GPUs and Heterogeneous Systems
(programming models and architectures)

Performance considerations

Optimizing the performance of CUDA kernels

- Accelerating a function in CUDA is pretty simple

- Optimizing the performance of the accelerated function is a hard task
  - It is necessary to “tailor” and tune the kernel code based on the underlying architecture
  - This is a profile-driven activity
    - Identify the bottleneck and work on it!

- This presentation presents an overview of possible optimizations

Maximizing occupancy

- **Benefit to compute cores**
  - More work to hide long-latency operations
- **Benefit to memory**
  - More parallel memory accesses to hide memory latency
- **Strategies**
  - Tuning usage of streaming multiprocessors’ resources (threads per block, shared memory per block, registers per thread)

Enabling coalesced global memory accesses

- **Benefit to compute cores**
  - Fewer stalls waiting for global memory accesses
- **Benefit to memory**
  - Less global memory traffic and better utilization of caches
- **Strategies**
  - Transferring data from global memory to shared memory in a coalesced way and accessing shared memory in a uncoalesced way
  - Rearranging thread to data mapping
  - Rearranging the layout of the data

Minimizing control divergence

- **Benefit to compute cores**
  - High SIMT efficiency
- **Benefit to memory**
  - N/A
- **Strategies**
  - Rearranging thread to data/work mapping
  - Rearranging the layout of the data

Tiling of reused data

- **Benefit to compute cores**
  - Fewer stalls waiting for global memory accesses
- **Benefit to memory**
  - Less global memory traffic
- **Strategies**
  - Placing data that is reused within a block in shared memory or registers so that it is transferred from global memory only once

Privatization

- **Benefit to compute cores**
  - Fewer stalls waiting for atomic updates
- **Benefit to memory**
  - Less contention and serialization of atomic updates
- **Strategies**
  - Applying partial updates to a private copy of the data and then updating the final result

Thread coarsening

- **Benefit to compute cores**
  - Less redundant work, divergence, or synchronization
- **Benefit to memory**
  - Less global memory traffic
- **Strategies**
  - Assigning multiple chunks of data to be processed to each thread to reduce the price of parallelism when it is incurred unnecessary

Other instruction-level / architecture specific optimizations

- Properly selecting data types
- Use intrinsic functions offering higher performance at a reduced result accuracy
- Follow guidelines for the specific GPU family

References

- Slides mainly based on:
  - W.-m. W. Hwu , D. B. Kirk, I. El Hajj, **Programming Massively Parallel Processors: A Hands-on Approach**, **Chapter 6**
  - J. Chen, M. Grossman, T. McKercher, **Professional Cuda C Programming**, **Chapters 3, 4, 5, 7**