The growth of parallelism in machine learning inference

Tim Harris

2024-06-18

The growth of parallelism in machine learning inference

U Cambridge computer lab

Microsoft Research

Oracle Labs

Amazon S3

Microsoft ~~Research~~

The growth of parallelism in machine learning inference

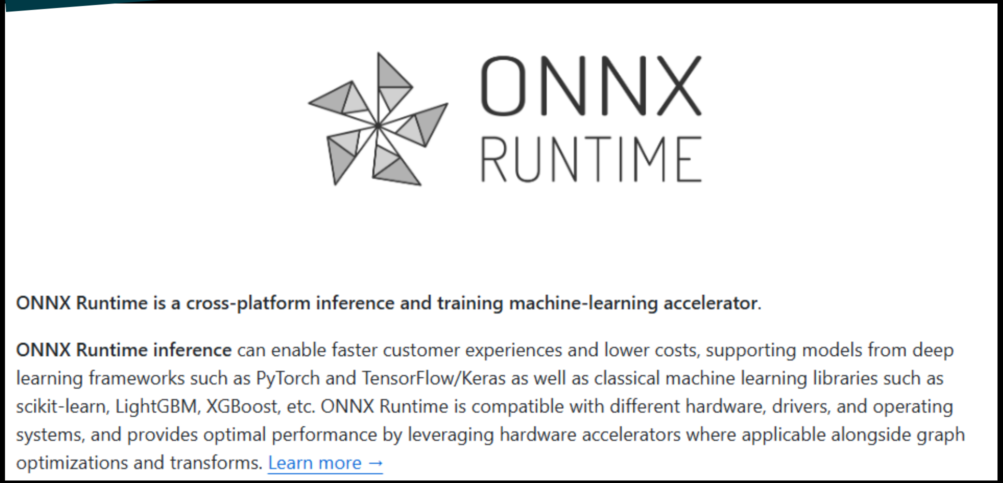
U Cambridge computer lab

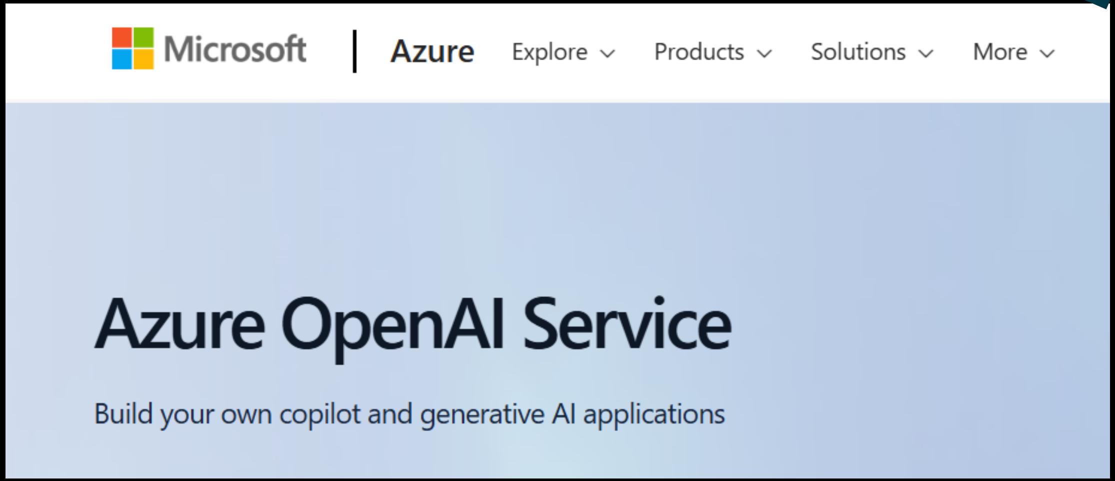
Microsoft Research

Oracle Labs

Amazon S3

Microsoft ~~Research~~





Language model growth – a few examples

BERT

Base – 110M
Large – 340M

GPT-3

Ada – 350M
Babbage – 1.3B
Curie – 6.7B
Davinci – 175B

PaLM

540 B

2018

2020

2022

~1600x over 4 years
~6x annually

Language model growth – a few examples

BERT

Base – 110M
Large – 340M

GPT-3

Ada – 350M
Babbage – 1.3B
Curie – 6.7B
Davinci – 175B

PaLM

540 B

2018

Nvidia V100
16-32 GB HBM
900 GB/s b/w
112 TFLOP/s

2020

Nvidia A100
40-80 GB HBM
~2 TB/s b/w
312 FP16 TFLOP/s

2022

Nvidia H100
80 GB HBM
~3.35 TB/s b/w
~990 FP16 TFLOP/s
~1979 FP8 TFLOP/s

~1600x over 4 years
~6x annually

Specifications from NVIDIA A100 | Tensor Core GPU H100 Tensor Core GPU | NVIDIA, not including impact of sparsity

Overview

Introduction

Generating text with LLMs

Parallelism in LLM inference

Generating text with LLMs

Input prompt
"Tell me a rhyming poem about a tortoise and a friendly dinosaur"

```
Possible rhyming poem:

A dinosaur named Rex was feeling rather bored
He roamed the land in search of something new to explore
He stomped and snorted, making quite a lot of noise
But nothing seemed to catch his eye or give him any joy

He wandered to a river where he saw a shiny shell
He thought it was a rock or maybe some kind of jewel
He picked it up with curiosity and gave a gentle poke
But to his great surprise, the shell began to move and spoke

"Hello there, Mr. Dinosaur, you're quite a mighty beast
But please don't hurt me, I'm a tortoise, and I want to live in peace
I know I'm slow and small, but I have wisdom and a heart
And maybe we can be friends, if you're willing to do your part"
```

- A toy example with a few words
- Real examples may use a metaprompt before the user's input to provide guidance on the kind of output to generate
- Maximum prompt sizes continue to grow -- GPT-4 Turbo | OpenAI Help Center announced support for prompts with the equivalent of 300 pages of text

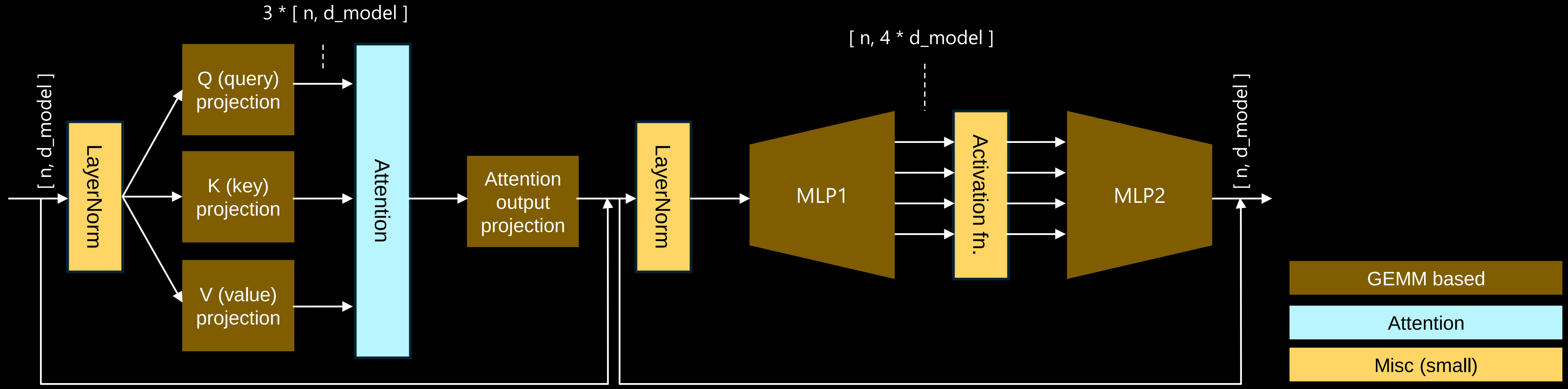
Key ideas in the computation

Input and output are tokens (few characters, short words, …) – vocabulary size ~50K+	An embedding layer maps each token to a vector of based of size d_model (model dimension), e.g., 4096 for Llama2	Autoregressive decoder only LLMs generate a new vector of size d_model for token N+1 from the tokens 0..N. i.e., incrementally, one token at a time	An unembedding layer maps the vector d_model back to a token value
int[input_size]	float[input_size, d_model]	float[d_model]	int

The small print: Quantization to smaller data types in place of float is an active research topic, as are different sampling algorithms that generate and pick between a small number of possible next tokens. Embedding can be extended with techniques like RoPE to encode position information about tokens.

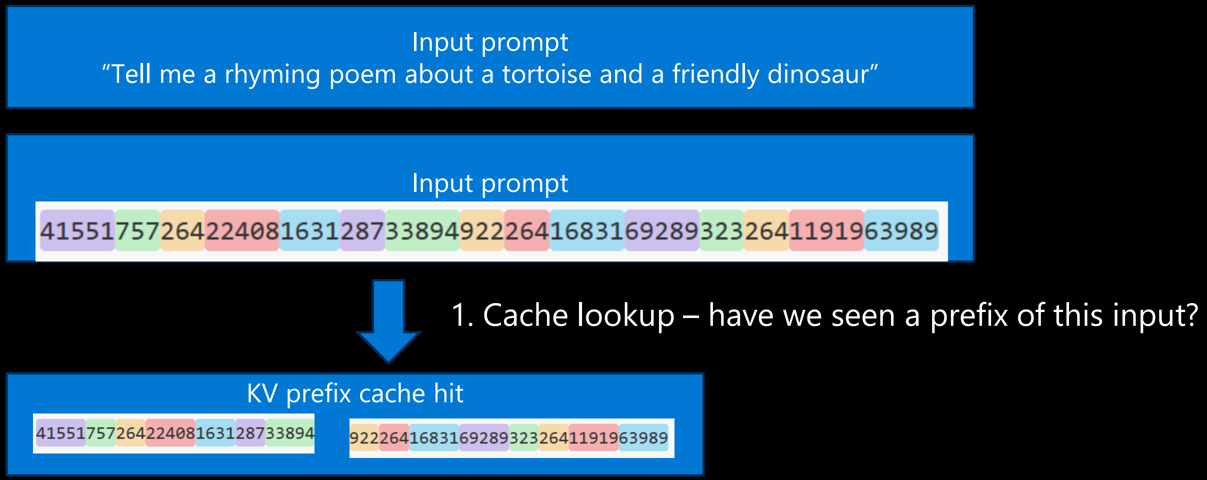
Generating text with LLMs

Looking into an LLM decoder layer – repeated many times, e.g. 96 in GPT-3 175B

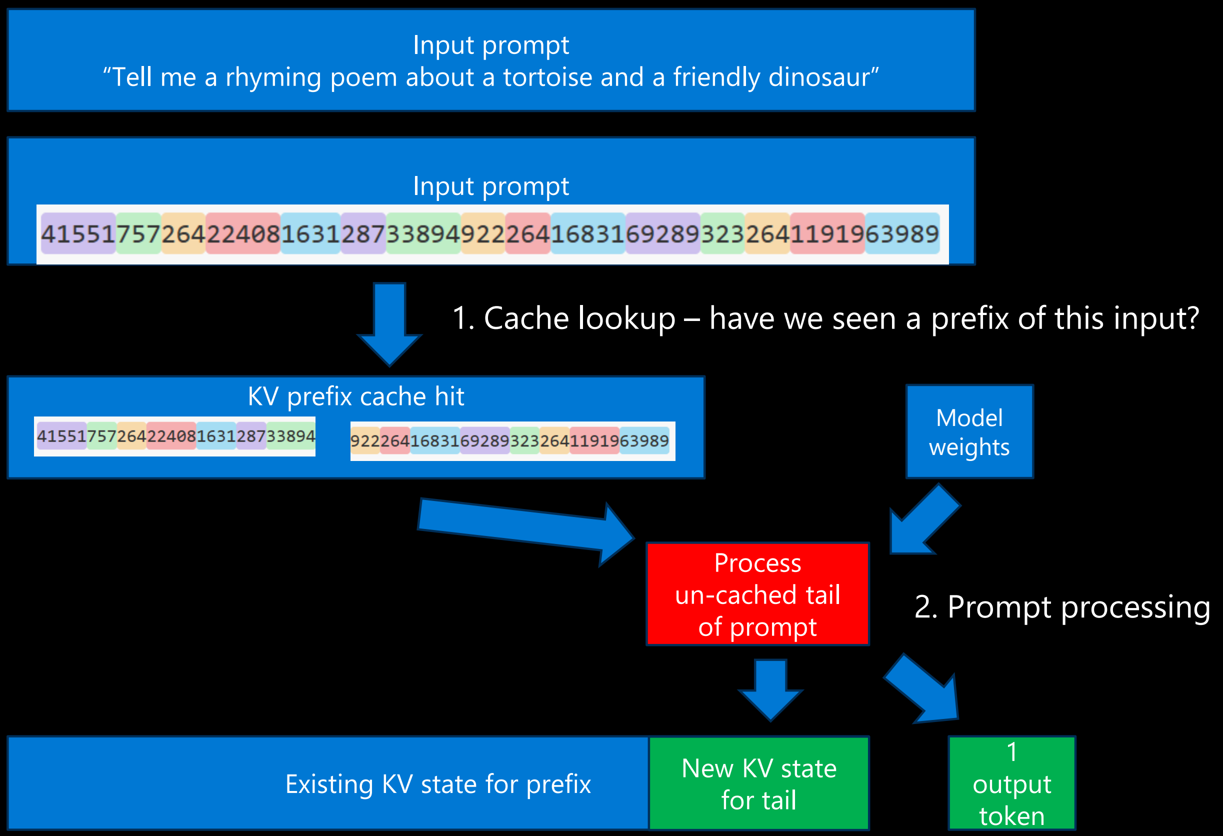


- Except for attention:
  - All operators parallelize across tokens, most of these are based on matrix multiply
- Attention:
  - Based on Q K V projections looking back through prior tokens
  - Can cache and re-use state from tokens 0..N when computing attention to generate token N+1

Generating text with LLMs

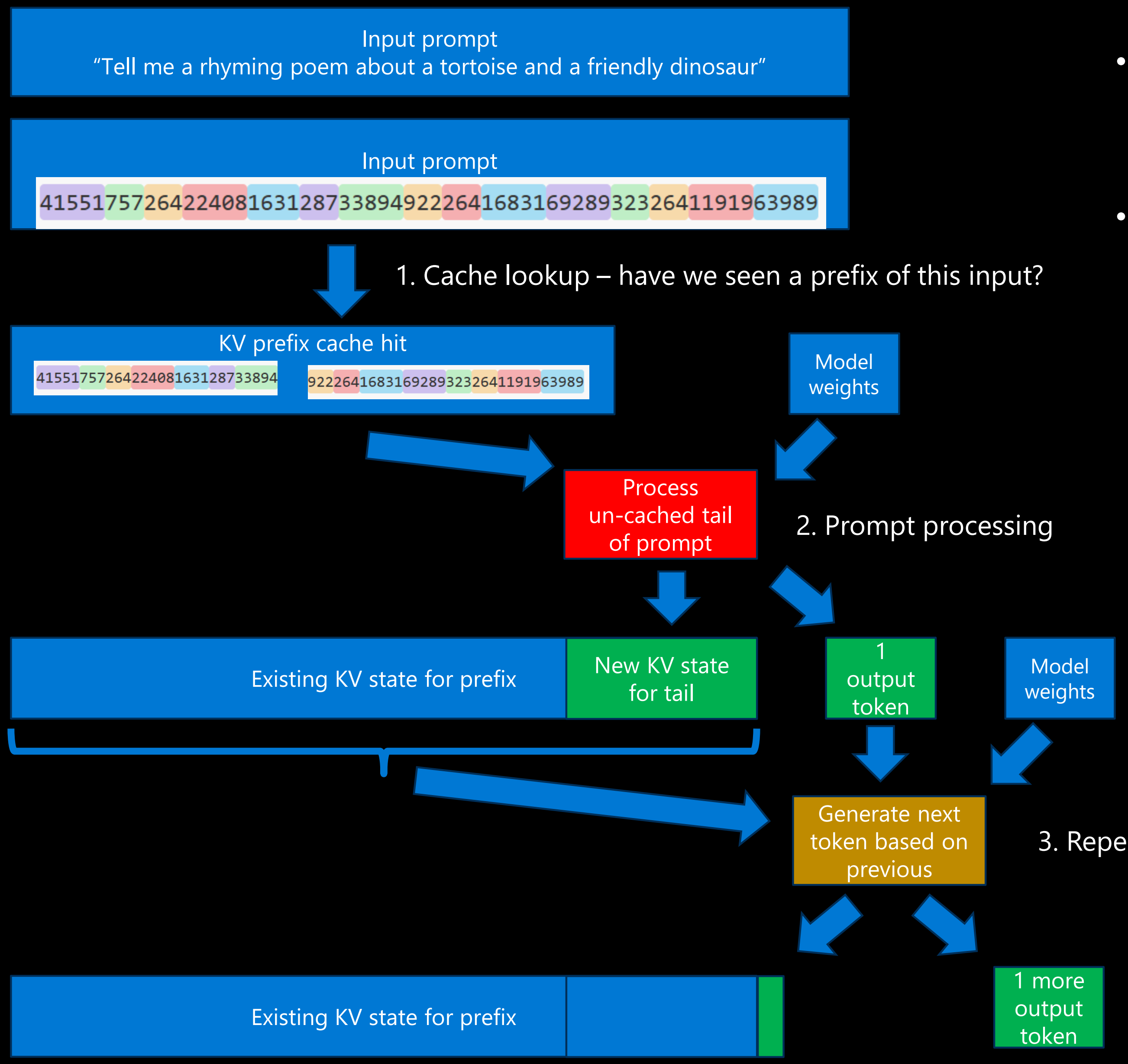


Generating text with LLMs



- Prompt processing (red) is processing the whole input prompt => compute intensive, expect to be close to achievable h/w capabilities

Generating text with LLMs



- Prompt processing (red) is processing the whole input prompt => compute intensive, expect to be close to achievable h/w capabilities
- Sampling (yellow), is bottlenecked on memory b/w: for every token we generate we must load the request’s cached KV state and model weights. Longer contexts => more KV state
- The more tokens we generate, the more significant the bandwidth b/w bound is for overall performance

```
Possible rhyming poem:

A dinosaur named Rex was feeling rather bored
He roamed the land in search of something new to explore
He stomped and snorted, making quite a lot of noise
But nothing seemed to catch his eye or give him any joy

He wandered to a river where he saw a shiny shell
He thought it was a rock or maybe some kind of jewel
He picked it up with curiosity and gave a gentle poke
But to his great surprise, the shell began to move and spoke

"Hello there, Mr. Dinosaur, you're quite a mighty beast
But please don't hurt me, I'm a tortoise, and I want to live in peace
I know I'm slow and small, but I have wisdom and a heart
And maybe we can be friends, if you're willing to do your part"
```

Overview

Introduction

Generating text with LLMs

Parallelism in LLM inference

Overview

Parallelism in LLM inference

Specializing implementations

Interconnect demands
Optimizing for customer latency targets

Pipelining across multi-GPU nodes

Scheduling choices, impact of bubbles

Sharding across GPUs

Implementation of communication – access to remote GPU memory
Achieving compute / communication overlap

GEMM implementation & batching

Dividing and scheduling work
Handling varying GPU bottlenecks
Ensuring load balance within a GPU

Overview

Parallelism in LLM inference

Specializing implementations
Interconnect demands
Optimizing for customer latency targets

Pipelining across multi-GPU nodes
Scheduling choices, impact of bubbles

Sharding across GPUs
Implementation of communication – access to remote GPU memory
Achieving compute / communication overlap

GEMM implementation & batching
Dividing and scheduling work
Handling varying GPU bottlenecks
Ensuring load balance within a GPU

GEMM implementation and batching

Textbook algorithm for basic matmul

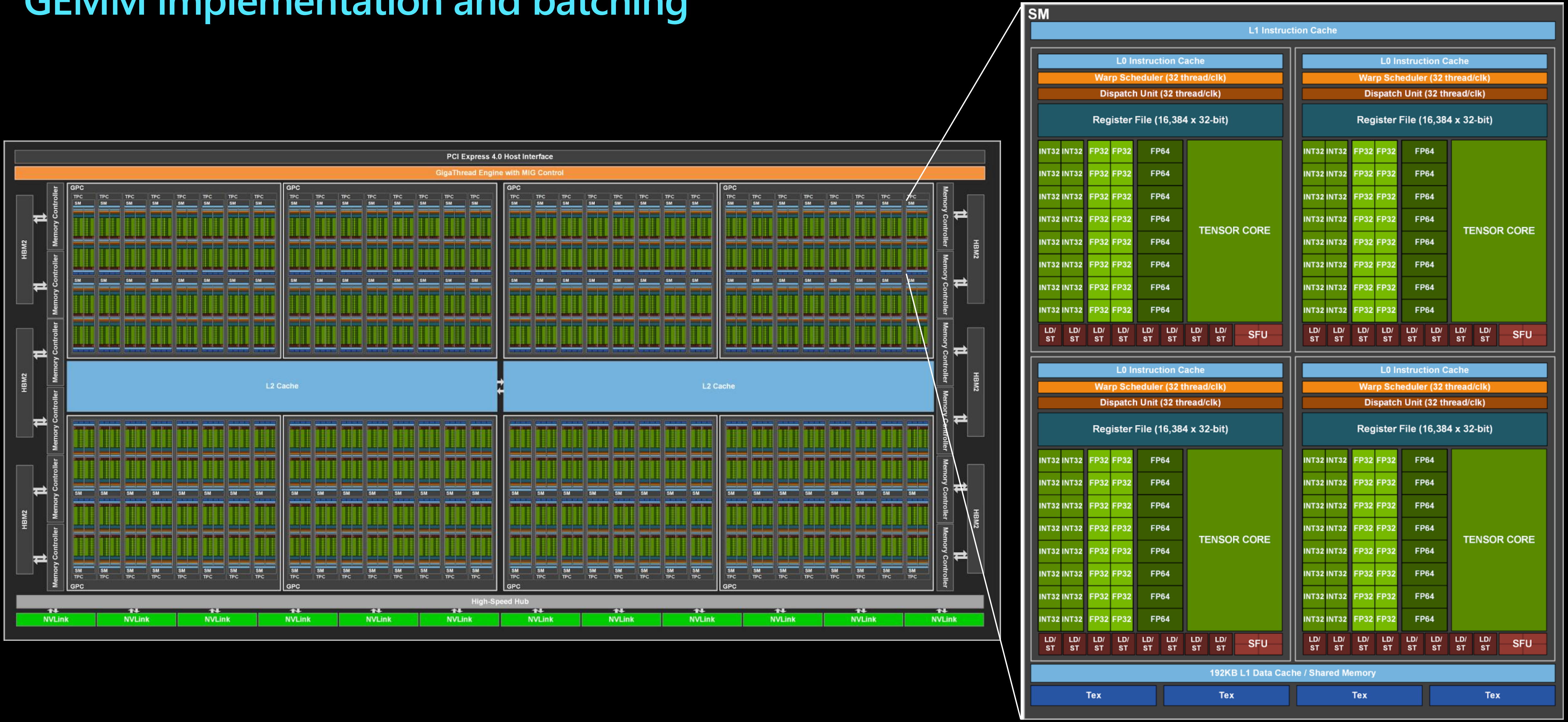
Number of tokens

Model dimension

```
// Inputs : A[M,K] * B[K,N]
// Output : C[M,N]
for m = 0..M
  for n = 0..N
    for k = 0..K
      C[m,n] += A[m,k] * B[k,n]
```

In principle we can parallelize any or all of these loops

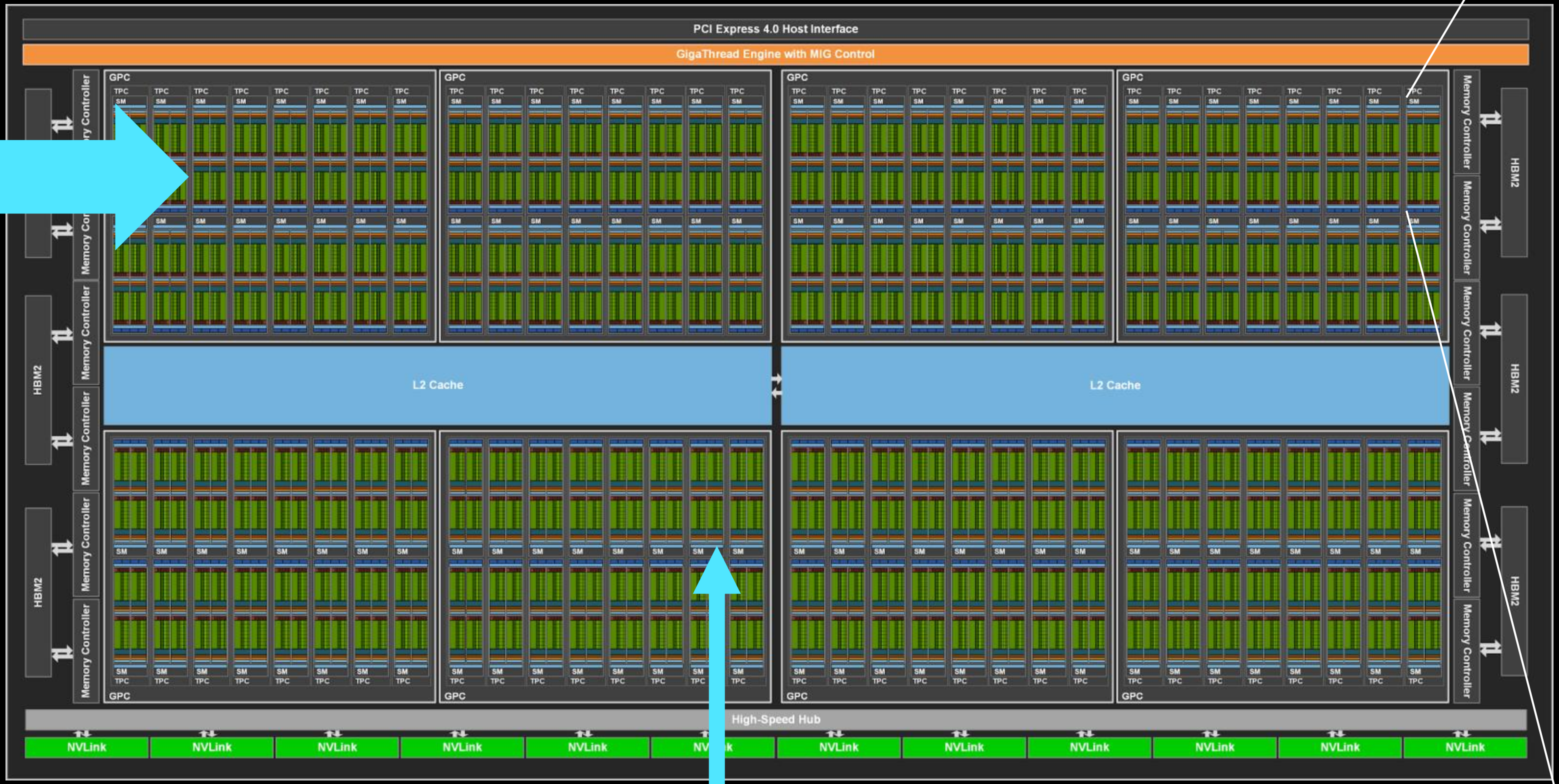
GEMM implementation and batching



https://developer.download.nvidia.com/devblogs/ga100-full-gpu-128-sms.png

GEMM implementation and batching

Matrices will be in HBM, use tiling to maximize re-use of data from faster shared memory or registers

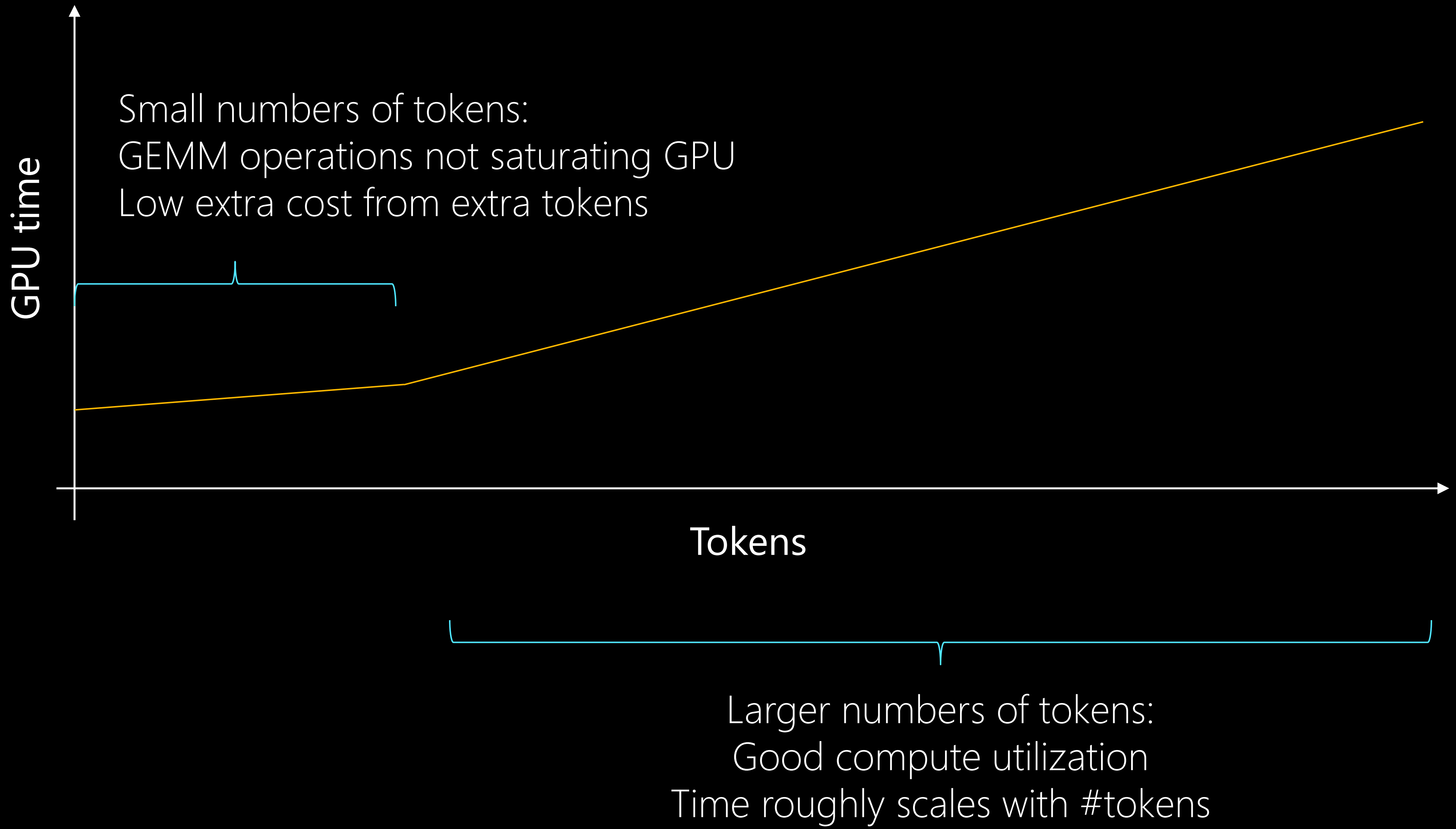


108 SMs * 4 tensor cores per SM = 432 tensor cores
...all need to be kept busy

...with work chunks 8 x 8 x 4
8 x 8 x 32
8 x 8 x 128
8 x 32 x 16
16 x 16 x 8
16 x 16 x 16
32 x 8 x 16

https://developer.download.nvidia.com/devblogs/ga100-full-gpu-128-sms.png
https://github.com/NVIDIA/cutlass?tab=readme-ov-file – implementation technique

Batching requests – GPU time vs #tokens



Batching requests – GPU time vs #tokens

Increasing and tuning batch sizes

GPU time per token becomes vastly more efficient as we go beyond tiny numbers of tokens

Token generation is serial within a request => form batches across multiple requests, specialize attention to run per-request

Divide large prompts into smaller chunks, => can generate tokens as part of each chunk

Iteration-level batching and flexible scheduling – Orca osdi22-yu.pdf (usenix.org)

Paged attention – vLLM Efficient Memory Management for Large Language Model Serving with PagedAttention (arxiv.org)

Combine sampling with prompt processing and control for target latency - SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills - Microsoft Research

Overview

Parallelism in LLM inference

Specializing implementations
Interconnect demands
Optimizing for customer latency targets

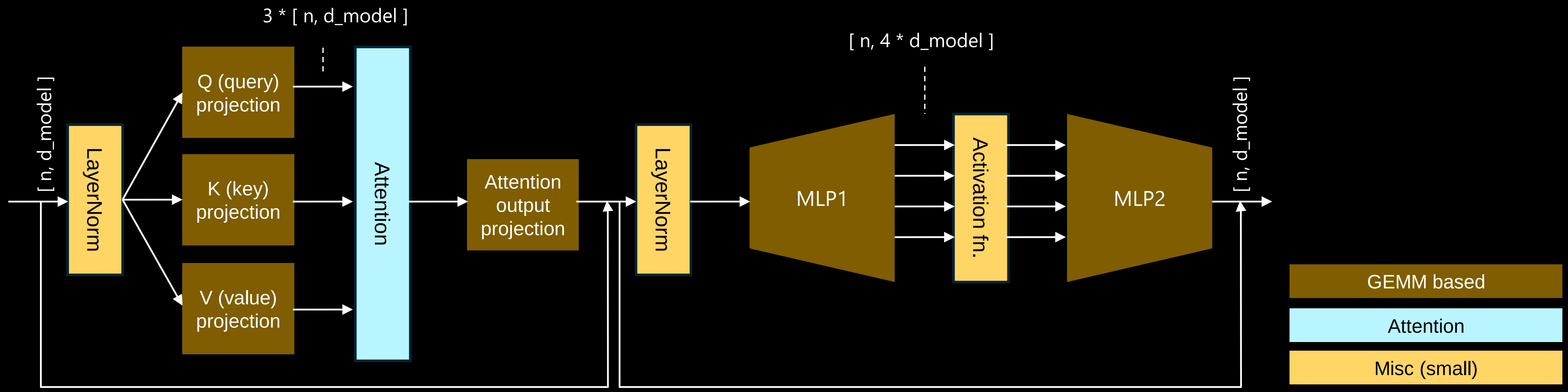
Pipelining across multi-GPU nodes
Scheduling choices, impact of bubbles

Sharding across GPUs
Implementation of communication – access to remote GPU memory
Achieving compute / communication overlap

GEMM implementation & batching
Dividing and scheduling work
Handling varying GPU bottlenecks
Ensuring load balance within a GPU

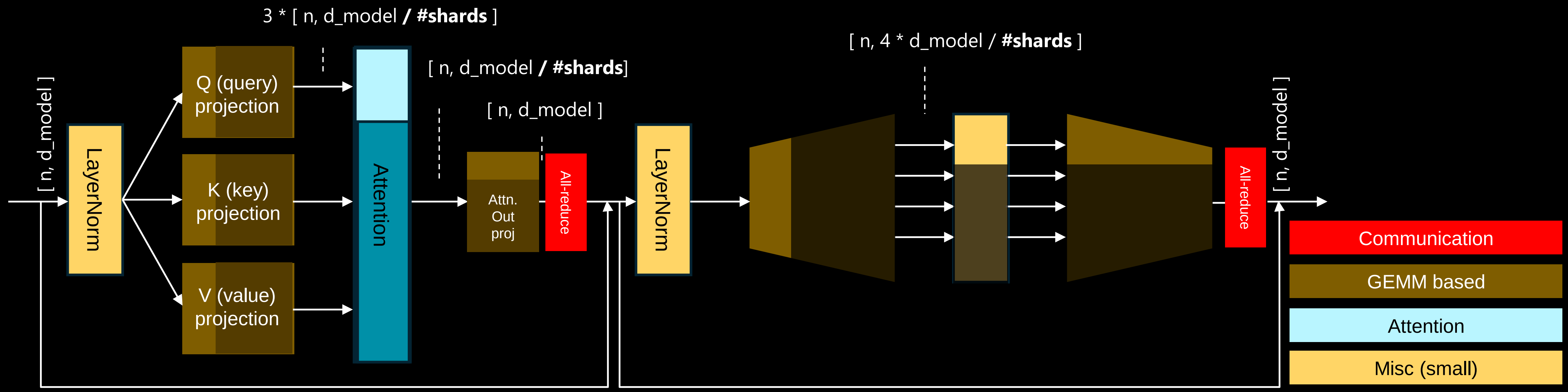
Sharding across GPUs

- Spread weights over more GPUs (more capacity for KV state)
- Run each layer faster
- Potentially form larger batches
- Approach: divide d_model between devices



Sharding across GPUs

- Spread weights over more GPUs (more capacity for KV state)
- Run each layer faster
- Potentially form larger batches
- Approach: divide d_model between devices



- Megatron-LM paper from Nvidia
- See microsoft/msccl, microsoft/mscclpp repos on github for communication techniques

Overview

Parallelism in LLM inference

Specializing implementations

Interconnect demands
Optimizing for customer latency targets

Pipelining across multi-GPU nodes

Scheduling choices, impact of bubbles

Sharding across GPUs

Implementation of communication – access to remote GPU memory
Achieving compute / communication overlap

GEMM implementation & batching

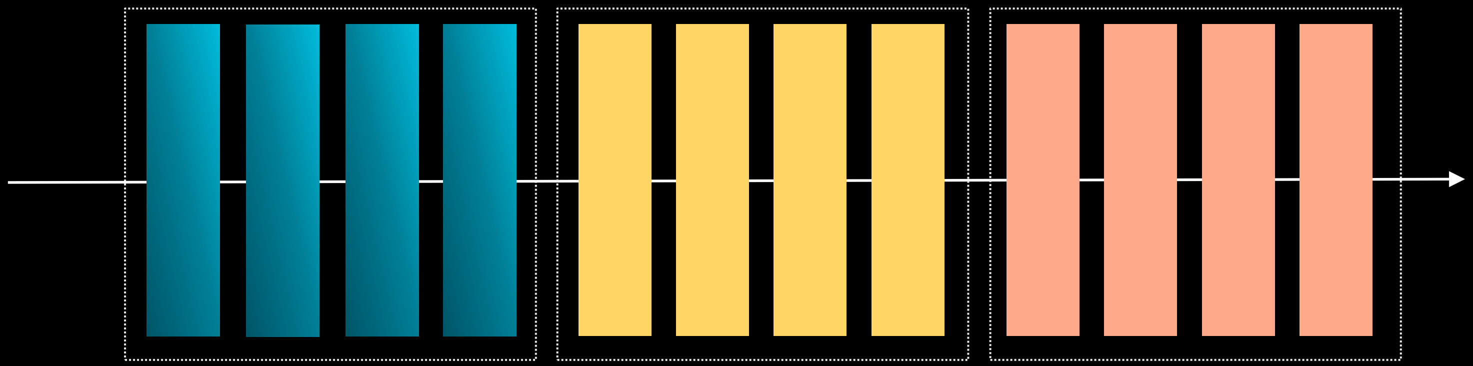
Dividing and scheduling work
Handling varying GPU bottlenecks
Ensuring load balance within a GPU

Pipelining

- Spread weights and layers over multiple GPUs or VMs
- KV state stays local to each layer

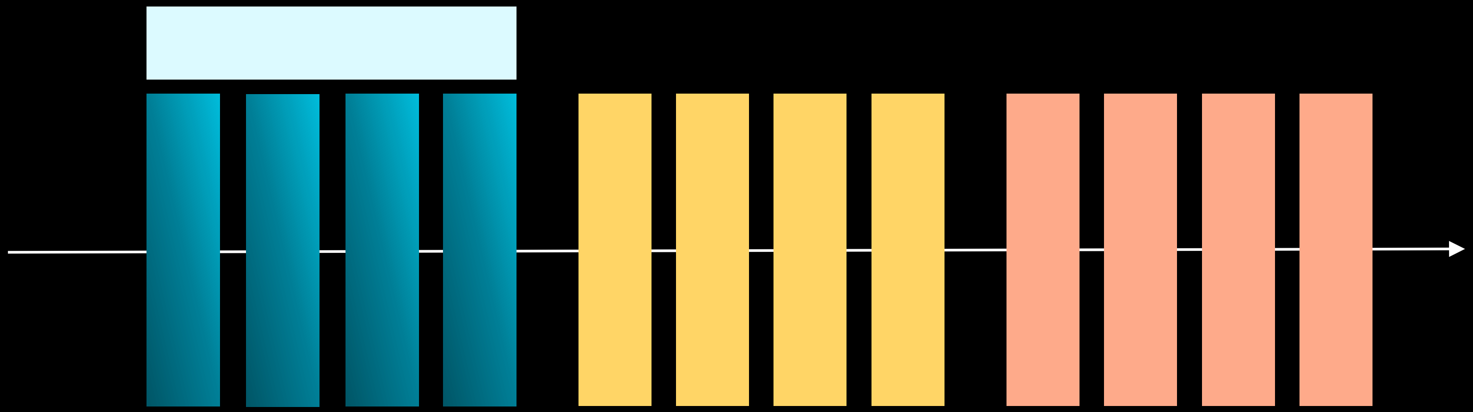
Pipelining

- Spread weights and layers over multiple GPUs or VMs
- KV state stays local to each layer



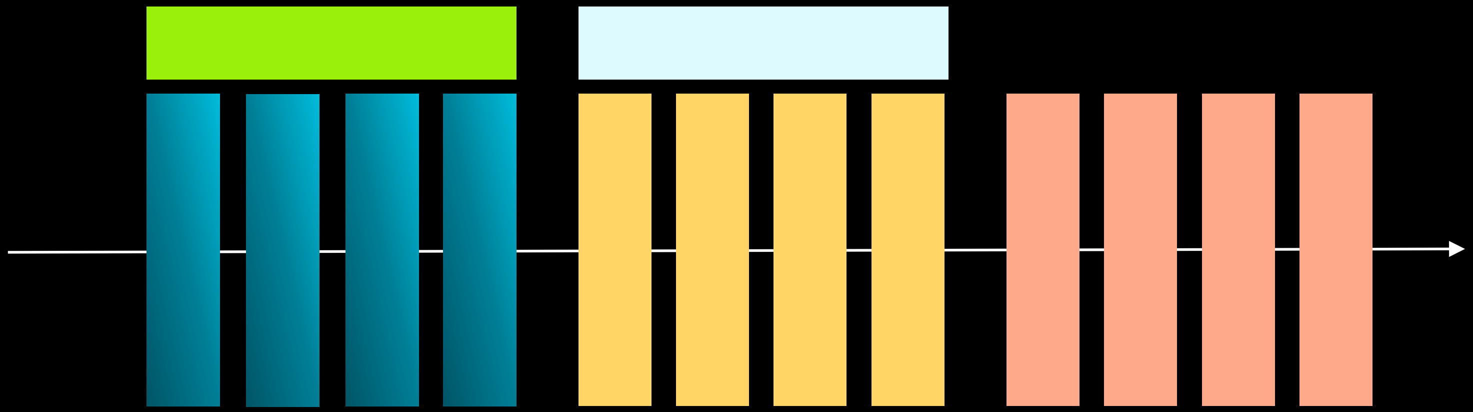
Pipelining

- Spread weights and layers over multiple GPUs or VMs & divide prompt into chunks
- KV state stays local to each layer



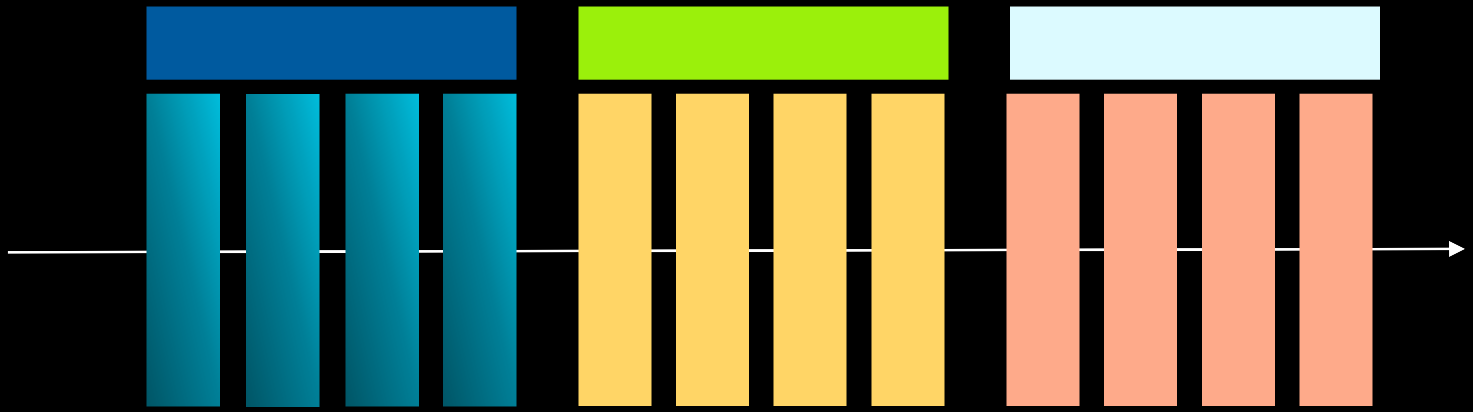
Pipelining

- Spread weights and layers over multiple GPUs or VMs & divide prompt into chunks
- KV state stays local to each layer



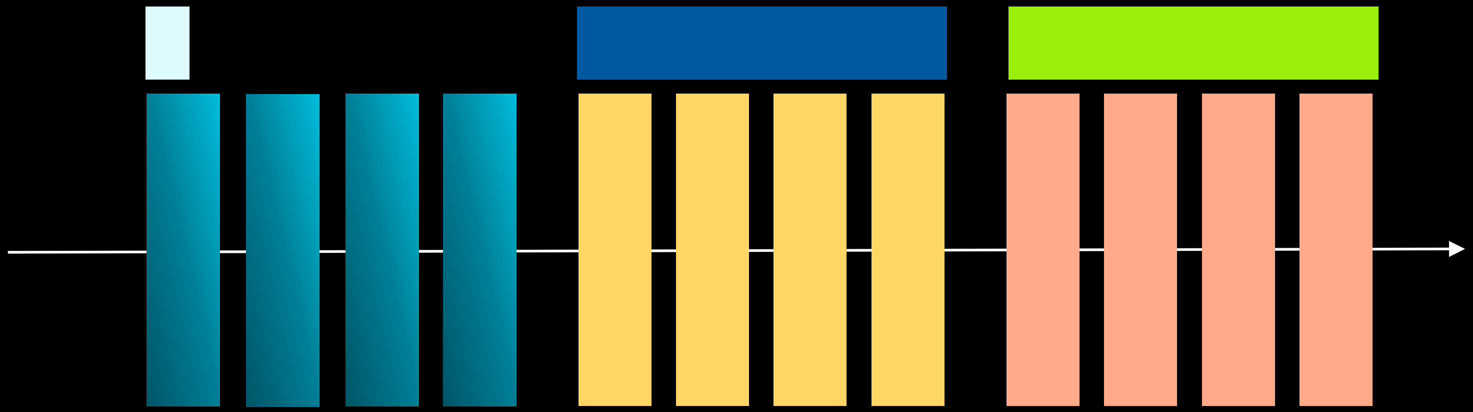
Pipelining

- Spread weights and layers over multiple GPUs or VMs & divide prompt into chunks
- KV state stays local to each layer



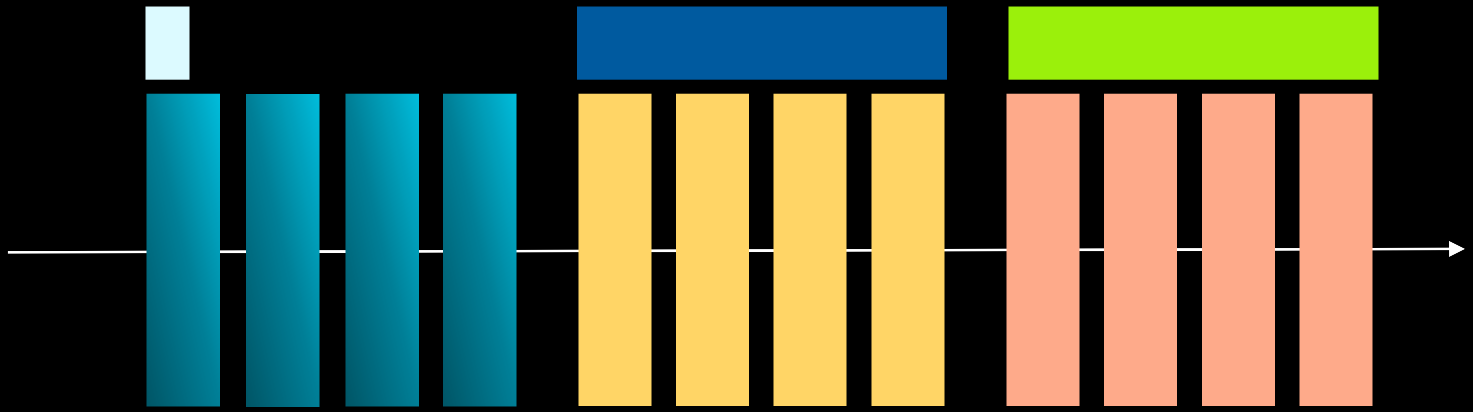
Pipelining

- Spread weights and layers over multiple GPUs or VMs & divide prompt into chunks
- KV state stays local to each layer



Pipelining

- Spread weights and layers over multiple GPUs or VMs & divide prompt into chunks
- KV state stays local to each layer



Overview

Parallelism in LLM inference

Specializing implementations
Interconnect demands
Optimizing for customer latency targets

Pipelining across multi-GPU nodes
Scheduling choices, impact of bubbles

Sharding across GPUs
Implementation of communication – access to remote GPU memory
Achieving compute / communication overlap

GEMM implementation & batching
Dividing and scheduling work
Handling varying GPU bottlenecks
Ensuring load balance within a GPU

Specialization

- Two separate implementations: optimize one for prompt, one for sampling
- Ship KV state from prompt to sampling

Prompt-optimized

- Compute-heavy
- Optimize for throughput

Sampling-optimized

- Memory b/w-heavy
- Optimize within a latency target

SplitWise https://arxiv.org/abs/2311.18677

Wrapping up

Introduction

Generating text with LLMs

Parallelism in LLM inference

Wrapping up

Introduction

Generating text with LLMs

Parallelism in LLM inference

What abstractions would let us reduce per-model work

...per-device work, e.g. moving between GPUs

...link low-level and customer workload-level perf targets

Microsoft

We're hiring!

https://jobs.careers.microsoft.com/ – search #aifx

tiharr@microsoft.com