Deep Learning with Multiple GPUs

How to Use Multiple GPUs for Deep Learning

Deep learning is a subset of machine learning that does not rely on structured data to develop accurate predictive models. This method uses networks of algorithms modeled after neural networks in the brain to distill and correlate large amounts of data. The more data you feed your network, the more accurate the model becomes. You can functionally train deep learning models using sequential processing methods. However, the amount of data needed and the length of data processing make it impractical if not impossible to train models without parallel processing.

Parallel processing enables multiple data objects to be processed at the same time, drastically reducing training time. This parallel processing is typically accomplished through the use of graphical processing units (GPUs). GPUs are specialized processors created to work in parallel. These units can provide significant advantages over traditional CPUs, including up to 10x more speed. Typically, multiple GPUs are built into a system in addition to CPUs. While the CPUs can handle more complex or general tasks, the GPUs can handle specific, highly repetitive processing tasks.

In this guide, you will learn:

Also refer to our other detailed guides about:

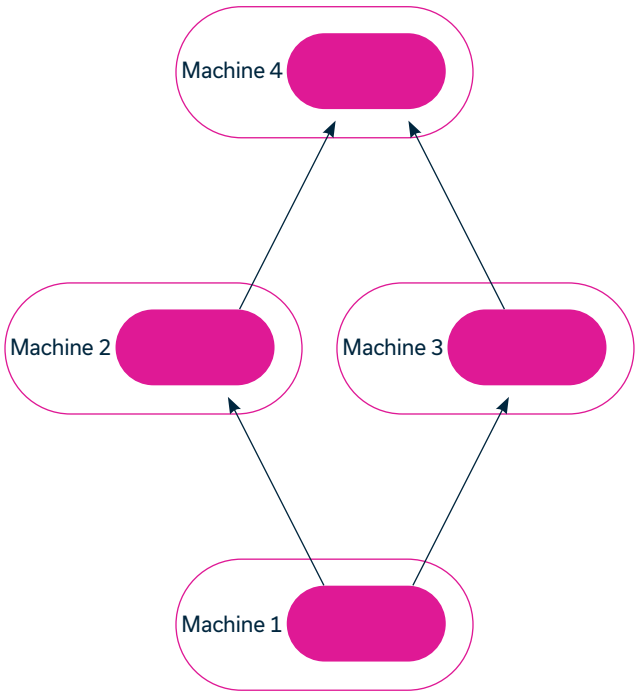
Multi GPU Deep Learning Strategies

Once multiple GPUs are added to your systems, you need to build parallelism into your deep learning processes. There are two main methods to add parallelism—models and data.

Model Parallelism

Model parallelism is a method you can use when your parameters are too large for your memory constraints.
Using this method, you split your model training processes across multiple GPUs and perform each process in parallel (as illustrated in the image below) or in series. Model parallelism uses the same dataset for each portion of your model and requires synchronizing data between the splits.

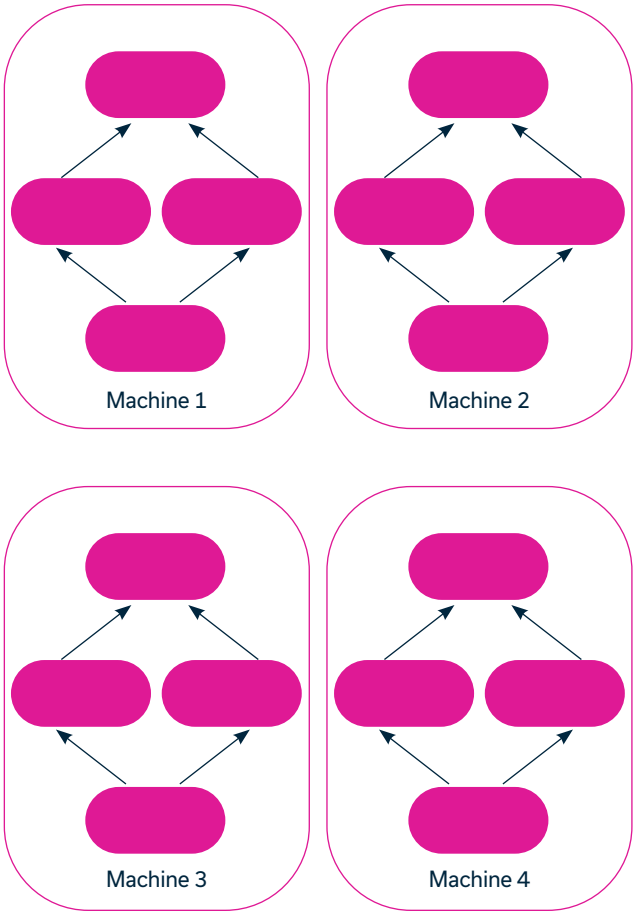
Model Parallelism



Data Parallelism

Data parallelism is a method that uses duplicates of your model across GPUs. This method is useful when the batch size used by your model is too large to fit on a single machine, or when you want to speed up the training process. With data parallelism, each copy of your model is trained on a subset of your dataset simultaneously. Once done, the results of the models are combined and training continues as normal.

Data Parallelism



How Does Using Multiple GPUs Work in Common Deep Learning Frameworks?

When working with deep learning models, there are several frameworks you may use, including Keras, PyTorch and TensorFlow. Depending on the framework you choose, there are different ways to implement multi GPU systems.

TensorFlow

TensorFlow is an open source framework, created by Google, that you can use to perform machine learning operations. The library includes a variety of machine learning and deep learning algorithms and models that you can use as a base for your training. It also includes built-in methods for distributed training using GPUs.

Through the API, you can use the tf.distribute.Strategy method to distribute your operations across GPUs, TPUs or machines. This method enables you to create and support multiple user segments and to switch between distributed strategies easily.

Two additional strategies that extend the distribute method are MirroredStrategy and TPUStrategy. Both of these enable you to distribute your workloads, the former across multiple GPUs and the latter across multiple Tensor Processing Units (TPUs). TPUs are units available through Google Cloud Platform that are specifically optimized for training with TensorFlow.

Both of these methods use roughly the same data-parallel process, summarized as follows:

- Your dataset is segmented so data is distributed as evenly as possible.
- Replicas of your model are created and assigned to a GPU. Then, a subset of the dataset is assigned to that replica
- The subset for each GPU is processed and gradients are produced
- The gradients from all model replicas are averaged and the result is used to update the original model
- The process repeats until your model is fully trained

PyTorch

PyTorch is an open source scientific computing framework based on Python. You can use it to train machine learning models using tensor computations and GPUs. This framework supports distributed training through the torch.distributed backend.

With PyTorch, there are three parallelism (or distribution) classes that you can perform with GPUs. These include:

- **Data Parallel:** enables you to distribute model replicas across multiple GPUs in a single machine. You can then use these models to process different subsets of your data set.
- **Distributed Data Parallel:** extends the DataParallel class to enable you to distribute model replicas across machines in addition to GPUs. You can also use this class in combination with model_parallel to perform both model and data parallelism.
- **Model Parallel:** enables you to split large models across multiple GPUs with partial training happening on each. This requires syncing training data between the GPUs since operations are performed sequentially.

Multi GPU Deployment Models

There are three main deployment models you can use when implementing machine learning operations that use multiple GPUs. The model you use depends on where your resources are hosted and the size of your operations.

GPU Server

GPU servers are servers that incorporate GPUs in combination with one or more CPUs. When workloads are assigned to these servers, the CPUs act as a central management hub for the GPUs, distributing tasks and collecting outputs as available.

GPU Cluster

GPU clusters are computing clusters with nodes that contain one or more GPUs. These clusters can be formed from duplicates of the same GPU (homogeneous) or from different GPUs (heterogeneous). Each node in a cluster is connected via an interconnect to enable the transmission of data.

GPU Cluster

Kubernetes is an open source platform you can use to orchestrate and automate container deployments. This platform offers support for the use of GPUs in clusters to enable workload acceleration, including for deep learning.

When using GPUs with Kubernetes, you can deploy heterogeneous clusters and specify your resources, such as memory requirements. You can also monitor these clusters to ensure reliable performance and optimize GPU utilization.

- Replicas of your model are created and assigned to a GPU. Then, a subset of the dataset is assigned to that replica
- The subset for each GPU is processed and gradients are produced
- The gradients from all model replicas are averaged and the result is used to update the original model
- The process repeats until your model is fully trained

Using Multiple GPUs with Run:ai

Run:ai automates resource management and workload orchestration for machine learning infrastructure. With Run:ai, you can automatically run as many deep learning experiments as needed on multi-GPU infrastructure.

Here are some of the capabilities you gain when using Run:ai:

- **Advanced Visibility:** create an efficient pipeline of resource sharing by pooling GPU compute resources
- **No More Bottlenecks:** you can set up guaranteed quotas of GPU resources, to avoid bottlenecks and optimize billing.
- **A Higher Level of Control:** Run:ai enables you to dynamically change resource allocation, ensuring each job gets the resources it needs at any given time

Run:ai simplifies machine learning infrastructure pipelines, helping data scientists accelerate their productivity and the quality of their models.

Learn more about the Run:ai GPU virtualization platform.