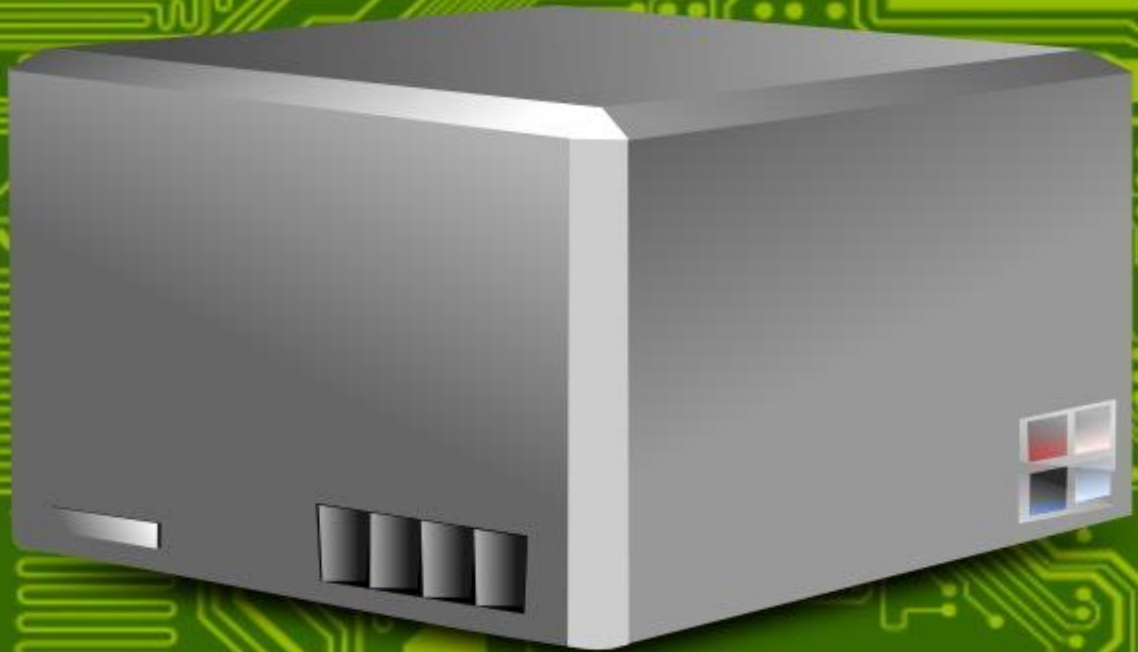
GPU Computing



CUDA 程序优化

Hui Liu
Email: hui.sc.liu@gmail.com

优化内存使用

基本策略

- 计算 vs 内存访问
  - Processing data is cheaper than moving it around
  - Especially for GPUs as they devote many more transistors to ALUs than memory

- And will be increasingly so
  - The less memory bound a kernel is, the better it will scale with future GPUs

- So you want to:
  - Maximize use of low-latency, high-bandwidth memory
  - Optimize memory access patterns to maximize bandwidth
  - Leverage parallelism to hide memory latency by overlapping memory accesses with computation as much as possible
    - Kernels with high arithmetic intensity (ratio of math to memory transactions)
  - Sometimes recompute data rather than cache it

极小化 CPU ↔ GPU 数据传输

- CPU ↔ GPU memory bandwidth much lower than GPU memory bandwidth
  - Use page-locked host memory (`cudaMallocHost()`) for maximum CPU ↔ GPU bandwidth
    - Be cautious however since allocating too much page-locked memory can reduce overall system performance
- Minimize CPU ↔ GPU data transfers by moving more code from CPU to GPU
  - Even if that means running kernels with low parallelism computations
  - Intermediate data structures can be allocated, operated on, and deallocated without ever copying them to CPU memory
- Group data transfers
  - One large transfer much better than many small ones

极大化使用共享内存

- Shared memory is hundreds of times faster than global memory
- Threads can cooperate via shared memory
  - Not so via global memory
- Common CUDA kernel structure:
  1. Load data from global memory to shared memory
  2. __syncthreads()
  3. Process the data in shared memory with many threads
  4. __syncthreads() (if needed)
  5. Store results from shared memory to global memory
  - Notes:
    - Steps 2-4 may be repeated, looped, etc.
    - Step 4 is not necessary if there is no dependence of stored data on other threads

优化内存使用模式

- Effective bandwidth can vary by an order of magnitude depending on access pattern

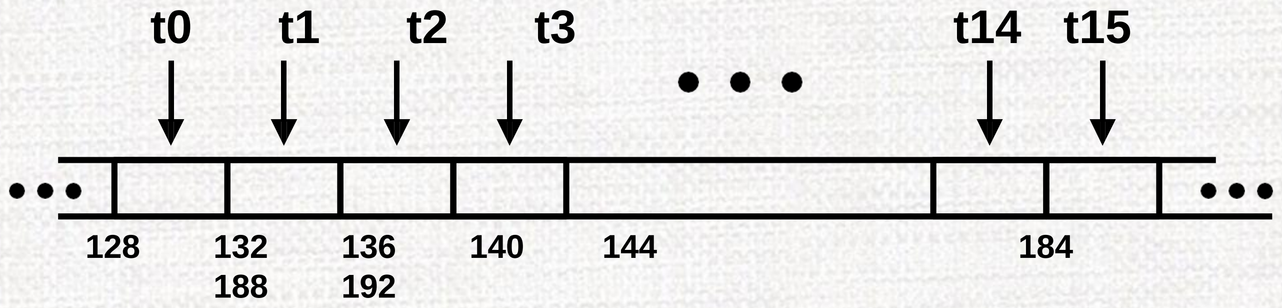
- Optimize access patterns to get:
  - *Coalesced* global memory accesses
  - Shared memory accesses with *no or few bank conflicts*
  - *Cache-efficient* texture memory accesses
  - *Same-address* constant memory accesses

全局内存: 对齐与合并访问

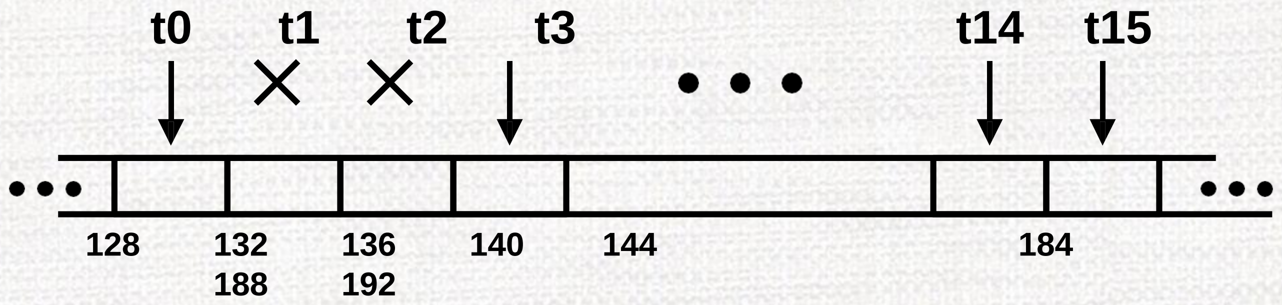
- The simultaneous global memory accesses by each thread of a half-warp during the execution of a single read or write instruction will be *coalesced* into a single access if:
  - The size of the memory element accessed by each thread is either 4, 8, or 16 bytes
  - The elements form a contiguous block of memory
  - The Nth element is accessed by the Nth thread in the half-warp
  - The address of the first element is aligned to 16 times the element's size

- Coalescing happens even if some threads do not access memory (divergent warp)

全局内存合并访问



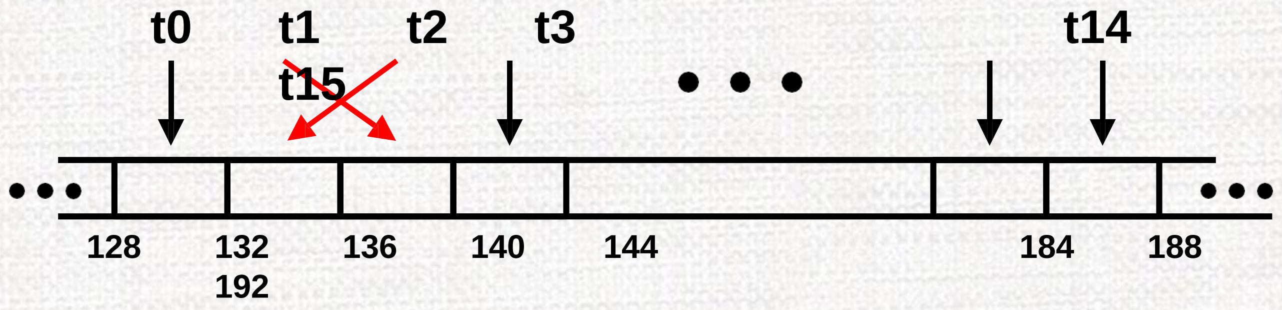
Coalesced `float` memory access



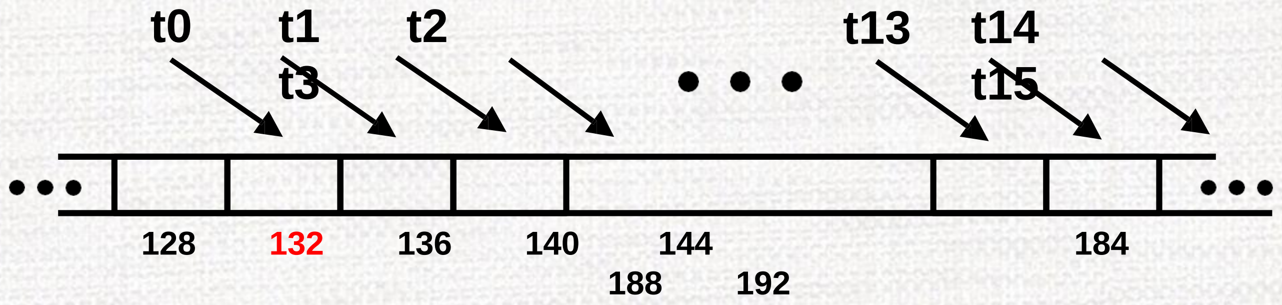
Coalesced `float` memory access

(divergent warp)

非合并全局内存访问

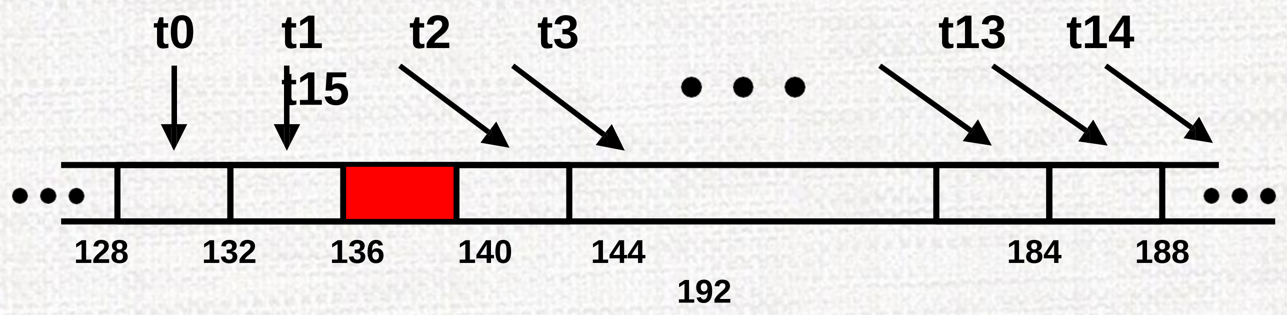


Non-sequential `float` memory access

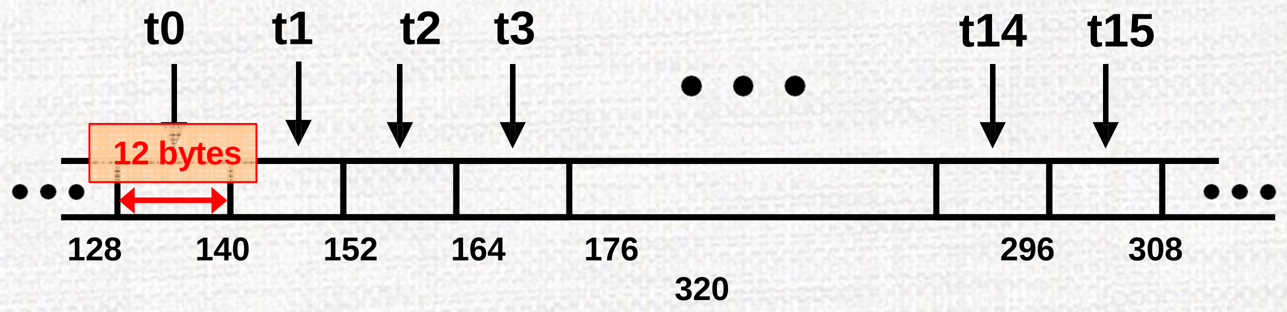


Misaligned starting address

非合并全局内存访问



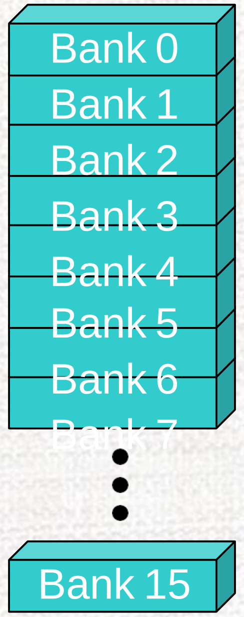
Non-contiguous `float` memory access



Non-coalesced `float3` memory access

共享内存

- In a parallel machine, many threads access memory
  - Therefore, memory is divided into *banks*
  - Essential to achieve high bandwidth
- Each bank can service one address per cycle
  - A memory can service as many simultaneous accesses as it has banks
- Multiple simultaneous accesses to a bank result in a *bank conflict*
  - Conflicting accesses are serialized

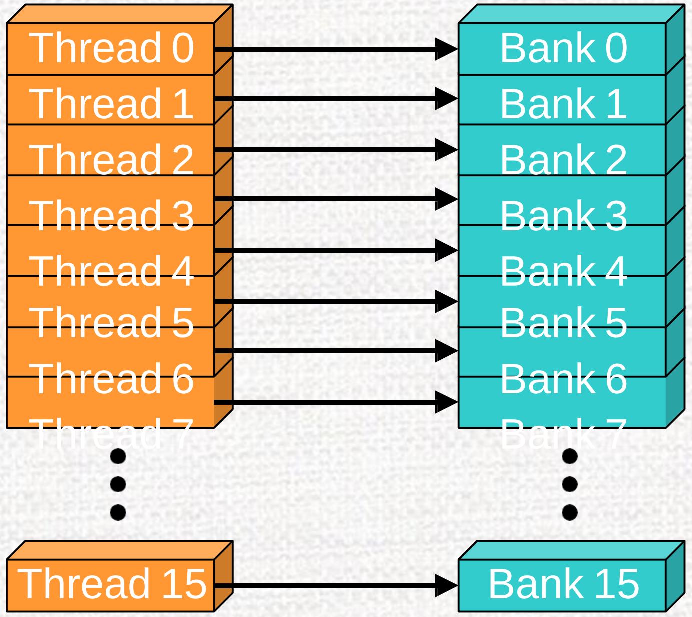


共享内存: banked

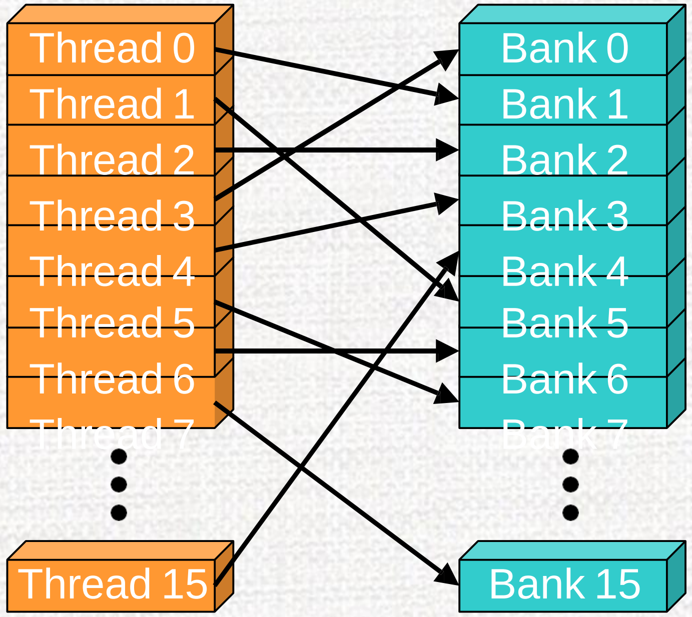
- Bandwidth of each bank is 32 bits per2 clock cycles
- Successive 32-bit words are assigned to successive banks
- G80 has 16 banks
  - So bank = address % 16
  - Same as the size of a half-warp
  - No bank conflicts between different half-warps, only within a single half-warp

共享内存访问

- No bank conflicts
  - Linear addressing stride == 1

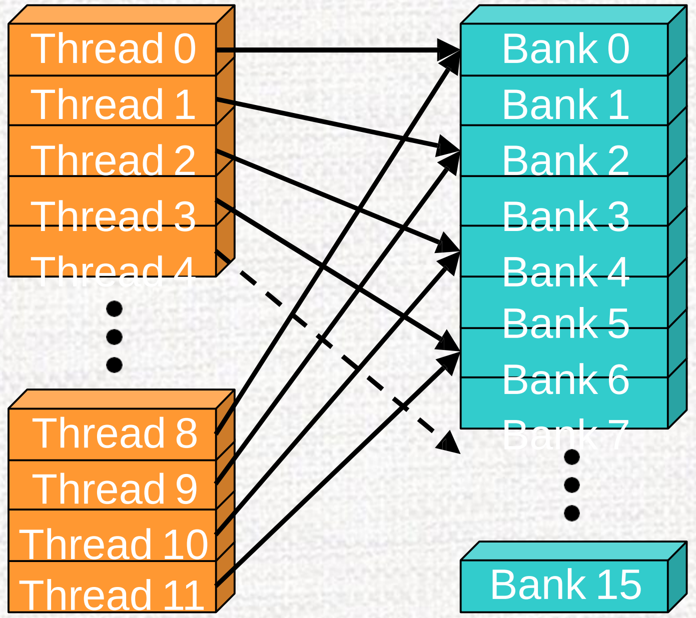


- No bank conflicts
  - Random 1:1 permutation

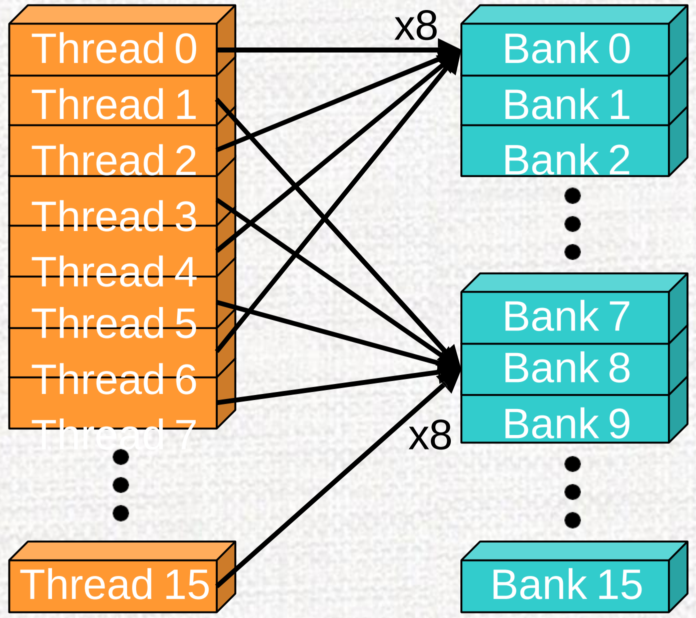


共享内存访问

- 2-way bank conflicts
  - Linear addressing stride == 2



- 8-way bank conflicts
  - Linear addressing stride == 8



共享内存访问冲突

- Shared memory is as fast as registers if there are no bank conflicts
- The fast case:
  - If all threads of a half-warp access different banks, there is no bank conflict
  - If all threads of a half-warp read the same word, there is no bank conflict (*broadcast*)
- The slow case:
  - Bank conflict: multiple threads in the same half- warp access the same bank
  - Must serialize the accesses
  - Cost = max # of simultaneous accesses to a single bank

优化优先级

- Coalesce global memory accesses
  - #1 priority, highest !/$ optimization
- Take advantage of shared memory
- Hide memory latency by running many threads with high arithmetic intensity
- Leave bank conflicts for last!
  - 4-way and smaller conflicts are not usually worth avoiding if it will cost more instructions

THANK YOU