NVIDIA®

NVIDIA CUDA Software and GPU Parallel Computing Architecture

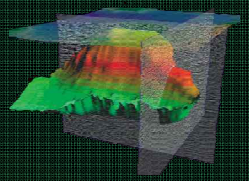
David B. Kirk, Chief Scientist

Outline

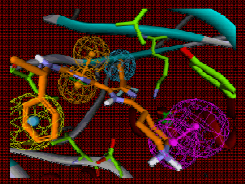
- Applications of GPU Computing
- CUDA Programming Model Overview
- Programming in CUDA – The Basics
- How to Get Started!

- Exercises / Examples Interleaved with Presentation Materials
  - Homework for later ☺

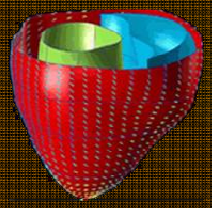
Future Science and Engineering Breakthroughs Hinge on Computing



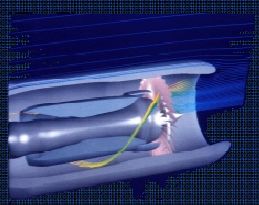
Computational Geoscience



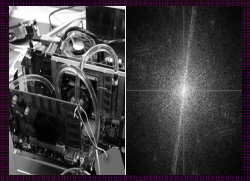
Computational Chemistry



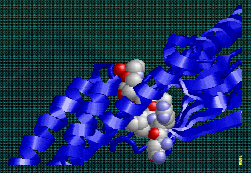
Computational Medicine



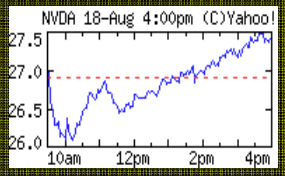
Computational Modeling



Computational Physics



Computational Biology



Computational Finance

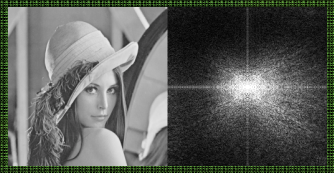
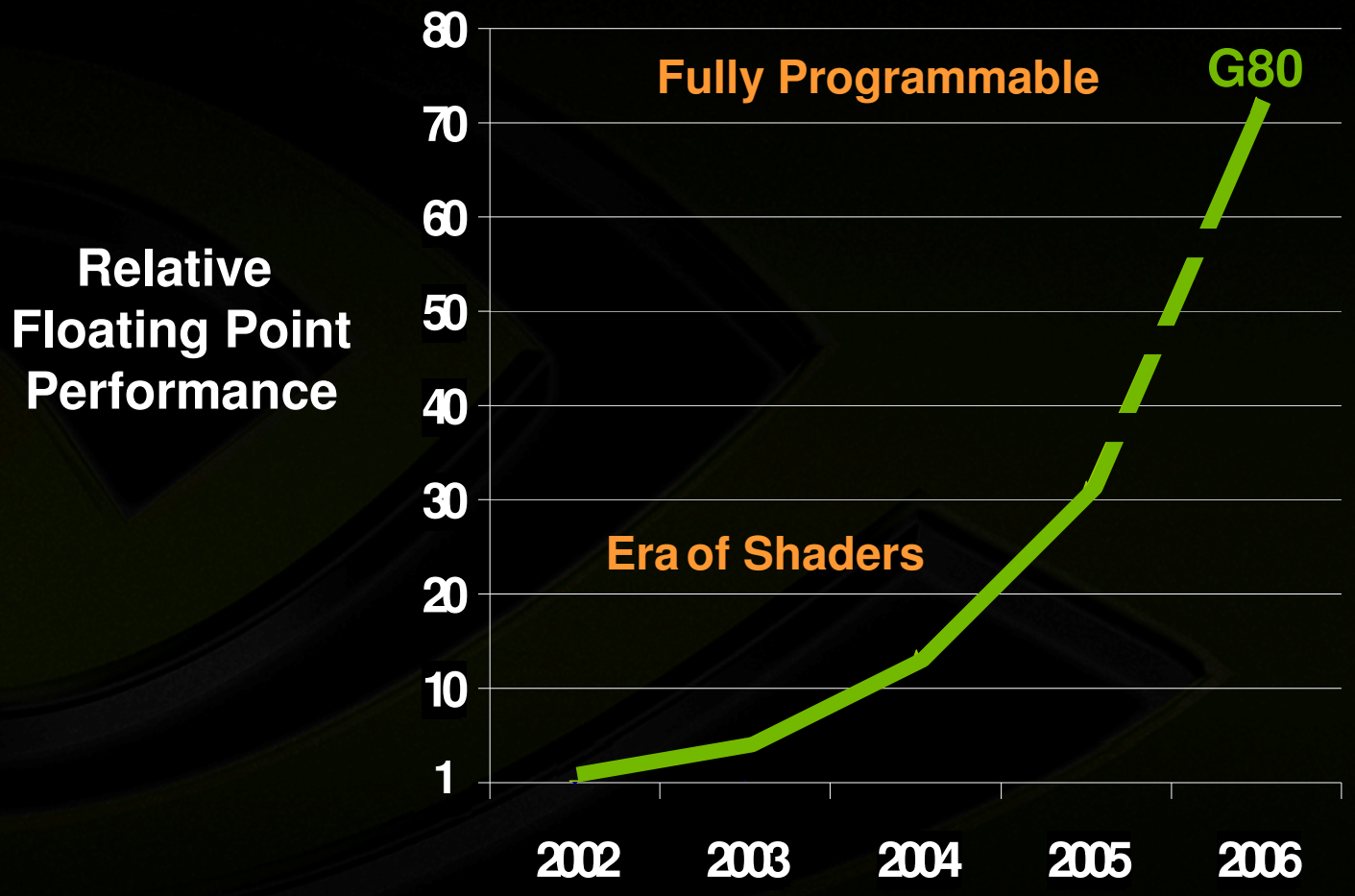


Image Processing

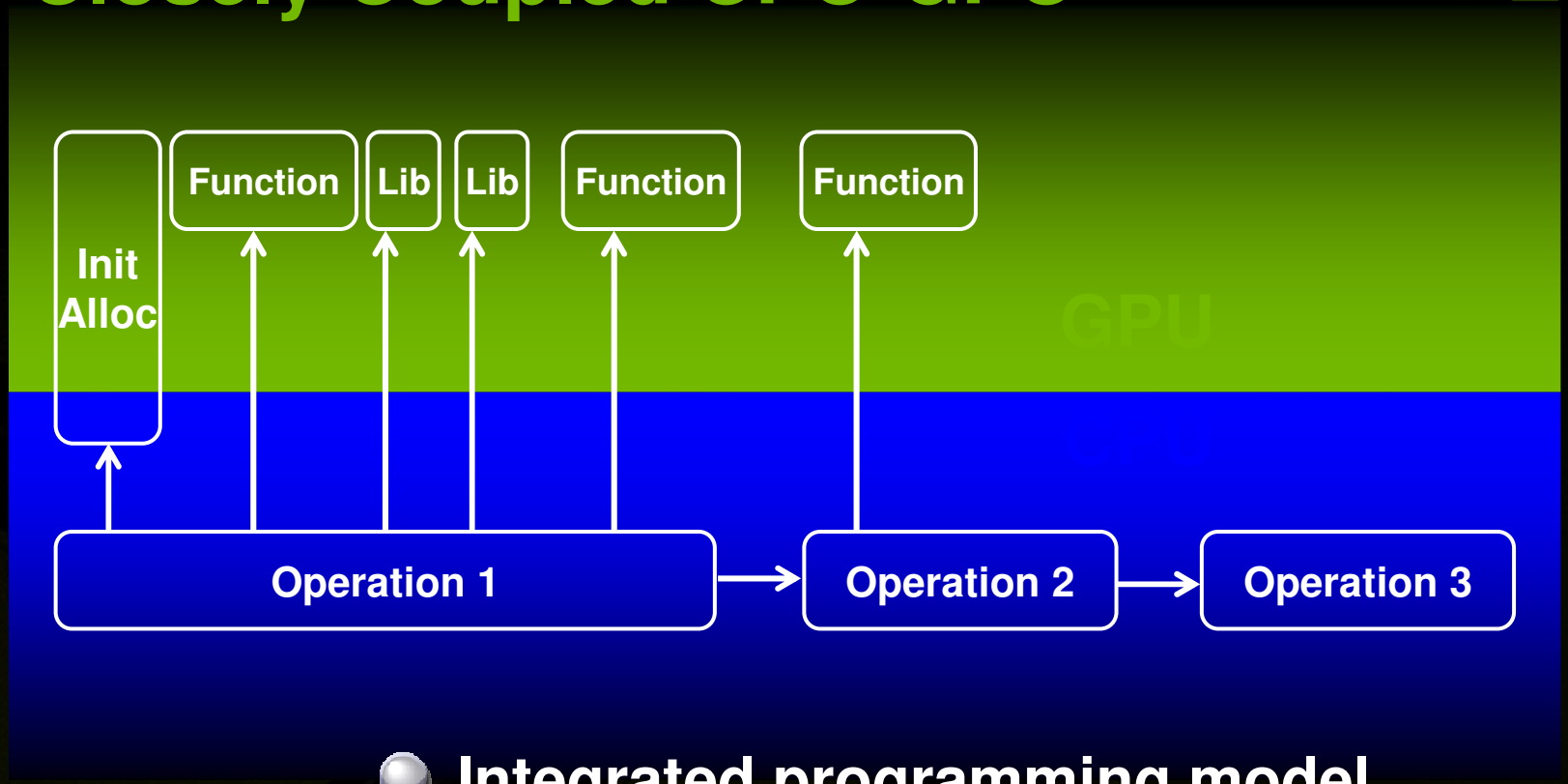
Faster is not “just Faster”

- 2-3X faster is “just faster”
  - Do a little more, wait a little less
  - Doesn’t change how you work
- 5-10x faster is “significant”
  - Worth upgrading
  - Worth re-writing (parts of) the application
- 100x+ faster is “fundamentally different”
  - Worth considering a new platform
  - Worth re-architecting the application
  - Makes new applications possible
  - Drives “time to discovery” and creates fundamental changes in Science

The GPU is a New Computation Engine



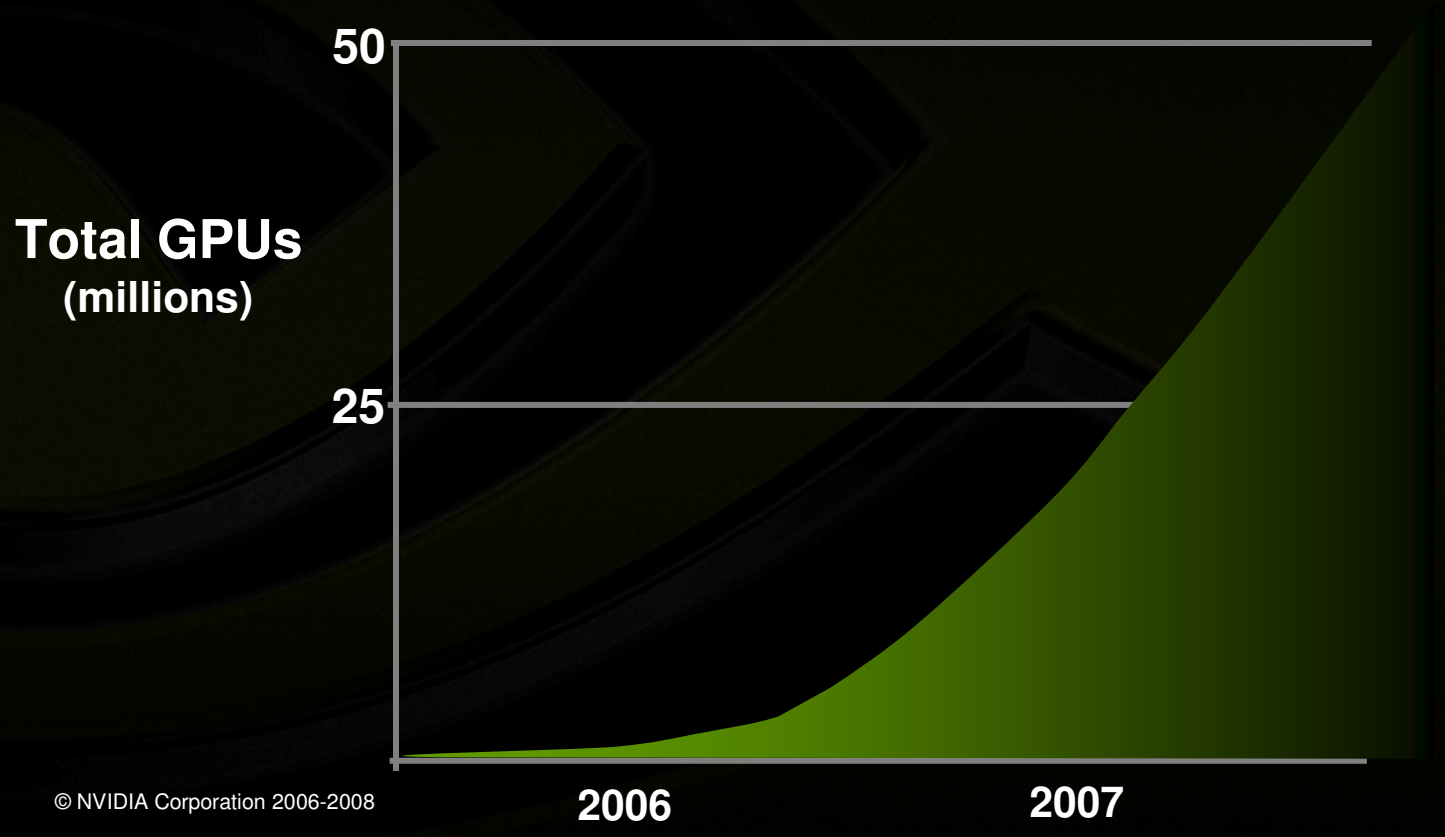
Closely Coupled CPU-GPU



- Integrated programming model
- High speed data transfer – up to 3.2 GB/s
- Asynchronous operation
- Large GPU memory systems

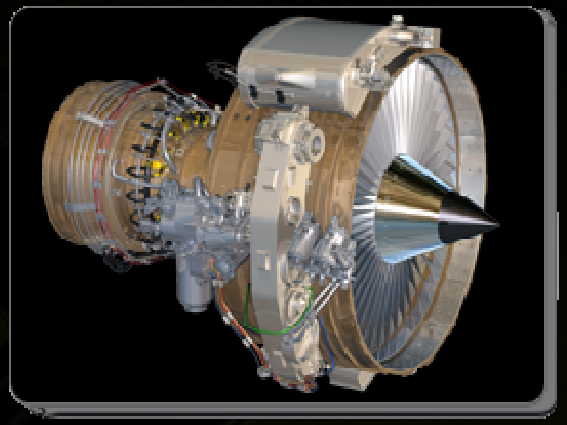
Millions of CUDA-enabled GPUs

- Dedicated computing
- C on the GPU
- Servers through Notebook PCs

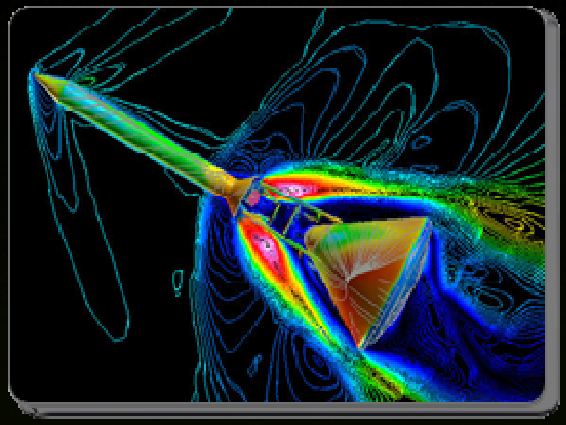


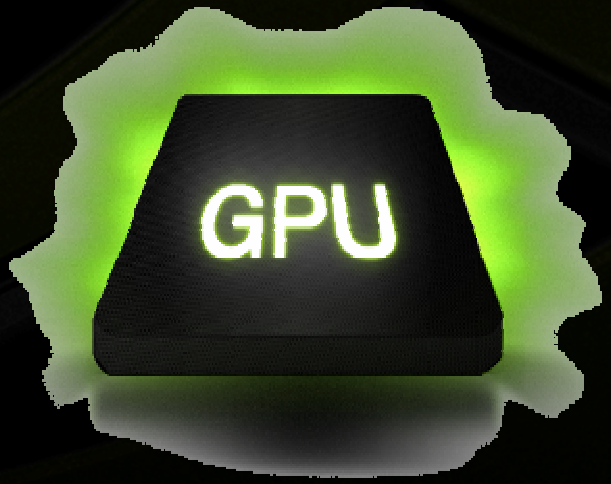
GeForce®
Entertainment

Quadro®
Design & Creation



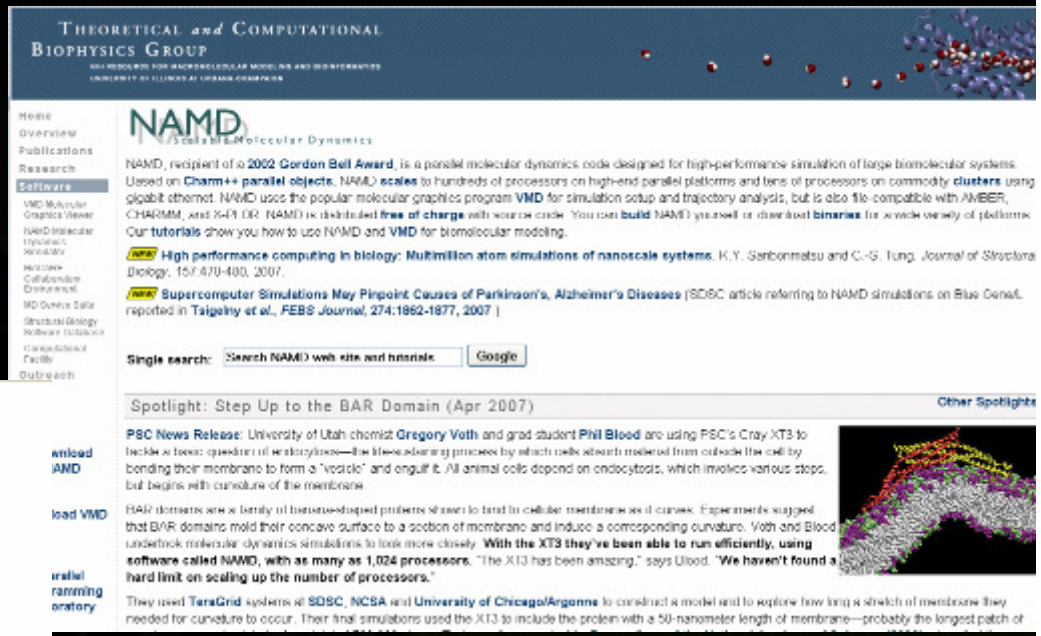
Tesla™
High Performance Computing





VMD/NAMD Molecular Dynamics

- 240X speedup
- Computational biology



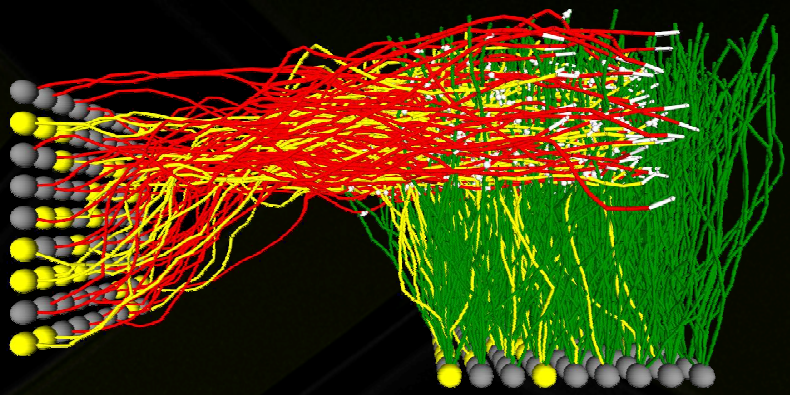
Parallel GPUs with Multithreading: 705 GFLOPS /w 3 GPUs

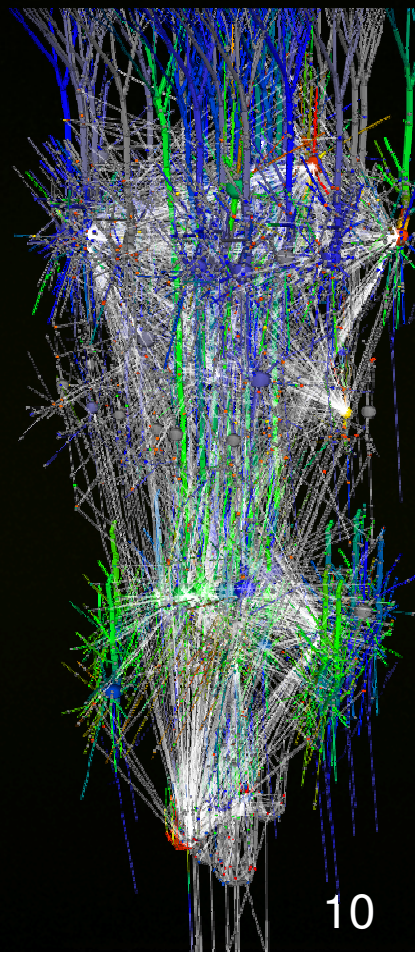
- One host thread is created for each CUDA GPU
- Threads are spawned and attach to their GPU based on their host thread ID
  - First CUDA call binds that thread's CUDA context to that GPU for life
  - Handling error conditions within child threads is dependent on the thread library and, makes dealing with any CUDA errors somewhat tricky, left as an exercise to the reader.... ☺
- Map slices are computed cyclically by the GPUs
- Want to avoid false sharing on the host memory system
  - map slices are usually much bigger than the host memory page size, so this is usually not a problem for this application
- Performance of 3 GPUs is stunning!
- Power: 3 GPU test box consumes 700 watts running flat out

http://www.ks.uiuc.edu/Research/vmd/projects/ece498/lecture/

EvolvedMachines

- Simulate the brain circuit
- Sensory computing: vision, olfactory
- 130X Speed up

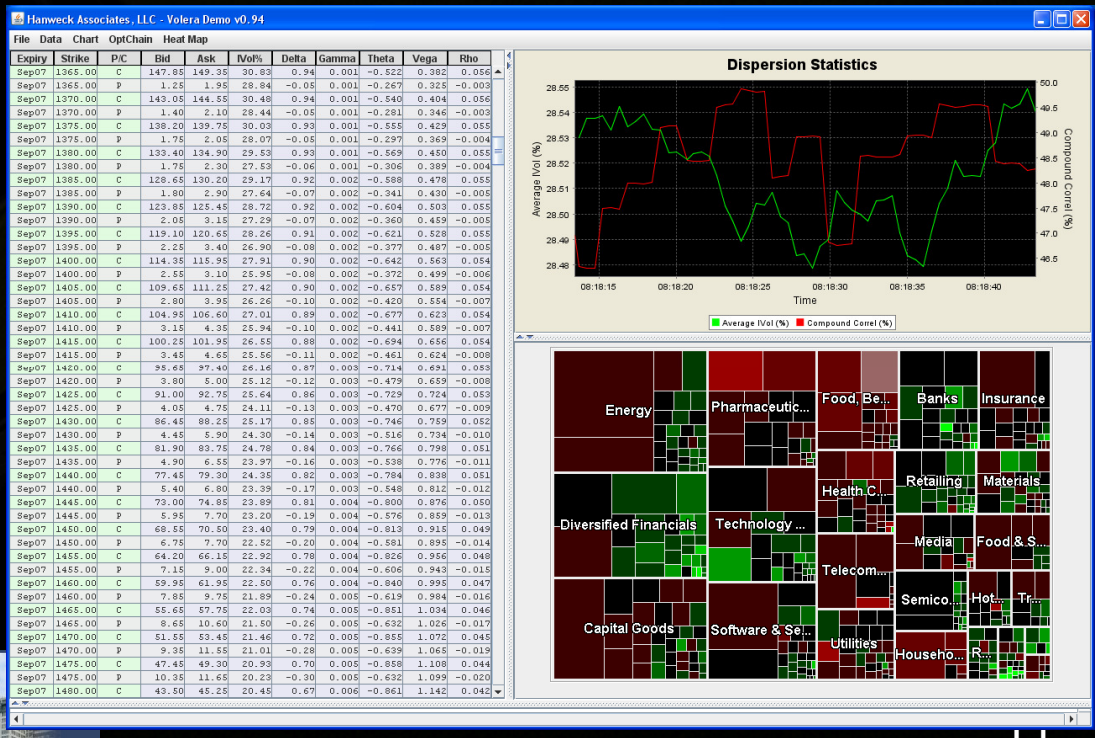




EvolvedMachines

Hanweck Associates

- VOLERA, real-time options implied volatility engine
- Accuracy results with SINGLE PRECISION
- Evaluate all U.S. listed equity options in <1 second



(www.hanweckassoc.com)

LIBOR APPLICATION:

Mike Giles and Su Xiaoke
Oxford University Computing Laboratory

- LIBOR Model with portfolio of swaptions
- 80 initial forward rates and 40 timesteps to maturity
- 80 Deltas computed with adjoint approach

	No Greeks		Greeks	
Intel Xeon	18.1s	-	26.9s	-
ClearSpeed Advance 2 CSX600	2.9s	6x	6.4s	4x
NVIDIA 8800 GTX	0.045s	400x	0.18s	149x

"The performance of the CUDA code on the 8800 GTX is exceptional"
-Mike Giles

Source codes and papers available at:
http://web.comlab.ox.ac.uk/oucl/work/mike.giles/hpc

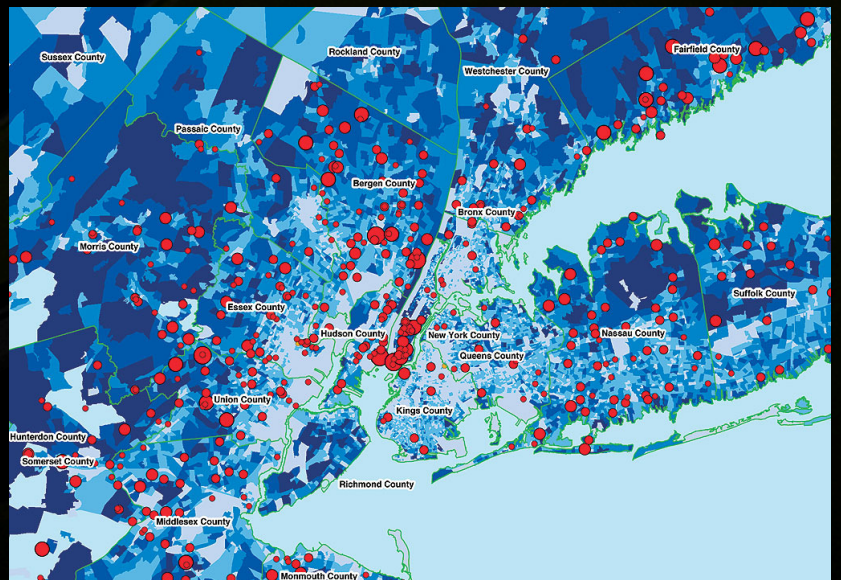
Manifold 8 GIS Application

From the Manifold 8 feature list:

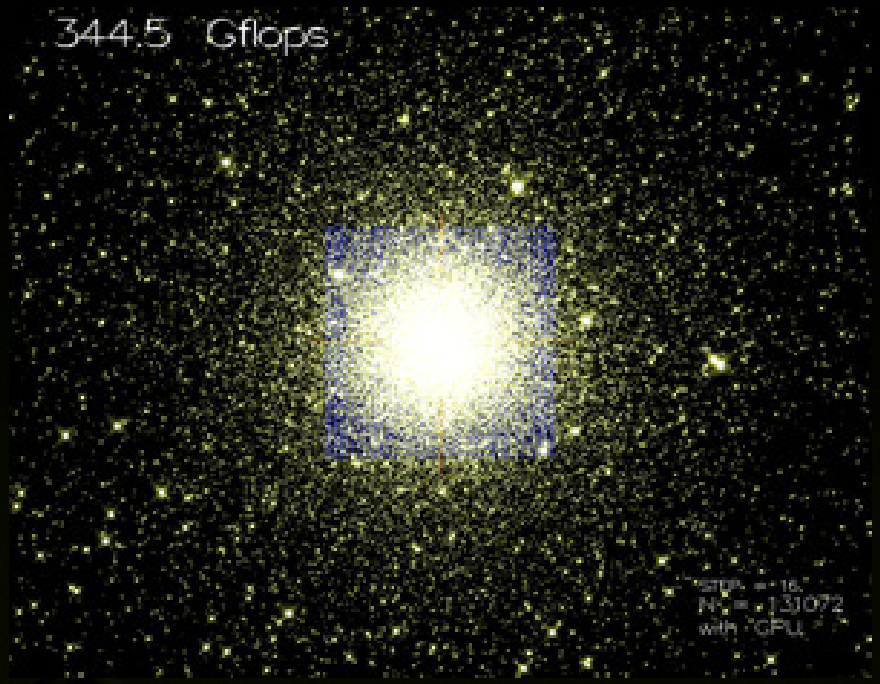
… applications fitting CUDA capabilities that might have taken **tens of seconds or even minutes** can be accomplished in **hundredths of seconds**. … **CUDA will clearly emerge to be the future of almost all GIS computing**

From the user manual:

"NVIDIA CUDA … could well be the most revolutionary thing to happen in computing since the invention of the microprocessor



nbody Astrophysics



Astrophysics research

1 GF on standard PC

300+ GF on GeForce 8800GTX

http://progrape.jp/cs/

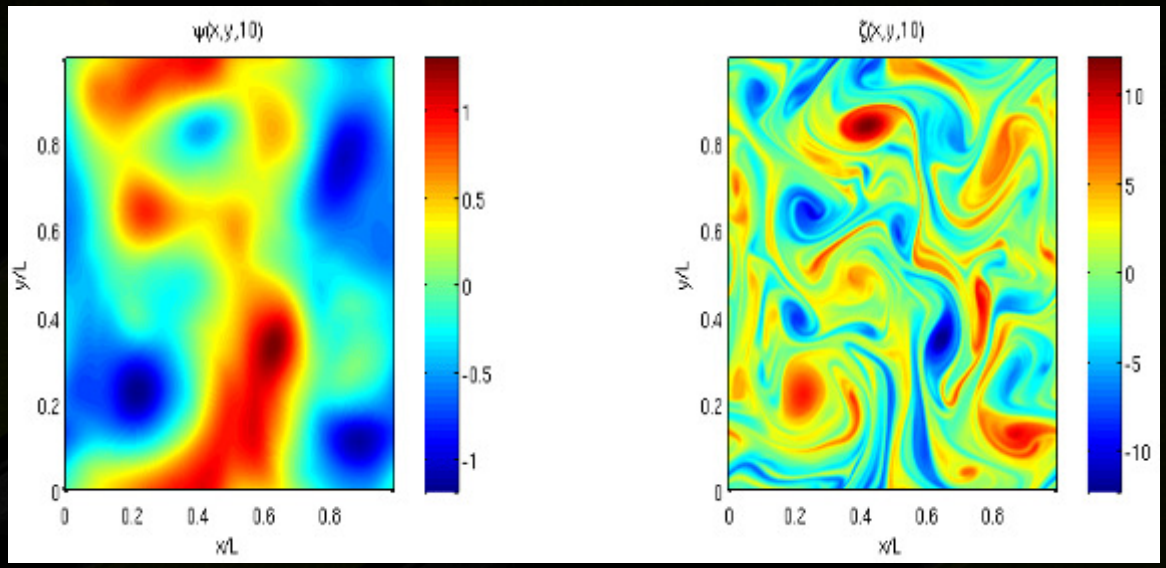
Faster than GRAPE-6Af custom simulation computer

Video demo

Matlab: Language of Science

17X with MATLAB CPU+GPU

http://developer.nvidia.com/object/matlab_cuda.html



Pseudo-spectral simulation of 2D Isotropic turbulence

http://www.amath.washington.edu/courses/571-winter-2006/matlab/FS_2Dturb.m

NVIDIA®

CUDA Programming Model Overview

GPU Computing

- GPU is a massively parallel processor
  - NVIDIA G80: 128 processors
  - Support thousands of active threads (12,288 on G80)
- GPU Computing requires a programming model that can efficiently express that kind of parallelism
  - Most importantly, data parallelism
- CUDA implements such a programming model

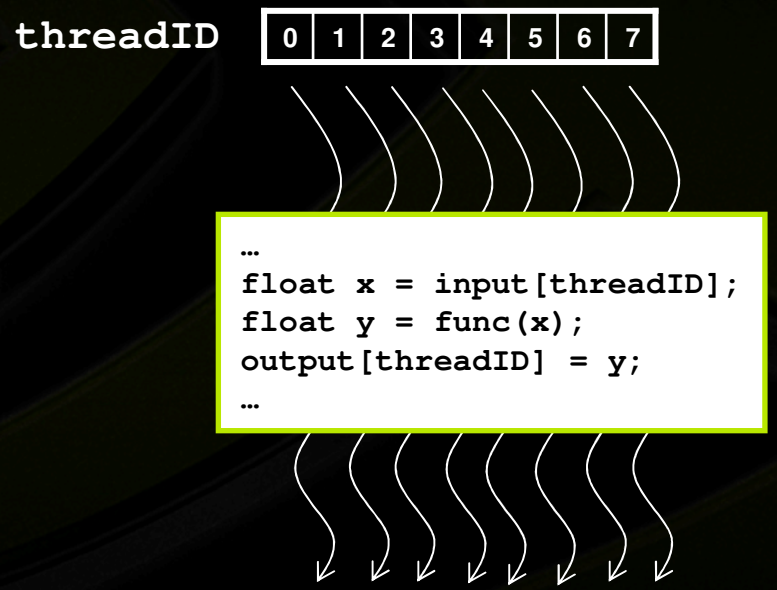
CUDA Kernels and Threads

- Parallel portions of an application are executed on the device as kernels
  - One kernel is executed at a time
  - Many threads execute each kernel
- Differences between CUDA and CPU threads
  - CUDA threads are extremely lightweight
    - Very little creation overhead
    - Instant switching
  - CUDA uses 1000s of threads to achieve efficiency
    - Multi-core CPUs can use only a few

Definitions:
Device = GPU; *Host* = CPU
Kernel = function that runs on the device

Arrays of Parallel Threads

- A CUDA kernel is executed by an array of threads
  - All threads run the same code
  - Each thread has an ID that it uses to compute memory addresses and make control decisions



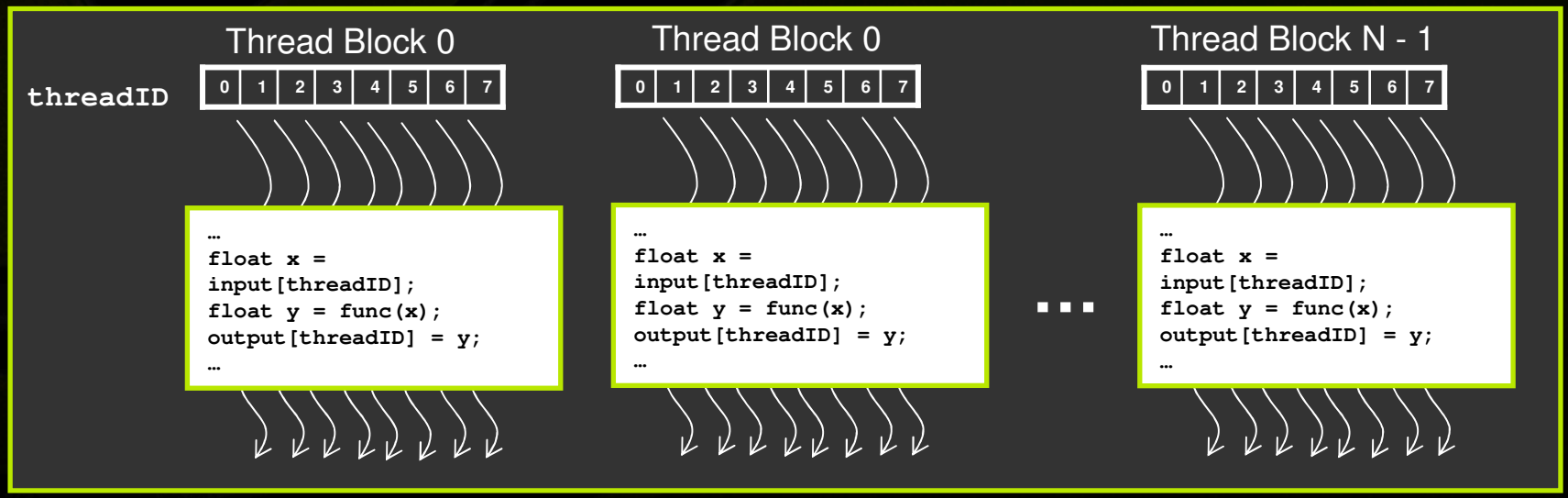
Thread Cooperation

- The Missing Piece: threads may need to cooperate
- Thread cooperation is valuable
  - Share results to save computation
  - Synchronization
  - Share memory accesses
    - Drastic bandwidth reduction
- Thread cooperation is a powerful feature of CUDA

Thread Blocks: Scalable Cooperation

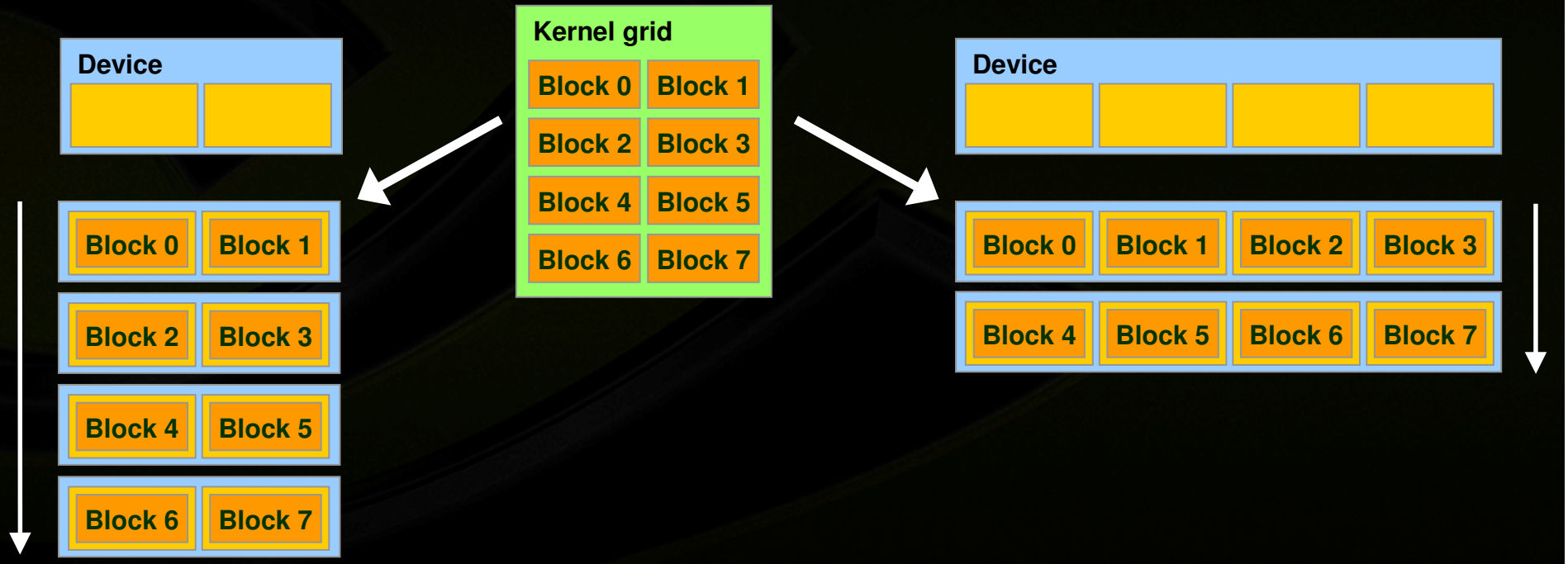
- Divide monolithic thread array into multiple blocks
  - Threads within a block cooperate via **shared memory**
  - Threads in different blocks cannot cooperate

- Enables programs to **transparently scale** to any number of processors!



Transparent Scalability

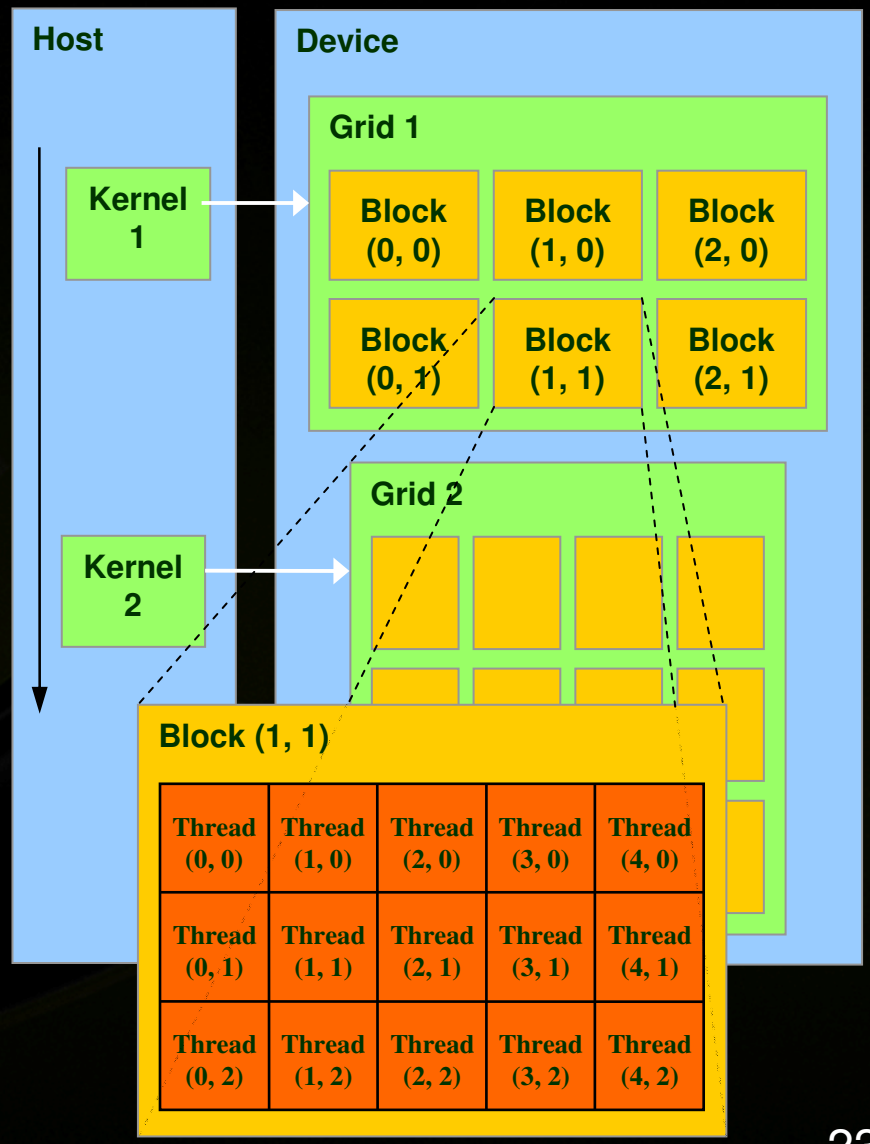
- Hardware is free to schedule thread blocks on any processor at any time
  - A kernel scales across any number of parallel multiprocessors



CUDA Programming Model

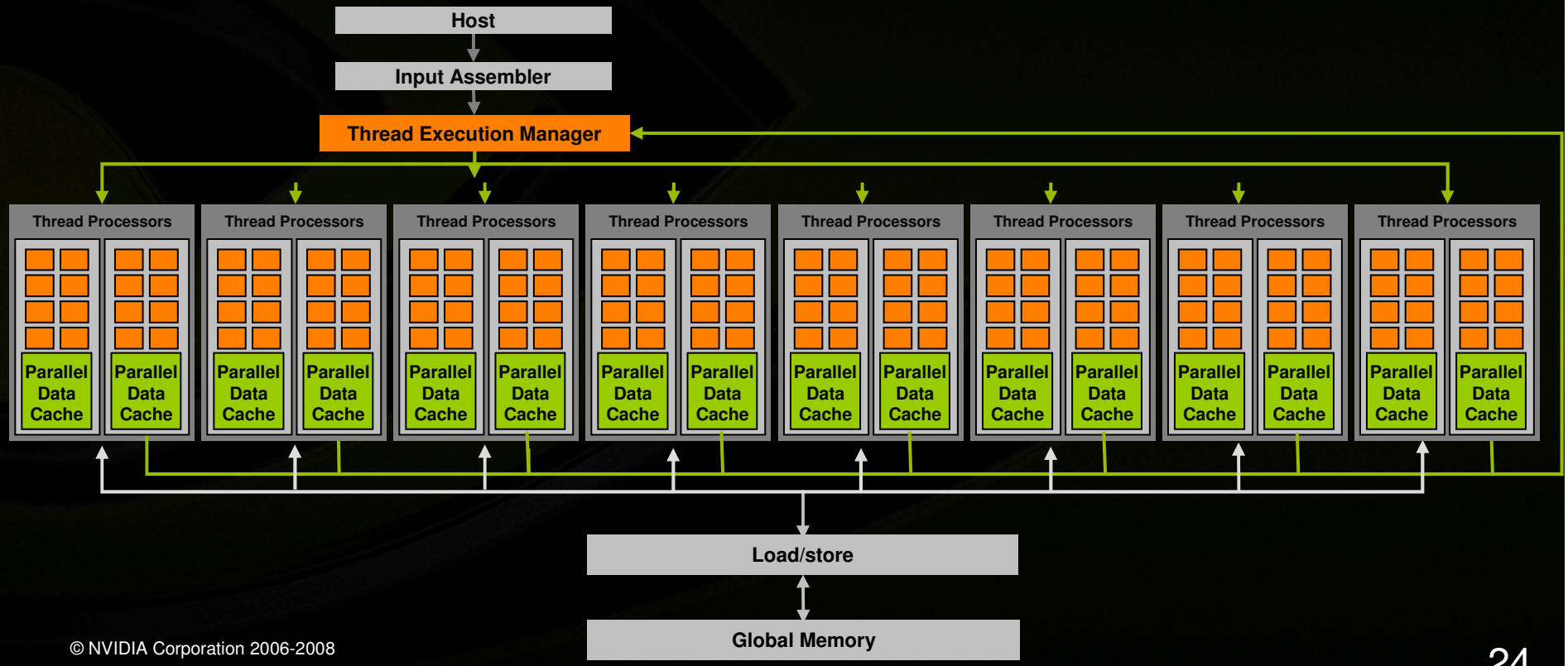
A kernel is executed by a grid of thread blocks

- A thread block is a batch of threads that can cooperate with each other by:
  - Sharing data through shared memory
  - Synchronizing their execution
- Threads from different blocks cannot cooperate



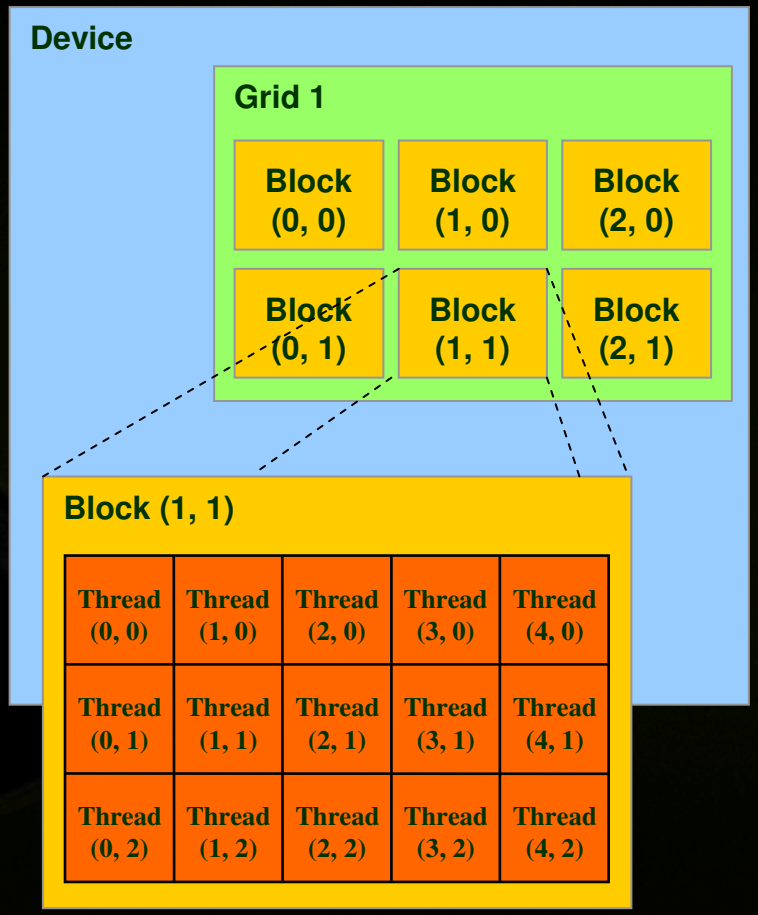
G80 Device

- Processors ■ execute computing threads
- Thread Execution Manager issues threads
- 128 Thread Processors grouped into 16 Multiprocessors (SMs)
- Parallel Data Cache (Shared Memory) enables thread cooperation



Thread and Block IDs

- Threads and blocks have IDs
  - Each thread can decide what data to work on
- Block ID: 1D or 2D
- Thread ID: 1D, 2D, or 3D
- Simplifies memory addressing when processing multi-dimensional data
  - Image processing
  - Solving PDEs on volumes



Kernel Memory Access

- Registers
- Global Memory (external DRAM)
  - Kernel input and output data reside here
  - Off-chip, large
  - Uncached
- Shared Memory (Parallel Data Cache)
  - Shared among threads in a single block
  - On-chip, small
  - As fast as registers

- The host can read & write global memory but not shared memory

Execution Model

- Kernels are launched in grids
  - One kernel executes at a time
- A block executes on one Streaming Multiprocessor (SM)
  - Does not migrate
- Several blocks can reside concurrently on one SM
  - Control limitations (of G8X/G9X GPUs):
    - At most 8 concurrent blocks per SM
    - At most 768 concurrent threads per SM
  - Number is further limited by SM resources
    - Register file is partitioned among all resident threads
    - Shared memory is partitioned among all resident thread blocks

CUDA Advantages over Legacy GPGPU
(Legacy GPGPU is programming GPU through graphics APIs)

- Random access byte-addressable memory
  - Thread can access any memory location
- Unlimited access to memory
  - Thread can read/write as many locations as needed
- Shared memory (per block) and thread synchronization
  - Threads can cooperatively load data into shared memory
  - Any thread can then access any shared memory location
- Low learning curve
  - Just a few extensions to C
  - No knowledge of graphics is required
- No graphics API overhead

CUDA Model Summary

- Thousands of lightweight concurrent threads
  - No switching overhead
  - Hide instruction and memory latency
- Shared memory
  - User-managed L1 cache
  - Thread communication / cooperation within blocks
- Random access to global memory
  - Any thread can read/write any location(s)
- Current generation hardware:
  - Up to 128 streaming processors

Memory	Location	Cached	Access	Scope ("Who?")
Shared	On-chip	N/A	Read/write	All threads in a block
Global	Off-chip	No	Read/write	All threads + host

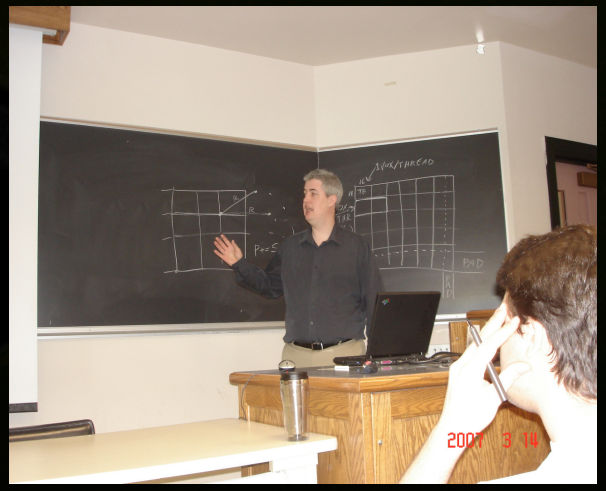
Getting Started

Get CUDA

- CUDA Zone: http://nvidia.com/cuda
  - Programming Guide and other Documentation
  - Toolkits and SDKs for:
    - Windows
    - Linux
    - MacOS
  - Libraries
  - Plugins
  - Forums
  - Code Samples

Come visit the class!

- UIUC ECE498AL – Programming Massively Parallel Processors (http://courses.ece.uiuc.edu/ece498/al/)
  - David Kirk (NVIDIA) and Wen-mei Hwu (UIUC) co-instructors
  - CUDA programming, GPU computing, lab exercises, and projects
  - Lecture slides and voice recordings



Questions?