

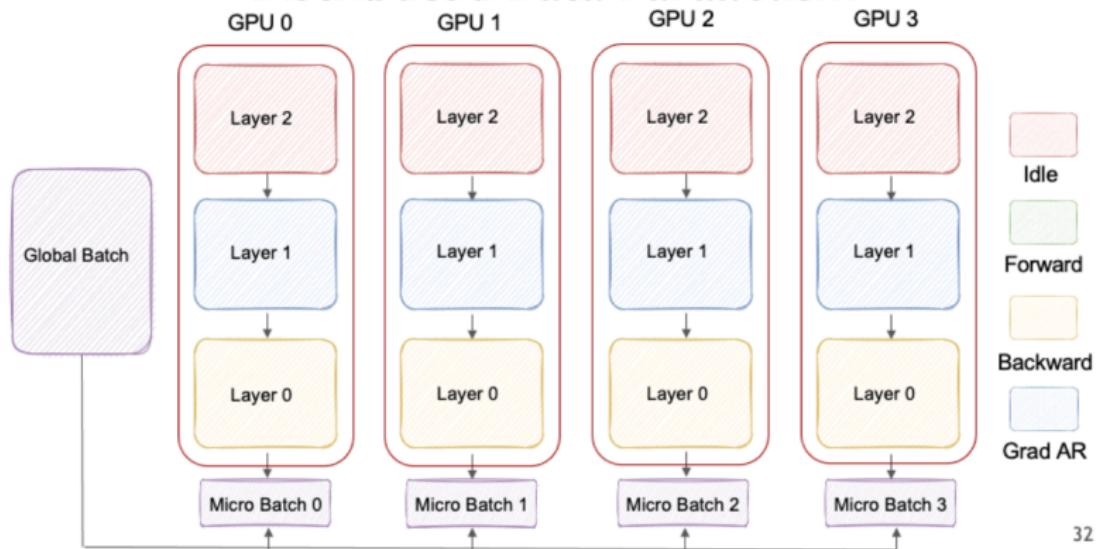
RESTORE LECTURE COMMAND

DISTRIBUTED TRAINING

- Training LLMs faster on many GPUs
- Avoiding OOM issues
- **Data parallelism:** split the data on different model replicas
- **Tensor parallelism:** split model parameters across GPUs

DATA PARALLELISM (1) (ANIMATED GIF!)

Distributed Data Parallelism



Source: Nvidia

DATA PARALLELISM (2)

- **Data Splitting:**

- The dataset is divided into smaller chunks, and each chunk is assigned to a different processing unit (e.g., GPU or CPU) on different nodes
- Each node processes a different subset of the data in parallel, reducing the overall training time

- **Model Replication:**

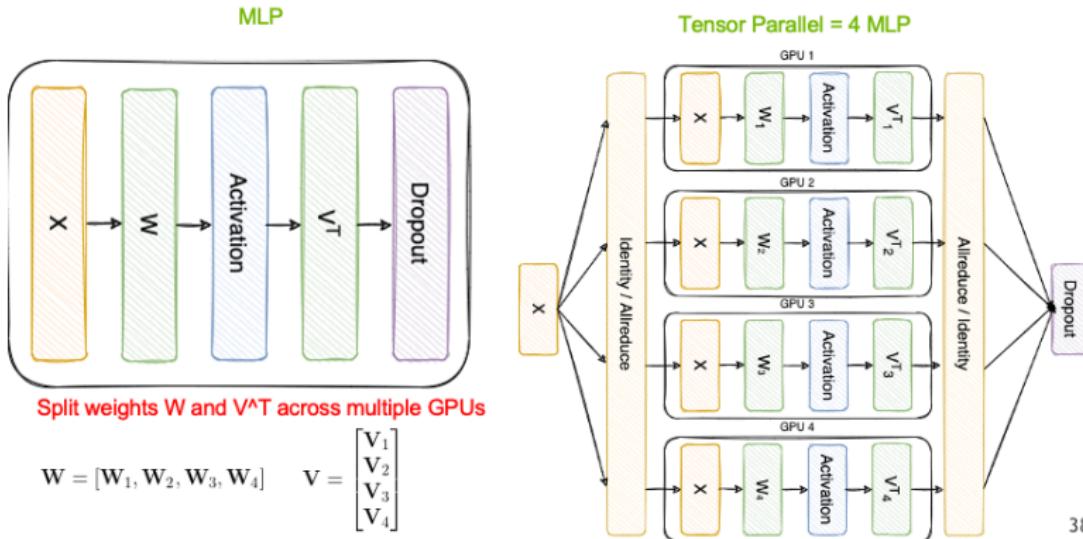
- Each processing unit has a replica of the neural network model
- These replicas are trained independently on their respective data subsets

- **Gradient Aggregation:**

- After each forward and backward pass, gradients are computed locally on each node

TENSOR PARALLELISM (1) (ANIMATED GIF!)

Tensor Parallelism (Intra-Layer)



Source: Nvidia

TENSOR PARALLELISM (2)

- **Model Partitioning:**

- The model's layers or tensors are split across multiple devices
- Different parts of the model are assigned to different devices, enabling them to work on separate portions of the computations simultaneously

- **Forward and Backward Passes:**

- During the forward pass, each device processes its portion of the tensors with intermediate results passed between devices
- In the backward pass, gradients are computed in the reverse order, again with necessary data transfers between devices

- **Parameter Updates:**

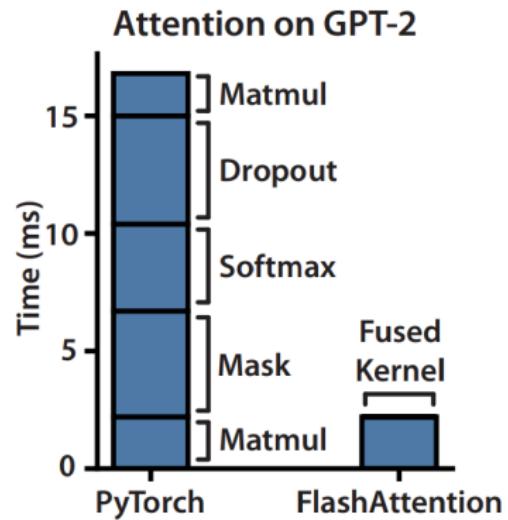
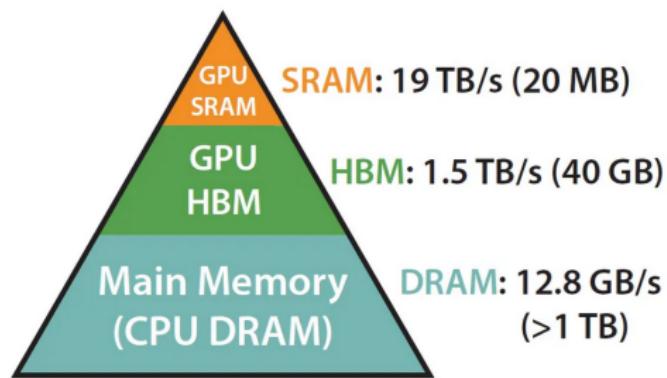
- Parameter updates can be performed independently on each device for the parameters they own
- After each update step, the devices synchronize to ensure consistency across the distributed model

FlashAttention

Fast and Memory-Efficient Exact Attention with IO-Awareness

- Fast
 - 15 % faster than BERT
 - 3x faster than GPT-2
 - 2.4x faster than Megatron-LM
- Memory-efficient
 - Reducing from $O(n^2)$ to $O(n)$
- Exact
 - Same as “vanilla attention”, not an approximation
- IO aware
 - Reducing memory load/store operations

GPU MEMORY HIERARCHY



► Source: Dao, 2022

GPU MEMORY HIERARCHY

The diagram illustrates matrix multiplication ($A \times B = C$) while highlighting memory access patterns. Matrix A (3x3) has a purple block (row 1, col 2), a blue block (row 2, col 1), and a green block (row 2, col 2). Matrix B (3x3) has a purple block (row 1, col 1), a blue block (row 1, col 2), and a green block (row 2, col 1). The result, Matrix C (3x3), contains an orange block at position (1,1) and a green block at position (2,1). This visualizes how specific memory locations from both matrices are accessed to produce the result.

Matrix A

Matrix B

\times

$=$

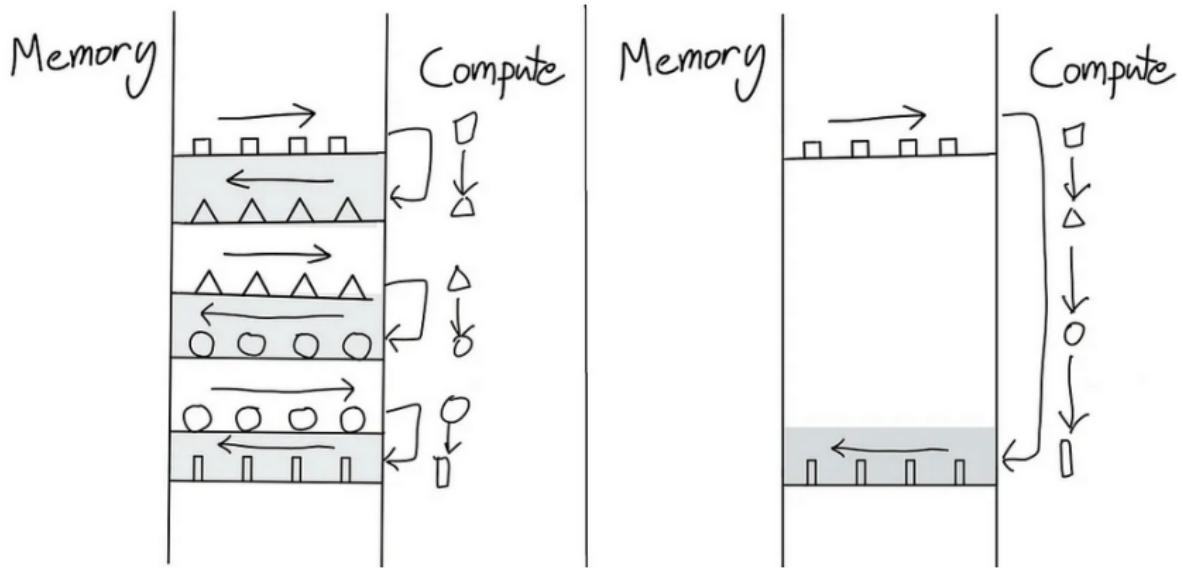
Matrix C

► Source: Dao, 2022

COMPUTING CONSIDERATIONS

- GPU compute has been growing faster than memory bandwidth
 - GPU has to wait for data
- Transformer operations are memory-bound
 - Elementwise operations with high memory access
- IO aware means reducing memory load/store operations
- FlashAttention implements the following:
 - Operation fusion to reduce memory access
 - Tiling or chunking the softmax matrix into blocks
 - Recomputation for better memory utilization

OPERATION FUSION



Source: https://horace.io/brrr_intro.html

LIMITATIONS AND PROSPECTS

- FlashAttention requires writing attention to CUDA language
 - A new CUDA kernel for each new attention implementation
 - CUDA is lower-level than PyTorch
 - Implementation may not be transferable across GPUs
- Towards IO-Aware Deep Learning
 - Extending beyond attention
- Multi-GPU IO-Aware Methods
 - FlashAttention computation may be parallelizable across multiple GPUs