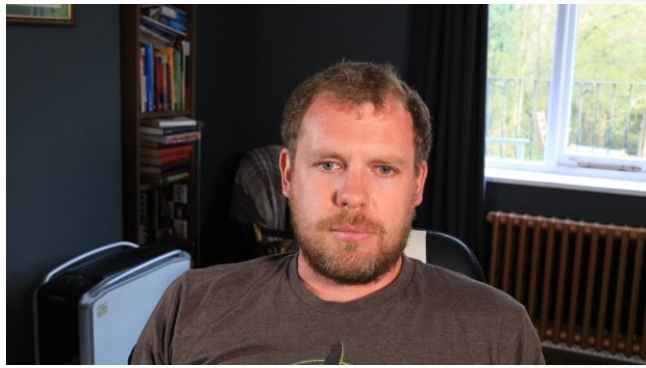
- Streams and events belong to a single device
  - The device which is active when created
  - Synchronising and Querying of streams across devices is allowed

```
cudaStream_t streamA, streamB;
cudaEvent_t eventA, eventB;

cudaSetDevice(0);
cudaStreamCreate(&streamA); // streamA and eventA belong to device-0
cudaEventCreate(&eventA);
```

```
cudaSetDevice(1);
cudaStreamCreate(&streamB); // streamB and eventB belong to device-1
cudaEventCreate(&eventB);
kernel << <..., streamB >> >(...);
cudaEventRecord(eventB, streamB);
```

```
cudaSetDevice(0);
cudaEventSynchronize(eventB);
kernel << <..., streamA >> >(...);
```

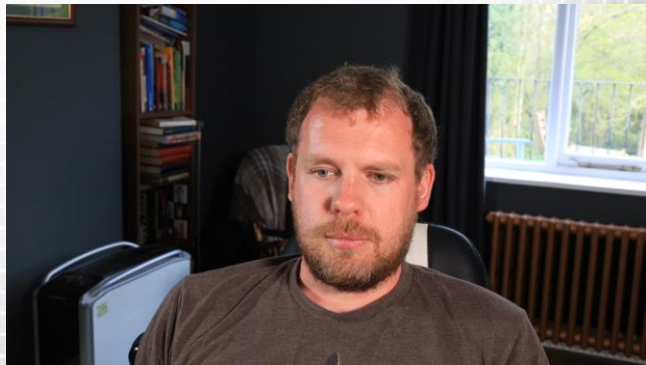


Error: eventA belongs to device 0

Event can be synchronised across devices

Peer to Peer Memory Copies

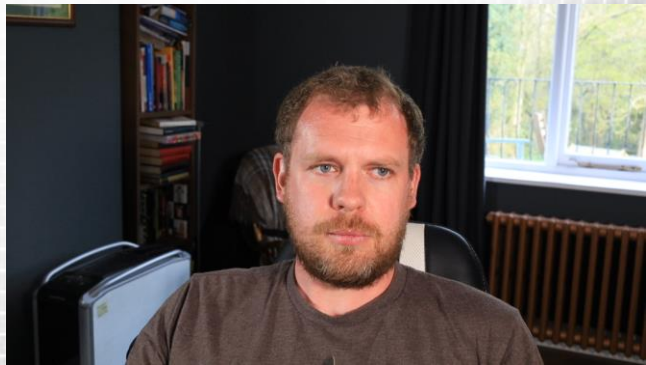
- For devices to interact memory must be copied between them
- Memory can be copied using
  - `cudaMemcpyPeerAsync( void* dst_addr, int dst_dev, void* src_addr, int src_dev, size_t num_bytes, cudaStream_t stream )`
    - Uses shortest PCI path or GPUDirect if available
    - Not staged through CPU
- You can check that a peer (device) can access another using
  - `cudaDeviceCanAccessPeer( &accessible, dev_X, dev_Y )`
- Also possible to use CUDA aware MPI
  - Allows direct transfers over the network
  - With NVLink this will allow GPU to GPU peer access via infiniband
  - *Not covered in this course…*



Further Reading & Acknowledgements

- Most slide examples are based on the excellent GTC and SC material
  - http://www.sie.es/wp-content/uploads/2015/12/cuda-streams-best-practices-common-pitfalls.pdf
  - http://on-demand.gputechconf.com/gtc-express/2011/presentations/StreamsAndConcurrencyWebinar.pdf
  - http://www.nvidia.com/docs/IO/116711/sc11-multi-gpu.pdf

- More reading
  - https://devblogs.nvidia.com/parallelforall/gpu-pro-tip-cuda-7-streams-simplify-concurrency/
  - https://devblogs.nvidia.com/parallelforall/how-overlap-data-transfers-cuda-cc/



Summary

- Multi GPU Programming
  - Demonstrate how to change the GPU device
  - Explain how devices can be concurrently operated and synchronised using streams
  - Demonstrate mechanisms for device to device asynchronous memory copying