Midterm Review

COMP4901Y

Binhang Yuan

What We Have Explored.

RELAXED
SYSTEM LAB

Date	Topic
W1 - 02/04, 02/06	Introduction and Logistics & ML Preliminary
W2 - 02/11, 02/13	Stochastic Gradient Descent & Automatic Differentiation
W3 - 02/18, 02/20	Language Model Architecture & Large Scale Pretrain Overview
W4 - 02/25, 02/27	Nvidia GPU Performance & Collective Communication Library
W5 - 03/04, 03/06	Data-, Pipeline- Parallel Training & Tensor Model-, Optimizer- Parallel Training
W6 - 03/11, 03/13	Sequence-, MoE- parallelism & Mid-Term Review
W7 - 03/18	Mid-Term Exam

Midterm Exam Arrangement

RELAXED SYSTEM LAB

- Date: **March 18, 2025 (Tuesday)**
- Time:
  - 80 minutes, 13:30-14:50
  - If you come late, you still must hand in your paper at 14:50.
- Bring:
  - Your student ID;
  - One page of your cheating sheet in A4 size, printed or hand-written.
  - No electronic devices are allowed.
- Do NOT use a pencil in the exam. Otherwise, you are not allowed to appeal for any grading disagreements!
- The **HKUST Academic Honor Code** applies! I am very serious about this!
- Absence:
  - I must be informed and your appeal must be confirmed by email before the exam starts.

Machine Learning Preliminary & PyTorch Tensors

Einstein Notation in PyTorch

RELAXED
SYSTEM LAB

- Einstein summation in PyTorch:
  - *free index:* index on the right-hand side (e.g., i, j in the above example).
  - *summation index*: index only on the left-hand side, index to be summed over (e.g., k in the above example).
- Execution:
  - Repeated indices among different input operands are multiplied.
  - Summation indices are summed over.
  - The indices on the output side can be permutated.
  - If the right-hand side is ignored, the indices that appear only once on the left-hand side will be placed on the right-hand side by default.

Sample Questions

RELAXED SYSTEM LAB

```
import torch
a = torch.ones(2,2)
# a = tensor([[1., 1.],
#             [1., 1.]])
b = torch.arange(4).reshape(2,2).float()
# a = tensor([[0., 1.],
#             [2., 3.]])
```

What is the right output of the PyTorch Einstein operation `torch.einsum("ij,jk",a,b)`: B

- A. `tensor([[2., 2.], [4., 4.]])`
- B. `tensor([[2., 4.], [2., 4.]])`
- C. `tensor([[1., 3.], [1., 3.]])`
- D. `tensor([[1., 1.], [3., 3.]])`

What is the right output of the PyTorch Einstein operation `torch.einsum("kj,ji",a,b)`: A

- A. `tensor([[2., 2.], [4., 4.]])`
- B. `tensor([[2., 4.], [2., 4.]])`
- C. `tensor([[1., 3.], [1., 3.]])`
- D. `tensor([[1., 1.], [3., 3.]])`

RELAXED
SYSTEM LAB

Stochastic Gradient Descent

Define the Empirical Risk

RELAXED
SYSTEM LAB

- Suppose we have:
  - a dataset $\mathcal{D} = \{(x_1, y_1), (x_1, y_2), \ldots, (x_N, y_N)\}$, where
  - $x_i \in \mathcal{X}$ is the input and
  - $y_i \in \mathcal{Y}$ is the output.
  - Let $h: \mathcal{X} \to \mathcal{Y}$ be a hypothesized model (mapping from input to output) we are trying to evaluate, which is parameterized by $w \in \mathbb{R}^d$.
  - Let $L: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a non-negative loss function which measures how different two outputs are
- The emprical risk R is defined as:

$$R(h_w) = \frac{1}{N} \sum_{i=1}^{N} L(h_w(x_i), y_i)$$

Optimizing the Empirical Loss

- Full gradient descent;
- Stochastic gradient descent;
- Mini-batch Stochastic gradient descent;
- Acceleration of SGD with momentum;
- Second order method;
- Adaptive moment estimation (Adam).

Sample Questions

RELAXED SYSTEM LAB

Given a fixed training dataset, which of the following optimization algorithms requires the least amount of computation in one iteration? B

- A. First-order gradient descent over the whole training set.
- B. First-order stochastic gradient descent.
- C. First-order mini-batch stochastic gradient descent.
- D. Second-order method over the whole training set.

Given an optimization problem of $\min_{\mathbf{w}} f(\mathbf{w})$ where $\mathbf{w} \in \mathbb{R}^D$, which of the following statement is right? D

- A. $\nabla f(\mathbf{w}) \in \mathbb{R}^D$, $\nabla^2 f(\mathbf{w}) \in \mathbb{R}^D$.
- B. $\nabla f(\mathbf{w}) \in \mathbb{R}^{D \times D}$, $\nabla^2 f(\mathbf{w}) \in \mathbb{R}^{D \times D}$.
- C. $\nabla f(\mathbf{w}) \in \mathbb{R}^{D \times D}$, $\nabla^2 f(\mathbf{w}) \in \mathbb{R}^D$.
- D. $\nabla f(\mathbf{w}) \in \mathbb{R}^D$, $\nabla^2 f(\mathbf{w}) \in \mathbb{R}^{D \times D}$.

Automatic Differentiation

Forward Mode AD

RELAXED
SYSTEM LAB

- For computing the derivative of f with respect to x_1, we start by associating with each intermediate variable v_i a derivative (tangent):

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1}$$

- Apply the chain rule to each elementary operation in the forward primal trace;
- Generate the corresponding tangent (derivative) trace;
- Evaluating the primals v_i in lockstep with their corresponding tangents $\dot{v}_i$ gives us the required derivative in the final variable $\dot{v}_5 = \frac{\partial y_1}{\partial x_1}$.

Reverse Mode AD

RELAXED SYSTEM LAB

- Reverse mode AD propagates derivatives backward from a given output.
- We start by complementing each intermediate variable v_i with an adjoint (cotangent) representing the sensitivity of a considered output y_j with respect to changes in v_i:

$$\bar{v}_i = \frac{\partial y_j}{\partial v_i}$$

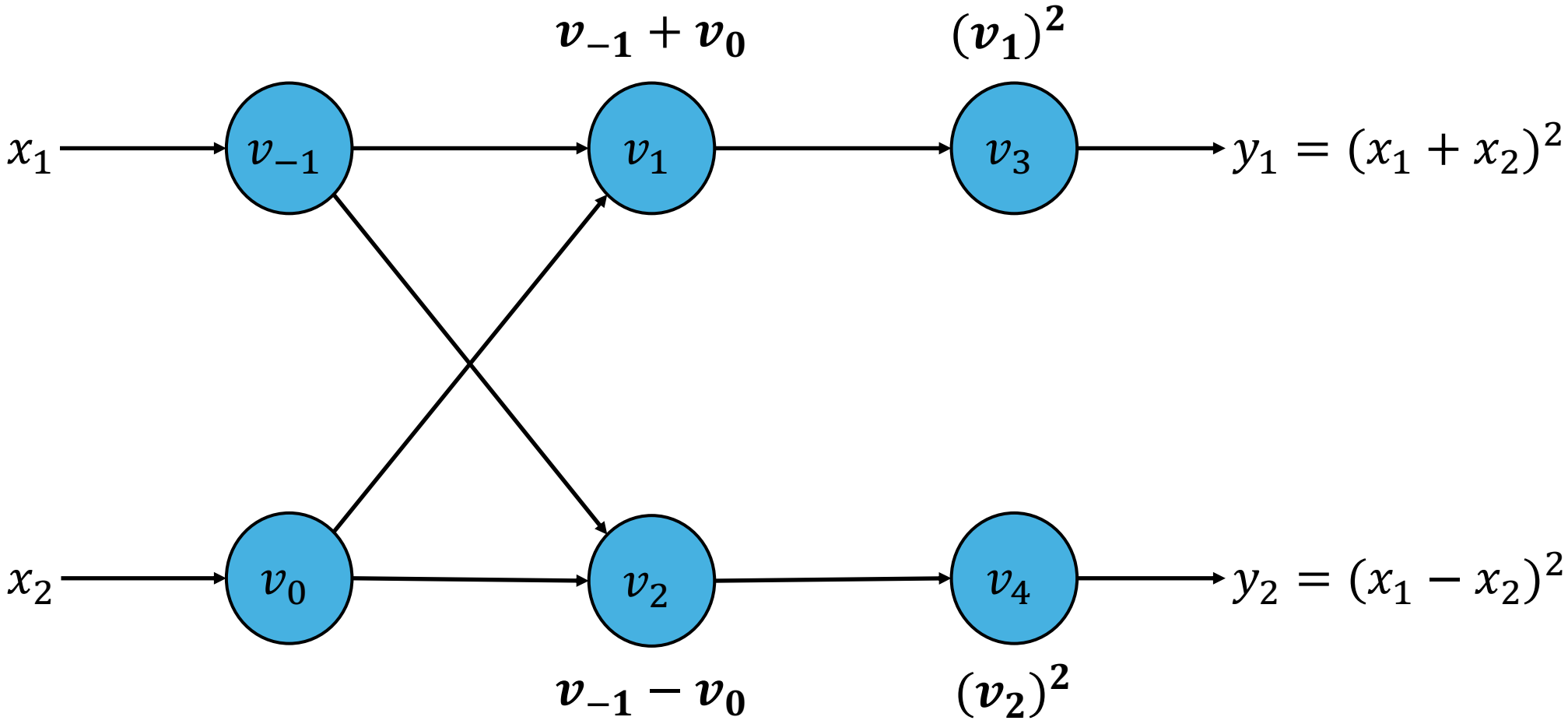
- In the first phase, the original function code is run forward, populating intermediate variables v_i and recording the dependencies in the computational graph.
- In the second phase, derivatives are calculated by propagating adjoints $\bar{v}_i$ in reverse, from the outputs to the inputs.

Chain rule in the multivariable case:

- $y = f(g_1(x), g_2(x), \dots, g_n(x))$;
- $\frac{\partial y}{\partial x} = \sum_{i=1}^{n} \frac{\partial y}{\partial g_i(x)} \frac{\partial g_i(x)}{\partial x}$.

Homework 1 - Q4

RELAXED
SYSTEM LAB



Homework 1 - Q4 Forward Mode

RELAXED SYSTEM LAB

Table 1: forward mode to compute $\frac{\partial y_1}{\partial x_1}, \frac{\partial y_2}{\partial x_1}$:

Forward Primal	Value	Forward Tangent	Value
$v_{-1} = x_1$	2	$\dot{v}_{-1} = \dot{x}_1$	1
$v_0 = x_2$	4	$\dot{v}_0 = \dot{x}_2$	0
$v_1 = v_{-1} + v_0$	6	$\dot{v}_1 = \dot{v}_{-1} + \dot{v}_0$	1
$v_2 = v_{-1} - v_0$	-2	$\dot{v}_2 = \dot{v}_{-1} - \dot{v}_0$	1
$v_3 = (v_1)^2$	36	$\dot{v}_3 = 2v_1\dot{v}_1$	12
$v_4 = (v_2)^2$	4	$\dot{v}_4 = 2v_2\dot{v}_2$	-4
$y_1 = v_3$	36	$\dot{y}_1 = \dot{v}_3$	12
$y_2 = v_4$	4	$\dot{y}_2 = \dot{v}_4$	-4

Table 2: forward mode to compute $\frac{\partial y_1}{\partial x_2}, \frac{\partial y_2}{\partial x_2}$:

Forward Primal	Value	Forward Tangent	Value
$v_{-1} = x_1$	2	$\dot{v}_{-1} = \dot{x}_1$	0
$v_0 = x_2$	4	$\dot{v}_0 = \dot{x}_2$	1
$v_1 = v_{-1} + v_0$	6	$\dot{v}_1 = \dot{v}_{-1} + \dot{v}_0$	1
$v_2 = v_{-1} - v_0$	-2	$\dot{v}_2 = \dot{v}_{-1} - \dot{v}_0$	-1
$v_3 = (v_1)^2$	36	$\dot{v}_3 = 2v_1\dot{v}_1$	12
$v_4 = (v_2)^2$	4	$\dot{v}_4 = 2v_2\dot{v}_2$	4
$y_1 = v_3$	36	$\dot{y}_1 = \dot{v}_3$	12
$y_2 = v_4$	4	$\dot{y}_2 = \dot{v}_4$	4

Homework 1 - Q4 Reverse Mode

RELAXED SYSTEM LAB

Table 3: reverse mode to compute $\frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}$:

Forward Primal	Value	Reverse Adjoint	Value
$v_{-1} = x_1$	2	$\bar{x}_1 = \bar{v}_{-1}$	12
$v_0 = x_2$	4	$\bar{x}_2 = \bar{v}_0$	12
$v_1 = v_{-1} + v_0$	6	$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \times 1$	12
		$\bar{v}_0 = \bar{v}_0 + \bar{v}_1 \times 1$	12
$v_2 = v_{-1} - v_0$	-2	$\bar{v}_{-1} = \bar{v}_2 \times 1$	0
		$\bar{v}_0 = \bar{v}_2 \times -1$	0
$v_3 = (v_1)^2$	36	$\bar{v}_1 = \bar{v}_3 \times 2v_1$	12
$v_4 = (v_2)^2$	4	$\bar{v}_2 = \bar{v}_4 \times 2v_2$	0
$y_1 = v_3$	36	$\bar{y}_1 = \bar{v}_3$	1
$y_2 = v_4$	4	$\bar{y}_2 = \bar{v}_4$	0

Table 4: reverse mode to compute $\frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}$:

Forward Primal	Value	Reverse Adjoint	Value
$v_{-1} = x_1$	2	$\bar{x}_1 = \bar{v}_{-1}$	-4
$v_0 = x_2$	4	$\bar{x}_2 = \bar{v}_0$	4
$v_1 = v_{-1} + v_0$	6	$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \times 1$	-4
		$\bar{v}_0 = \bar{v}_0 + \bar{v}_1 \times 1$	4
$v_2 = v_{-1} - v_0$	-2	$\bar{v}_{-1} = \bar{v}_2 \times 1$	-4
		$\bar{v}_0 = \bar{v}_2 \times -1$	4
$v_3 = (v_1)^2$	36	$\bar{v}_1 = \bar{v}_3 \times 2v_1$	0
$v_4 = (v_2)^2$	4	$\bar{v}_2 = \bar{v}_4 \times 2v_2$	-4
$y_1 = v_3$	36	$\bar{y}_1 = \bar{v}_3$	0
$y_2 = v_4$	4	$\bar{y}_2 = \bar{v}_4$	1

Summary of a Linear Layer Computation

RELAXED SYSTEM LAB

- Forward computation of a linear layer: $\boldsymbol{Y} = \boldsymbol{XW}$
  - Given input: $\boldsymbol{X} \in \mathbb{R}^{B \times D_1}$
  - Given weight matrix: $\boldsymbol{W} \in \mathbb{R}^{D_1 \times D_2}$
  - Compute output: $\boldsymbol{Y} \in \mathbb{R}^{B \times D_2}$
- Backward computation of a linear layer:
  - Given gradients w.r.t output: $\frac{\partial L}{\partial \boldsymbol{Y}} \in \mathbb{R}^{B \times H_2}$
  - Compute gradients w.r.t weight matrix: $\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{X^T} \frac{\partial L}{\partial \boldsymbol{Y}} \in \mathbb{R}^{B \times H_2}$
  - Compute gradients w.r.t input: $\frac{\partial L}{\partial \boldsymbol{X}} = \frac{\partial L}{\partial \boldsymbol{Y}} \boldsymbol{W}^T \in \mathbb{R}^{B \times H_2}$

RELAXED
SYSTEM LAB

Language Model

Autoregressive Language Models

RELAXED SYSTEM LAB

- A common way to write the joint distribution $p(x_{1:L})$ of a sequence to $x_{1:L}$ is using the *chain rule of probablity*:

$$p(x_{1:L}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \dots p(x_L|x_{1:L-1}) = \prod_{i=1}^{L} p(x_i|x_{1:i-1})$$

- In particular, $p(x_i|x_{1:i-1})$ is a conditional probability distribution of the next token x_i given the previous tokens $x_{1:i-1}$.
- An autoregressive language model is one where each conditional distribution $p(x_i|x_{1:i-1})$ can be computed efficiently (e.g., using a feedforward neural network).

TransformerBlocks$(x_{1:L} \in R^{L\times D}) \rightarrow \mathbb{R}^{L\times D}$

RELAXED SYSTEM LAB

- B is the batch size;
- L is the sequence length;
- D is the model dimension;
- Multi-head attention: $D = n_H \times H$
- H is the head dimension;
- n_h is the number of heads.

Computation	Input	Output
$Q = xW^Q$	$x \in \mathbb{R}^{L\times D}, W^Q \in \mathbb{R}^{D\times D}$	$Q \in \mathbb{R}^{L\times D}$
$K = xW^K$	$x \in \mathbb{R}^{L\times D}, W^K \in \mathbb{R}^{D\times D}$	$K \in \mathbb{R}^{L\times D}$
$V = xW^V$	$x \in \mathbb{R}^{L\times D}, W^V \in \mathbb{R}^{D\times D}$	$V \in \mathbb{R}^{L\times D}$
$[Q_1, Q_2 \dots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L\times D}$	$Q_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$[K_1, K_2 \dots, K_{n_h}] = \text{Partition}_{-1}(K)$	$K \in \mathbb{R}^{L\times D}$	$K_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$[V_1, V_2 \dots, V_{n_h}] = \text{Partition}_{-1}(V)$	$V \in \mathbb{R}^{L\times D}$	$V_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$\text{Score}_i = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \dots n_h$	$Q_i, K_i \in \mathbb{R}^{L\times H}$	$\text{score}_i \in \mathbb{R}^{L\times L}$
$Z_i = \text{score}_i\ V_i, i = 1, \dots n_h$	$\text{score}_i \in \mathbb{R}^{L\times L}, V_i \in \mathbb{R}^{L\times H}$	$Z_i \in \mathbb{R}^{L\times H}$
$Z = \text{Merge}_{-1}([Z_1, Z_2 \dots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$	$Z \in \mathbb{R}^{L\times D}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{L\times D}, W^O \in \mathbb{R}^{D\times D}$	$\text{Out} \in \mathbb{R}^{L\times D}$
$A = \text{Out}\ W^1$	$\text{Out} \in \mathbb{R}^{L\times D}, W^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{L\times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{L\times 4D}$	$A' \in \mathbb{R}^{L\times 4D}$
$x' = A'W^2$	$A' \in \mathbb{R}^{L\times 4D}, W^2 \in \mathbb{R}^{4D\times D}$	$x' \in \mathbb{R}^{L\times D}$

Scaling Law

RELAXED SYSTEM LAB

- Intuitively:
  - Increase parameter # N → better performance
  - Increase dataset scale D → better performance
- But we have a fixed computational budget on $C \approx 6ND$
- ***To maximize model performance, how should we allocate C to N and D?***

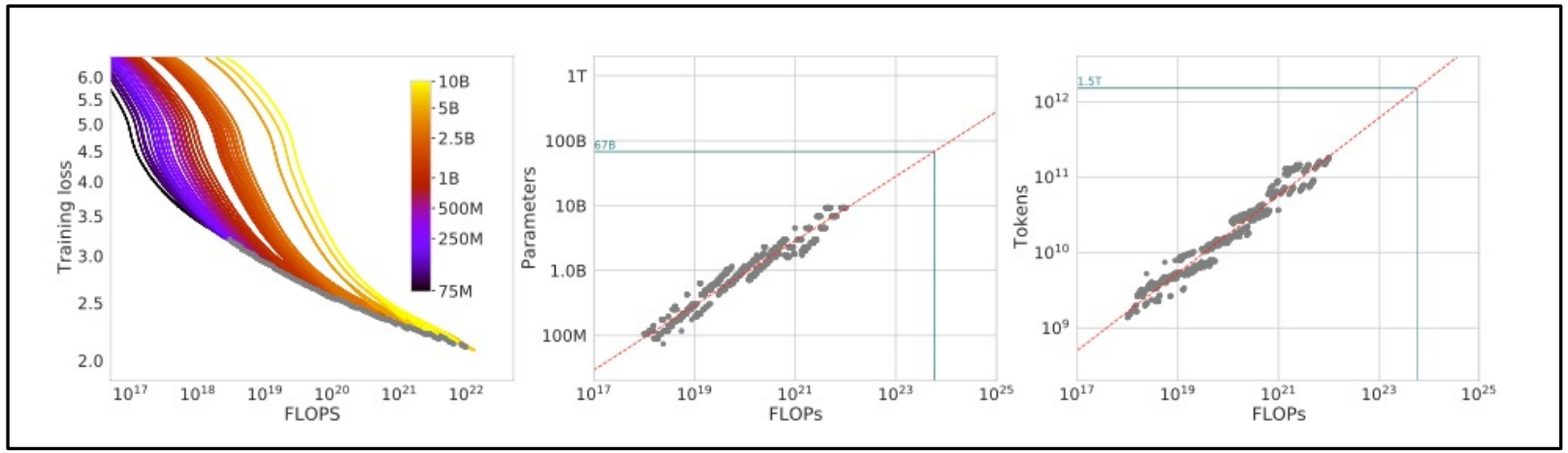
DeepMind

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*
*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly under-trained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

Chinchilla Scaling Law



- Conduct a series of benchmarks and optimizations to fit the function;
- Results:
  - $\alpha = 0.34,\ \beta = 0.28, A = 406.4, B = 410.7, E = 1.69$
- Conclusion:
  - $$\begin{cases} N_{opt}(C) = 0.1C^{0.5} \\ D_{opt}(C) = 1.7C^{0.5} \end{cases}$$

Sample Questions

RELAXED SYSTEM LAB

The widely adopted foundation model Llama-2 belongs to which of the following categories? C

- A. Encoder-only model.
- B. Encoder-decoder model.
- C. Decoder-only model.
- D. None of the above.

Suppose C is the computation budget in terms of FLOPS, $N_{\text{opt}}(C)$ and $D_{\text{opt}}(C)$ are the optimal parameter scale and data scale according to the Chinchilla Scaling Law. Which of the following should be the most reasonable guess in the cell marked by "?" in the following table? A

C	$N_{\text{opt}}(C)$	$D_{\text{opt}}(C)$
$1.21E+20$	1 Billion	20.2 Billion
$1.23E+22$	10 Billion	205.1 Billion
$1.27E+26$	1 Trillion	?

- A. 21.2 Trillion
- B. 19.7 Billion
- C. 20.6 Million
- D. 216.2 Trillion

Evaluating Distributed Computation

RELAXED SYSTEM LAB

- Scaling law tells us given a fixed computation budget, how should we decide the model scale and data corpus.
- The computation budget is formulated by the total FLOPs demanded during the computation.
- But the GPU cannot usually work at its peak FLOPs.
- *How can we evaluate the performance of a distributed training workflow?*
  - Training throughput (token per second);
  - Scalability;
  - Model FLOPs Utilization.

Sample Questions

RELAXED SYSTEM LAB

For a distributed program, suppose the proportion of execution time spent on the serial part $s = 0.1$ (the parallelized part $p = 0.9$), what is the **strong** scalability speedup when the number of processes $N = 10$? B

- A. 2.7
- B. 5.3
- C. 9.1
- D. 10.0

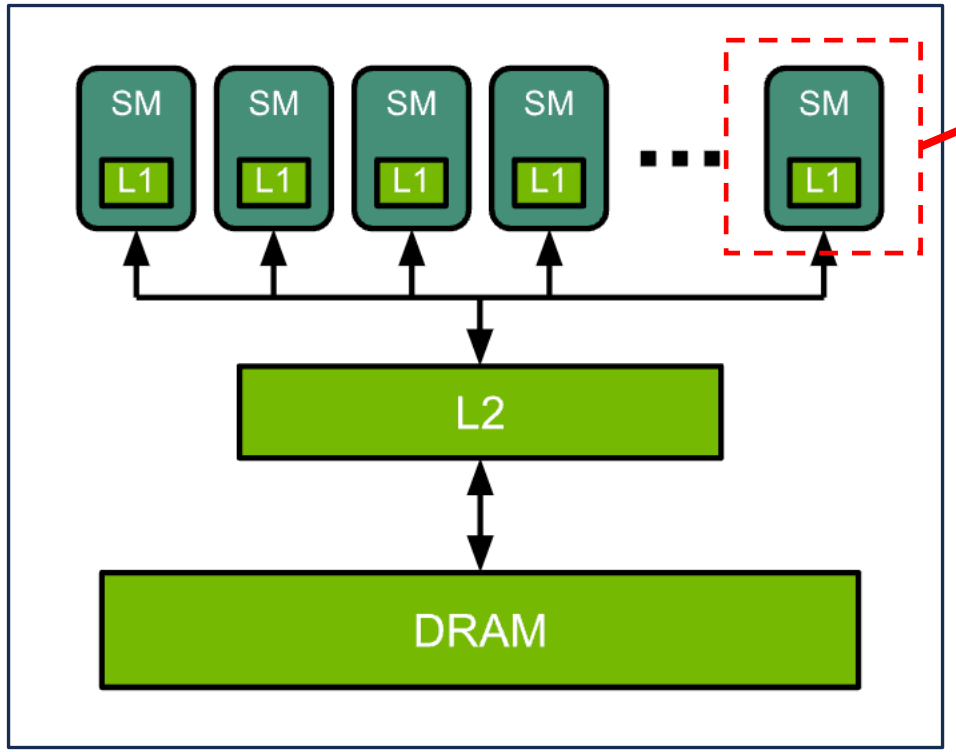
For a distributed program, suppose the proportion of execution time spent on the serial part $s = 0.1$ (the parallelized part $p = 0.9$), what is the **weak** scalability speedup when the number of processes $N = 10$? C

- A. 2.7
- B. 5.3
- C. 9.1
- D. 10.0

GPU Computation and Communication

Ampere GPU Architecture

RELAXED SYSTEM LAB

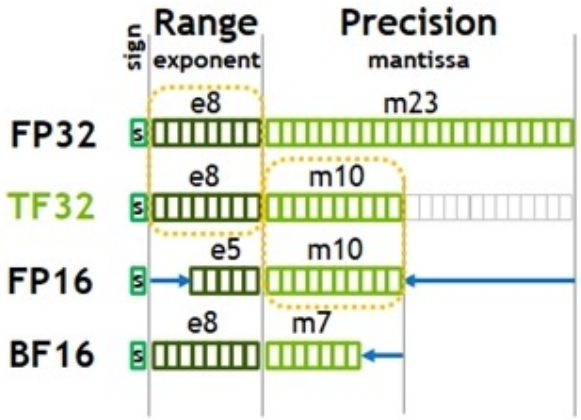


108 SM in a A100 GPU

A100 GPU Tensor Core Computation

RELAXED SYSTEM LAB

- Multiply-add is the most frequent operation in modern neural networks. This is known as the fused multiply-add (FMA) operation.
- Includes one multiply operation and one add operation, counted as two float operations.
- A100 GPU has 1.41 GHz clock rate.
- The Ampere A100 GPU Tensor Cores multiply-add operations per clock:



Ampere A100 GPU FMA per clock on a SM					
FP64	TF32	FP16	INT8	INT4	INT1
64	512	1024	2048	4096	16384

Arithmetic Intensity

RELAXED
SYSTEM LAB

- Thus, on a given processor a given algorithm is math limited if:
  - $T_{math} > T_{mem}$
  - $\frac{\#op}{BW_{math}} > \frac{\#bytes}{BW_{mem}}$

- By simple algebra, the inequality can be rearranged to:
  - $\frac{\#op}{\#bytes} > \frac{BW_{math}}{BW_{mem}}$

- The left-hand side: the algorithm's *arithmetic intensity*.
- The right-hand side: *ops:byte ratio*.

Sample Questions

RELAXED SYSTEM LAB

The Nvidia A100 GPU has 1.41 GHz clock rate; there are 108 SM in one A100 GPU; The peak tensor core TF32 performance is 156 TFLOPS. Given the above facts, how many TF32 FMA can one SM process per clock? C

- A. 128.
- B. 256.
- C. 512.
- D. 1024.

Which of the following matrix operations has the largest arithmetic intensity? A

- A. $\mathbf{Y} = \mathbf{X}\mathbf{W}, \mathbf{X} \in \mathbb{R}^{4096\times4096}, \mathbf{W} \in \mathbb{R}^{4096\times4096}$
- B. $\mathbf{Y} = \mathbf{X}\mathbf{W}, \mathbf{X} \in \mathbb{R}^{4096\times32}, \mathbf{W} \in \mathbb{R}^{32\times4096}$
- C. $\mathbf{Y} = \mathbf{X}\mathbf{W}, \mathbf{X} \in \mathbb{R}^{32\times4096}, \mathbf{W} \in \mathbb{R}^{4096\times32}$
- D. $\mathbf{Y} = \mathbf{X} + \mathbf{W}, \mathbf{X} \in \mathbb{R}^{4096\times4096}, \mathbf{W} \in \mathbb{R}^{4096\times4096}$

NCCL Operators

RELAXED SYSTEM LAB

- Optimized collective communication library from Nvidia to enable high-performance communication between CUDA devices.
- NCCL Implements:
  - **AllReduce**;
  - **Broadcast;**
  - **Reduce;**
  - **AllGather;**
  - **Scatter;**
  - **Gather;**
  - **ReduceScatter;**
  - **AlltoAll.**
- Easy to integrate into DL framework (e.g., PyTorch).

Ring based AllReduce

RELAXED
SYSTEM LAB

- In ring based AllReduce, we assume:
  - N: bytes to aggregate
  - B: bandwidth of each link
  - k: number of GPUs
  - The original tensor is equally split into k chunks.
- Ring based AllReduce implementation has two phases:
  - Reduction phrase (Aggregation phrase);
  - AllGather Phrase.
  - Total time: $\frac{2(k-1)N}{kB}$.

Sample Questions

RELAXED SYSTEM LAB

Assume there are two GPUs in the default communication group, GPU-0 stores a tensor `X=tensor([0., 1., 2., 3.,])`, GPU-1 stores a tensor `X=tensor([3., 2., 1., 0.,])`, what will be stored in tensor A after executing `dist.reduce(X, dst=0, op=RedOpType.SUM,async_op=False)` in GPU-1? B

- A. `X=tensor([0., 1., 2., 3.,])`
- B. `X=tensor([3., 2., 1., 0.,])`
- C. `X=tensor([0., 0., 0., 0.,])`
- D. `X=tensor([3., 3., 3., 3.,])`

Assume there are two GPUs in the default communication group, GPU-0 stores a tensor `X=tensor([0., 1., 2., 3.,])`, GPU-1 stores a tensor `X=tensor([3., 2., 1., 0.,])`, what will be stored in tensor A after executing `dist.all_reduce(X, op=RedOpType.SUM,async_op=False)` in GPU-1? D

- A. `X=tensor([0., 1., 2., 3.,])`
- B. `X=tensor([3., 2., 1., 0.,])`
- C. `X=tensor([0., 0., 0., 0.,])`
- D. `X=tensor([3., 3., 3., 3.,])`

Parallel Training

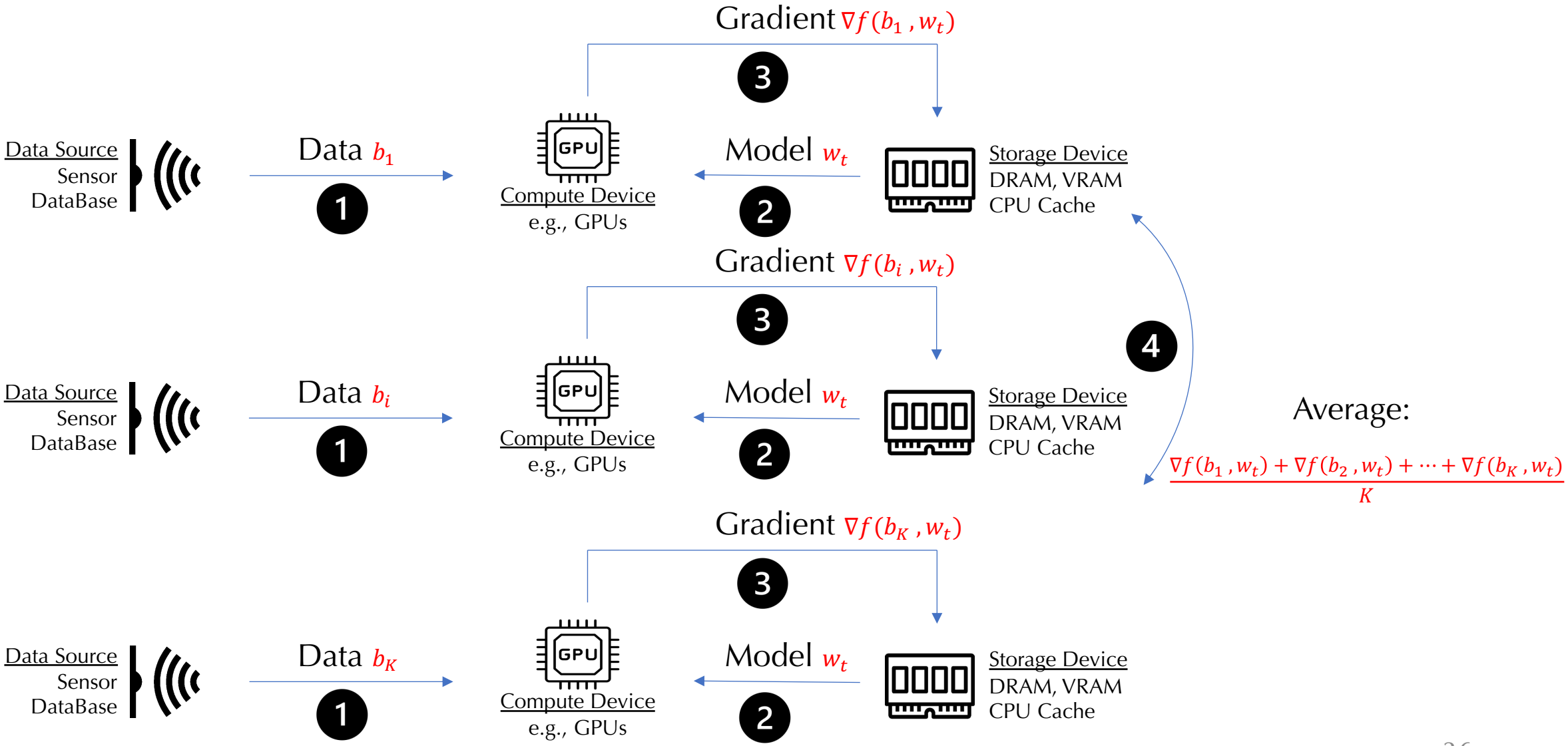
Parallel Training

RELAXED
SYSTEM LAB

- Categories:
  - Data parallelism;
  - Pipeline parallelism;
  - Tensor model parallelism;
  - Optimizer parallelism.
  - MoE parallelism;
  - Parallelism for long sequence.
- What are the communication paradigms?
  - Communication targets;
  - (Collective) Communication operator(s);
  - Communication volume (i.e., corresponding number of bytes of the communication targets).
- What are their advantages and disadvantages?

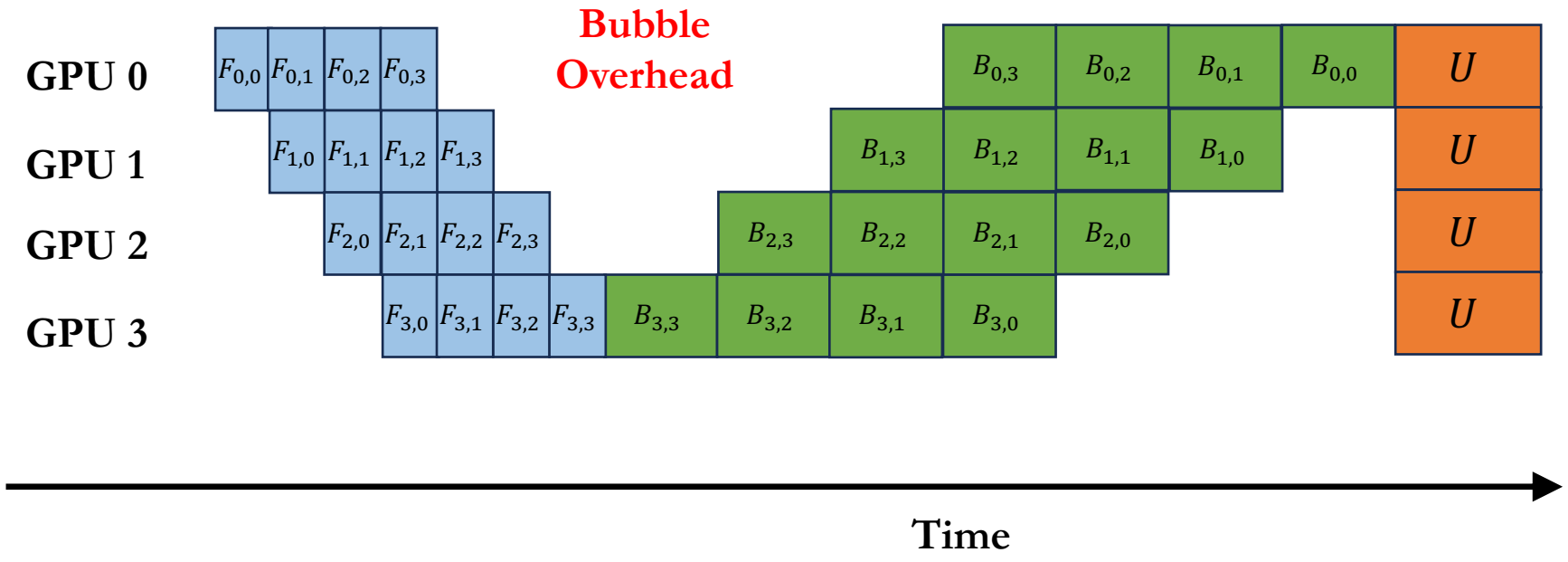
Data Parallel SGD

RELAXED SYSTEM LAB



Pipeline Parallelism - GPipe

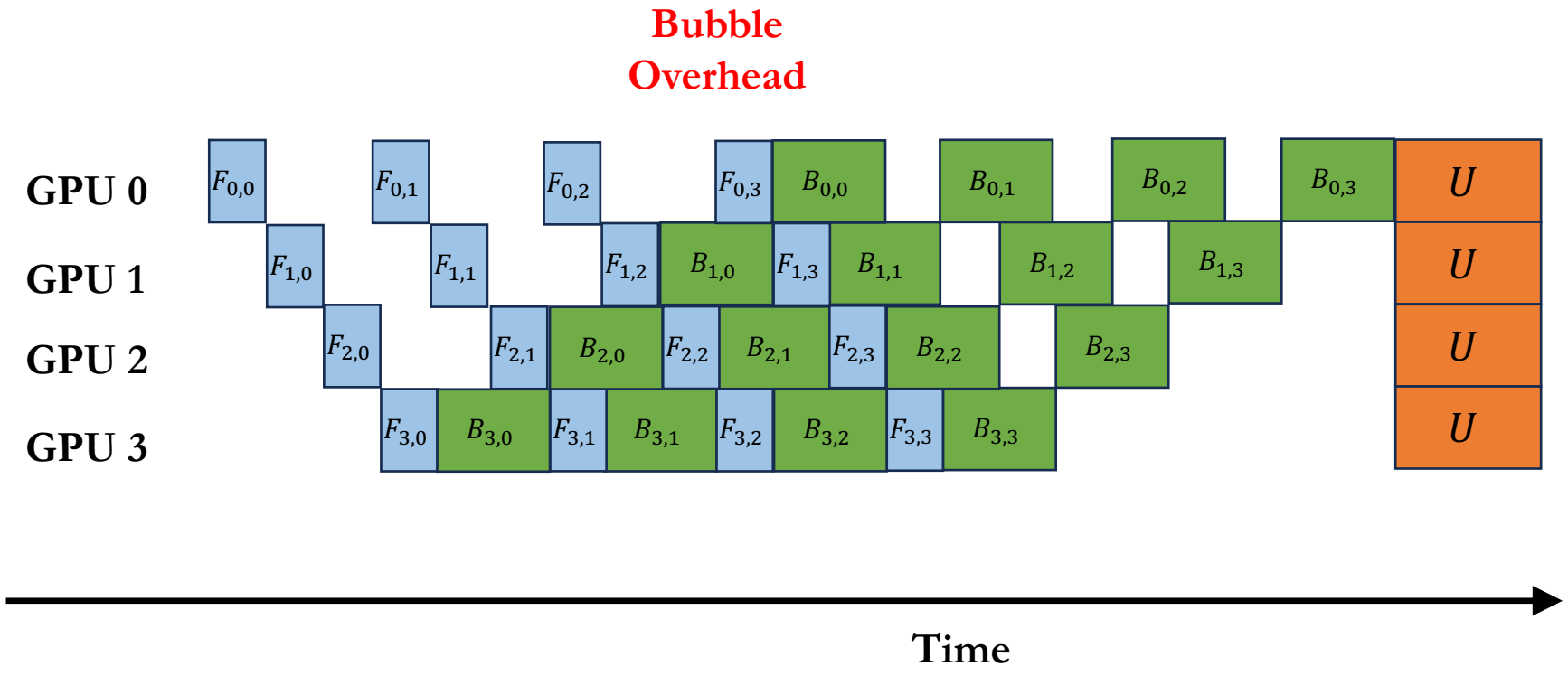
RELAXED SYSTEM LAB



- If we ignore the computation time of optimizer updates.
- Suppose:
  - K is the number of GPUs;
  - M is the number of micro-batches;
- What is the percentage of bubble overhead? $\frac{K-1}{M+K-1}$

The number on each block represents the stage index and the micro-batch index.

Pipeline Parallelism - 1F1B

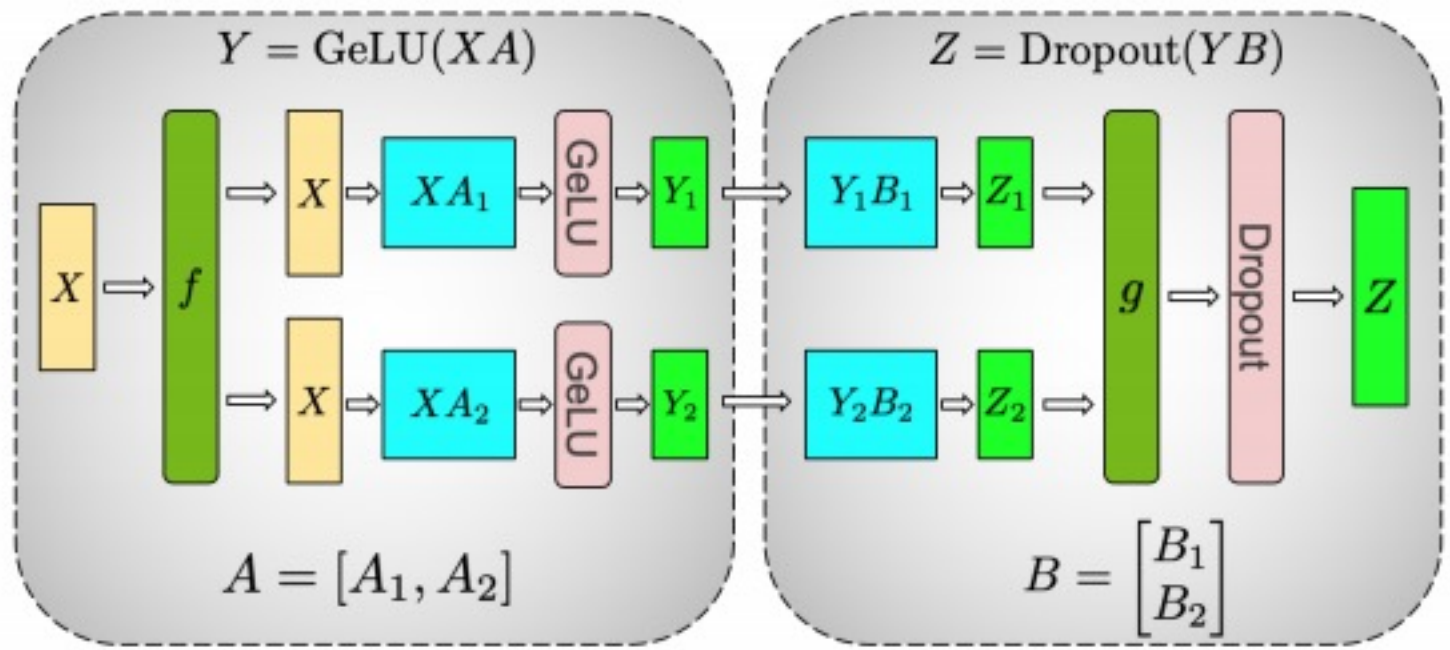


- If we ignore the computation time of optimizer updates.
- Suppose:
  - K is the number of GPUs;
  - M is the number of micro-batches;
- What is the percentage of bubble overhead? $\frac{K-1}{M+K-1}$

The number on each block represents the stage index and the micro-batch index.

MLP in Tensor Model Parallelism

RELAXED SYSTEM LAB

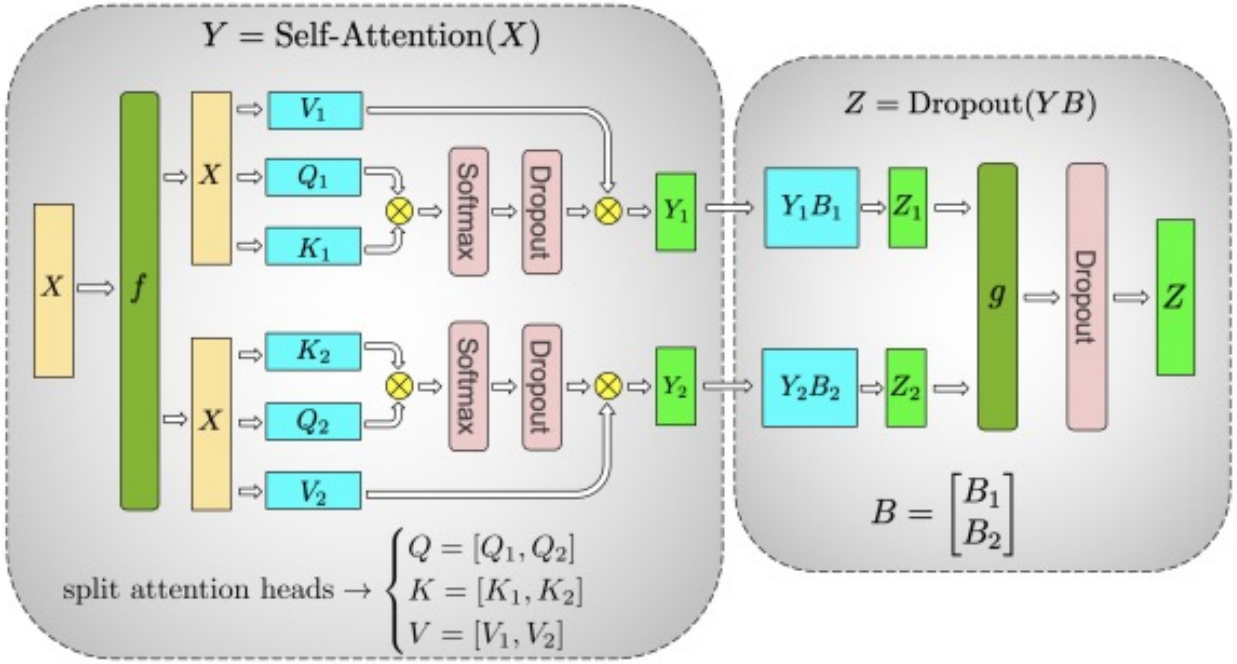


(a) MLP

- f is the identity operator in the forward pass and the **AllReduce** operator in the backward pass.
- g is the **AllReduce** operator in the forward pass and the identity operator in the backward pass.

Multi-Head Attention in Tensor Model Parallelism

RELAXED SYSTEM LAB

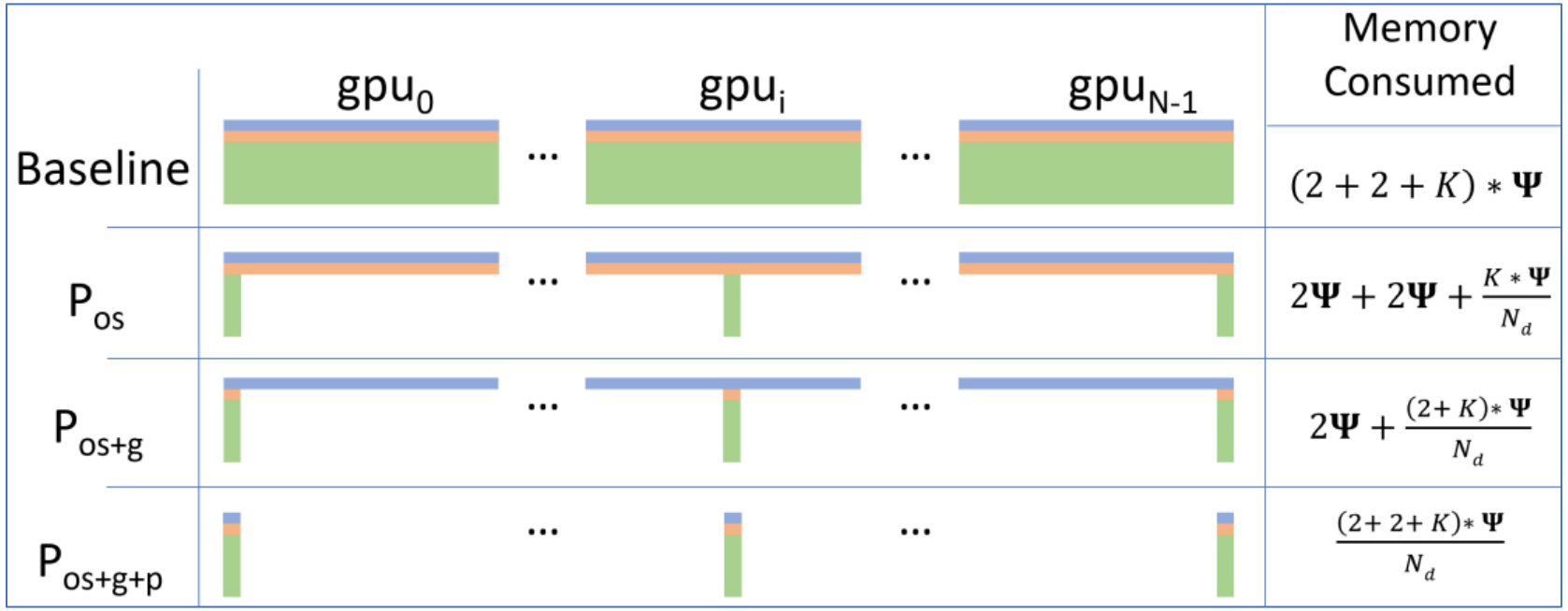


(b) Self-Attention

- f is the identity operator in the forward pass and the **AllReduce** operator in the backward pass.
- g is the **AllReduce** operator in the forward pass and the identity operator in the backward pass.

Zero Redundancy Optimizer (ZeRO)

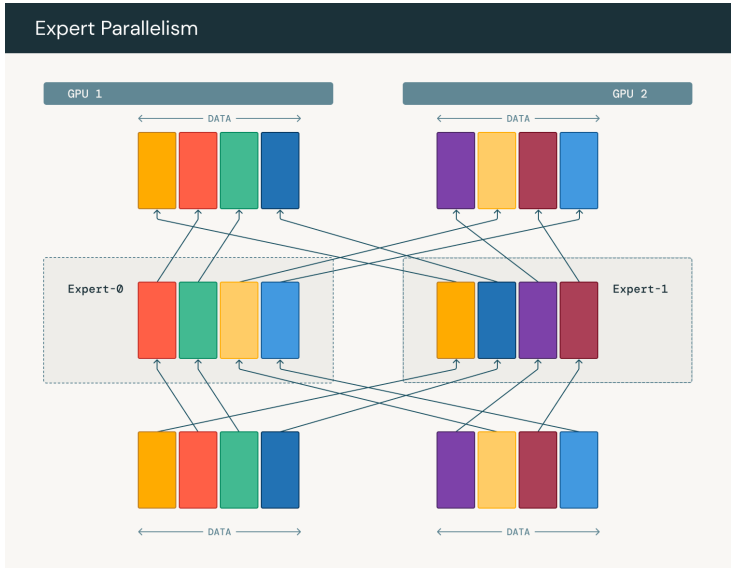
RELAXED SYSTEM LAB

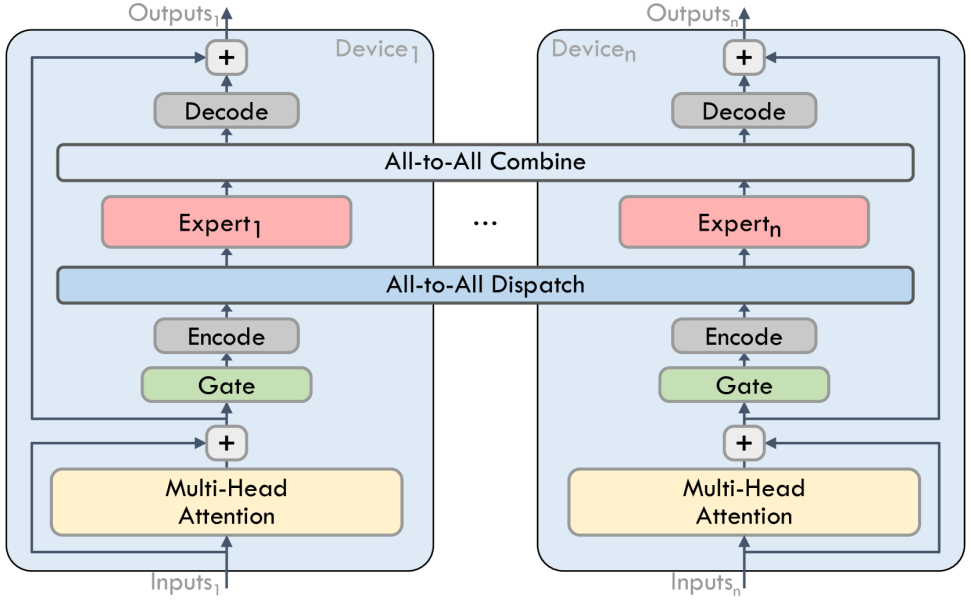


- ψ is the total number of parameters;
- K denotes the memory multiplier of optimizer states;
- N_d denotes the parallel degree.

MoE Parallelism

- In the forward pass:
  - Two **AlltoAll** operations to communicate the activations:
    - One **AlltoAll** operation dispatches the routed activations in the current data batch to the corresponding experts;
    - One **AlltoAll** operation combines the computed expert output in the data batch that needs to be processed by the device.
  - Notice that there could be a little additional overhead to first share the shape of the dispatch tensors.
- In the backward pass:
  - Two **AlltoAll** operations to communicate the corresponding gradients of the activations back to the original device.
  - **AllReduce** operations to synchronize the gradients of non-expert model weights.

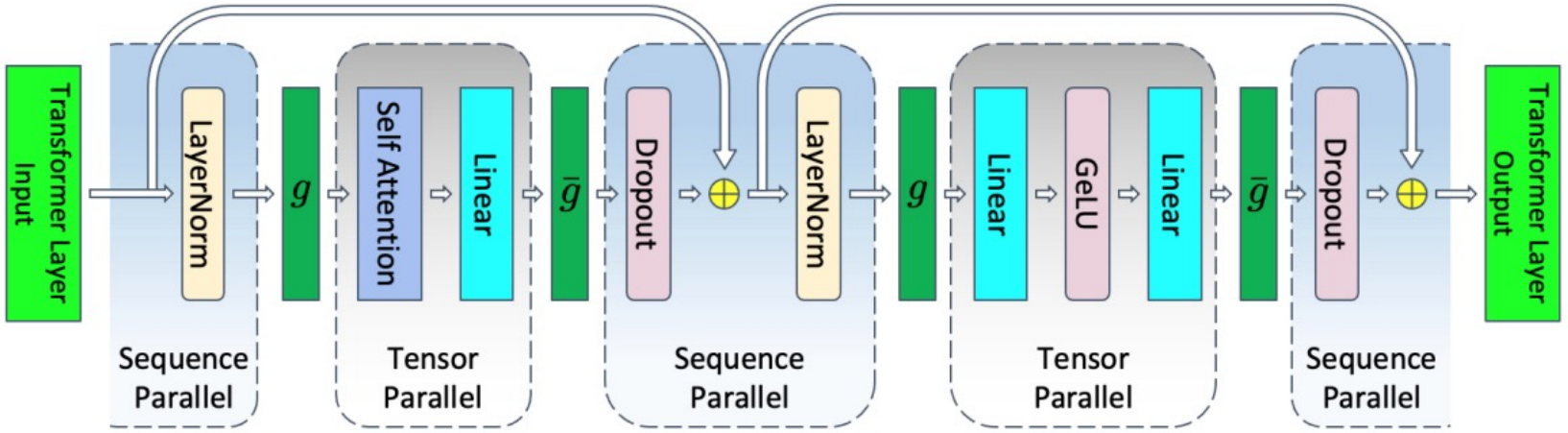




RELAXED SYSTEM LAB

Megatron Sequence Parallelism

RELAXED SYSTEM LAB



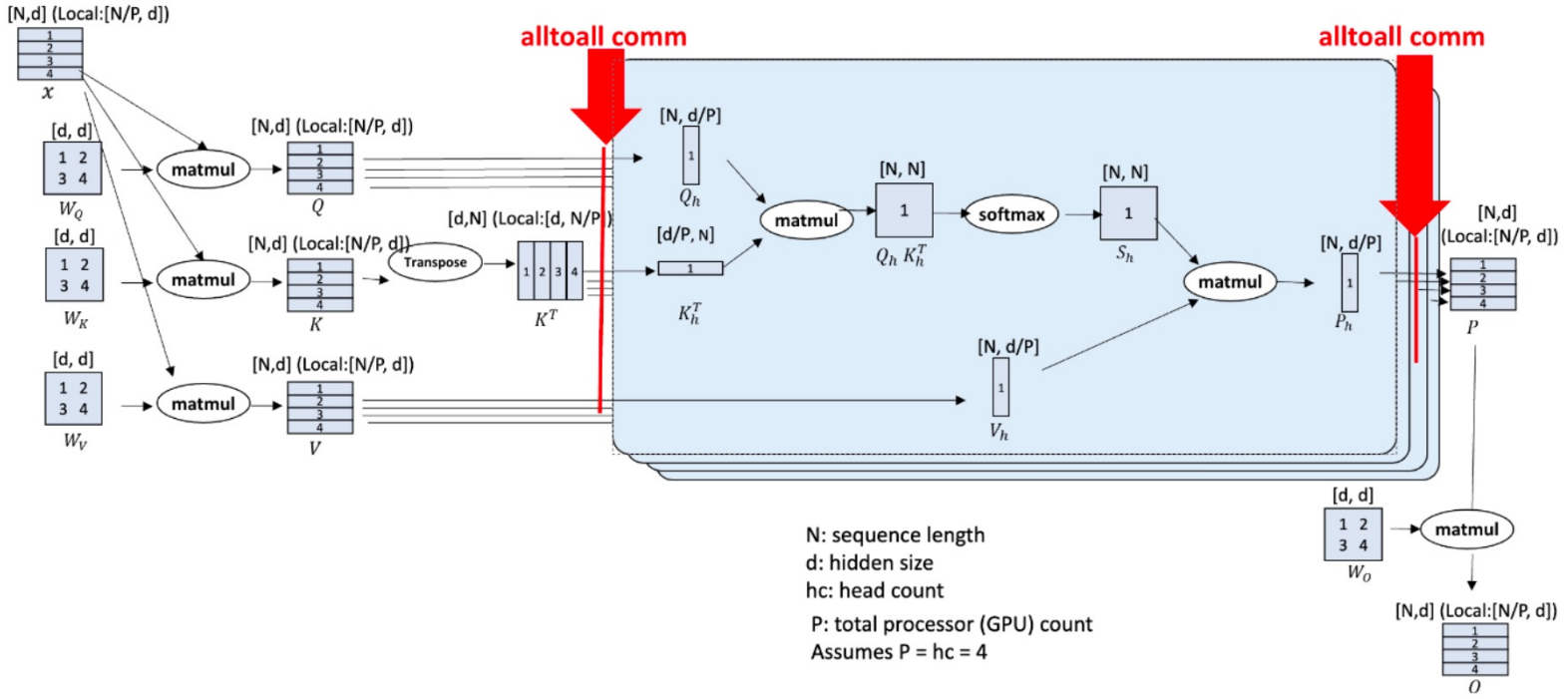
In tensor model parallelism combined with sequence parallelis:
g is **AllGather** operation in the forward pass and **ReduceScatter** in the backward pass.
$\bar{g}$ is **ReduceScatter** in the forward pass and **AllGather** operation in the backward pass.

- Sequence parallelism is an extension of the previous tensor model parallelism:
- Partitioning along the sequence dimension reduces the memory required for the activations. This extra level of parallelism introduces new collective communication paradigm.

Deepspeed-Ulysses

RELAXED SYSTEM LAB

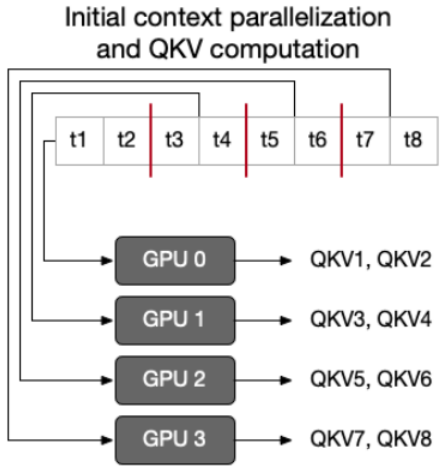
- **Core idea：**
  - Partition the computation of attention heads among different GPUs;
  - Use one All-to-All to shuffle the input Query, Key, Value;
  - Use one All-to-All to collect the corresponding output.

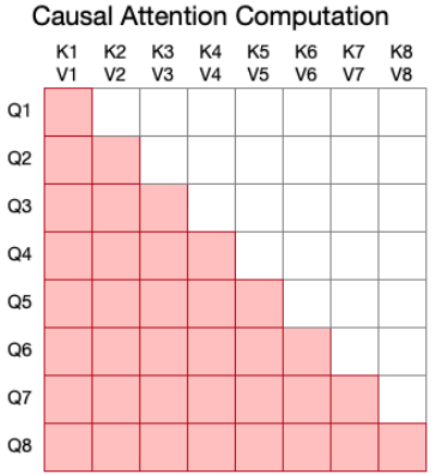


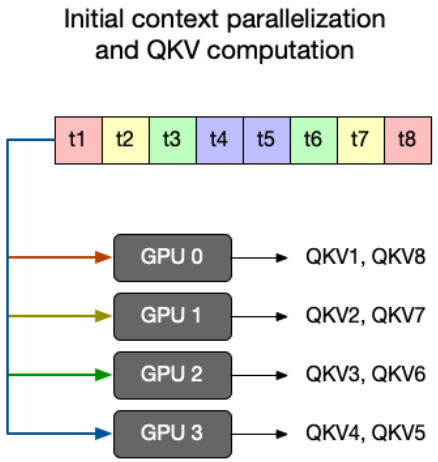
Ring Attention/Megatron Context Parallelism

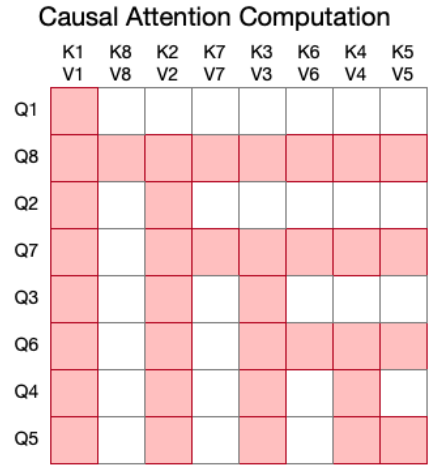
RELAXED SYSTEM LAB

- Ring attention computation for the attention mechanism:
  - Split the data by sequence dimension;
  - Iteratively accumulate local attention results by iterating Q and K/V as a nested loop;
  - Each GPU holds a portion of queries, and sends its keys and values to the next GPU and receives them from the previous GPU (ring).
- A trivial issue to be fixed in Ring Attention: the load-balance in each iteration:
  - Megatron context parallelism implementation: an optimization not to compute unnecessary masked parts to balance the workflow.









https://insujang.github.io/2024-09-20/introducing-context-parallelism

Homework2 Q2

RELAXED SYSTEM LAB

- Given a model based on stacking transformer layers, where
  - N_{layer} is the number of layers in the model;
  - B is the training batch size;
  - L is the training sequence length;
  - D is the model dimension;
  - n_H is the number of heads;
  - H is the head dimension. Note that we have $D = n_H \times H$.
- Let us ignore all the other parts of the model (e.g., EmbedToken, position embedding, etc.)
- Suppose all the computation is based on FP16 (2 bytes).

Homework2 Q2 Review

RELAXED SYSTEM LAB

- What are the communication paradigms (three key aspects)?
  - Communication targets;
  - (Collective) Communication operator(s);
  - Communication volume (i.e., **corresponding number of bytes of the communication targets**).
- Data parallelism:
  - Target: gradient of parameter;
  - Op: **AllReduce**.
- Pipeline parallelism:
  - Target: activation in forward, gradient of activation in backward;
  - Op: **Send-Recv**.
- Tensor model parallelism:
  - Target: activation in forward, gradient of activation in backward;
  - Op: **AllReduce**.
- FSDP:
  - Target: parameters in forward, parameters and gradient of parameters in backward
  - Op: **AllGather**, **ReduceScatter**.

Homework2 Q2 Data Parallelism

RELAXED SYSTEM LAB

- Suppose we train the model by data parallelism (using the standard synchronous, lossless communication) implemented by **AllReduce**. What is the volume (number of bytes) of the communication target that needs to be processed by the **AllReduce** operations?
  - Data parallelism communication target: **gradients of the parameter**;
  - Running **AllReduce** over all GPUs after backward communication, total bytes:
    $2 \times N_{\text{layer}} \times (D^2 + D^2 + D^2 + D^2 + 4D^2 + 4D^2) = 24 N_{\text{layer}} D^2$.

Homework2 Q2 Pipeline Parallelism

RELAXED SYSTEM LAB

- Suppose we train the model by pipeline parallelism implemented by Gpipe, where each stage handles 4 transformer layers. For a training iteration, how many bytes in total should be communicated between nearby stages i and $i+1$? Specify the peer-to-peer communication direction in your answer.
  - Pipeline parallelism p2p communicates the **activations** in the forward pass; and p2p communicates the **gradients of the activations** in the backward pass.
  - In the forward pass: the GPU handling stage i sends the GPU handling stage $i+1$ the activations with $2BLD$ bytes in total.
  - In the backward pass: the GPU handling stage $i+1$ sends the GPU handling stage i the gradient of the activations with $2BLD$ bytes in total.

Homework2 Q2 Tensor Model Parallelism

RELAXED SYSTEM LAB

- Suppose we train the model by tensor model parallelism (where the tensor parallel degree is D_{tp}). For a training iteration, how many bytes in total should be aggregated through the **AllReduce** operations?
- Tensor model parallelism aggregates the **activations** in the forward pass by **AllReduce**; and aggregates the **gradients of the activations** in the backward pass by **AllReduce**.
  - In the forward pass, each transformer block:
    - One **AllReduce** for Multihead-Attention $2BLD$ bytes.
    - One **AllReduce** for MLP $2BLD$ bytes.
  - In the backward, pass each transformer block:
    - One **AllReduce** for Multihead-Attention $2BLD$ bytes.
    - One **AllReduce** for MLP $2BLD$ bytes.
  - In total for one iteration: $N_{\text{layer}} \times 4 \times 2BLD = 8\, N_{\text{layer}} BLD$.

Homework2 Q2 Optimizer Parallelism

RELAXED SYSTEM LAB

- Suppose we are training the model by fully sharded data parallelism. What is the volume (number of bytes) of the communication target that needs to be processed by the **AllGather** and **ReduceScatter** operations respectively?
  - Fully sharded data parallelism communication target:
    - **AllGather** the **model parameters** in the forward pass;
    - **AllGather** the **model parameters** and **ReduceScatter** the **gradients of the parameter** in the backward pass;
  - In total, the **AllGather** APIs process bytes: $2\times2\times N_{\text{layer}}\times(D^2+D^2+D^2+D^2+4D^2+4D^2)=48N_{\text{layer}}D^2$.
  - In total, the **ReduceScatter** APIs process bytes: $2\times N_{\text{layer}}\times(D^2+D^2+D^2+D^2+4D^2+4D^2)=24N_{\text{layer}}D^2$.

RELAXED
SYSTEM LAB

Good Luck!