Multi-GPU

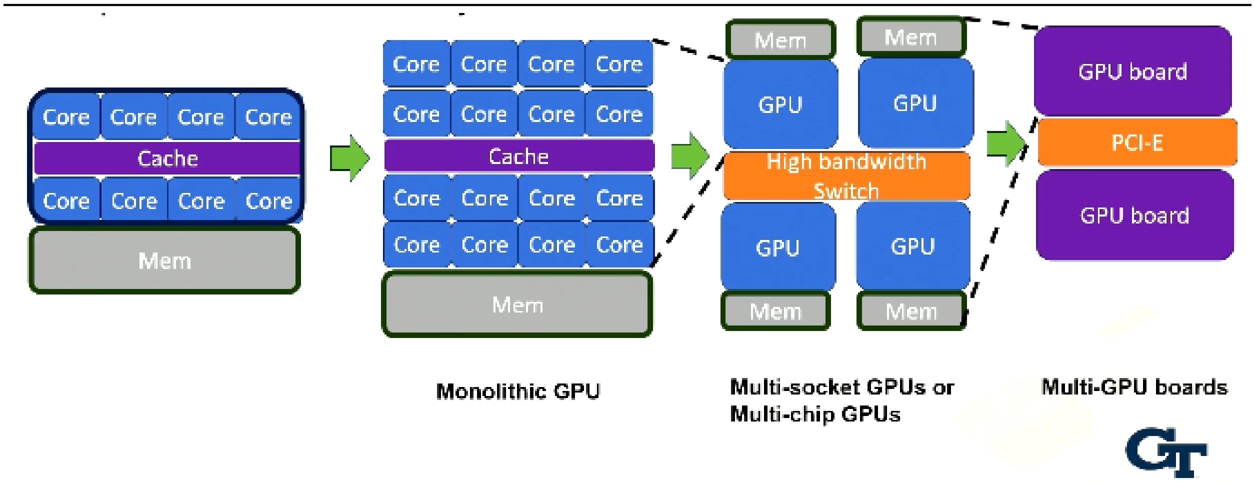
Multi-GPU Hardware

Learning Objectives

1. Describe multi-GPU architecture
2. Describe the challenges of multi-GPU systems
3. Introduce thread block scheduling and memory management in multi-GPUs
4. Explore alternative communication options in multi-GPUs

Multi-GPUs

1. To accomodate the high demand for computing power, GPUs have expanded into multi-GPU systems



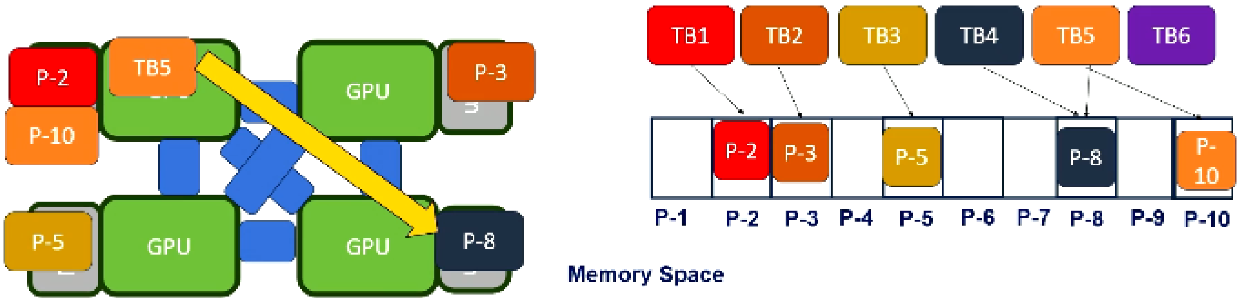
Multi-GPU Architecture

Multi-GPU Connections

1. High-speed connections (e.g., NVLINK)
2. I/O bandwidth needs to be shared
3. Programmer's viewpoint: many SMs
4. Shared memory space
5. Memory is NUMA (non-uniform)
6. Near memory and far memory from the latency's viewpoint

Thread Block Scheduling and Memory Mapping

1. Thread block scheduling policy can be improved
2. Memory Page is allocated based on which GPU touched for the first time



Multi-GPU Scheduling

NVLink/NVSwitch

1. NVLink: an alternative communication option to PCI-E
2. Nvidia's communication protocol
3. Provide high speed communication between GPU-GPU
4. 4th generation of NVLink provides 900 GB/s per GPU and 18 links per GPU
    - 3 TB HBM3 memory bandwidth in H100
5. Switch chip provides even more scalability

Memory Space

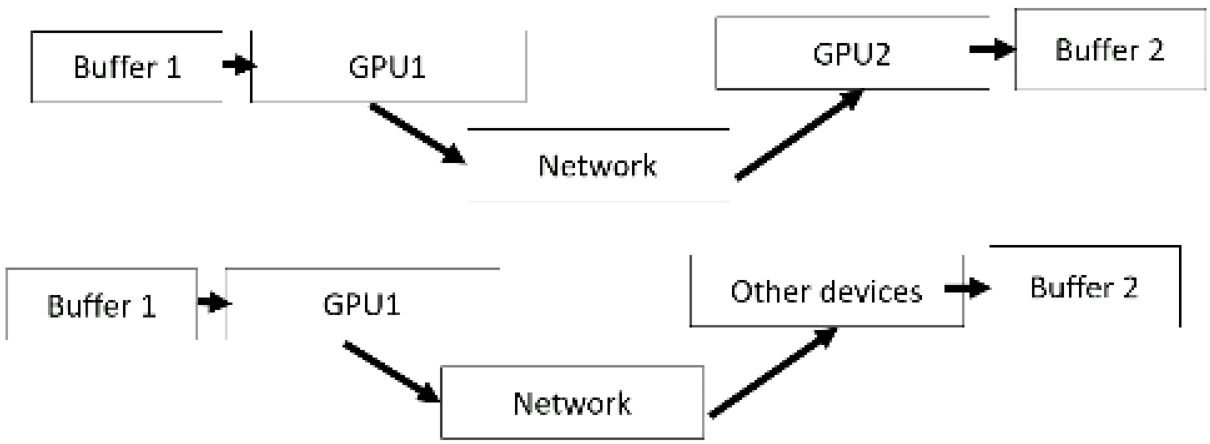
1. Across multi-GPU boards, it uses RDMA to communicate
2. Distributed memory systems

Background: RDMA Technology

1. Traditional communication
2. Use TCP/IP like network to go through the network software stack
3. Use CPU resource
4. RDMA: Remote Direct Memory Access
5. Similar to DMA
    - Communication between local and remote memory can be done without using the CPU resource
6. Host-bypass technology

GPUDirect RDMA

1. Can be used to communicate between GPUs and even other 3rd party devices



Remote Direct Memory Access

Summary

1. Introduced how to scale GPU
2. Explained different communication methods in multi-GPU environment
3. Introduced NVLink and communication benefits
4. Introduced RDMA technology

Concurrency on GPUs

Learning Objectives

1. Describe how to increase GPU utilization, particularly in the context of multi-GPUs
2. Explore diverse strategies for efficiently managing multi-job workloads
3. Discover different GPU concurrency mechanisms:
   - Multi-instance GPUs (MIG)
   - Multi-process Service (MPS)
   - Stream based programming

Increasing Utilization of GPUs

1. Multi-GPUs provide high computing power
2. GPUs are well-suited for multi-data processing, but not all jobs utilize them
3. How can we improve utilization?
4. GPUs for data centers
   - Traditional data centers handle multi-tenant jobs, whereas workloads like LLM using GPUs involve one tenant using all available resources
   - AI workload performance is limited by the slowest task

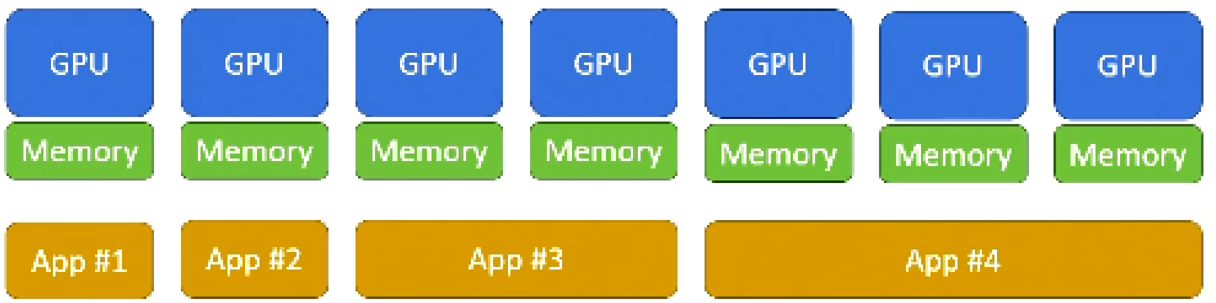
GPU Concurrency Mechanisms

1. Multi-stream: No resource partitioning
2. Multi-process Service: Different clients use different SMs, other resources are shared
3. Multi-instance GPU: GPUs separate for each client

Multi-Instance GPUs (MIG)

1. Multi-Instance GPUs (MIG) allow multiple jobs to run concurrently on a single GPU

2. Efficient way of utilizing large GPUs
3. MIG offers memory and GPU core isolation to host multiple users
4. It partitions memory, L2 cache ports, DRAM memory bandwidth, and on-chip crossbar ports to provide Quality of Service (QoS)
5. For example, the A100 can host up to 7 instances, making it crucial for cloud service providers in multi-tenant use cases



Multi-Instance GPUs

Multi-Process Service (MPS) Support

1. Spatial partitioning was supported from V100 GPU
2. Earlier was supported with only time slicing
3. Multiple jobs can be run concurrently
4. Limitations; No strict resource partitioning, which might cause unpredictable performance but provide limited QoS
5. MPI jobs can be easily ported to CUDA using MPS methods
6. Can provide better utilizations than MIG if QoS is less critical

Stream-based Programming

1. Stream allows concurrent execution of multiple kernels
2. It allows multiple CPU threads to submit kernels
3. OpenMP programs can be easily ported with stream-based programming
4. Scheduling among multiple streams cause performance overhead
    - To overcome, CUDA graph is proposed
    - Dependency chain among kernels are constructed

Example of Programming for Multi-GPU

1. use cudaSetDevice to indicate which GPU
2. Use cudaMemcpy to allocate device on each GPU
3. Call the kernel on each GPU

GPU Support for Multi-Tenant Computing

Mechanisms	Stream	MPS	MIG
Partition type	No	Logical	Physical
Max Partition	Unlimited	48	7
SM isolation	No	By percentage	Yes
Mem BW isolation	No	No	Yes
Performance QoS	No	partial	Yes
Reconfiguration	Dynamic	Process launch	When idle

GPU Support

Summary

1. Explored GPU concurrency mechanisms, including Multi-Instance GPUs (MIG) and multi-process service (MPS) support
2. Introduced stream-based programming