THOMAS MORE

GPU ARCHITECTURES

LID VAN ASSOCIATIE KU LEUVEN

INSTRUCTOR NOTES

- Lecture begins with a contrast of conventional multi-core CPU architectures with modern GPU architectures
- AMD, NVIDIA, and Cell architectures are presented
- Brief description of OpenCL Installable Client Driver and compilation flow

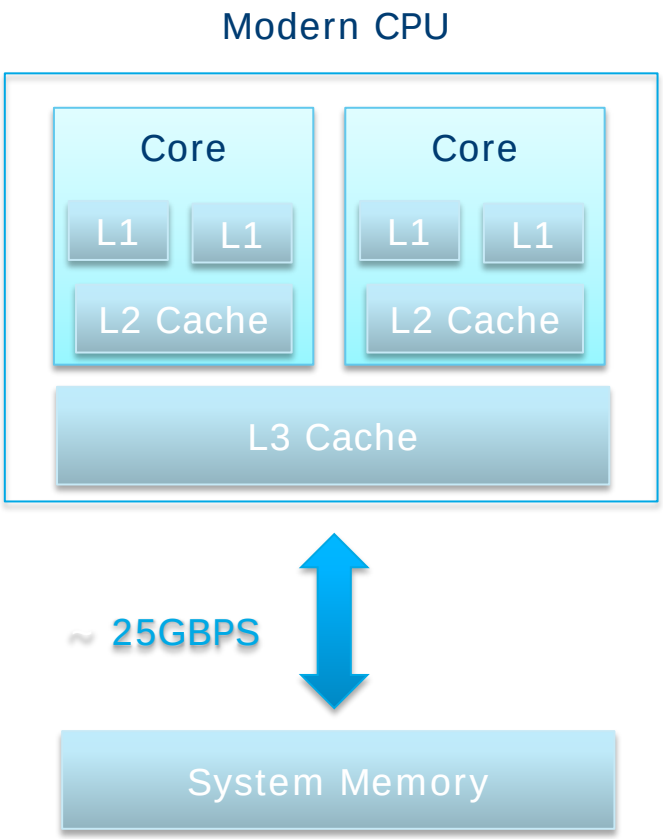
THOMAS MORE

TOPICS

- GPU architectures
  - AMD Sourthern Islands GPU Architecture
  - Nvidia Fermi GPU Architecture
  - Cell Broadband Engine
- OpenCL Specific Topics
  - OpenCL Compilation System
  - Installable Client Driver (ICD)

CONVENTIONAL CPU ARCHITECTURE

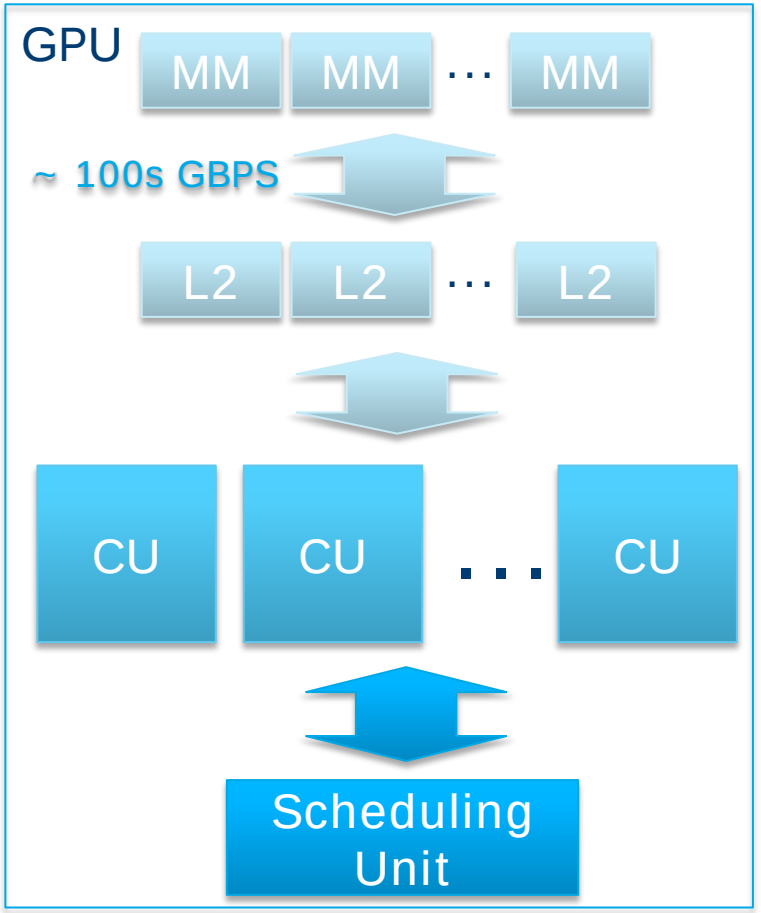
- CPUs are optimized to minimize the latency of a single thread
  - Can efficiently handle control flow intensive workloads
- Lots of space devoted to caching and control logic
  - Multi-level caches used to avoid latency
- Limited number of registers due to smaller number of active threads
- Control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



THOMAS MORE

MODERN GPGPU ARCHITECTURE

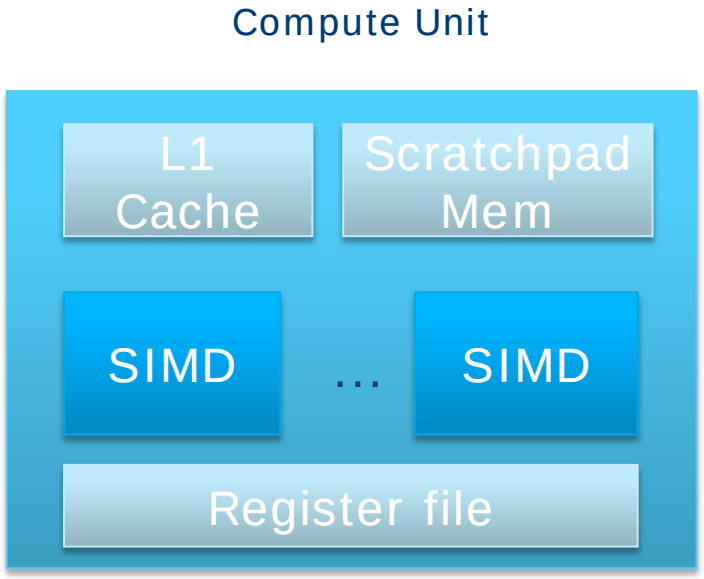
- Array of independent "cores" called *Compute Units*
- High bandwidth, banked L2 caches and main memory
  - Banks allow multiple accesses to occur in parallel
  - 100s of GB/s
- Memory and caches are generally non-coherent



THOMAS MORE

MODERN GPGPU ARCHITECTURE

- Compute units are based on SIMD hardware
  - Both AMD and NVIDIA have 16-element wide SIMDs
- Large register files are used for fast context switching
  - No saving/restoring state
  - Data is persistent for entire thread execution
- Both vendors have a combination of automatic L1 cache and a user-managed scratchpad
- Scratchpad is heavily banked and very high bandwidth (~terabytes/second)



THOMAS MORE

MODERN GPGPU ARCHITECTURE

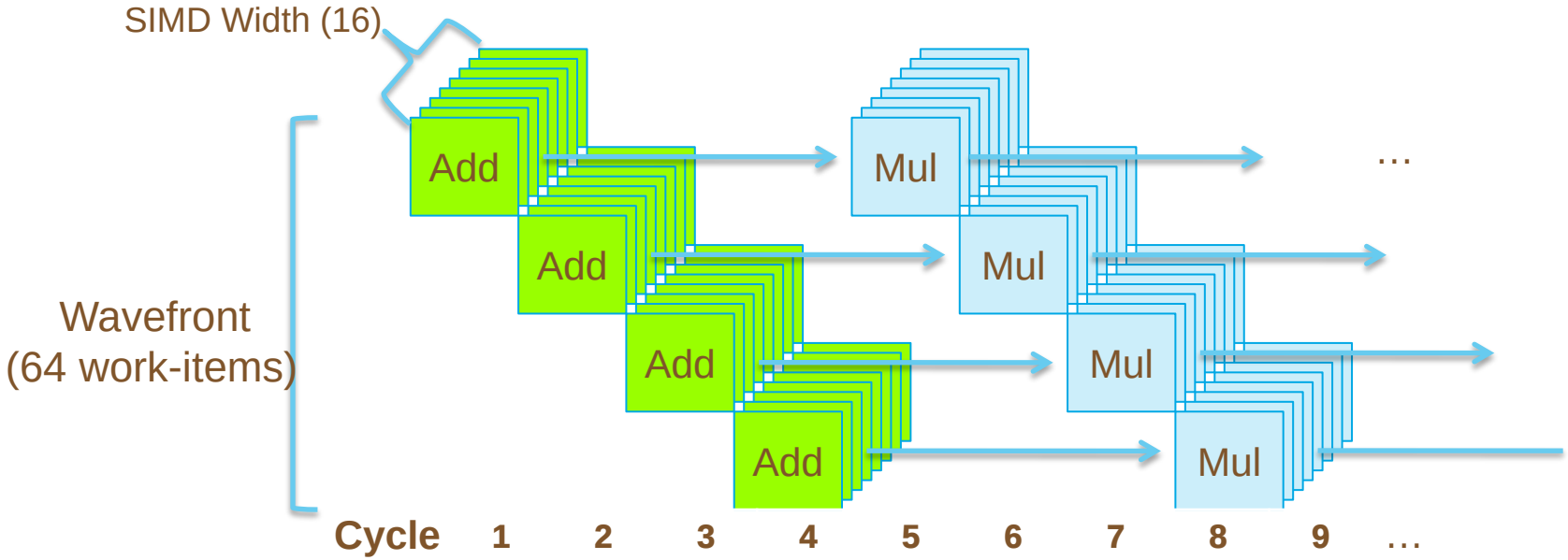
- Work-items are automatically grouped into hardware threads called “wavefronts” (AMD) or “warps” (NVIDIA)
  - Single instruction stream executed on SIMD hardware
  - 64 work-items in a wavefront, 32 in a warp
    - Instruction is issued multiple times on 16-lane SIMD unit
- Control flow is handled by masking SIMD lanes

SIMT AND SIMD

- NVIDIA coined “Single Instruction Multiple Threads” (SIMT) to denote multiple (software) threads sharing an instruction stream
- Work-items execute in lock-step on SIMD hardware
  - Multiple software threads are executed on a single hardware thread
  - Divergence between threads handled using predication
- Correctness is transparent to OpenCL model
- Performance is highly dependent on understanding work-items to SIMD mapping

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



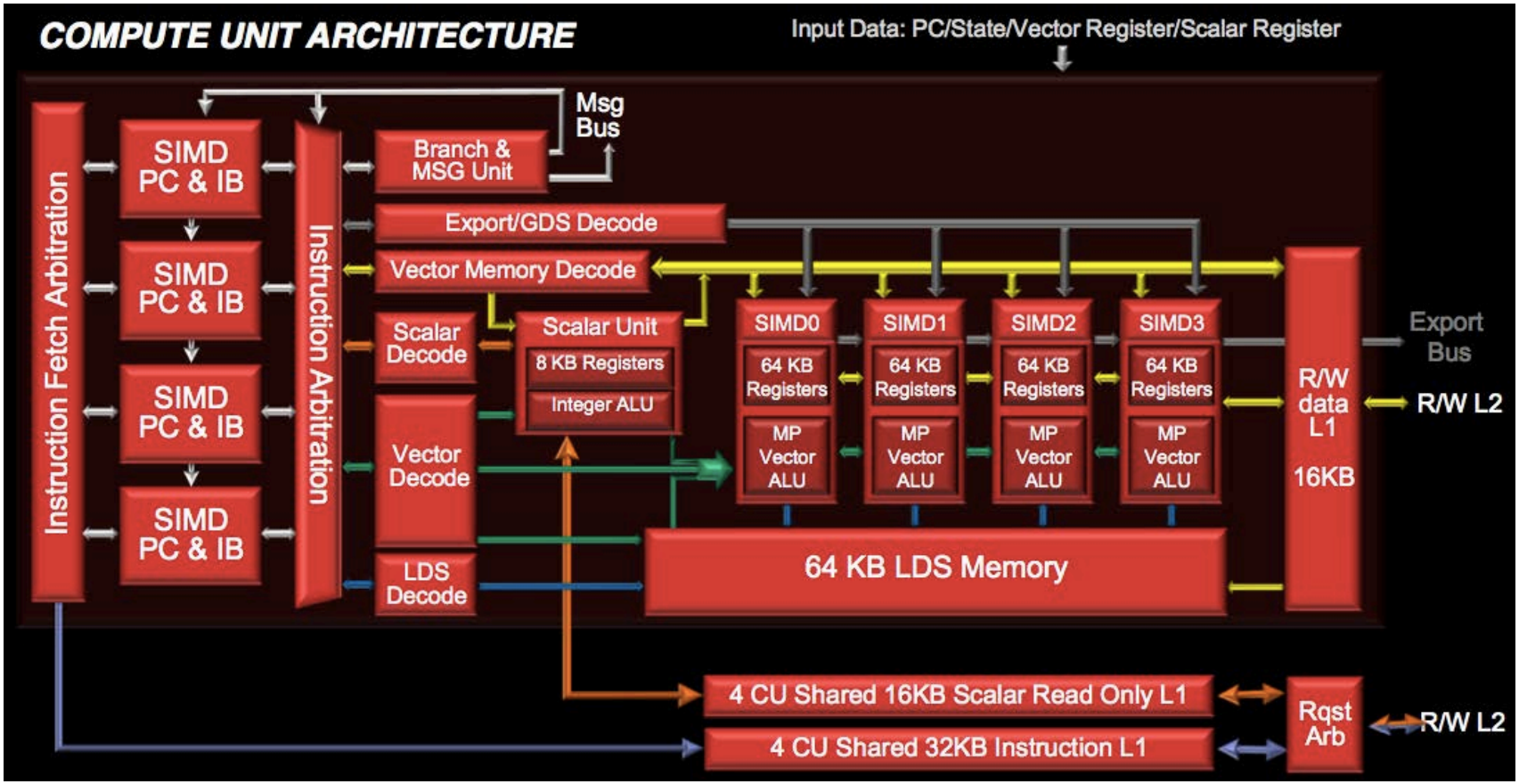
THOMAS MORE

AMD SOUTHERN ISLANDS

- 7000-series GPUs based on Graphics Core Next (GCN) architecture
- 4 SIMDs per compute unit
- 1 Scalar Unit to handle instructions common to wavefront
  - Loop iterators, constant variable accesses, branches
  - Has a single, integer-only ALU unit
  - Separate branch unit used for some conditional instructions
- Radeon HD7970
  - 32 compute units
  - < Max performance>

AMD Fusion developer Summit 2011
http://developer.amd.com/afds/assets/presentations/2620_final.pdf

AMD SOUTHERN ISLANDS



http://developer.amd.com/afds/assets/presentations/2620_final.pdf

THOMAS MORE

AMD SOUTHERN ISLANDS

- Wavefronts are associated with a SIMD unit and a subset of the vector registers
  - Up to 10 wavefronts can be associated with each SIMD
  - 4 SIMDs
  - 40 wavefronts can be active per compute unit
- All hardware units except for the SIMDs are shared by all wavefronts on a compute unit

AMD SOUTHERN ISLANDS

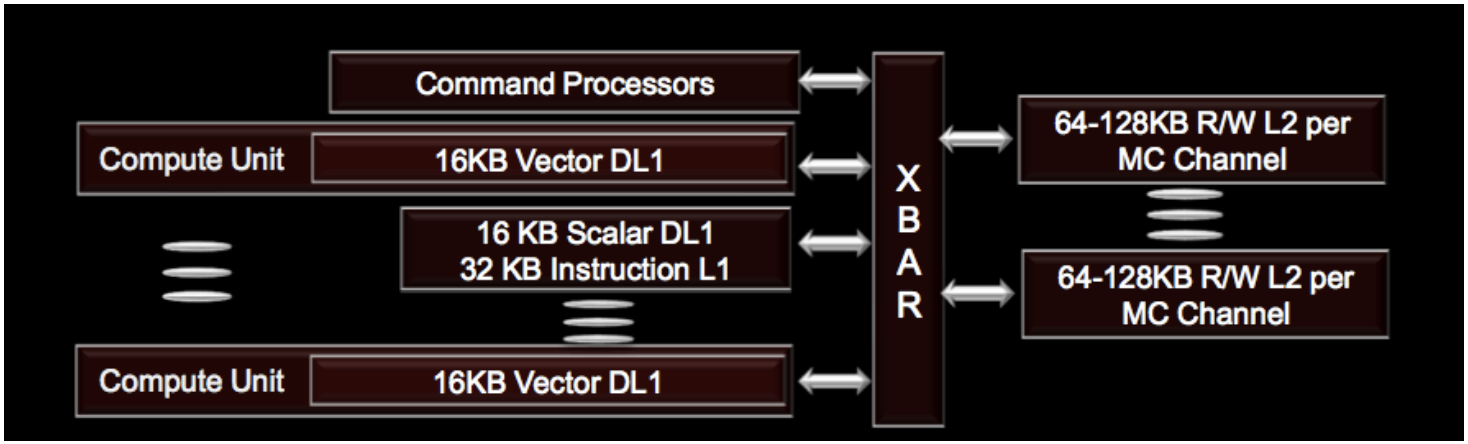
- Each cycle, wavefronts targeting one of the SIMDs are allowed to issue instructions
  - Every fourth cycle a wavefront will be active
- An instruction takes 4 cycles to enter the SIMD pipeline (4 subwavefronts per wavefront)
- Scalar unit and branch unit can take 1 instruction per cycle
- All hardware units can remain fully utilized with a simplified front-end using this round-robin technique

AMD SOUTHERN ISLANDS

- Up to 5 instructions can be issued per cycle
  - Only 1 per wavefront
  - Only 1 per instruction type (i.e., per hardware unit)
  - Need multiple instructions types present to fully utilize hardware units
- Instruction types
  - Vector Arithmetic Logic Unit (ALU)
  - Scalar ALU or Scalar Memory Read
  - Vector memory access (Read/Write/Atomic)
  - Branch/Message
  - Local Data Share (LDS)
  - Export or Global Data Share (GDS)
  - Internal (s_nop, s_sleep, s_waitcnt, s_barrier, s_setprio)

AMD SOUTHERN ISLANDS

- R/W L1 Caches
  - 16 KB/CU
  - Write through (dirty-byte mask)
  - 64B lines, 4 sets, 16 ways
- R/W L2 Caches
  - 64-128 KB/each
    - Up to 6 per GPU
  - Write back (dirty-byte mask)
  - 64B lines, 16 ways
    - Sets vary with size



http://developer.amd.com/afds/assets/presentations/2620_final.pdf

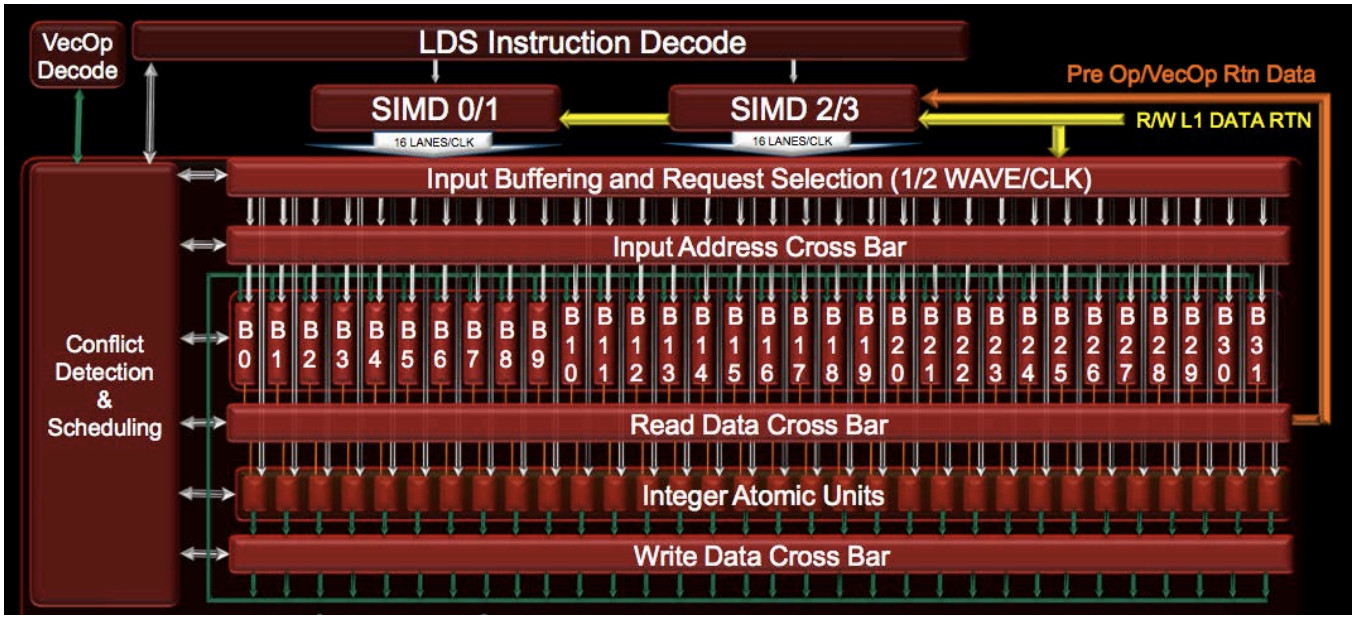
THOMAS MORE

AMD SOUTHERN ISLANDS

- Cache coherence supported at L2-level
  - GLC-bit allows L1 caches to be bypassed
  - Data is strided across L2s (all CUs access all caches)
  - Bypassing L1 allows coherent view at L2-level

AMD SOUTHERN ISLANDS

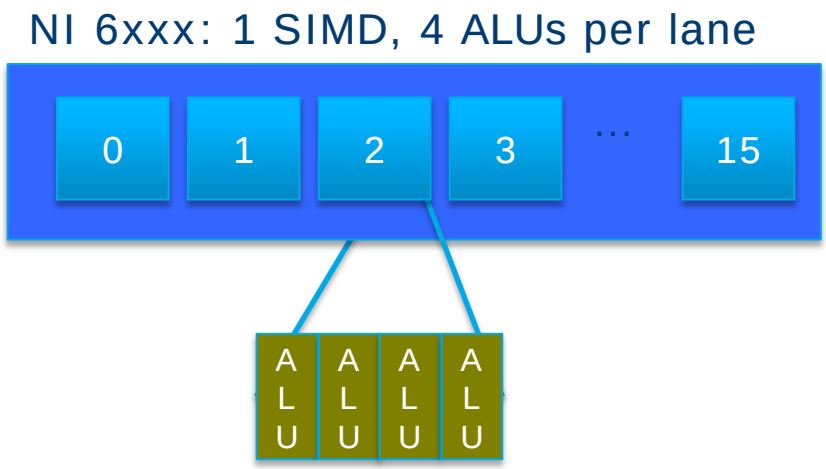
- Local Data Store (LDS, Local memory, Scratchpad)
  - 64KB per compute unit
  - 32 banks
  - Contains integer atomic units

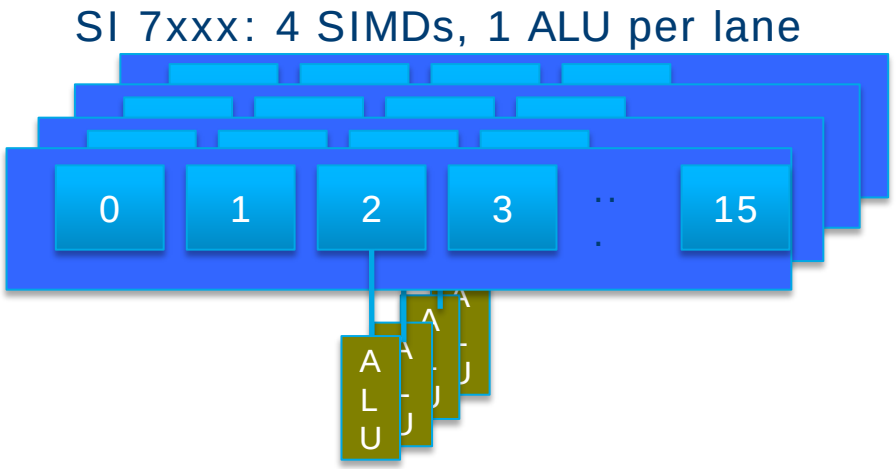


THOMAS MORE

COMPARISON TO NORTHERN ISLANDS

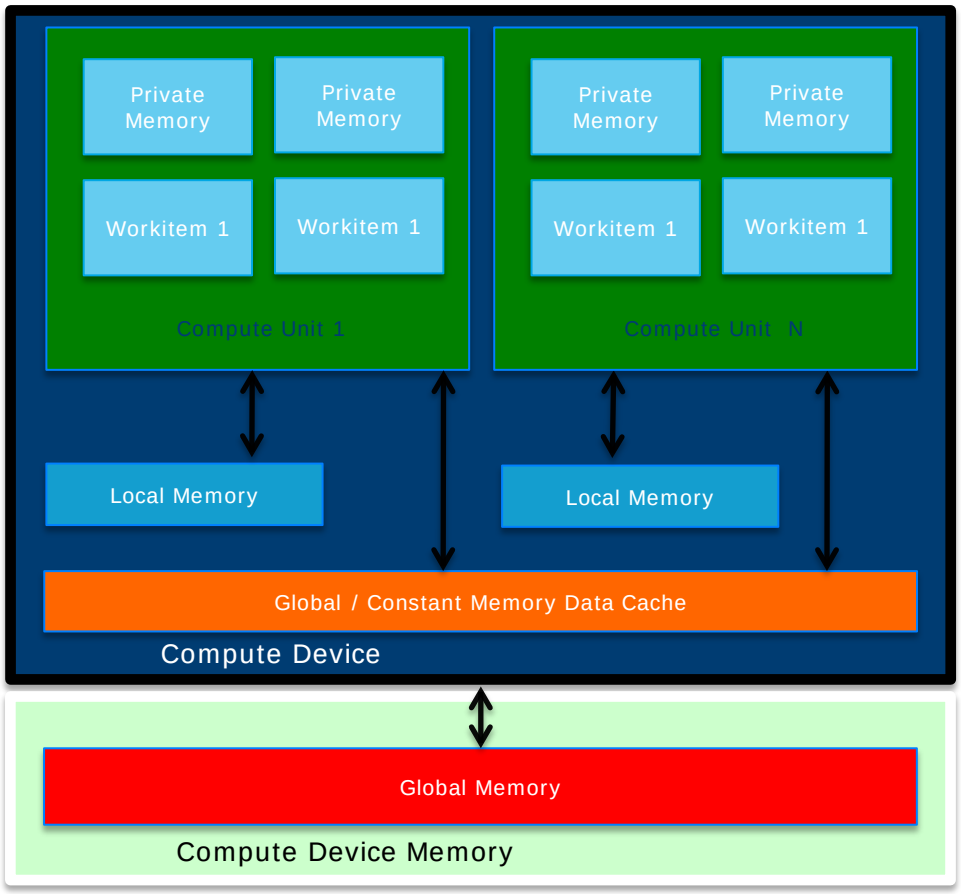
- Northern Islands (6xxx GPUs) SIMDs were based on VLIW
  - Each element of SIMD had 4 ALUs
  - Independent instructions (ILP) within single wavefront required to utilize hardware (VLIW instructions)
- Southern Islands
  - 4 SIMDs with one ALU in each element of SIMD
  - Multiple wavefronts and instruction mix required to utilize hardware
- Same number of ALUs in both NI and SI GPUs





THOMAS MORE

SI MEMORY MODEL IN OPENCL

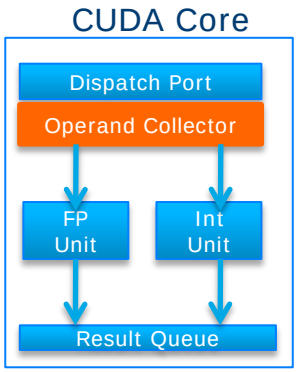


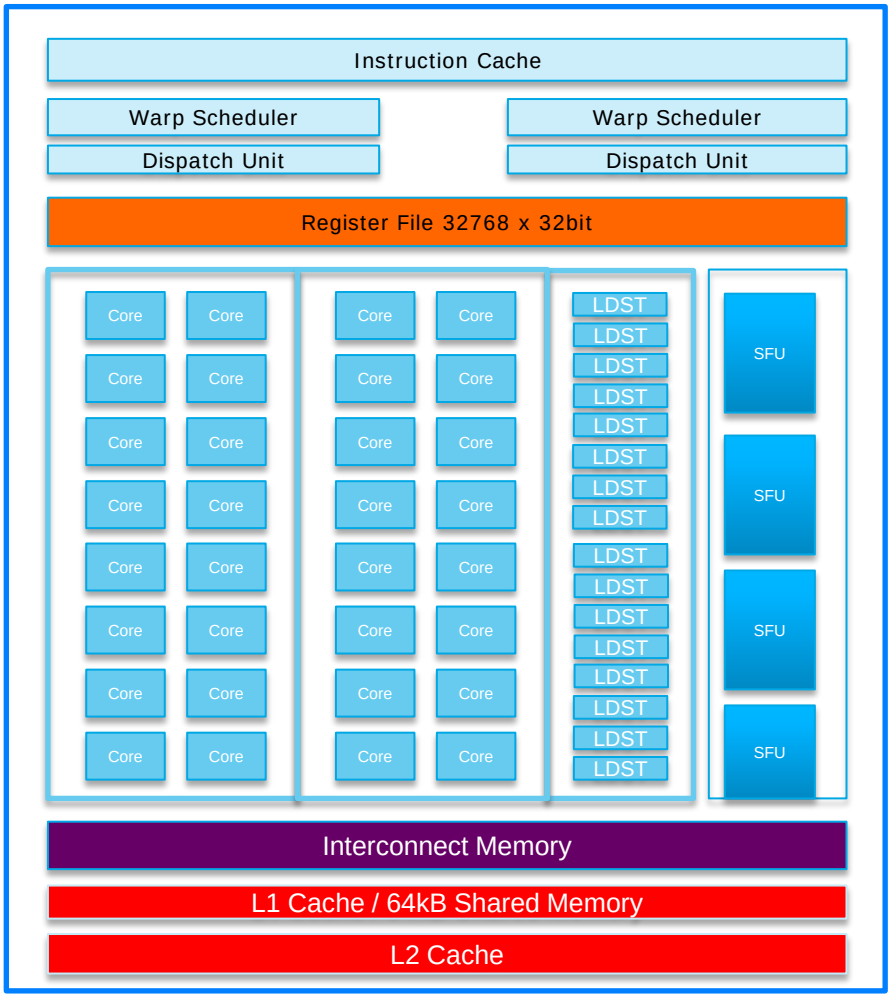
- Global Memory
  - Maps to cache hierarchy
  - GDDR5 video main memory
- Constant Memory
  - Maps to scalar unit reads
- Local Memory
  - Maps to the LDS
  - Shared data between work-items of a work group
  - High Bandwidth access from SIMDs
- Private memory
  - Maps to vector registers

THOMAS MORE

NVIDIA 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



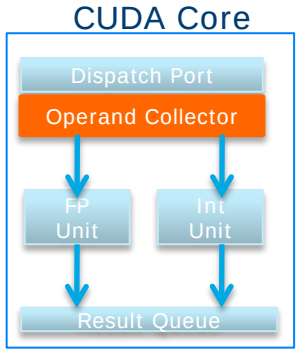


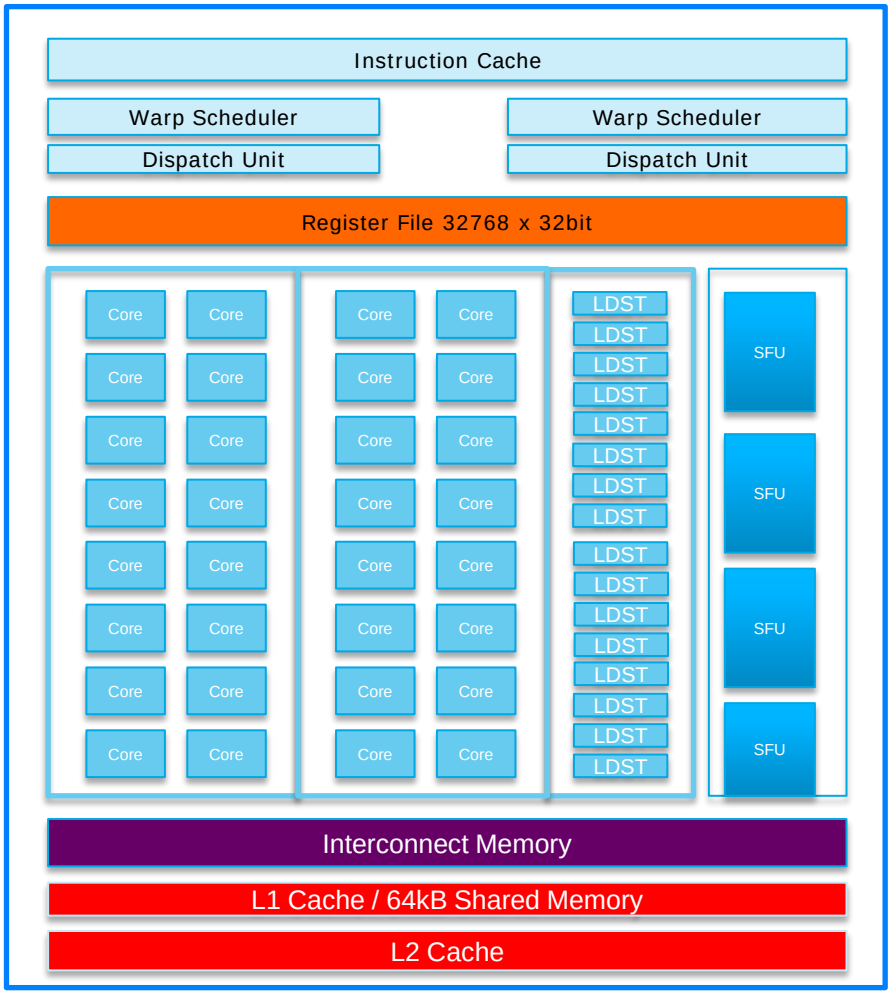
Source: NVIDIA's Next Generation CUDA Architecture Whitepaper

THOMAS MORE

NVIDIA 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



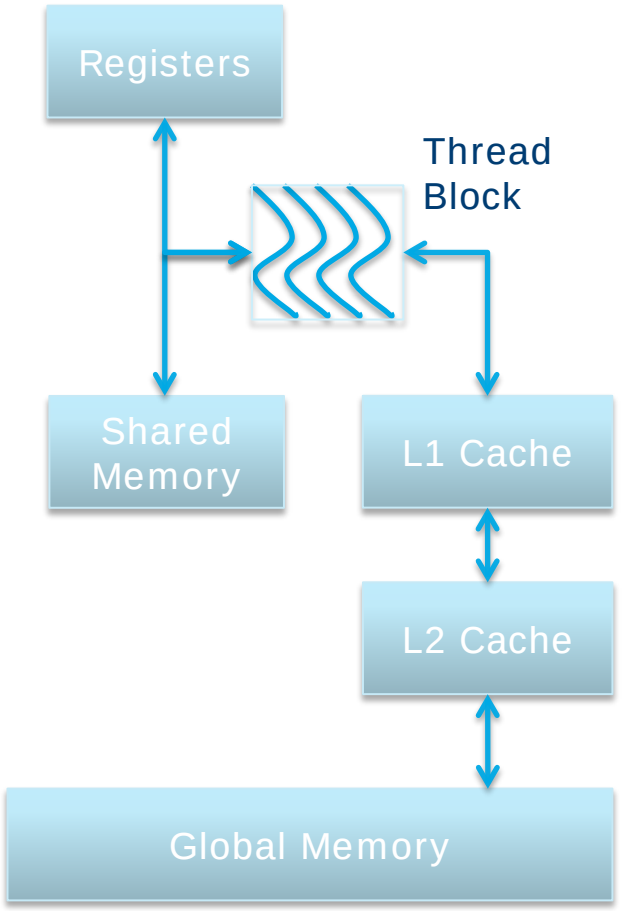


Source: NVIDIA's Next Generation CUDA Compute Architecture Whitepaper

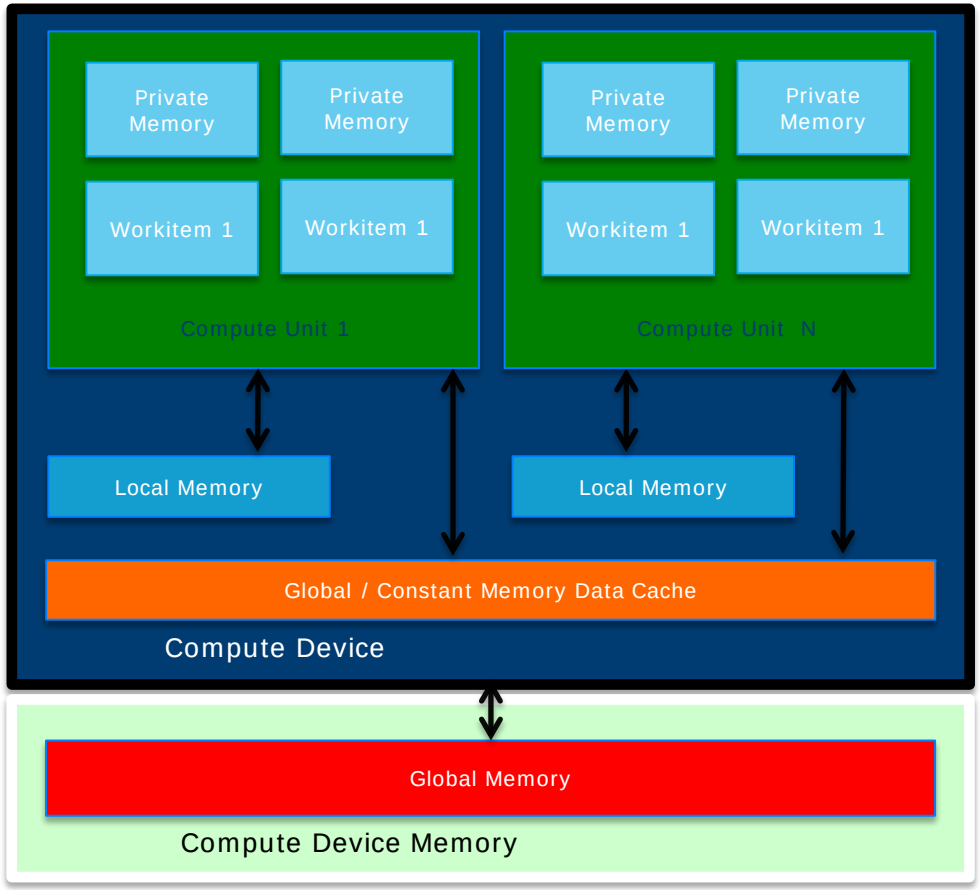
THOMAS MORE

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

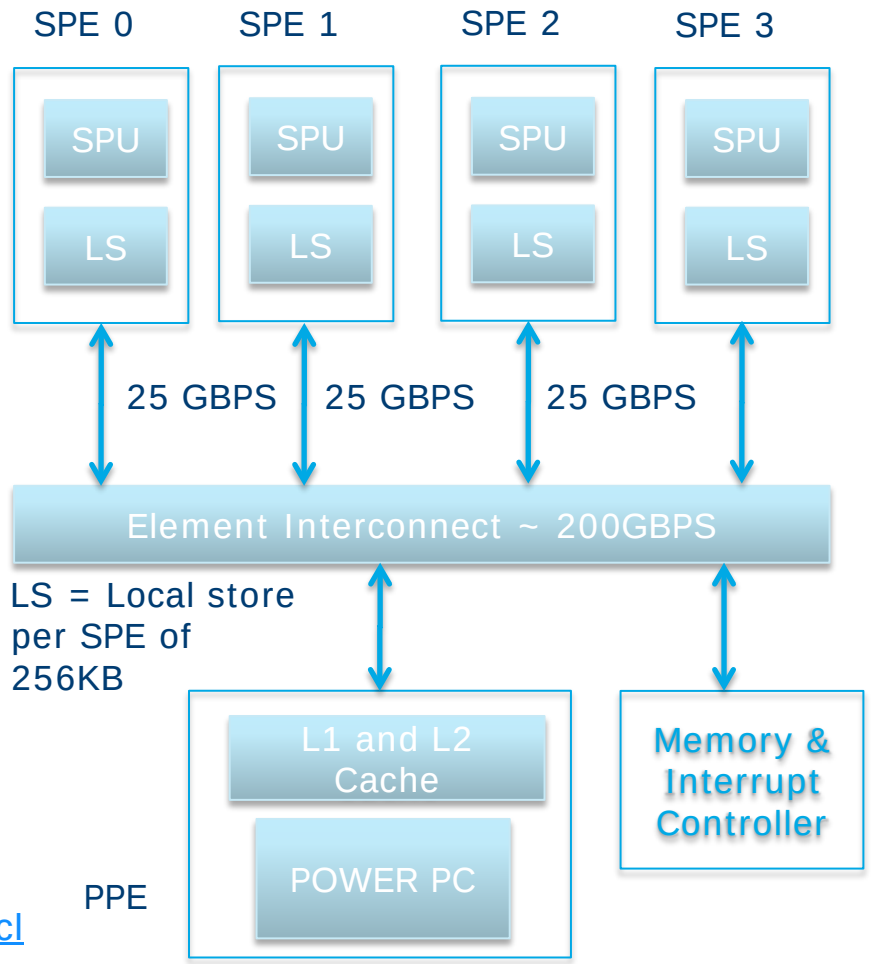


- Global memory
  - Maps to cache hierarchy
  - GDDR5 video main memory
- Constant memory
  - Maps to constant cache on GPU
- Local memory
  - Maps to “Shared memory” or LDS equivalent
  - Single memory unit configurable between L1 cache and local memory
- Private memory
  - Maps to vector registers

THOMAS MORE

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

THOMAS MORE

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

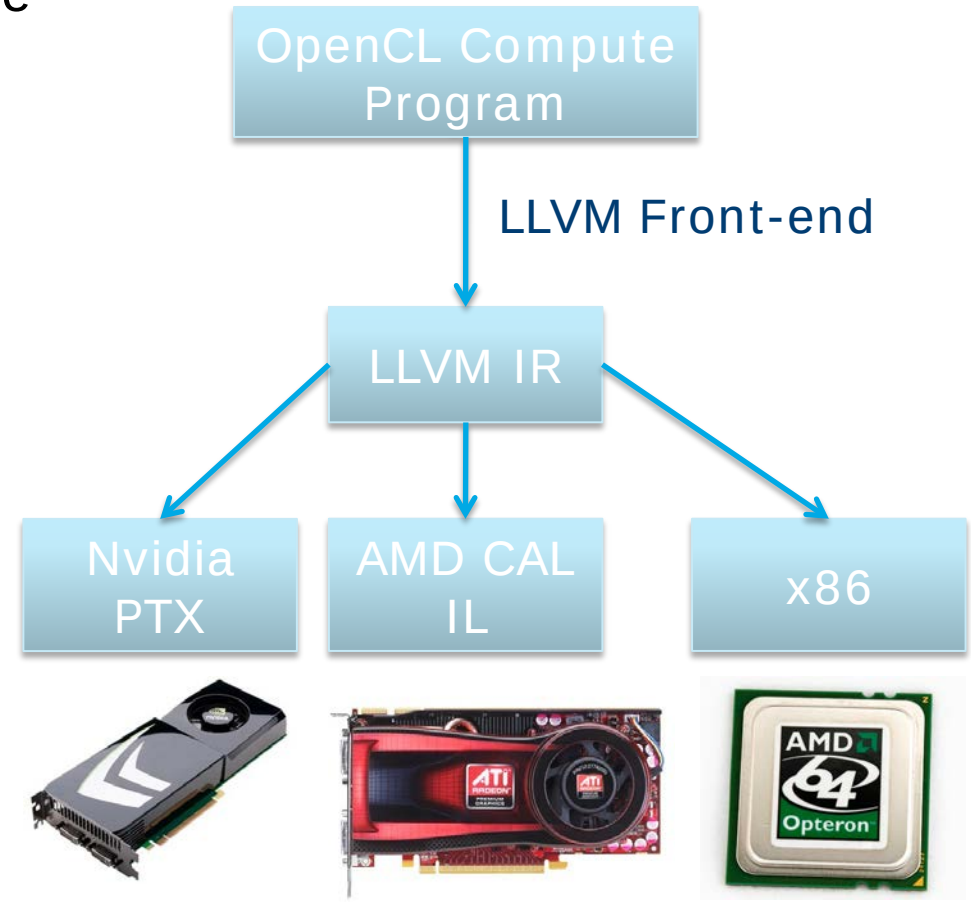
Source: http://www.alphaworks.ibm.com/tech/opencl

AN OPTIMAL GPGPU KERNEL

- An ideal kernel for a GPU
  - Has thousands of independent pieces of work
    - Uses all available compute units
    - Allows context switching to hide latency
  - 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

OPENCL COMPILATION SYSTEM

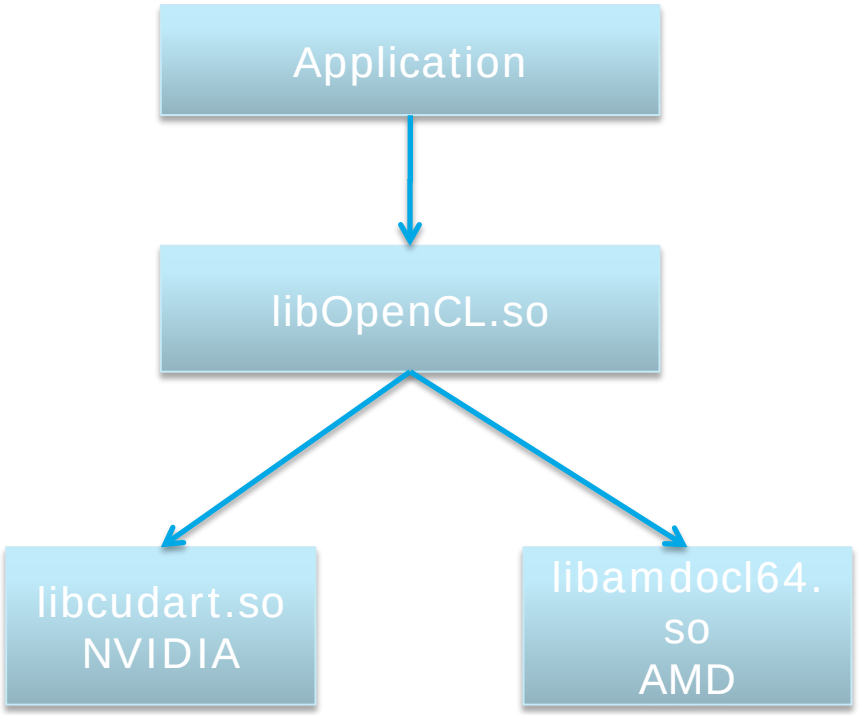
- LLVM - Low Level Virtual Machine
- Kernels compiled to LLVM IR
- Open Source Compiler
  - Platform, OS independent
  - Multiple back ends
- http://llvm.org



THOMAS MORE

INSTALLABLE CLIENT DRIVER

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



SUMMARY

- We have examined different GPU architectures 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 similar
- We have looked at the runtime compilation model for OpenCL to understand how programs and kernels for compute devices are created at runtime
- Next Lecture
  - Cover moving of data to a compute device and some simple but complete OpenCL examples