



Foundations of LLM Mastery: Fine-tuning with multi GPUs

25 February 2025
ONLINE



Large Language Model Parallelizaion

Or: how to chop up a Llama

Speaker: Simeon Harrison
Trainer at EuroCC Austria

Problems Arise

Data and Model too large

You might quickly encounter a situation in which your data and model no longer fit in your GPU's memory.

Memory footprint estimation for Mistral 7B (half precision):

$7 \times 2 = 14$ GB for the weights

$7 \times 2 = 14$ GB for the gradients

$7 \times 2 \times 2 = 28$ GB for the optimizer state(s)

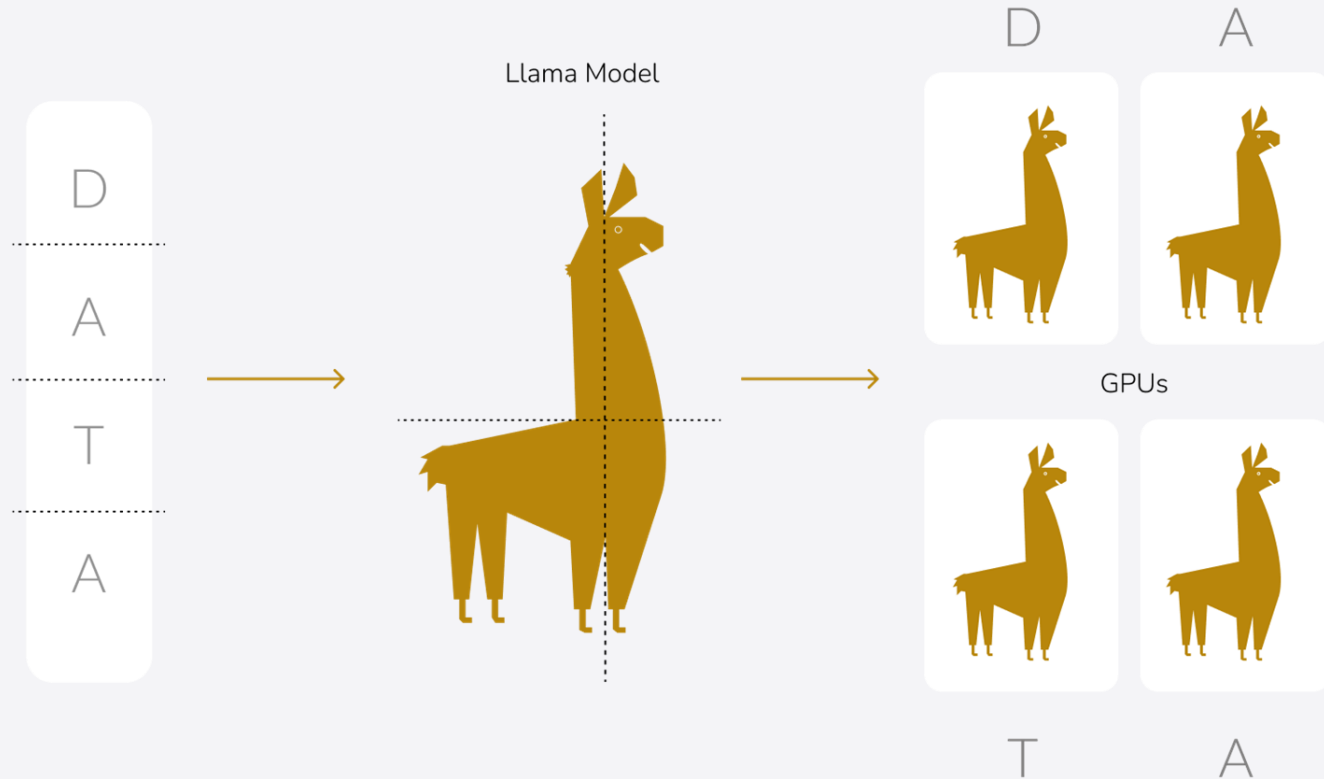
Total of 56GB for the model only!

7 comes from 7B parameters

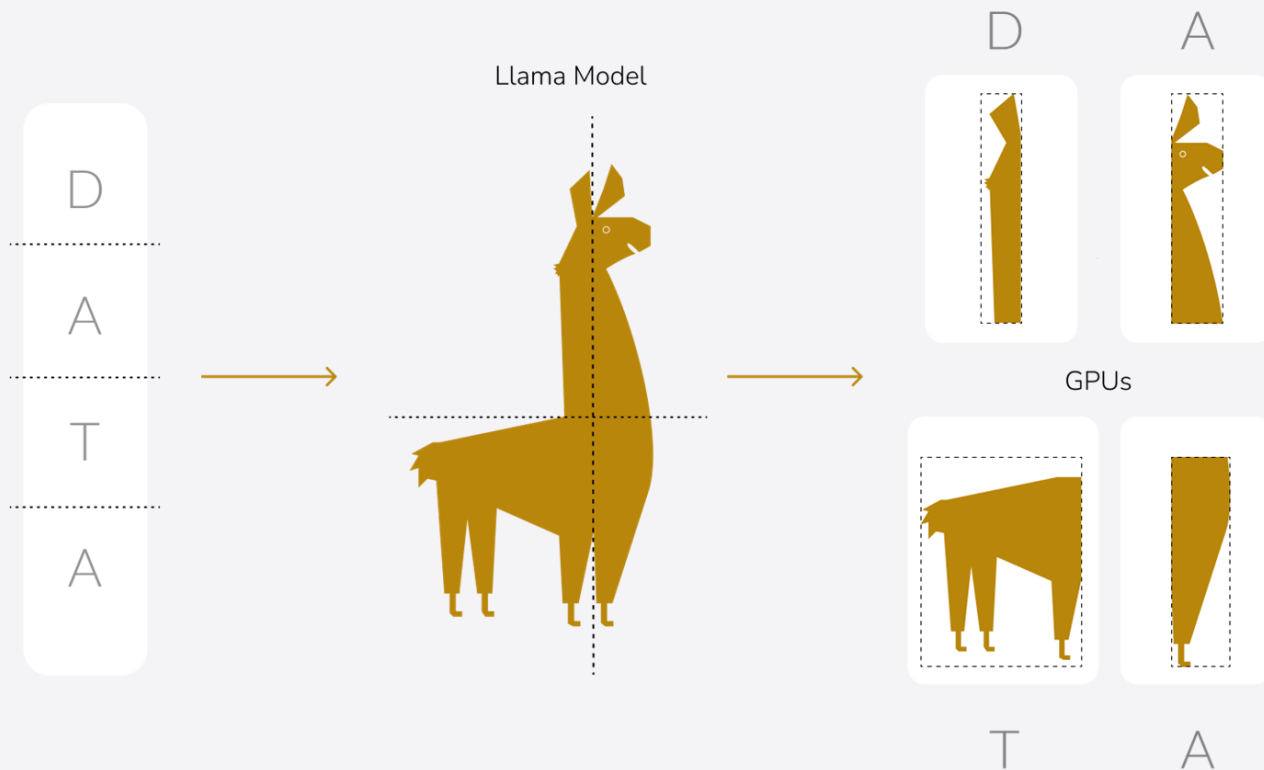
2 stands for 2 Bytes per parameter



Data Parallelism



Model Parallelism



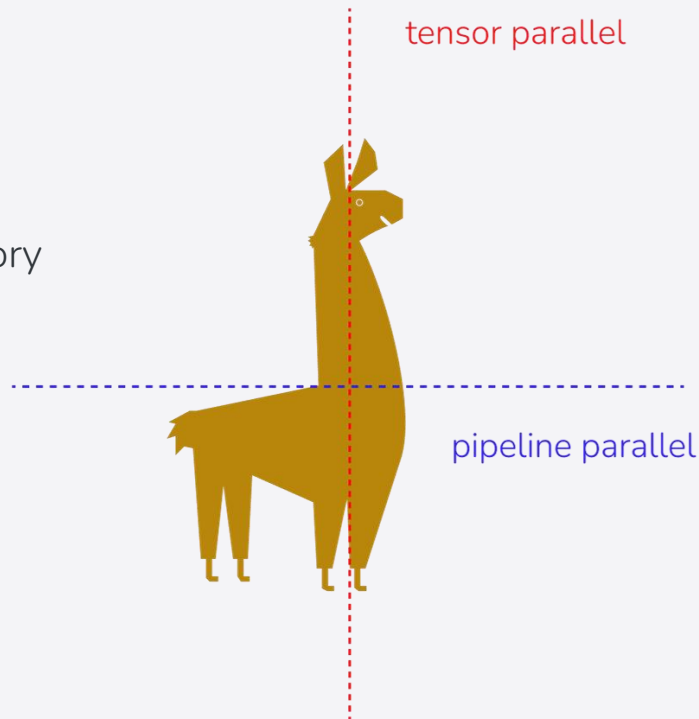
Model Parallelism

Pipeline parallel

- Model split up along layers
- Each GPU gets one or several layers
- Results are synced at the end of every step
- Important: Largest layer needs to fit in GPU's memory

Tensor parallel

- Every tensor is split up into several chunks
- One GPU gets one shard of the whole tensor
- Each shard gets processed separately
- Results are synced at the end of every step



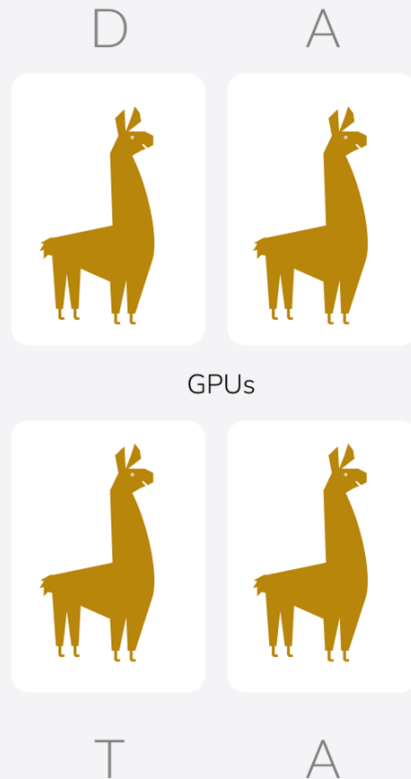
DDP

DistributedDataParallel

replicates the model across GPUs and processes batches independently on each GPU.

DDP synchronizes gradients efficiently across GPUs, making it more scalable and performant, especially when dealing with large models or clusters of machines.

Ideal, if model fits in GPUs memory with ease.



Hugging Face Accelerate

Accelerate is a library that enables the same PyTorch code to be run across any distributed configuration by adding just a few lines of code.

It simplifies the use of advanced parallelism techniques such as Fully Sharded Data Parallel (FSDP) and ZeRO (Zero Redundancy Optimizer), allowing users to scale large models across multiple GPUs and nodes with minimal changes to their code.

```
+ from accelerate import Accelerator
+ accelerator = Accelerator()

+ model, optimizer, training_dataloader, scheduler = accelerator.prepare(
+     model, optimizer, training_dataloader, scheduler
+ )

for batch in training_dataloader:
    optimizer.zero_grad()
    inputs, targets = batch
    inputs = inputs.to(device)
    targets = targets.to(device)
    outputs = model(inputs)
    loss = loss_function(outputs, targets)
+     accelerator.backward(loss)
    optimizer.step()
    scheduler.step()
```

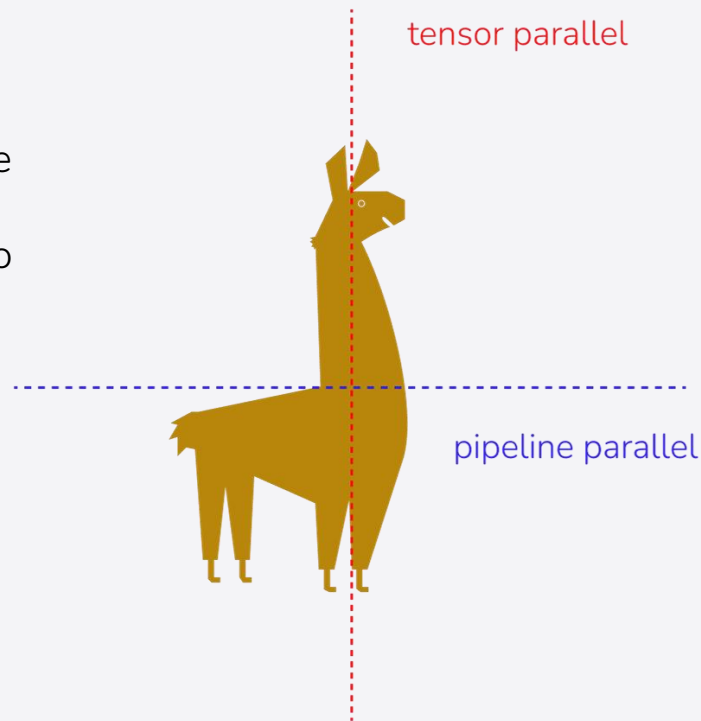

ZeRO with DeepSpeed

Zero Redundancy Optimizer

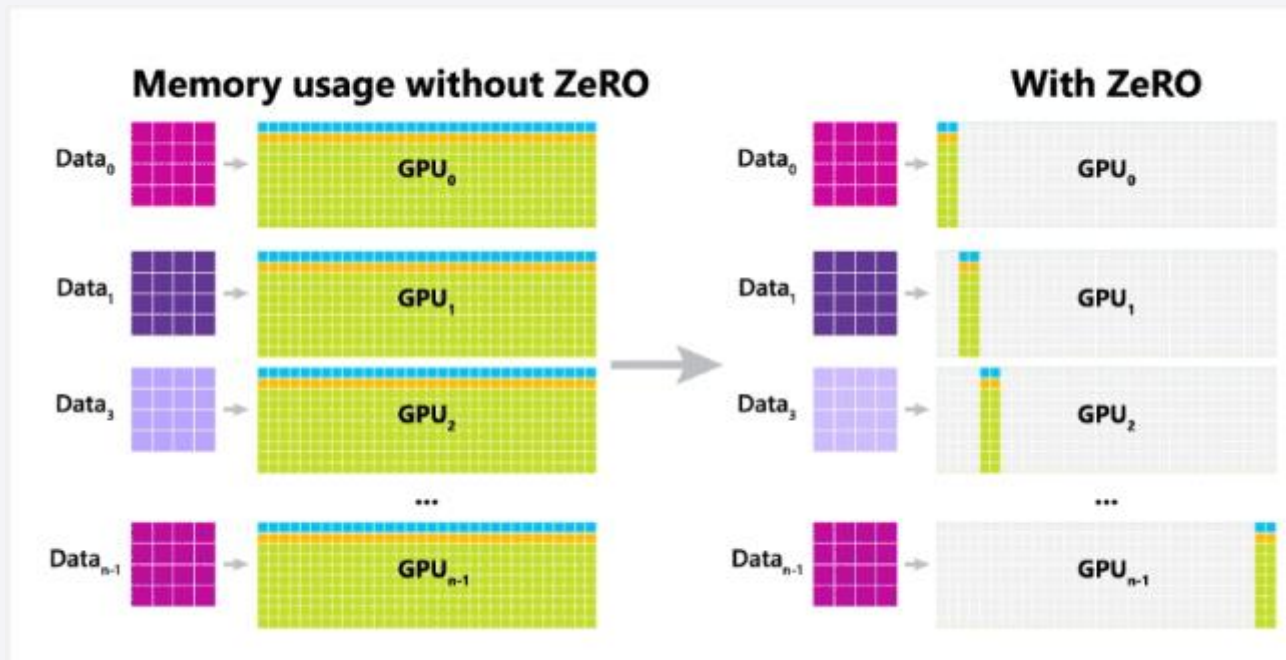
Is a memory optimization technique designed to scale large models efficiently across multiple GPUs. The core idea behind ZeRO is to minimize memory redundancy when training large-scale models, making it possible to train models that wouldn't otherwise fit in memory.

- ZeRO Stage 1 - Optimizer State Sharding
- ZeRO Stage 2 - Gradient Sharding
- ZeRO Stage 3 - Parameter

ZeRO is typically used with DeepSpeed



ZeRO with DeepSpeed



Source: <https://www.microsoft.com/en-us/research/blog/>

FSDP

Fully Sharded Data Parallel

To accelerate training huge models on larger batch sizes, we can use a fully sharded data parallel model. This type of data parallel paradigm enables fitting more data and larger models by sharding the optimizer states, gradients and parameters.

Mapping between FSDP and ZeRO

- `FULL_SHARD` maps to the DeepSpeed ZeRO Stage-3.
- `SHARD_GRAD_OP` maps to the DeepSpeed ZeRO Stage-2.
- `NO_SHARD` maps to ZeRO Stage-0.

THANK YOU



This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101101903. The JU receives support from the Digital Europe Programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia