MSc Informatics Eng.

2011/12

A.J.Proença

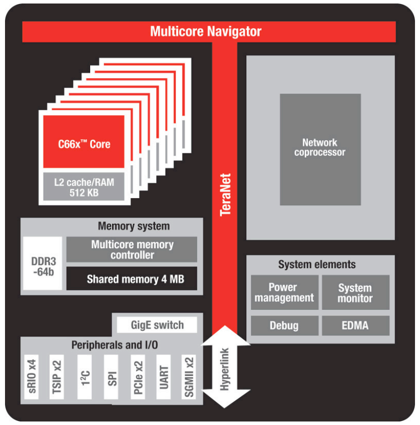
Data Parallelism 2 (*Cell BE, GPU, ...*)

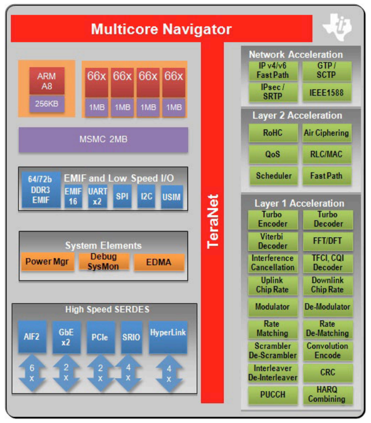
(most slides are borrowed)

Beyond Vector/SIMD architectures

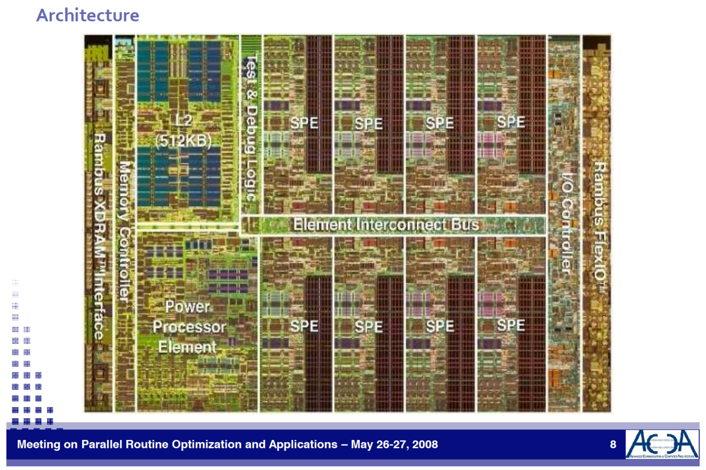
- Vector/SIMD-extended architectures are hybrid approaches
  - mix **scalar + vector** operation capabilities on a single device
  - **highly pipelined** approach to reduce memory access penalty
  - **tightly-closed access to shared memory**: lower latency
- Evolution of Vector/SIMD-extended architectures
  - CPU cores with wider vectors and/or SIMD cores:
    - DSP VLIW cores with vector capabilities: **Texas Instrument**
    - PPC cores coupled with SIMD cores: **Cell Broadband Engine**
    - ARM64 cores coupled with SIMD cores: project Denver (**NVidia**)
    - future x86 hybrid cores: **Intel, AMD ...**
  - devices with no scalar processor: **accelerator devices**
    - penalty on disjoint physical memories
    - based on the **GPU** architecture: SIMD—>SIMT to hide memory latency
    - ISA-free architecture, code compiled to silica: **FPGA**

Texas Instruments: Keystone DSP architecture





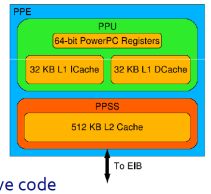
Cell Broadband Engine (1)



Meeting on Parallel Routine Optimization and Applications – May 26-27, 2008 8

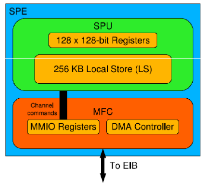
Cell Broadband Engine (2)

- Heterogeneous multicore processor
  - 1 x Power Processor Element (PPE)
    - 64-bit Power-architecture-compliant processor
    - Dual-issue, in-order execution, 2-way SMT processor
    - PowerPC Processor Unit (PPU)
      - 32 KB L1 IC, 32 KB L1 DC, VMX unit
    - PowerPC Processor Storage Subsystem (PPSS)
      - 512 KB L2 Cache
    - General-purpose processor to run OS and control-intensive code
    - Coordinates the tasks performed by the remaining cores



Cell Broadband Engine (3)

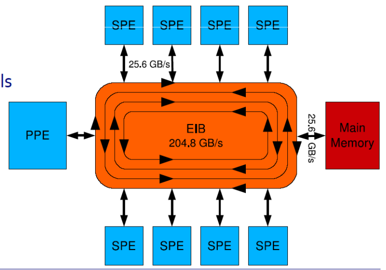
- Heterogeneous multicore processor
  - 8 x Synergistic Processing Element (SPE)
    - Dual-issue, in-order execution, 128-bit SIMD processors
    - Synergistic Processor Unit (SPU)
      - SIMD ISA (four different granularities)
      - 128 x 128-bit SIMD register file
      - 256 KB Local Storage (LS) for code/data
    - Memory Flow Controller (MFC)
      - Memory-mapped I/O registers (MMIO Registers)
      - DMA Controller: commands to transfer data in and out
    - Custom processors specifically designed for data-intensive code
    - Provide the main computing power of the Cell BE



Cell Broadband Engine (4)

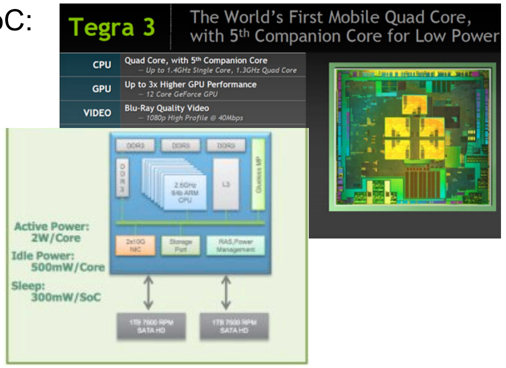
- Element Interconnect Bus (EIB)
  - Interconnects PPE, SPEs, and the memory and I/O interface controllers
    - 4 x 16 Byte-wide rings (2 clockwise and 2 counterclockwise)
  - Up to three simultaneous data transfers per ring
  - Shortest path algorithm for transfers
- Memory Interface Controller (MIC)
  - 2 x Rambus XDR I/O memory channels

    (accesses on each channel of 1-8, 16, 32, 64 or 128 Bytes)
- Cell BE Interface (BEI)
  - 2 x Rambus FlexIO I/O channels



NVidia: Project Denver

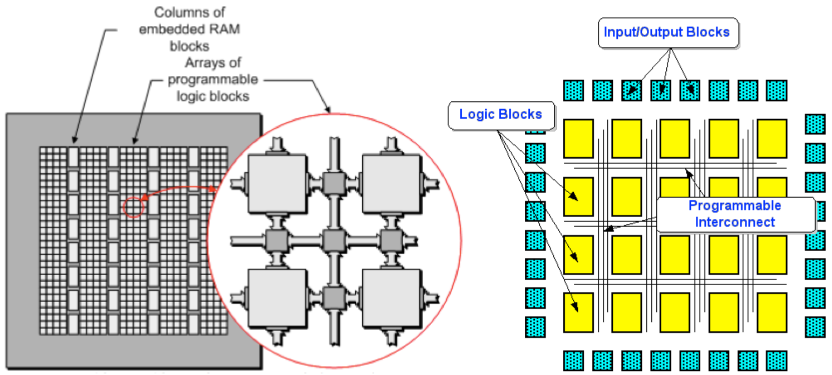
- Pick a successful SoC: Tegra 3
- Replace the 32-bit ARM Cortex 9 cores by 64-bit ARM cores
- Add some Fermi SIMT cores into the same chip



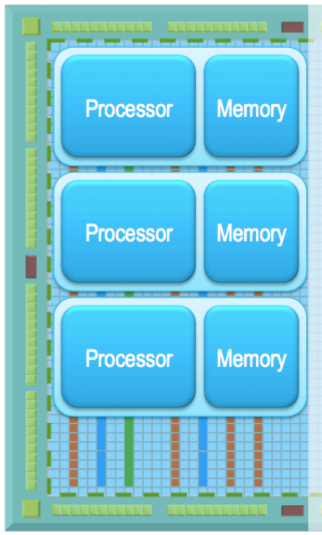
What is an FPGA

Field-Programmable Gate Arrays (FPGA)

A fabric with 1000s of simple configurable logic cells with LUTs, on-chip SRAM, configurable routing and I/O cells

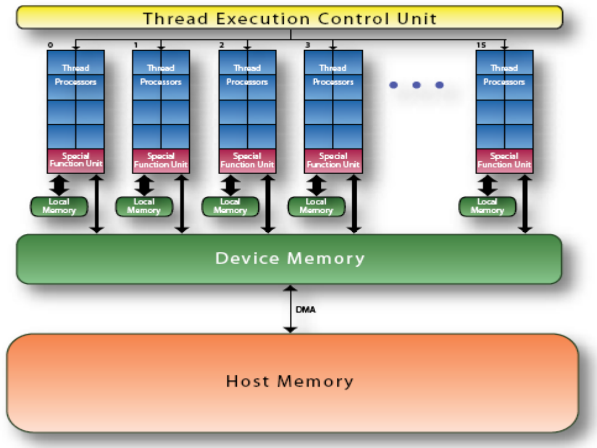

Columns of embedded RAM blocks
Arrays of programmable logic blocks
Input/Output Blocks
Logic Blocks
Programmable Interconnect

FPGA as a multiple configurable ISA



- Many coarse-grained processors
  - Different Implementation Options
    - Small soft scalar processor
    - or Larger vector processor
    - or Customized hardware pipeline
  - Each with local memory
- Each processor can exploit the fine grained parallelism of the FPGA to more efficiently implement it's "program"
- Possibly heterogeneous
  - Optimized for different tasks
- Customizable to suit the needs of a particular application

The GPU as a compute device: the G80



The CUDA programming model

- *Compute Unified Device Architecture*
- CUDA is a recent programming model, designed for
  - Manycore architectures
  - Wide SIMD parallelism
  - Scalability
- CUDA provides:
  - A thread abstraction to deal with SIMD
  - Synchr. & data sharing between small groups of threads
- CUDA programs are written in C with extensions
- OpenCL inspired by CUDA, but hw & sw vendor neutral
  - Programming model essentially identical

CUDA Devices and Threads

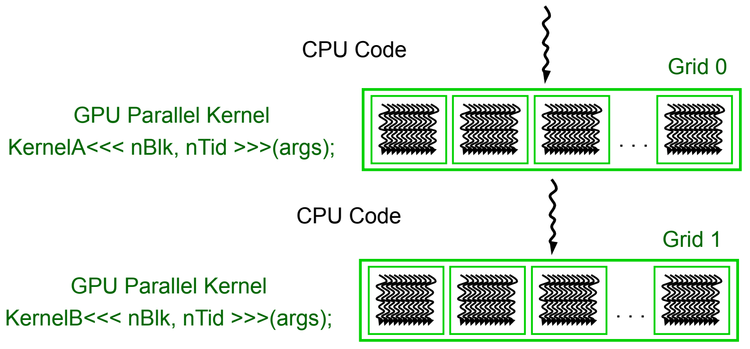
- A compute device
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads in parallel
  - Is typically a GPU but can also be another type of parallel processing device
- Data-parallel portions of an application are expressed as device kernels which run on many threads - **SIMT**
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - Very little creation overhead, requires LARGE register bank
  - GPU needs 1000s of threads for full efficiency
    - Multi-core CPU needs only a few

Terminology *(and in **NVidia**)*

- *Threads of SIMD instructions (**warps**)*
  - Each has its own PC (up to 48 per SIMD processor)
  - Thread scheduler uses scoreboard to dispatch
  - No data dependencies between threads!
  - Threads are organized into blocks & executed in groups of 32 threads (***thread block***)
    - Blocks are organized into a grid
- The thread block scheduler schedules blocks to SIMD processors (***Streaming Multiprocessors***)
- Within each SIMD processor:
  - 32 SIMD lanes (***thread processors***)
  - Wide and shallow compared to vector processors

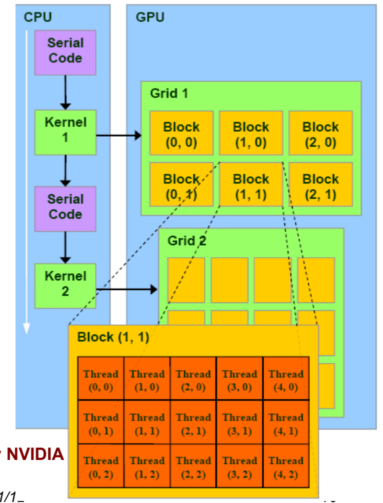
CUDA basic model: Single-Program Multiple-Data (SPMD)

- CUDA integrated CPU + GPU application C program
  - Serial C code executes on CPU
  - Parallel Kernel C code executes on GPU thread blocks



Programming Model: SPMD + SIMT/SIMD

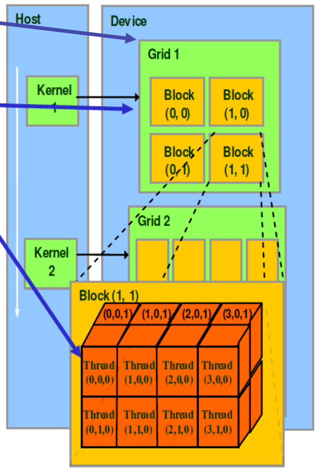
- Hierarchy
  - Device => Grids
  - Grid => Blocks
  - Block => Warps
  - Warp => Threads
- Single kernel runs on multiple blocks (SPMD)
- Threads within a warp are executed in a lock-step way called single-instruction multiple-thread (SIMT)
- Single instruction are executed on multiple threads (SIMD)
  - Warp size defines SIMD granularity (32 threads)
- Synchronization within a block using shared memory



Courtesy NVIDIA

The Computational Grid: Block IDs and Thread IDs

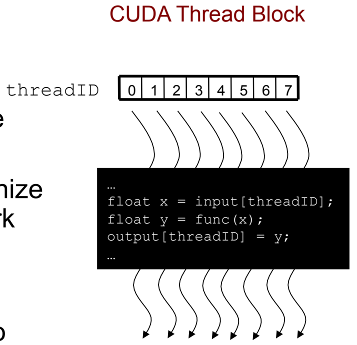
- A kernel runs on a computational grid of thread blocks
  - Threads share global memory
- Each thread uses IDs to decide what data to work on
  - Block ID: 1D or 2D
  - Thread ID: 1D, 2D, or 3D
- A thread block is a batch of threads that can cooperate by:
  - Sync their execution w/ barrier
  - Efficiently sharing data through a low latency shared memory
  - Two threads from two different blocks cannot cooperate



CUDA Thread Block

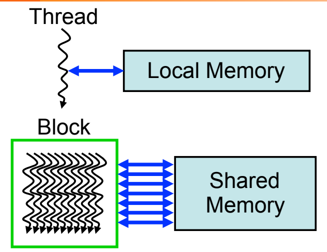
- Programmer declares (Thread) Block:
  - Block size 1 to **512** concurrent threads
  - Block shape 1D, 2D, or 3D
  - Block dimensions in threads
- All threads in a Block execute the same thread program
- Threads share data and synchronize while doing their share of the work
- Threads have thread id numbers within Block
- Thread program uses thread id to select work and address shared data

CUDA Thread Block

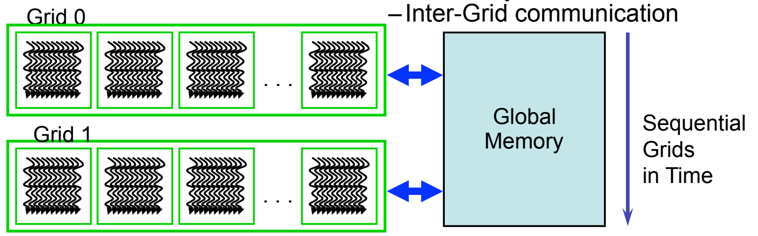


```
…
float x = input[threadID];
float y = func(x);
output[threadID] = y;
…
```

Parallel Memory Sharing

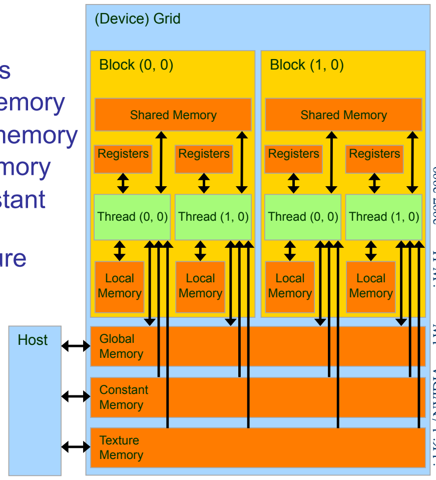


- Local Memory: **per-thread**
  - Private per thread
  - Auto variables, register spill
- Shared Memory: **per-block**
  - Shared by threads of the same block
  - Inter-thread communication
- Global Memory: **per-application**
  - Shared by all threads
  - Inter-Grid communication



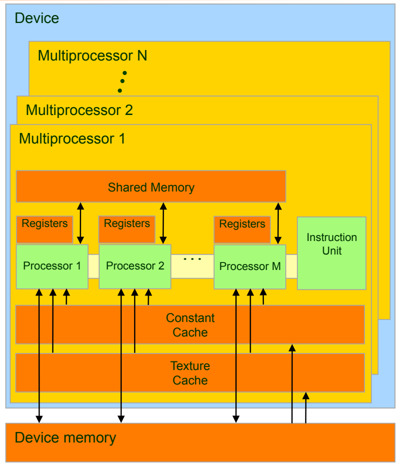
CUDA Memory Model Overview

- Each thread can:
  - R/W per-thread registers
  - R/W per-thread local memory
  - R/W per-block shared memory
  - R/W per-grid global memory
  - Read only per-grid constant memory
  - Read only per-grid texture memory
- The host can R/W global, constant, and texture memories



Hardware Implementation: Memory Architecture

- Device memory (DRAM)
  - Slow (2~300 cycles)
  - Local, global, constant, and texture memory
- On-chip memory
  - Fast (1 cycle)
  - Registers, shared memory, constant/texture cache



Courtesy NVIDIA

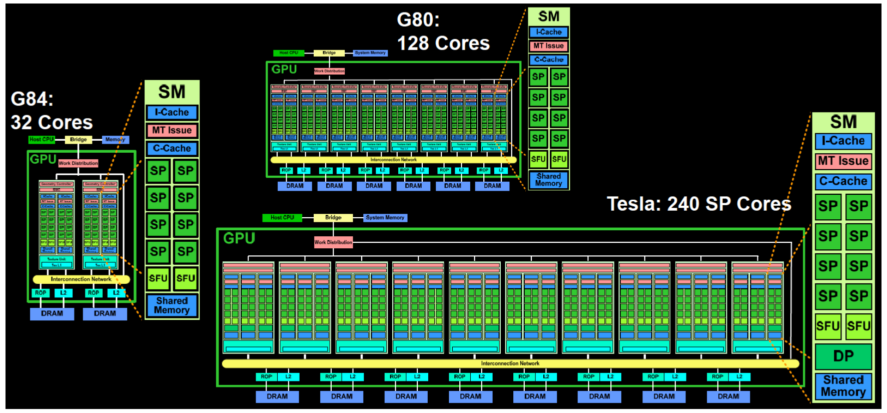
NVIDIA GPU Memory Structures

- Each SIMD Lane has private section of **off-chip DRAM**
  - "Private memory" *(Local Memory)*
  - Contains stack frame, spilling registers, and private variables
- Each multithreaded SIMD processor also has local memory *(Shared Memory)*
  - Shared by SIMD lanes / threads within a block
- Memory shared by SIMD processors is GPU Memory *(Global Memory)*
  - Host can read and write GPU memory

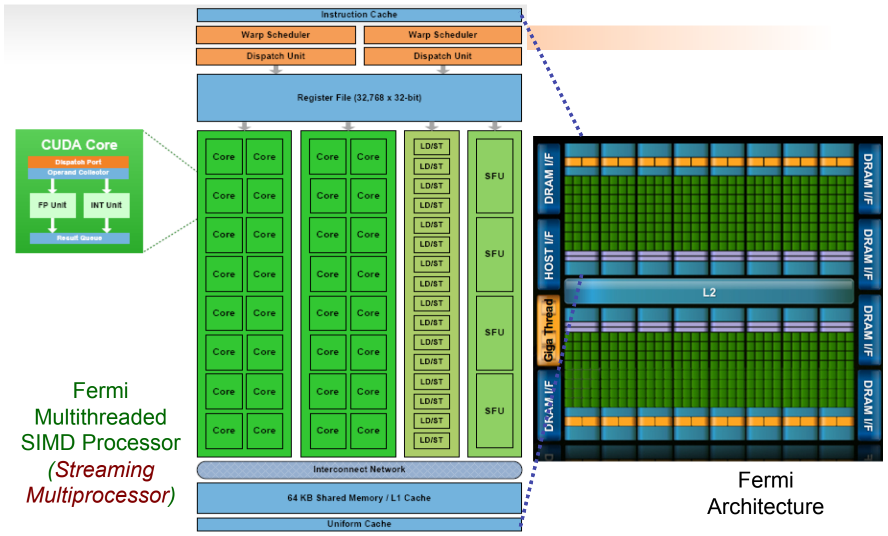
Families in NVidia GPU

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double-Precision Floating Point	None	30 FMA ops per clock	256 FMA ops per clock
Single-Precision Floating Point	128 MADD ops per clock	240 MADD ops per clock	512 FMA ops per clock
Warp Schedulers per Streaming Multiprocessor (SM)	1	1	2
Special Function Units per SM	2	2	4
Shared Memory per SM	16KB	16KB	Configurable 48KB or 16KB
L1 Cache per SM	None	None	Configurable 16KB or 48KB
L2 Cache	None	None	768KB
ECC Memory Protection	No	No	Yes
Concurrent Kernels	No	No	Up to 16

NVidia GPU structure & scalability



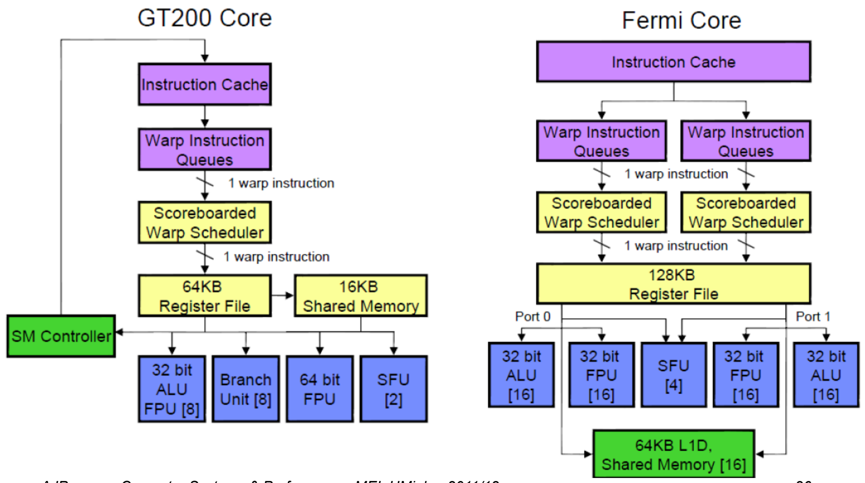
The NVidia Fermi architecture



Fermi Multithreaded SIMD Processor *(Streaming Multiprocessor)*

Fermi Architecture

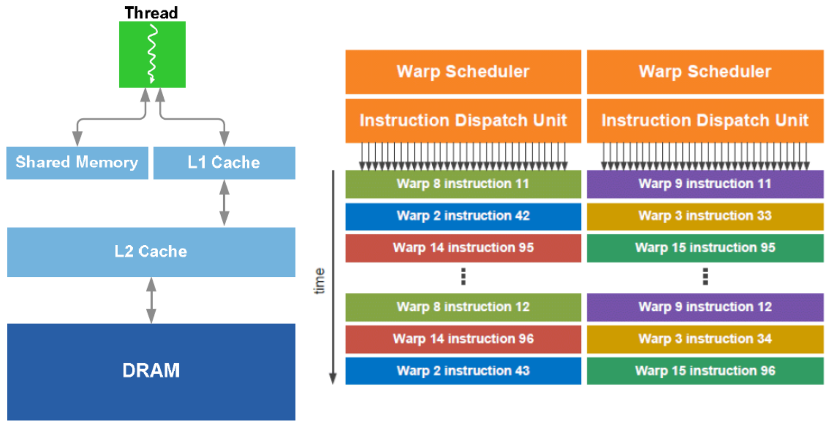
GT200 and Fermi SIMD processor



Fermi Architecture Innovations

- Each SIMD processor has
  - Two SIMD thread schedulers, two instruction dispatch units
  - 16 SIMD lanes (SIMD width=32, chime=2 cycles), 16 load-store units, 4 special function units
  - Thus, two threads of SIMD instructions are scheduled every two clock cycles
- Fast double precision
- Caches for GPU memory
- 64-bit addressing and unified address space
- Error correcting codes
- Faster context switching
- Faster atomic instructions

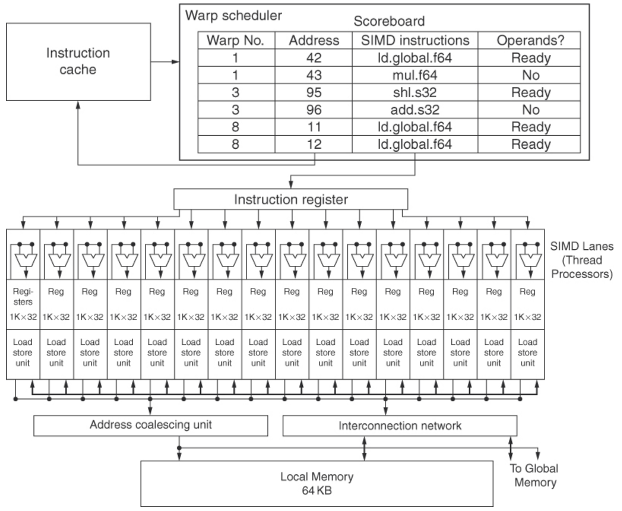
Fermi: Multithreading and Memory Hierarchy



Example

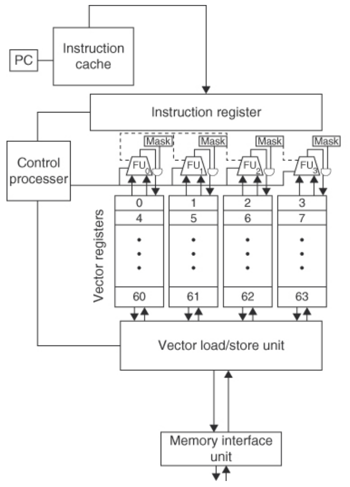
- Multiply two vectors of length 8192
  - Code that works over all elements is the grid
  - Thread blocks break this down into manageable sizes
    - 512 threads per block
  - SIMD instruction executes 32 elements at a time
  - Thus grid size = 16 blocks
  - Block is analogous to a strip-mined vector loop with vector length of 32
  - Block is assigned to a *multithreaded SIMD processor* by the *thread block scheduler*
  - Current-generation GPUs (Fermi) have 7-16 multithreaded SIMD processors

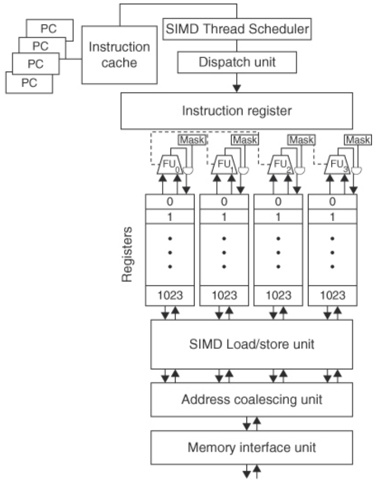
Example



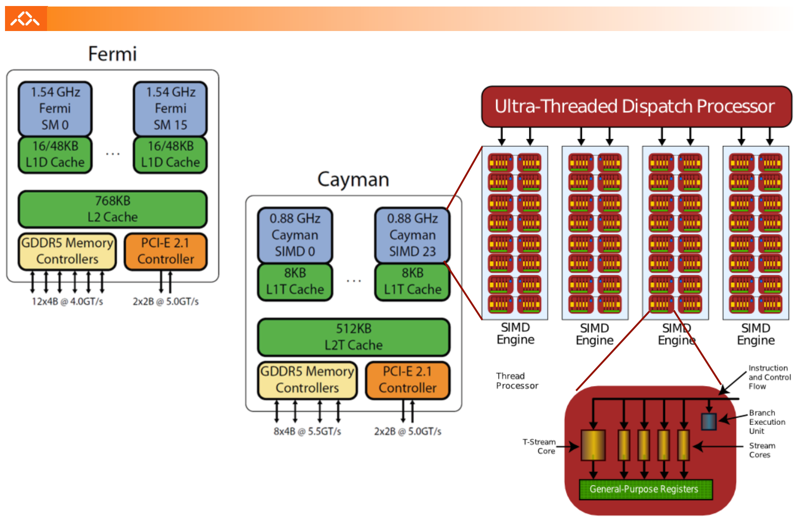
Warp No.	Address	SIMD instructions	Operands?
1	42	ld.global.f64	Ready
1	43	mul.f64	No
3	95	shl.s32	Ready
3	96	add.s32	No
8	11	ld.global.f64	Ready
8	12	ld.global.f64	Ready

Vector Processor *versus* CUDA core





GPU: NVidia Fermi versus AMD Cayman


Fermi
1.54 GHz Fermi SM 0
1.54 GHz Fermi SM 15
16/48KB L1D Cache
768KB L2 Cache
GDDR5 Memory Controllers
PCI-E 2.1 Controller
12x4B @ 4.0GT/s
2x2B @ 5.0GT/s
Cayman
0.88 GHz Cayman SIMD 0
0.88 GHz Cayman SIMD 23
8KB L1T Cache
512KB L2T Cache
GDDR5 Memory Controllers
PCI-E 2.1 Controller
8x4B @ 5.5GT/s
2x2B @ 5.0GT/s
Ultra-Threaded Dispatch Processor
SIMD Engine
Thread Processor
Instruction and Control Flow
Branch Execution Unit
T-Stream Core
Stream Cores
General-Purpose Registers