Pytorch Meetup Nepal

HPC for AI Researchers:
Demystifying GPU Clusters for PyTorch Workloads

by

Aatiz Ghimire

Scan for Slides & Codes

Aatiz Ghimire

$whoami

Aatiz Ghimiré

AI / HPC Research Engineer
Tribhuvan University Supercomputing Center.

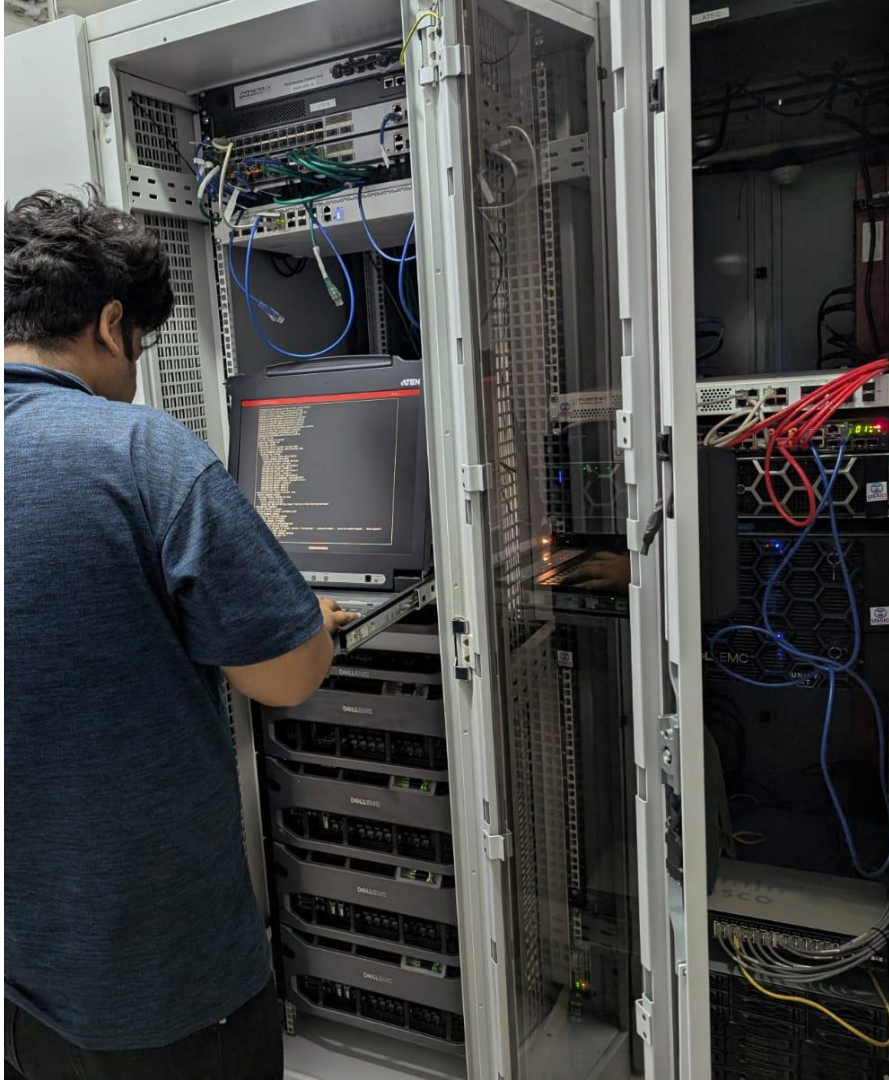
Lecturer / PI
Center for AI & ET
Herald College Kathmandu

CENTRE FOR AI & EMERGING TECHNOLOGIES

Repair, Optimization and Maintenance works on TU-HPC by

“Aatiz Ghimire”

- May, 2025



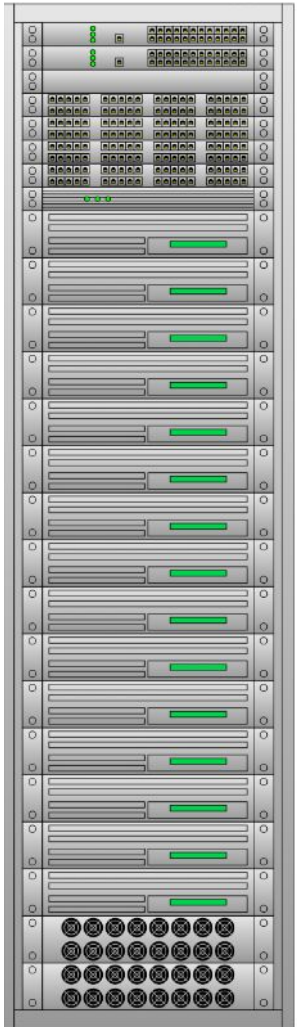
High Performance Computing

The Node Anatomy

Head Node: The "Brain". Handles login, job scheduling (Slurm), and management. Users never compute here.

Compute Nodes: The "Muscle". Where the actual number crunching happens.

Rack Architecture: Dense packing for power and cooling efficiency.



Infiniband Network
BMC Network
TCP Eth Network

Head Node
.
Storage Node
.
Compute Node (C1)
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
Compute Node (C10)
.
GPU Nodes (GPU1)
.
.

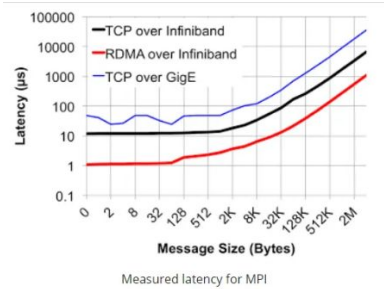
The Backbone: Infiniband vs Ethernet

Why Latency Matters

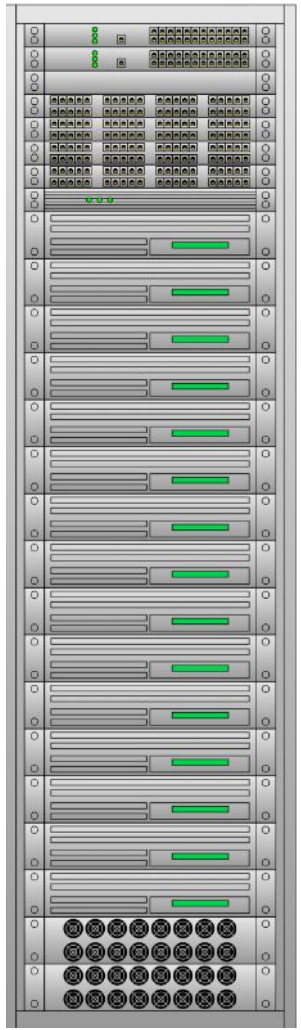
For distributed training, nodes must exchange gradients thousands of times per second.

Ethernet (TCP/IP): High overhead, unpredictable latency. Good for web, bad for HPC.

InfiniBand (RDMA): Remote Direct Memory Access allows one computer to read the RAM of another without CPU involvement.



Measured latency for MPI



Infiniband Network
BMC Network
TCP Eth Network

Head Node
.
Storage Node
.
Compute Node (C1)
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
Compute Node (C10)
.
GPU Nodes (GPU1)
.
.

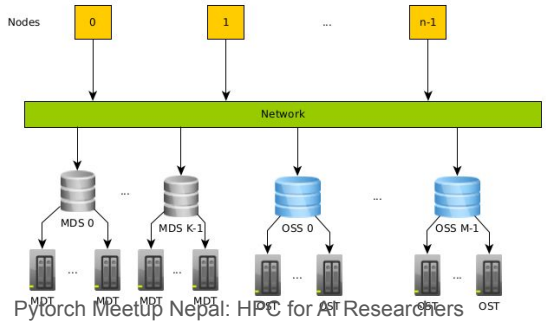
Feeding the Beast: Parallel Storage

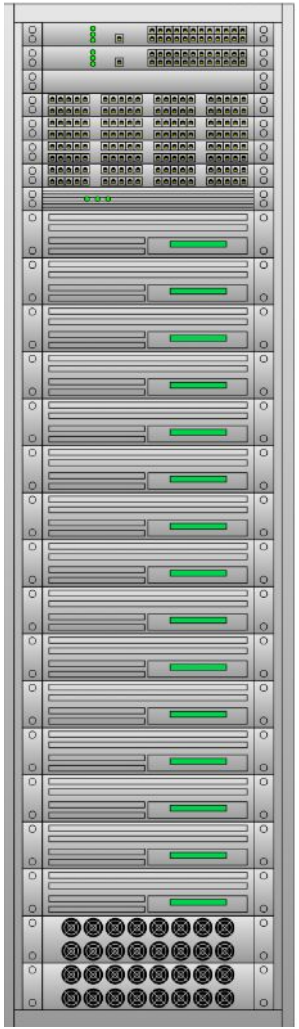
The I/O Bottleneck

Deep Learning models read random image/text batches continuously. A standard NAS would crash under this load.

Lustre File System: Stripes a single file across multiple physical disks (OSTs).

MDS (Metadata Server): "Where is the file?"
OSS (Object Storage Server): "Here is the data."





Infiniband Network
BMC Network
TCP Eth Network

Head Node
.
Storage Node
.
Compute Node (C1)
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
Compute Node (C10)
.
GPU Nodes (GPU1)
.
.

Orchestration: The Slurm Manager

Fairness in a Shared Resource

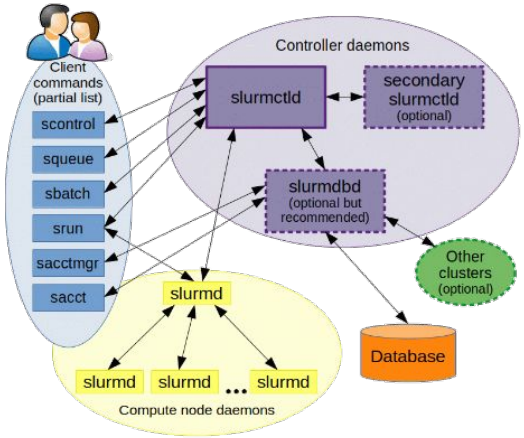
Hundreds of students need access. We cannot let one user hog all GPUs.

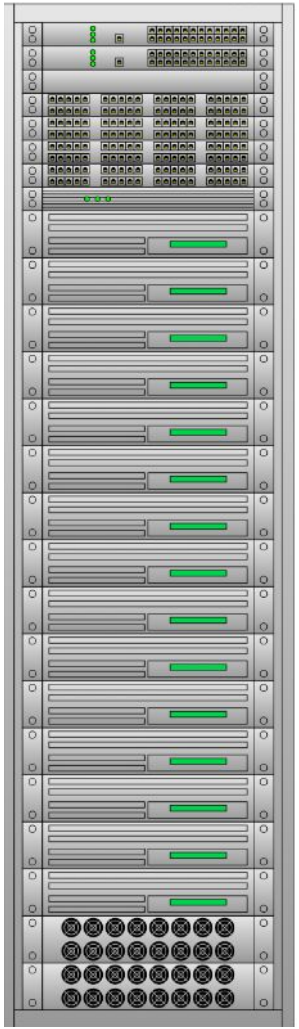
Key Commands:

sbatch script.sh: Submit a job to the queue.

squeue: Check status.

scancel: Stop a job.





Infiniband Network
BMC Network
TCP Eth Network

Head Node

.

Storage Node

.

Compute Node (C1)

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

Compute Node (C10)

.

GPU Nodes (GPU1)

.
.

Compiler Optimization: The Right tool

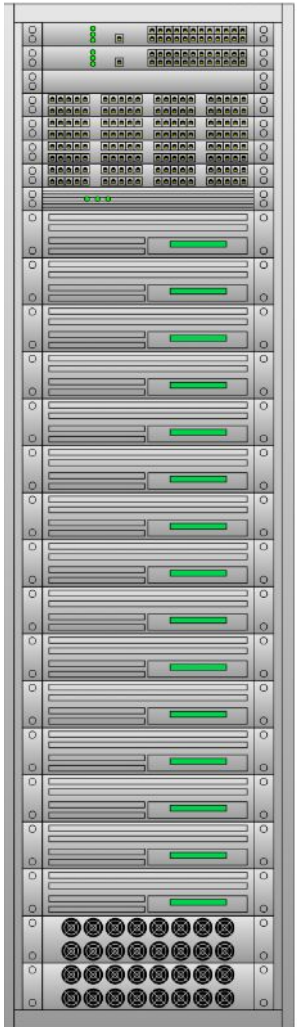
Acceleration based on Architecture

Correctly compiled code will run better on correct machines.

Key Commands:

Make the script executable: **chmod +x compilation-gcc.sh**

Check: **./compilation-gcc.sh**



Infiniband Network
BMC Network
TCP Eth Network

Head Node
.
Storage Node
.
Compute Node (C1)
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
Compute Node (C10)
.
GPU Nodes (GPU1)
.
.

Parallel Execution: The Legacy Systems

Parallel Execution: The Legacy

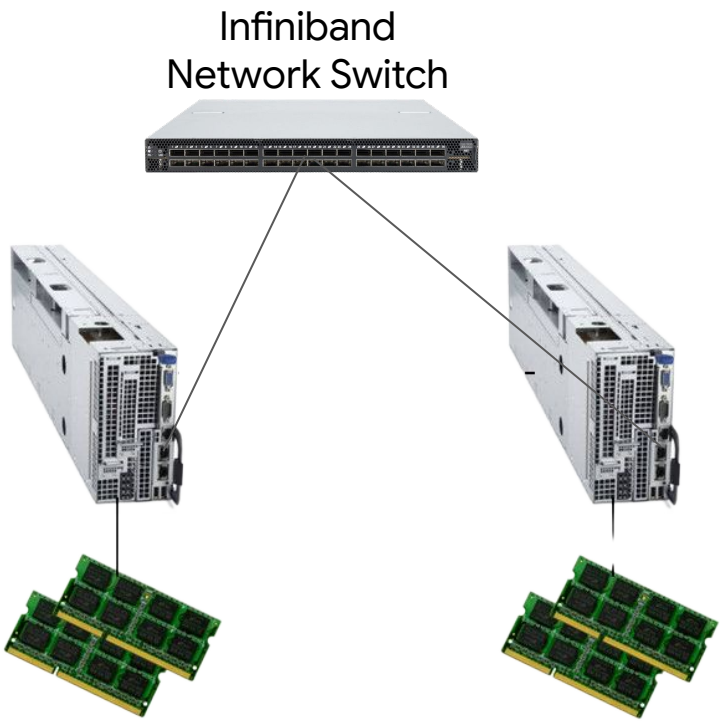
MPI: Message Passing Interface

The standard for decades in Scientific Computing (Weather, Physics).

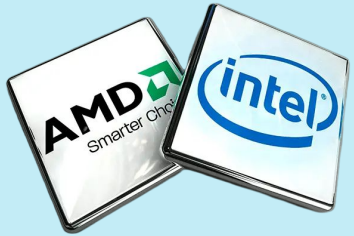
Model: Explicit communication. "Rank 0 sends data to Rank 1."

Pros: Highly optimized for CPU clusters.

Cons: Difficult to program; hard to adapt for dynamic AI graphs.



The Accelerator Shift

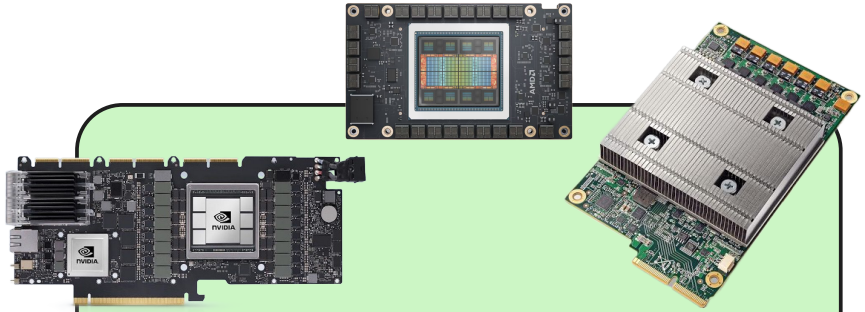


CPU (Latency Oriented)

Few powerful cores.

Good for sequential logic, OS, I/O.

Analogy: A few Ferraris transporting packages.



AI Accelerators (Throughput Oriented)

Thousands of smaller cores.

Good for parallel matrix math (Pixels, Neural Nets).

Analogy: A thousand motorbikes transporting packages.

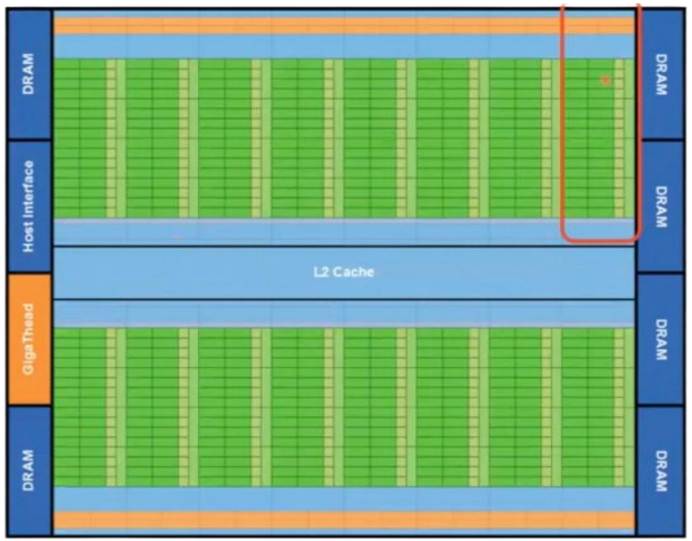
Inside the Architecture

SIMT: Single Instruction, Multiple Threads

In Deep Learning, we perform the same operation (multiplication) on millions of data points.

Streaming Multiprocessors (SMs): The fundamental compute units.

Tensor Cores: Specialized hardware units in NVIDIA GPUs designed strictly for 4x4 matrix multiplication (essential for AI).



The CUDA Programming Model

How do we talk to the GPU?

1. Host to Device

Copy data from CPU RAM to GPU VRAM.

`cudaMemcpy(H2D)`

2. Kernel Launch

CPU tells GPU to execute a function across thousands of threads.

`cudaMemcpy(H2D)`

3. Device to Host

Copy results back to CPU RAM.

`cudaMemcpy(D2H)`

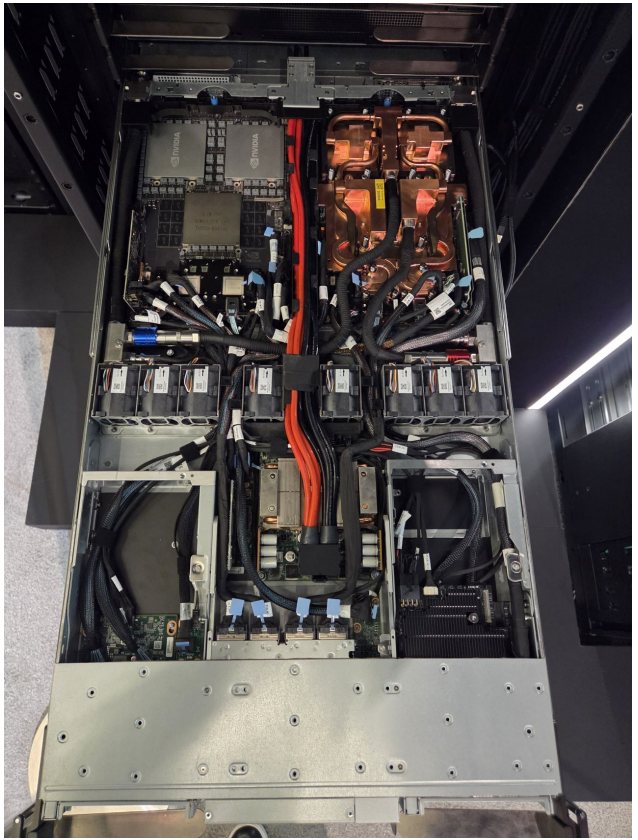
The Memory Bottleneck

Compute is Fast, Data is Slow

The GPU cores are often idle waiting for data to arrive.

HBM (High Bandwidth Memory): ~2000+ GB/s.
Stacked directly on the GPU chip.

Challenge: HBM is expensive and small (144 GB max.).
Large Model doesn't fit it.



© Konstantin Cvetanov - Linkedin

Intelligence: Neural Networks

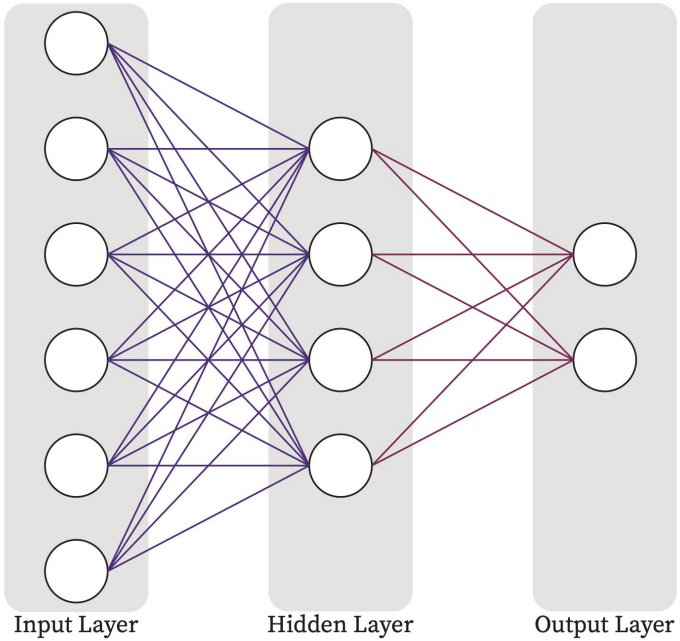
Intelligence: Neural Networks

1. Forward Pass
Pass data through the model. Calculate prediction.

2. Loss Calculation
Compare prediction to actual label.
(Error = Prediction - Truth).

3. Backward Pass
Use Chain Rule (Calculus) to find how much each weight contributed to the error.

4. Optimizer Step
Nudge weights in the opposite direction of error (SGD/Adam).



$$x_i = [x_1\ x_2\ x_3\ x_4\ x_5\ x_6] \qquad y_i = [y_1\ y_2]$$

$$W_1 = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_5 & w_6 & w_7 & w_8 \\ w_9 & w_{10} & w_{11} & w_{12} \\ w_{13} & w_{14} & w_{15} & w_{16} \\ w_{17} & w_{18} & w_{19} & w_{20} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \qquad W_2 = \begin{bmatrix} z_1 & z_2 \\ z_3 & z_4 \\ z_5 & z_6 \\ z_7 & z_8 \end{bmatrix}$$

$$b_1 = [b_1\ b_2\ b_3\ b_4] \qquad b_2 = [c_1\ c_2]$$

PyTorch: The New Lingua Franca

Why PyTorch?

Replaced C++ for research because of its flexibility.

Dynamic Computation Graph: Define the model as you run it (Pythonic).

Autograd: Automatically calculates gradients. No manual calculus needed.

Distributed Support: Built-in primitives for multi-GPU.

```
import torch

# Define a Tensor on GPU
x = torch.randn(5, 3).cuda()

# Automatic Gradient Calculation
y = x * 2
y.backward()  # Magic happens here
```

The Cost of Intelligence

Why do we need a High Performance Computing?

10^{18}

FLOPs (Floating Point Operations) needed to train a modern Transformer model.

Months

Time required on a single GPU to train GPT-3.

Breaking the Single Node Barrier

When One GPU is Not Enough

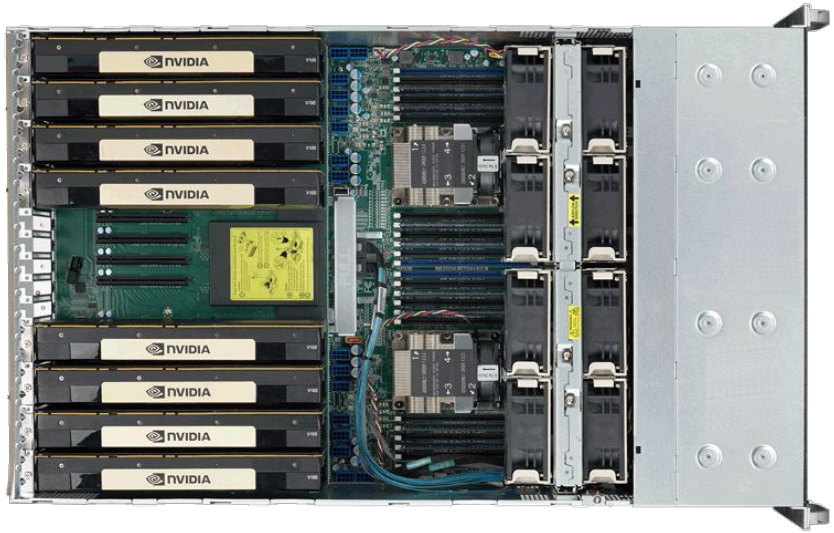
Two main scenarios force us to scale out :

1. Dataset is too large: Taking too long to iterate through data (Epochs).

Solution: Data Parallelism

2. Model is too large: Parameters don't fit in VRAM (OOM Error).

Solution: Model Parallelism



8 GPUs in a single node

Single GPU : Efficiency at One

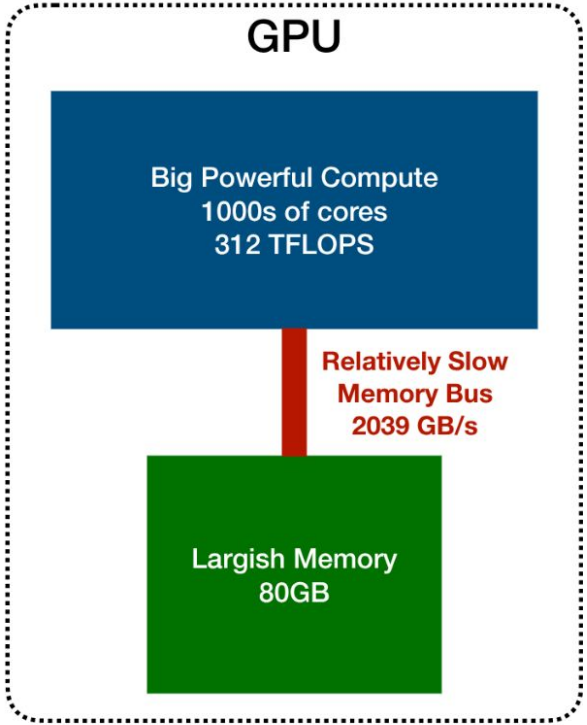
GPU Problems

GPUs have powerful compute and large memory but can only access it slowly
Best for large compute heavy operations

- Do multiple operations per memory load
- Good: Matrix-Matrix Multiplication
- Bad: Element-wise operations (e.g. addition)
- Bad: Operations on many small inputs

Tips for good performance:

- Use large batch sizes (parallelize)
- Fuse small and element-wise operations (torch.compile)
- Pre-load data to avoid stalling the GPU



Adopted from Atil Kosson

Mixed Precision Training: Floating Point Formats

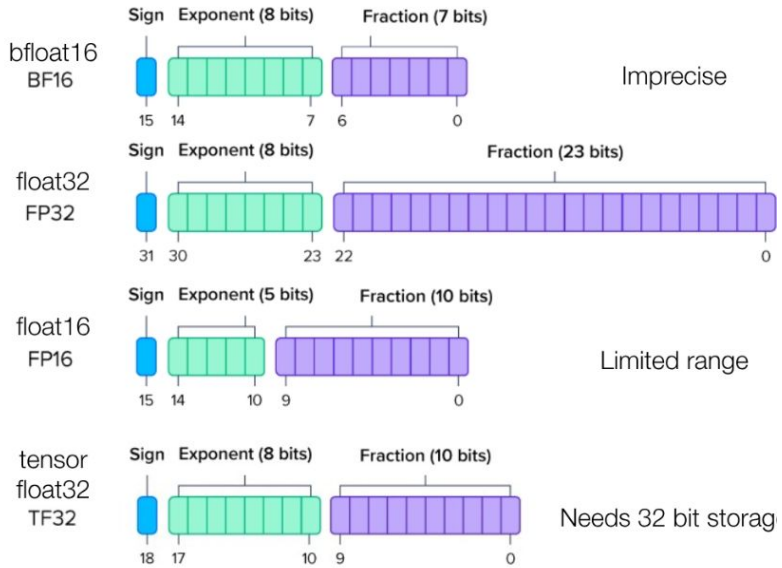
- Computers approximate real numbers with floating point numbers
- Correspond to the scientific notation in decimal for example $\pm 1.23 \cdot 10^{-3}$
- Formats allow for different numbers of significant digits and exponent range
- Get numerical approximation errors (e.g. with 2 significant digits and exponent in [-2,2])
  a. Rounding: $10/3 + 10/3 + 10/3 = 3.3 + 3.3 + 3.3 = 9.9 \neq 10.0$
  b. Swamping: $\sum_{i=1}^{1000} 1.0 = 1.0 \cdot 10^2 \neq 1000$ **if done directly because with limited precision** $1.0 \cdot 10^2 + 1.0 \cdot 10^0 = 1.0 \cdot 10^2$
  c. Underflow: $0.01/10^2 = 0.0$

Less precise formats are cheaper to load and operate on => speedup

Adopted from Atil Kosson

Mixed Precision Training: Floating Point Formats

- A100 performance:
  - fp32 19.5 TFLOPS
  - fp16 312 TFLOPS
  - bf16 312 TFLOPS
  - tf32 156 TFLOPS
- Using lower precision formats can be much faster but may result in numerical issues
- Mixed Precision Training uses different formats depending on the type of operation, resulting in speedups while (hopefully) avoiding the downsides



Adopted from Atil Kosson

Mixed Precision Training: In Practice

```
use_amp = True
net = make_model(in_size, out_size, num_layers)
opt = torch.optim.SGD(net.parameters(), lr=0.001)
scaler = torch.cuda.amp.GradScaler(enabled=use_amp)
start_timer()
for epoch in range(epochs):
    for input, target in zip(data, targets):
        with torch.autocast(device_type=device, dtype=torch.float16, enabled=use_amp):
            output = net(input)
            loss = loss_fn(output, target)
        scaler.scale(loss).backward()
        scaler.step(opt)
        scaler.update()
        opt.zero_grad(set_to_none=True)
```

PyTorch handles most of this for you:

- torch.autocast
- torch.cuda.amp.GradScaler (needed for float16, but not bfloat16)
- TF32 is used by default for some ops if available but need to enable for matmuls with torch.backends.cuda.matmul.allow_tf32 = True

Note: Some extra memory overhead and optimizer states are all kept in float32

Scalability:
Scale the AI Workload

Scalability: DDP

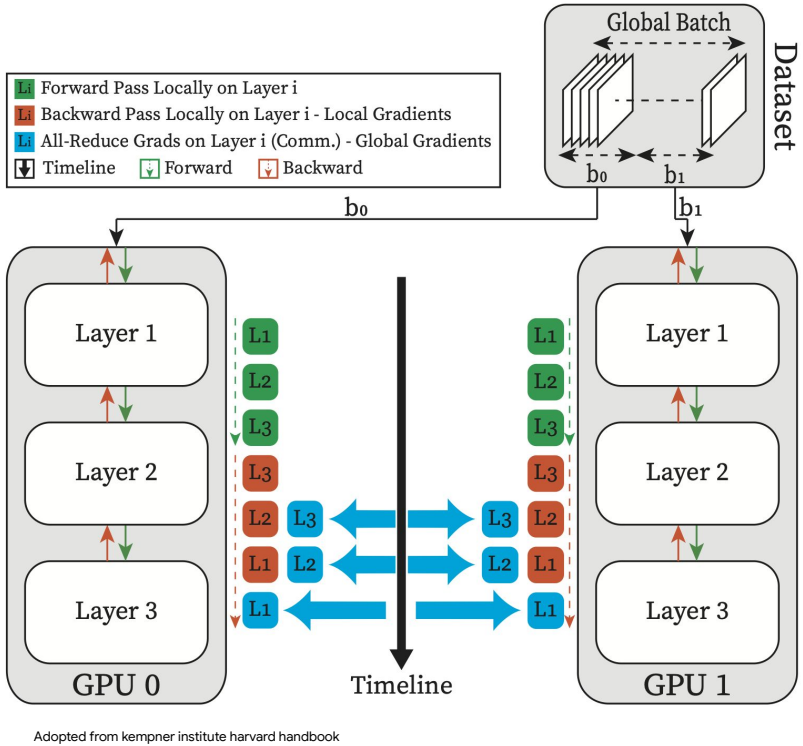
Distributed Data Parallel (DDP)

The workhorse of modern AI training.

Replication: Copy the model to every GPU.

Sharding: Split the dataset into chunks. Each GPU sees different data.

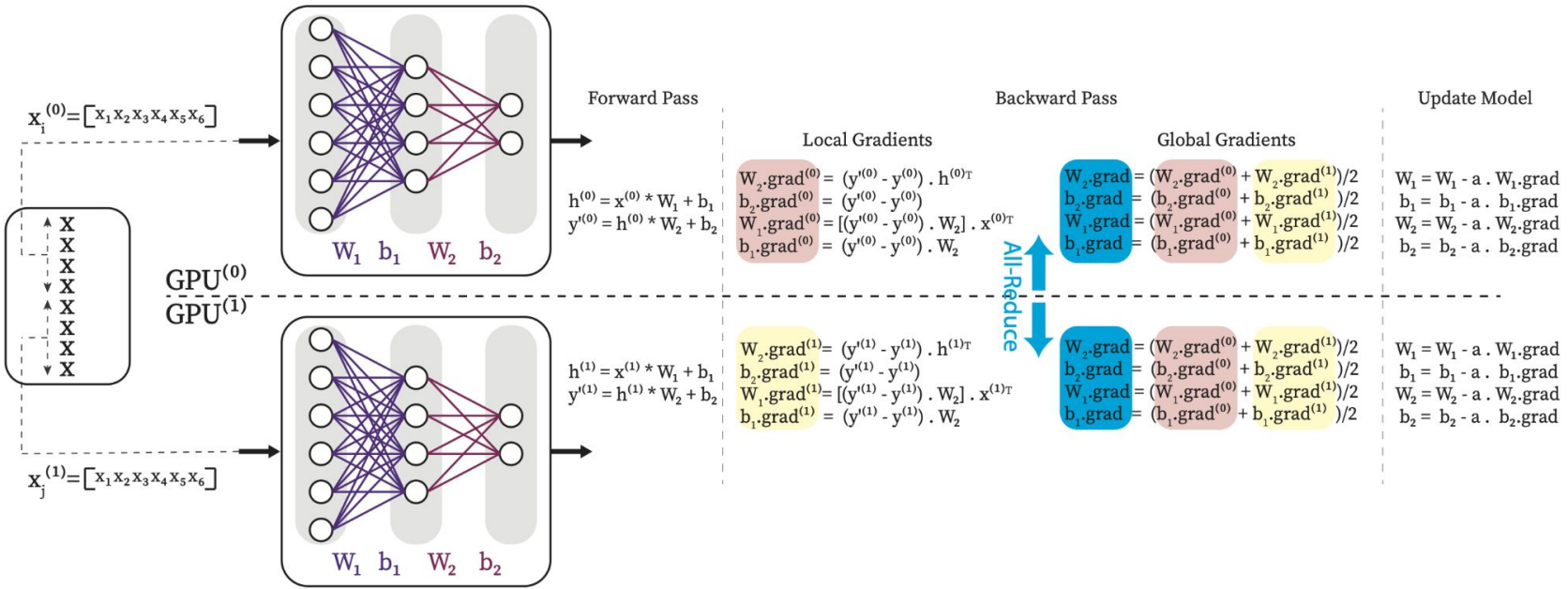
Synchronization: Average the gradients from all GPUs before updating weights.



Adopted from kempner institute harvard handbook

Scalability: DDP

Distributed Data Parallel (DDP)



Adopted from kempner institute harvard handbook

Scalability: MP

Model Parallelism (MP)

Too large to fit into a single GPU

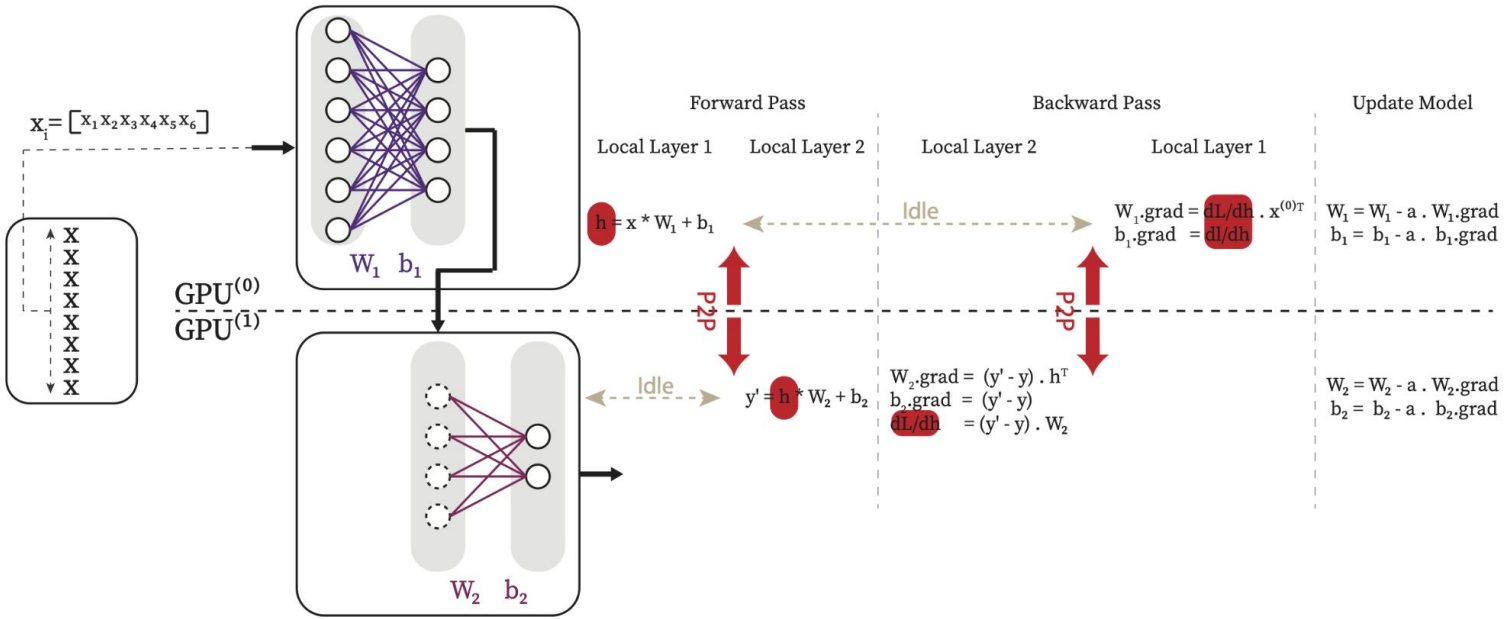
Focuses on parallelizing the model

A naive implementation could be dividing the model

Scalability: MP

Cons: Any given time only one of the GPUs are active and the other GPUs are idle leading to underutilization of the compute resources.

Model Parallelism (MP)



Adopted from kempner institute harvard handbook

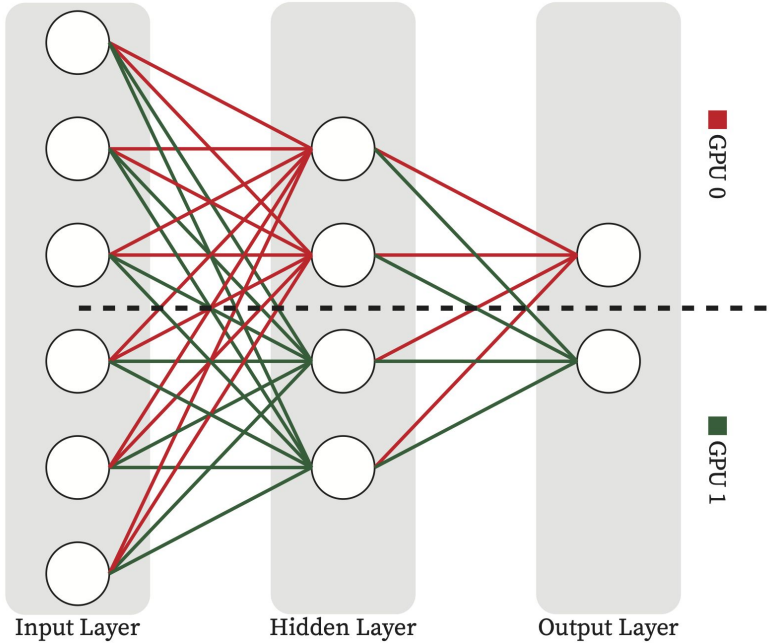
Splitting the Brain: Tensor Parallelism

When the Model > VRAM

We cannot replicate the model. We must split the matrices themselves.

Concept: Split a giant Matrix Multiplication A x B = C across GPUs.

Constraint: Requires extremely high bandwidth (NVLink) because communication happens inside every layer.



Adopted from kempner institute harvard handbook

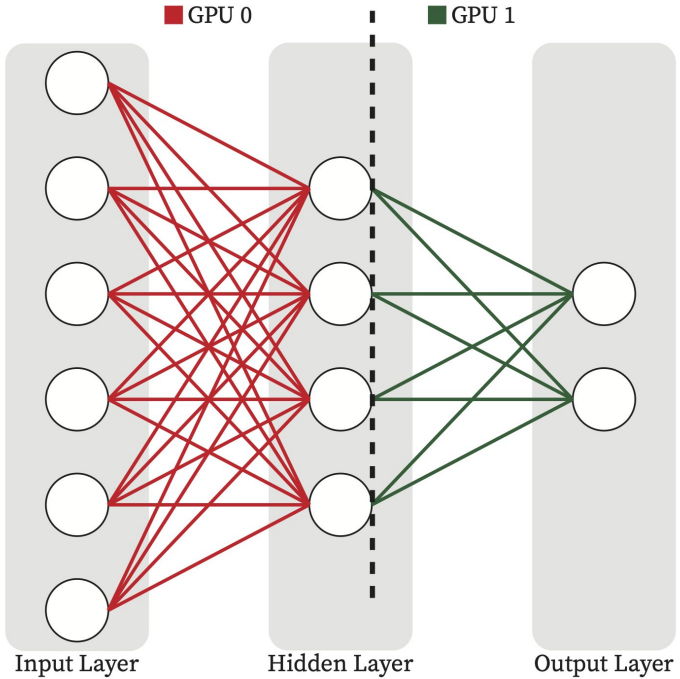
The Assembly Line: Pipeline Parallelism

Vertical Splitting

Place layers 1-10 on GPU 0, layers 11-20 on GPU 1.

Problem: GPU 1 sits idle while GPU 0 works (The Bubble).

Solution: Micro-batching. Feed small chunks rapidly to keep the pipeline full.



Adopted from kempner institute harvard handbook

Speaking the Same Language: NCCL

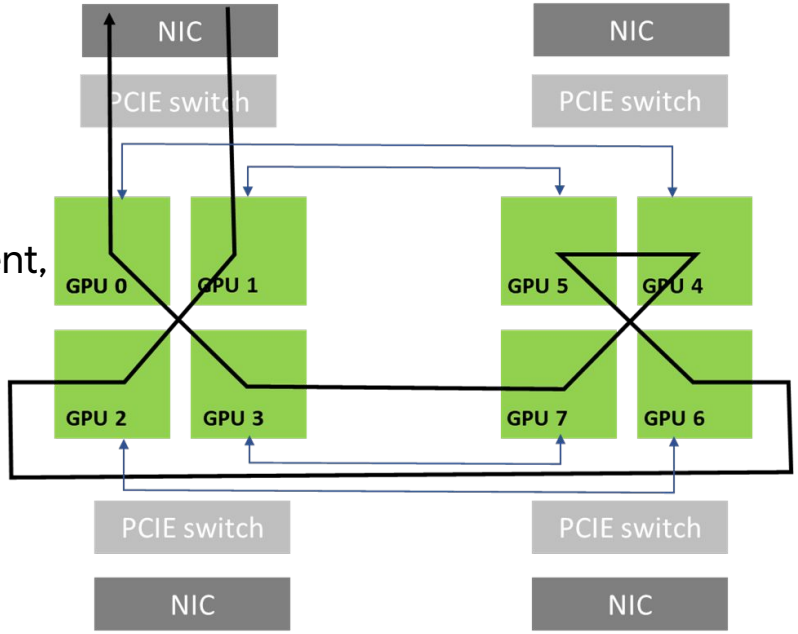
How do GPUs talk?

The "Ring All-Reduce" algorithm.

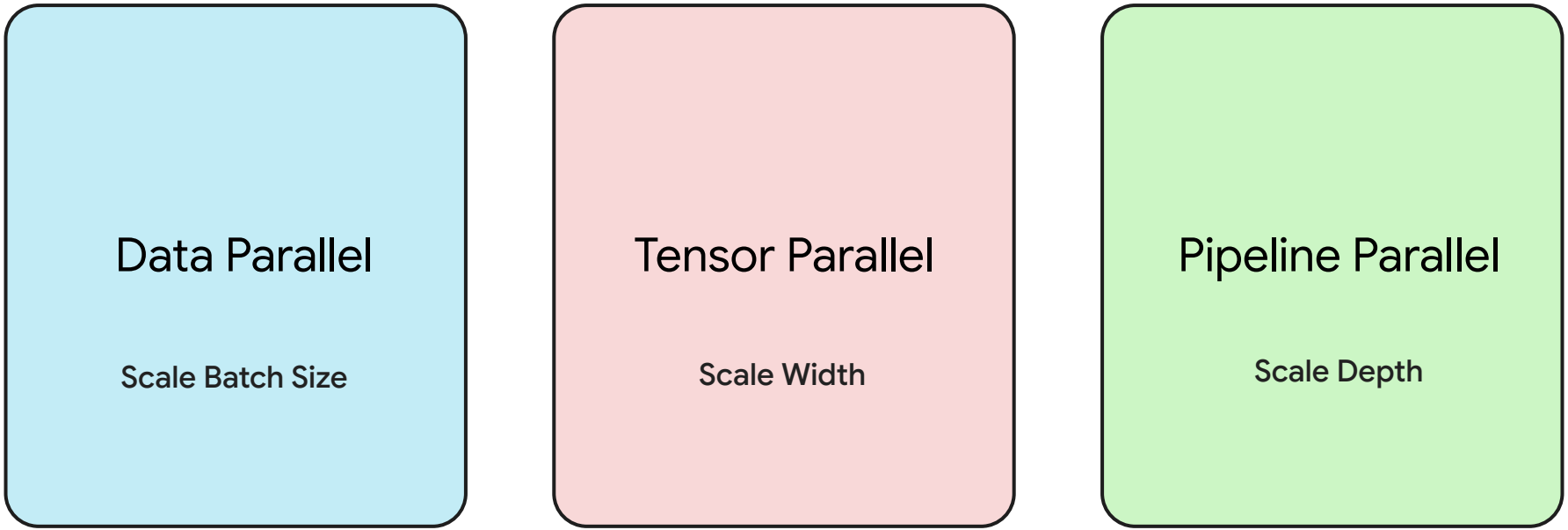
Data flows in a logical ring.

Each GPU receives a chunk, adds its own gradient, and passes it on.

Bandwidth Optimal!



The Holy Grail: 3D Parallelism



Combining all three allows us to train models with Trillions of parameters (like GPT-5) on thousands of GPUs.

Language Abstraction: LLMs

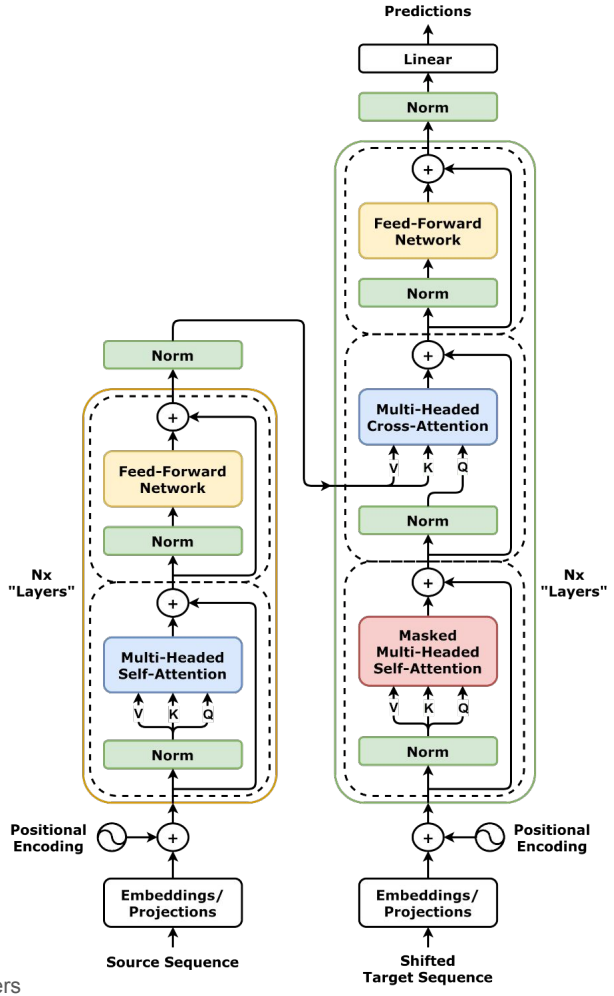
Language Abstraction: LLMs

The Transformer Revolution

The architecture that changed everything. It allows us to process massive sequences of text in parallel.

Self-Attention: The model learns which words relate to each other, regardless of distance.

Generative: Predicts the next token based on context.



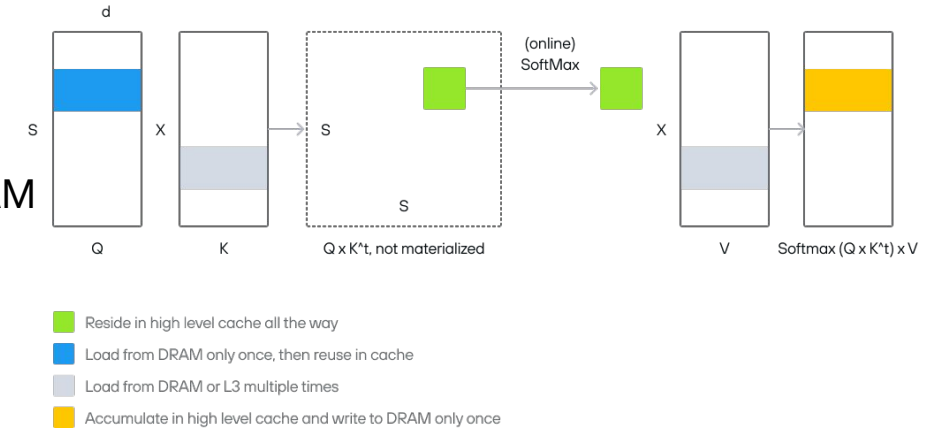
Optimizing Attention: FlashAttention

The IO-Aware Algorithm

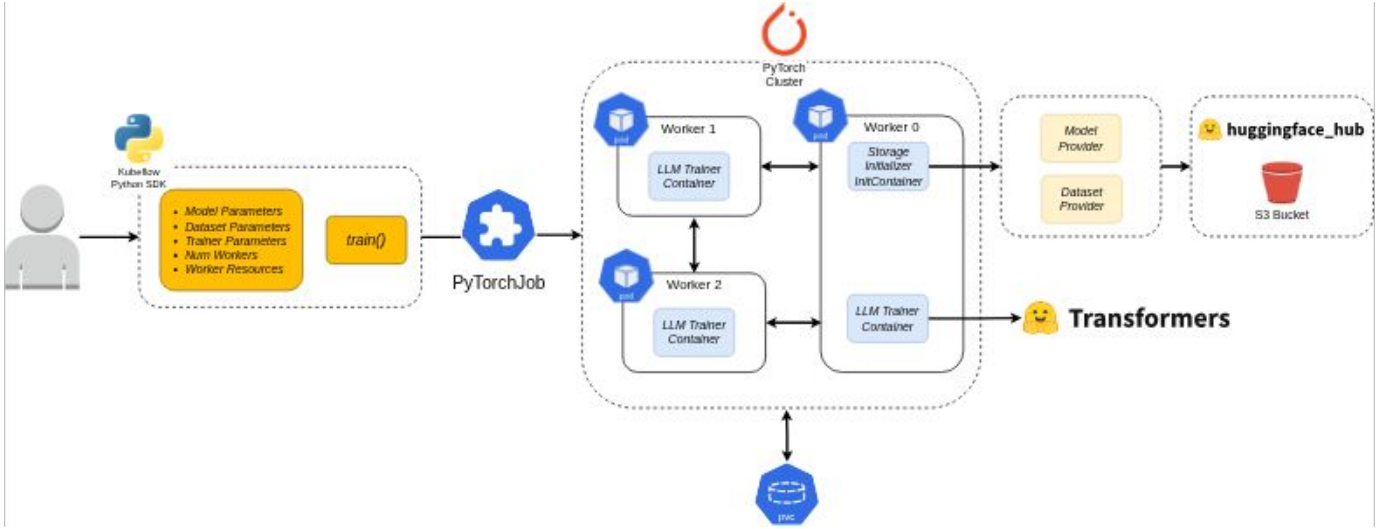
Standard Attention reads/writes to HBM too often ($O(N^2)$ memory access).

FlashAttention: Keeps data in the GPU's fast SRAM (cache), computing attention in blocks.

Result: 3 x times faster for longer sequences.



End-to-End LLM Workflow



1. Pre-training

Learn grammar/facts from massive text corpus. (Expensive)

2. SFT

Supervised Fine-Tuning. Learn instruction following. (Cheap)

3. RLHF

Reinforcement Learning. Align with human values.

Scan for Slides & Codes

End

Thank You!

Any Questions ?

hpc@tu.edu.np
hpc.tu.edu.np
aatiz.ghimire@heraldcollege.edu.np
aatizghimire
aatizghimire
aatizghimire