GPUs & Technical Writing

CSE 471 Spring 2015
Mark Wyse
May 28, 2015

Technical Writing

- Disclaimer: I’m not a technical writing expert

- Planning – outline your paper, understand the purpose
- Clarity – don’t hide the message
- Brevity – be concise and efficient with words
- Revise & Edit
  - Read your paper before submission!

Graphics Processing Unit (GPU)

- Parallelism
- Execution Model
- Modern GPU Microarchitecture

GPU Overview

- Originally fixed-function processors for 2D/3D graphics
- Increasing Programmability (1990s-2006)
  - Pixel shader, vertex shader, transform & lighting
- GPGPU: General Purpose Computing on GPU
  - Top 500 supercomputers…

GPU Parallelism

- Focus on Data Parallelism
- Identical, Independent, Streaming computations
- Approaches to Data Parallelism
  - MIMD – high overhead, good for general purpose
  - SIMD – reduces overhead, but resource contention is an issue
  - SIMT – reduces overhead, reduces resource contention

Execution Model

- Single Instruction, Multiple Thread (SIMT)
- Identical, Independent work over multiple lockstep threads
- A GPU “core” has many SIMT “threads”
  - Groups are exposed to programmers
  - Each thread knows its location in N (<= 3) dimensional space
- Efficient Gather/Scatter operations
- Highly parallel

- In summary: multicore, multithreaded SIMT

GPU Microarchitecture (GCN)

- Lane = executes a single thread
  - Executes work-items
- SIMT unit = executes Lanes in lockstep
  - 16 Lanes per SIMT in GCN
  - Runs Wavefronts (64 work-items, 4 cycles)
  - 64 KB register state (compared to ~1KB on x86 CPU)
- GPU Core = group of SIMT units
  - 4 SIMT units per Core in GCN
  - 10 active Wavefronts per SIMT unit
  - 2560 active work-items per Core
- GPU Chip = collection of GPU Cores

GPUs & Memory

- Thousands of threads
  - Radeon R9 290X = 44 GPU Cores * 2560 work-items/core
  - = 112,640 active work-items (threads)!!!
- Use TLP to hide memory latency
- Streaming memory access

- Many threads, different access patterns & requirements than CPU – caches?

GPU Caches

- Similar L1 capacity per GCN Core as x86 processor…
- But, many thousands (instead of 1 or 2) threads!

- Objective: maximize throughput
  - Not to hide latency
  - No temporal locality; spatial locality barely exists

- L1: coalesce requests to same block by different work-items
  - i.e., streaming thread locality
  - Keep around long enough for a single hit
  - Reduce bandwidth to DRAM
- L2: DRAM staging buffer
  - Tolerate spikes in DRAM bandwidth