GPU Architecture

A Collaboration Between
David Kaeli, Northeastern University
Benedict R. Gaster, AMD

Instructor Notes

- We describe motivation for talking about underlying device architecture because device architecture is often avoided in conventional programming courses
- Contrast conventional multicore CPU architecture with high level view of AMD and Nvidia GPU Architecture
- This lecture starts with a high level architectural view of all GPUs, discusses each vendor's architecture and then converges back to the OpenCL spec
  - Stress on the difference between the AMD VLIW architecture and Nvidia scalar architecture
  - Also discuss the different memory architecture
- Brief discussion of ICD and compilation flow of OpenCL provides a lead to Lecture 5 where the first complete OpenCL program is written

Topics

- Mapping the OpenCL spec to many-core hardware
  - AMD GPU Architecture
  - Nvidia GPU Architecture
  - Cell Broadband Engine
- OpenCL Specific Topics
  - OpenCL Compilation System
  - Installable Client Driver (ICD)

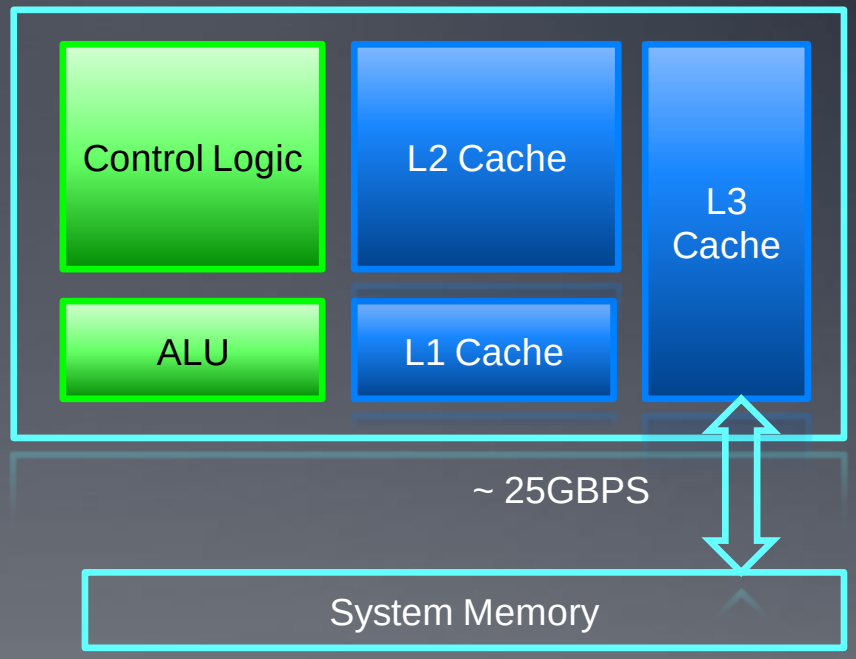
Motivation

- Why are we discussing vendor specific hardware if OpenCL is platform independent ?
  - Gain intuition of how a program's loops and data need to map to OpenCL kernels in order to obtain performance
  - Observe similarities and differences between Nvidia and AMD hardware
  - Understanding hardware will allow for platform specific tuning of code in later lectures

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

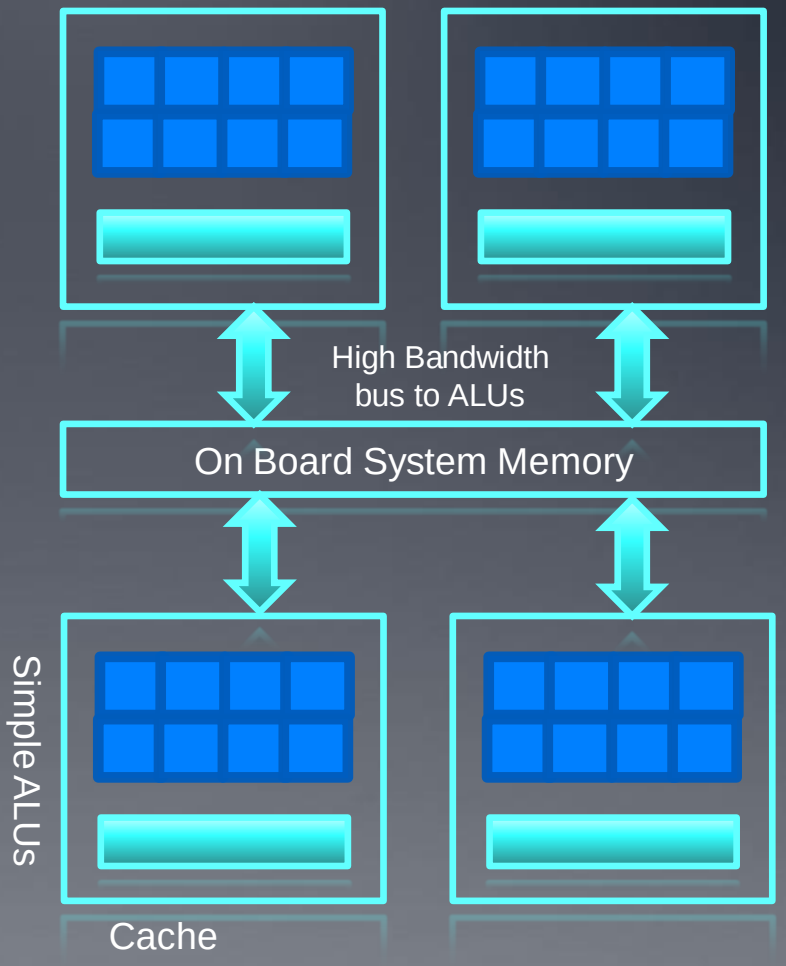
Conventional CPU Block Diagram



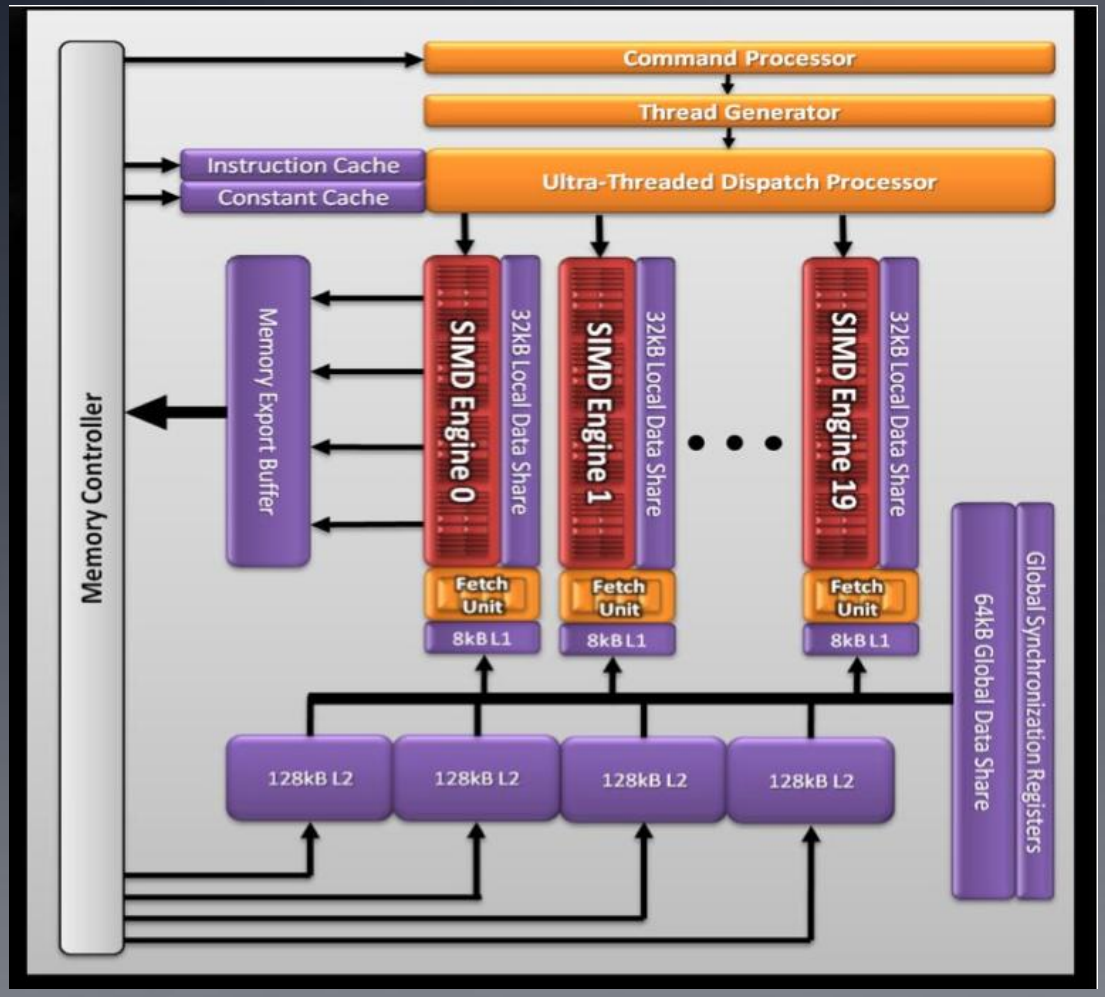
A present day multicore CPU could have more than one ALU (typically < 32) and some of the cache hierarchy is usually shared across cores

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



AMD GPU Hardware Architecture

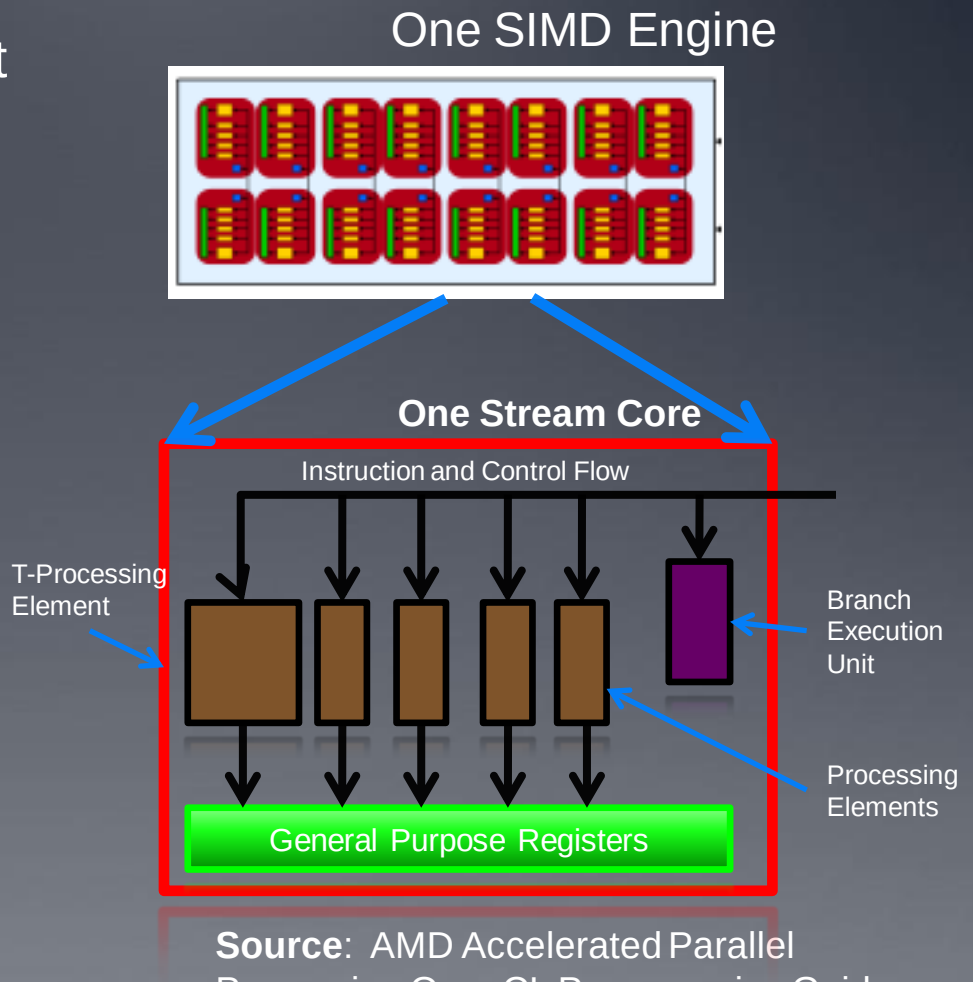


- AMD 5870 – Cypress
- 20 SIMD engines
- 16 SIMD units per core
- 5 multiply-adds per functional unit (VLIW processing)
- 2.72 Teraflops Single Precision
- 544 Gigaflops Double Precision

Source: Introductory OpenCL *SAAHPC2010,* Benedict R. Gaster

SIMD Engine

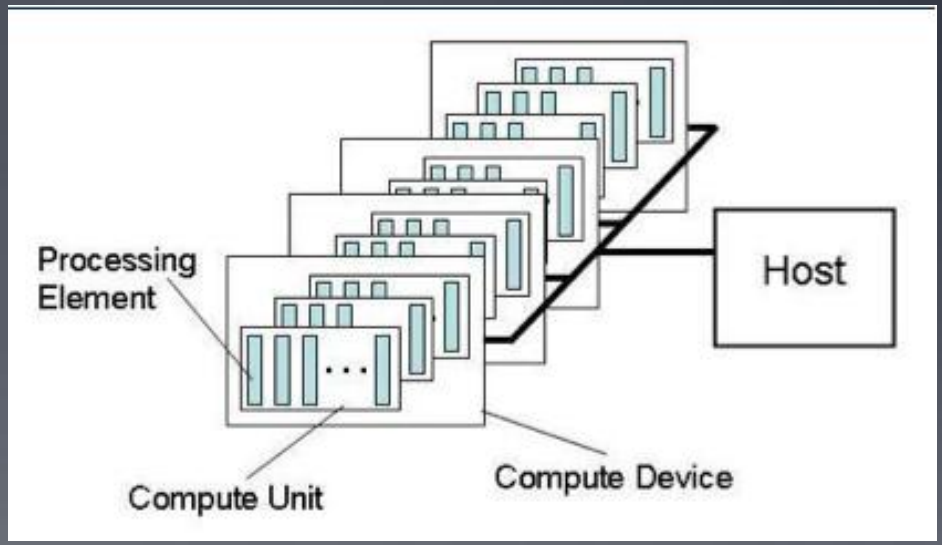
- A SIMD engine consists of a set of “Stream Cores”
- Stream cores arranged as a five way Very Long Instruction Word (VLIW) processor
  - Up to five scalar operations can be issued in a VLIW instruction
  - Scalar operations executed on each processing element
- Stream cores within compute unit execute same VLIW instruction
  - The block of work-items that are executed together is called a wavefront.
  - 64 work items for 5870



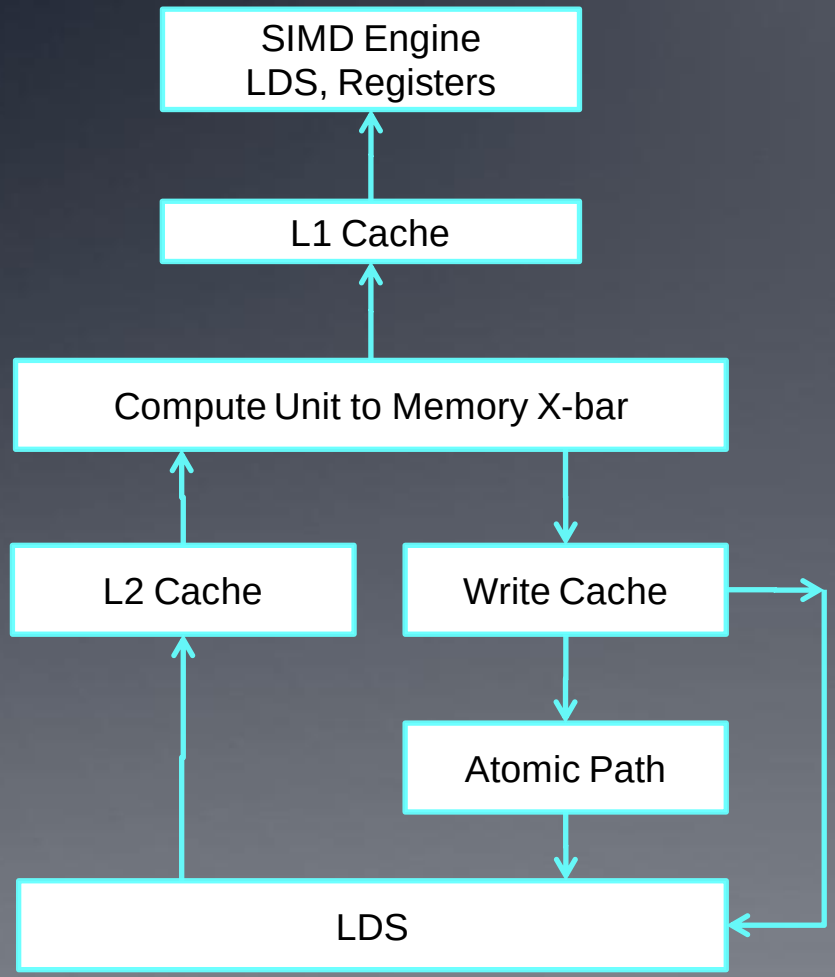
Source: AMD Accelerated Parallel Processing OpenCL Programming Guide

AMD Platform as seen in OpenCL

- Individual work-items execute on a single processing element
- Processing element refers to a single VLIW core
- Multiple work-groups execute on a compute unit
- A compute unit refers to a SIMD Engine

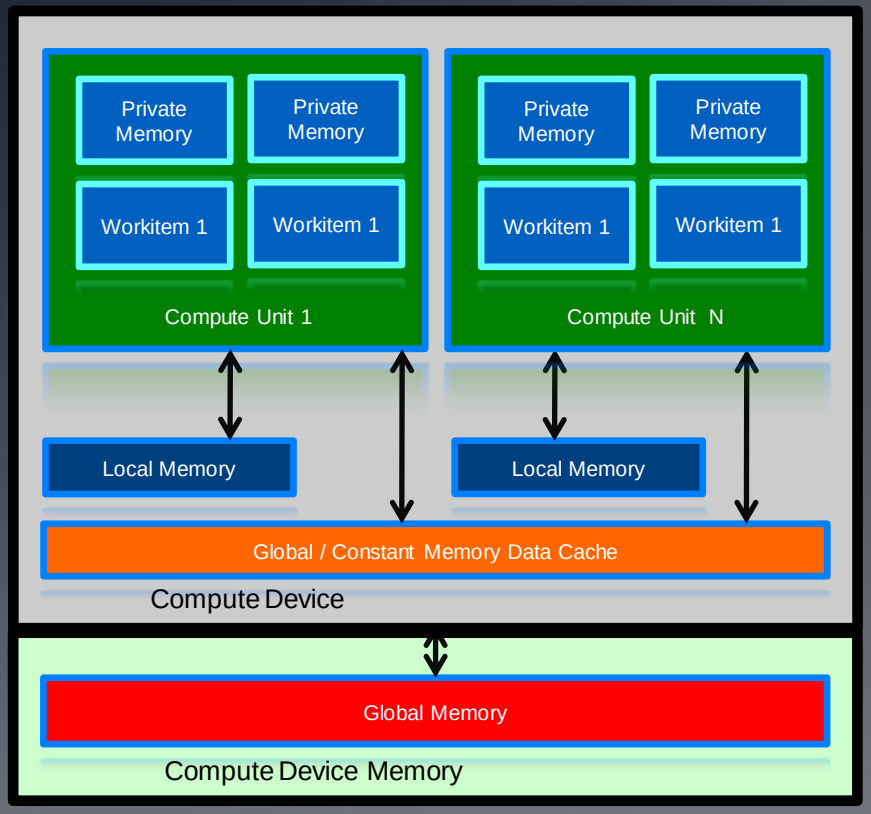


AMD GPU Memory Architecture



- Memory per compute unit
  - Local data store (on-chip)
  - Registers
- L1 cache (8KB for 5870) per compute unit
- L2 Cache shared between compute units (512KB for 5870)
- Fast path for only 32 bit operations
- Complete path for atomics and < 32bit operations

AMD Memory Model in OpenCL



- Subset of hardware memory exposed in OpenCL
- Local Data Share (LDS) exposed as local memory
  - Share data between items of a work group designed to increase performance
  - High Bandwidth access per SIMD Engine
- Private memory utilizes registers per work item
- Constant Memory
  - __constant tags utilize L1 cache.

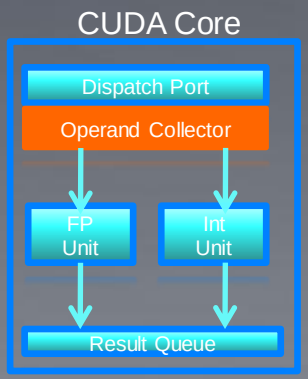
AMD Constant Memory Usage

- Constant Memory declarations for AMD GPUs only beneficial for following access patterns
  - **Direct-Addressing Patterns:** For non array constant values where the address is known initially
  - **Same Index Patterns:** When all work-items reference the same constant address
  - **Globally scoped constant arrays:** Arrays that are initialized, globally scoped can use the cache if less than 16KB
- Cases where each work item accesses different indices, are not cached and deliver the same performance as a global memory read

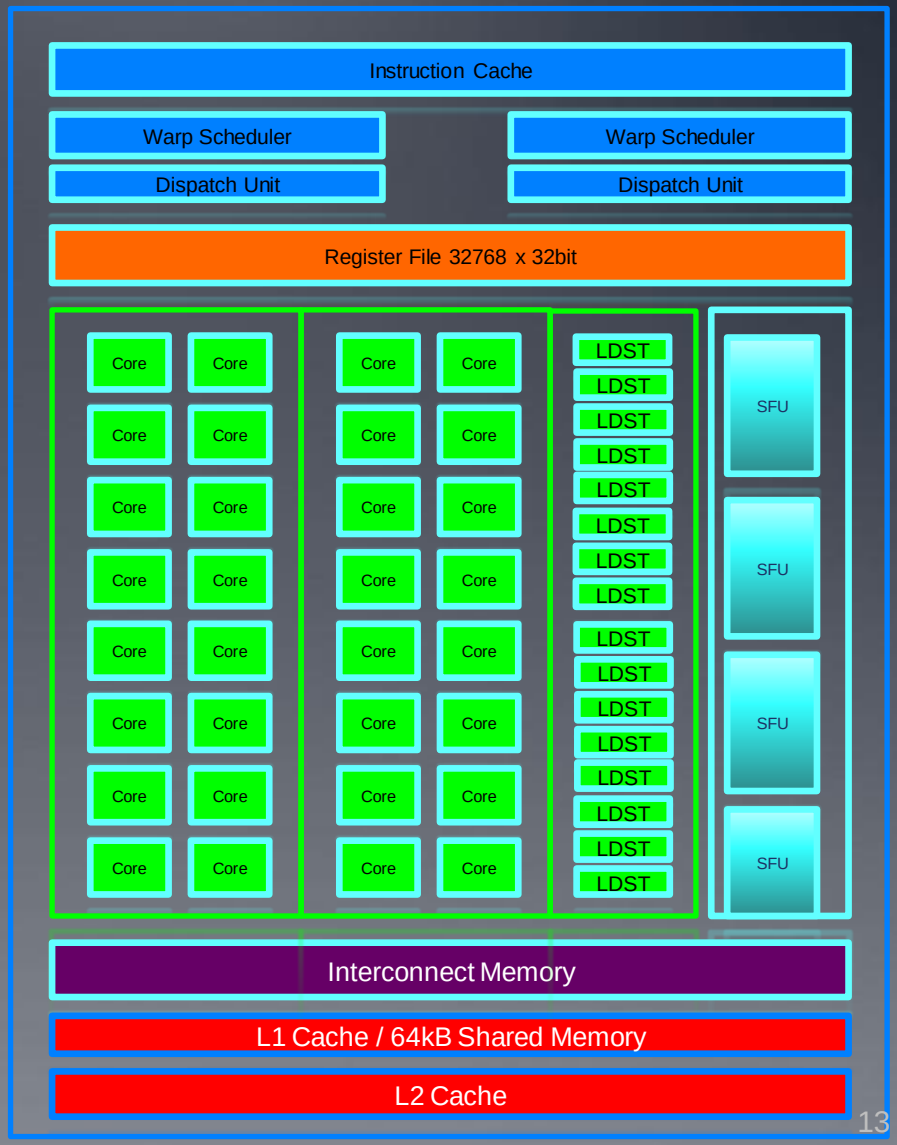
Source: AMD Accelerated Parallel Processing OpenCL Programming Guide

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

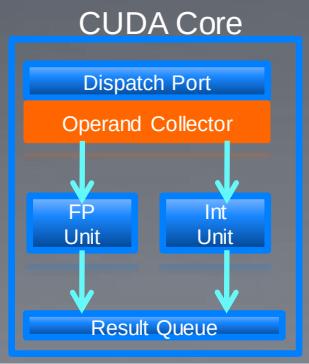


Source: NVIDIA's Next Generation CUDA Architecture Whitepaper

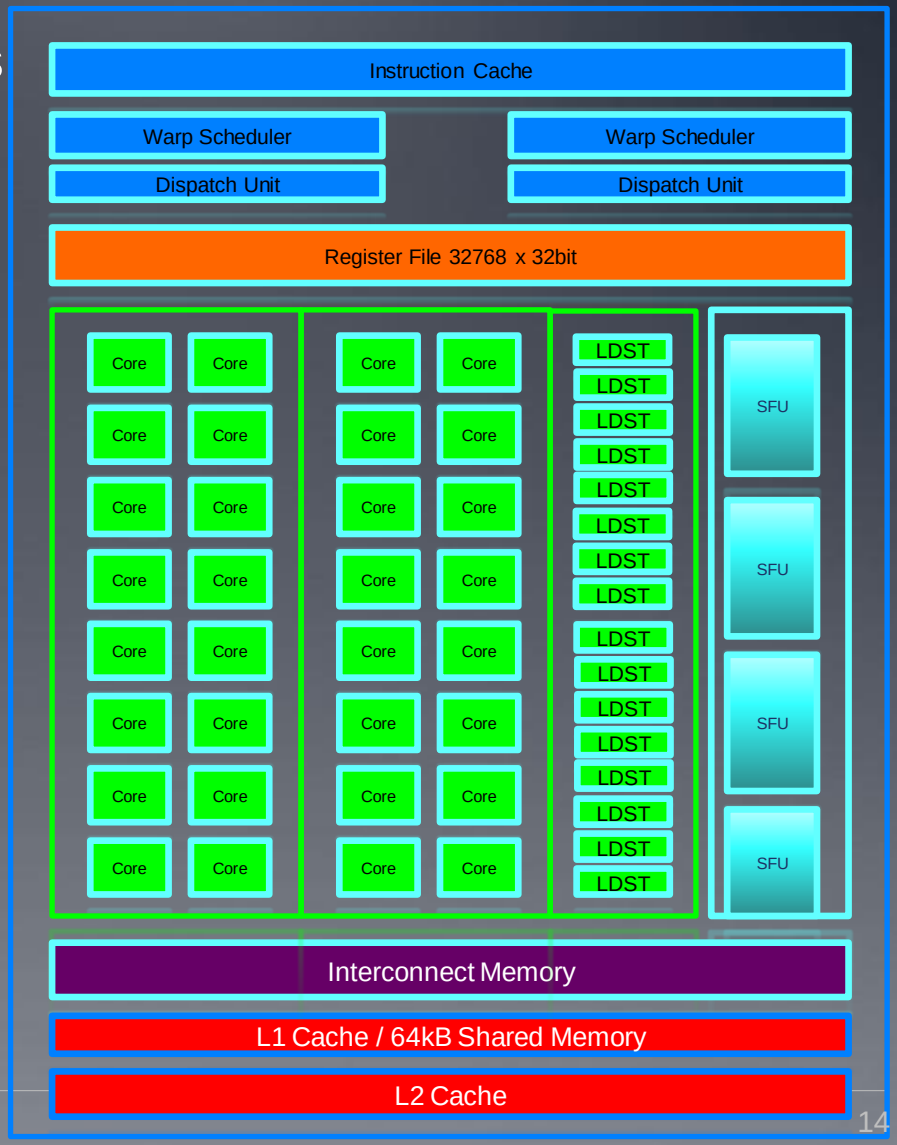


Nvidia GPUs – Fermi Architecture

- 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



Source: NVIDIA's Next Generation CUDA Compute Architecture Whitepaper

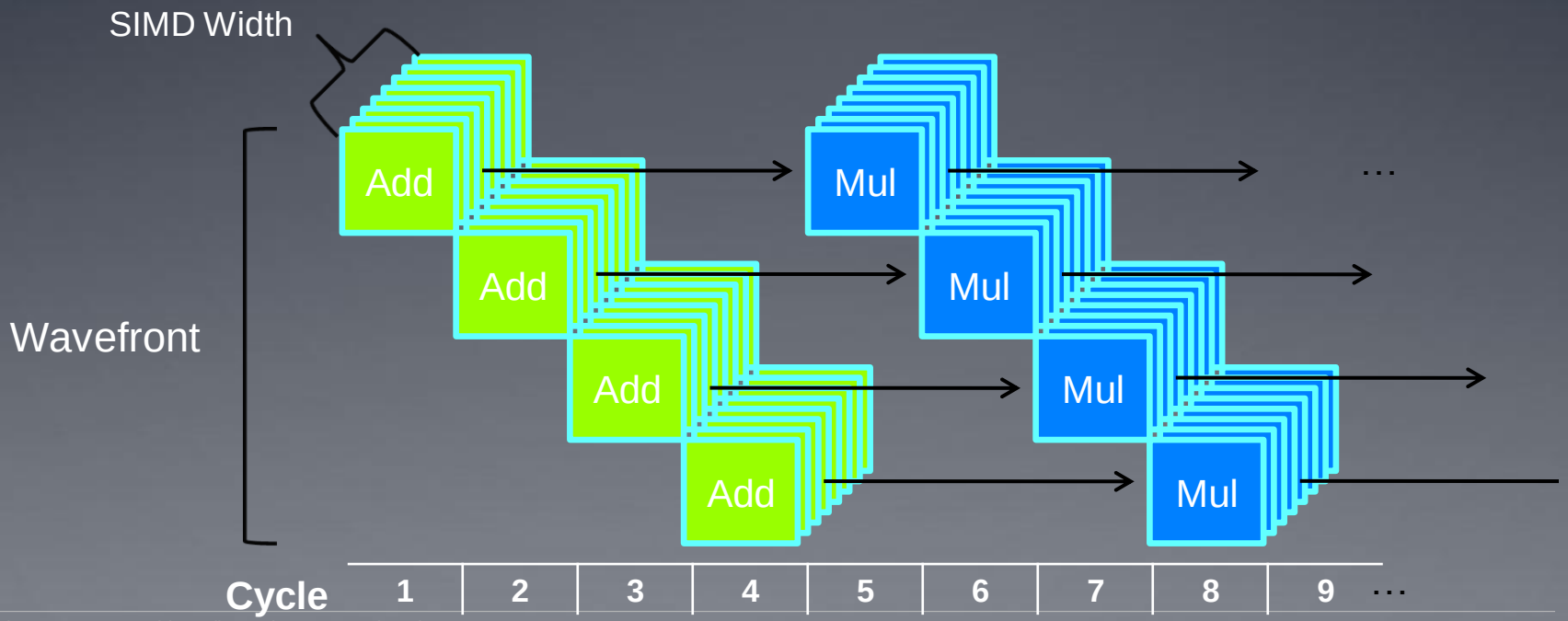


SIMT and SIMD

- 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

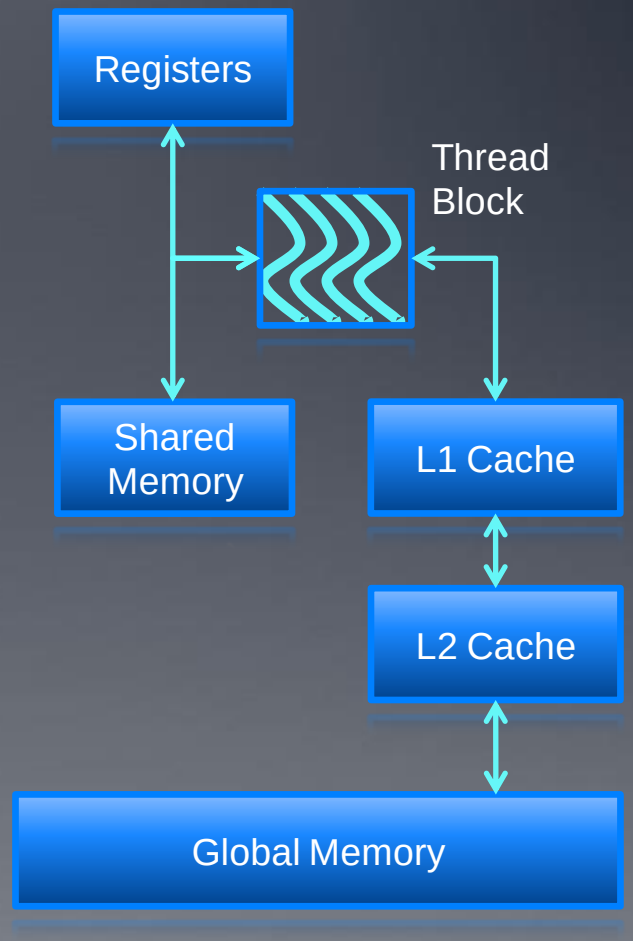
SIMT Execution Model

- 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

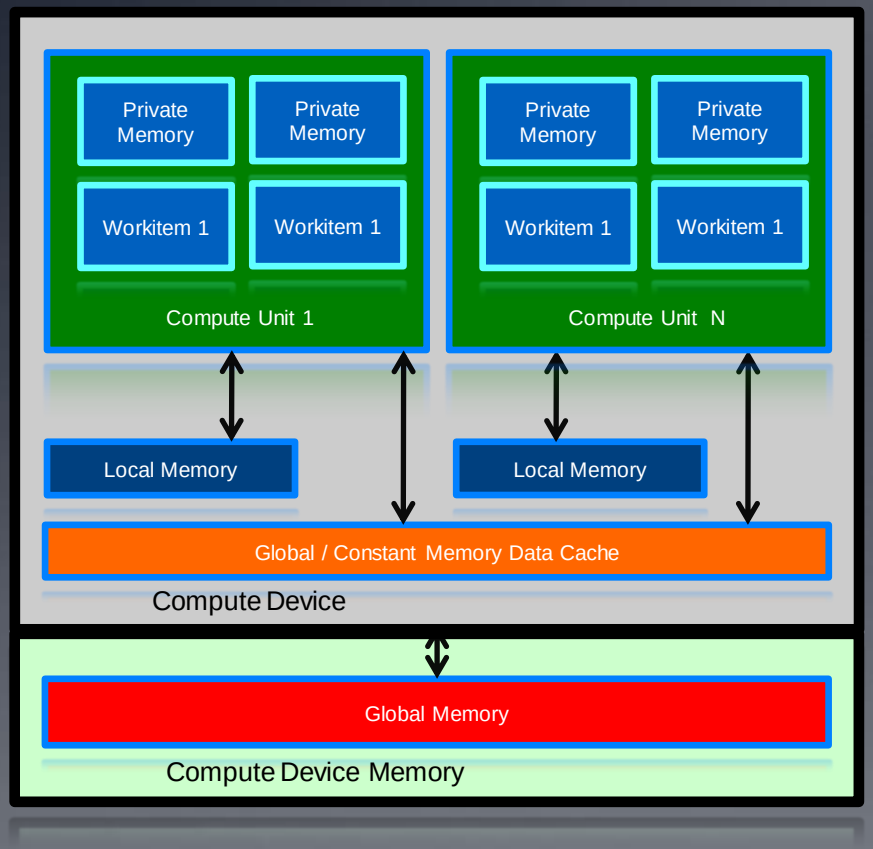


Nvidia Memory Hierarchy

- 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

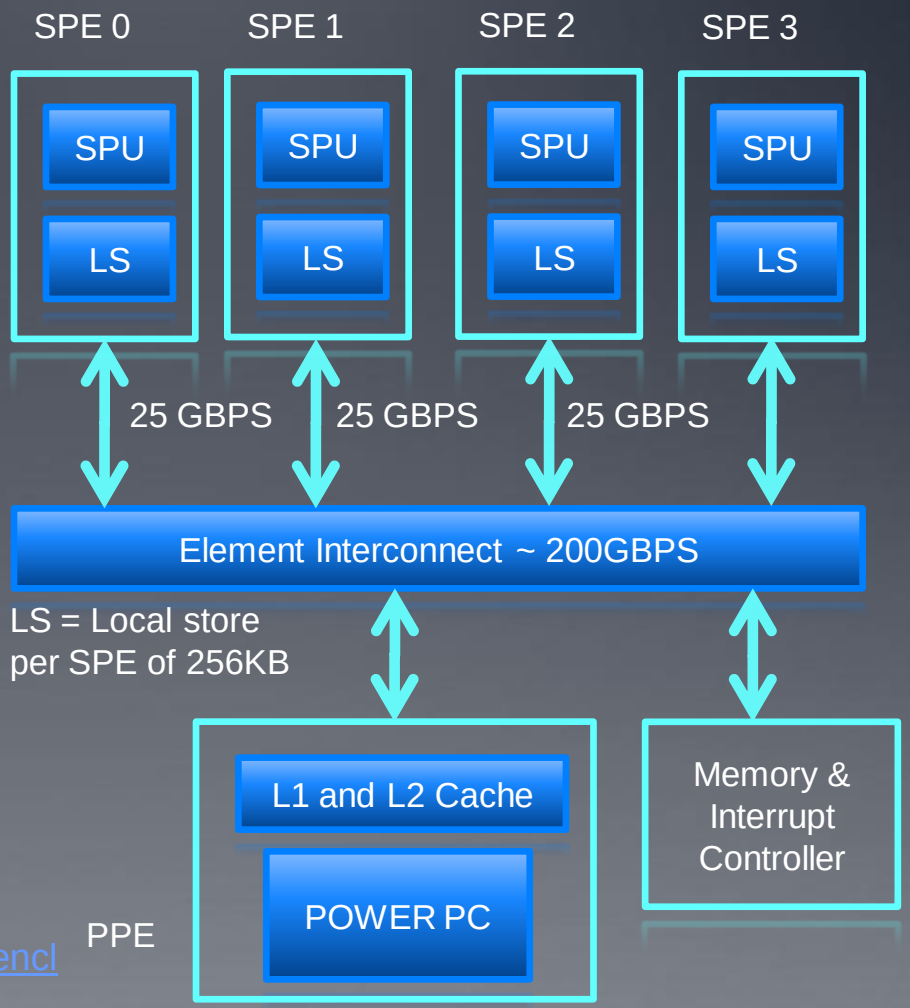


- Like AMD, a subset of hardware memory exposed in OpenCL
- Configurable shared memory is usable as local memory
  - Local memory used to share data between items of a work group at lower latency than global memory
- Private memory utilizes registers per work item

Cell Broadband Engine

- Developed by Sony, Toshiba, IBM
- Transitioned from embedded platforms into HPC via the Playstation 3
- OpenCL drivers available for Cell Bladecenter servers
- Consists of a Power Processing Element (PPE) and multiple Synergistic Processing Elements (SPE)
- Uses the IBM XL C for OpenCL compiler

Source: http://www.alphaworks.ibm.com/tech/opencl



Cell BE and OpenCL

- Cell Power/VMX CPU used as a CL_DEVICE_TYPE_CPU
- Cell SPU (CL_DEVICE_TYPE_ACCELERATOR)
  - No. of compute units on a SPU accelerator device is <=16
  - Local memory size <= 256KB
  - 256K of local storage divided among OpenCL kernel, 8KB global data cache, local, constant and private variables
- OpenCL accelerator devices, and OpenCL CPU device share a common memory bus
- Provides extensions like “Device Fission” and “Migrate Objects” to specify where an object resides (discussed in Lecture 10)
- No support for OpenCL images, sampler objects, atomics and byte addressable memory

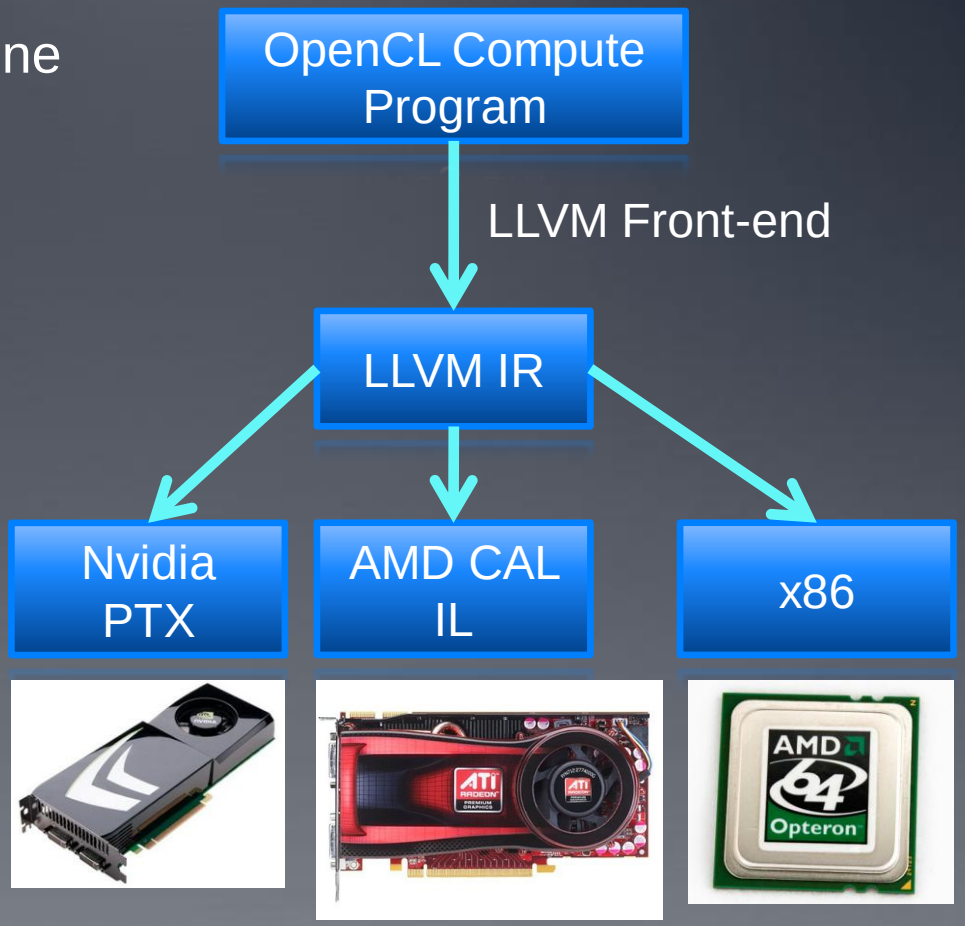
Source: http://www.alphaworks.ibm.com/tech/opencl

An Optimal GPGPU Kernel

- From the discussion on hardware we see that an ideal kernel for a GPU:
  - Has thousands of independent pieces of work
    - Uses all available compute units
    - Allows interleaving for latency hiding
  - Is amenable to instruction stream sharing
    - Maps to SIMD execution by preventing divergence between work items
  - Has high arithmetic intensity
    - Ratio of math operations to memory access is high
    - Not limited by memory bandwidth
- Note that these caveats apply to all GPUs

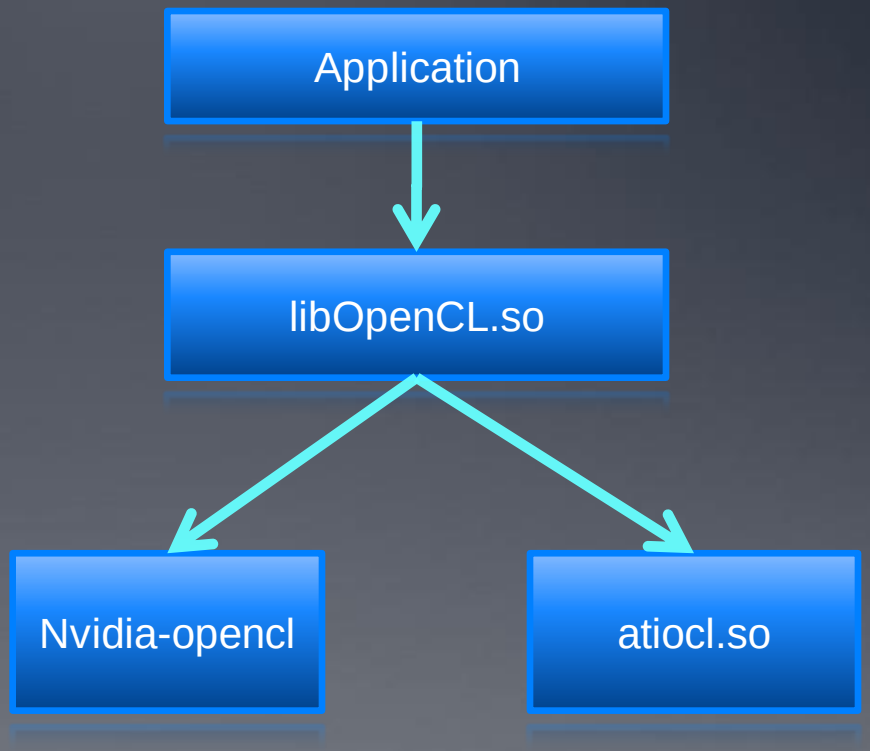
OpenCL Compilation System

- LLVM - Low Level Virtual Machine
- Kernels compiled to LLVM IR
- Open Source Compiler
  - Platform, OS independent
  - Multiple back ends
- http://llvm.org



Installable Client Driver

- ICD allows multiple implementations to co-exist
- Code only links to libOpenCL.so
- Application selects implementation at runtime
- Current GPU driver model does not easily allow multiple devices across manufacturers
- clGetPlatformIDs() and clGetPlatformInfo() examine the list of available implementations and select a suitable one



Summary

- We have examined different many-core platforms and how they map onto the OpenCL spec
  - An important take-away is that even though vendors have implemented the spec differently the underlying ideas for obtaining performance by a programmer remain consistent
- We have looked at the runtime compilation model for OpenCL to understand how programs and kernels for compute devices are created at runtime
  - We have looked at the ICD to understand how an OpenCL application can choose an implementation at runtime
- Next Lecture
  - Cover moving of data to a compute device and some simple but complete OpenCL examples