GPU Architecture Optimizations 2

GPU Virtual Memory

Learning Objectives

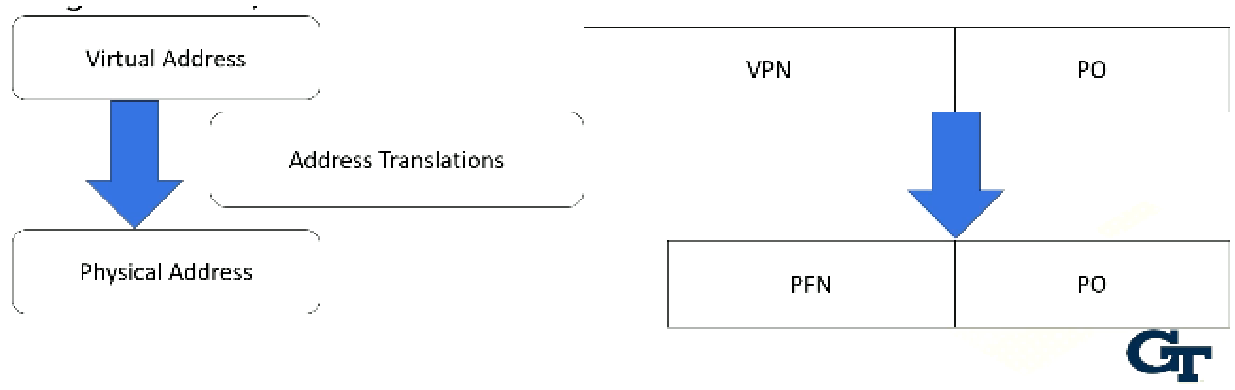
1. Explain the concept of GPU virtual memory and its importance in modern GPU architectures
2. Explain challenges of GPU address translation, including scalability and handling uncoalesced memory accesses
3. Explore hardware optimization opportunities for GPU address translations to improve GPU performance and efficiency

Supporting Virtual Memory

1. Two aspects: address translation and page allocations
2. CPU: O/S handles page allocation
3. GPU: CPU handles GPU page allocations
   - cudaMalloc
   - cudaMallocManaged
   - cudaMemcpy
4. GPU driver and CPU collaborate on page allocations

Reivew of Address Translations

1. Virtual address vs Physical address
2. VPN: Virtual Page Number
3. PFN: Physical Frame Number
4. PO: Page Offset
5. Page size: 4KB, 2MB



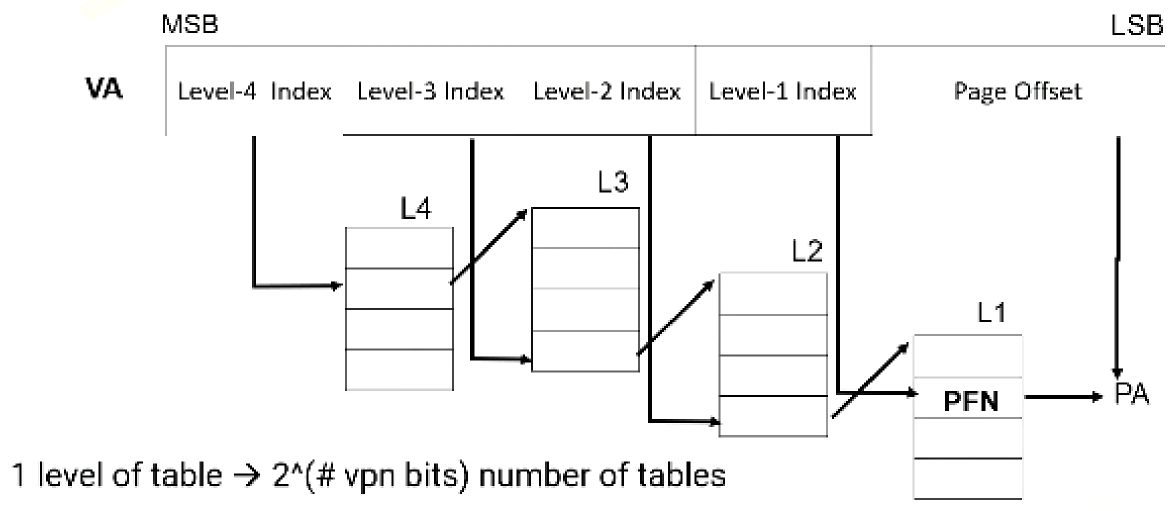
Virtual Memory

TLB and Page Size

1. TLB: Translation Lookaside Buffer
   - Cache to store address translation
2. PTE: Page table entry
   - Contains mapping from VPN to PFN
3. TLB is a cache for PTE
4. TLB miss requires a page table access to retrieve address translation
5. Page fault: Occurs when no physical page is mapped yet, and the OS needs to allocate a page

4 Level Address Translation

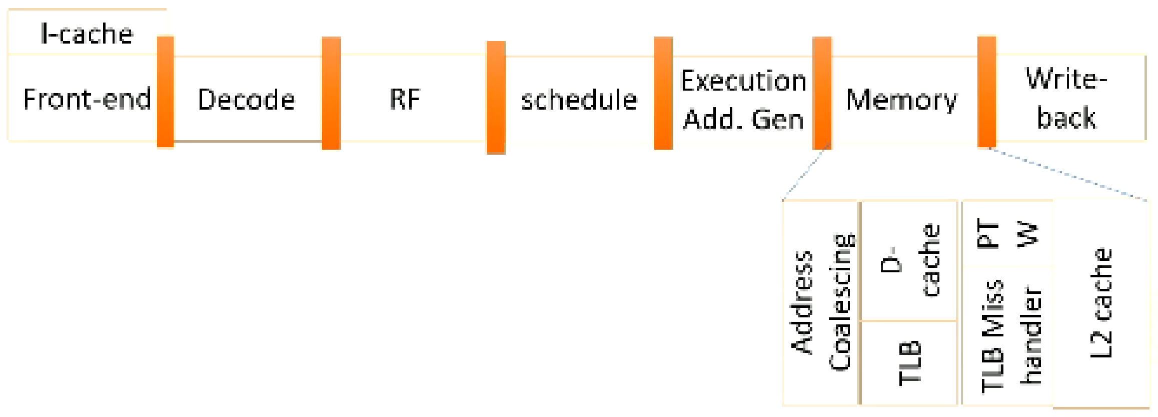
1. 1 level of table -> 2 ^ (# VPN bits) number of tables
   - Hierarchical page tables resolve this
   - Address translation requires traversing multiple page tables
2. One address translation requires at least four memory addresses (walk)
3. Hardware page table walker (PTW) or done by software



4 Level Address Translation

Virtual Memory on GPUs

1. In earlier GPUs, no virtual memory or 1-1 mapping
2. As GPUs became more programmable, complex memory system support is required
   - e.g., Unified Virtual Address
3. Address translation shares similar mechanisms as CPUs except for address coalescing

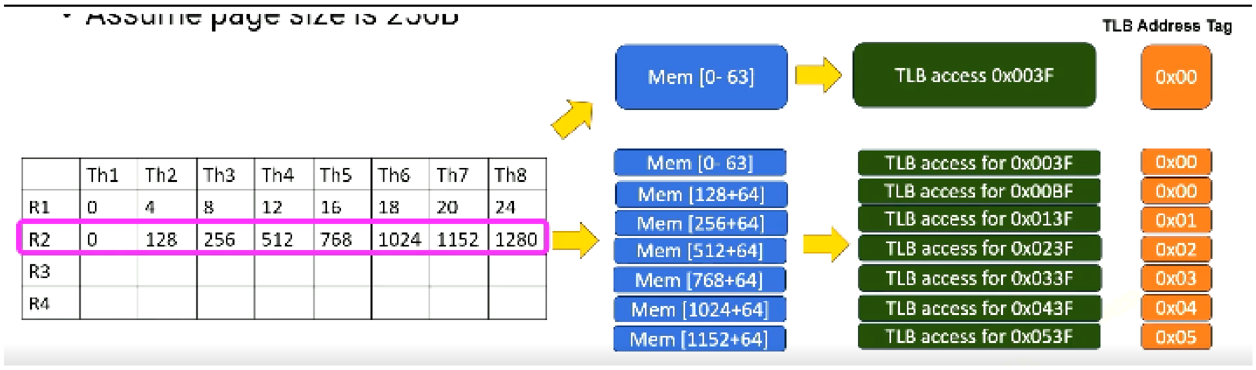


Pipeline

Memory Coalescing and TLB Accesses

1. Uncoalesced memory requests can also generate many TLB misses

2. Assume page size is 256 B
    - The total number of TLB requests from one warp would be the same as the width
    - One memory instruction generated 7 TLB accesses



	Th1	Th2	Th3	Th4	Th5	Th6	Th7	Th8
R1	0	4	8	12	16	18	20	24
R2	0	128	256	512	768	1024	1152	1280
R3								
R4								

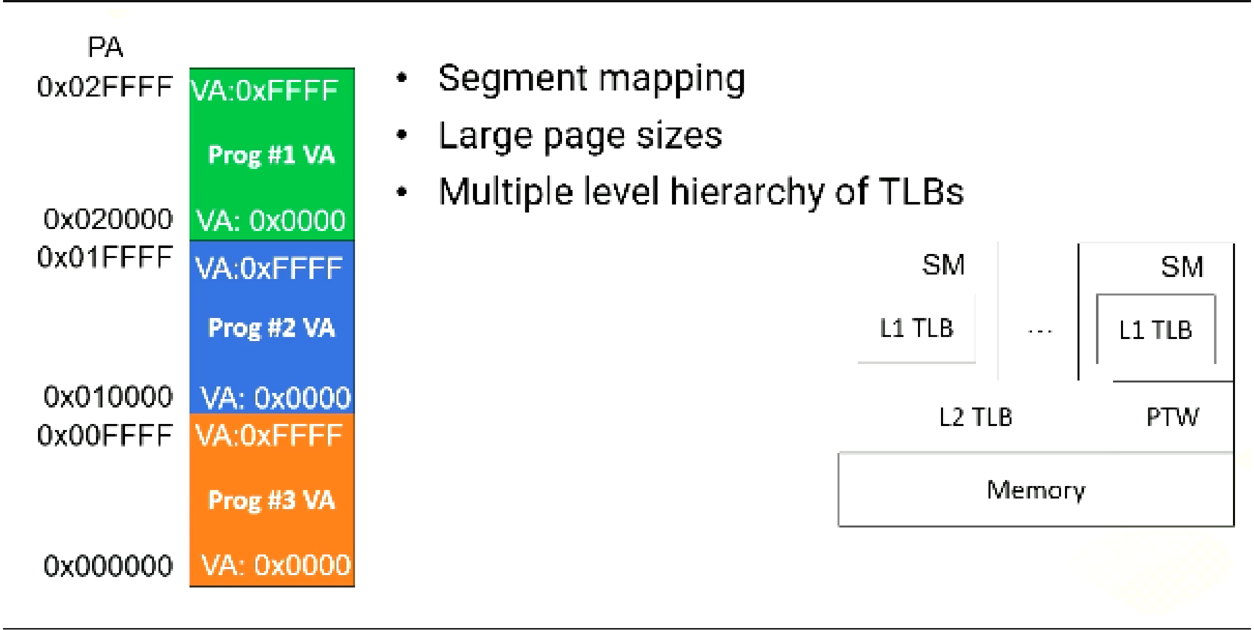
TLB Accesses

Challenges of GPU Address Translation

1. Challenge of address translation of GPUs is providing scalability
2. One warp TLB miss can generate several TLB misses (uncoalesced memory accesses)
3. High bandwidth of the PTW is critical, and sharing a high-bandwidth PTW can be an option

Techniques to Reduce Address Translation Cost

1. Segment mapping
2. Large page sizes
3. Multiple level hierarchy of TLBs



Reducing Address Translation Cost

Large Pages

1. Number of TLB entries required is reduced by 2M/4K (512)

2. Downside: memory fragmentation
    - 1KB is needed
        - If page size is 4KB, wasting 3KB
        - If page size is 2MB, wasting almost 2MB

Summary

1. Memory address coalescing is critical for address translation as well
2. Providing scalable address translation is important such as resource sharing and the use of large pages

GPU Virtual Memory - UVA

Learning Objectives

1. Explain Unified Virtual Address space (UVA)
2. Explore challenges associated with on-demand page allocations

Review: Global Memory Management

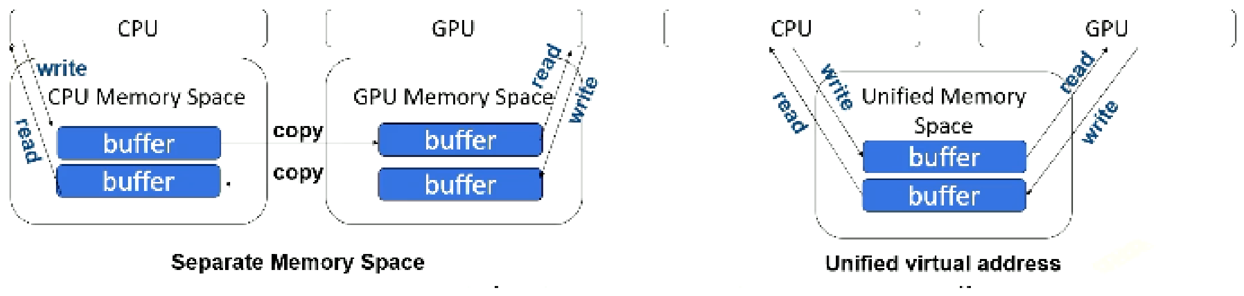
1. cudaMalloc: Allocate memory in the GPU
2. cudaMemcpy: Transfer data between CPU and GPU

GPU Memory Page Allocations

1. Follow the classic "copy then execute" model
2. Utilize cudaMalloc and cudaMemcpy for memory management
3. Require explicit data copying between host and device before kernel launch
4. Benefits: Utilize PCI-E bandwidth efficiently
5. GPU driver and data management are completed before kernel launch
6. Downside: Programmers need to explicitly manage pages

Unified Virtual Address Space (UVA)

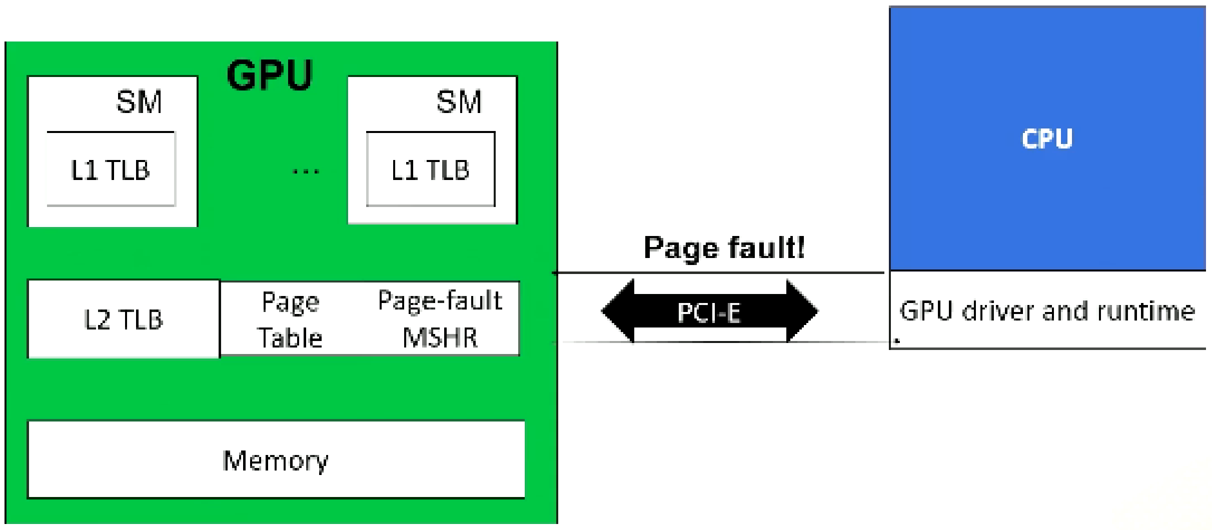
1. CPU and GPU share the same address space
2. Use in OpenCL/CUDA programming model (SVM: Shared Virtual Memory)
3. UVA increases programmability but requires dynamic page allocations
4. Separated memory space allows for bulk data transfer, fully utilizing PCI-E bandwidth



Unified Virtual Address Space

Page Fault Handling Path

1. cudaMallocManaged API



Page Fault Handling Path

Background: IOMMU

1. IOMMU: input-output memory management unit
2. IOMMU connects IO devices to the memory management unit, enabling DMA accesses with virtual addresses
3. Allows I/O virtualization, using virtual address for DMA operations instead of physical addresses
4. Both CPU and GPU have IOMMU, with GPU's virtual page management integrated into it

Challenges of On-Demand GPU Memory

1. GPU page management is performed by the host processor
2. Retrieving page mapping from CPU's IOMMU is expensive
3. Involves the cost of PCI-E communication and CPU's interrupt service handler, often taking several tens of microseconds

Techniques to Reduce the Overhead of On-Demand Memory

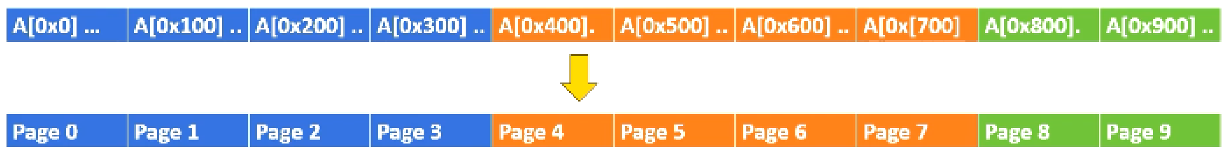
1. Prefetch pages to predict future memory accesses and bring them in advance
2. Move page mapping management to the GPU side
3. Co-operative work between CPU and GPU drivers, with CPU handling large memory chunks and GPU managing small pages

Background of Memory Prefetching

1. Memory prefetching invovles predicting future memory addresses and fetching them preemptively
   - Addr: 0x01, 0x02, 0x03 -> prefetch 0x04, 0x05
2. Vector add example: a[idx], b[idx] accesses data sequentially

Page Prefetching

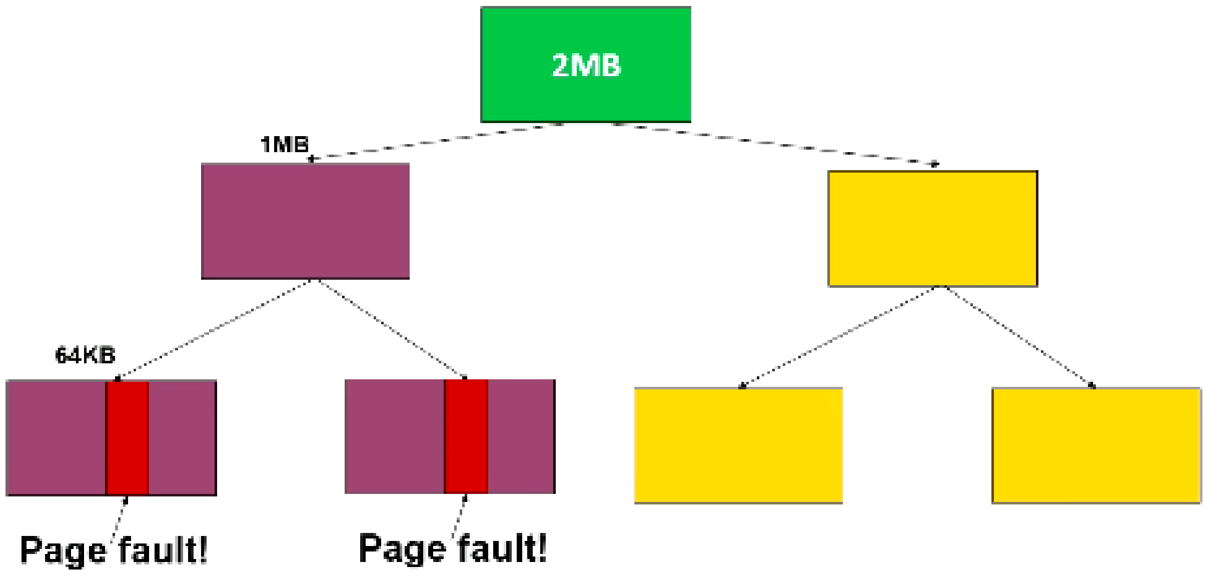
1. Utilizing sequential address predictions
2. Page address can be easily predictable
3. GPU thread block execution order is not sequential
4. Each color represents data accesses from different thread blocks



Page Prefetching

Tree-based Neighborhood Prefetching

1. 4KB page fault
2. Bring the rest of 64KB pages (16 pages) in the same graph node



Tree-Based Neighborhood Prefetching

GPU Memory Oversubscription

1. Problem occurs when allocating more virtual memory space to GPUs
2. Page eviction is unavoidable: not enough pages
3. Which pages to evist is an important question
4. Prefetching is critical
5. As GPU kernel memory demands get higher, the problem gets even more serious

Summary

1. Summarizes several challenges in Unified Virtual Address Space (UVA) management
2. Emphasizes the significance of page prefetching techniques in addressing these challenges
3. On-demand memory allocations vs copy and execute model

GPU Warp Scheduling

Learning Objectives

1. Explain the fundamentals of GPU warp scheduling policies
2. Explore the challenges and considerations involved in GPU warp scheduling

3. Explain round-robin, GTO, and other scheduling policies

Review: GPU Pipeline

1. How do we choose a warp when multiple are ready?

CPU Instruction Scheduler

1. Choose among ready instructions
2. Hard to pipeline the scheduler: single cycle operation
3. Requires to check all instructions in the scheduler
4. Scheduling policies
   - Oldest first
   - Critical instruction first
   - Load instruction first
   - Instruction with many dependent instructions first

GPU Warp Scheduler

1. Implement in-order scheduling within a warp
2. Choose any read warp across multiple warps

RR vs GTO

1. Round-robin
   - Each warp gets equal time on the GPU
2. Greedy-the-oldest
   - Switch when there is a long latency event
   - Schedule same warp until a long-latency event, then the odlest

Two-Level Scheduling

1. Scheduler chooses one warp from multiple active warps (10s of them)
2. Two-level scheduler to improve energy efficiency
   - Two sets of warps: pending and active
   - Move warps from pending to active, but only schedule from active

Warp Scheduling and Cache Behavior

1. What if cache can only have three cache blocks?
2. With RR, none of warps have cache hits with LRU
3. What about GTO?
4. Only compulsory misses

Warp Scheduler and Working Set Size

1. Warp scheduler can affect L1 cache working set size
   - If each warp's working set size is SS_W and there are N number of warps in an SM
     - Using RR: SS_W * N
     - If we limit the number of scheduling warps as SW: SS_W * SW
   - For GTO, SW is not necessarily 1 because GTO schedules another warp in the event of a long-latency instruction

Cache Conscious Scheduling Policy

1. A cache-aware scheduler limits the number of schedulable warps to a small number

2. In case of a stall, choose from warps W1-W3, and once all W1-W3 are completed, schedule instructions from W4
3. Challenges: thread synchronization, ensuring that all warps are executed

Other Warp Scheduling Policies

1. Prefetching-aware warp scheduler: Based on two-level scheduler
    - Concept of fetching from non-consecutive warps
    - Aim to increase bank-level parallelism
2. CTA-Aware scheduling
    - Consider Cooperative Thread Array (CTA) characteristics
    - Control which CTAs are scheduled

Summary

1. Round Robin, Greedy-then-oldest (GTO) scheduling policies
2. 2-level warp scheduler can reduce energy consumption
3. Warp scheduler can affect effective working set size and an example scheduler is cache conscious scheduling policy
4. Other scheduling optimization opportunities exist