OpenCL

Lecture 4

Things to think about when optimising OpenCL

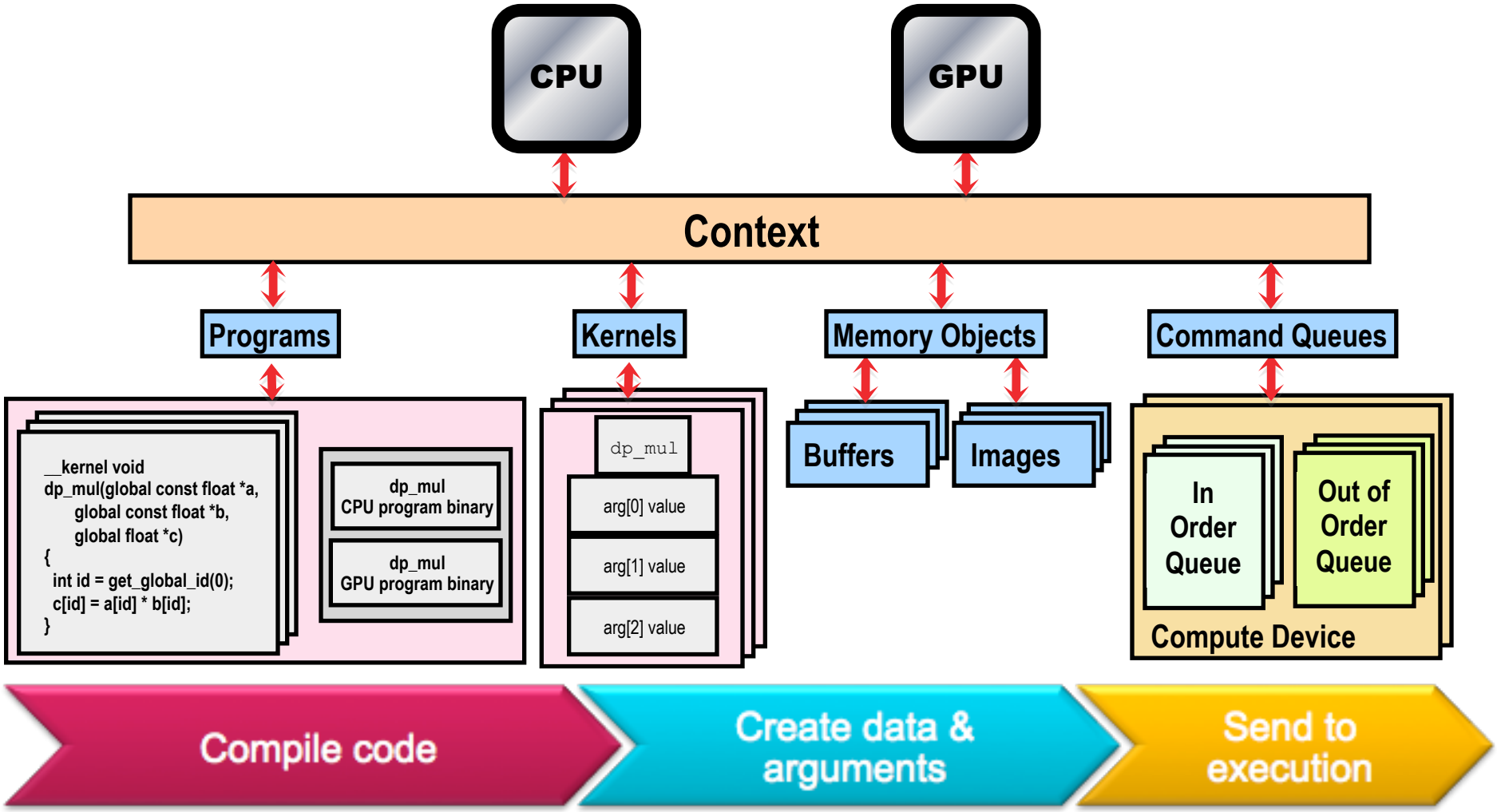
Based on material by Benedict Gaster and Lee Howes (AMD), Tim Mattson (Intel) and several others.

University of BRISTOL

Agenda

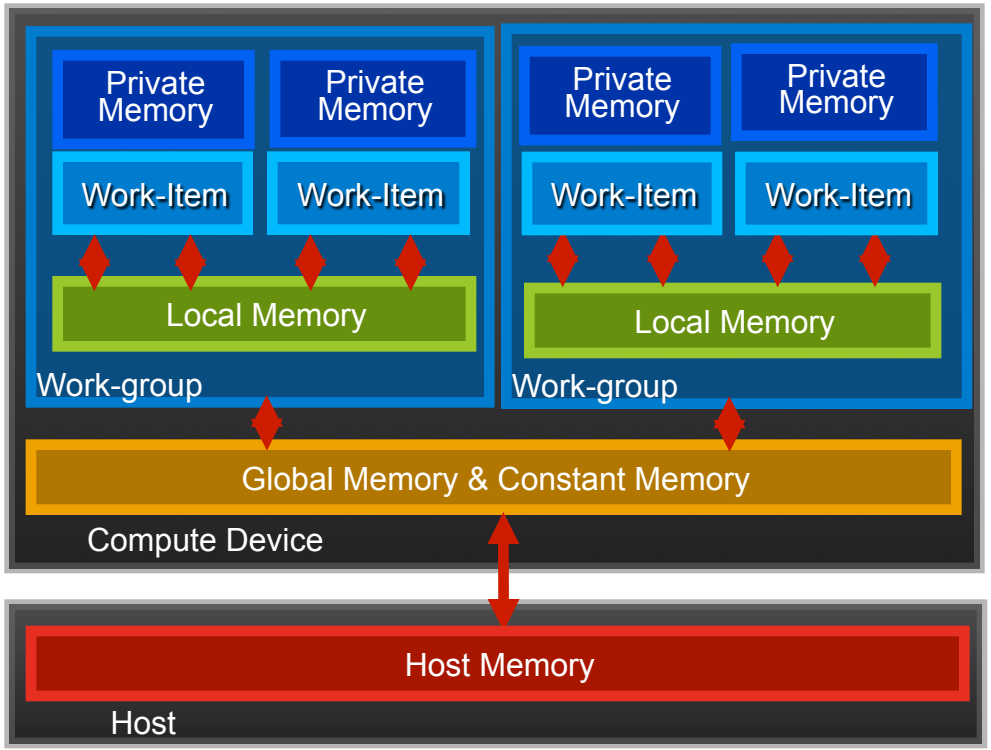
- **Heterogeneous computing and the origins of OpenCL**
- **OpenCL overview**
- **Exploring the spec through a series of examples**
  - **Vector addition:**
    - *the basic platform layer*
  - **Matrix multiplication:**
    - *writing simple kernels*
  - **Optimizing matrix multiplication:**
    - *work-groups and the memory model*
  - **General optimisation tips**
- **A survey of OpenCL 1.1**

OpenCL summary



OpenCL Memory Model

- **Private Memory**
  - Per work-item
- **Local Memory**
  - Shared within a work-group
- **Global / Constant Memories**
  - Visible to all work-groups
- **Host Memory**
  - On the CPU



- **Memory management is explicit:**
  You must move data from host -> global -> local *and* back

Optimisation issues (I)

- **Efficient access to memory**
  - Memory coalescing
    - Ideally get work-item *i* to access *data[i]* and work-item *j* to access *data[j]* in parallel
  - Memory alignment
    - GPU memory interfaces are most efficient when accessing aligned multiples of 32, 64 or 128 bytes in DRAM. Try to use all of the data in each access
  - Be careful not to use too much private or local memory
    - Check with Nvidia's tools to see how much private memory (registers) being used

- **Exploit the memory hierarchy**
  - Host → Global / Constant → Local → Private
  - Biggest/Slowest (Smallest/Fastest)

- **Use asynchronous memory transfers**
  - Do useful work on the host/GPU overlapped with data movement
  - Especially useful during relatively slow PCI Express data transfers

Static Memory Access Pattern Analysis on a Massively Parallel GPU by Jang, et. al discusses how to effectively map work-items (threads) to the data access patterns of an algorithm

Optimisation issues (II)

- **Number of work-items and work-group sizes**
  - Ideally want 8-12 work-items per PE in a Compute Unit on Fermi
    - They are run multi-threaded on the PE to hide latency (so more is better)
  - However there are diminishing returns, and there is an upper limit
    - Each work-item consumes finite PE resources (private memory, local memory, maximum number of threads)
    - Private, local, global and constant memory usages are reported by the OpenCL compiler if you use the "*-cl-nv-verbose*" clBuildProgram option

- **Work-item divergence**
  - What happens when work-items branch?
  - Remember this is a SIMD (data parallel) model
  - Both paths (if-else) may need to be executed, so avoid where possible

- **All of these optimisations aim to maximise *utilisation***
  - Nvidia's tools report occupancy – aiming for >= 50% (>80% rare)

Optimisation issues (III)

- **Don’t optimise too much!**
  - Architecture-specific optimisations may spoil the portability of your code
  - Instead ride the brute-force advantage of the many-core GPUs and the ease of heterogeneous computing using OpenCL
  - E.g. I don’t like optimising for warp size (NV) or wavefront size (AMD)

- **Use built-in functions and fast-math optimisations**
  - Sin/cos/sqrt/dot et al all *very* fast in hardware
  - The quick reference guide has a good list of what’s available – use them!
  - Add e.g. “-cl-fast-relaxed-math -DMAC” build options to your clBuildProgram () call

- **Use single precision in preference to double if you can**
  - Will go faster, use less memory and less energy too
  - Don’t forget to identify single precision constants as such: e.g. 1.0**f**

Optimisation issues (III)

- **Don’t forget you can go multi-GPU and heterogeneous**
  - The joys of OpenCL…

- **Lots of great stuff in the appendices of the Nvidia OCL guide:**
  - http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/OpenCL_Programming_Guide.pdf

Use the tools, e.g. occupancy calculator

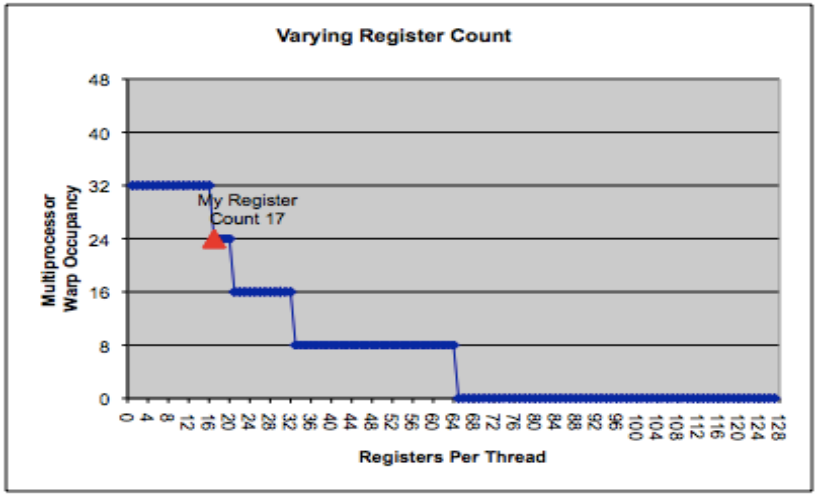
1. Enter hardware model and kernel requirements

1.) Select Compute Capability (click):	2.0
2.) Enter your resource usage:	
Threads Per Block	256
Registers Per Thread	8
Shared Memory Per Block (bytes)	1024

2. Resource usage and limiting factors are displayed

Allocation Per Thread Block	
Warps	8
Registers	4608
Shared Memory	1024
These data are used in computing the occupancy data in blue	
Maximum Thread Blocks Per Multiprocessor	Blocks
Limited by Max Warps / Blocks per Multiprocessor	6
Limited by Registers per Multiprocessor	7
Limited by Shared Memory per Multiprocessor	48

3. Graphs are shown to visualize



Use the tools: e.g. Nvidia visual profiler

Agenda

- **Heterogeneous computing and the origins of OpenCL**
- **OpenCL overview**
- **Exploring the spec through a series of examples**
  - **Vector addition:**
    - *the basic platform layer*
  - **Matrix multiplication:**
    - *writing simple kernels*
  - **Optimizing matrix multiplication:**
    - *work-groups and the memory model*
  - **General optimisation tips**
- **A survey of OpenCL 1.1**

OpenCL

New features in OpenCL 1.1

University of BRISTOL

OpenCL 1.1 – Language

- **Implicit Conversions**
  - OpenCL 1.0 requires widening for arithmetic operators

```
float4 a, b;
float c;
b = a + c; // c is widened to a float4 vector
           // first and then the add is performed
```

  - OpenCL 1.1 extends this feature for all operators
    - relational, equality, bitwise, logical, ternary

OpenCL 1.1 – Language

- **3-component vector data types**
  - Still aligned to multiples of four in storage
- **cl_khr_byte_addressable as core feature**
- **Atomic extensions become core features**
  - **cl_khr_global_int32_{base | extended}_atomics**
  - **cl_khr_local_int32_{base | extended}_atomics**

OpenCL 1.1 – Language

- **New built-in functions:**
  - **get_global_offset**
  - clamp for integer data types
  - **async_work_group_strided_copy**
  - strided async copy of data from global <---> local memory
  - **shuffle** – construct a permutation of elements from 1 or 2 input vectors and a mask

OpenCL 1.1 – API

- Thread-safety
  - All API calls, except **clSetKernelArg**, are thread safe
- Sub-buffer objects
  - Create an object that represents a specific region in a buffer object
  - Easy and efficient mechanism to distribute regions of a buffer object across multiple devices
  - OpenCL synchronization mechanism ensures modifications to sub-buffer object reflected in appropriate region of parent buffer object

OpenCL 1.1 – API

- User Events
  - **clEnqueue*** ** commands can wait on events
  - In OpenCL 1.0, events can only refer to OpenCL commands
  - Need ability to enqueue commands that wait on an external, user defined, event
- Event CallBacks
  - **clSetEventCallbackFn** to register a user callback function
  - called when command identified by event has completed
  - Allows applications to enqueue new OpenCL commands based on event state changes in a non-blocking manner
- Lots more API stuff too

OpenCL 1.1 – OpenCL/OpenGL Sharing

- **Improve performance of OpenCL/ OpenGL interoperability**
  - Portable OpenCL/ OpenGL sharing requires
    - a **glFinish** before **clEnqueueAcquireGLObjects**
    - a **clFinish** after **clEnqueueReleaseGLObjects**
  - **glFinish** / **clFinish** are heavyweight APIs

OpenCL 1.1 – OpenCL/OpenGL Sharing

- **Improve performance of OpenCL/ OpenGL interoperability**
  - Create a OpenCL event from an OpenGL sync object
  - Create a OpenGL sync object from a OpenCL event
  - Allows for a finer grained waiting mechanism
    - Use **event_wait_list** argument for events that refer to OpenGL commands to complete
    - Use OpenGL sync APIs to wait for specific OpenCL™ commands to complete

Conclusion

- **OpenCL defines a platform-API/framework for heterogeneous computing … not just GPGPU or CPU-offload programming**

- **OpenCL has the potential to deliver portably performant code; but it must be used correctly to achieve this:**
  - Implicit SIMD data parallel code has the best chance of mapping onto a diverse range of hardware as OpenCL implementations mature

- **The future is clear:**
  - Heterogeneous parallelism mixing task parallel and data parallel code in a single program … balancing the load among ALL of the platform's available resources, both CPUs and GPUs

Become part of the UK GPU community!

- **Go to the "UK GPU developer conference"**
  - This year at Imperial
- **Apply to use the HECToR GPU cluster**
- **Be part of gpucomputing.net**
  - There's a "UK" community already set up
- **Join the MRSN email list (Many-core and Reconfigurable Supercomputing)**
  - MRSN-request@JISCMAIL.AC.UK
- **Get using it and have FUN!!**