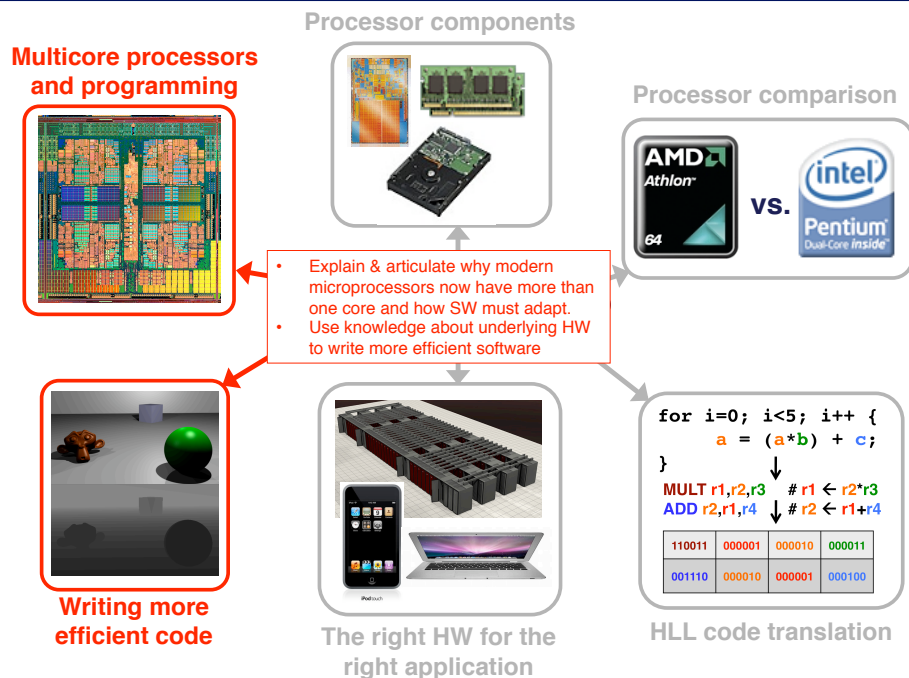


Lecture 26 GPU Wrap Up

Suggested Readings

- Readings
 - H&P: Chapter 7 – especially 7.1-7.8
 - (Over next 2 weeks)
 - Introduction to Parallel Computing
 - https://computing.llnl.gov/tutorials/parallel_comp/
 - POSIX Threads Programming
 - <https://computing.llnl.gov/tutorials/pthreads/>
 - How GPUs Work
 - www.cs.virginia.edu/~gfx/papers/pdfs/59_HowThingsWork.pdf



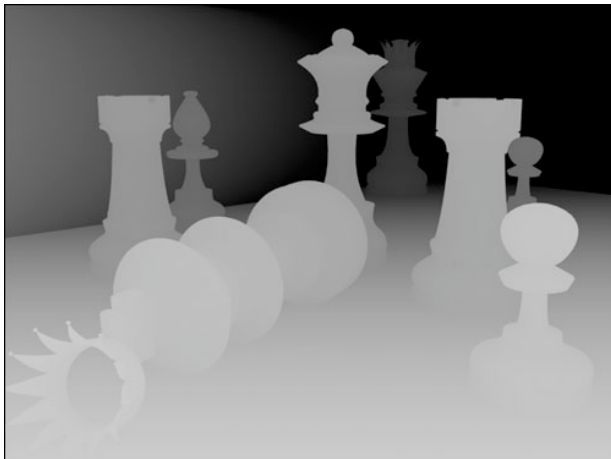
GPU discussion points

- Motivation for GPUs:
- Necessary processing
- Example problem:
 - Generic CPU pipeline
 - GPU-based vs. Uni-processor Z-buffer problem
- What does a GPU architecture look like?
 - Explain in context of SIMD
- Applicability to other computing problems

Recap: How is a frame rendered?

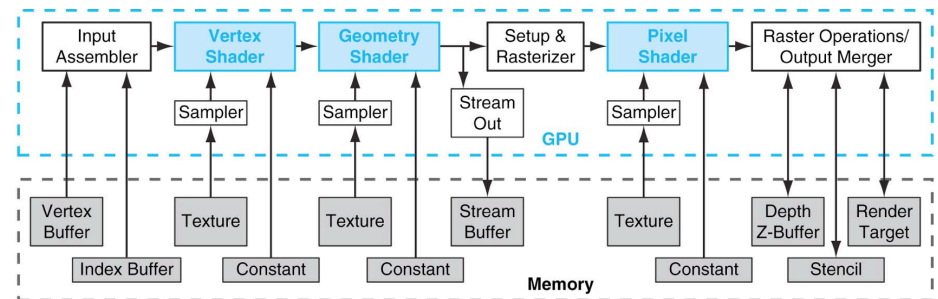
- Helpful to consider how the 2 standard graphics APIs – OpenGL and Direct 3D – work.
 - These APIs define a logical graphics pipeline that is mapped onto GPU hardware and processors – along with programming models and languages for the programmable stages
 - *In other words, API takes primitives like points, lines and polygons, and converts them into pixels*
- How does the graphics pipeline do this?
 - First, important to note that “pipeline” does not mean the 5 stage pipeline we talked about earlier
 - Pipeline describes sequence of steps to prepare image/scene for rendering

Example: Z-buffer



<http://blog.yoz.sk/examples/pixelBenderDisplacement/zbuffer1Map.jpg>

Recap: How is a frame rendered? (Direct3D pipeline)



GPUs

- GPU = Graphics Processing Unit
 - Efficient at manipulating computer graphics
 - Graphics accelerator uses custom HW that makes mathematical operations for graphics operations fast/efficient
 - Why SIMD? Do same thing to each pixel
- Often part of a *heterogeneous* system
 - GPUs don't do all things CPU does
 - Good at some specific things
 - i.e. matrix-vector operations
- GPU HW:
 - No multi-level caches
 - Hide memory latency with threads
 - To process all pixel data
 - GPU main memory oriented toward bandwidth

Circa 2008

NVIDIA Examples: GeForce 8800, GeForce 500 series

Late 2010, early 2011

- **NVIDIA GPU: Split into N “nodes” (N ~16)**
 - **Each node = “multiprocessor” (MP)**
 - **8800:** 16 MP @ 1.35 GHz
 - **500:** 16 MP @ 750 MHz?
- **“Multiprocessor” is NVIDIA term, but reasonable given our early definition**
- **Each MP = M streaming processors (SP) (M: 8–32)**
 - **8000:** 8 SP per MP
 - **500:** 32 SP per MP
- **SP consists of a floating point unit and integer unit**

Example NVIDIA architecture

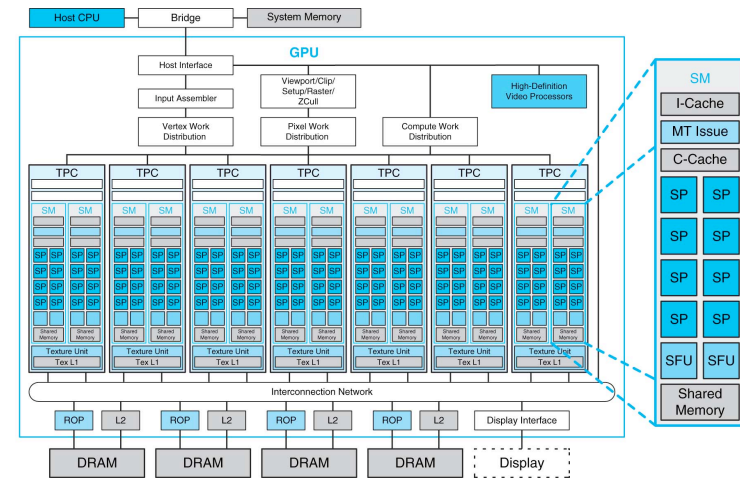


FIGURE A.2.5 Basic unified GPU architecture. Example GPU with 112 streaming processor (SP) cores organized in 14 streaming multiprocessors (SMs); the cores are highly multithreaded. It has the basic Tesla architecture of an NVIDIA GeForce 8800. The processors connect with four 64-bit-wide DRAM partitions via an interconnection network. Each SM has eight SP cores, two special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory. Copyright © 2009 Elsevier, Inc. All rights reserved.

Peak Performance:

- **8800 has floating point multiply-add instruction**
 - (see your project for utility of this instruction)
- **Peak performance obtained if all SP run in parallel:**

$$16 \text{ MP} \times \frac{8 \text{ SP}}{\text{MP}} \times \frac{2 \text{ FLOPs / instruction}}{\text{SP}} \times \frac{1 \text{ instruction}}{\text{clock}} \times \frac{1.35 \times 10^9 \text{ clocks}}{\text{s}} = \frac{345.6 \text{ GFLOPS}}{\text{s}}$$
- **What about 500?**
 - **More limited knowledge about HW, but assume same multiply-add instruction:**

$$16 \text{ MP} \times \frac{32 \text{ SP}}{\text{MP}} \times \frac{2 \text{ FLOPs / instruction}}{\text{SP}} \times \frac{1 \text{ instruction}}{\text{clock}} \times \frac{0.75 \times 10^9 \text{ clocks}}{\text{s}} = \frac{768 \text{ GFLOPS}}{\text{s}}$$
- **Speedup ~ 2.22**
 - **Consistent with Moore’s Law:**
 - **500 = 40 nm technology, 8800 = 65 nm technology**
 - **Should see 2X improvement per technology generation**

Cache and Memory in NVIDIA GPU

- **On 8800, each MP just 16 Kbytes of cache**
 - **Shared by 8 SPs**
- **Also, each MP = 8192 registers**
 - **512 per SP**
- **Memory:**
 - **768 Mbytes DRAM @ 900 MHz**
 - **DRAM is wide**
 - **8 Bytes**
 - **Want >> bandwidth given parallel nature of GPU**
 - **Often DRAM components specific to GPU**
 - **How much data can memory interface deliver to GPU?**

$$6 \text{ Partitions} \times \frac{8 \text{ Bytes}}{\text{Transaction}} \times \frac{2 \text{ Transactions}}{\text{clock}} \times \frac{0.9 \times 10^9 \text{ clocks}}{\text{s}} = \frac{86.4 \text{ GBytes}}{\text{s}}$$

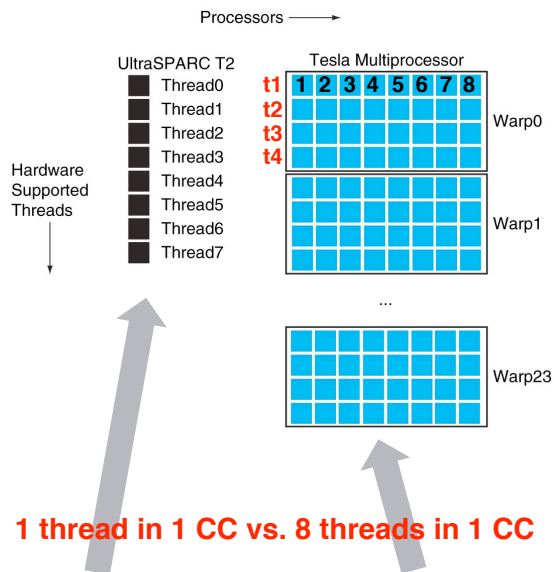
Data consumption vs. Data delivery

- What if each FLOP required 1 or 2 data words?
 - (Assume data word = 32 bits OR 4 bytes)
- What data delivery rate is required?

$$\frac{[4,8] \text{ Bytes}}{\text{FLOP}} \times \frac{345.6 \text{ GFLOPS}}{s} = \frac{1382.4 \text{ GBytes}}{s} \text{ or } \frac{2764.8 \text{ GBytes}}{s}$$

- But, our memory interface can only deliver 1/16th or 1/32th of this...
 - How do we mask memory latency?

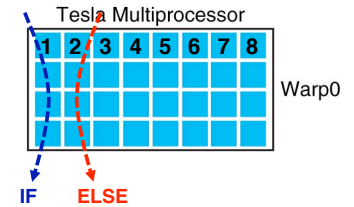
GPU vs. (Fine Grain) Threaded single core



- T2 can switch threads each CC
- 8800 supports 24 warps
 - Can switch warps every 2 or 4 CCs

Threads mask memory latencies

- Each SP supports HW-based threads
- In 8800, MP manages group of 32 threads called a *warp*
 - Ideally:
 - All 32 threads execute the same instruction on different data
 - True SIMD!
 - All 8 SPs busy each CC...
 - More realistically:
 - Don't always get true SIMD
 - Why?
 - Problem not 100% parallelizable
 - Still need to deal with conditions, etc. (i.e. BEQ...)
 - Interesting note:
 - No branch prediction HW
 - Just assign each execution path to a thread and pick 1!



GPUs

- GPU = Graphics Processing Unit
 - Efficient at manipulating computer graphics
 - Graphics accelerator uses custom HW that makes mathematical operations for graphics operations fast/efficient
 - Why SIMD? Do same thing to each pixel

Programming GPUs

- API language compilers target industry standard intermediate languages instead of machine instructions
 - GPU driver software generates optimized GPU-specific machine instructions
- Also, SW support for non graphics programming:
 - NVIDIA has graphics cards that support API extension to C – CUDA (“Computer Unified Device Architecture”)
 - Allows specialized functions from a normal C program to run on GPU’s stream processors
 - Allows C programs that can benefit from integrated GPU(s) to use where appropriate, but also leverage conventional CPU


GPUs for other problems

- More recently:
 - GPUs found to be more efficient than general purpose CPUs for many complex algorithms
 - Often things with massive amount of vector ops
 - Example:
 - ATI, NVIDIA team with Stanford to do GPU-based computation for protein folding
 - Found to offer up to 40 X improvement over more conventional approach

A bit more on CUDA...

- Developed by NVIDIA to program GPU processors
- Unified C/C++ programming for heterogeneous CPU-GPU system
 - Runs on CPU, sends work to GPU
 - Involves data transfer from main memory & “thread dispatch”
 - “Thread dispatch” is piece of program for GPU
 - Programmer can specify # of threads/block + number of blocks to run on GPU
 - Can make threads within block share same local memory
 - » Therefore communicate with loads and stores
 - CUDA compiler allocates registers to threads

GPU Example

<p>Power Efficient Supercomputing</p> <p>William Dally Bell Professor of Engineering, Stanford University Chief Scientist and Sr. VP of Research, NVIDIA</p> 	<p>High-Performance Computing is Power Limited</p> <ul style="list-style-type: none"> • What can be put on a chip is limited by power, not area. • Cost of energy to run a cluster equals purchase cost in less than 18 months. • Cost of provisioning a machine room with adequate power is typically many times cost of cluster. • What matters now is μFLOP 	<p>GPU Computing</p> <ul style="list-style-type: none"> • GPUs are already much of the way to being efficient supercomputer nodes <ul style="list-style-type: none"> – Simple control – Explicit control of storage hierarchy – Large fraction of energy goes to FLOPs • Future GPUs will close the gap <ul style="list-style-type: none"> – More efficient instruction and data supply – Agile memory – Seamless integration into larger machines • 5GFLOPS/W and beyond – soon
<p>High performance computing is <i>power limited.</i></p>	<p>Power-Efficient Supercomputing: Goal</p> <ul style="list-style-type: none"> • 200pJ/FLOP (5GFLOPS/W) sustained • 25% of energy in FPU 	