Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

Nicholas Satchanov
9/11/2024

Background

- Natural language processing uses are attractive
  - Question answering
- NLP field growing quickly due to 2 factors
  - Growth of the Internet -> Increase in availability of training data
  - Better GPUs -> Increase in availability of compute
- Larger models yield higher performance in NLP
- Transformers prove to be the best in NLP
  - Great accuracy
  - Great compute efficiency

Motivation

- Desire for high performing models -> Need models with many parameters
  - Many parameters leads to high memory footprint
  - How to train quickly?
- Current solutions have issues
  - Do not scale well as model size increases
  - Uproots current tools requiring rewriting of compilers
- Want to make training large models more feasible

Existing Work

- Activation checkpointing - Recompute activations in backward pass to avoid storing
  - Reduce memory footprint
  - Utilized in this work between transformers
- ADAM - Training optimizer
  - Requires additional memory per parameter for storing state
- GPipe & Mesh-Tensorflow
  - Frameworks for model parallelism but require model tampering or custom compiler

Key Concepts & Definitions

- Data parallelism vs. model parallelism
  - Data = Split training minibatch across multiple workers
    - Can result in reduced model accuracy or longer convergence time
  - Model = Memory usage and model computation split across multiple workers
    - Alleviate memory pressure
    - Increases amount of parallelism independently of microbatch size
- Model parallelism paradigms
  - Layer-wise pipeline parallelism = Each device performed different computation like pipeline stages
  - General distributed tensor computation = Split tensor ops across different devices
- GEMM = General Matrix Multiply
  - $\boldsymbol{C} \leftarrow \alpha \boldsymbol{A}\boldsymbol{B} + \beta \boldsymbol{C}$

Key Concepts & Definitions

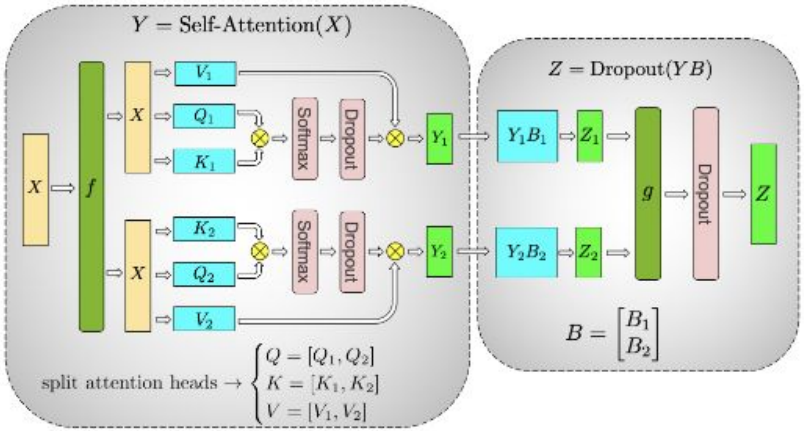
- Data parallelism vs. model parallelism
  - Data = Split training minibatch across multiple workers
    - Can result in reduced model accuracy or longer convergence time
  - Model = Memory usage and model computation split across multiple workers
    - Alleviate memory pressure
    - Increases amount of parallelism independently of microbatch size
- Model parallelism paradigms
  - Layer-wise pipeline parallelism = Each device performed different computation like pipeline stages
  - General distributed tensor computation = Split tensor ops across different devices
- GEMM = General Matrix Multiply
  - $\boldsymbol{C} \leftarrow \alpha \boldsymbol{A}\boldsymbol{B} + \beta \boldsymbol{C}$
- Transformer layer architecture
  - Self attention block
  - Two-layer, multi layer perceptron (MLP) - Feedforward layer

Design - Their Model Parallelism

- Core idea
  - Keep GPUs compute bound
  - Minimize communication between GPUs
- Parallelism in transformers
  - Parallelize attention and MLP separately

Design - Attention Parallelism

- Multi-attention head inherently parallel
- Compute each attention head on separate GPUs
- Since operations independent, don’t need inter-GPU communication



(b) Self-Attention

Design - MLP Parallelism

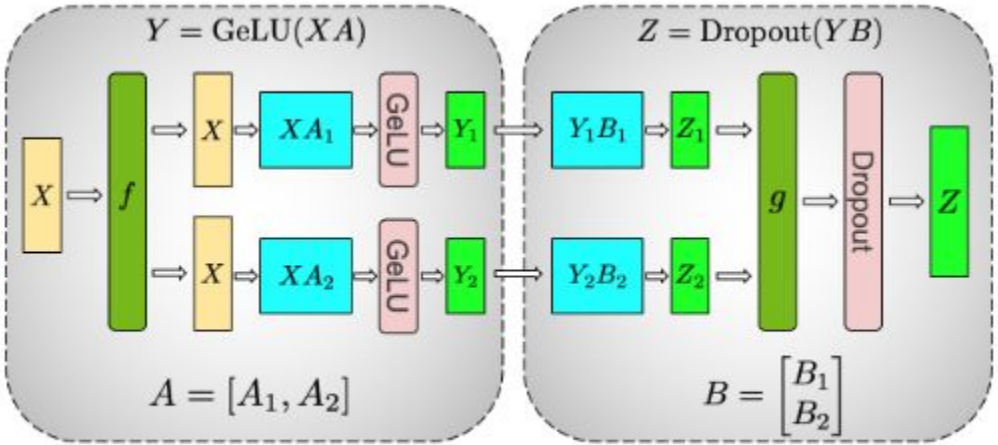
$$X = [X_1, X_2],\ A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$A = [A_1, A_2]$$

- 2 ways to parallelize GEMM operations
  - Split A (weight matrix) among rows -> Non-linearity of GeLU requires synchronization
  - Split A among columns -> Can apply GeLU to individual components; no sync needed
- GEMM 1 -> Split among columns
- GEMM 2 -> Split among rows
- With results split across GPUs

  Need to reunite with all-reduce op
  - Minimal communication cost

```
class f(torch.autograd.Function):
    def forward(ctx, x):
        return x
    def backward(ctx, gradient):
        all_reduce(gradient)
        return gradient
```



(a) MLP

Evaluation Methodology

- Evaluated on real hardware
  - 32 DGX-2H servers (512 Tesla V100 32GB GPUs)
  - NVSwitch - 300GB/sec bandwidth between GPUs in a server
  - InfiniBand - 100GB/sec bandwidth between servers
- Focused on two models
  - BERT
  - GPT2
- Baseline: Single NVIDIA V100 32GB GPU
- 2 improvement methods
  - Model parallelization
  - Model + Data parallelization

Experimental Results

Hidden Size	Attention heads	Number of layers	Number of parameters (billions)	Model parallel GPUs	Model +data parallel GPUs
1536	16	40	1.2	1	64
1920	20	54	2.5	2	128
2304	24	64	4.2	4	256
3072	32	72	8.3	8	512

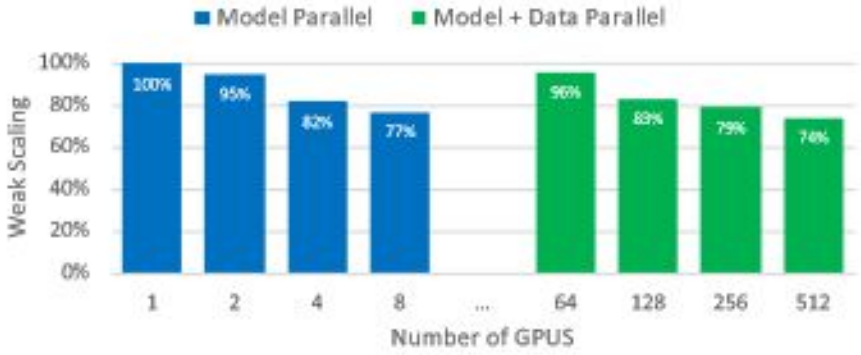
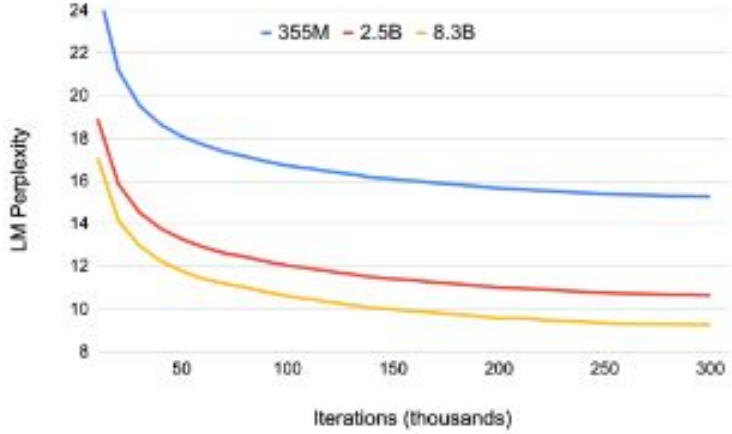


Table 2. Model configurations used for GPT-2.

Parameter Count	Layers	Hidden Size	Attn Heads	Hidden Size per Head	Total GPUs	Time per Epoch (days)
355M	24	1024	16	64	64	0.86
2.5B	54	1920	20	96	128	2.27
8.3B	72	3072	24	128	512	2.10



Thoughts on Paper

- Strengths
  - Can interleave multiple parallelism methods
  - Good scalability
- Weaknesses
  - Not much focus or acknowledgement of model parallelism weaknesses
  - Padding size of parameters for easy parallelism
    - Model changes required
  - Duplication of weights across GPUs
- Purpose of design choices
  - Inherent parallelism of operations exploited
- Future directions of work
  - Design space exploration to see what joint parallelization methods mesh well

The End