HERIOT WATT UNIVERSITY

Heterogeneous Computing using openCL lecture 1

F21DP Distributed and Parallel Technology

Sven-Bodo Scholz

My Coordinates

HERIOT WATT UNIVERSITY

- Office EM G.27
- email: S.Scholz@hw.ac.uk
- contact time:
  - Thursday after the lecture or
  - on appointment

The Big Picture

- Introduction to Heterogeneous Systems
- OpenCL Basics
- Memory Issues
- Scheduling
- Optimisations

HERIOT WATT UNIVERSITY

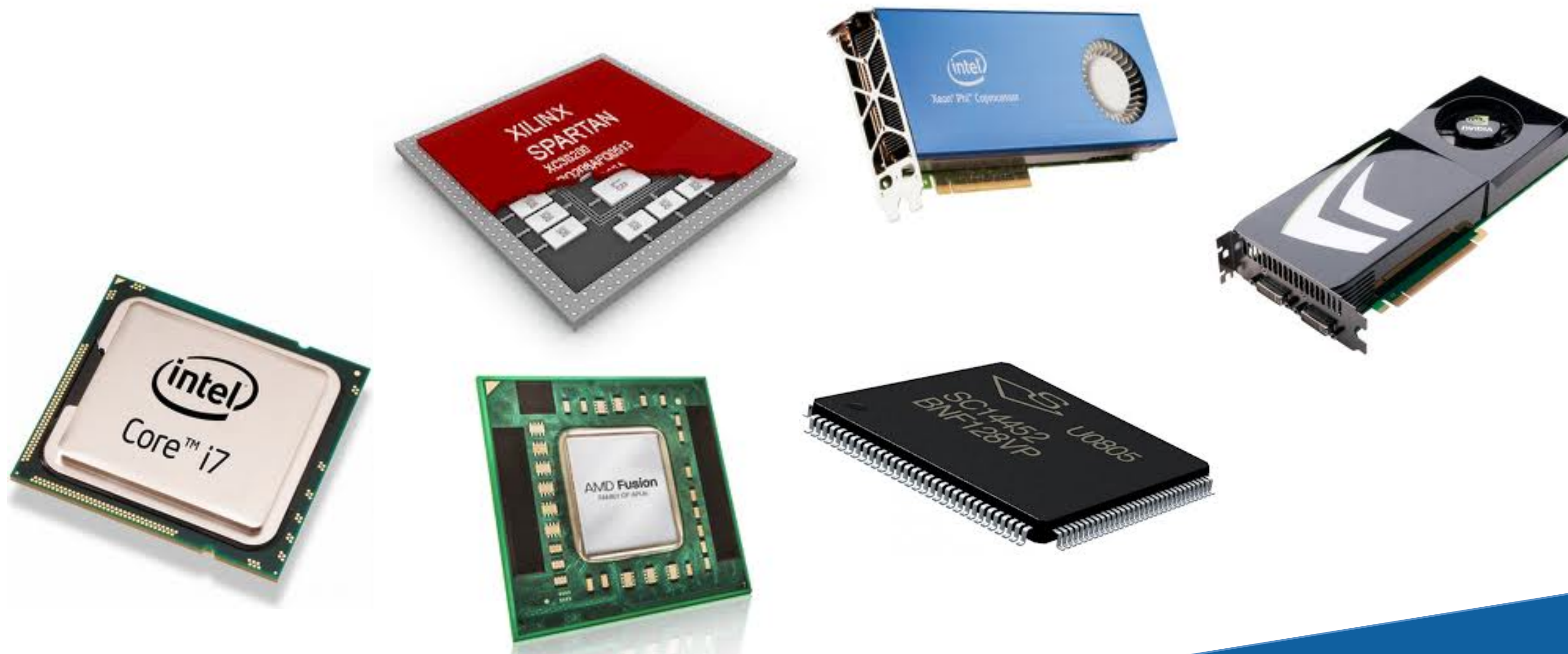
Reading

+ all about openCL

Heterogeneous Systems

HERIOT WATT UNIVERSITY

A Heterogeneous System is a Distributed System containing different kinds of hardware to jointly solve problems.



Focus

- Hardware: SMP + GPUs:

- Programming Model: Data Parallelism

Recap Concurrency

HERIOT WATT UNIVERSITY

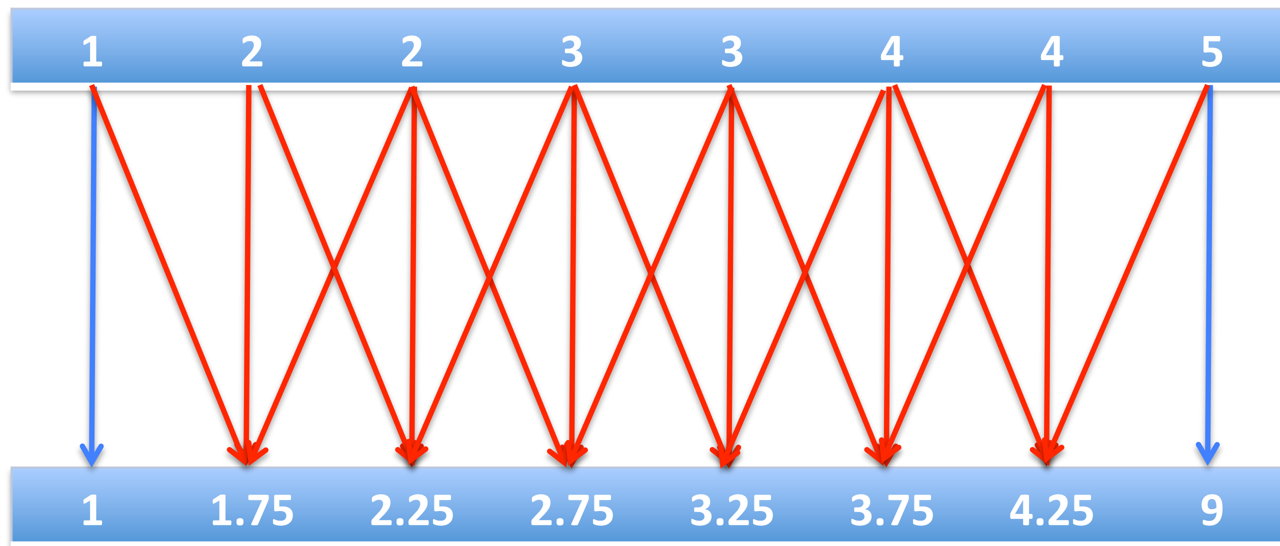
"Concurrency describes a situation where two or more activities happen in the same time interval and we are not interested in the order in which it happens"

Recap Parallelism

HERIOT WATT UNIVERSITY

"In CS we refer to Parallelism if two or more concurrent activities are executed by two or more physical processing entities"

Recap Data-Parallelism



b[i] = .25 * a[i-1] + 0.5 * a[i] + 0.25 * a[i+1]

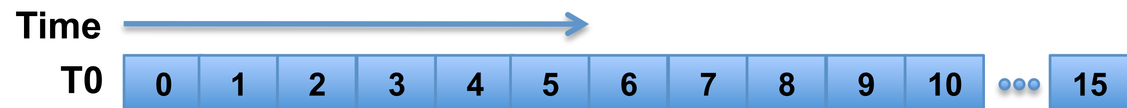
b[i] = a[i]

HERIOT WATT UNIVERSITY

A Peek Preview ☺

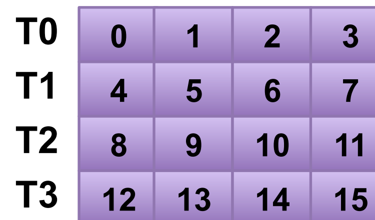
Single-threaded (CPU)

```
// there are N elements
for(i = 0; i < N; i++)
  C[i] = A[i] + B[i]
```



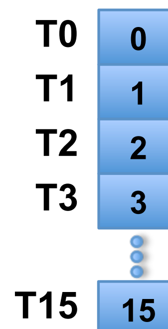
Multi-threaded (CPU)

```
// tid is the thread id
// P is the number of cores
for(i = 0; i < tid*N/P; i++)
  C[i] = A[i] + B[i]
```



Massively Multi-threaded (GPU)

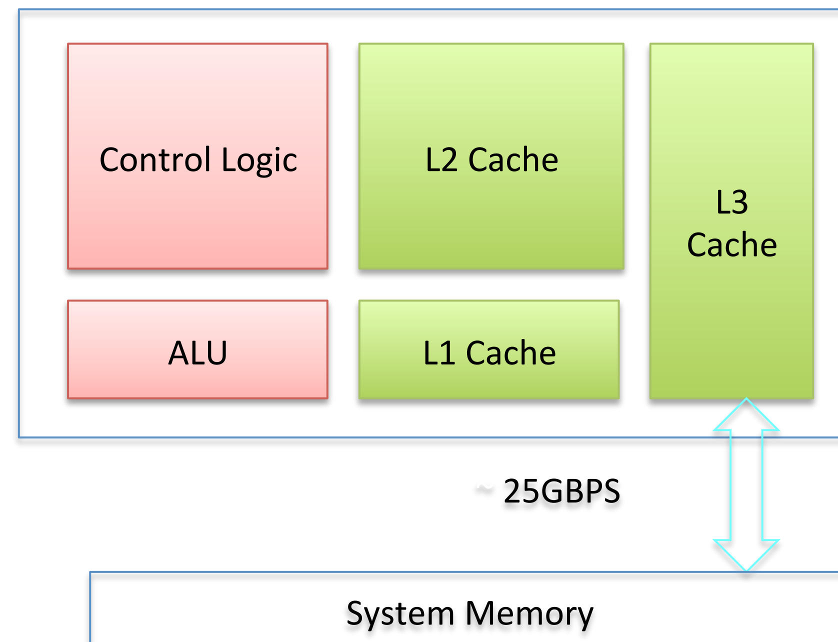
```
// tid is the thread id
C[tid] = A[tid] + B[tid]
```



Conventional CPU Architecture

- Space devoted to control logic instead of ALU
- CPUs are optimized to minimize the latency of a single thread
  - Can efficiently handle control flow intensive workloads
- Multi level caches used to hide latency
- Limited number of registers due to smaller number of active threads
- Control logic to reorder execution, provide ILP and minimize pipeline stalls

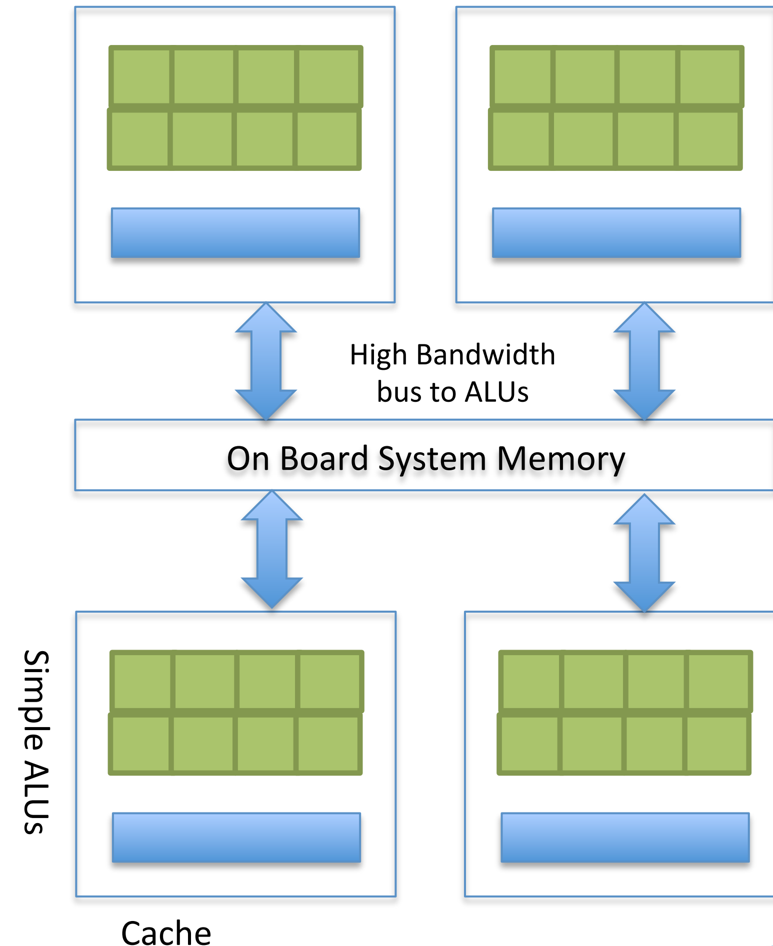
Conventional CPU Block Diagram



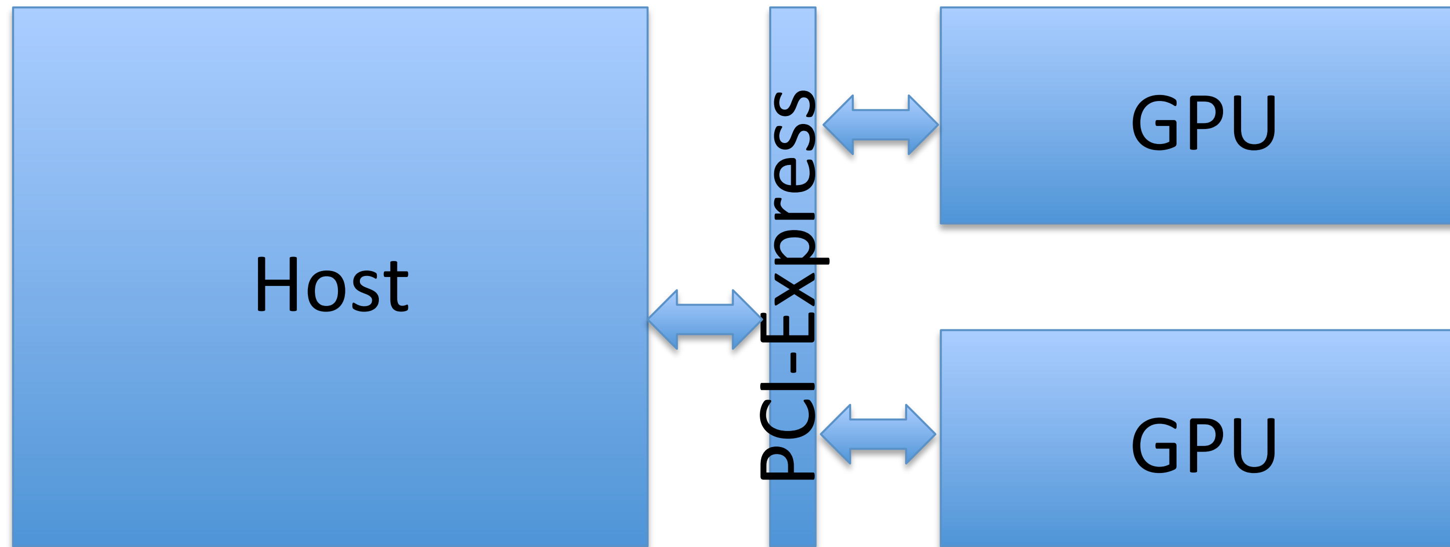
A present day multicore CPU could have more than one ALU (typically < 32) and some of the cache hierarchy is usually shared across cores

Modern GPGPU Architecture

- Generic many core GPU
  - Less space devoted to control logic and caches
  - Large register files to support multiple thread contexts
- Low latency hardware managed thread switching
- Large number of ALU per "core" with small user managed cache per core
- Memory bus optimized for bandwidth
  - ~150 GBPS bandwidth allows us to service a large number of ALUs simultaneously



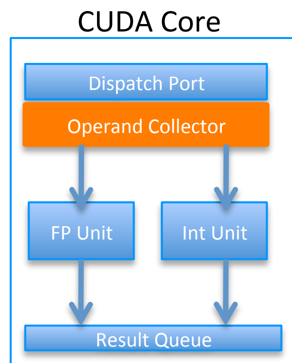
Typical System

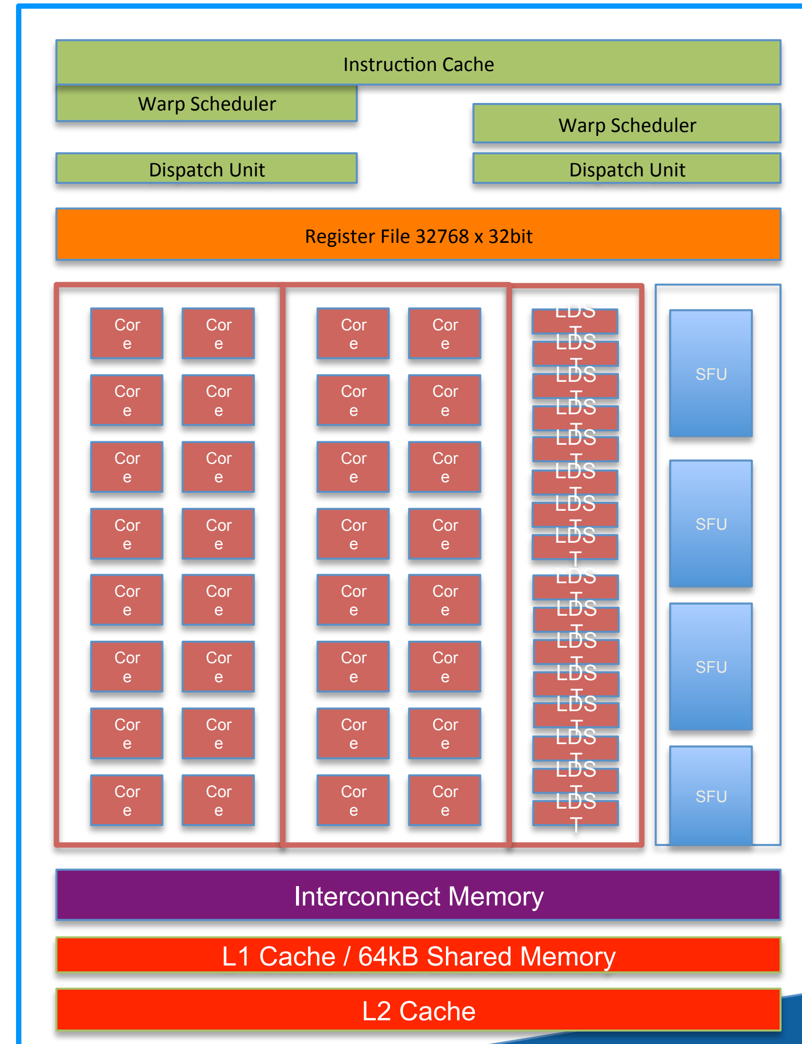


- Host initiated memory transfers
- Host initiated computations on the GPU (kernels)

Nvidia GPUs
Fermi Architecture

- GTX 480 - Compute 2.0 capability
  - 15 cores or Streaming Multiprocessors (SMs)
  - Each SM features 32 CUDA processors
  - 480 CUDA processors
- Global memory with ECC

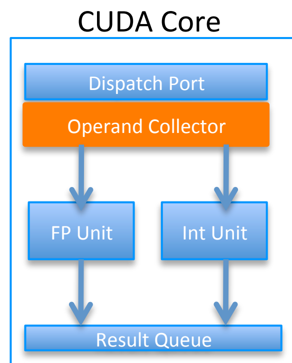


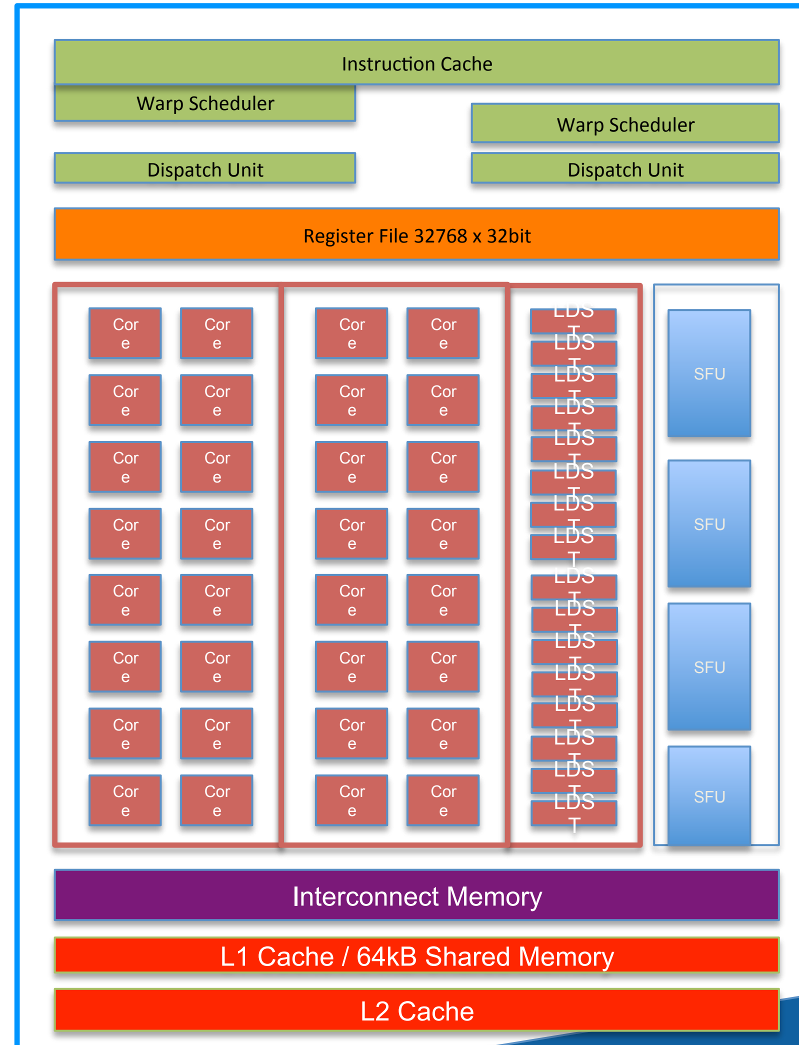


Nvidia GPUs
Fermi Architecture

HERIOT WATT UNIVERSITY

- SM executes threads in groups of 32 called warps.
  - Two warp issue units per SM
- Concurrent kernel execution
  - Execute multiple kernels simultaneously to improve efficiency
- CUDA core consists of a single ALU and floating point unit FPU





SIMD vs SIMT

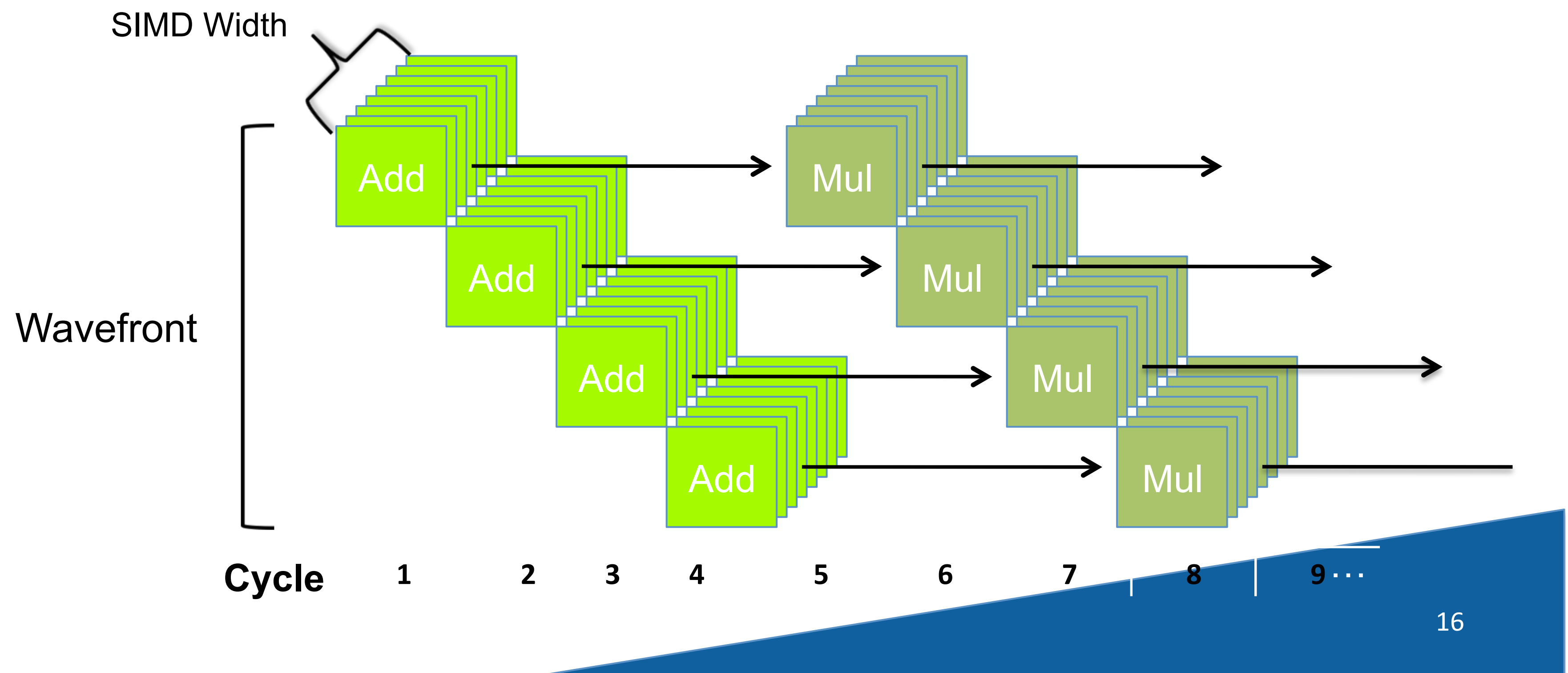
HERIOT WATT UNIVERSITY

- SIMT denotes scalar instructions and multiple threads sharing an instruction stream
  - HW determines instruction stream sharing across ALUs
  - E.g. NVIDIA GeForce ("SIMT" warps), AMD Radeon architectures ("wavefronts") where all the threads in a warp /wavefront proceed in lockstep
  - Divergence between threads handled using predication
- SIMT instructions specify the execution and branching behavior of a single thread
- SIMD instructions exposes vector width,
  - E.g. of SIMD: explicit vector instructions like x86 SSE

SIMT Execution Model

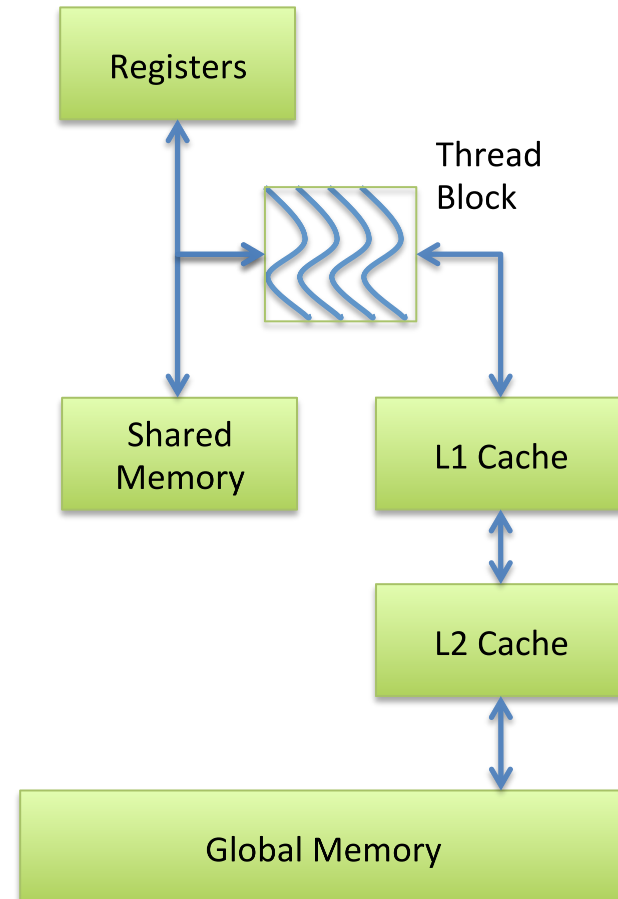
HERIOT WATT UNIVERSITY

- SIMD execution can be combined with pipelining
  - ALUs all execute the same instruction
  - Pipelining is used to break instruction into phases
    - When first instruction completes (4 cycles here), the next instruction is ready to execute



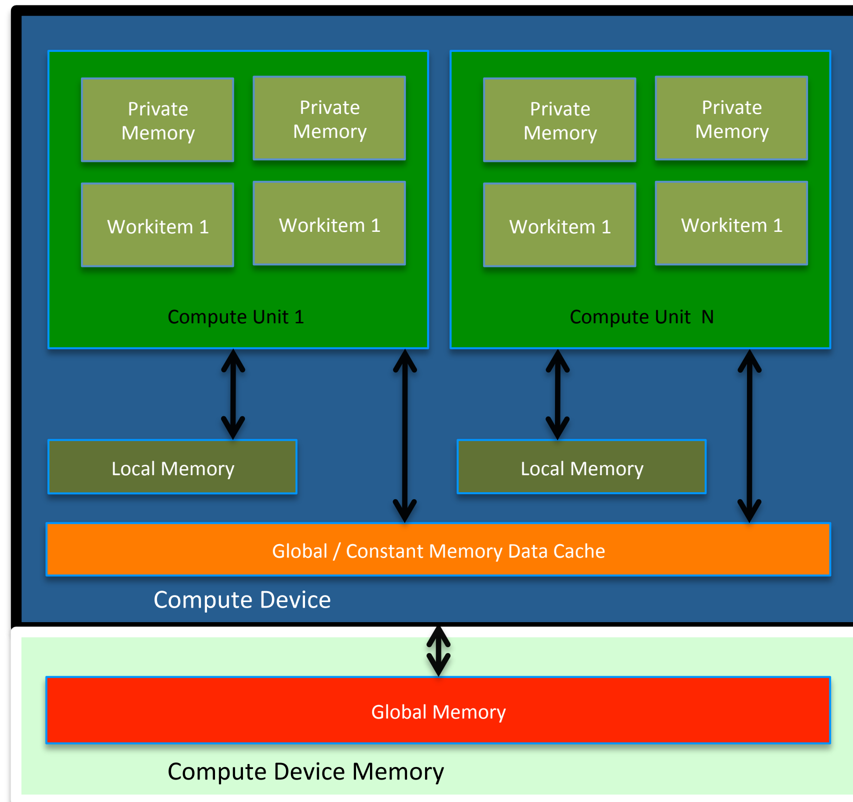
NVIDIA Memory Hierarchy

- L1 cache per SM configurable to support shared memory and caching of global memory
  - 48 KB Shared / 16 KB of L1 cache
  - 16 KB Shared / 48 KB of L1 cache
- Data shared between work items of a group using shared memory
- Each SM has a 32K register bank
- L2 cache (768KB) that services all operations (load, store and texture)
  - Unified path to global for loads and stores



NVIDIA Memory Model in OpenCL

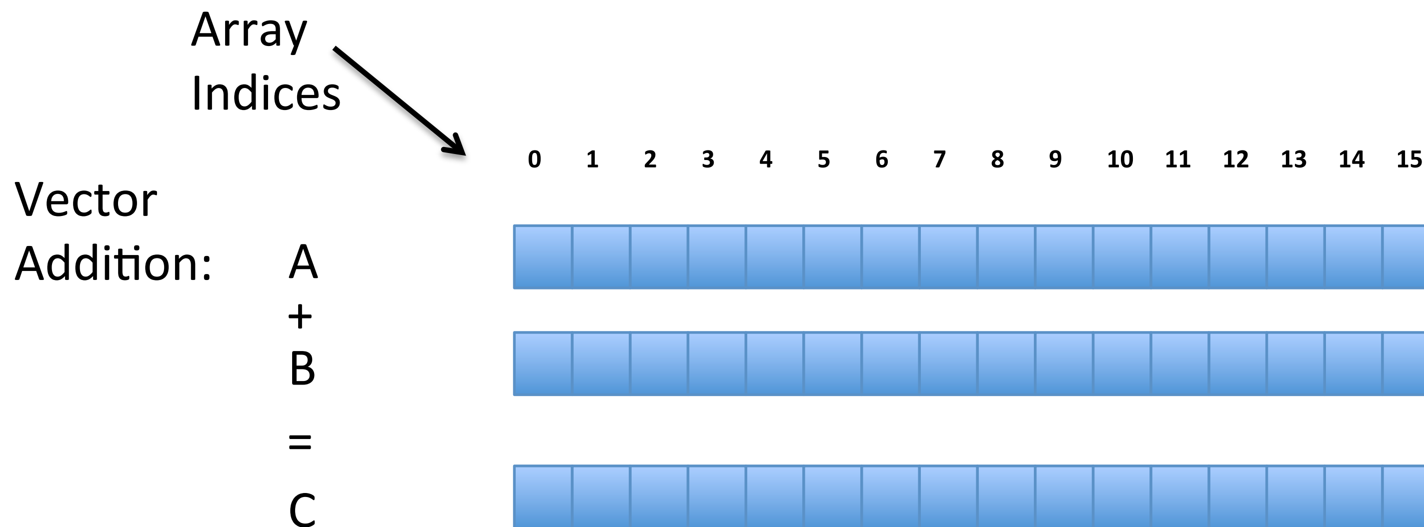
HERIOT WATT UNIVERSITY



- Like AMD, a subset of hardware memory exposed in OpenCL
- Configurable shared memory is usable as local memory
  - Local memory used to share data between items of a work group at lower latency than global memory
- Private memory utilizes registers per work item

Mapping Threads to Data

- Consider a simple vector addition of 16 elements
  - 2 input buffers (A, B) and 1 output buffer (C) are required



Mapping Threads to Data

- Create thread structure to match the problem
  - 1-dimensional problem in this case

Thread Code:

C[tid] = A[tid] + B[tid]

Thread IDs

Thread Structure

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Vector Addition: A + B = C

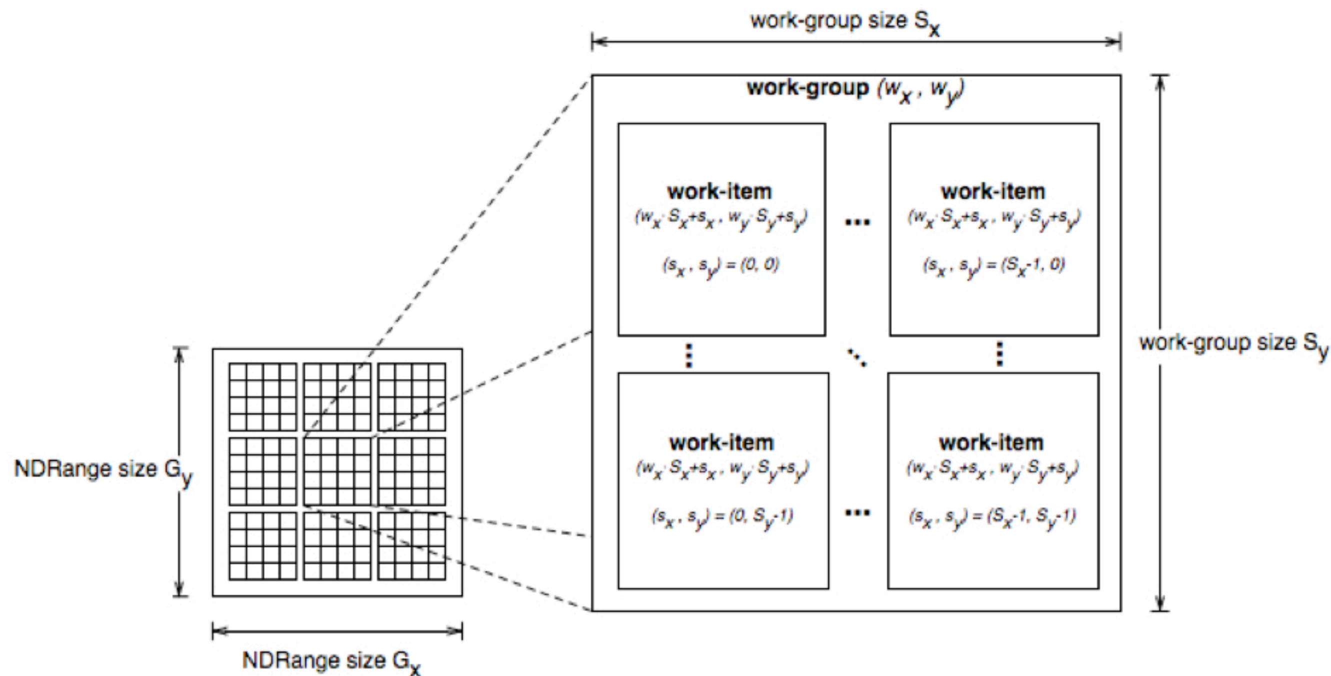
Thread Structure

HERIOT WATT UNIVERSITY

- OpenCL's thread structure is designed to be scalable
- Each instance of a kernel is called a work-item (though "thread" is commonly used as well)
- Work-items are organized as work-groups
  - Work-groups are independent from one-another (this is where scalability comes from)
- An index space defines a hierarchy of work-groups and work-items

Thread Structure

- Work-items can uniquely identify themselves based on:
  - A global id (unique within the index space)
  - A work-group ID and a local ID within the work-group



If you are excited.....

HERIOT WATT UNIVERSITY

- you can get started on your own ☺
- google AMD openCL
- or Intel openCL
- or Apple openCL
- or Khronos
- download an SDK and off you go!