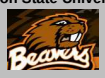
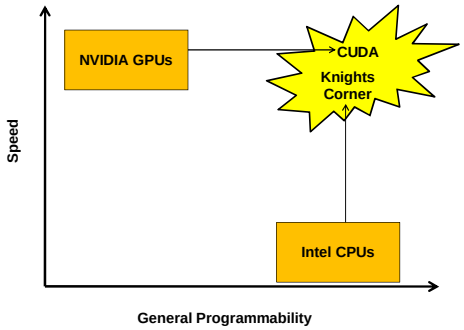
NVIDIA's Compute Unified Device Architecture (CUDA)

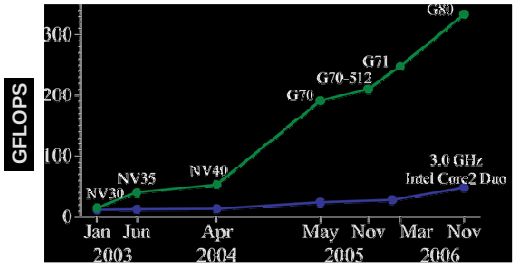
Mike Bailey
mjb@cs.oregonstate.edu
Oregon State University



Reaching the Promised Land



History of GPU Performance vs. CPU Performance



Source: NVIDIA

G80 = GeForce 8800 GTX
G71 = GeForce 7900 GTX
G70 = GeForce 7800 GTX
NV40 = GeForce 6800 Ultra
NV35 = GeForce FX 5950 Ultra
NV30 = GeForce FX 5800

Why have GPUs Been Outpacing CPUs?

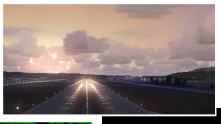
Due to the nature of graphics computations, GPU chips are customized to handle *streaming* data. This means that the data is already sequential and thus the GPU chips do not need the significant amount of cache space that dominates the real estate on general-purpose CPU chips. The GPU die real estate can then be re-targeted to hold more cores and thus to produce more processing power.

The other reason is that general CPU chips contain on-chip logic to process some instructions **out-of-order** if the CPU is blocked waiting on something (e.g., a memory fetch). This, too, takes up chip die space.

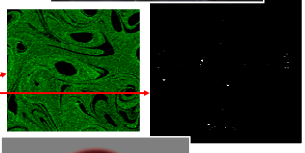
For example, while Intel and AMD are now shipping CPU chips with 4 cores, NVIDIA is shipping GPU chips with 512. Overall, in four years, GPUs have achieved a 17.5-fold increase in performance, a compound annual increase of 2.05X.

How Can You Gain Access to that GPU Power?

1. Write a graphics display program (≥ 1985)



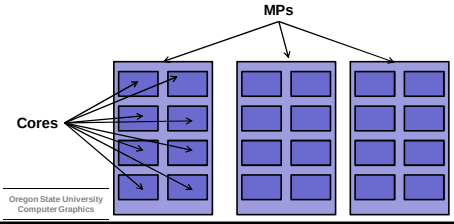
2. Write an application that looks like a graphics display program (≥ 2002)



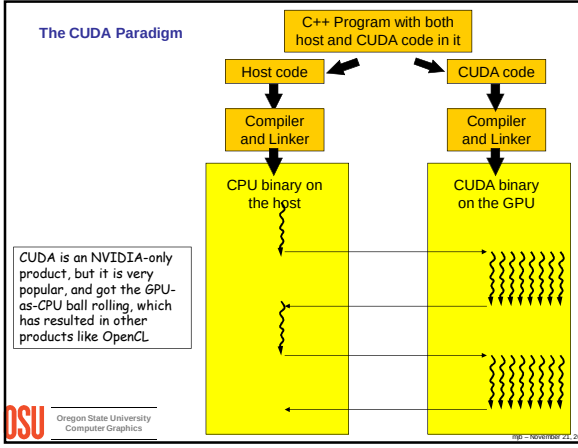
3. Write in CUDA, which looks like C++ (≥ 2006)

CUDA Architecture

- The GPU has some number of MultiProcessors (MPs), depending on the model
- The NVIDIA Fermi 480 and above have 16 MPs
- An 8000-based MP has 8 independent processors (cores)
- A Fermi-based MP has 32 independent processors (cores)
- Memory is divided into Shared Memory and Constant Memory



The CUDA Paradigm



CUDA is an NVIDIA-only product, but it is very popular, and got the GPU-as-CPU ball rolling, which has resulted in other products like OpenCL

If GPUs have so Little Cache, how can they Execute General C++ Code Efficiently?

1. Multiple Multiprocessors
2. Threads – lots and lots of threads

- CUDA expects you to not just have a few threads, but to have ***thousands*** of them!
- All threads execute the same code (called the *kernel*), but operate on different data
- Each thread can figure out which number it is, and thus what its job is
- Think of all the threads as living in a "pool", waiting to be executed
- All processors start by grabbing a thread from the pool
- When a thread gets blocked somehow (a memory access, waiting for information from another thread, etc.), the processor quickly returns the thread to the pool and grabs another one to work on.
- This thread-swap happens within a single cycle

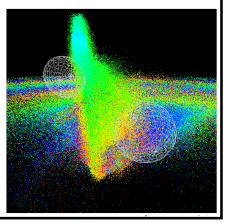
A full memory access requires 200 instruction cycles to complete

So, the Trick is to Break your Problem into Many, Many Small Pieces

Particle Systems are a great example.

1. Have one thread per *each particle*.
2. Put all of the initial parameters into an array in GPU memory.
3. Tell each thread what the current Time is.
4. Each thread then computes its particle's position, color, etc. and writes it into arrays in GPU memory.
5. The CPU program then initiates drawing of the information in those arrays.

Note: once setup, the data never leaves GPU memory!



Ben Weiss