High Performance LLMs From First Principles (2024)

Session 7
https://github.com/rwitten/HighPerfLLMs2024

rwitten@

Goal: learn how to achieve **high performance** for LLMs

This week:

- Finish up end-to-end Training Performance
- Discuss Inference Performance

Program (write code in Jax)

Predict (roofline on napkin or spreadsheet)

Profile (run code, compare to predictions)

My Asks

Please ask lots of questions! Just raise your hand or speak up!

If there are topics you're interested in, message me between sessions.

Join the discord! https://discord.gg/2AWcVatVAw

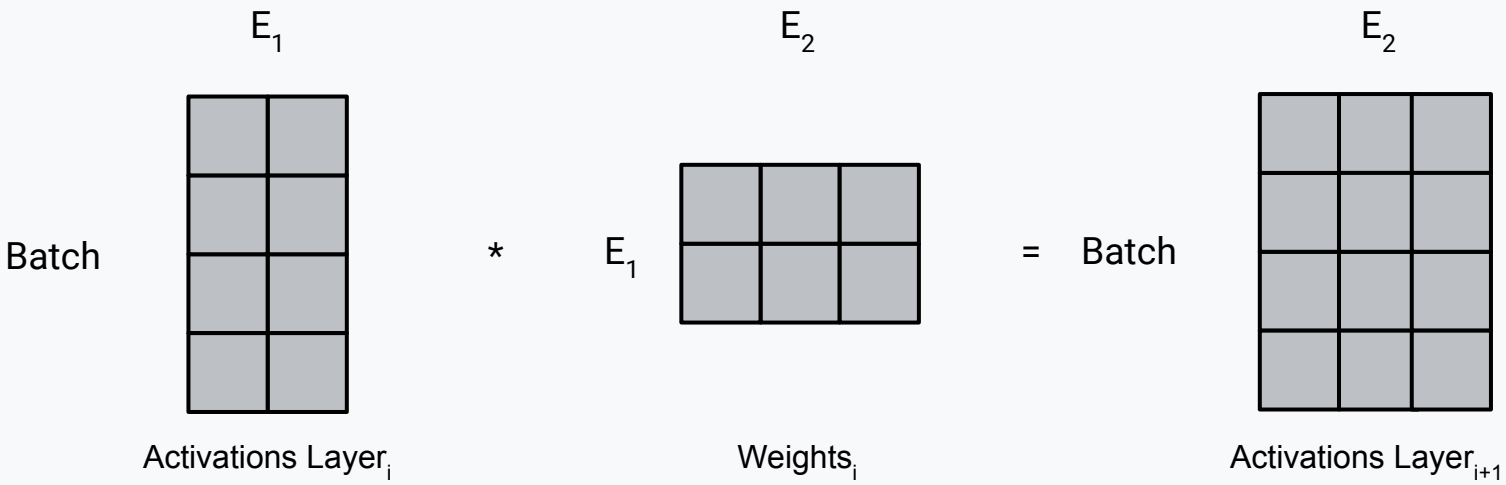
Do the exercises! Give feedback, ask questions!

Website: https://github.com/rwitten/HighPerfLLMs2024

How long should training take?

- We discussed the forwards pass in the past.
- Training adds the backwards pass! Backwards pass is computing the derivatives!

Forwards Pass

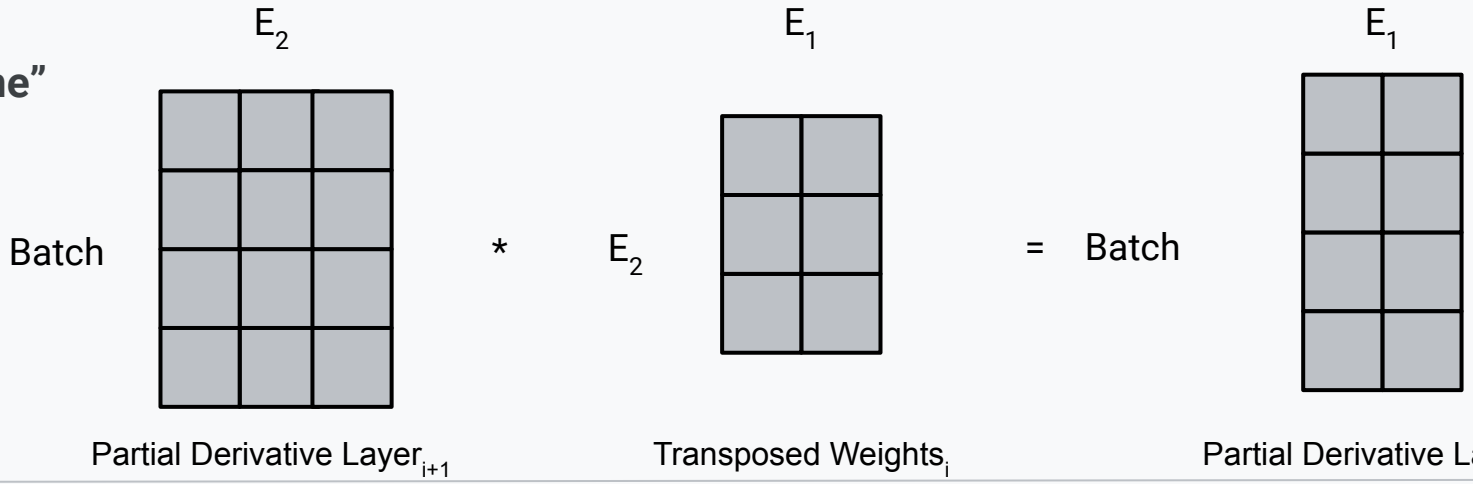


How long should training take? (forwards pass)

- We discussed the forwards pass in the past.
- All the matmuls take $2*B*E_1*E_2$ flops since they all have a B dimension, an E_1 and an E_2 dimensions!
- Conveniently:
  - Parameters = E_1*E_2
  - So this matmul during forwards pass takes $2*B*P$ total flops.

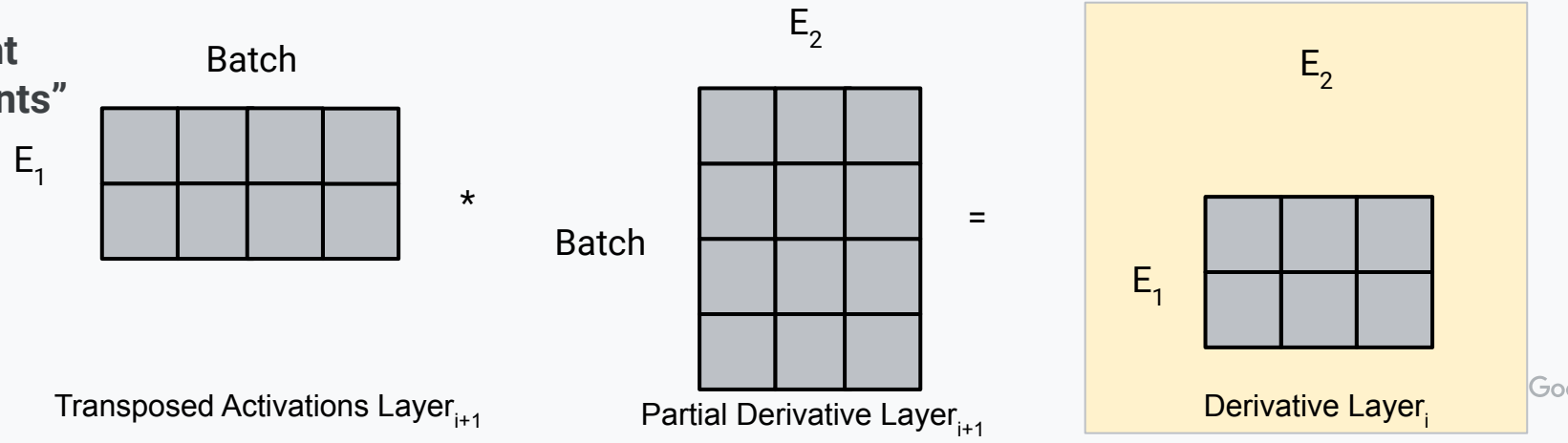
Corresponding Backwards Pass

"Spine"



Partial Derivative Layer$_{i+1}$ | Transposed Weights$_i$ | Partial Derivative Layer$_i$

"Weight Gradients"



Transposed Activations Layer$_{i+1}$ | Partial Derivative Layer$_{i+1}$ | Derivative Layer$_i$

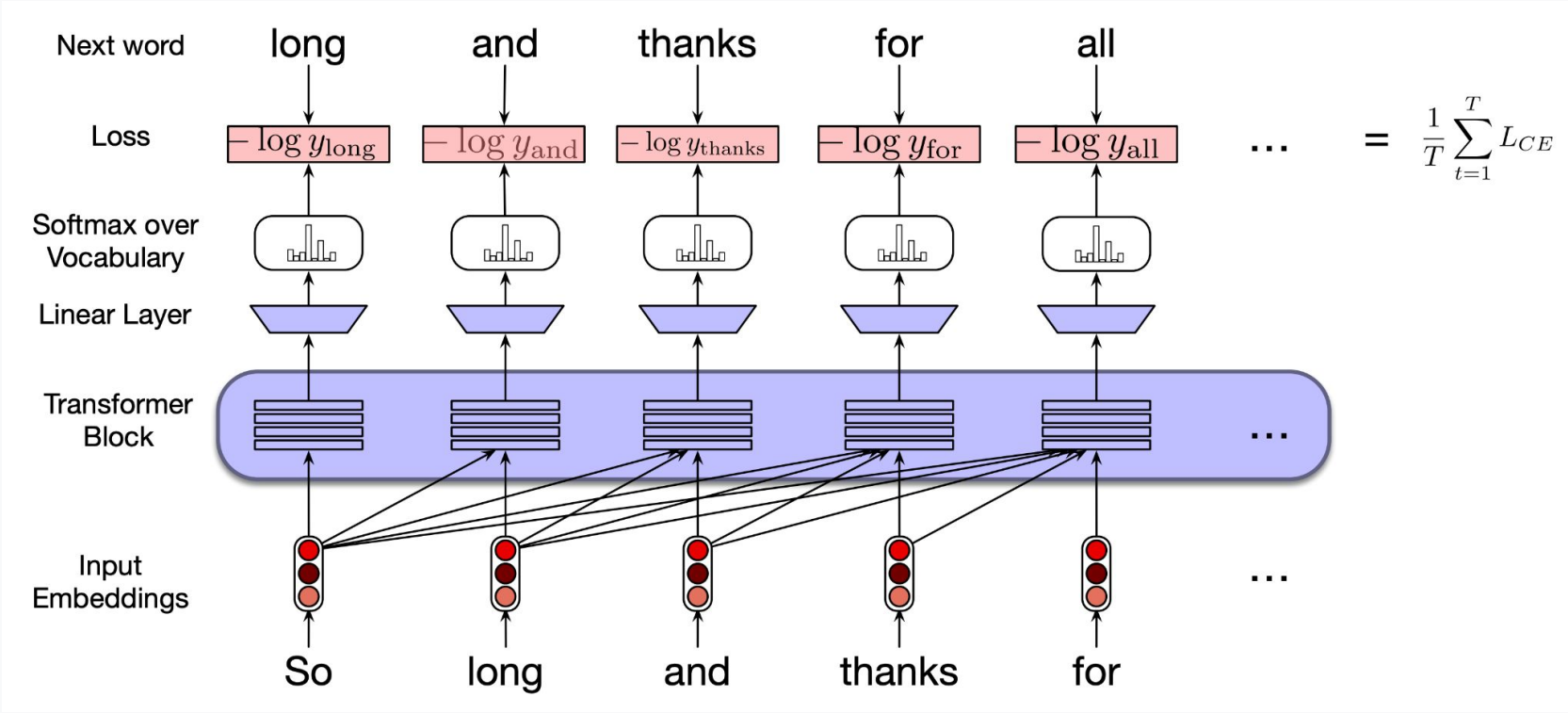
How long should training take?

- We discussed the forwards pass in the past.
- Training adds the backwards pass! Backwards pass is computing the derivatives!
- For every matmul in the forwards pass, two matmuls in the backwards pass:
  - "Spine" – generate the new partial derivations
  - "Weight gradients" – output the gradients we need!
  - All the matmuls take $2*B*E_1*E_2$ flops since they all have a B dimension, an E_1 and an E_2 dimensions! So $6*B*E_1*E_2$ total flops.
- Conveniently:
  - Parameters = E_1*E_2
  - So this matmul during training takes $6*B*P$ total flops.
  - And that is true for all matmuls – so the number of flops from matmuls is $6*B*P$!
- Warning: this calculation ignores attention flops! Attention flops are typically a small fraction of the overall flops...

Implementation

- Remember our handy flags:
  - export LIBTPU_INIT_ARGS="--xla_enable_async_all_gather=true TPU_MEGACORE=MEGACORE_DENSE"
- Count Parameters
- 6*B*P
- Xprof: https://xprof.corp.google.com/trace_viewer/rwitten-6098379516334927l7?host_index=0
- jax.lax.with_sharding_constraint

LLM Overview (Session 1)



- Speech and Language Processing. Daniel Jurafsky & James H. Martin. Copyright © 2023. All rights reserved. Draft of January 7, 2023.

What is Our Setting For Inference?

- LLM:
  - Receives as input: “So long and thanks for”
  - It needs to output: “all the fish”
- If we’ve seen the input “So long and thanks for”, where do we use that information during next token generation?
  - Just in Attention!
  - In Attention, this is called the KV cache (key value cache).
  - So the KV cache is [Batch, PreviousTokens, NumberHeads, HeadDimension]
- When processing a new token, we are able to do the matrix multiplies just on the new data and the attention requires a read/write from the KV cache.
- But critically we CANNOT process “all the fish” together – the neural network needs to choose the tokens one at a time!

What is Our Setting For Inference?

- LLM:
  - Receives as input: “So long and thanks for”
  - It needs to output: “all the fish”
- First prefill:
  - Prepare KV cache for “So long and thanks for”
  - Suggest next token – “all”
- Then generate:
  - If “all” is the next token, run “all” through the NN, writing to the KV cache and output “the” next.
- So this computation splits into:
  - 1 prefill
  - N generates, 1 per output token

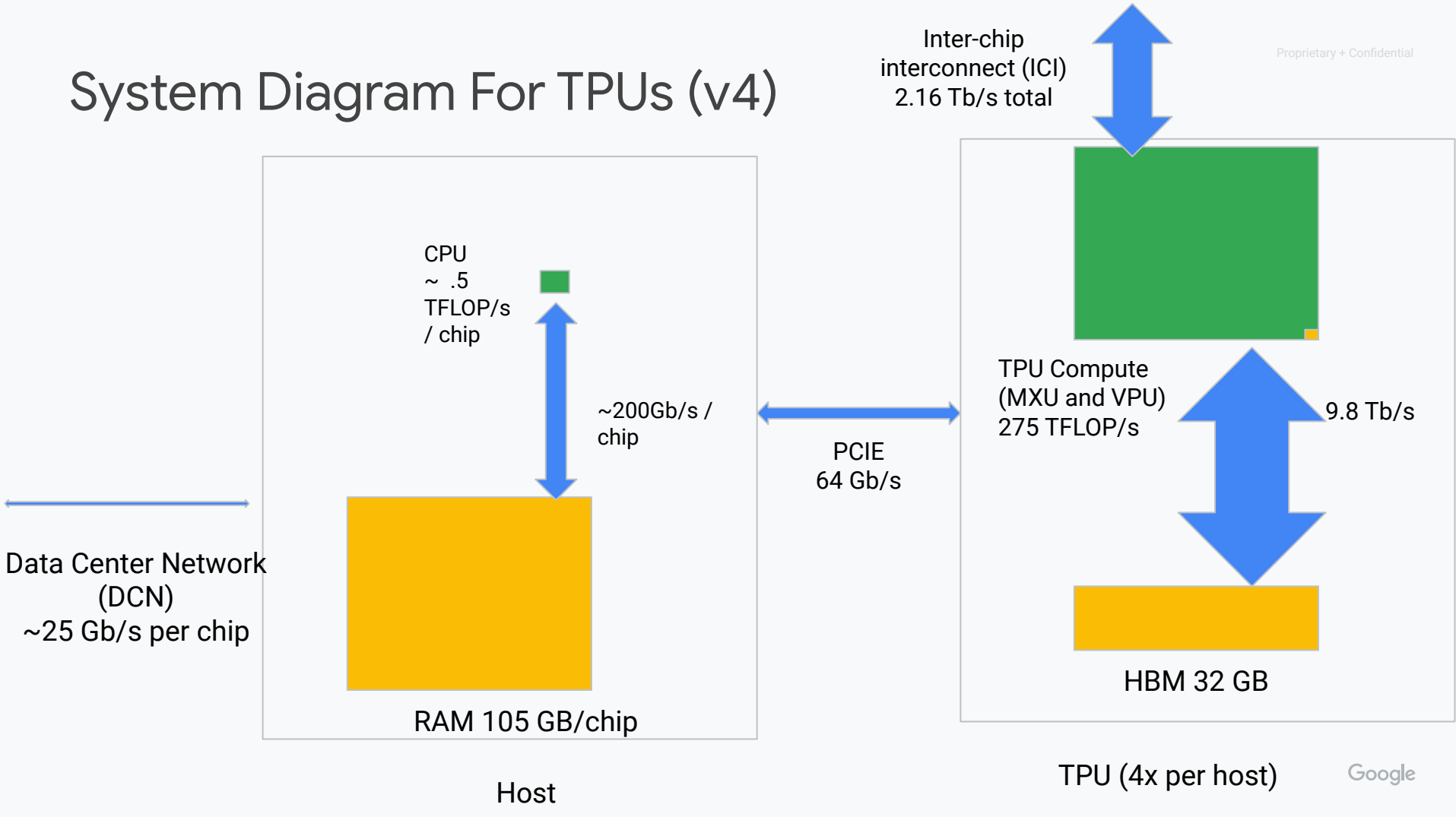
How Long Does Generate Take (Matrix Multiplies)

- Flops = 2 * BatchInTokens * Parameters
- Memory for loading weights (bytes) = 2 * Parameters
- If BatchInTokens < HBM arithmetic intensity, we're memory bound
    - Recall HBM Arithmetic instead is 224 Flops/byte on v4
- But ... we need to process only one token at a time.
- Most common solution: process many sequences at once! So instead of just processing one customer's query, process 224 queries at once!

How Long Does Generate Take (Attention)

- We need to read through a key and value KV cache of [Batch, PreviousTokens, NumHeads, HeadDim] by NumLayers
    - For Gemma7B: num heads is 16, headdim is 256, and we might want 2K tokens of history and num layers is 28
        - ~939 MB per sequence!
- This attention op is memory bandwidth bound!

System Diagram For TPUs (v4)



How Can We Shard The Generate Step?

- Typically we can shard the generate steps via tensor parallelism.
- (Fully-sharded data parallelism doesn't help – instead of loading the full weights from HBM we load them over inter-chip-interconnect which is slower!)
- Recall that for tensor-parallelism we move activations over ICI – which is good during inference since the activations are small.
    - And we shard the KV cache over the batch dimension.
- So full path:
    - [B,1,E\T] -> [B,1,F\T] (collective matmul)
    - [B,1,F\T] -> [B,1,E\T] (collective matmul)
    - [B,1,E\T] -> 3x[B,1,H\T, D] (collective matmul)
    - 3x[B,1,H\T, D] and [B, S, H\T, D] -> [B, 1 H\T, D] (attention and purely local)
    - [B, 1 H\T, D] -> [B,1,E\T]

Typical Overall Inference Story

- Prefill:
  - Takes 2*BATCH*PREFILL_SEQUENCE*P Flops (and is flop bound)
- Generate
  - Takes 2*P bytes to load weights and 2*BATCH*TOTAL_LENGTH*NUM_HEADS*HEAD_DIM*LAYERS*2 bytes to load weights
- Typical numbers (Gemma 7B)
  - ~7B params so 14GB of weights
  - We already calculated .94GB per example. V4-8 has 32GB*4 = 128GB total
    - So we can have (128GB-14GB)/.94GB = 121 batch size in theory
    - In practice, we need to have some margin for error, so maybe batch = 110 for 103GB of KV cache.
  - So each step requires loading 117GB to generate 110 new tokens!
  - (By growing the number of examples, we're amortizing down loading the weights)

Bigger Inference Story

- That was sequence 2048! Nowadays we want to do sequence 1,000,000 – that'll require 500x more memory!
- Many important ideas, but one is Grouped Query Attention. Instead of having 1 K and V head for every Q head, we can have fewer K and V heads and X queries share the same K and V.
  - This helps inference speed by shrinking KV cache, but does it converge as well?
- Quantization!
  - We're super memory bound, could we represent the weights and KV cache in int8?
- Speculative Decoding:
  - What if we thought we knew the next few tokens? Then we could "speculate". But our speculation might be wrong in which case the work is wasted! (But it was wasted anyways.)

Next Week

- Inference Implementation!
- Look at a good Xprof!

Thanks!
Ping me (rwitten@google.com) with feedback, suggested topics, etc!