Distributive

Introduction to GPUs
(probably part 1!)

Distributive Book Club
by Amir Sojoodi

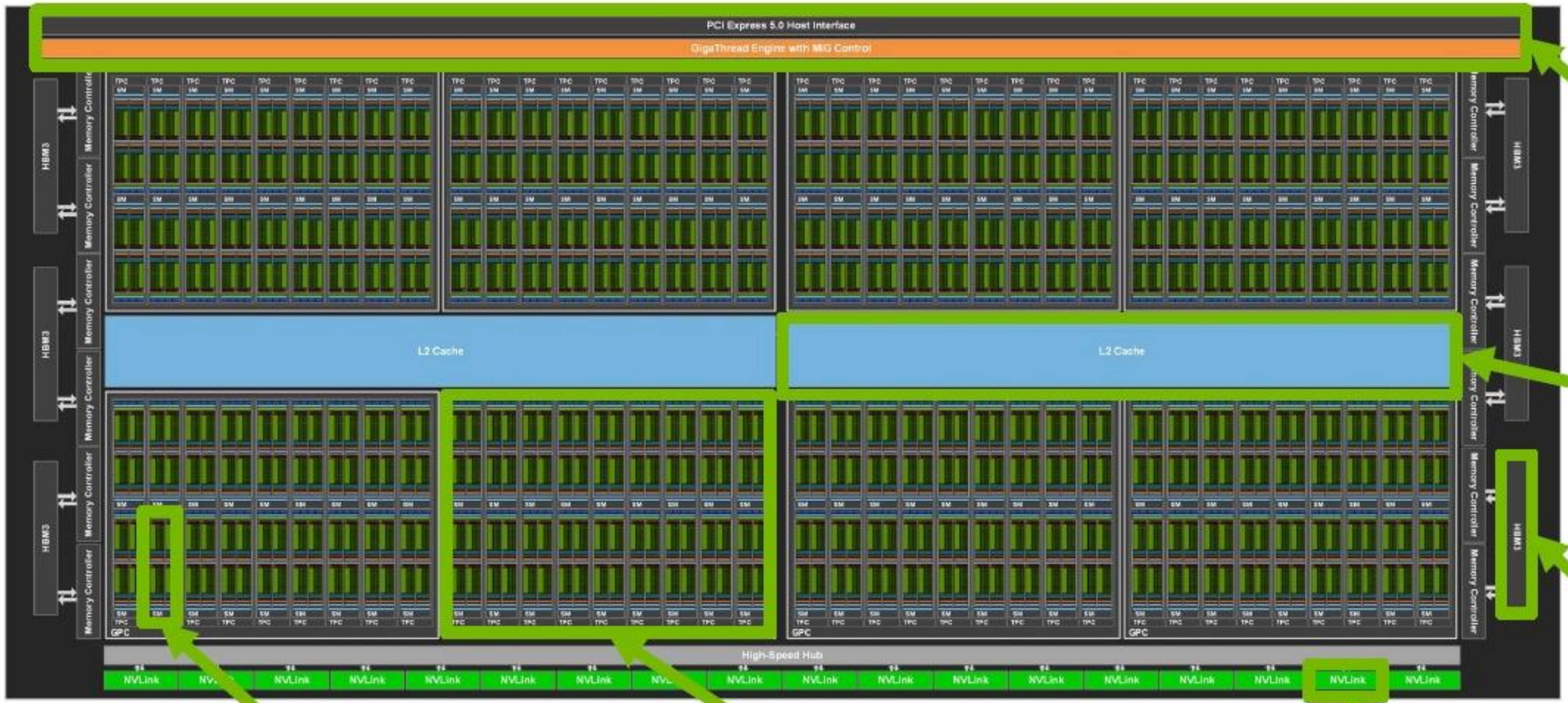
September 28th , 2022

Introduction

- Graphical Processing Units (GPUs)
  - Many cores and tremendous memory bandwidth
- GPUs vs. CPUs
  - Slower
  - Throughput-oriented
  - Let’s look inside!

Hopper Architecture

H100 GPU Key features



2nd Gen Multi-Instance GPU
Confidential Computing
PCIe Gen5

Larger 50 MB L2

80GB HBM3, 3 TB/s bandwidth

132 SMs
4th Gen Tensor Core

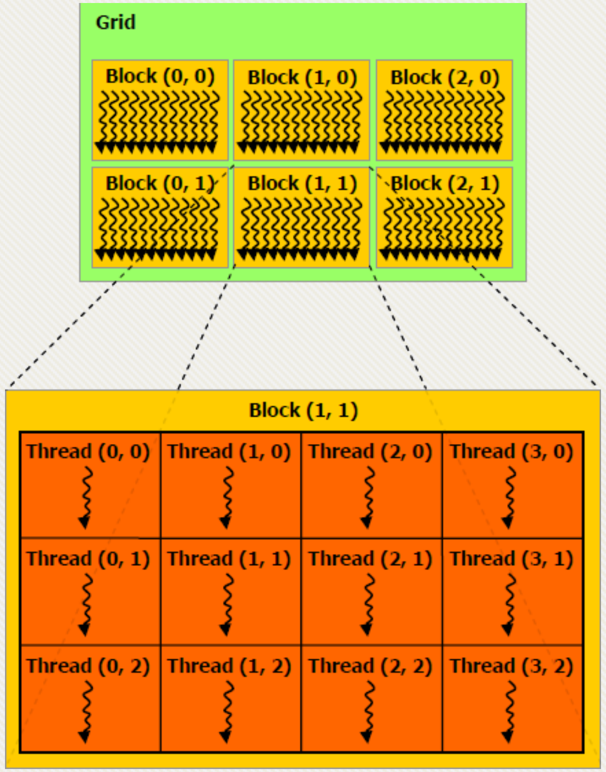
Thread Block Clusters

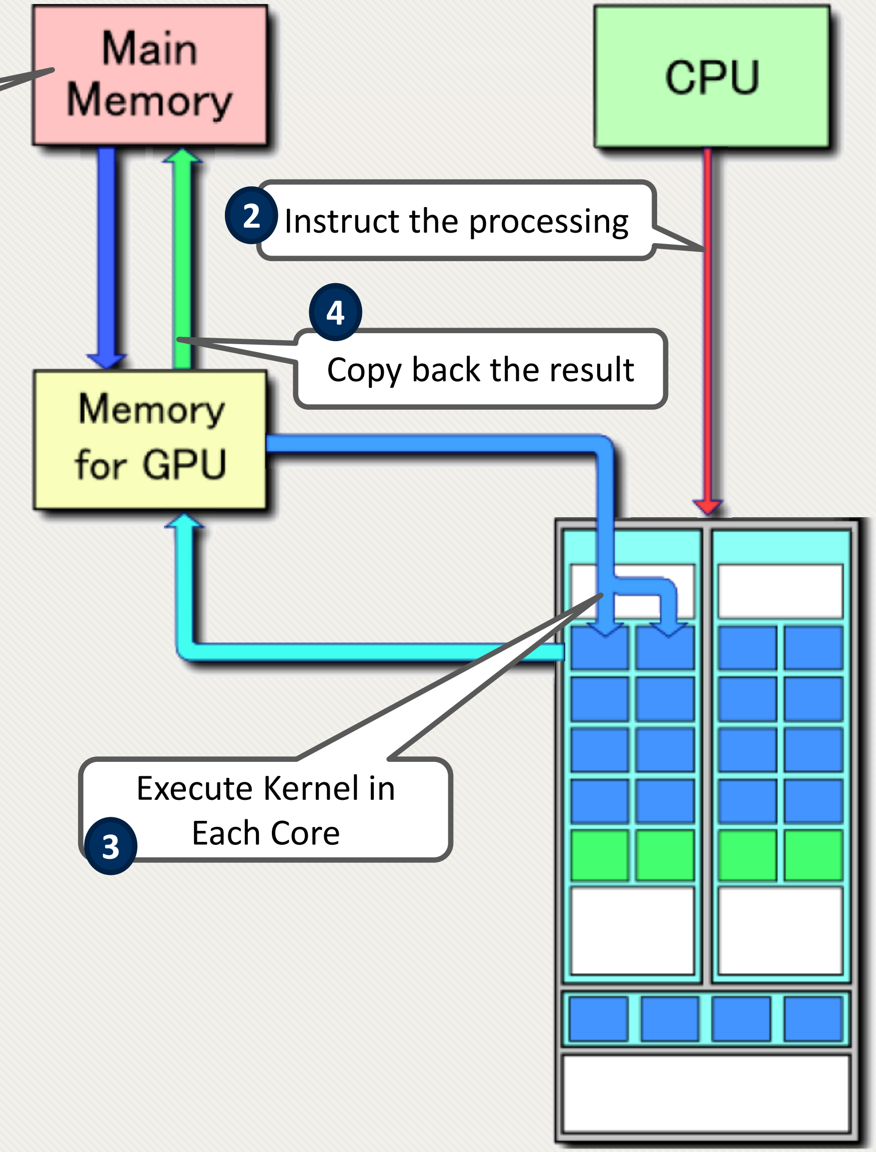
4th Gen NVLink
900 GB/s total bandwidth

Image source: NVIDIA

GPU Processing Flow

- How can we run a code in GPU?
- There are very Mature environments & Platforms:
  - CUDA, OpenCL, …
- Keywords to remember:
  - SM (Streaming Multiprocessor)
  - ThreadBlock & GridBlock
  - Warp (SIMD execution)
  - Kernel
- Let's see a code!





https://github.com/amirsojoodi/GPGPU-Course/tree/master/12th/vectorOpGPU/CUDA

CUDA Features

1. Shared Memory
2. CUDA Streams
3. Unified Memory
4. Dynamic Parallelism
5. Hyper-Q
6. Warp-Level Primitives (Shuffle Instructions)
7. MPS
8. NCCL library (It’s not a feature!)
9. Cooperative Groups
10. CUDA Graphs
11. Multi Instance GPU (MIG)
12. Async-Copy
13. Thread Collectives
14. C++17 STL, templates, pointer aliasing…

GPUs architecture – Simultaneous Multiprocessor (SM)

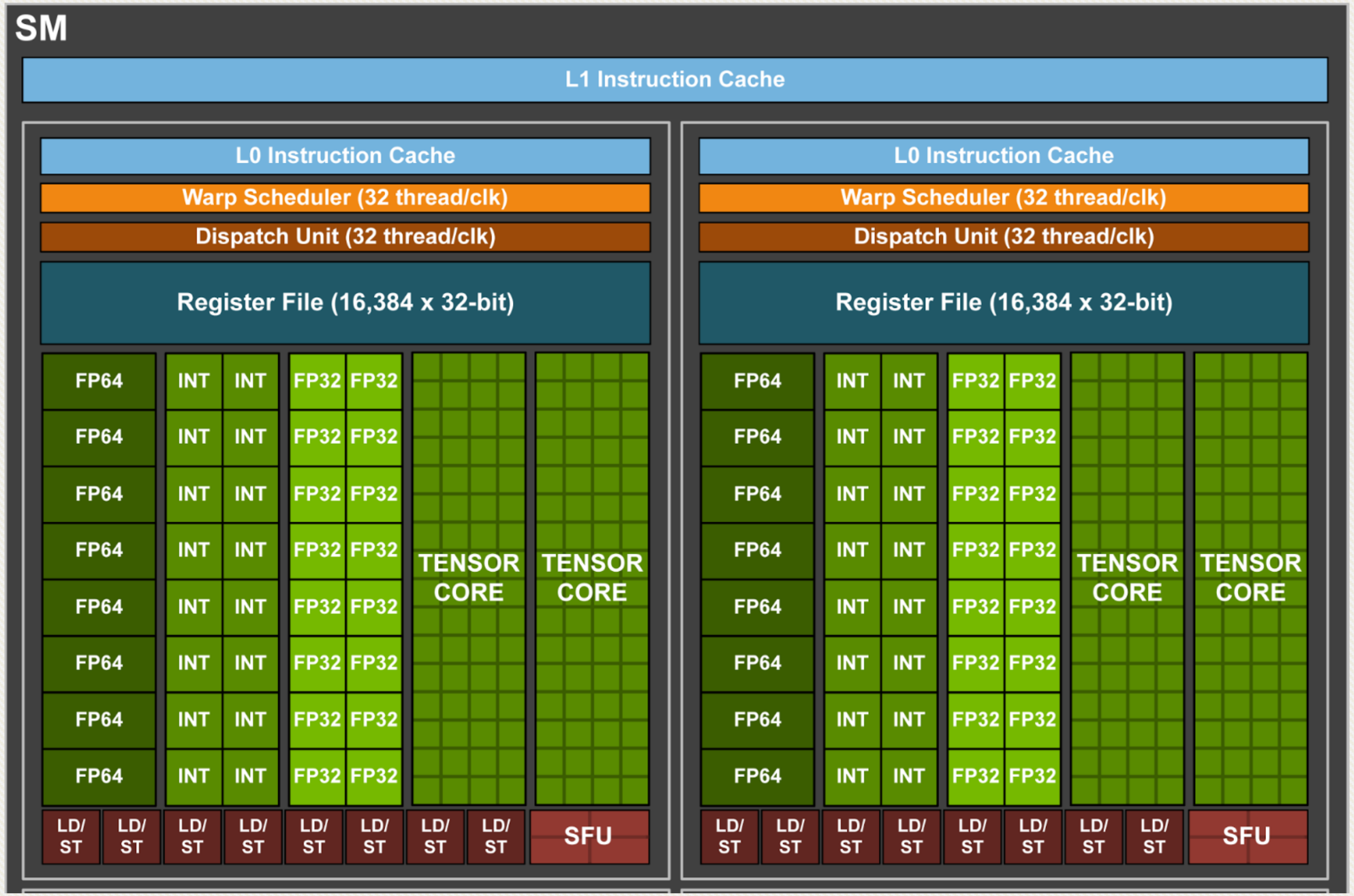


Image source: NVIDIA

NVLink-V3

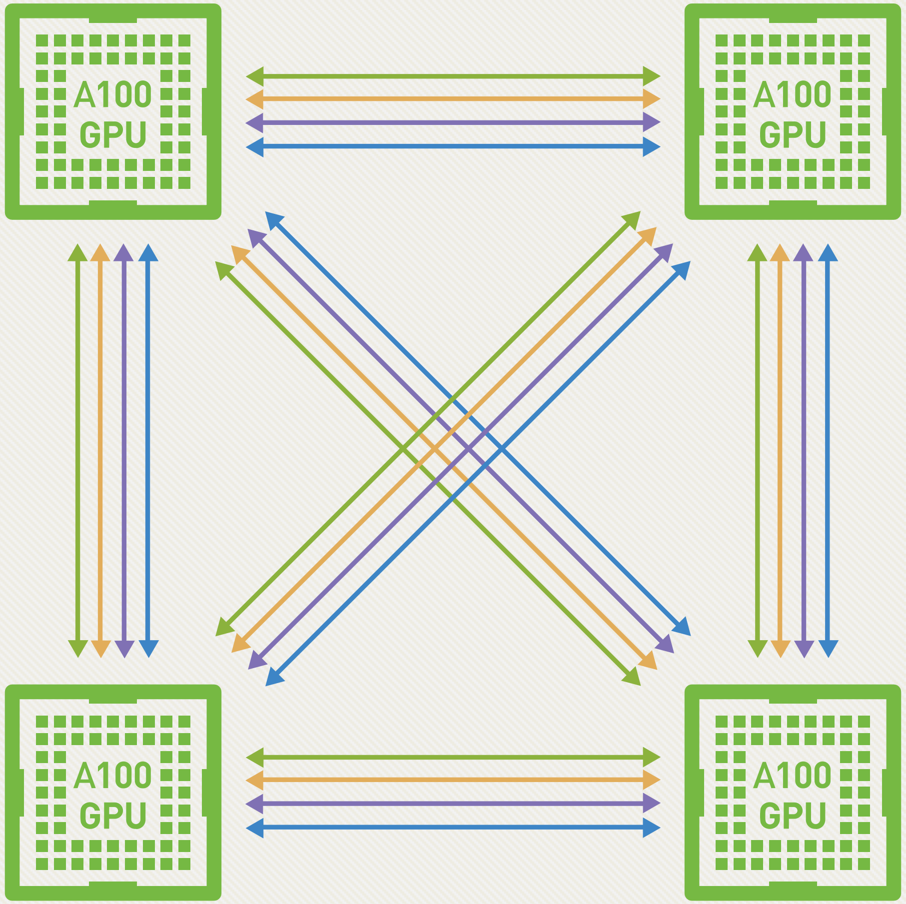


Image source: NVIDIA

NVSwitch

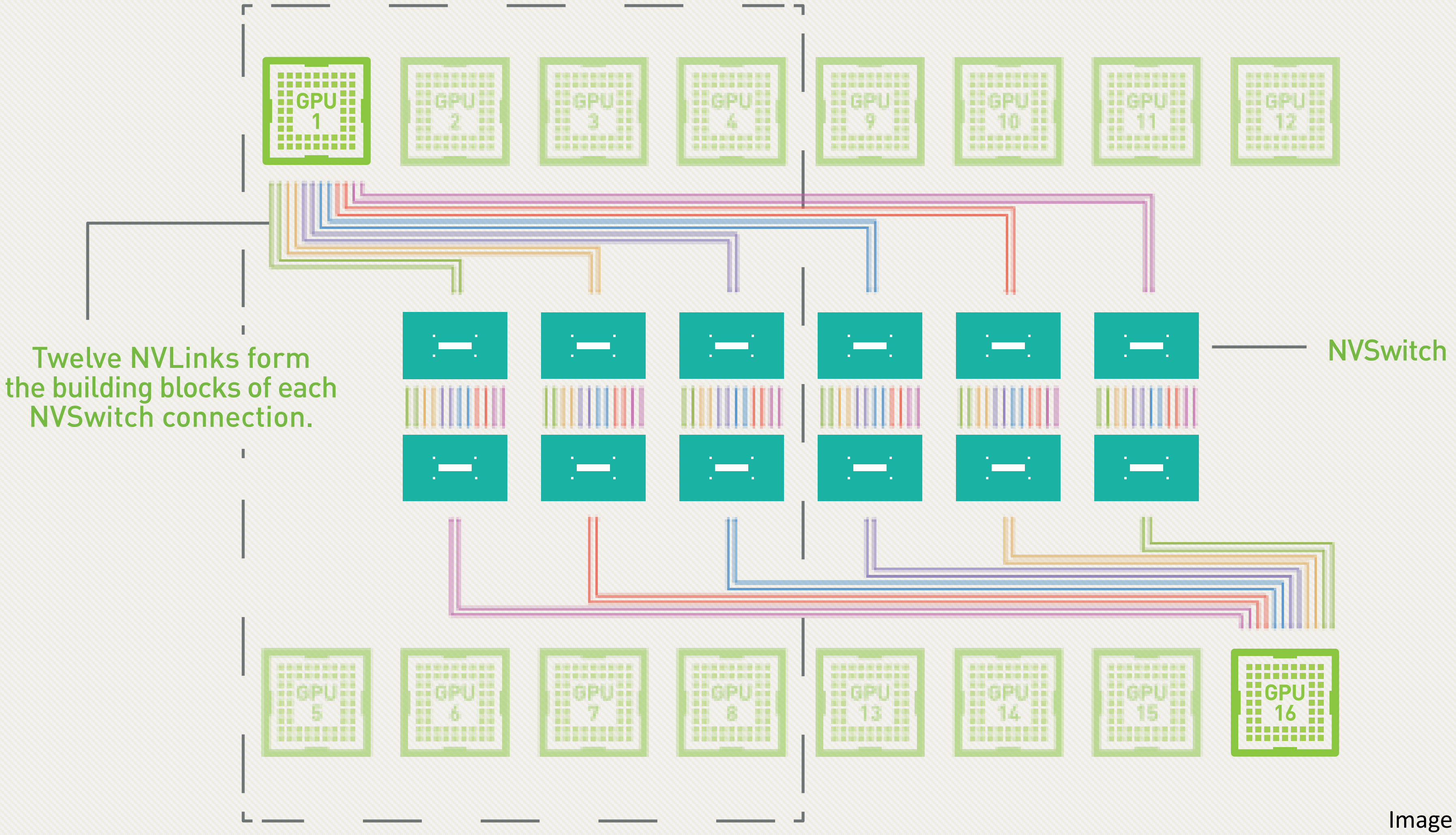
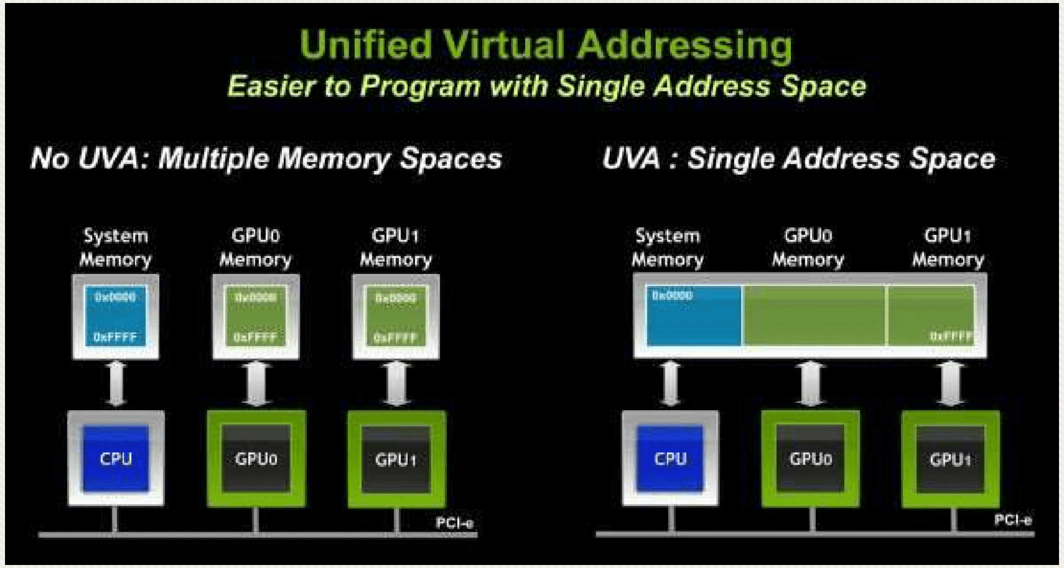


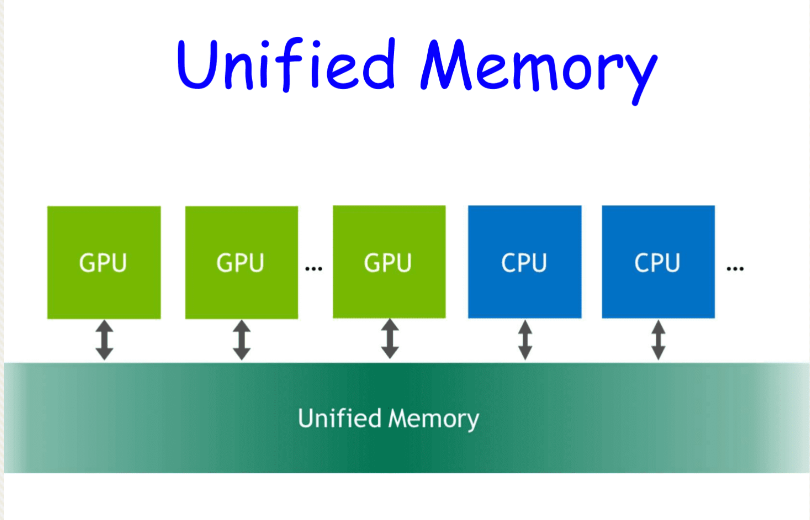
Image source: NVIDIA

UVA vs. Unified Memory

- Unified memory depends on UVA
- UVA does NOT move data automatically between CPU and GPU.



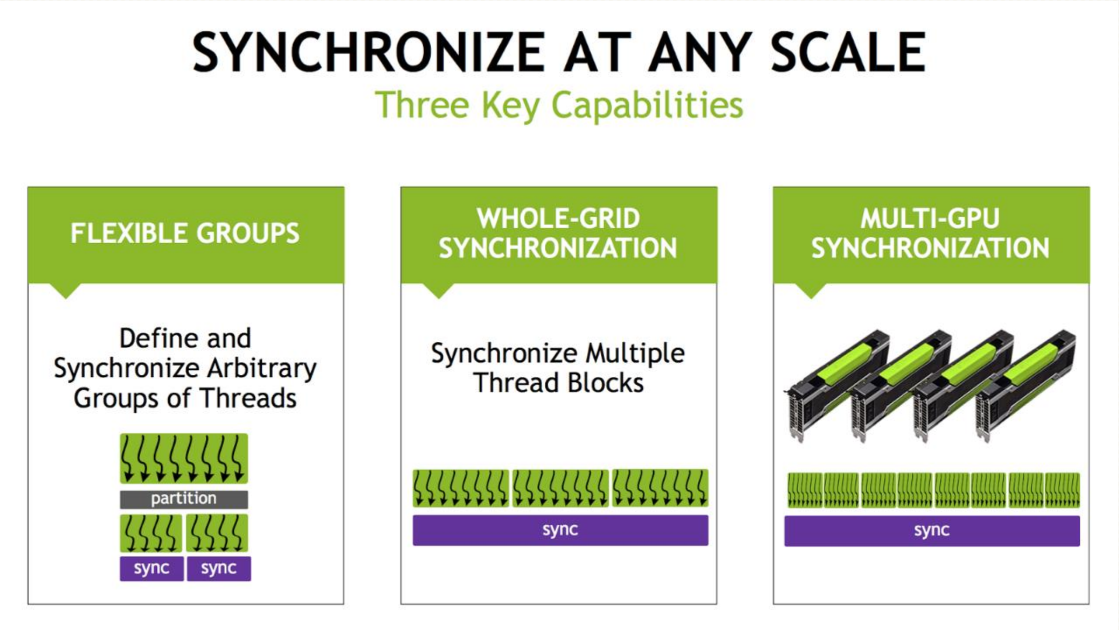
- Advantages:
  - Ease of programming
  - Data is migrated on demand
  - Very efficient with complex data structures (e.g. linked list)
- Disadvantage
  - Carefully tuned CUDA program that uses streams to efficiently overlap execution with data transfers may perform better than a CUDA program that only uses Unified Memory.



Images source: https://nichijou.co/cudaRandom-UVA/

Cooperative Groups

- **Intra-block** synchronization
- **Inter-block** synchronization
- **Tiled** groups



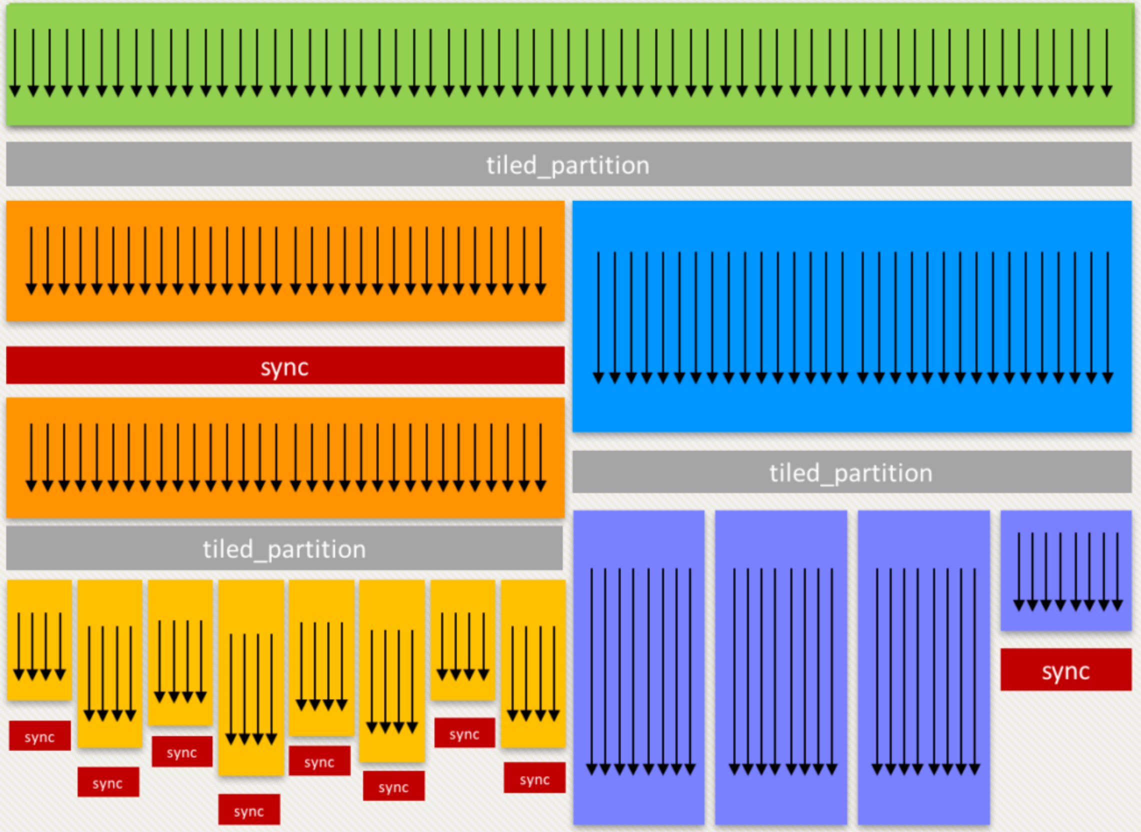


Image source: NVIDIA

NVIDIA Hyper-Q

- Multiple work queues between the host and the GPU

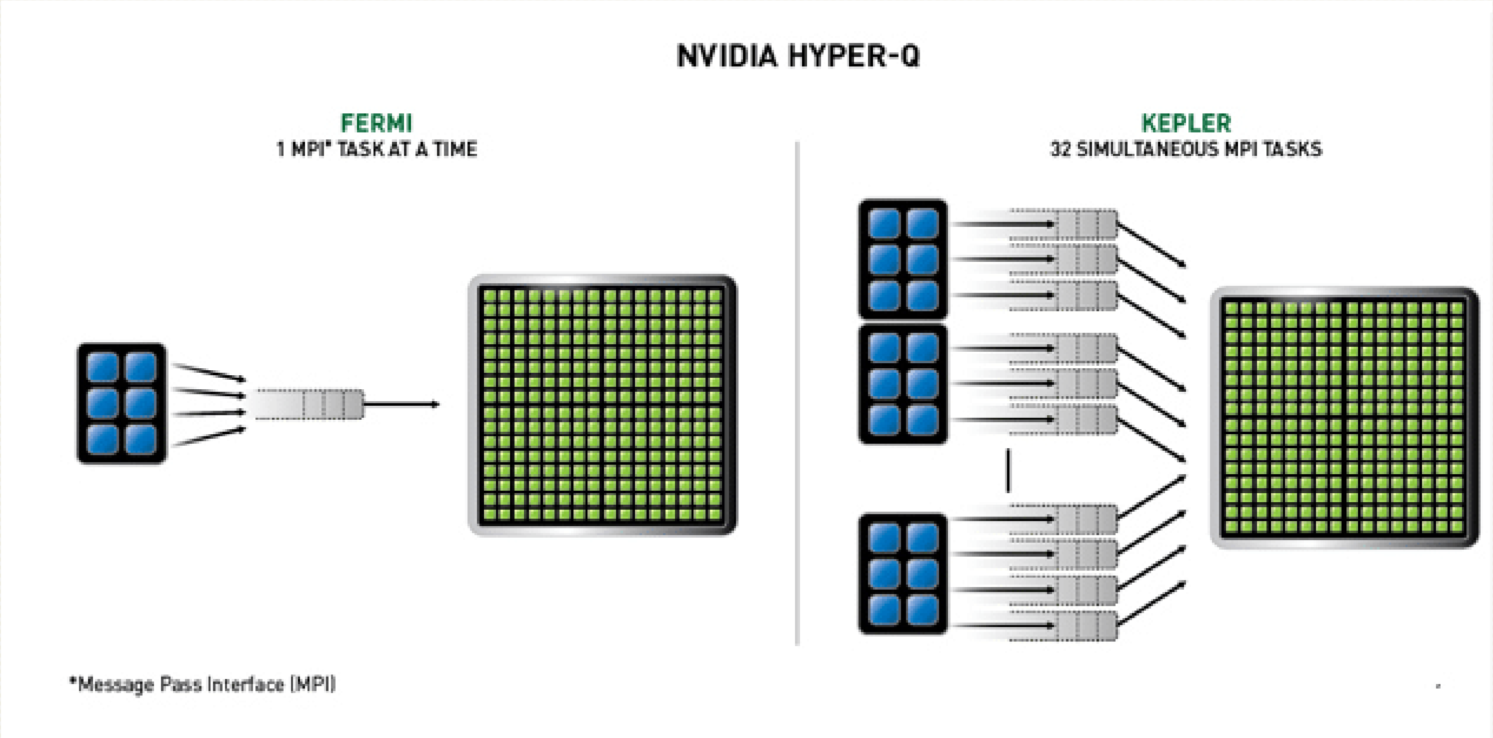


Image source: NVIDIA

NVIDIA Multi-Process Service

- Enable co-operative multi-process CUDA applications, typically MPI jobs, to utilize Hyper-Q capabilities

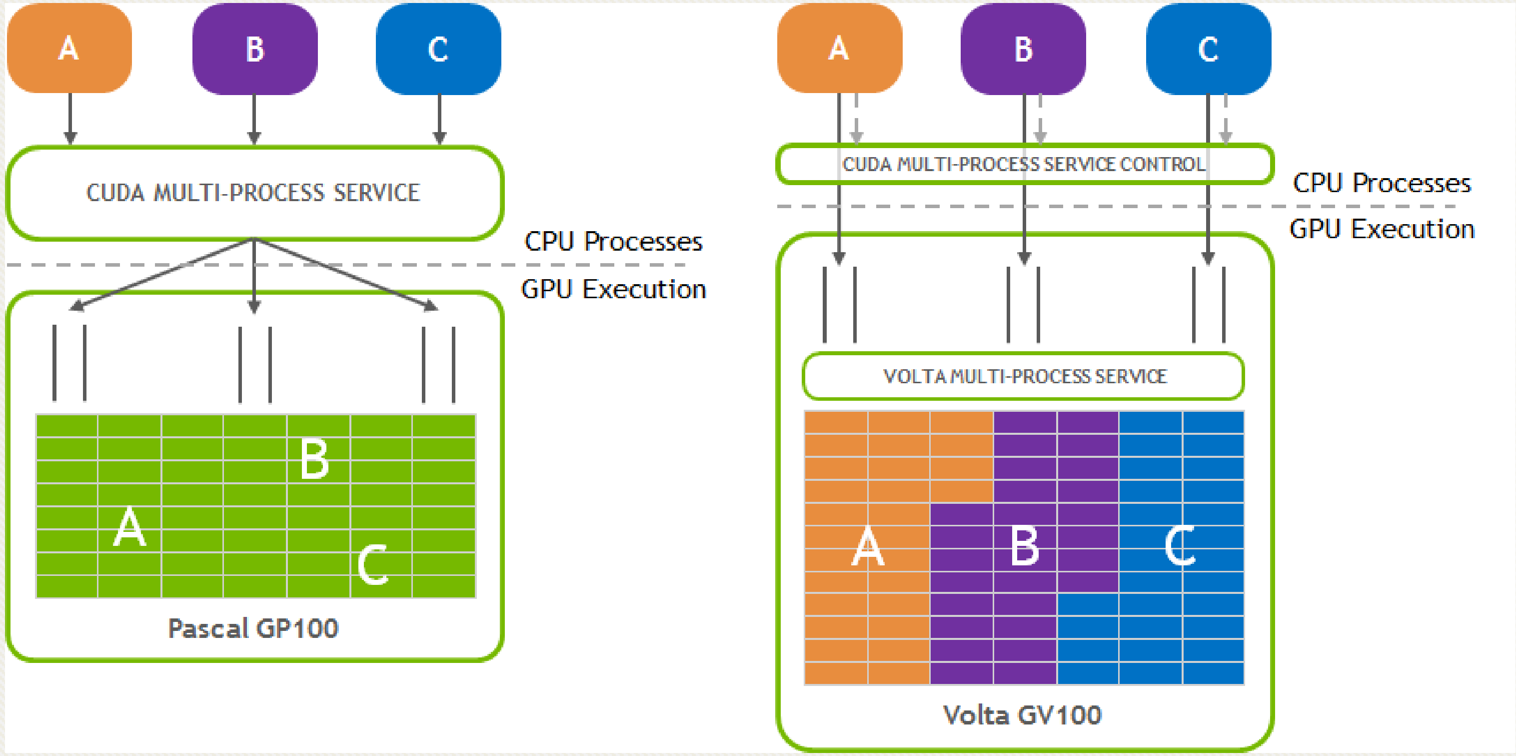


Image source: NVIDIA

CUDA Async Copy

- Overlaps copying data from global to shared memory with computation
- Avoids the use of intermediate registers or the L1 cache.
- Benefits:
  - Control flow no longer traverses the memory pipeline twice
  - Not using intermediate registers can reduce register pressure and increase occupancy

```
//Without async-copy

using namespace nvcuda::experimental;
__shared__ extern int smem[];



// algorithm loop iteration
while ( ... ) {

  __syncthreads();

  // load element into shared mem
  for ( i = ... ) {
    // uses intermediate register
    // {int tmp=g[i]; smem[i]=tmp;}
    smem[i] = gldata[i];
  }
```

```
//With async-copy

using namespace nvcuda::experimental;
__shared__ extern int smem[];

pipeline pipe;

// algorithm loop iteration
while ( ... ) {

  __syncthreads();

  // load element into shared mem
  for ( i = ... ) {
    // initiate async memory copy
    memcpy_async(smem[i],
                 gldata[i],
                 pipe);
  }

  // wait for async-copy to complete
  pipe.commit_and_wait();

  __syncthreads();

  /* compute on smem[] */

}
```

Image source: NVIDIA

Multi-Instance GPU

- Allows the NVIDIA A100 GPU to be securely partitioned into up to seven separate GPU Instances for CUDA applications

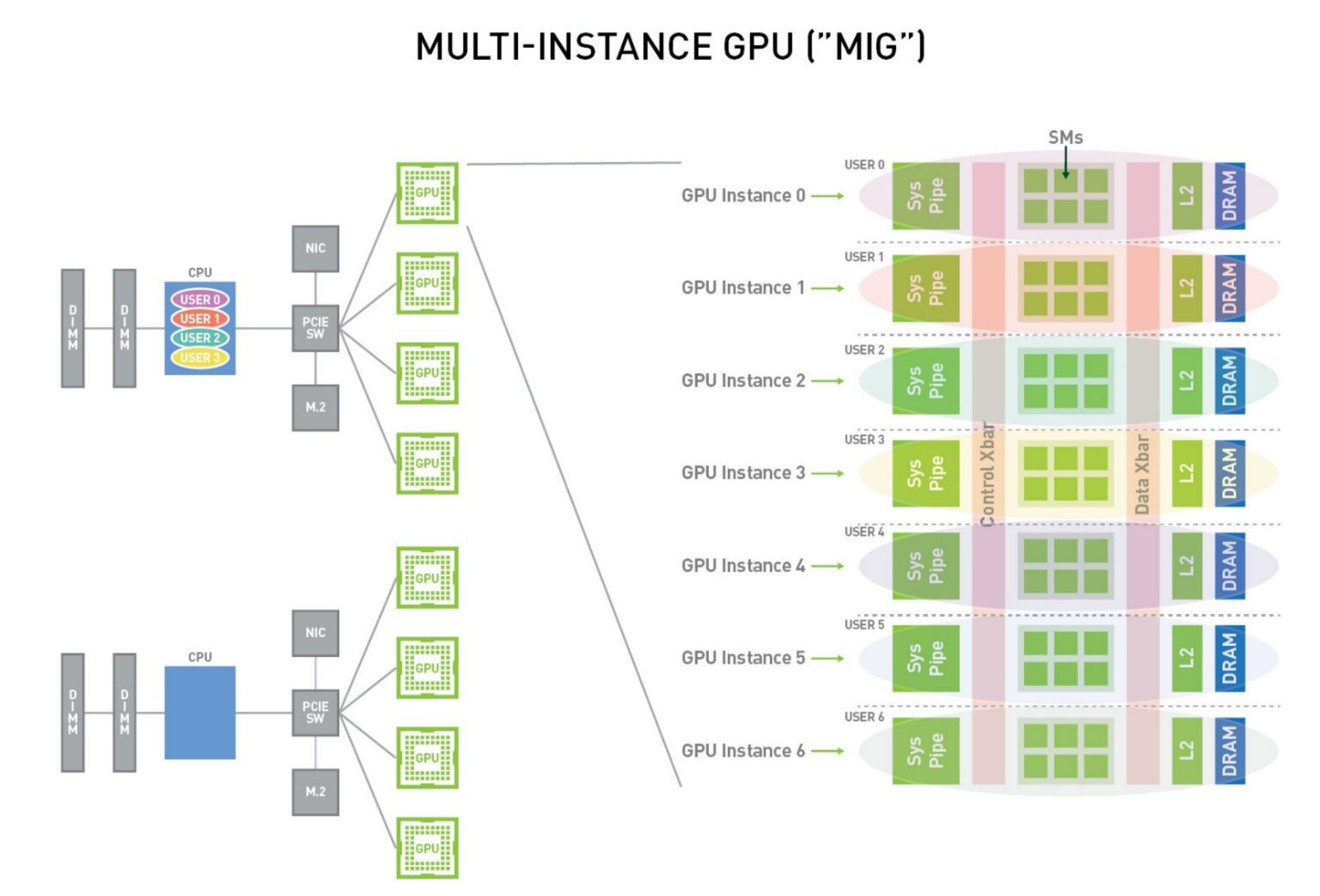


Image source: NVIDIA

CUDA Thread Collectives

- CUDA 11 improvements on top of cooperative groups

```
// Simple Reduction Sum
#include <cooperative_groups/reduce.h>

   ...
   const int threadId = cta.thread_rank();
   int val = A[threadId];
   // reduce across tiled partition
   reduceArr[threadId] = cg::reduce(tile, val, cg::plus<int>());
   // synchronize partition
   cg::sync(cta);
   // accumulate sum using a leader and return sum
```

Intra- and Inter-node GPU-Aware Communications

- CUDA-aware MPI
  - Transfer data buffers across the GPUs efficiently
- Send GPU buffers **directly** instead of staging GPU buffers through the host memory
- Not only P2P Communications, but also one-sided and collectives.

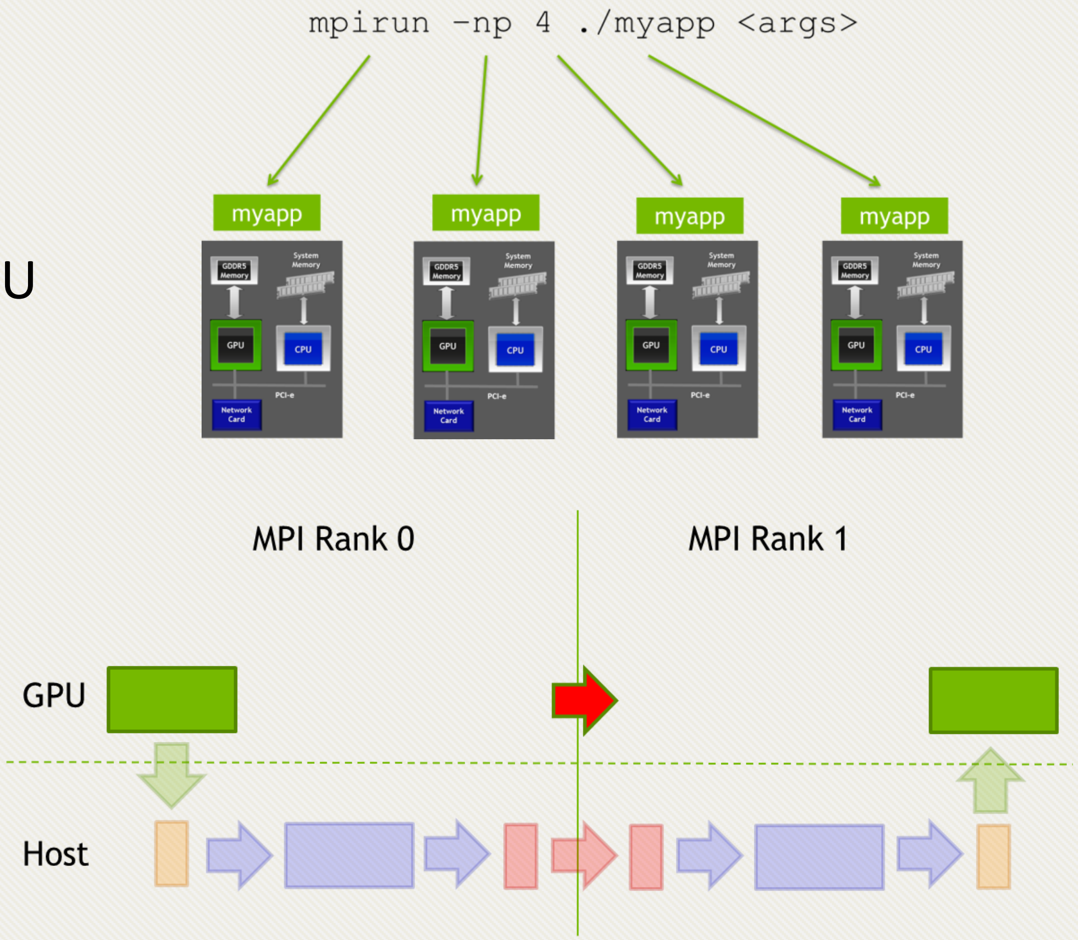
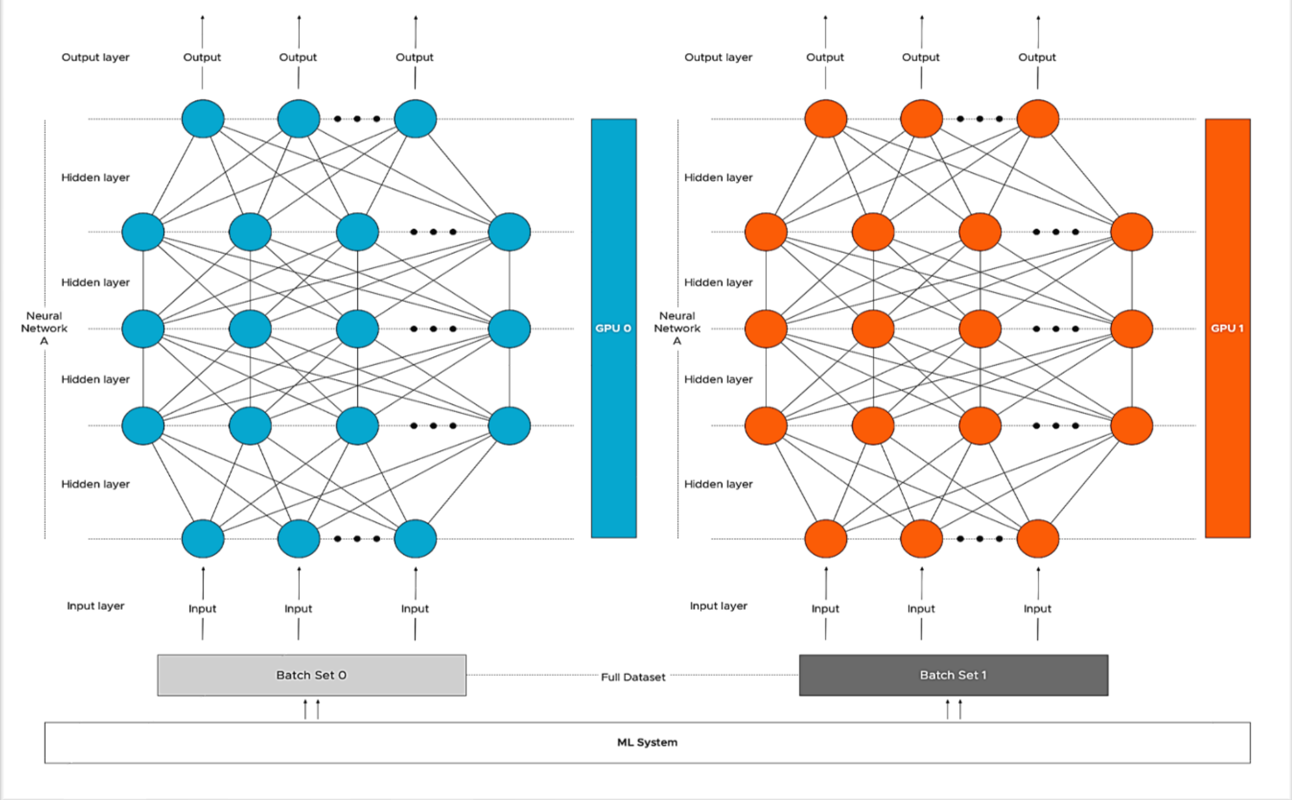


Figure 6: CUDA-aware MPI
source: https://developer.nvidia.com/blog/benchmarking-cuda-aware-mpi

Deep Learning on GPU Clusters

- Data Parallelism vs. Model Parallelism

Data Parallelism



Model Parallelism

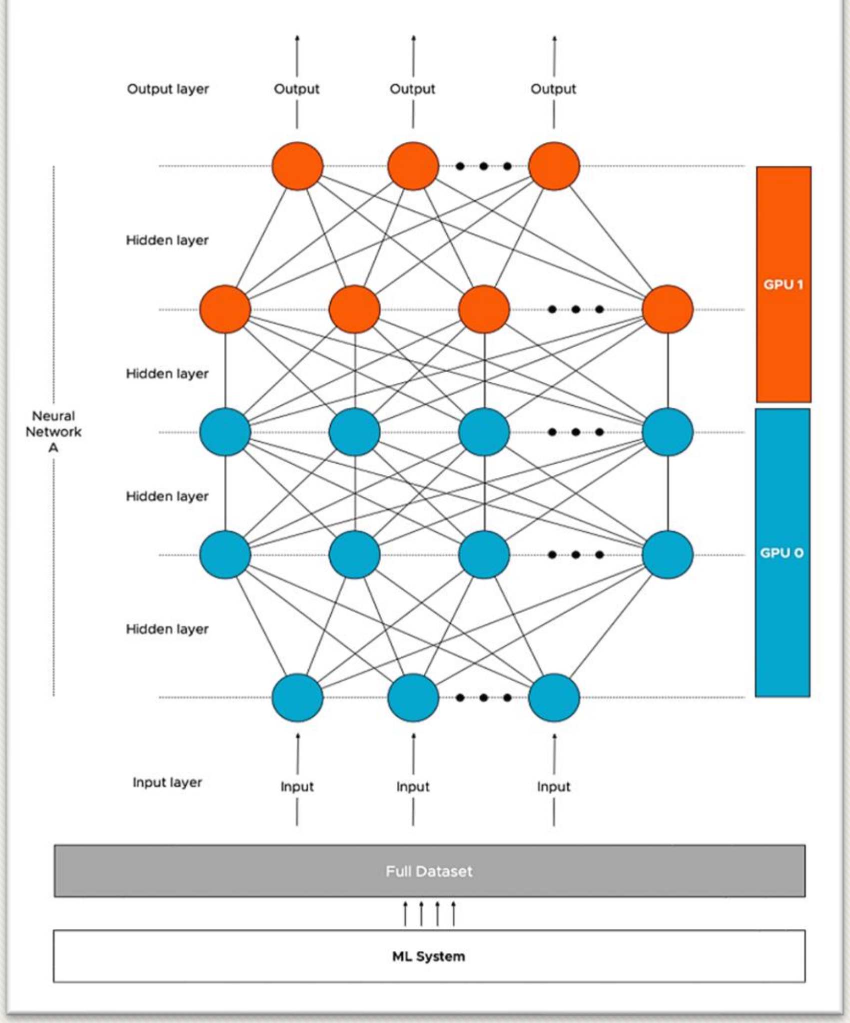


Figure 9: Data Parallelism vs. Data Parallelism, source: https://frankdenneman.nl/2020/02/19/multi-gpu-and-distributed-deep-learning/

Utilizing Modern GPU Features

- CUDA-aware MPI
  - Transfer data buffers across the GPUs efficiently
- GPUDirect RDMA (GDR)
  - Enables on-node or off-node GPUs to directly exchange data without staging it on the host memory.
- GPUDirect P2P
  - Enables the same feature between the GPUs of a node.

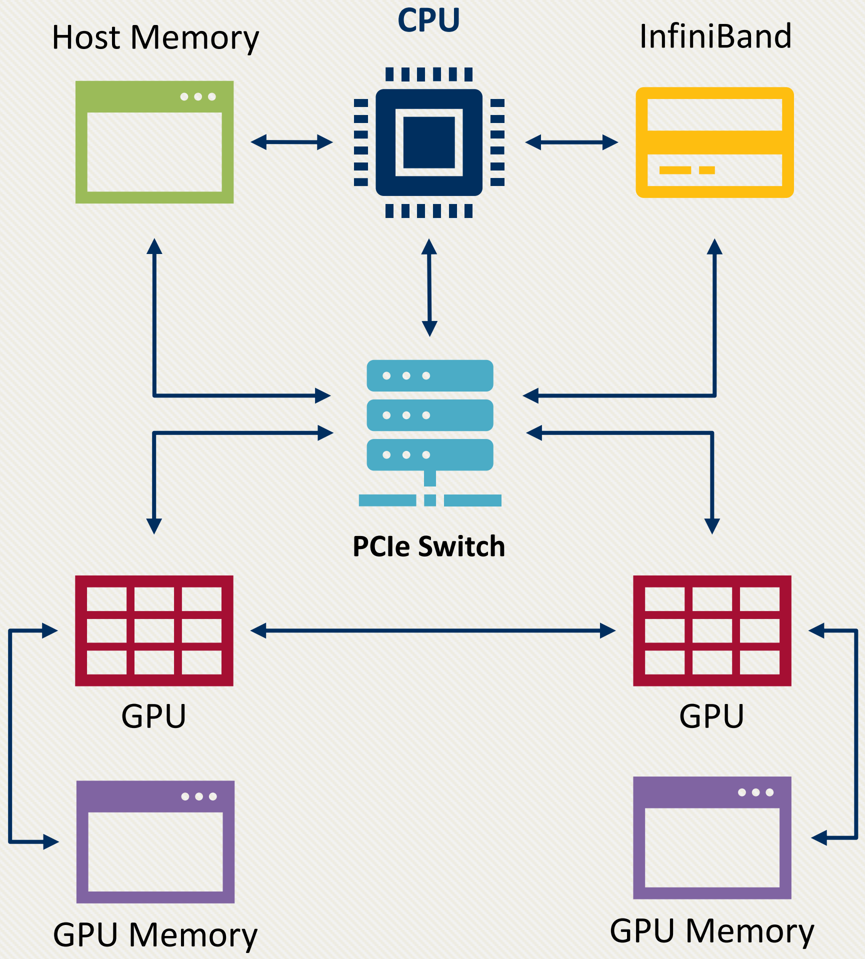


Figure 6: GPUDirect Technologies

Thank You ☺

Instead of blaming darkness, let’s light a candle!

Questions, Comments, and Ideas are Welcome!