CUDA

Introduction
Resources

- **Book**
  - “Programming Massively Parallel Processors”
    - David B. Kirk, Wen-mei W. Hwu

- **On-line**
  - Or
GPU History

- **GPU** – graphical processing unit
- Polygon pipelines / streaming processors
  - Multiple pipelines
  - Screen subdivision
- Surface enhancement through
  - Mapping
  - Vertex shaders
  - Pixel shaders
Texture Mapping
Bump / Displacement Mapping
Mapping and Shaders

- Mapping operations in graphics libraries
  - Transparency (multiplication), composition (addition, logical operators), multiresolution (summations, interpolations)

- Shaders – functions for dynamically altering polygons and pixels
  - Initially software only (ILM and Pixar)
  - Fixed hardware functionality
  - Limited programmability
GPGPU Programming

- Use graphical operations to perform scientific computing
- Limited but effective
- Works in contexts where full parallel programming not available (e.g. Javascript)
GPU Architecture

- Registers
- Warp Scheduler
- SFU
- Shared Memory
- Global Memory
- Constant Memory
nVIDIA GPU Architectures

- Tesla – 2008
- Fermi – 2010
- Kepler – 2012
- Maxwell – 2014
- Volta
CUDA

- **CUDA** – Compute Unified Device Architecture
- Data parallel problems
- Great for 1D, 2D, 3D regular grids
- Not true SIMD
  - SPMT – Single Program Multiple Threads
  - SIMT – Single Instruction Multiple Threads
Getting Started

- **Host** – CPU, main memory
- **Device** – GPU, GPU memory

- CUDA is like C/C++ with extensions
- **Kernel** – GPU function written in CUDA
- Source code placed in .cu files
- Nvidia compiler – **nvcc** – must be used
  - Link in runtime library **-lcudart**
NVCC

- Preprocessor strips out CUDA specific code and compiles it.
- Also creates necessary wrappers between CUDA and non-CUDA code
- Sends non-CUDA code to system compiler (default gcc)
Error Handling

- camelCase function names
- cudaError_t returned by most functions
  - cudaSuccess

- const char *
  cudaGetErrorString(cudaError_t)
Device Information

- `cudaGetDeviceCount(int * device_count)`
- Return values
  - `cudaErrorNoDevice`
  - `cudaErrorInsufficientDriver`
- `cudaDeviceProp` – holds device information
- `cudaGetDeviceProperties`
  ```c
  cudaDeviceProp *, int device);
  ```
Typical Program Structure

• Allocate data memory on GPU (device)
• Loop
  – Copy data from host (CPU) to device (GPU)
  – Compute on device
  – Copy output from device to host
• Free data memory on device
Device Management

- `cudaGetDevice(int * device)`

- `cudaSetDevice(int device)`
  - All memory management, kernels, will be on this device
Memory Management

- `cudaMalloc( void ** devPtr, size_t nbytes)`
  - Pass in a memory pointer which is altered
  - `CudaErrorMemoryAllocation` returned if malloc fails

- `cudaFree( void * devPtr )`
  - `cudaErrorInvalidDevicePointer` returned on failure
Memory Management

- cudaMemcpy(void * dst,
  const void * src,
  size_t count,
  enum cudaMemcpyKind kind)

- cudaMemcpyHostToDevice
- cudaMemcpyDeviceToDevice
- cudaMemcpyHostToDevice
- cudaMemcpyDeviceToDevice
- cudaMemcpyHostToHost
- cudaMemcpyDeviceToHost
- cudaMemcpyDeviceToDevice