Parallel Computing with GPUs

GPU Architectures
Part 2 – Programming GPUs

The University Of Sheffield.

Dr Paul Richmond

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

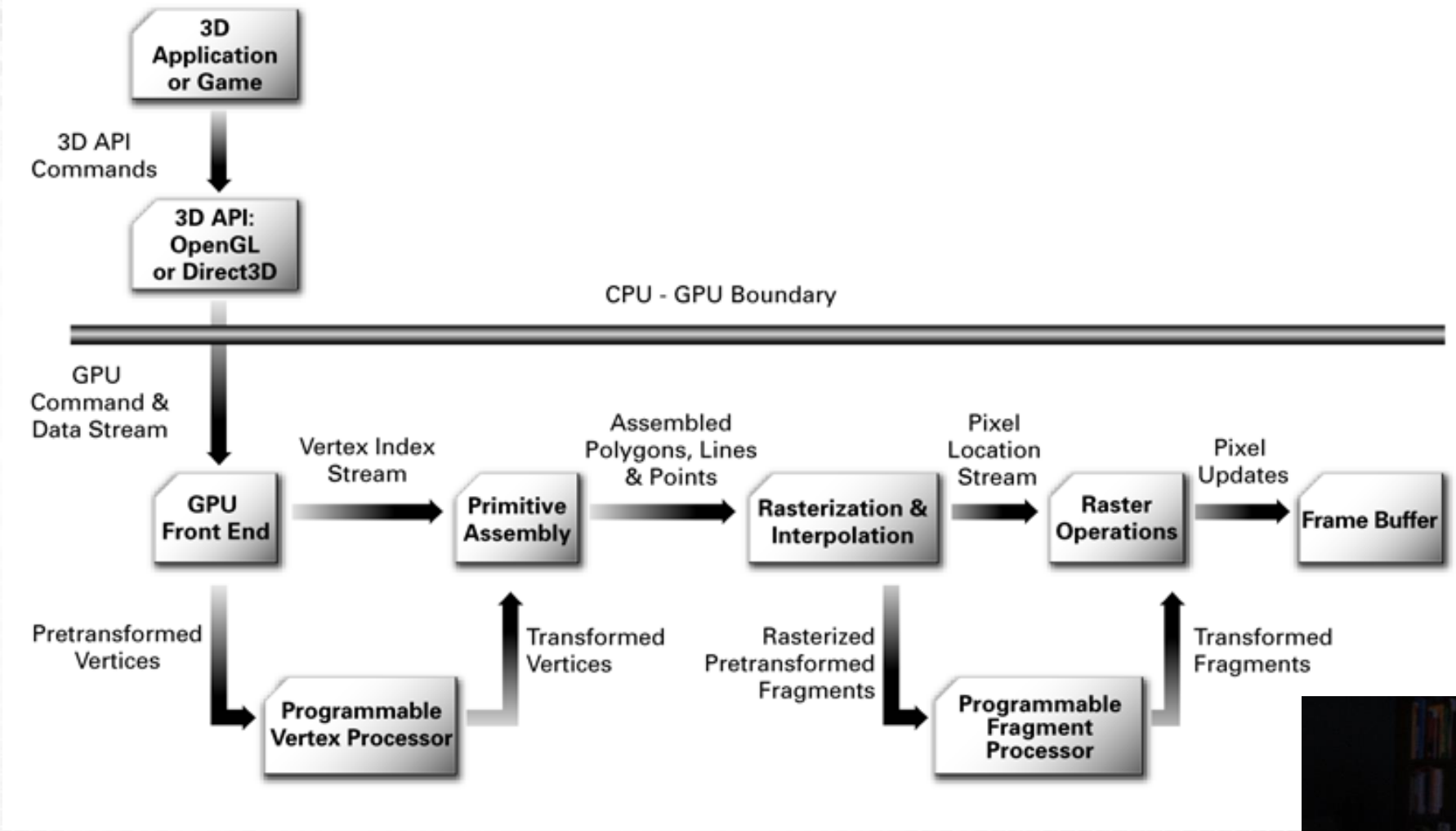
This Lecture (learning objectives)

- Programming GPUs
  - Summarise history around the development of GPU programming techniques
  - Compare a range of approaches for GPU programming

GPU Early History

- Hardware has evolved from the demand for increased quality of 3D computer graphics
- Initially specialised processors for each part of the graphics pipeline
  - Vertices (points of triangles) and Fragments (potential pixels) can be manipulated in parallel
- The stages of the graphics pipeline became programmable in early 2000's
  - NVIDIA GeForce 3 and ATI Radeon 9700
  - DirectX 9.0 required programmable pixel and vertex shaders

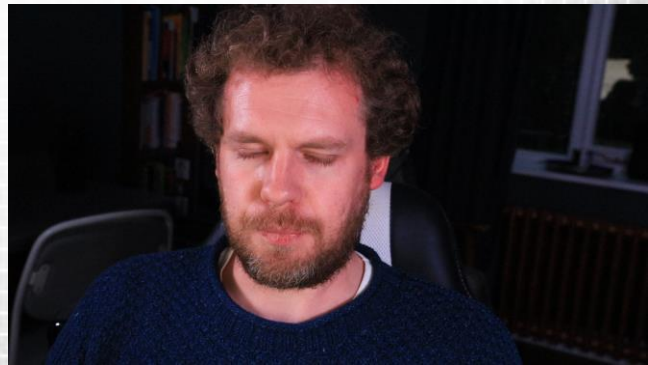
The Graphics Pipeline



Source: *NVidia Cg Users Manual*

GPGPU

- General Purpose computation on Graphics Hardware
  - First termed by Mark Harris (NVIDIA) in 2002
  - Recognised the use of GPUs for non graphics applications
- Requires mapping a problem into graphics concepts
  - Data into textures (images)
  - Computation into shaders
- Later unified processors were used rather than fixed stages
  - 2006: GeForce 8 series



Unified Processors and CUDA

- Compute Unified Device Architecture (CUDA)
  - First released in 2006/7
- Targeted new bread of unified “streaming multiprocessors”
- C like programming for GPUs
  - No computer graphics: General purpose programming model
  - Revolutionised GPU programming for general purpose use

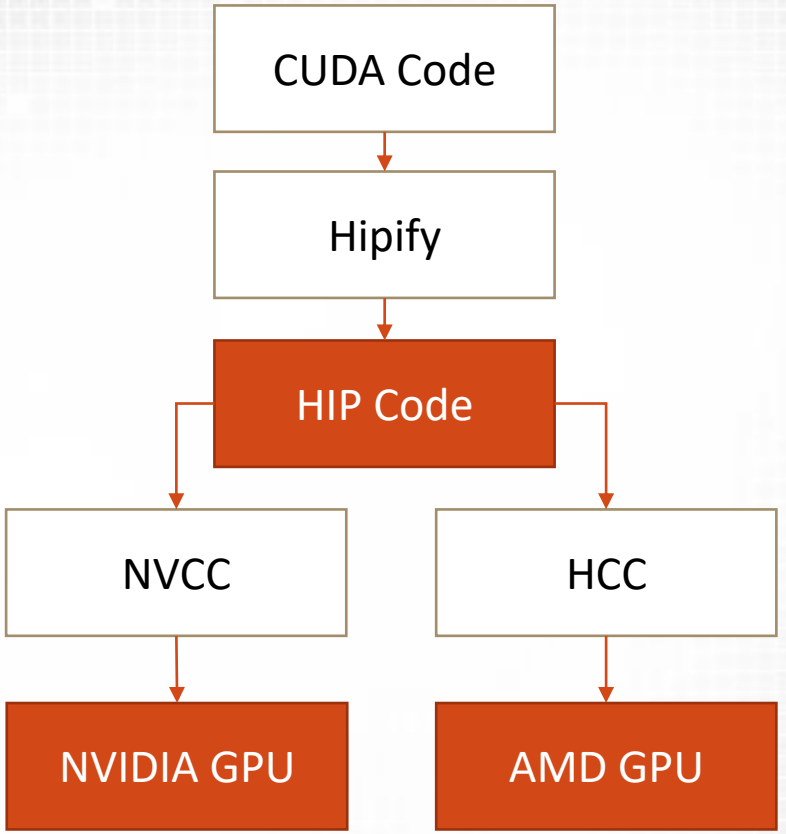
Directive based GPU programming

- GPU Accelerated Directives (OpenACC)
  - Helps compiler auto generate code for the GPU
  - Very similar to OpenMP
  - *Pros: Performance portability, limited understanding of hardware required*
  - *Cons: Limited fine grained control of optimisation*
- OpenMP 4.0
  - GPU offload for parallelism
  - *Pros: Platform and hardware independent, write once*
  - *Cons: Difficult to obtain high performance or use cutting edge features*

```
#pragma omp target data map (to: c[0:N], b[0:N]) map(tofrom: a[0:N])
#pragma omp target teams distribute parallel for
for (j=0; j<N; j++){
    a[j] = b[j]+scalar*c[j];
}
```

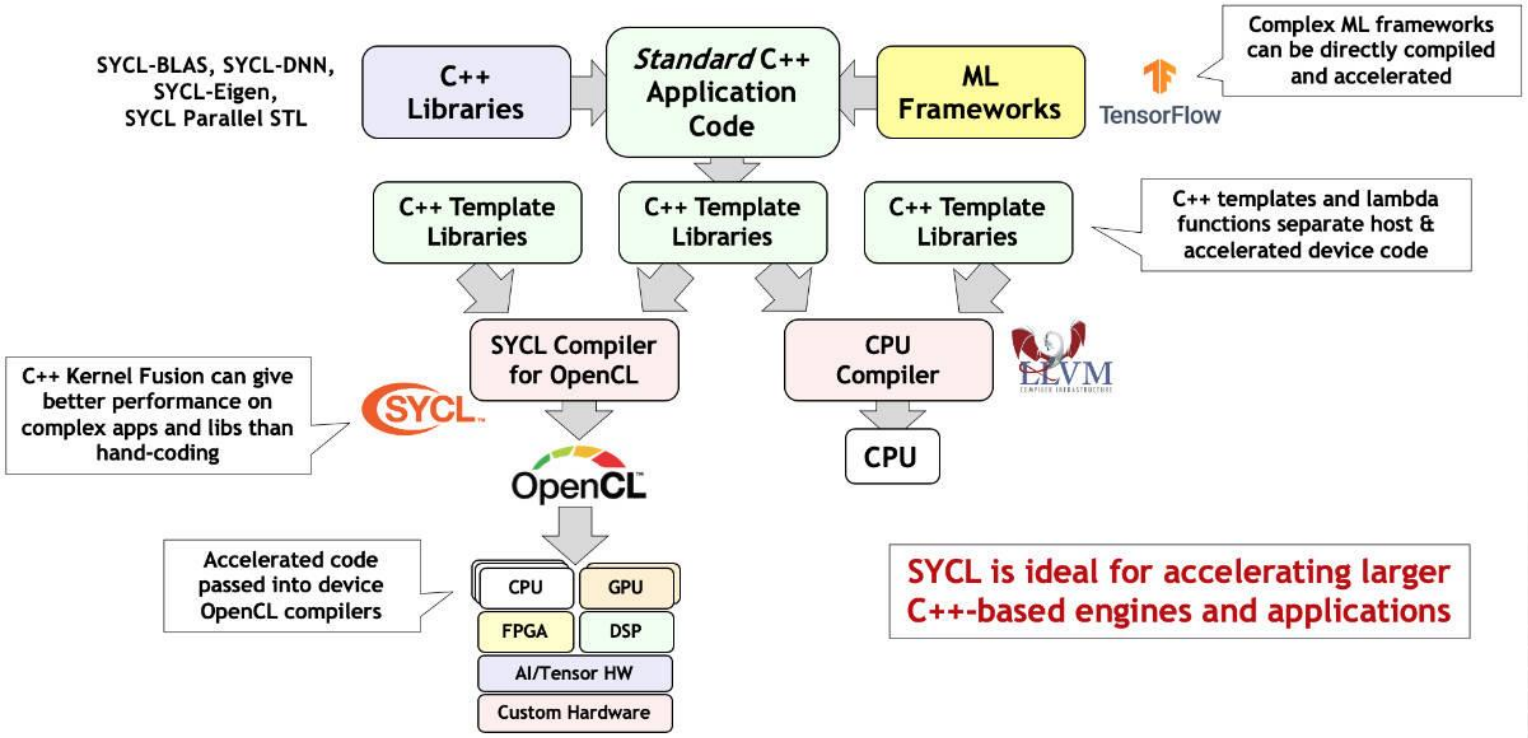
ROCm and HIP

- Radeon Open Compute (ROCm)
  - Platform and runtime for Gpu compute
  - AMD open equivalent of CUDA
- Heterogeneous-Compute Interface for Portability (HIP)
  - C++ interface
  - One to one replacement for CUDA
  - HIP source to source conversion tools
- *Pros: Can run on AMD and NVIDIA GPU hardware*
- *Cons: Subset of the CUDA language, not truly performance portable*



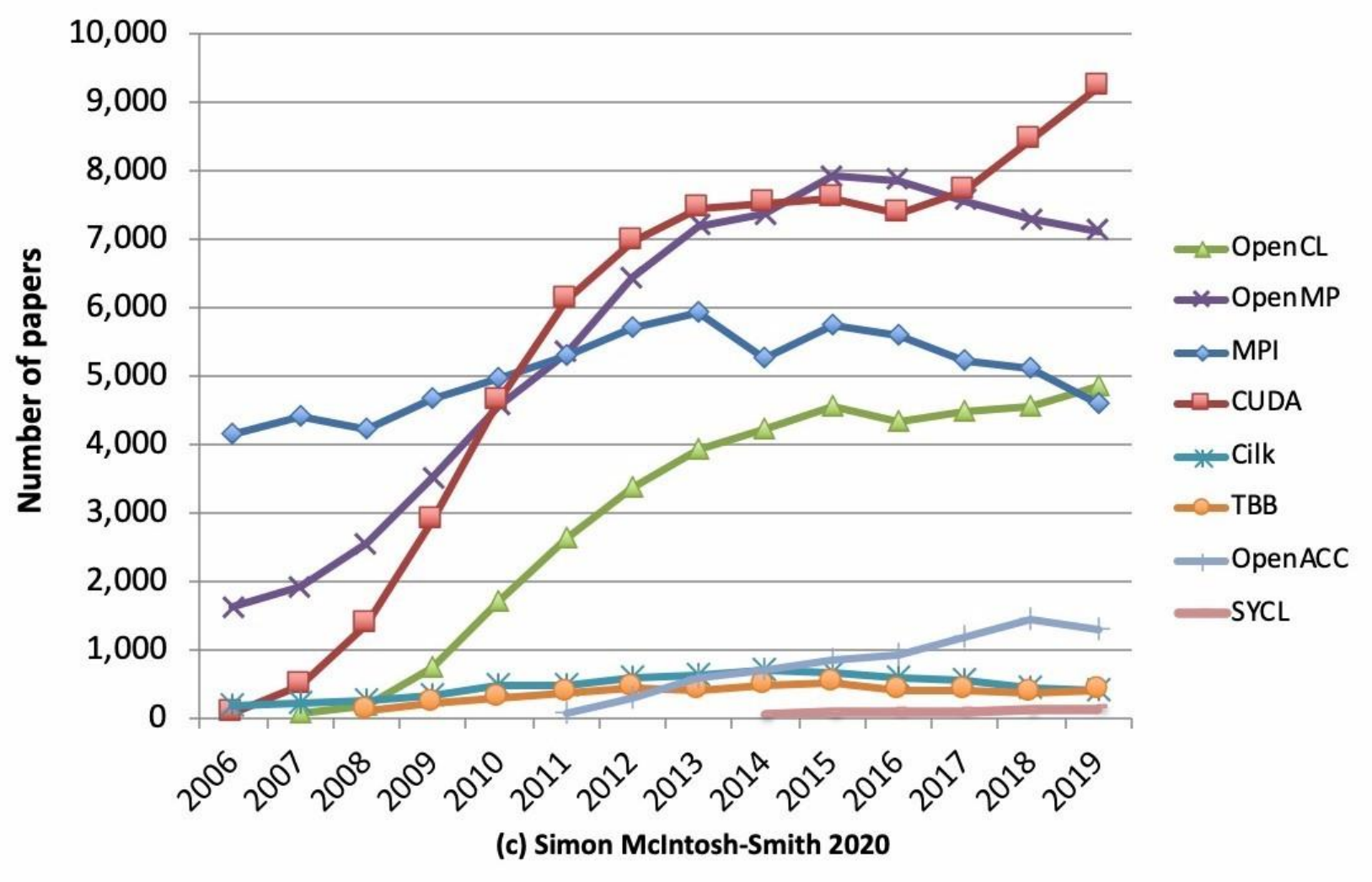
OpenCL and SYCL

- OpenCL: Multiple architecture support (CPUs/GPUs/FPGA)
  - Lower level than CUDA
  - Portability diminished if code is “targeted”
- SYCL: Based on modern C++ (C++17)
  - Performance portable
  - Implementations supported by different vendors (e.g. hipSYCL)



https://www.khronos.org/sycl/

HPC language use in research



https://www.nextplatform.com/2020/04/28/programming-in-the-parallel-universe/

Summary

- Programming GPUs
  - Summarise history around the development of GPU programming techniques
  - Compare a range of approaches for GPU programming

- Next Lecture: NVIDIA Hardware Model