GPUs and CUDA programming

Lecture 12
April 10, 2025

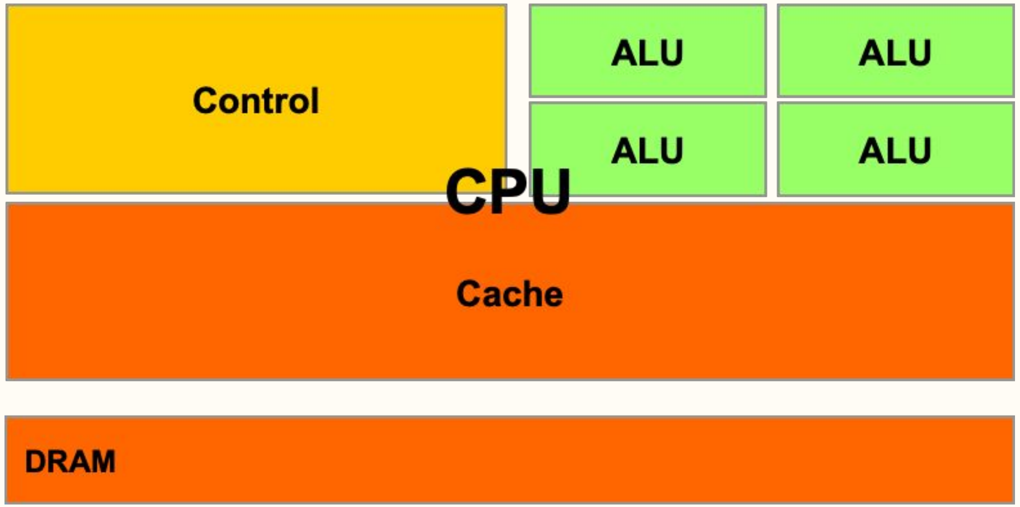
To Dos

Reading for next time (GPUs!)

Program 4 presentations

CPUs: Latency Oriented Design

- Optimize for single thread performance
- High clock frequency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency
- Powerful ALU
  - Reduced operation latency



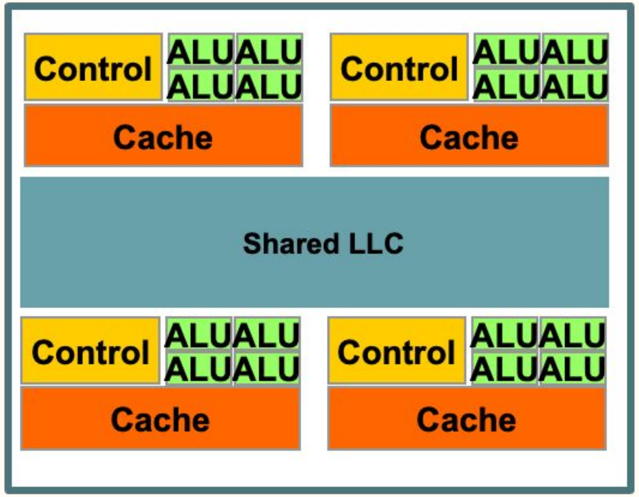
Latency Hiding Techniques

- Non-blocking caches
- Out-of-order execution
- Speculative execution
- Hyperthreading

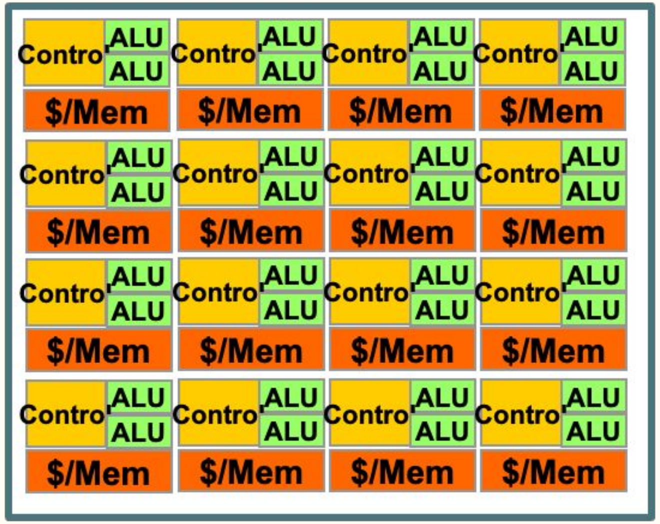
Why Switch To Multiprocessor Systems?

- Loss of ability to gain speed by shrinking technology
- Hard to transmit signal across entire chip in small clock cycle
- Difficult to verify increasingly complex chip designs
- Difficult to extract more instruction level parallelism from sequential applications

Move to Multi-Core and Many Core Systems



Multi-core



Many-Core

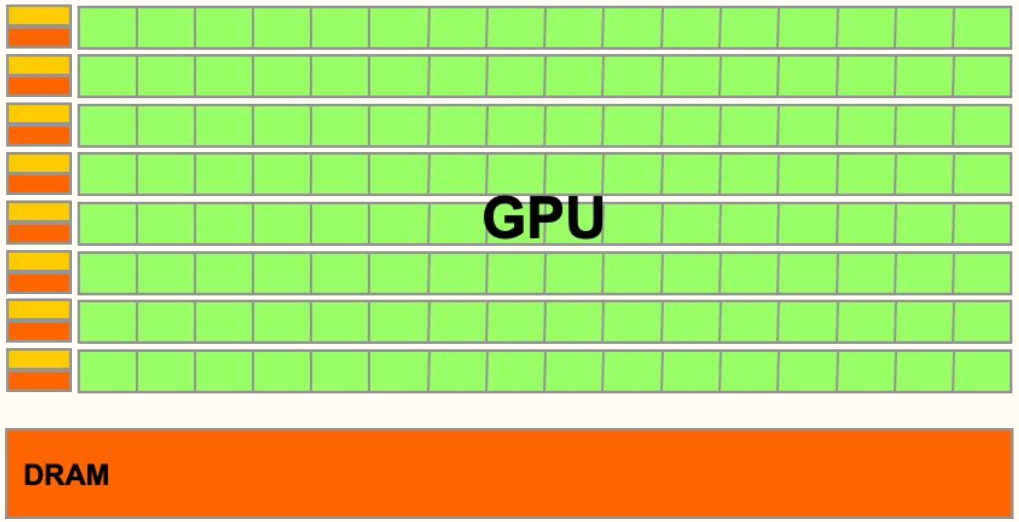
Concurrently, Multimedia/Graphics were Growing as Application Domains

Multimedia/Graphics applications

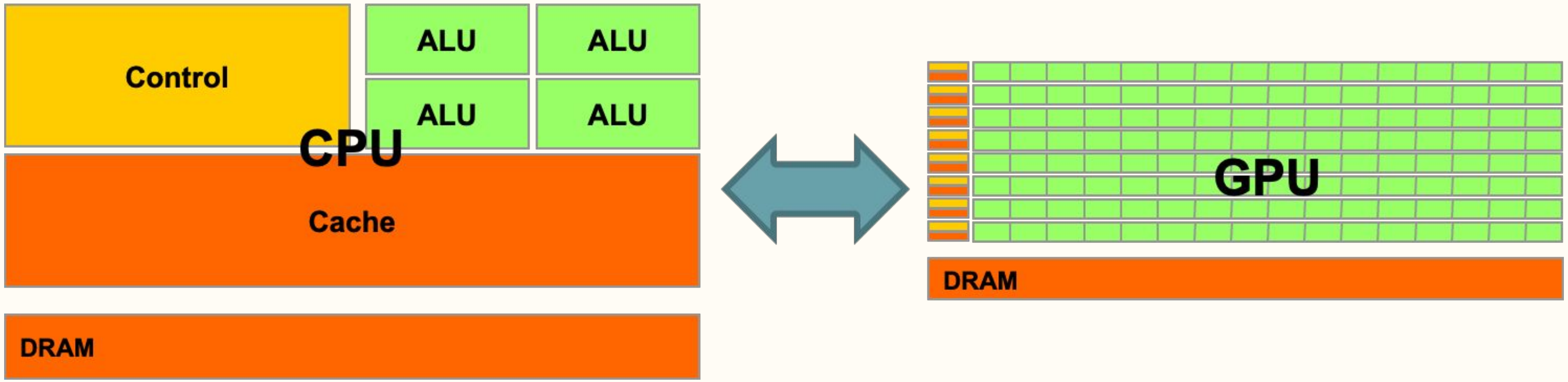
- Lots of data
- Lots of computation on independent groups of data (data parallelism)
- Latency less important than throughput
- In graphics programs, increased demand for programmability

GPUs: Throughput Oriented Design

- Moderate clock frequency
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies



Applications Benefit from Both CPU and GPU



CPUs for sequential parts where latency matters

- CPUs can be 10+X faster than GPUs for sequential code

GPUs for parallel parts where throughput wins

- GPUs can be 10+X faster than CPUs for parallel code

Writing Applications to use GPUs

Write code for CPU and for GPU

- CPU is in control
- CPU must transfer data to GPU device memory
- CPU invokes GPU kernel and typically waits for completion
- CPU transfers data from GPU device memory back to CPU memory

Developers job

- Must orchestrate movement of data between CPU and GPU
- Must orchestrate movement of data into special, fast, software managed memory
- Must orchestrate the collaboration of threads and the sharing of fast memory among threads

Issues That Impact GPU Performance

- Global Memory Bandwidth
    - Need to keep parallel compute resources fed with data
    - Need to have enough computation to use all resources and hide memory latency

Global Memory Bandwidth

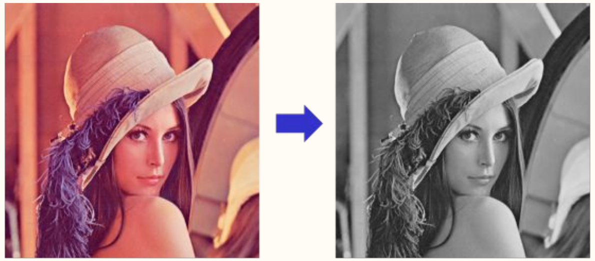
Ideal

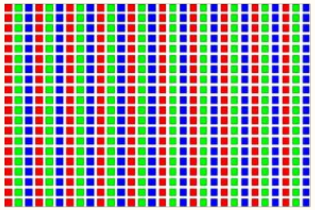
Reality

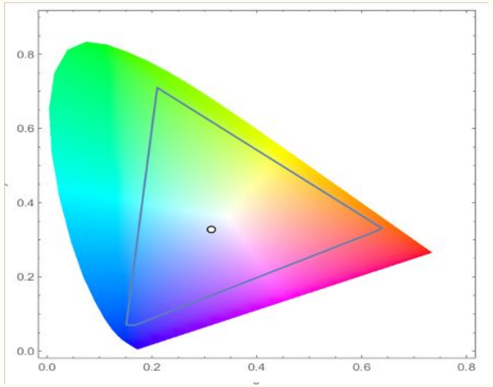
Issues That Impact GPU Performance

- Global Memory Bandwidth
  - Need to keep parallel compute resources fed with data
  - Need to have enough computation to use all resources and hide memory latency
- Need to prevent load imbalance
  - Need to decrease likelihood of divergent control paths
- Need to prevent serialization due to locks

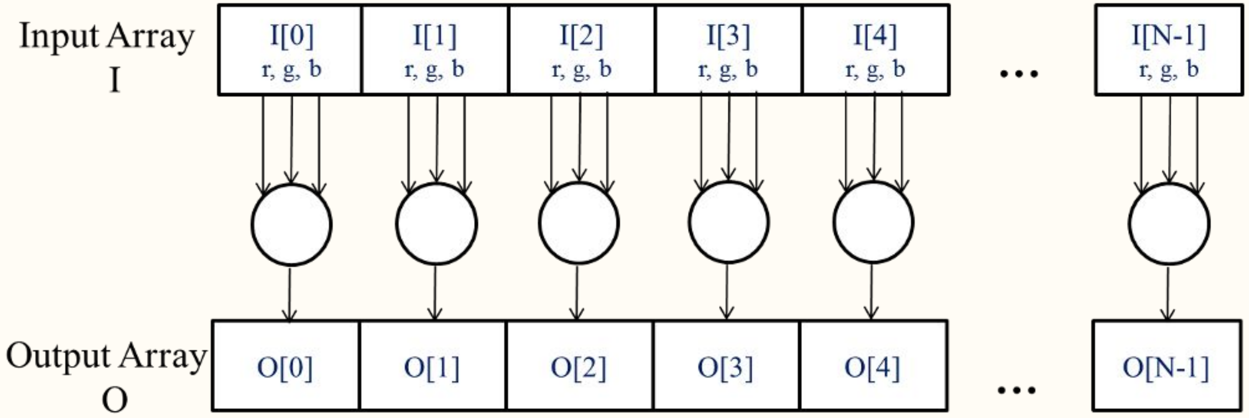
Conversion of a color image to grey–scale image







Pixels can be calculated independently

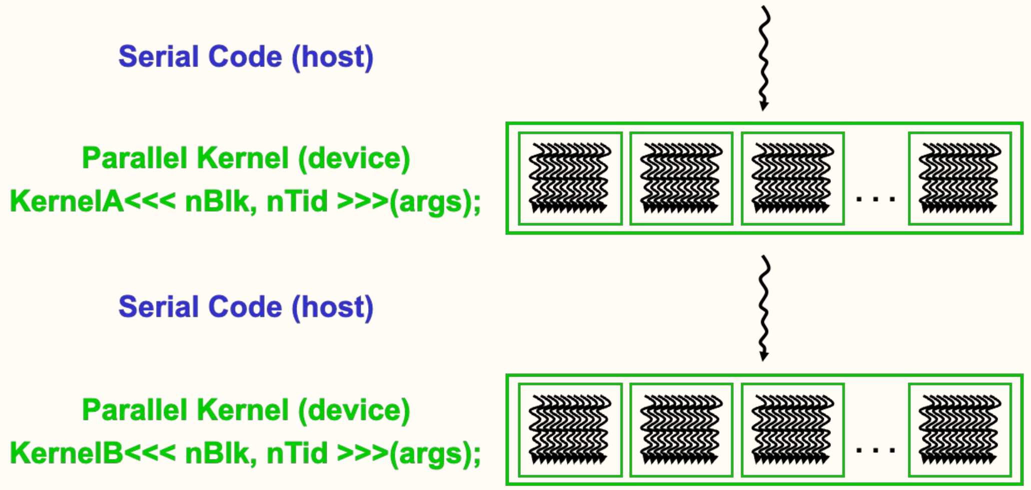


$$L = r * 0.21 + g * 0.72 + b * 0.07$$

CUDA/OpenCL – Execution Model

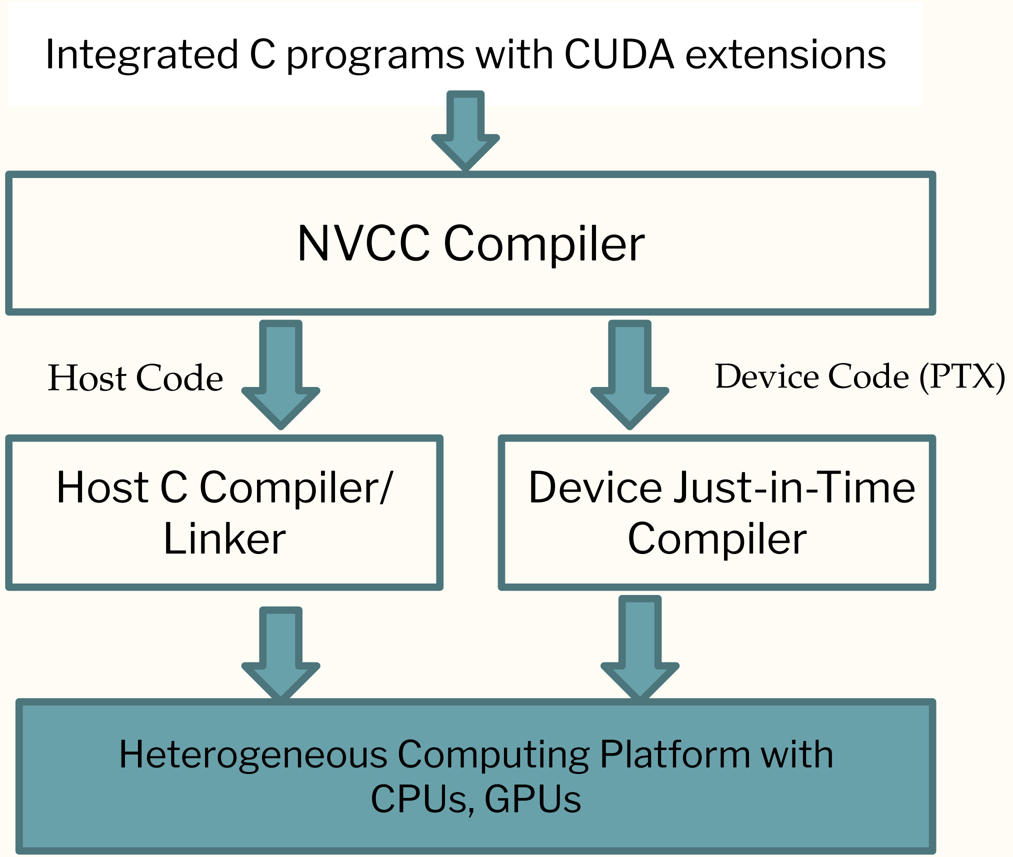
Integrated host+device app C program

- Serial or modestly parallel parts in host C code
- Highly parallel parts in device SPMD kernel C code



```
KernelA<<< nBlk, nTid >>>(args);
KernelB<<< nBlk, nTid >>>(args);
```

Compiling A CUDA Program


Integrated C programs with CUDA extensions
NVCC Compiler
Host Code
Device Code (PTX)
Host C Compiler/ Linker
Device Just-in-Time Compiler
Heterogeneous Computing Platform with CPUs, GPUs