NVIDIA GPU CODING & COMPUTING
WHY GPU’S?
ARCHITECTURE & PROGRAM MODEL
CPU v. GPU

- Control
- ALU
- ALU
- ALU
- Cache

CPU

- DRAM

GPU

- DRAM
Multiprocessor Model
Memory Model: Thread Level
Programing Model: Logical Mapping of Threads
Programing Model: Another Look
PROGRAMING MODEL:
KIRCHHOFF MIGRATION EXAMPLE
Program Model: Kirchhoff Migration Example

• See Moton, S. “Seismic Imaging on GPUs: Algorithms and Porting & Production Experiences,” Hess Corporation, Slides 15-19
EXECUTION MODEL
Execution Model: Mapping of Thread Blocks to HW
Execution Model: Thread Level

- C Program
  - Sequential Execution
    - Serial code
    - Parallel kernel Kernel0<<<>>>()
  - Serial code
  - Parallel kernel Kernel1<<<>>>()

- Device
  - Grid 0
    - Block (0, 0)
    - Block (1, 0)
    - Block (2, 0)
    - Block (0, 1)
    - Block (1, 1)
    - Block (2, 1)
  - Grid 1
    - Block (0, 0)
    - Block (1, 0)
    - Block (0, 1)
    - Block (1, 1)
    - Block (0, 2)
    - Block (1, 2)
CUDA Architecture
C/C++ EXTENSIONS
C/C++ Extensions: Function Type Qualifiers

• __device__

• __global__

• __host__

• There are use restrictions for these

C/C++ Extensions: Value Type Qualifiers

• __device__

• __constant__

• __shared__

• volatile

• There are use restrictions for these too.

C/C++ Extensions: Built-in Vector Types

- `char1, uchar1, char2, uchar2, ..., char4, uchar4`
- `short1, ushort1, ..., short4, ushort4`
- `int1, uint1, ..., int4, uint4`
- `long1, ulong1, ..., long4, ulong4`
- `longlong1, longlong2`
- `float1, ..., float4`
- `double1, double2`
- `dim3` (like a struct with 3 ints)

C/C++ Extensions: Built-in Variables

- gridDim
- blockIdx
- blockDim
- threadIdx
- warpSize

- There are restrictions for these

See NVIDIA_CUDA_Programming_Guide_2.3.pdf pg 110.
C/C++ Extensions: Memory Fence Functions

- __threadfence()
- __threadfence_block()

See NVIDIA_CUDA_Programming_Guide_2.3.pdf pages 110 and 111.
C/C++ Extensions: Synchronization Function

• `__syncthreads()`

See NVIDIA_CUDA_Programming_Guide_2.3.pdf pg 111.
C/C++ Extensions: Mathematical Functions

• See Appendix C

See NVIDIA_CUDA_Programming_Guide_2.3.pdf begins at pg 119.
C/C++ Extensions: Other Functions

- Texture Functions
- Time Function
- Atomic Functions
- Bitwise Functions

See NVIDIA_CUDA_Programming_Guide_2.3.pdf begins at pg 113.
CUDA KERNEL EXAMPLES
CUDA Kernel Examples

• See CUDA SDK and Tim’s “Hello World” examples, and the CUDA Programming Guide v. 2.3
PERFORMANCE GUIDELINES:
GLOBAL MEMORY ACCESS
CUDA RUNTIME API
CUDA Runtime API

- Allocate Memory
- Free Memory
- Copy Memory
- Set Memory
- Texture Functions
- etc ...

- See CUDA SDK examples and CudaReferenceManual.pdf for more
CUDA DRIVER API
Allocate Memory
Free Memory
Copy Memory
Set Memory
Texture Functions
etc ... 

See CUDA SDK examples and CudaReferenceManual.pdf for more
PERFORMANCE GUIDELINES:
GLOBAL MEMORY ACCESS
# Global Memory Access: Coalesced Read/Write

## Random Single Coalesced Read/Write

- Thread 0
- Thread 1
- Thread 2
- Thread 3
- Thread 4
- Thread 5
- Thread 6
- Thread 7
- Thread 8
- Thread 9
- Thread 10
- Thread 11
- Thread 12
- Thread 13
- Thread 14
- Thread 15

## Missaligned Single Coalesced Read/Write

- Address 120
- Address 124
- Address 128
- Address 132
- Address 136
- Address 140
- Address 144
- Address 148

## Two Coalesced Reads/Writes

- Address 96
- Address 100
- Address 104
- Address 108
- Address 112
- Address 116
- Address 120
- Address 124

---

### Diagram Details

- **32B segment**: 128 addresses
- **64B segment**: 256 addresses
- **128B segment**: 512 addresses
- **256B segment**: 1024 addresses

---

- **Thread 0**: Addresses 120, 124, 128, 132, 136, 140, 144, 148
- **Thread 1**: Addresses 152, 156, 160, 164, 168, 172, 176, 180
- **Thread 2**: Addresses 184, 188, 192, 196, 200, 204, 208, 212

---

- **Random Single Coalesced Read/Write**
- **Missaligned Single Coalesced Read/Write**
- **Two Coalesced Reads/Writes**
PERFORMANCE GUIDELINES: SHARED MEMORY ACCESS
Shared Memory Access: NO Bank Conflicts
Shared Memory Access: NO Bank Conflicts
Shared Memory Access: With Bank Conflicts
Shared Memory Access: Broadcasting (NO Conflict*)

*Possible two way back conflict
PERFORMANCE GUIDELINES:
OTHER STUFF
Performance Guidelines: Other Stuff

• Try to avoid divergent warps
• Trade "Precision for Speed"
• Bitwise Tricks (i.e. \(x/4 \iff x \gg 2\))

• See CUDA Programming Guide v. 2.3 for more

See NVIDIA_CUDA_Programming_Guide_2.3.pdf pages 77-100.
PERFORMANCE GUIDELINES:
KIRCHHOFF MIGRATION EXAMPLE
Performance Guidelines: Kirchhoff Migration Example

DEBUGGING
Debugging

- Device Emulation Mode
- `gdb`
- Threads Have Unique IDs
- CUDA API Functions Return Error Statuses

See NVIDIA_CUDA_Programming_Guide_2.3.pdf pages 44 and 45.
PROFILING
Profiling

• $ export CUDA_PROFILE=1

• CUDA GI Profiler

• CUDA Simple Text Based Profiler
  • cuda_profile.log in cwd/exec-dir

• See Cuda_Profiler_2.3.txt
RELEVANT/IMPORTANT TOPICS THAT WERE NOT COVERED
Topics Not Covered (at least not in great detail)

• Texture Memory and Fetching
• CUDA API’s
• Asynchronous Concurrent Execution
  • Streams
  • Events
• Technical Specifications
  • Compute Capability
  • Intrinsic Functions
• And Other Stuff . . .
END