Advanced GPU Programming

Advanced GPU Programming

Learning Objectives

1. Describe GPU optimization techniques
2. Prepare for advanced technique materials

Applications Suitable for GPUs

1. Handle massive parallel data processing
2. Low Dominance in Host-Device Communication Costs
3. Coalesced Data Access (Global Memory Coalescing)

Profiling

1. Identify hotspots of applications
2. Measure key performance factors
   - Reported throughput
   - **of divergent branches**
   - **of divergent memory accesses (coalesced/uncoalesced)**
   - Occupancy
   - Memory bandwidth utilizations
3. Use vendor-provided profiler to profile kernels

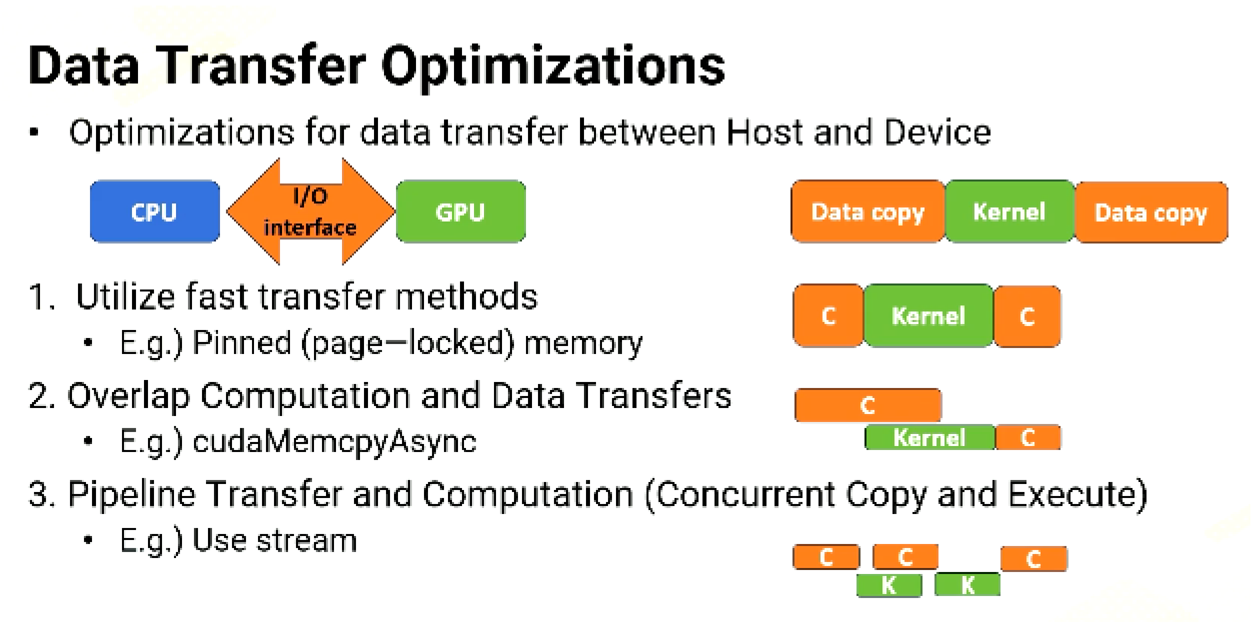
Optimization Techniques

1. Overall execution time = data transfer time + compute time + memory access time
2. Data transfer time optimizations
3. Memory access pattern optimizations
4. Computation overhead reduciton optimizations
5. Utilize libraries

Data Transfer Optimizations

1. Optimizations for data transfer between host and device
2. Utilize fast transfer methods
   - Pinned (page-locked) memory
3. Overlap computation and data transfer
   - cudaMemcpyAsync
4. Pipeline transfer and computation (concurrent copy and execute)
   - Use stream
5. Direct host memory access
   - Zero copy: access CPU data directly in the CPU and GPU integrated memory
   - Unified virtual addressing (UVA)
     - Driver/runtime system hides the physically separated memory spaces and provides an interface as if CPU and GPU can access any memory

– Ideally no data copy cost but the implementation still requires data copy and the overhead exists



Data Transfer Optimizations

- Optimizations for data transfer between Host and Device

CPU
I/O interface
GPU
Data copy
Kernel
Data copy
C
Kernel
C
C
Kernel
C
C
C
C
K
K

1. Utilize fast transfer methods
   - E.g.) Pinned (page–locked) memory
2. Overlap Computation and Data Transfers
   - E.g.) cudaMemcpyAsync
3. Pipeline Transfer and Computation (Concurrent Copy and Execute)
   - E.g.) Use stream

Data Transfer Optimizations

Memory Access Pattern Optimizations

1. Utilize caches
2. Global memory coalescing accesses
3. Aligned global memory accesses
4. Reducing the number of DRAM memory transactions is the key
5. Check the average memory bandwidth
6. Reduce shared memory bank conflicts

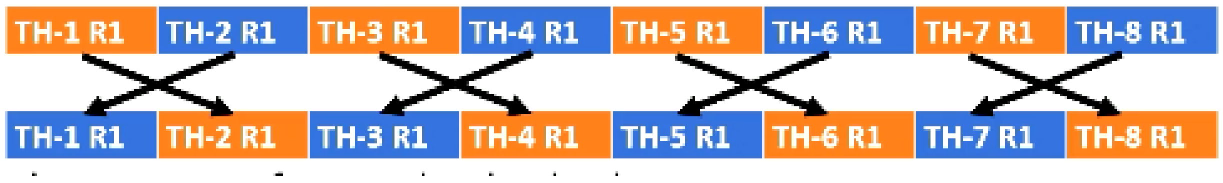
Reduce Computation Overhead

1. Instruction level optimizations
   - Replace with low-cost instructions (use shift operations instead of multiplication or divisions)
2. Use low precisions when possible (use fewer number of bits)
3. Use special hardware built-in functions (rsqrtf)
4. Utilize math libraries
5. Reduce branch statements
   - Use predicated execution if possible
6. Avoid atomic operations if possible
7. Use tensor operations and utilize tensor cores

Newer CUDA Features

1. Warp-level operations: warp shuffle/vote/ballot
   - Communication between threads is expensive
   - Register files are unique to threads
   - Need to use shred memory which is expensive
   - Warp-level operations allow data movement within a warp
   - Hardware support for warp-level reduction

2. Cooperative groups to utilize different formations of warps
3. Instead of fixed warp size, smaller warp-level operations are allowed
    - NVIDIA Volta's independent thread scheduling
4. New features
    - Depature from SIMT execution is allowed
        - Each thread could have different PCs, different thread group execution
        - Shared memory no longer belongs to one SM
    - Unified memory for shared memory, L1 data cache, and texture memory
    - Hardware acceleration for split arrive/wait barrier



Warp-level Operations

CUDA Libraries

1. Continuously developed
2. Linear Algebra applications: cuBLAS
3. Sparse matrices: cuSPARSE
4. Tensor based linear algebra: cuTENSOR
5. Parallel algorithm libraries: Thrust
6. Communication libraries: target for multi GPUs
7. NCCL, NVSHMEM
8. Deep learning libraries: cuDNN

Other GPU Programming Platforms

1. Many other programming languages, platforms, and libraries exist
    - OpenCL
    - OneAPI/SysCL
    - OpenACC
    - HIP
    - Kokkos
    - Python
    - Julia
    - more

Summary

1. GPU programming optimizations focus on:
    - Reducing compute amount
    - Reducing data transfer overhead from host
    - Ensuring efficient memory access patterns in global and shared memory