GeForce 8800 & NVIDIA CUDA
A New Architecture for Computing on the GPU
Digital Tomosynthesis: Signal Processing

Pioneering work at Massachusetts General Hospital

- 100X speed-up with NVIDIA GPU
  - 32 node server takes 5 hours
  - 1 GPU takes 5 minutes
- Improved diagnostic value
  - Clearer images
  - Fewer obstructions
  - Earlier detection

Advanced Imaging Solution of the Year

“reduced reconstruction time from 5 hours to 5 minutes”

Low-dose X-ray Projections
Computationally Intense Reconstruction
Electromagnetic Simulation

Acceleware FDTD acceleration technology for the GPU

- 3D Finite-Difference and Finite-Element
- Modeling of:
  - Cell phone irradiation
  - MRI Design / Modeling
  - Printed Circuit Boards
  - Radar Cross Section (Military)
- Large speedups with NVIDIA GPUs

Pacemaker with Transmit Antenna

Performance (Mcells/s)

- 1X
- 5X
- 10X
- 20X

CPU 3.2 GHz

1 GPU

2 GPUs

4 GPUs
CUDA & GPU Computing

New Architecture for Computing

Standard C Programming

Unprecedented Performance

New Applications

dim3 DimGrid(100, 50);    // 5000 thread blocks
Dim3 DimBlock(4, 8, 8);   // 256 threads per block
size_t SharedMemBytes = 64; // 64 bytes of shared memory
KernelFunc<<< DimGrid, DimBlock, SharedMemBytes >>>(...);
GPGPU Programming Model

Start by creating a quad

“Programs” created with raster operation

Read textures as input to OpenGL shader program

Write answer to texture memory as a “color”

All this just to do A + B
Current Constraints

- Graphics API
- Addressing modes
  - Limited texture size/dimension
- Shader capabilities
  - Limited outputs
- Instruction sets
  - Integer & bit ops
- Communication limited
  - Between pixels
  - Scatter $a[i] = p$
GeForce 7800 Pixel

Input Registers

Fragment Program

Texture

Constants

Registers

Output Registers
Thread Programs

Features
- Millions of instructions
- Full Integer and Bit instructions
- No limits on branching, looping
- 1D, 2D, or 3D thread ID allocation

Thread Number

Thread Program

Output Registers

Texture

Constants

Registers
Global Memory

Features
- Fully general load/store to GPU memory: Scatter/Gather
- Programmer flexibility on how memory is accessed
- Untyped, not limited to fixed texture types
- Pointer support

Thread Number
Thread Program
Global Memory

Texture
Constants
Registers
Parallel Data Cache

Features
- Dedicated on-chip memory
- Shared between threads for inter-thread communication
- Explicitly managed
- As fast as registers

Thread Number

Thread Program

Parallel Data Cache

Texture

Constants

Registers

Global Memory
Example Algorithm - Fluids

Goal: Calculate PRESSURE in a fluid

Pressure = Sum of neighboring pressures

\[ P_{n'} = P_1 + P_2 + P_3 + P_4 \]

So the pressure for each particle is...

Pressure_1 = P_1 + P_2 + P_3 + P_4
Pressure_2 = P_3 + P_4 + P_5 + P_6
Pressure_3 = P_5 + P_6 + P_7 + P_8
Pressure_4 = P_7 + P_8 + P_9 + P_{10}

Pressure depends on neighbors
Example Fluid Algorithm

CPU

GPGPU

CUDA
GPU Computing

Single thread out of cache

Multiple passes through video memory

Parallel execution through cache

Data/Computation

Program/Control

P_n' = P_1 + P_2 + P_3 + P_4

P_n' = P_1 + P_2 + P_3 + P_4

P_n' = P_1 + P_2 + P_3 + P_4

Parallel Data

Shared Data

© NVIDIA Corporation 2006
Parallel Data Cache

Addresses a fundamental problem of stream computing

Bring the data closer to the ALU

- Stage computation for the parallel data cache
- Minimize trips to external memory
- Share values to minimize overfetch and computation
- Increases arithmetic intensity by keeping data close to the processors
- User managed generic memory, threads read/write arbitrarily
Streaming vs. GPU Computing

Streaming
- Gather in, Restricted write
- Memory is far from ALU
- No inter-element communication

CUDA
- More general data parallel model
- Full Scatter / Gather
- PDC brings the data closer to the ALU
- App decides how to decompose the problem across threads
- Share and communicate between threads to solve problems efficiently
GeForce 8800 GTX Graphics Board

- 128 Multi-processors
- 575MHz Core
- 1350MHz Shader
- 900MHz Memory
- 768MB GDDR3 Memory

$599 e-tail

© NVIDIA Corporation 2006
GeForce 8800 GPU Computing

Processors execute computing threads
Thread Processor

- 128, 1.35 GHz processors
- 16KB Parallel Data Cache per cluster
- Scalar architecture
- IEEE 754 Precision
- Full featured instruction set
CUDA Programming Model
Programming Model: A Highly Multi-threaded Coprocessor

The GPU is viewed as a compute device that:
- Is a coprocessor to the CPU or host
- Has its own DRAM (device memory)
- Runs many threads in parallel

Data-parallel portions of an application execute on the device as kernels which run many cooperative threads in parallel

Differences between GPU and CPU threads
- GPU threads are extremely lightweight
- Very little creation overhead
- GPU needs 1000s of threads for full efficiency
- Multi-core CPU needs only a few
C on the GPU

- A simple, explicit programming language solution
- Extend only where necessary

```c
__global__ void KernelFunc(...);
__device__ int GlobalVar;
__shared__ int SharedVar;

KernelFunc<<< 500, 128 >>>(...);
```
Runtime Component: Memory Management

- Explicit GPU memory allocation
  - Returns pointers to GPU memory
- Device memory allocation
  - `cudaMalloc()`, `cudaFree()`
- Memory copy from host to device, device to host, device to device
  - `cudaMemcpy()`, `cudaMemcpy2D()`, ...
  - `cudaGetSymbolAddress()`
- OpenGL & DirectX interoperability
  - `cudaGLMapBufferObject()`
CUDA SDK

- Standard Libraries: FFT, BLAS,…
- Integrated CPU and GPU C Source Code
- NVIDIA C Compiler
- NVIDIA Assembly for Computing
- CPU Host Code
- CUDA Runtime & Driver
- Profiler
CUDA Stable Fluids Demo

CUDA port of:
New Applications Enabled by CUDA

- Rigid Body Physics Solver
- Matrix Numerics
- Wave Equation
- Biological Sequence Match
- Finance

CUDA Advantage

GeForce 8800 vs. 2.66 GHz Core 2 Duo
CUDA Performance Advantages

Performance:
- BLAS1: 60+ GB/sec
- BLAS3: 100+ GFLOPS
- FFT: 52 benchFFT* GFLOPS
- FDTD: 1.2 Gcells/sec
- SSEARCH: 5.2 Gcells/sec
- Black Scholes: 4.7 GOptions/sec

Benefits:
- Leveraging the parallel data cache
- GPU memory bandwidth
- GPU GFLOPS performance
- Custom hardware intrinsics
  - __sinf, __cosf, __expf, __logf, ...

All benchmarks are compiled code!

*GFLOPS as defined by http://www.fftw.org/benchfft
© NVIDIA Corporation 2006
Conclusions

- GPU Computing on GeForce 8800
  - Simple threading model
  - Parallel data cache
  - General global memory access

- CUDA Programming model
  - C on GPUs
  - Tool chain and driver designed for computation

- Libraries optimized for GPU Computing
  - CUFFT, CUBLAS

- Availability
  - Linux and Windows
  - Register for the Beta online

http://developer.nvidia.com/CUDA
Questions?

http://developer.nvidia.com/CUDA