OUTLINE

- Architecture:
  - Kepler/Maxwell/Pascal/Volta
- Kernel optimizations
  - Launch configuration
- Part 2 (this session):
  - Global memory throughput
  - Shared memory access

Most concepts in this presentation apply to any language or API on NVIDIA GPUs
GLOBAL MEMORY THROUGHPUT
MEMORY HIERARCHY REVIEW

- Local storage
  - Each thread has own local storage
  - Typically registers (managed by the compiler)

- Shared memory / L1
  - Program configurable: typically up to 48KB shared (or 64KB, or 96KB...)
  - Shared memory is accessible by threads in the same threadblock
  - Very low latency
  - Very high throughput: >1 TB/s aggregate
MEMORY HIERARCHY REVIEW

- L2
  - All accesses to global memory go through L2, including copies to/from CPU host
- Global memory
  - Accessible by all threads as well as host (CPU)
  - High latency (hundreds of cycles)
  - Throughput: up to ~900 GB/s (Volta V100)
MEMORY ARCHITECTURE
MEMORY HIERARCHY REVIEW
GMEM OPERATIONS

- Loads:
  - Caching
    - Default mode
    - Attempts to hit in L1, then L2, then GMEM
    - Load granularity is 128-byte line

- Stores:
  - Invalidate L1, write-back for L2
GMEM OPERATIONS

- Loads:
  - Non-caching
    - Compile with `-Xptxas -dlcm=cg` option to nvcc
    - Attempts to hit in L2, then GMEM
      
      Do not hit in L1, invalidate the line if it’s in L1 already
    - Load granularity is 32-bits

We won’t spend much time with non-caching loads in this training session
LOAD OPERATION

- Memory operations are issued *per warp* (32 threads)
  - Just like all other instructions
- Operation:
  - Threads in a warp provide memory addresses
  - Determine which lines/segments are needed
  - Request the needed lines/segments
CACHING LOAD

- Warp requests 32 aligned, consecutive 4-byte words
- Addresses fall within 1 cache-line
  - Warp needs 128 bytes
  - 128 bytes move across the bus on a miss
  - Bus utilization: 100%
- int c = a[idx];

addresses from a warp

Memory addresses

0  32  64  96  128  160  192  224  256  288  320  352  384  416  448
CACHING LOAD

- Warp requests 32 aligned, permuted 4-byte words
- Addresses fall within 1 cache-line
  - Warp needs 128 bytes
  - 128 bytes move across the bus on a miss
  - Bus utilization: 100%
- `int c = a[rand()%warpSize];`

addresses from a warp

![Memory addresses diagram](image-url)
CACHING LOAD

- Warp requests 32 misaligned, consecutive 4-byte words
- Addresses fall within 2 cache-lines
  - Warp needs 128 bytes
  - 256 bytes move across the bus on misses
  - Bus utilization: 50%
  - int c = a[dx-2];
CACHING LOAD

- All threads in a warp request the same 4-byte word
- Addresses fall within a single cache-line
  - Warp needs 4 bytes
  - 128 bytes move across the bus on a miss
  - Bus utilization: 3.125%
- int c = a[40];

addresses from a warp
CACHING LOAD

- Warp requests 32 scattered 4-byte words
- Addresses fall within N cache-lines
  - Warp needs 128 bytes
  - N*128 bytes move across the bus on a miss
  - Bus utilization: \( \frac{128}{N \times 128} \) (3.125% worst case N=32)
  - int c = a[rand()];

addresses from a warp

Memory addresses
NON-CACHING LOAD

- Warp requests 32 scattered 4-byte words
- Addresses fall within N segments
  - Warp needs 128 bytes
  - N*32 bytes move across the bus on a miss
  - Bus utilization: \( \frac{128}{N \times 32} \) (12.5% worst case \( N = 32 \))
- \( \text{int } c = a[\text{rand()}]; \quad \text{-Xptxas -dlcm=cg} \)

addresses from a warp

Memory addresses
GMEM OPTIMIZATION GUIDELINES

- Strive for perfect coalescing
  - (Align starting address - may require padding)
  - A warp should access within a contiguous region
- Have enough concurrent accesses to saturate the bus
  - Process several elements per thread
    - Multiple loads get pipelined
    - Indexing calculations can often be reused
  - Launch enough threads to maximize throughput
    - Latency is hidden by switching threads (warps)
- Use all the caches!
SHARED MEMORY

- Uses:
  - Inter-thread communication within a block
  - Cache data to reduce redundant global memory accesses
  - Use it to improve global memory access patterns

- Organization:
  - 32 banks, 4-byte wide banks
  - Successive 4-byte words belong to different banks
SHARED MEMORY

- Performance:
  - Typically: 4 bytes per bank per 1 or 2 clocks per multiprocessor
  - shared accesses are issued per 32 threads (warp)
  - serialization: if $N$ threads of 32 access different 4-byte words in the same bank, $N$ accesses are executed serially
  - multicast: $N$ threads access the same word in one fetch
    - Could be different bytes within the same word
BANK ADDRESSING EXAMPLES

No Bank Conflicts

No Bank Conflicts
BANK ADDRESSING EXAMPLES

2-way Bank Conflicts

16-way Bank Conflicts
SHARED MEMORY: AVOIDING BANK CONFLICTS

- **32x32** SMEM array

- Warp accesses a column:
  - 32-way bank conflicts (threads in a warp access the same bank)
SHARED MEMORY: AVOIDING BANK CONFLICTS

- Add a column for padding:
  - 32x33 SMEM array

- Warp accesses a column:
  - 32 different banks, no bank conflicts
SUMMARY

- Kernel Launch Configuration:
  - Launch enough threads per SM to hide latency
  - Launch enough threadblocks to load the GPU

- Global memory:
  - Maximize throughput (GPU has lots of bandwidth, use it effectively)

- Use shared memory when applicable (over 1 TB/s bandwidth)

- Use analysis/profiling when optimizing:
  - “Analysis-driven Optimization” (future session)
FUTURE SESSIONS

- Atomics, Reductions, Warp Shuffle
- Using Managed Memory
- Concurrency (streams, copy/compute overlap, multi-GPU)
- Analysis Driven Optimization
- Cooperative Groups
FURTHER STUDY

- Optimization in-depth:

- Analysis-Driven Optimization:

- CUDA Best Practices Guide:

- CUDA Tuning Guides:
  - https://docs.nvidia.com/cuda/index.html#programming-guides
  (Kepler/Maxwell/Pascal/Volta)
HOMEWORK

- Log into Summit (ssh username@home.ccs.ornl.gov -> ssh summit)

- Clone GitHub repository:
  - Git clone git@github.com:olcf/cuda-training-series.git

- Follow the instructions in the readme.md file:

- Prerequisites: basic Linux skills, e.g. ls, cd, etc., knowledge of a text editor like vi/emacs, and some knowledge of C/C++ programming
QUESTIONS?