Neural Networks Speed-up and Compression

Lecture 6: Training Speed-up of Large Scale Models

Lecturer: Dr. Julia Gusak, Senior Research Scientist, Skoltech
TAs: Stanislav Abukhovich, Dmitry Ermilov, Konstantin Sobolev; PhD students, Skoltech

Outline

- Motivation
- Levels of Optimization
- Servers and Graphics Accelerators
- Performance Estimation
- Effective attention mechanisms
- Types of parallelism
- Checkpointing and Offloading
- Optimization methods for large-scale models
- Frameworks

Motivation

Lecture 6: Training Speed-up of Large Scale Models

NN Training

- ```
  model = Net()

  criterion = nn.CrossEntropyLoss()
  optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

  def train(epoch):
      model.train()
      for batch_idx, (data, target) in enumerate(train_loader):
          data, target = Variable(data), Variable(target)
          optimizer.zero_grad()
          output = model(data)
          loss = criterion(output, target)
          loss.backward()
          optimizer.step()
  ```

- **For many domains**, it has been found that larger models and larger datasets give better performance
- Examples: natural language processing (NLP), self-supervised learning, vision transformers, contrastive language image pre-training, etc.

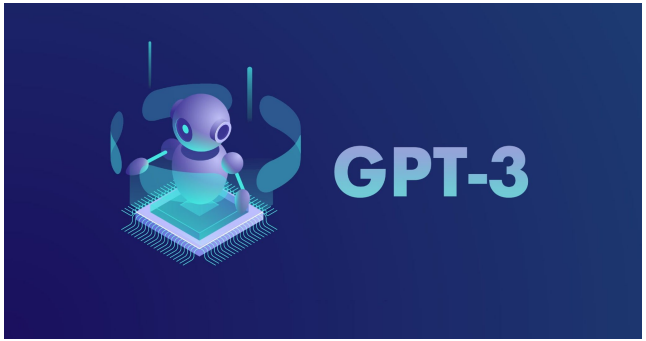
The time when you can train a large model on 1 GPU or on several GPUs is quickly going away!
  ```

NN Training: expensive activity

Number of parameters	Training time	Number of GPUs	Costs (1 GPU min costs 5 RU)
175 B	34 days	1024	~ 251 million RU
1000 B	84 days	3072	~ 1.9 billion RU

NN Training: emissions into the atmosphere

Energy: 190 MW*hours → Air emissions : 85 tonnes of CO_2



GPT-3 training

Heating for 126 houses in Denmark

Car drive to the Moon

Levels of Optimization

Lecture 6: Training Speed-up of Large Scale Models

Levels of Optimization

Type of Instructions	Approach	Result
High-level	New training algorithms	Training time reduction due to reduction of FLOP, acceleration of computational operations, reduction of allocated memory
Low-level	New approaches to work with memory having one/several GPUs	Training time reduction due to the usage of bigger batch size during one iteration
Communications	Organize optimal communications among GPUs from the cluster	Training time reduction due to faster communications

Servers & Graphics Accelerators

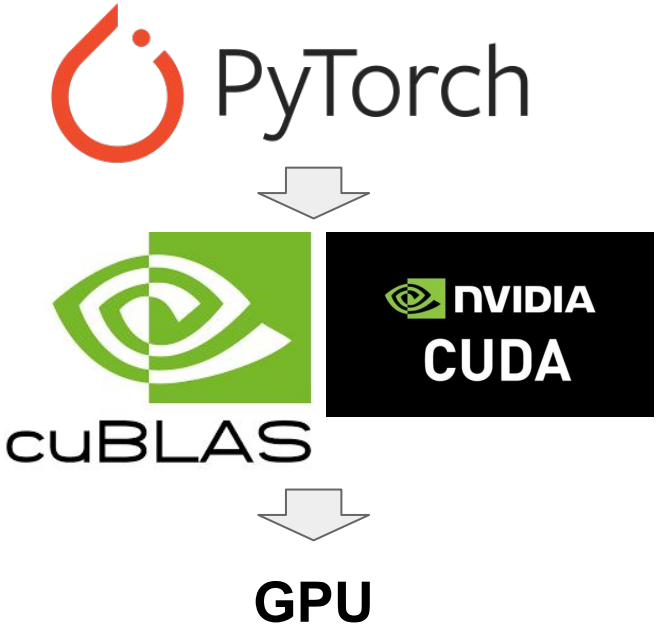
Lecture 6: Training Speed-up of Large Scale Models

Servers and Graphics Accelerators

Pytorch - Python library (computational operations on C++) for neural networks training; responsible for building a computational graph and memory allocation.

CUDA - a library responsible for directly performing calculations and memory operations on the GPU.

CUBLAS - a package from CUDA, used in GPT2 to perform matrix multiplication operations (called from under Pytorch).



Zhores Cluster

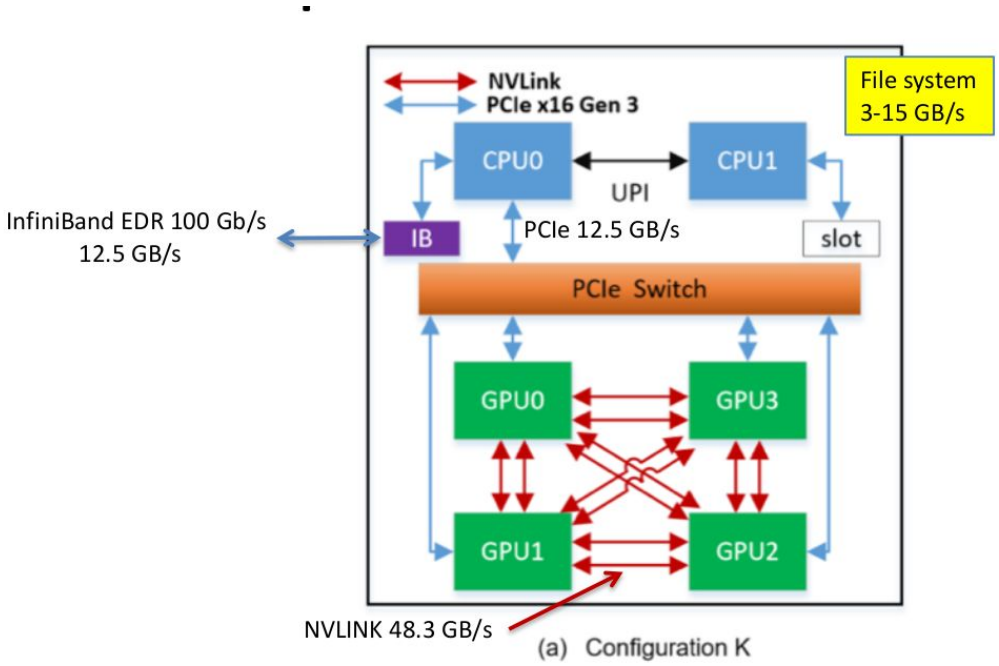
25 nodes, each has 4 GPU and 2 CPU

CPU: Intel Xeon Gold 6140 @ 2.3 GHz

GPU: Nvidia Tesla V100, 16 GB

Busses

- NVLink: GPU-GPU data transfer,
- PCIe: CPU-GPU data transfer,
- InfiniBand: CPU-CPU data transfer.



Configuration of Zhores node, equipped with GPU (V100)

Performance Estimation

Lecture 6: Training Speed-up of Large Scale Models

Model Profiling

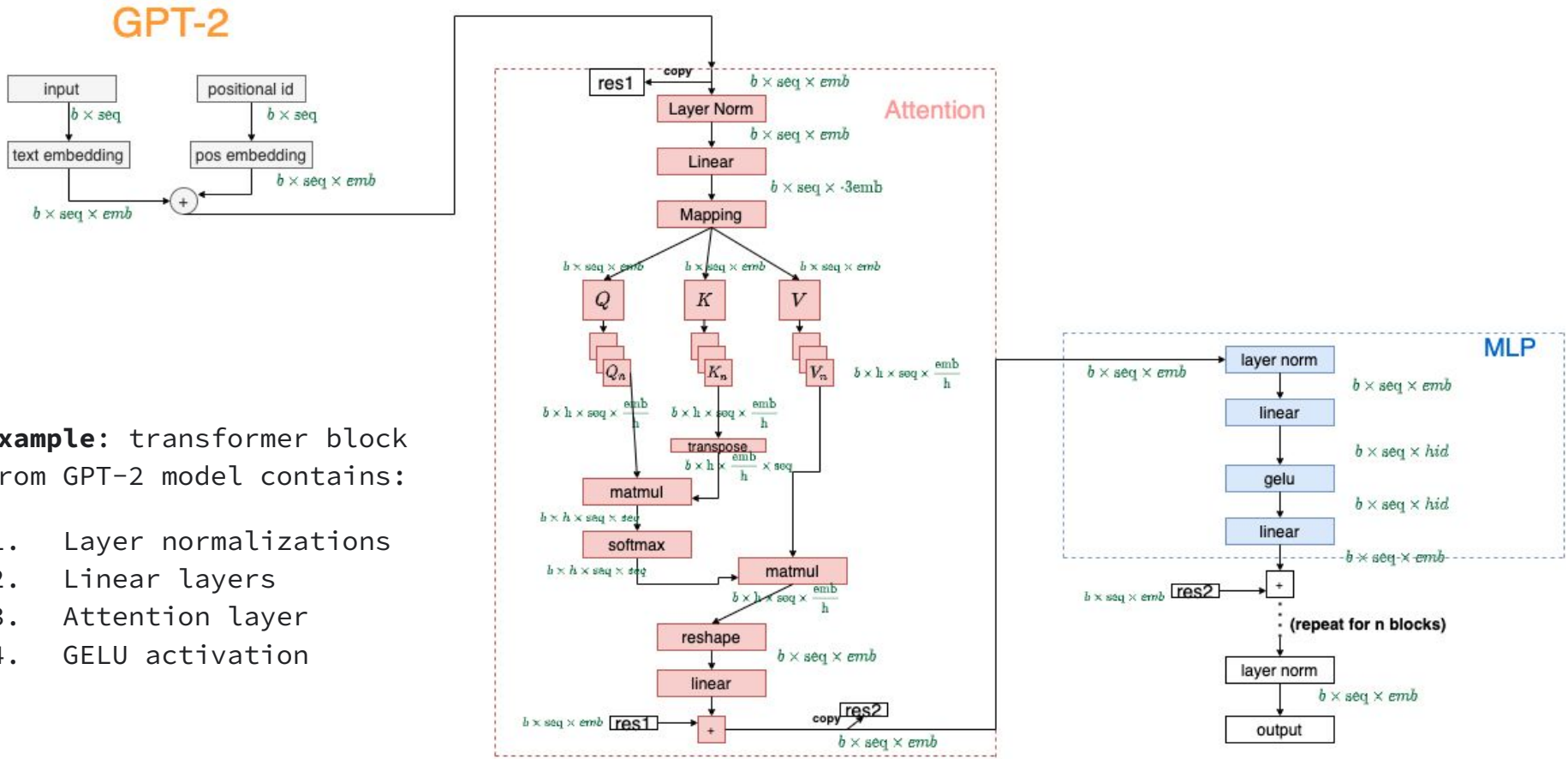
Goal: find the most costly operations in terms of time and memory, find out their dependence on the model parameters.

Tools:

1. NVidia Tools Extensions (NVTX)
2. NVidia Nsight profiler
3. PyTorch profiler
4. Torchviz

Sources:
- https://docs.nvidia.com/gameworks/content/gameworkslibrary/nvtx/nvidia_tools_extension_library_nvtx.htm
- https://developer.nvidia.com/nsight-systems
- https://github.com/pytorch/kineto
- https://github.com/szagoruyko/pytorchviz

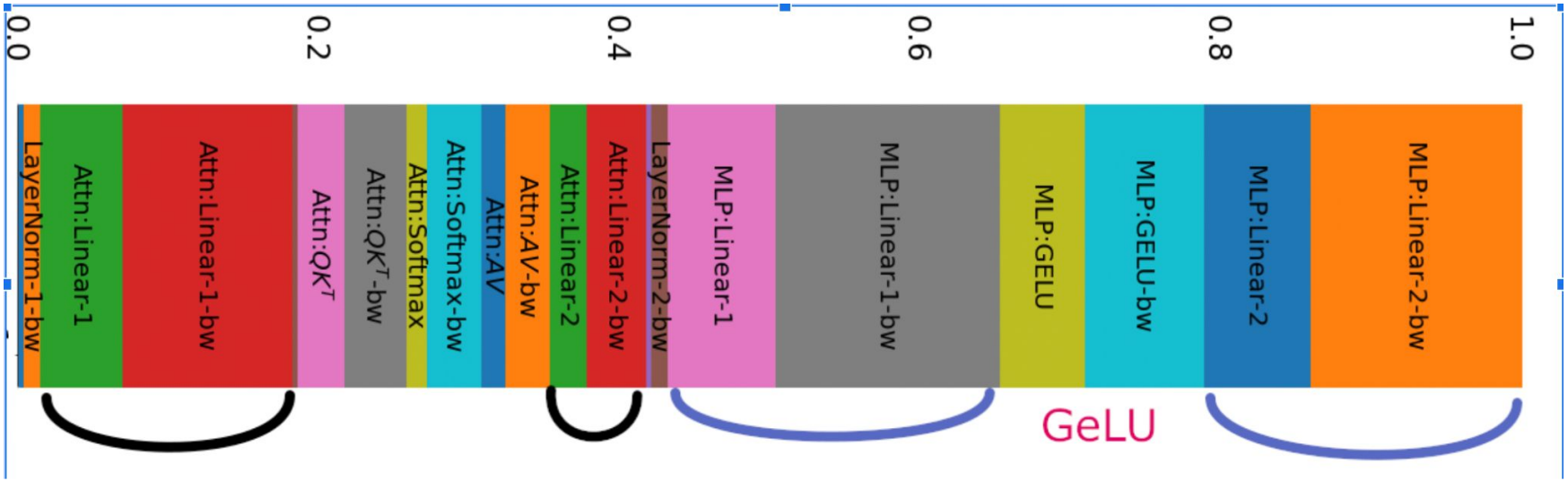
Computational Graph for GPT-2



Example: transformer block from GPT-2 model contains:

1. Layer normalizations
2. Linear layers
3. Attention layer
4. GELU activation

GPT-2 Profiling: Time



Relative GPU time of specified operations during forward and backward passes.
Sequence length s = 512, batch b = 8, hidden layer dimension h = 768.

Profiling Time of Elementary Operations

Distribution of GPU time by elementary operations

GeLU in PyTorch saves 4(!) intermediate activations

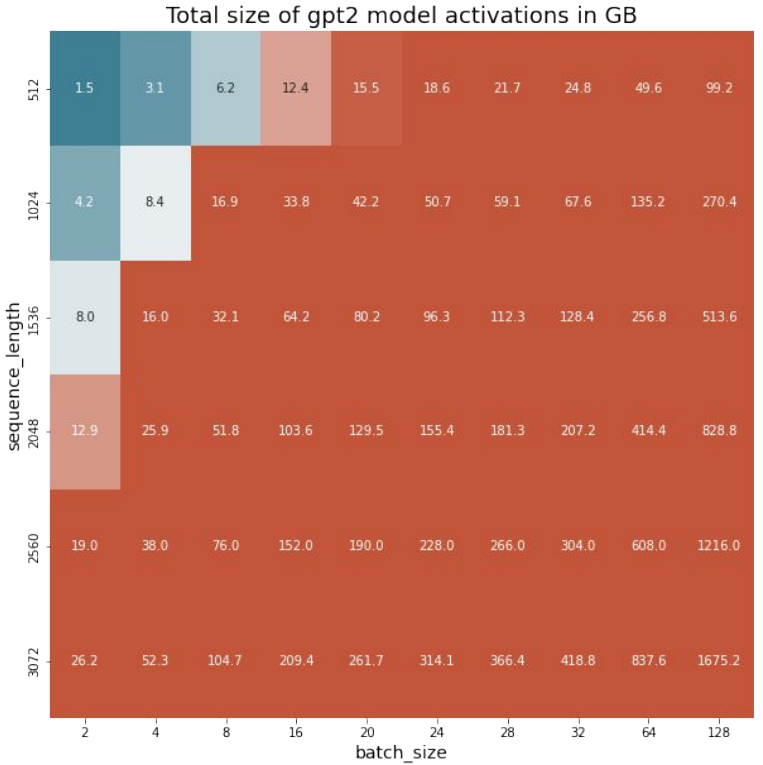
GeLU speed is limited by memory access speed (similar to other elementwise operations)

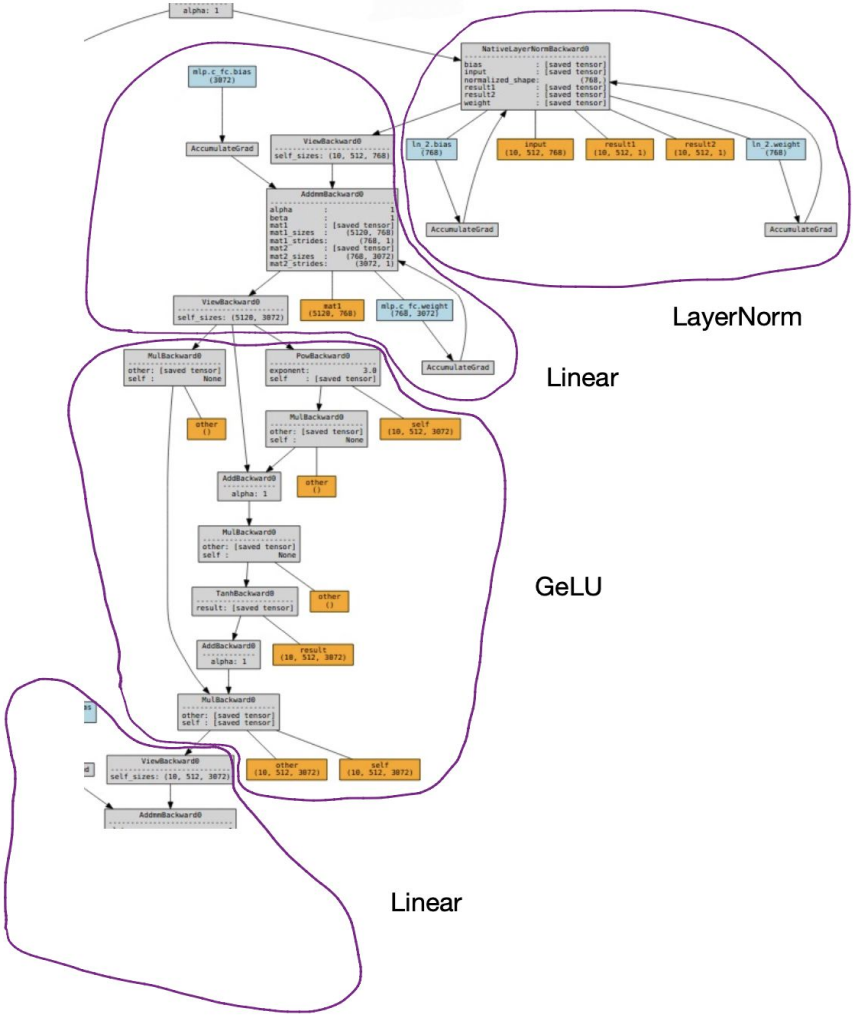
Many activation functions can be speeded up by recalculating activations (checkpointing) or by approximating the gradients

GPT-2 Profiling: Memory

Memory in GB required to store all activations, depending on the batch size and the length of the token sequence.

The memory limit of one GPU V100-16GB is highlighted in red.




alpha: 1
NativeLayerNormBackward0
bias : [saved tensor]
input : [saved tensor]
normalized_shape: (768,)
result1 : [saved tensor]
result2 : [saved tensor]
weight : [saved tensor]
mlp.c_fc.bias (3072)
ln_2.bias (768)
input (10, 512, 768)
result1 (10, 512, 1)
result2 (10, 512, 1)
ln_2.weight (768)
AccumulateGrad
ViewBackward0
self_sizes: (10, 512, 768)
AccumulateGrad
AccumulateGrad
AddmmBackward0
alpha : 1
beta : 1
mat1 : [saved tensor]
mat1_sizes : (5120, 768)
mat1_strides: (768, 1)
mat2 : [saved tensor]
mat2_sizes : (768, 3072)
mat2_strides: (3072, 1)
ViewBackward0
self_sizes: (5120, 3072)
mat1 (5120, 768)
mlp.c_fc.weight (768, 3072)
AccumulateGrad
LayerNorm
Linear
MulBackward0
other: [saved tensor]
self : None
PowBackward0
exponent: 3.0
self : [saved tensor]
other ()
MulBackward0
other: [saved tensor]
self : None
self (10, 512, 3072)
AddBackward0
alpha: 1
other ()
MulBackward0
other: [saved tensor]
self : None
TanhBackward0
result: [saved tensor]
other ()
AddBackward0
alpha: 1
result (10, 512, 3072)
MulBackward0
other: [saved tensor]
self : [saved tensor]
GeLU
ViewBackward0
self_sizes: (10, 512, 3072)
other (10, 512, 3072)
self (10, 512, 3072)
AddmmBackward0
Linear

Effective Attention Mechanisms

Lecture 6: Training Speed-up of Large Scale Models

Attention Layer Speed-up

Att(Q, K, V) = Softmax($Q K^T$) V, complexity $O(n^2)$, where n - sequence length

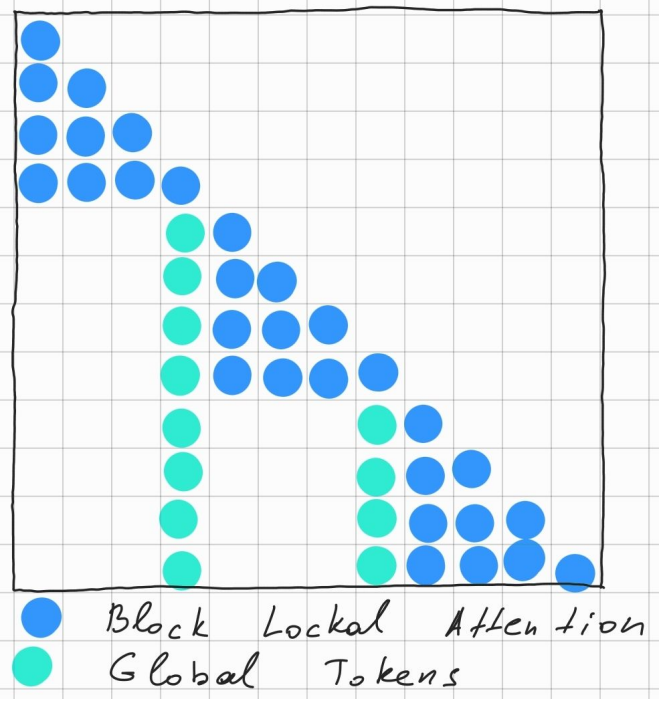
Goal: Reduce the training time and memory consumption of the Attention Layer

Many works based on different approaches:

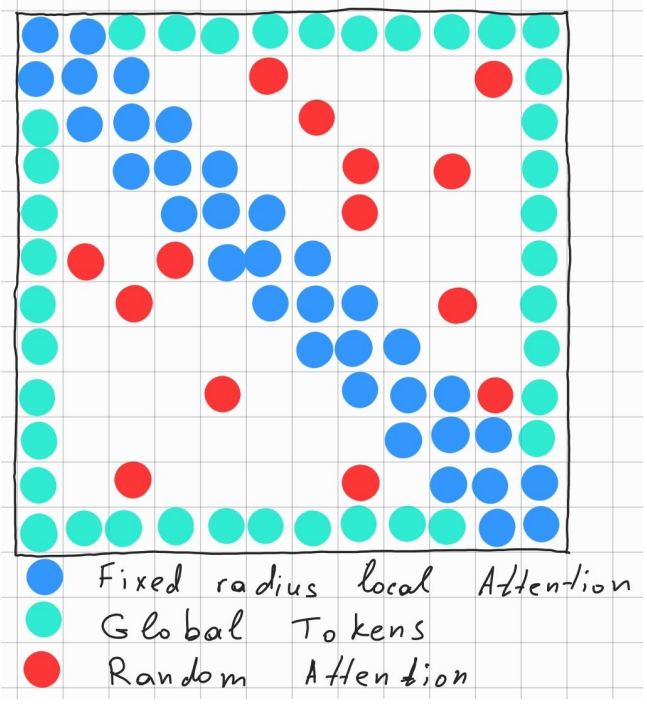
1. Sparsification of the Attention Matrix
    a. Fixed patterns
    b. Learnable Patterns
2. Low Rank Methods
3. Memory Based Methods
4. Nuclear methods
5. And many others…

Popular Sparsification Methods: $O(n^{1.5})$

Sparse Aattention
(Used in GPT3)



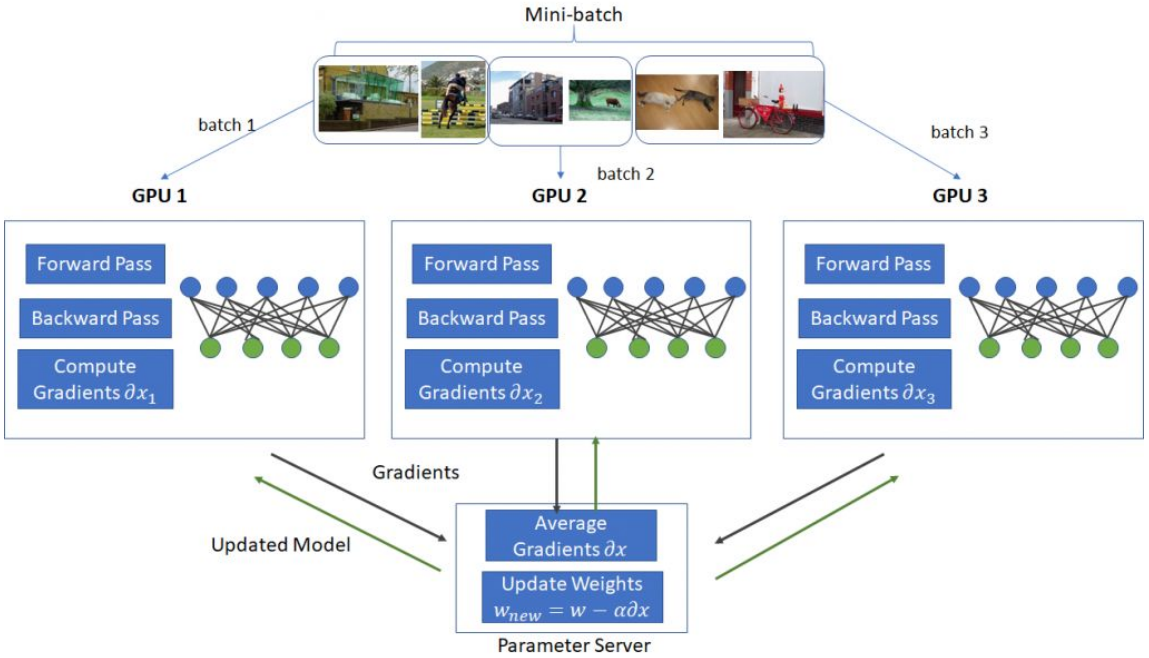
BigBird



Types of Parallelism

Lecture 6: Training Speed-up of Large Scale Models

Data Parallelism



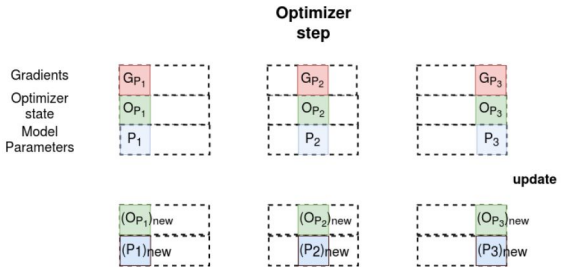
Data parallelism:

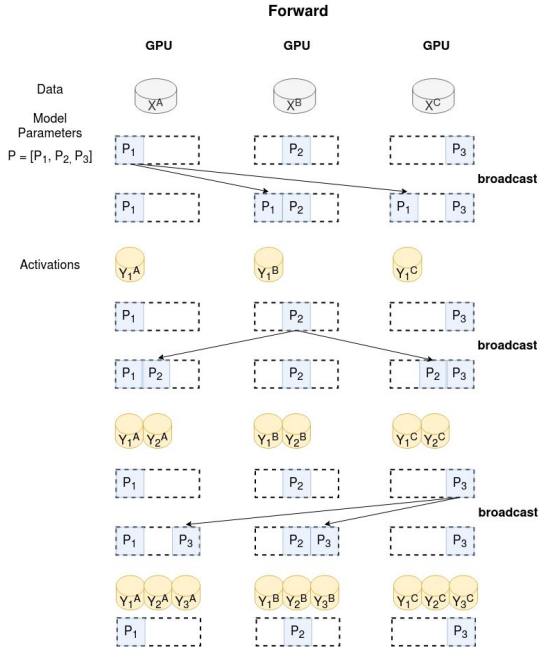
+ speeds up training
- weights and gradients must fit on the same device

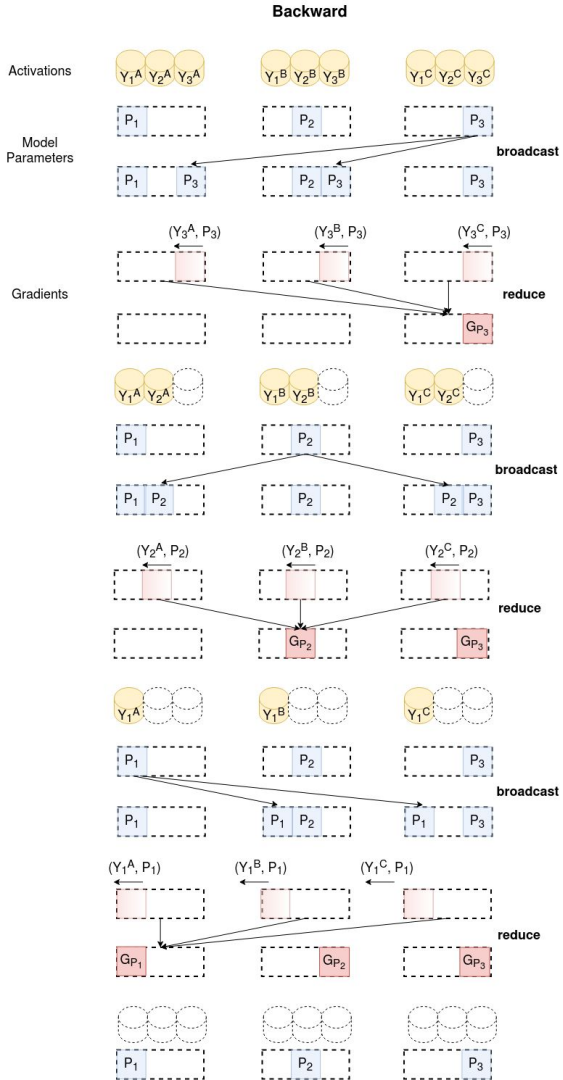
Data Parallelism using ZeRO

Data parallelism using ZeRO:

+ you can train models that do not fit on one device
- increases the number of transfers between devices



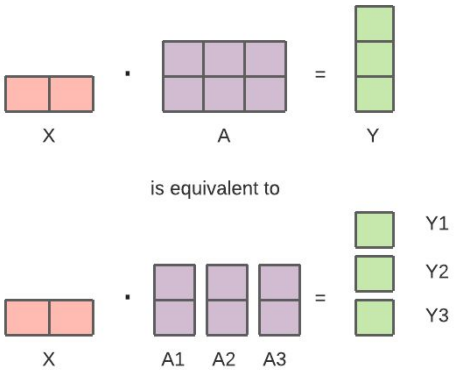


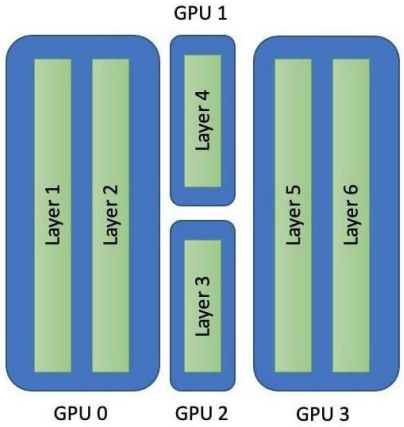


Model Parallelism

On tensor level
("MP", "Horizontal MP")

On layers' level
("Naive MP", "Vertical MP")

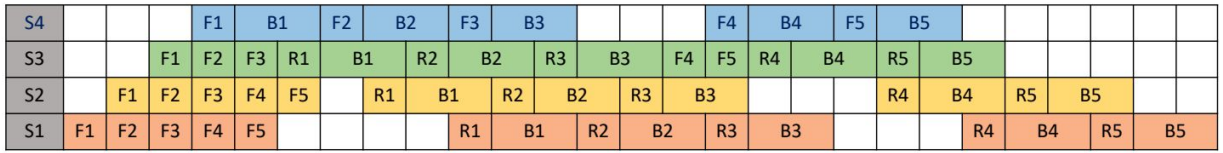




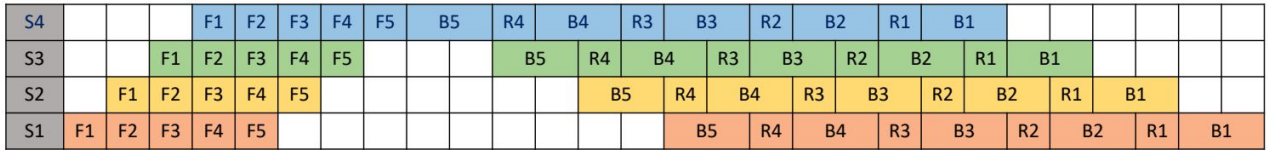
\+ you can train models that do not fit on one device

\- bad GPU utilization: the device waits for the output of the previous layer of the model

To reduce GPU idle time, several approaches have been developed to organize a data pipeline between devices: GPipe, Megatron-LM, Varuna.

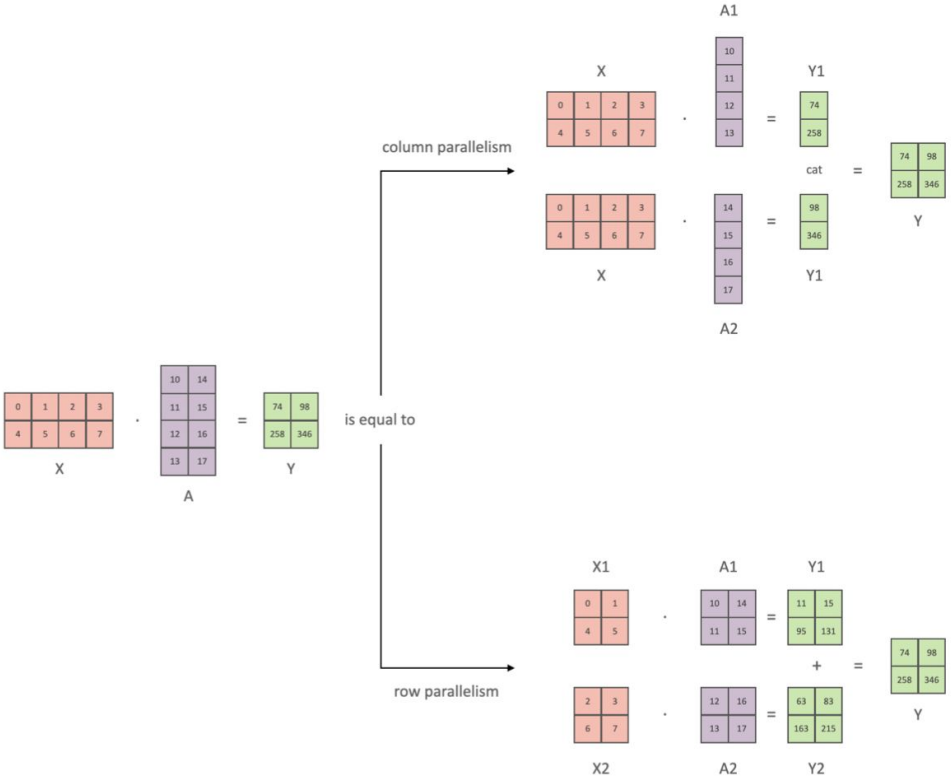


(a) Varuna Schedule

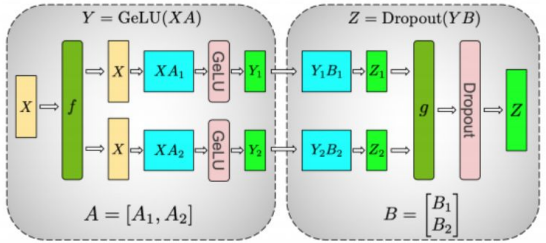


(b) Gpipe Schedule

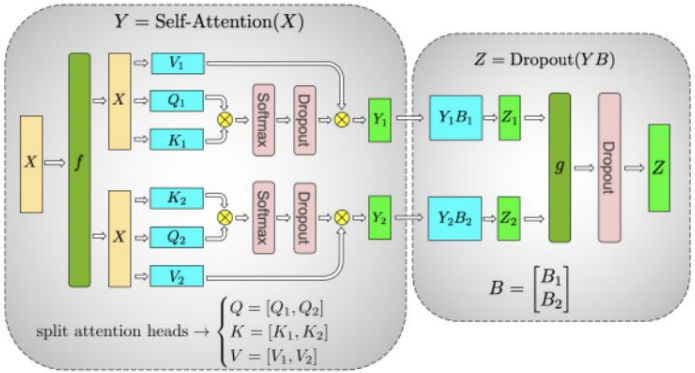
Model Parallelism: on Tensor Level



Tensor parallelism for Transformer

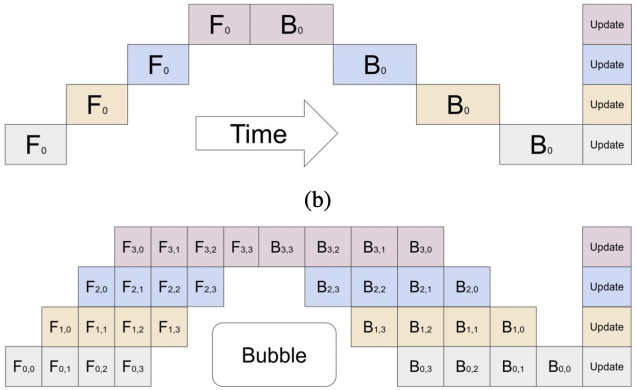


(a) MLP



(b) Self-Attention

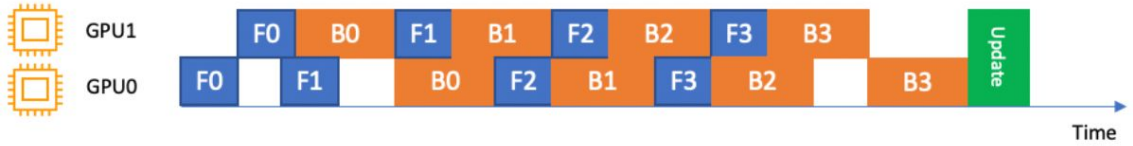
Pipeline Parallelism



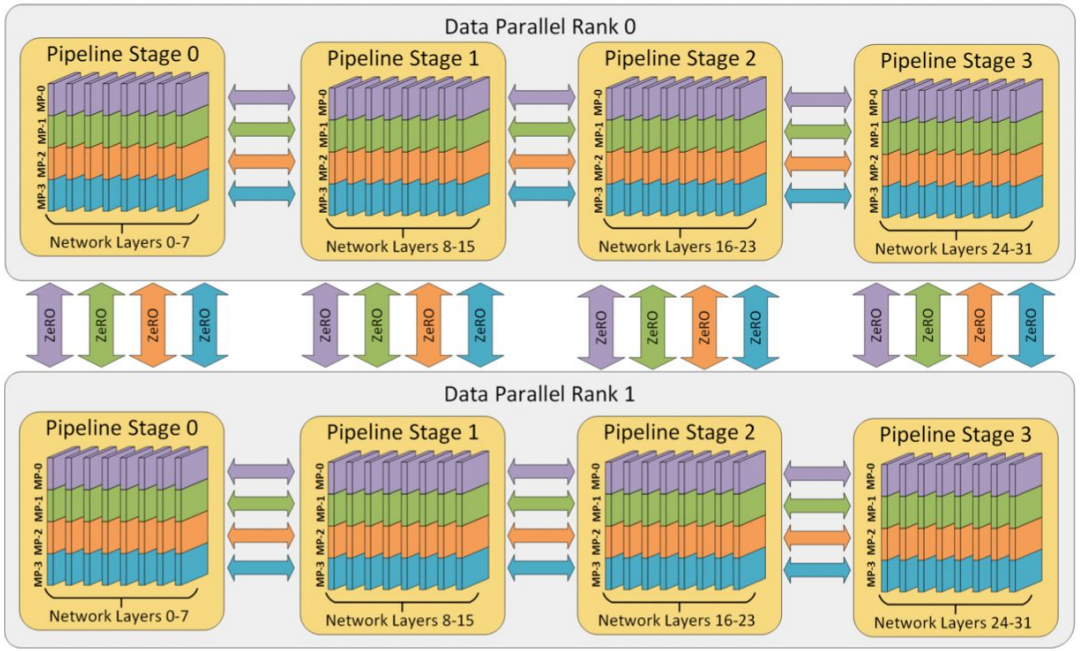
GPipe
batches are divided into micro-batches to reduce downtime

Interleaved Pipeline: Varuna, SageMaker, DeepSpeed

Backward for the first micro-batch is computed earlier than forward for the second micro-batch



3D Parallelism: PP+TP+DP(ZeRO)



32 GPUs are used: 4 groups tesor-parallelism * 4 groups pipeline-parallelism * 2 groups data-parallelism

MP-n denotes tensor-parallelism

Optimal Strategies for Pipeline Parallelism

Paper	FlexFlow	PipeDream	PipeDream-2BM	Piper
Types of parallelism	tensor-, model-, data-, operator-	data-, pipeline-	data-, pipeline-	data-, tensor-, pipeline-
Method for hyperparameters optimization	MCMC	Dynamic Programming	Dynamic Programming	Dynamic Programming
Activation recomputation during gradient computation (Activation checkpointing)	-	-	+	+
Activation offloading to CPU	-	-	-	-
Application	CNNs	CNNs	Transformers	Transformers

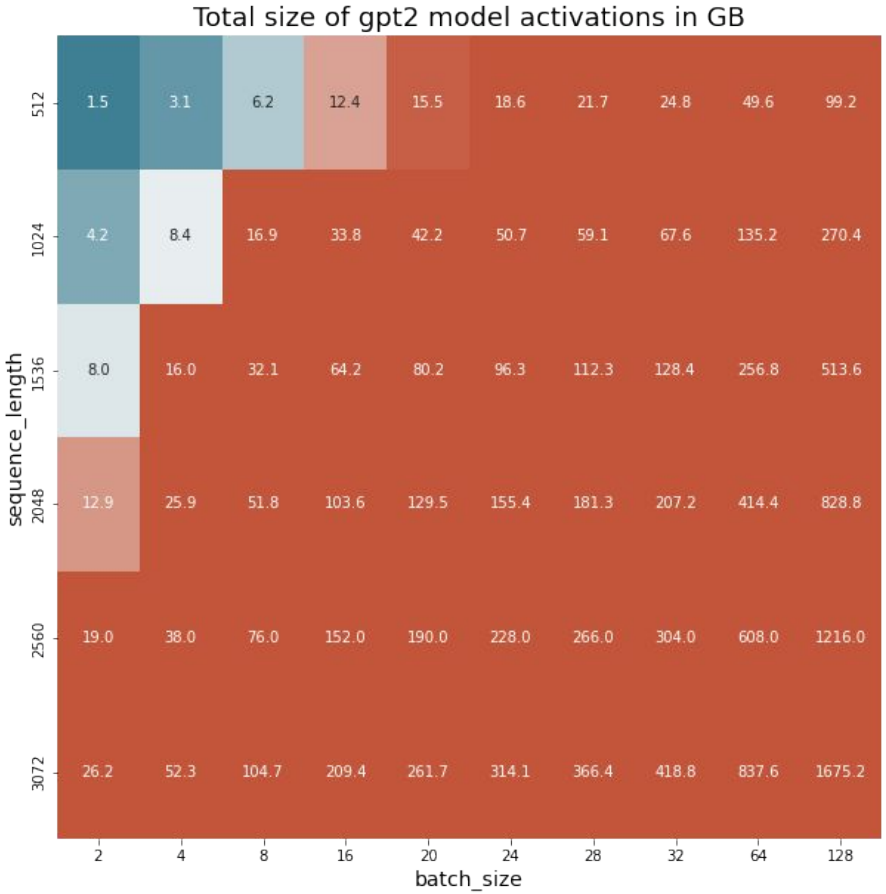
None of the known approaches use:

- activation offloading to the CPU, or
- a combination of two methods, recomputation of activations when computing gradients and offloading activations to CPU.

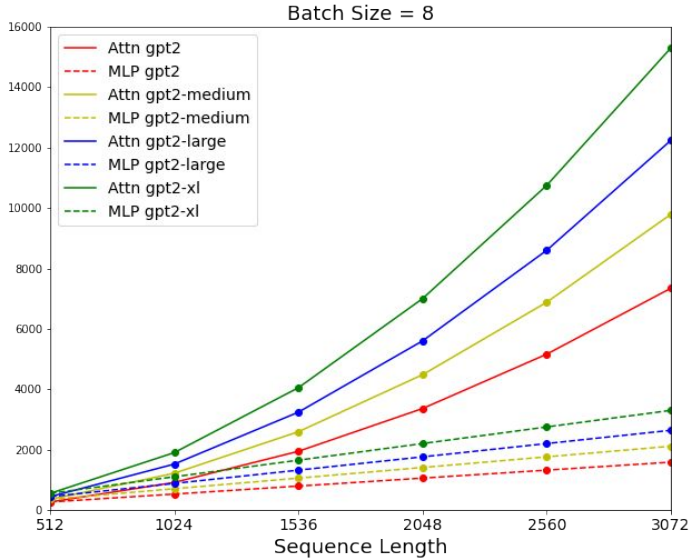
Activation Checkpointing & Offloading to CPU

Lecture 6: Training Speed-up of Large Scale Models

Memory allocated by Activations

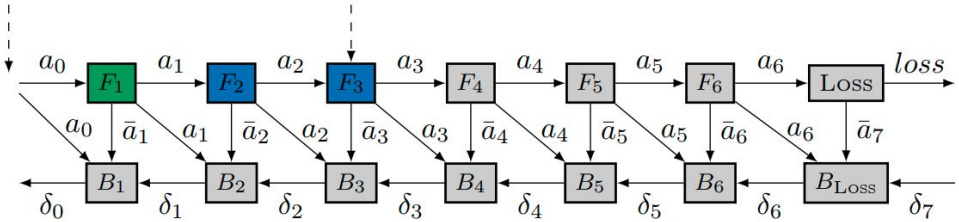


- Activations of Transformer model can take up more memory than a single standard V100 GPU can accommodate.
- Size of memory allocated for activations of the attention block grows quadratically from the length of the sequence.



Methods to Reduce Activations Memory: Rotor

- Saving only part of activations in the forward pass and recomputing the rest during gradients computation;

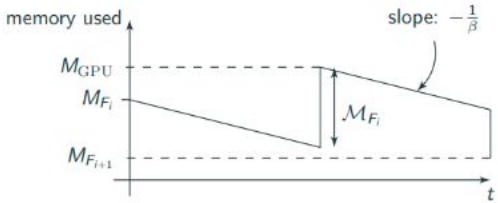


+ saves memory
- slows down training: when computing gradients, you have to recompute activations

Sequence

$F_1^c, F_2^n, F_3^n, F_4^e, F_5^e, F_6^e, \text{Loss}, B_{\text{Loss}}, B_6, B_5, B_4, F_1^c, F_2^n, F_3^e, B_3, F_1^e, F_2^e, B_2, B_1$

- Sending activations to CPU and loading from CPU as needed to calculate gradients;



+ saves memory
- slows down training at low bandwidth β

Optimization Methods

Lecture 6: Training Speed-up of Large Scale Models

Optimization of Large Scale Models

Problem:

$$\frac{1}{N}\sum_{i=1}^{N} L(\Phi(x_i, \boldsymbol{w}), y_i) \to \min_{\boldsymbol{w}},$$

batch size — loss function — training data — model weights

Algorithm:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \boldsymbol{h}_t,$$

step size

(SGD) $$\boldsymbol{h}_t = \boldsymbol{g}_t = \frac{1}{N_b} \sum_{(x_i, y_i) \in X_b} \frac{\partial L(\Phi(x_i, \boldsymbol{w}), y_i)}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{w}=\boldsymbol{w}_t}$$

batch size — batch

Questions:

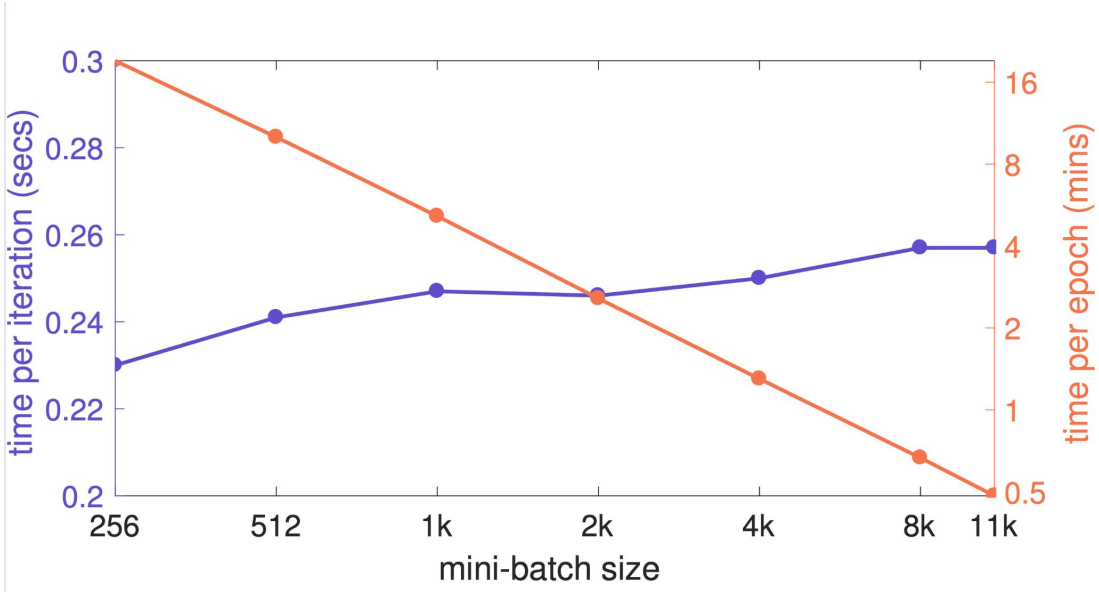
- How to choose step size?
- How to store vectors so they occupy less space?
- How to choose batch size?
- How to use many devices to speed up optimization?
- How to initialize weights?

Practical Recommendations on Optimization

Using batches of bigger sizes

- Training a model with a large batch takes less time due to parallelism.
- However, with a simple increase in the batch, the generalizing ability of the model is worse.
- When the batch size increases by k times, the step size must be increased by k times.

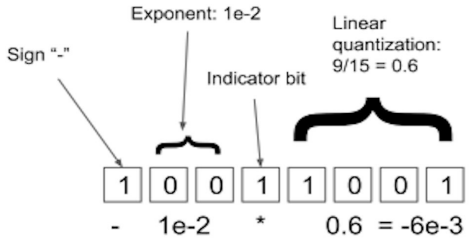
- Increasing the step should be carried out gradually (warmup - phase of the first few epochs).
- Layer-by-layer step size change allows you to increase the batch even more.

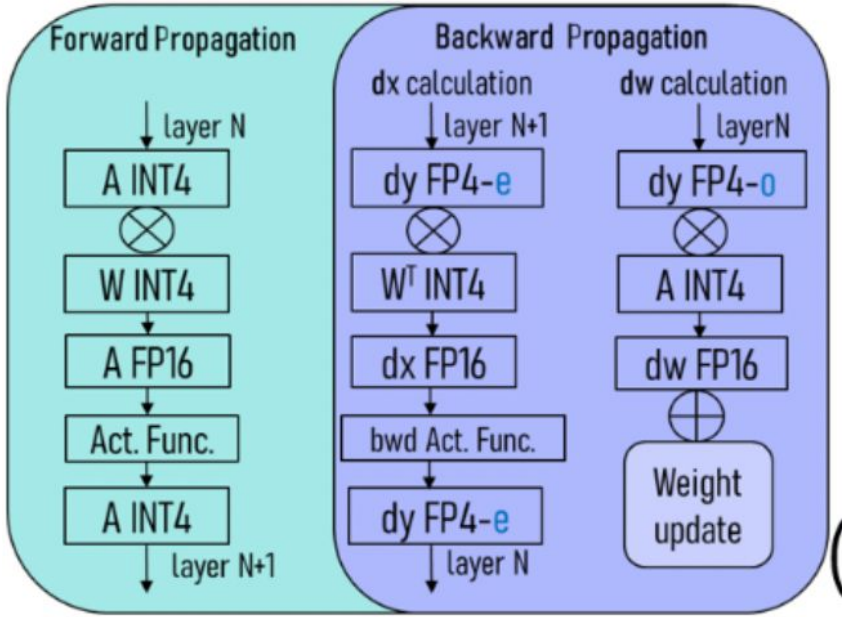


Practical Recommendations on Optimization

Using low-bit formats for data storage

- The use of floating point numbers and block quantization are essential.
- The bitsandbytes library from Facebook contains 8-bit optimizers

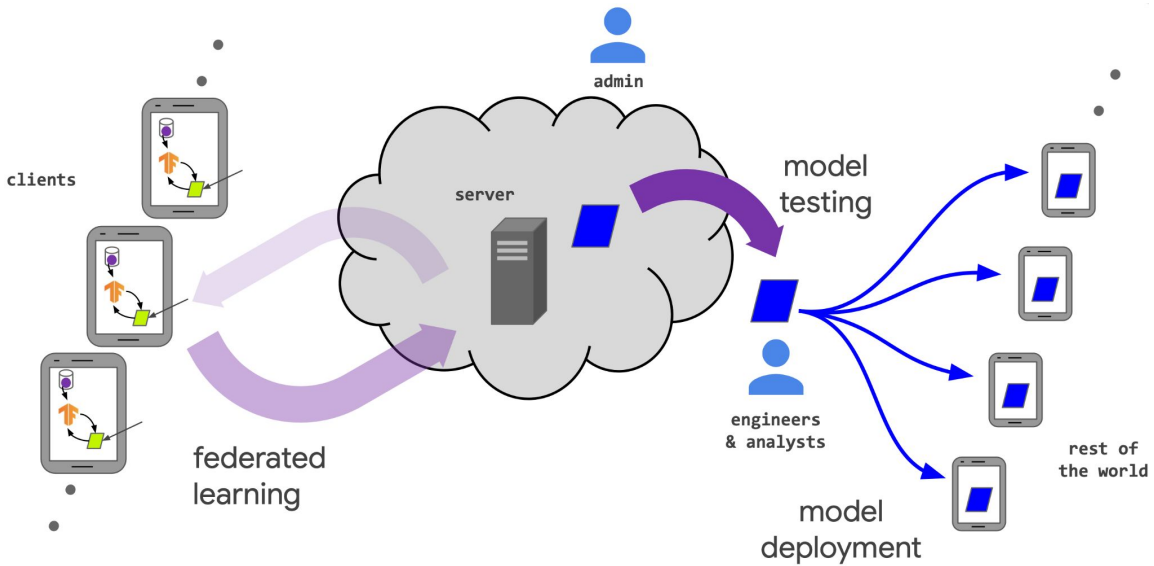




Practical Recommendations on Optimization

Distributed training and federated learning

- By using a large number of parallel computers, you can increase the batch and speed up training.
- Communications can be optimized by transmitting low-rank representations of gradients (PowerSGD and GradZIP methods); sparsification of gradients (Sketched SGD) or quantization of gradients.
- It is possible to do multiple local gradient descent steps on the GPU before forwarding to avoid local minima (post-local SGD).



Frameworks

Lecture 6: Training Speed-up of Large Scale Models

Existing Frameworks

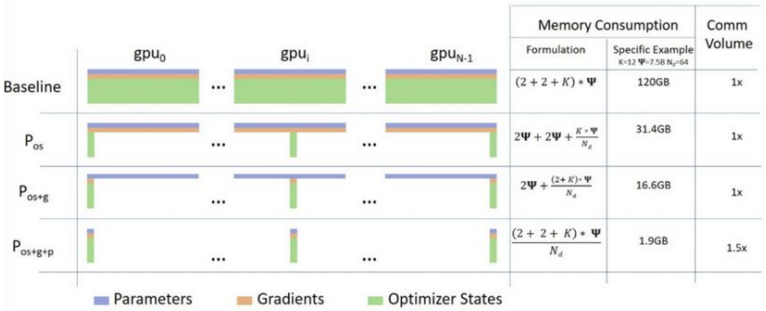
- **Megatron (NVIDIA)**
  - Frameworks to train large scale NLP models (GPT, BERT, T5 ... - Transformer like models) https://github.com/NVIDIA/Megatron-LM
  - Full-stack solution (optimization at all levels: from hardware to user API).

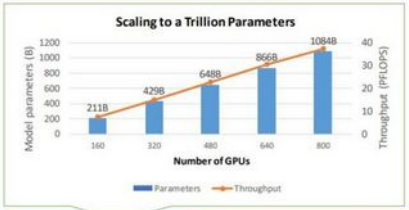
- **DeepSpeed (Microsoft)**
  - Framework for effective training on big number of devices https://www.deepspeed.ai/
  - Solution at the level of optimization of deep learning algorithms.

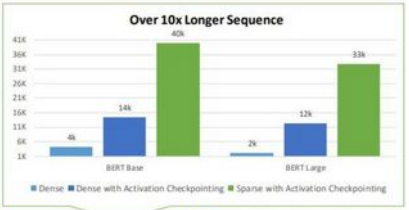
- **OneFlow (NVIDIA)**
  - Convenient framework for custom parallelism prototyping

DeepSpeed from Microsoft

- Data / Pipeline / Model parallelism (can be combined withTensor parallelism)
- Memory optimization with ZeRO (optimizer state/ gradients/ model state partitioning)
- Offloading
- Sparse attention
- и др.





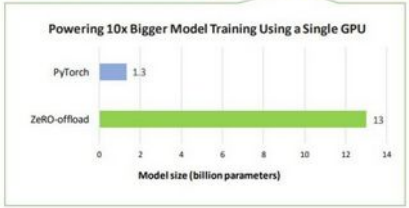


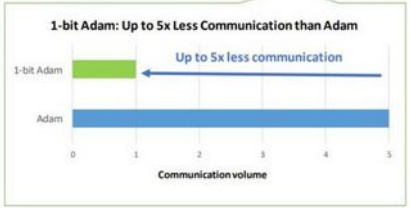
3D Parallelism
- 1 trillion parameter model training

ZeRO-Offload
- 13B model on single GPU, 10x bigger

Sparse Attention
- 10x longer sequence, up to 6x faster

1-bit Adam
- 5x less communication

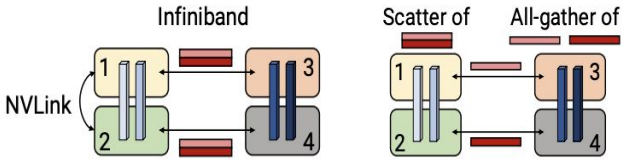




Megatron to Optimize Transformers Training

Megatron (PDT-P):

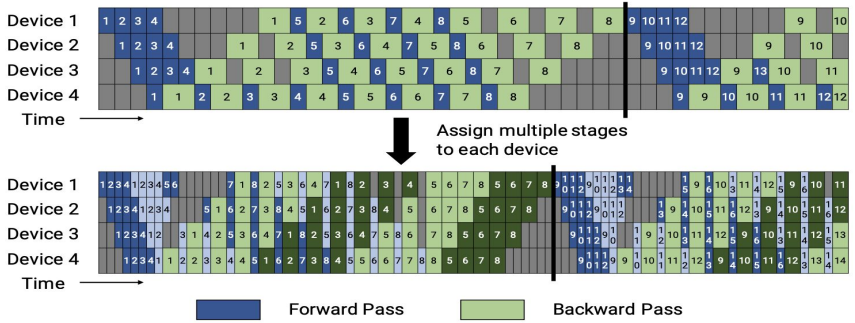
- Parallelism at the level of model layers (different layers on different maps)
- Parallelism at the level of tensors (weights of one layer on different maps)
- Data parallelism - efficient calculation schedule on different GPUs



Scheme of joint application of parallelism of layers and tensors

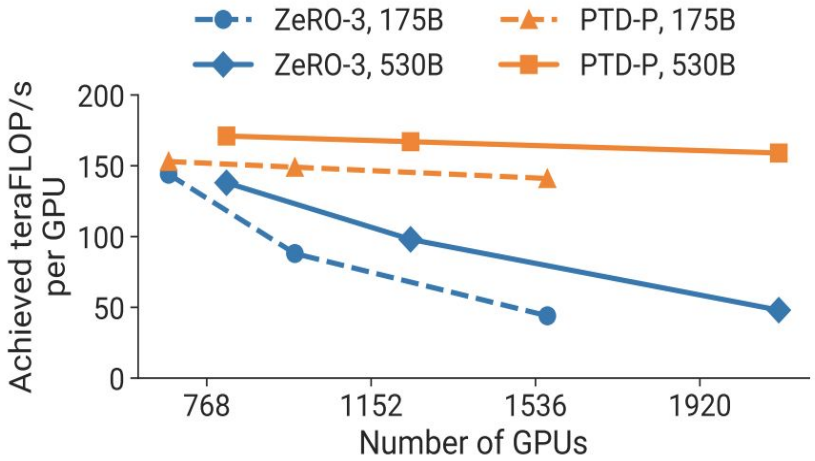
Goal:

- Reduce training time



Megatron (PDT-P) Efficiency

Efficiency of PDT-P and ZeRO-3 optimization method (measured in FLOP/s - number of floating point operations per second):



PDT-P GPT-3 with 175 billion parameters on 1024 GPU trains for 34 days