Intro to GPU's for Parallel Computing

Goals for Rest of Course

- Learn how to program massively parallel processors and achieve
  - high performance
  - functionality and maintainability
  - scalability across future generations
- Acquire technical knowledge required to achieve the above goals
  - principles and patterns of parallel programming
  - processor architecture features and constraints
  - programming API, tools and techniques
- Overview of architecture first, then introduce architecture as we go

Equipment

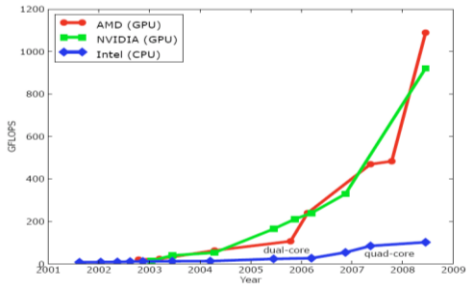
- Your own, if CUDA-enabled; will use CUDA SDK in C
  - Compute Unified Device Architecture
  - NVIDIA G80 or newer
  - G80 emulator won't quite work
- Lab machine – Tesla
  - Ubuntu
  - Quad core Xeon, 2 Ghz
  - 16 Gb memory
  - Two Tesla C1060 "Tesla C1060 Computing Processor Board"
    - 240 Cores
    - 1.3 Ghz Clock
    - 4 Gb memory
  - MD5 test
    - Average 363.67 Mhash/s
    - 2x 3.2 Ghz Xeon: 42 Mhash/s

Equipment

- May use our Beowulf cluster for MPI, beancounter.math.uaa.alaska.edu
  - 13 custom-built boxes each containing a dual processor 1 Ghz Pentium III, 768 Mb of shared memory
  - Total of 27 nodes, including the master.
  - NetBSD 2.0F
  - Connected through a 100Mbps switch.

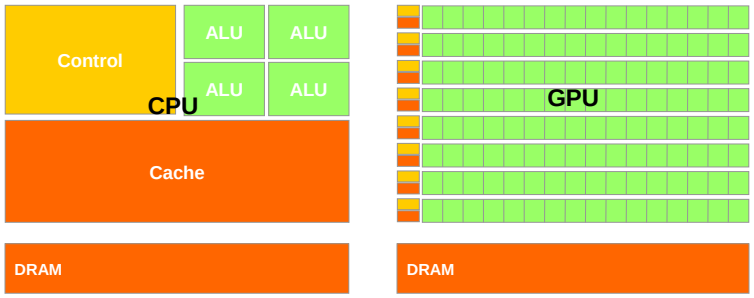
Why Massively Parallel Processors

- A quiet revolution and potential build-up
  - 2006 Calculation: 367 GFLOPS vs. 32 GFLOPS
  - G80 Memory Bandwidth: 86.4 GB/s vs. 8.4 GB/s
  - Until recently, programmed through graphics API

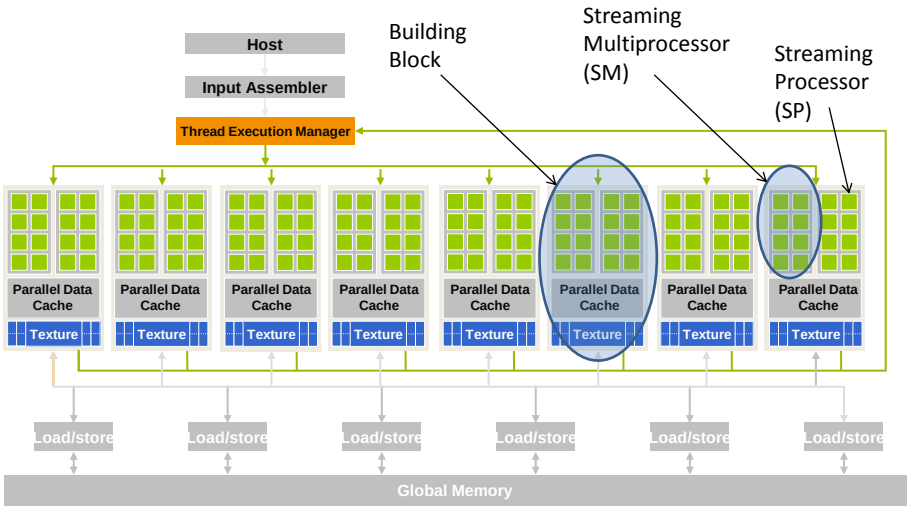


  - GPU in every PC and workstation – massive volume and potential impact

CPUs and GPUs have fundamentally different design philosophies



Architecture of a CUDA-capable GPU



30 SM's each with 8 SP's on the C1060

GT200 Characteristics

- 1 TFLOPS peak performance (25-50 times of current high-end microprocessors)
- 265 GFLOPS sustained for apps such as Visual Molecular Dynamics (VMD)
- Massively parallel, 128 cores, 90W
- Massively threaded, sustains 1000s of threads per app
- 30-100 times speedup over high-end microprocessors on scientific and media applications: medical imaging, molecular dynamics

"I think they're right on the money, but the huge performance differential (currently 3 GPUs ~= 300 SGI Altix Itanium2s) will invite close scrutiny so I have to be careful what I say publically until I triple check those numbers."

-John Stone, VMD group, Physics UIUC

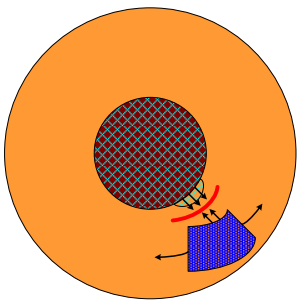
Future Apps Reflect a Concurrent World

- Exciting applications in future mass computing market have been traditionally considered "supercomputing applications"
  - Molecular dynamics simulation, Video and audio coding and manipulation, 3D imaging and visualization, Consumer game physics, and virtual reality products
  - These "Super-apps" represent and model physical, concurrent world
- Various granularities of parallelism exist, but…
  - programming model must not hinder parallel implementation
  - data delivery needs careful management

9

Stretching Traditional Architectures

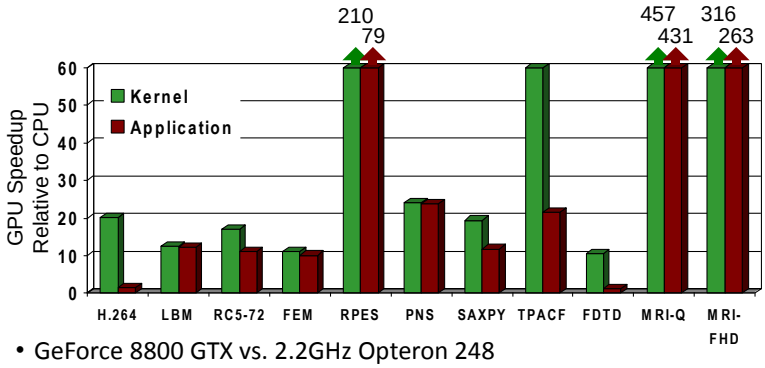
- Traditional parallel architectures cover some super-applications
  - DSP, GPU, network apps, Scientific
- The game is to grow mainstream architectures "out" or domain-specific architectures "in"
  - CUDA is latter



Sample of Previous GPU Projects

Application	Description	Source	Kernel	% time
H.264	SPEC '06 version, change in guess vector	34,811	194	35%
LBM	SPEC '06 version, change to single precision and print fewer reports	1,481	285	>99%
RC5-72	Distributed.net RC5-72 challenge client code	1,979	218	>99%
FEM	Finite element modeling, simulation of 3D graded materials	1,874	146	99%
RPES	Rye Polynomial Equation Solver, quantum chem, 2-electron repulsion	1,104	281	99%
PNS	Petri Net simulation of a distributed system	322	160	>99%
SAXPY	Single-precision implementation of saxpy, used in Linpack's Gaussian elim. routine	952	31	>99%
TRACF	Two Point Angular Correlation Function	536	98	96%
FDTD	Finite-Difference Time Domain analysis of 2D electromagnetic wave propagation	1,365	93	16%
MRI-Q	Computing a matrix Q, a scanner's configuration in MRI reconstruction	490	33	>99%

Speedup of Applications



- GeForce 8800 GTX vs. 2.2GHz Opteron 248
- 10× speedup in a kernel is typical, as long as the kernel can occupy enough parallel threads
- 25× to 400× speedup if the function's data requirements and control flow suit the GPU and the application is optimized