UNIVERSITY OF AMSTERDAM

Multi-GPU Training

High Performance Deep Learning

Eric Marcus & Jonas Teuwen

Overview

Overview of the subjects for the multi-GPU part

Theory

- Why GPU?
- Why multiple GPUS?
- Backends – NCCL, GLOO
- GPU operations – scatter, gather, all reduce etc.

Practice

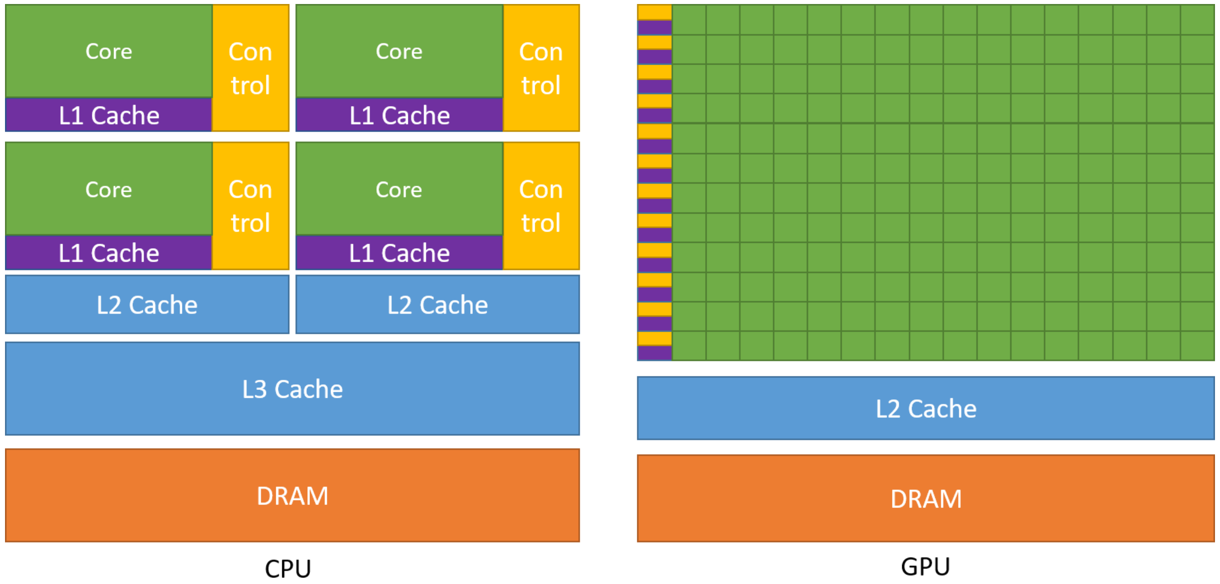
- Data parallel
- Distributed data parallel
- What more can we do?

Practice ++

- Pytorch Lightning
- Tracking performance

Why GPU?

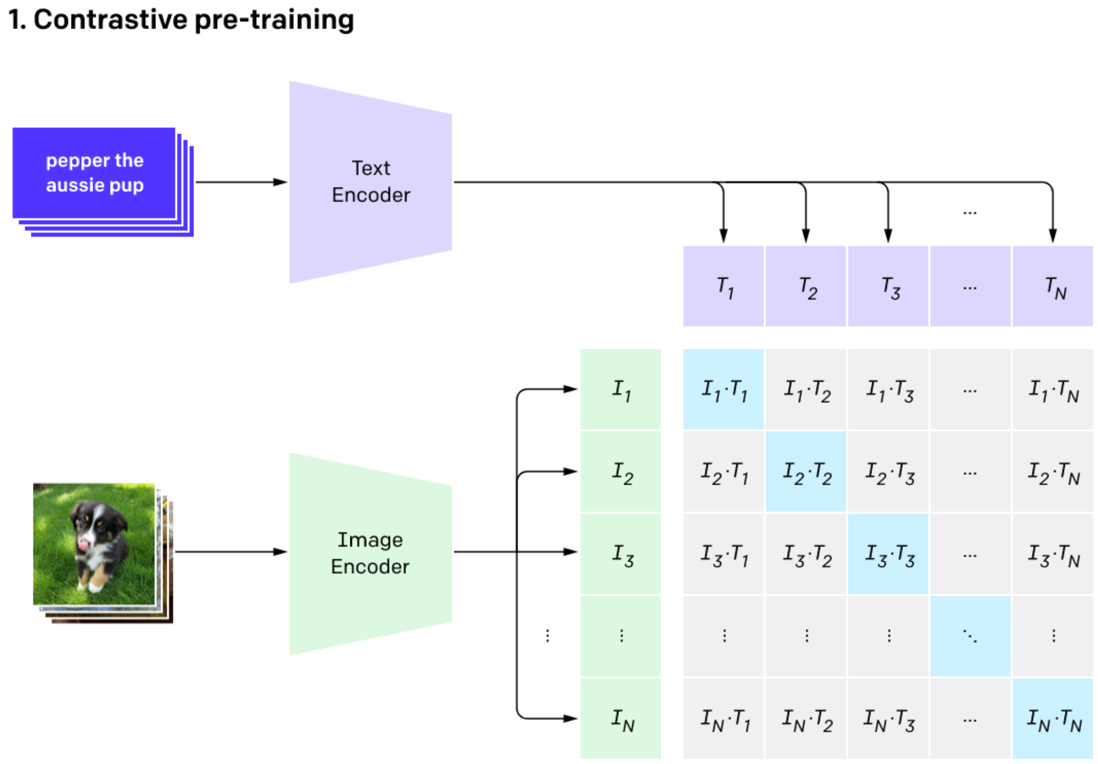
- Neural networks are *embarrassingly parallel*
- The computations (mostly matrix multiplications) can be executed in parallel and are independent
- GPUs excel exactly at such parallel computations due to their architecture



The large amount of cores is ideally suited for independent parallel computations. Source: NVIDIA

Why multiple GPU?

- Example: *OpenAI's CLIP*
- Large model trained on combination of images and text
- Dataset of 400 million (image, text) pairs

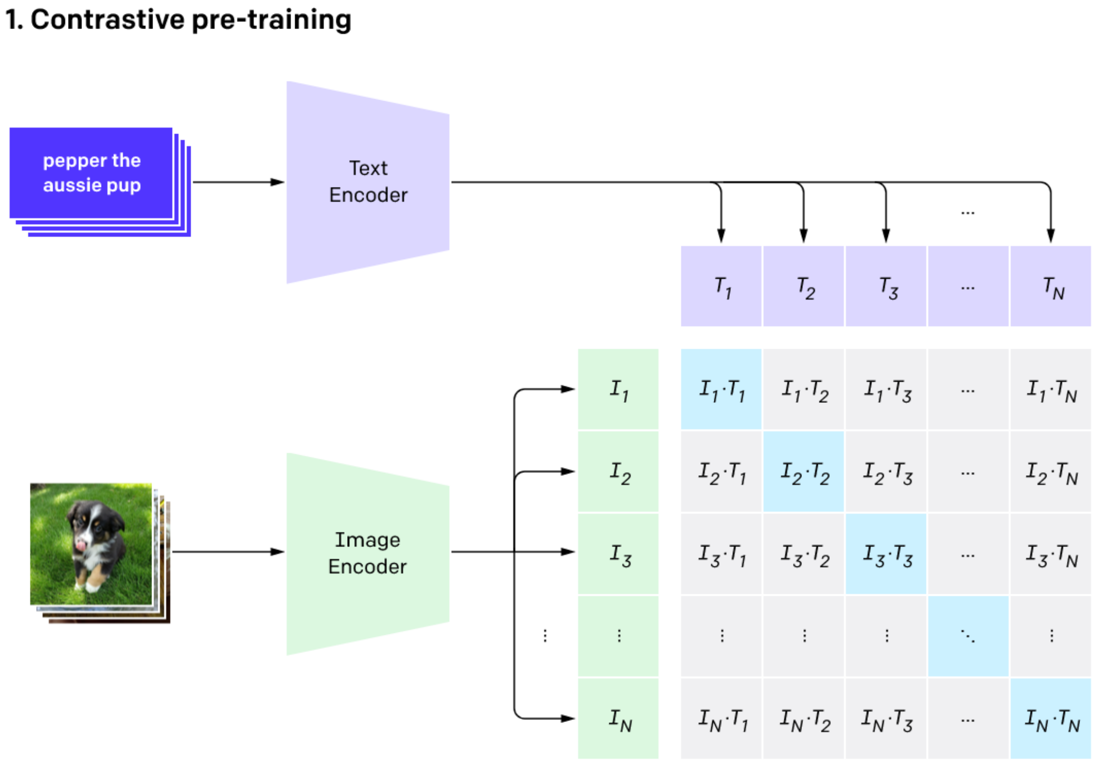


CLIP learns image and text combinations, to later predict captions for images. Source: OpenAI

Why multiple GPU?

- Example: *OpenAI's CLIP*
- If we were to train this setup on a single 'household' GPU
- Training time: ~ 30 years..

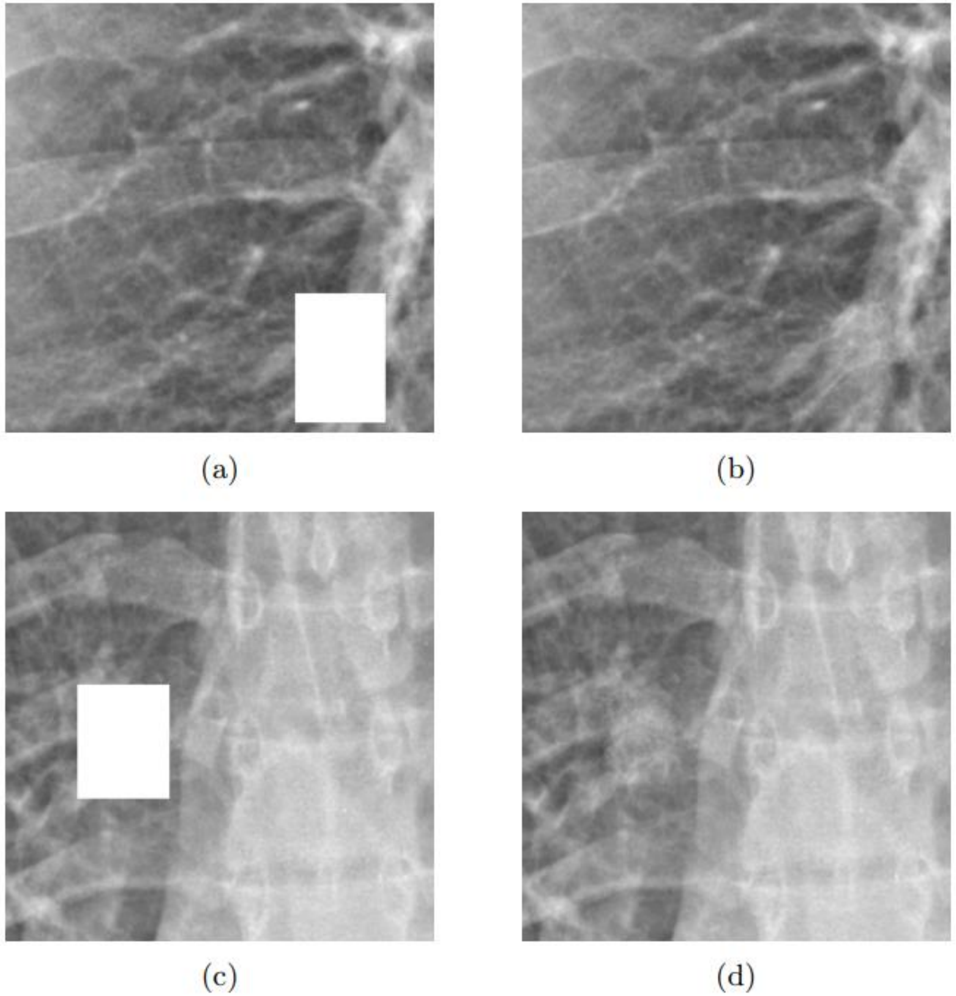
- Instead, it was trained on 592 GPUs in 18 days



CLIP learns image and text combinations, to later predict captions for images. Source: OpenAI

Why multiple GPU?

- Example: *large generative models*
- In this example we generated *lung nodules* on healthy lung images
- These generative models are large and need loads of data



Generative models can be used to expand datasets by generating 'fake' examples of lung nodules.
Source: NKI

Why multiple GPU?

- Example: *large language models*
- Models can be too large to fit in the GPU memory

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Multi-GPU Backends

How do we control and coordinate the multiple GPUs

- Levels of abstraction Lightning – Pytorch – Backend – C++/CUDA – …
- We will consider lightning, torch and the communication backends (NCCL/GLOO)

Multi-GPU Backends

- Pytorch distributed support three built-in backends

- For multi-GPU: *NCCL* is generally the fastest and most versatile

- When appropriately configured there is near-linear scalability

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✗	✓	?	✗	✓
recv	✓	✗	✓	?	✗	✓
broadcast	✓	✓	✓	?	✗	✓
all_reduce	✓	✓	✓	?	✗	✓
reduce	✓	✗	✓	?	✗	✓
all_gather	✓	✗	✓	?	✗	✓
gather	✓	✗	✓	?	✗	✓
scatter	✓	✗	✓	?	✗	✗
reduce_scatter	✗	✗	✗	✗	✗	✓
all_to_all	✗	✗	✓	?	✗	✓
barrier	✓	✗	✓	?	✗	✓

Multi-GPU Operations

Basic Dictionary

- Group, world and ranks

Point-to-Point Communication

- Send / Recv

Collective Communication

- Scatter
- Broadcast
- Reduce
- All-Reduce

Multi-GPU Operations

Basic Dictionary

- Node: a system in the compute cluster, e.g., a server with multiple GPUs
- Global/Node Rank: unique identifier for each node
- Local Rank: unique identifier for each process (usually each GPU corresponds to one process) on a single node
- World: a group containing all the processes, which can communicate with each other

- We will see these terms in practice in the tutorial

Multi-GPU Operations – Point-to-Point

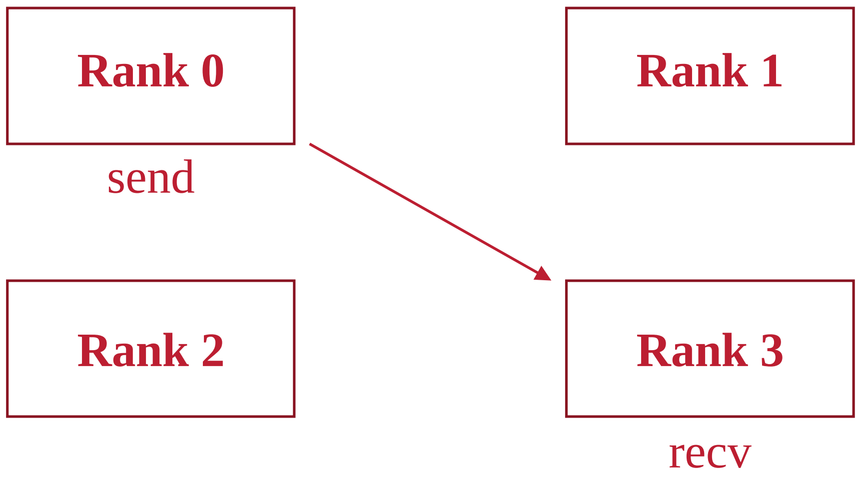
- Any Point-to-Point communication is achieved using Send and Recv
- Can be used for any communication pattern between ranks

Multi-GPU Operations – Point-to-Point

Send / Recv

- Send tensor to another rank
- Receive tensor from another rank

- *Example usage*: when we want very fine-grained control



Multi-GPU Operations – Point-to-Point

Send / Recv

- Example research usages

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

Deep Speech: Scaling up end-to-end speech recognition

Awni Hannun*, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. Ng

Baidu Research – Silicon Valley AI Lab

Multi-GPU Operations – Collective

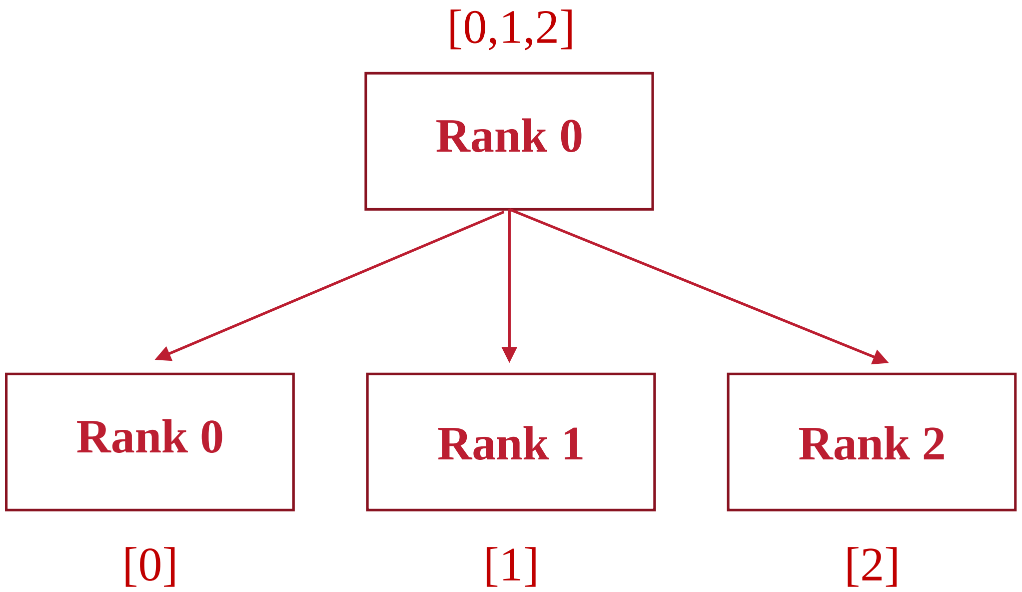
- Must be called for each rank

- If this does not happen, we can enter a so-called *deadlock*, in which we are forever waiting for one (or more) of the ranks

Multi-GPU Operations – Collective

Scatter

- Distributes from one rank to all others
- *Example usage*: divide batches among ranks

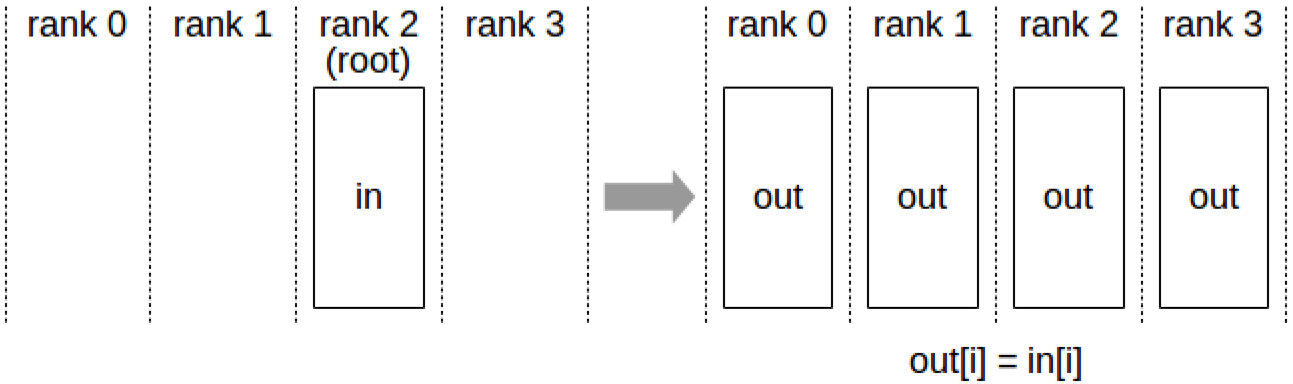


- Sidenote: scatter is not done using NCCL

Multi-GPU Operations – Collective

Broadcast

- Copy data to all other ranks
- *Example usage*: copy model to all ranks

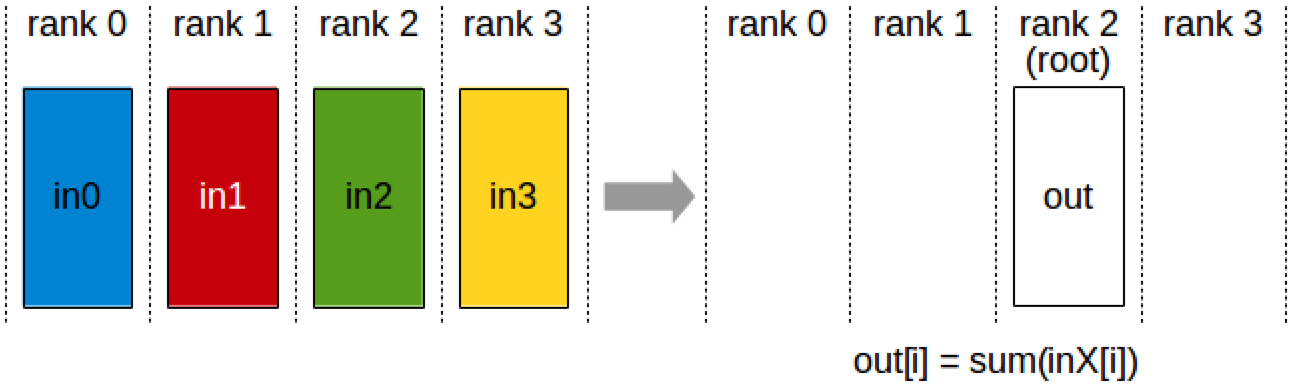


Broadcast copies data from a 'root' rank to all others
Source: NVIDIA

Multi-GPU Operations – Collective

Reduce

- Perform reductions (e.g., sum, average, max,..) across devices and write to one 'root' device
- *Example usage*: average or sum a metric from all ranks
- The **Gather** operation is a Reduce with a *concatenate* operation

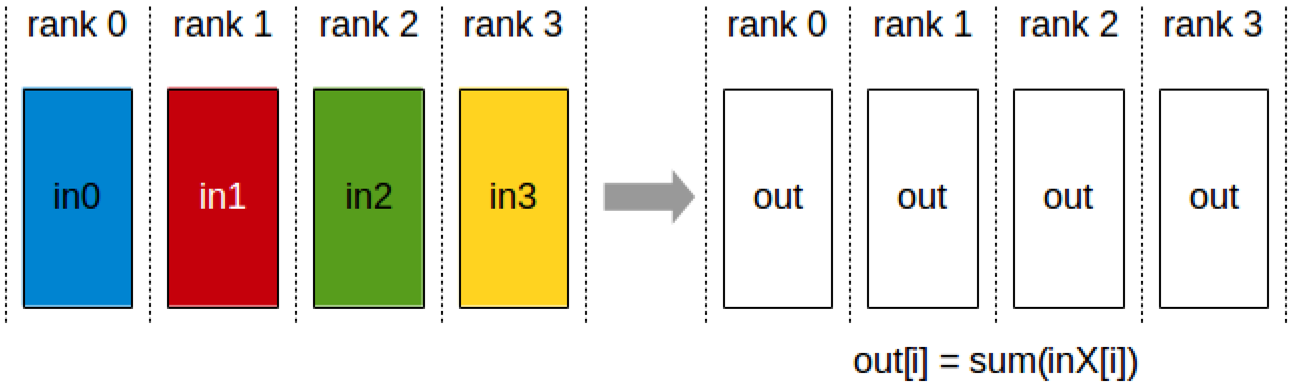


Example of a reduce operation with 'sum'
Source: NVIDIA

Multi-GPU Operations – Collective

All Reduce

- A *reduce* operation followed by a *broadcast*
- *Example usage*: averaging the gradients in a backpropagation (see distributed data parallel)



Example of an all-reduce operation with 'sum'
Source: NVIDIA

Multi-GPU Operations – Collective

Other operations

- In the tutorial we will discuss **AllGather** and **ReduceScatter**

Practice – PyTorch Frameworks

Overview

- Data Parallel
- Distributed Data Parallel
- Can we do more?

Practice – PyTorch Data Parallel

Overview

- 1: Split mini-batch and *scatter* over ranks
- 2: *Broadcast* model over all ranks
- 3: Forward the mini-batches through the model
- 4: *Reduce* the gradients to rank 0
- 5: Perform the backpropagation and obtain updated model
- Repeat

- Sidenote: data parallel allows for only one node – no multinode training

Practice – PyTorch Data Parallel

The GIFs shown in the lectures can be found at:

https://towardsdatascience.com/sharded-a-new-technique-to-double-the-size-of-pytorch-models-3af057466dba

DataParallel has a lot of communication overhead
Source: W. Falcon – PyTorch Lightning

Practice – PyTorch Distributed Data Parallel

Overview

- Only once: *broadcast* the model and initialize identically on each rank
    - 1: Gather data indices from a *distributed* sampler.
      Example: 2 GPUs and dataset [0,1,2,3], could yield [0,1] for GPU 0 and [2,3] for GPU 1

Practice – PyTorch Distributed Data Parallel

Overview

- Only once: *broadcast* the model and initialize identically on each rank
  - 1: Gather data indices from a *distributed* sampler
  - 2: Forward through model
  - 3: *All Reduce* the gradients whilst every rank is performing a backpropagation
  - 4: All ranks perform an optimization step with the synchronized gradients
  - Repeat

- Sidenote: multinode training is supported

Practice – PyTorch Distributed Data Parallel

The GIFs shown in the lectures can be found at:

https://towardsdatascience.com/sharded-a-new-technique-to-double-the-size-of-pytorch-models-3af057466dba

DDP avoids the large amounts of communication overhead by synchronizing gradients
Source: W. Falcon – PyTorch Lightning

Practice – PyTorch Distributed Data Parallel

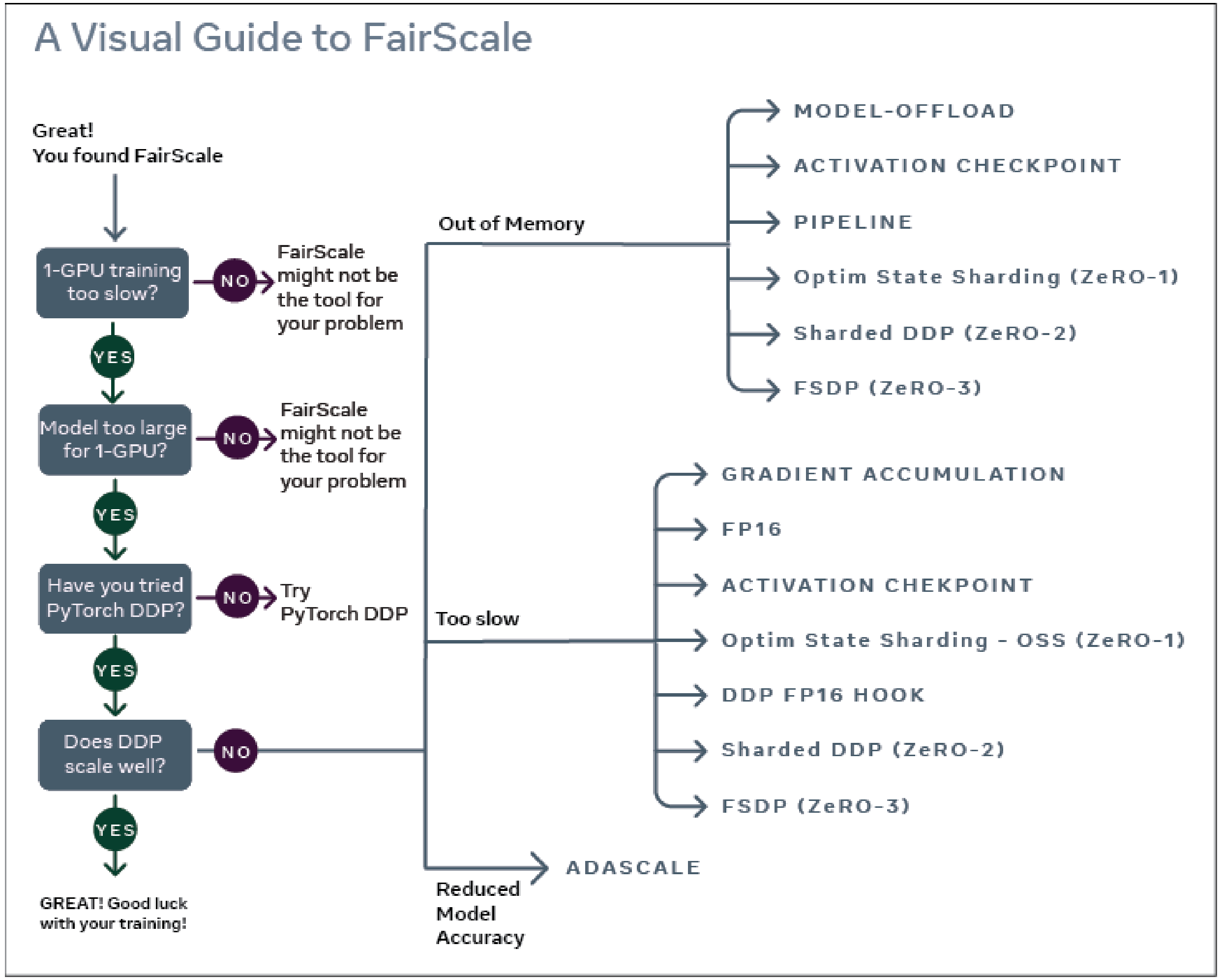
Overview

- Due to the lack of overhead, DDP is generally much faster than DP
- With DDP, each GPU gets its own process
- DDP is scalable across multiple nodes

Practice – PyTorch

What more can we do?

- FairScale
- Models might be too large
- DDP might not scale as expected



Practice ++ PyTorch Lightning

Overview

- Multi-GPU in Lightning
- Tracking performance

Practice ++ PyTorch Lightning

Multi-GPU in Lightning

- Lightning makes everything very easy
- Strategy argument determines the distributed backend (DP, DDP, …)

```
from pytorch_lightning import Trainer

# Two GPUs with the Data Parallel strategy
Trainer(gpus=2, strategy='dp')

# Two nodes, each with four GPUs using Distributed Data Parallel
Trainer(num_nodes=2, gpus=4, strategy='ddp')
```

Practice ++ PyTorch Lightning

Multi-GPU in Lightning

- There are many finetuning options that we have not discussed
- If you are interested, research them yourself, the documentation of PyTorch and Lightning should be sufficient

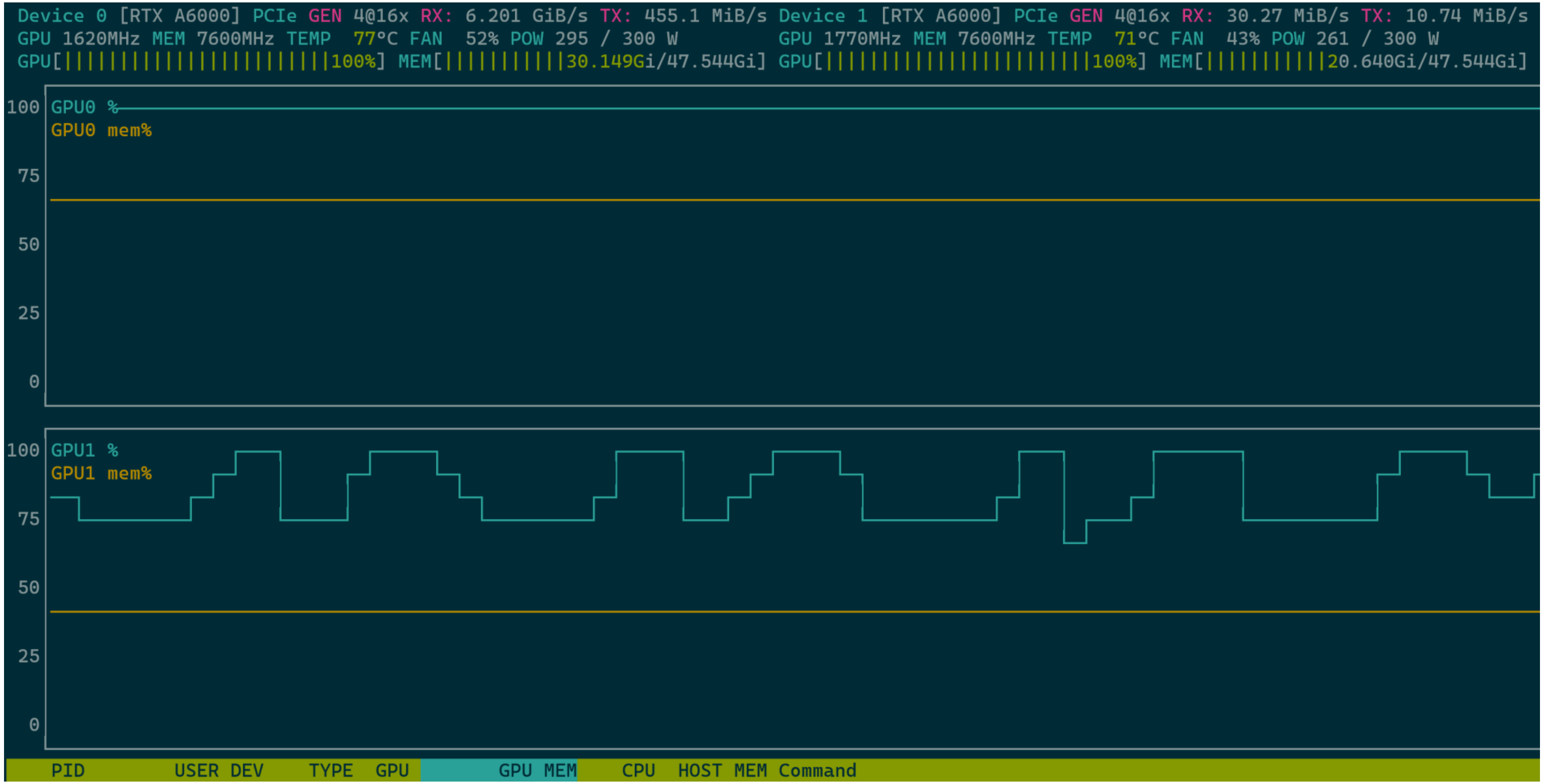
```
from pytorch_lightning import Trainer
# Some examples
# Two GPUs with the 'spawn-based' DDP
Trainer(gpus=2, strategy='ddp_spawn')

# Do not look for unused parameters every iteration
Trainer(num_nodes=2, gpus=4, strategy='ddp_find_unused_parameters_false')
```

Practice ++ PyTorch Lightning

Tracking Performance

- GPU performance with *nvtop*

Practice ++ PyTorch Lightning



Practice ++ PyTorch Lightning

Tracking Performance

- Not every cluster has *nvtop*, in such cases one can watch *nvidia-smi* as well
- It shows the GPU memory and utilization without any graphs

Practice ++ PyTorch Lightning

Tracking Performance

- Profilers
- A couple are built-in into Lightning
- The most comprehensive one is the PyTorch profiler

```
from pytorch_lightning import Trainer

# Profiling with the pytorch profiler
Trainer(profiler='pytorch')
```

Practice ++ PyTorch Lightning

Tracking Performance

- Detailed information on the operations performed
- Check whether the CPU is performing its tasks ahead of the GPU
- Why is the GPU not utilized?