DeepSpeed & ZeRO

Treino distribuído eficiente de redes neurais com bilhões de parâmetros em GPUs

Autor

WILL RAZEN

Data

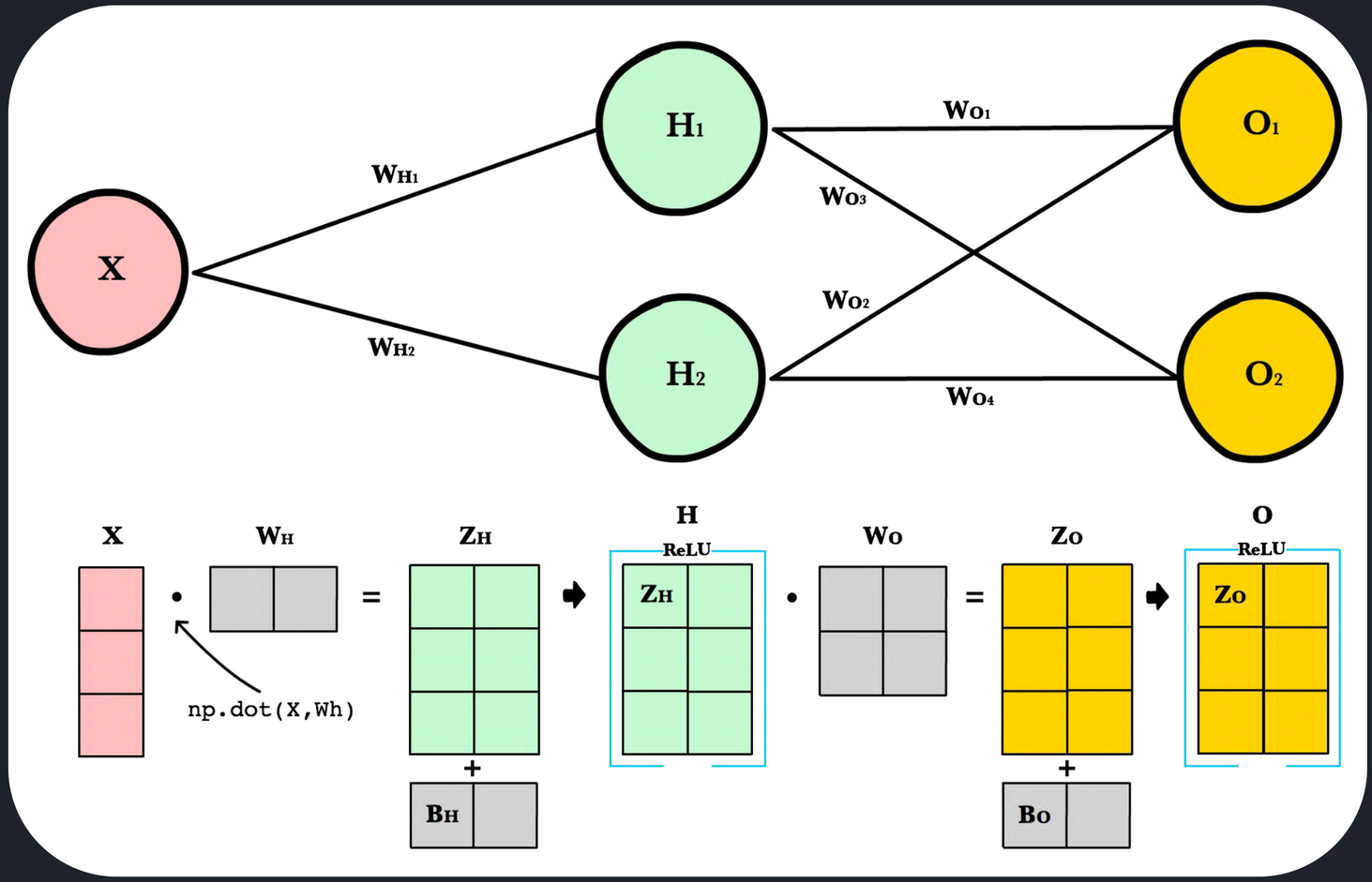
04 JUL 2022

Agenda

1. Redes Neurais
2. Backpropagation
3. PyTorch
4. Treino em GPUs
5. NVIDIA H100
6. ZeRO
7. ZeRO-Offload
8. DeepSpeed

Redes Neurais

- Compostas por **múltiplas camadas**: input, intermediárias ("hidden"), e output
- Cada camada contém neurônios, que possuem **parâmetros livres** e **funções de ativação** não lineares
- **Interligações** podem ser customizadas
- Representação em **matrizes / tensores**
- Objetivo: otimizar parâmetros para **minimizar perda ("loss")** dado um gabarito (pares input x output)



Fonte: https://ml-cheatsheet.readthedocs.io/en/latest/forwardpropagation.html

Backpropagation

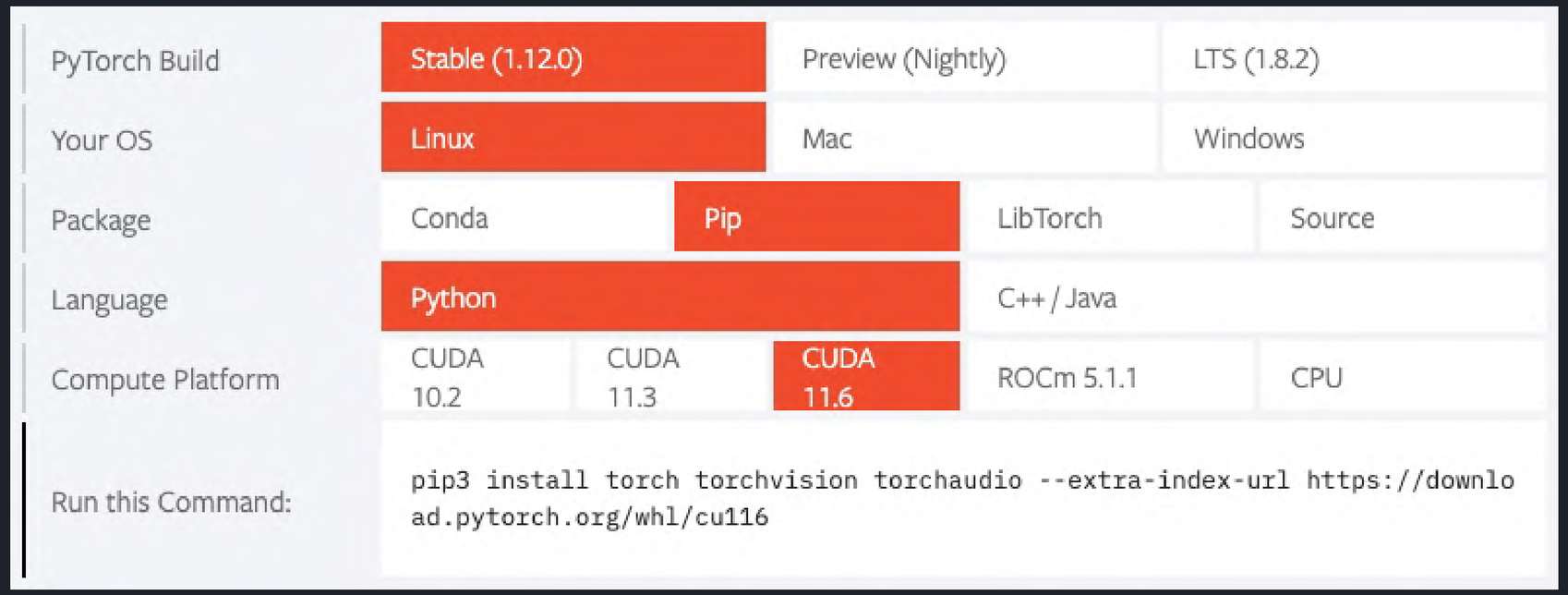
- Algoritmo para **otimização** dos parâmetros de redes neurais ("**treino**")
- Em cada iteração, começa com um lote de dados ("**batch**") e uma **forward pass**
- Utiliza regra da cadeia para calcular **gradiente** da loss function em respeito a cada parâmetro, começando pela última camada (**backward pass**)
- Atualiza cada parâmetro de acordo com uma regra ("**update rule**"), que usa o gradiente
- Possui diversas variações, como **SGD** (stochastic gradient descent), **Adam** e **LAMB**

Fonte: https://www.researchgate.net/figure/Training-a-neural-network-using-a-backpropagation-algorithm-15_fig3_336523522

```
Assign all network inputs and output
Initialize all weights with small random numbers, typically between -1 and 1
repeat
   for every pattern in the training set
      Present the pattern to the network
//       Propagated the input forward through the network:
    for each layer in the network
      for every node in the layer
         1. Calculate the weight sum of the inputs to the node
         2. Add the threshold to the sum
         3. Calculate the activation for the node
      end
    end
//       Propagate the errors backward through the network
     for every node in the output layer
      calculate the error signal
    end
        for all hidden layers
      for every node in the layer
         1. Calculate the node's signal error
         2. Update each node's weight in the network
      end
    end
//       Calculate Global Error
    Calculate the Error Function
   end
while ((maximum  number of iterations < than specified) AND
  (Error Function is > than specified))
```

PyTorch

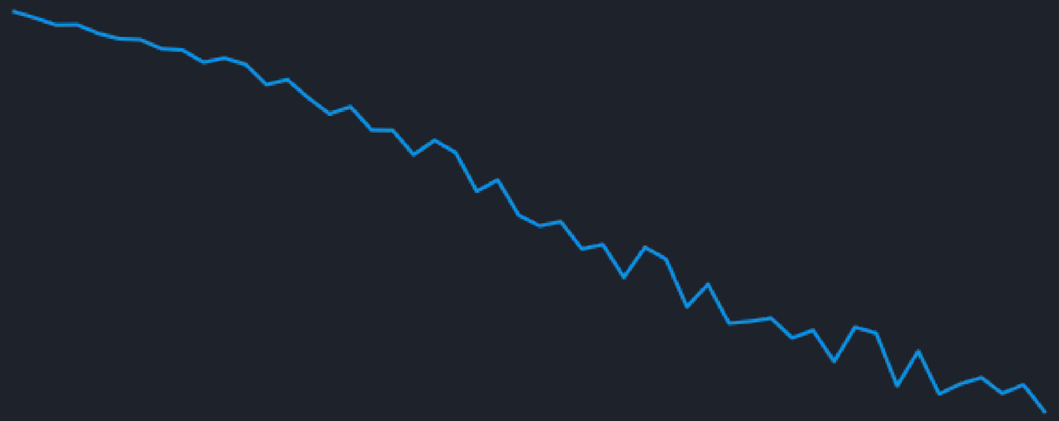
- Framework open source para desenvolvimento de redes neurais, escrito em **Python, C++** e **CUDA**
- Possui muitos **componentes populares prontos**, como camadas, funções de ativação e de perda, algoritmos de otimização e utilidades variadas
- Abstrai muito do código necessário para **treino paralelo e distribuído** em CPUs e GPUs



Fonte: https://pytorch.org/

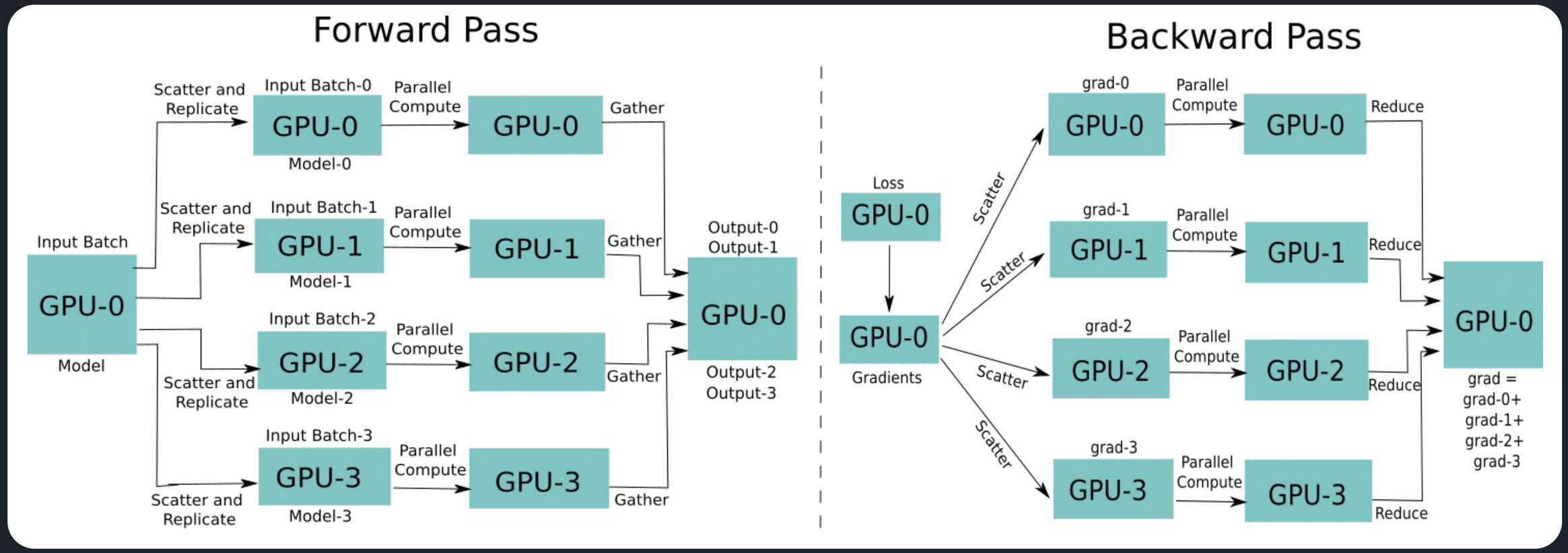
```python
import torch
from torch import nn
import torchvision
import matplotlib.pyplot as plt

# Dados
data = torchvision.datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=torchvision.transforms.ToTensor()
)
dataloader = torch.utils.data.DataLoader(data, batch_size=64)

# Arquitetura
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10))

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

device = "cuda" if torch.cuda.is_available() else "cpu"
model = NeuralNetwork().to(device)
```

```python
# Treino
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
epochs = 5
losses = []
for t in range(epochs):
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)
        pred = model(X)
        loss = loss_fn(pred, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if batch % 100 == 0:
            losses.append(loss.item())
plt.plot(losses)
```



Treino em GPUs

- Alta capacidade de **paralelizar** computação de tensores
- Limitado pela quantidade de **memória** por GPU: sizeof(parameter) * parameters * batch size
- Possibilidade de utilizar **múltiplas GPUs**: data parallel, model parallel, pipeline parallel, 3D
- Necessidade de **dimensionar rede** para treino multi-GPU / multi-node (e.g. NVLink, InfiniBand)



Fonte: https://naga-karthik.github.io/post/pytorch-ddp/

NVIDIA H100

80b transistors

4 nm

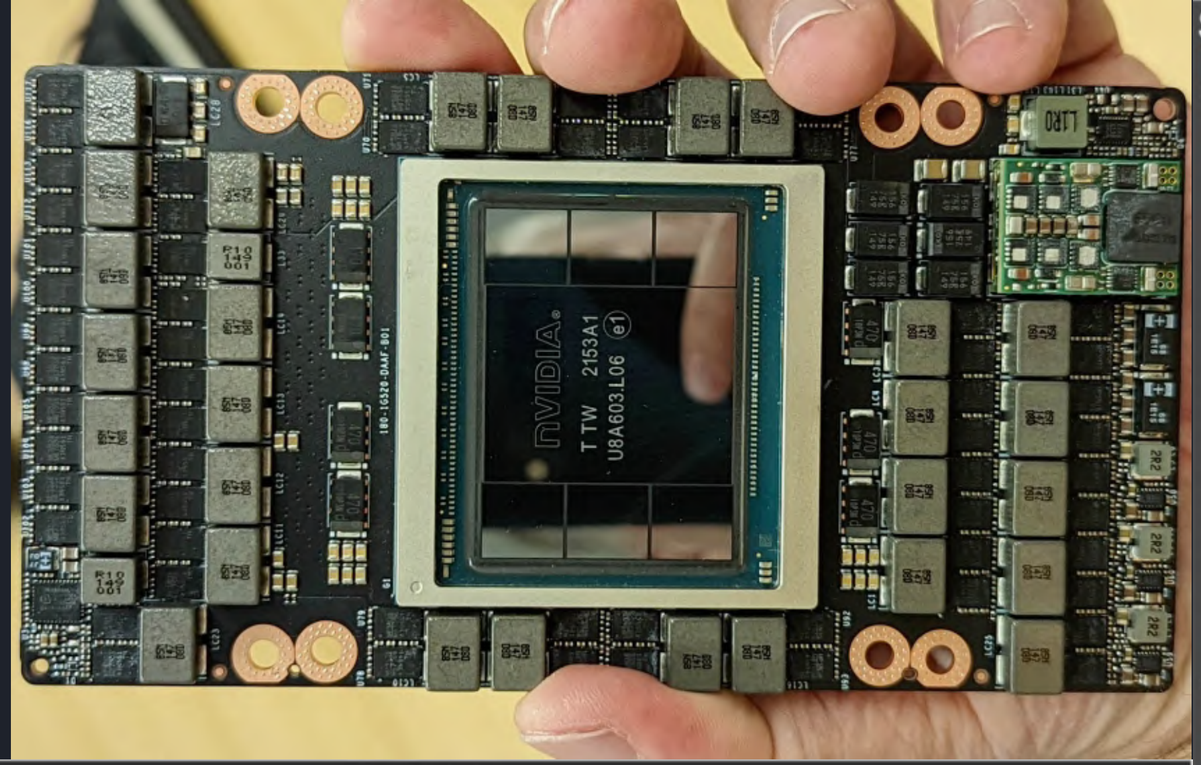
80GB mem.

3 TB/s

132 SMs

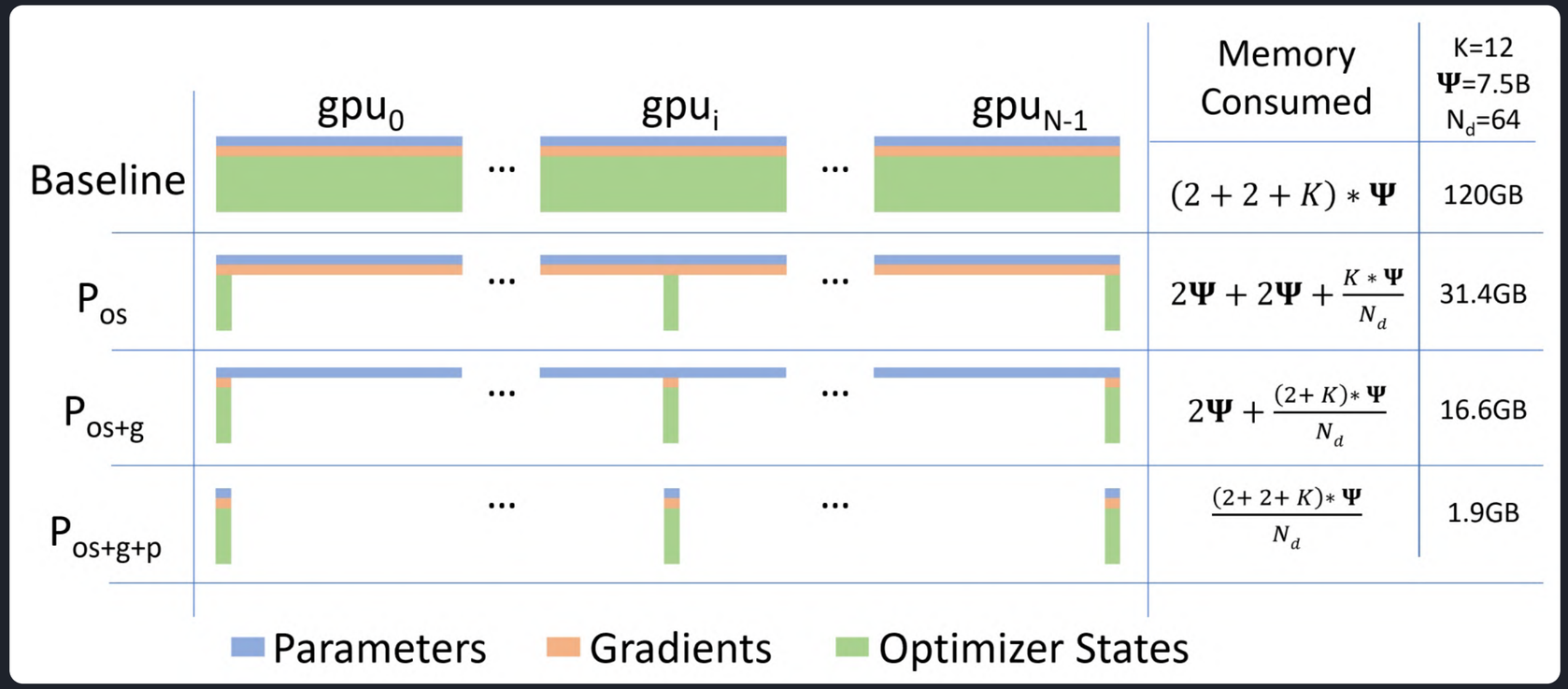
528 TCs

700W



Zero Redundancy Optimizer (ZeRO)

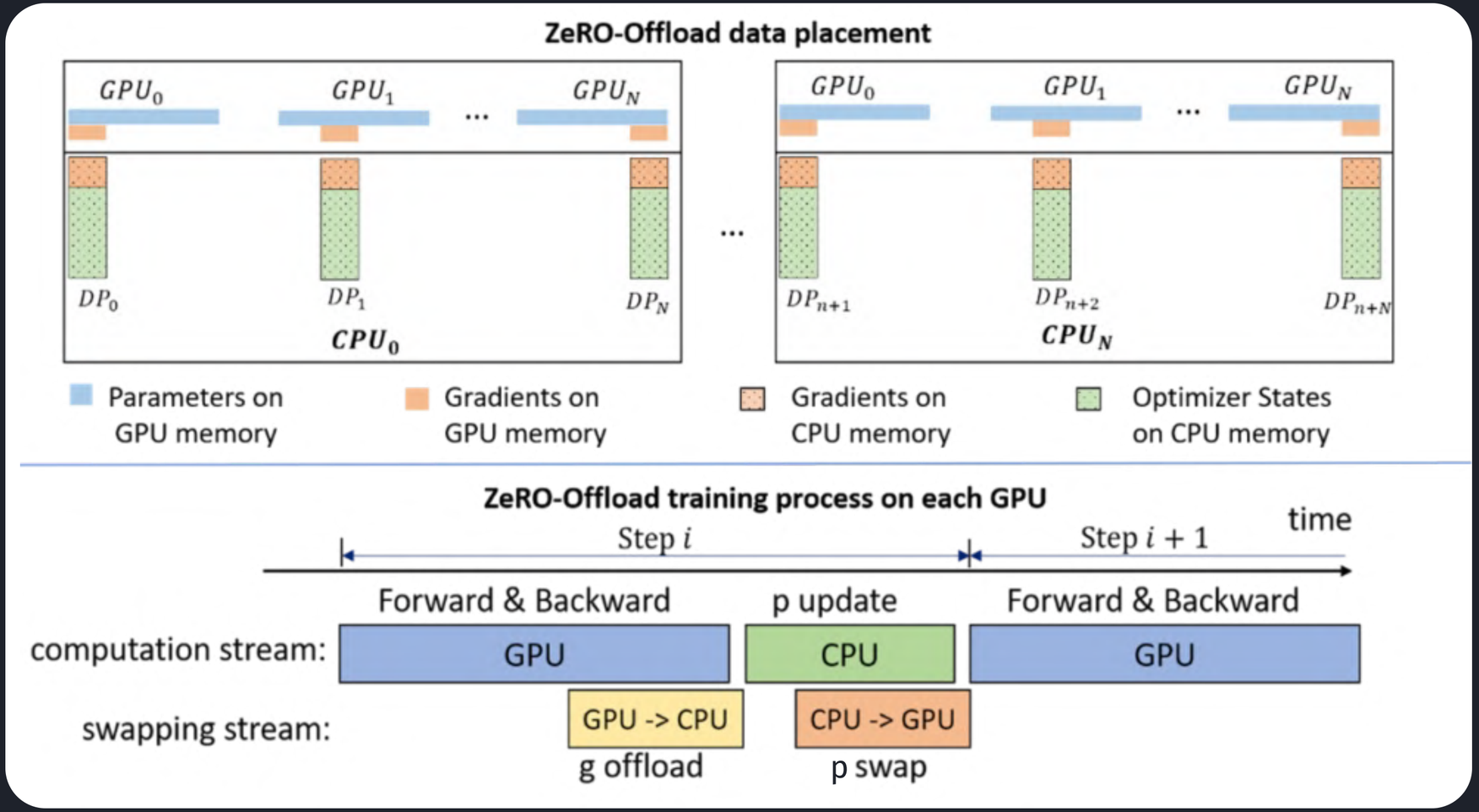
- Elimina **redundâncias de memória** presentes em data e model parallel
- Utiliza memória igual ou menor do que pipeline parallel, **sem desvantagens** funcionais
- Volume de **comunicação**
  - Pos+g: igual a DP
  - Pp: aumenta em 50%
- Utilizado para treinar o maior modelo da época (2020, **17B** parâmetros)



Fonte: https://arxiv.org/abs/1910.02054

ZeRO-Offload

- Utiliza **CPU na etapa de update**, que é menos intensiva computacionalmente
- Reduz ainda mais a necessidade de memória
- Variante ZeRO-Infinity utiliza NVMe para conseguir treinar modelo de **1 trilhão de parâmetros em apenas 16 NVIDIA V100** (1 DGX-2)



Fonte: https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/

DeepSpeed

- Biblioteca de Python open source com implementação do ZeRO e outras técnicas relacionadas
- Fácil de integrar com **PyTorch**
- Principais funcionalidades:
  - ZeRO
  - ZeRO-Offload
  - Mixed Precision Training
  - Single-GPU, Multi-GPU, Multi-Node
  - 3D parallelism
  - Otimizadores extras (e.g. 1-bit Adam)
  - Autotuning

Fonte: https://github.com/microsoft/DeepSpeed

```
{
  "train_batch_size": 8,
  "gradient_accumulation_steps": 1,
  "optimizer": {
    "type": "Adam",
    "params": {
      "lr": 0.00015
    }
  },
  "fp16": {
    "enabled": true
  },
  "zero_optimization": true
}
```

```
model_engine, optimizer, _, _ = \
    deepspeed.initialize(args=cmd_args,
                         model=model,
                         model_parameters=params)

#(...)

for batch, data in enumerate(dataloader):
    loss = model_engine(data)
    model_engine.backward(loss)
    #weight update
    model_engine.step()
```

Obrigado!