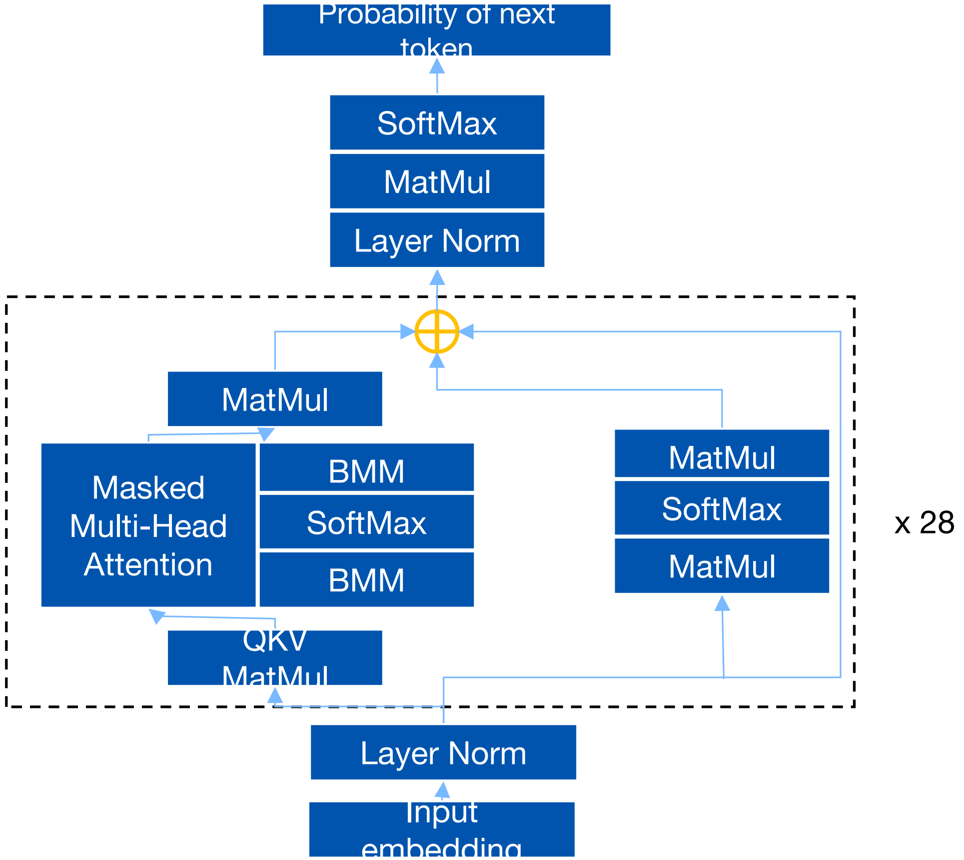
大语言模型时代：最大化CPU价值的优化策略

何普江
英特尔AI软件架构师

目录

背景 (为什么考虑最大化CPU价值?)

Computing Needs in LLM



GPT-J Model Structure

MatMul shapes in GPT-J

(suppose prompt token size = 2048, batch size=1, greedy search)

	1st token	**Next tokens**
QKV MatMul in MHA	A: 2048x4096 B: 4096x12288	A: 1x4096 B: 4096x12288
MHA (1st BMM)	A: 16x2048x256 B: 16x2048x256	A: 16x1x256 B: 16x2048x256
Output MatMul in MHA	A: 2048x4096 B: 4096x4096	A: 1x4096 B: 4096x4096
1st MatMul in FFN	A: 2048x4096 B: 4096x16384	A: 1x4096 B: 4096x16384
2nd MatMul in FFN	A: 2048x16384 B: 4096x4096	A: 1x16384 B: 4096x4096

Compute Bound

Memory Read Bandwidth Bound

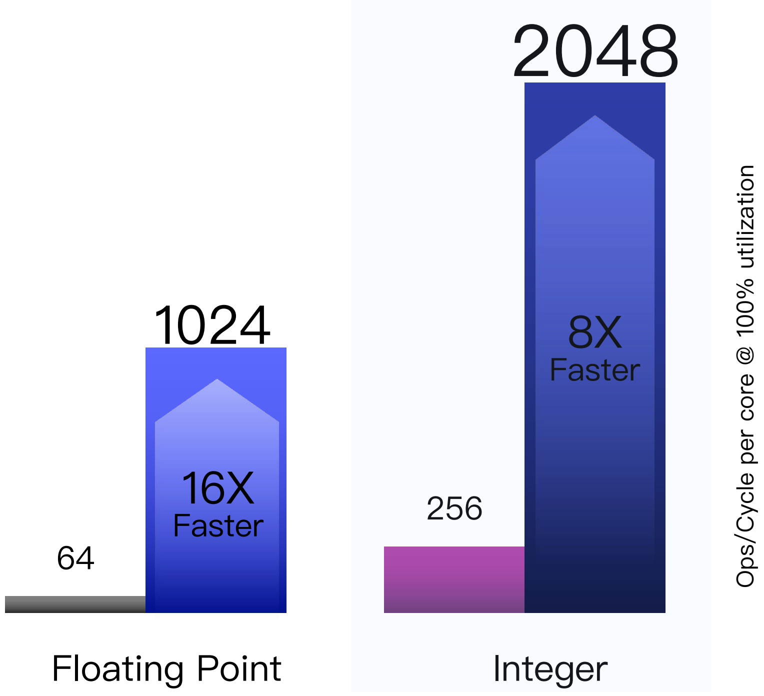
GPT Series Model Analysis

- Parameters visited during one time inference
  - $P = n_{layer} * (4 * d_{\text{hidden}}^2 + 2 \ * 4 \ * \ d_{\text{hidden}}^2) + n_{vocab} * d_{\text{hidden}}$

- Memory latency & Compute latency
  - $laetency_{memory} = \frac{P * n_{bytes}}{Peak\ memory\ bandwidth}$
  - $latency_{compute} = \frac{2 * P * (BatchSize * SeqLen)}{Peak\ FLOPs}$

- Arithmetic intensity
  - $AI = \frac{2 * P * B * S}{P * n_{bytes}} = \frac{2 * B * S}{n_{bytes}}$ FLOIPS/byte
  - Peak AI for SPR–SP with BF16 with AMX
    - $MAX\ AI_{spr-spbf16} = \frac{123.2TFLOPS}{307.2G/1000} = 401$ FLOIPS/byte
    - Compute bound
      - $AI > MAX\ AI_{spr-spbf16} \Rightarrow \mathrm{B} * \mathrm{S} > 401$
    - Memory bound
      - $AI > MAX\ AI_{spr-spbf16} \Rightarrow B * S < 401$

Latest CPU Features for LLM



Higher is Better

- AVX-512 (2xFMA) FP32
- AMX (TMUL) BF16
- AVX-512 (2xFMA) INT8
- AMX (TMUL) INT8

Results have been simulated. For workloads and configurations visit www.intel.com/ArchDay21claims . Results may vary

intel XEON MAX SERIES

Only x86 CPU with High Bandwidth Memory (HBM)

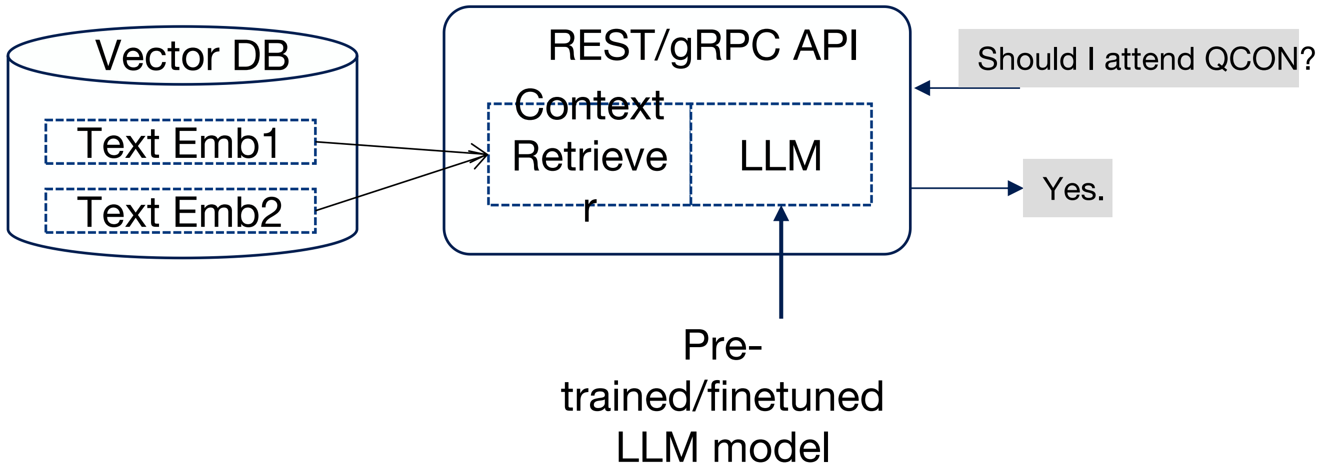
64GB HBM2e	112.5MB shared LLC	DDR5 8 channels per CPU @ 4800MTS (1DPC) / 16 DIMMs per socket

~1TB/s memory BW

>1GB/core HBM memory capacity

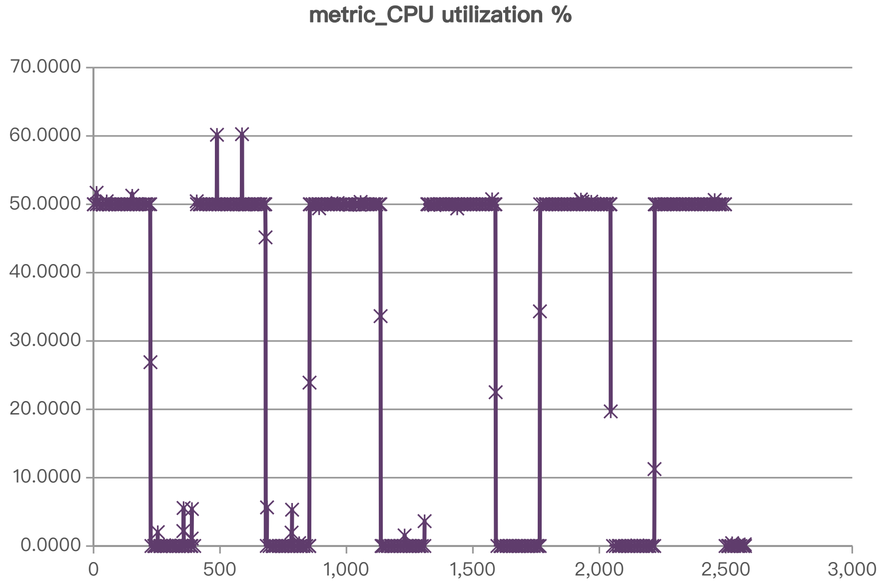
CPU is NOT Fully Utilized!

LLM Inference Pipeline



Pre-processing and post processing in LLM inference is relatively simple and do not need too much CPU resource.

CPU Utilization in LLM Training
(offload mode)



Even for offload LLM training, CPU is still not fully utilized.

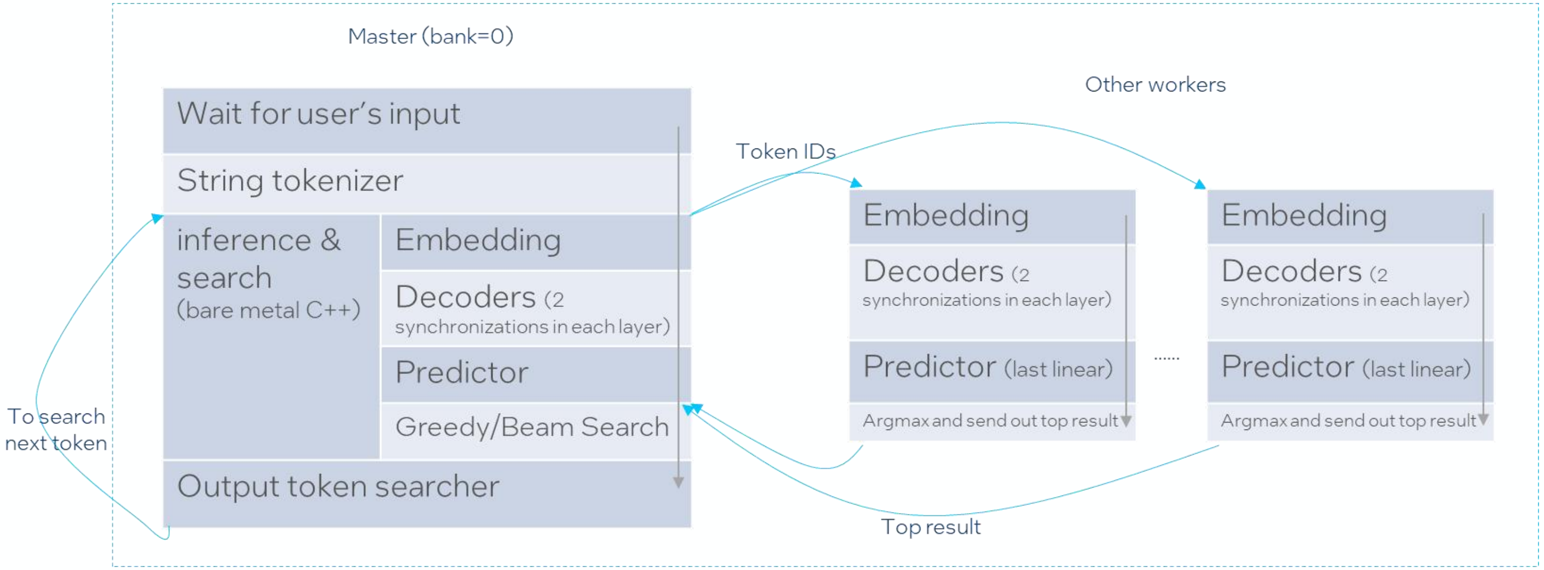
CPU上如何优化大语言模型?

Optimization

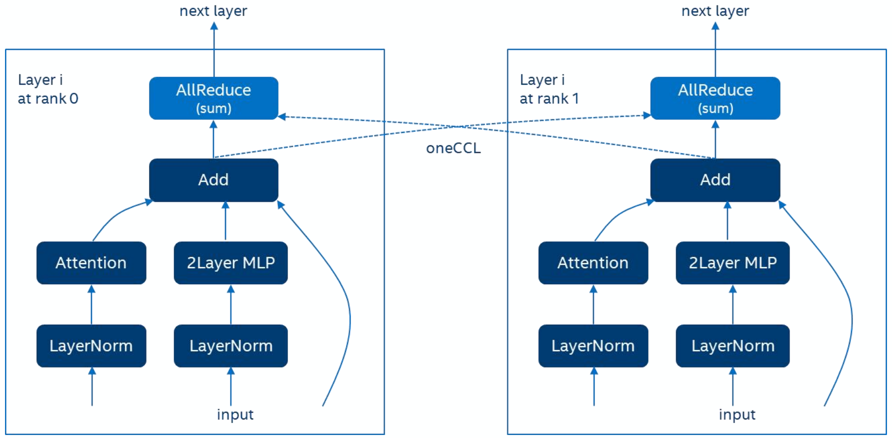
- Leverage the high-performance kernel (e.g., oneDNN)
- Avoid redundant computing
  - Continuous batching
  - Causal masking
  - Prefix sharing
- Lower precision & Sparsity
- Graph fusion
- Minimize memory copy and reorder
- Reuse the memory
- Distributed inference & use efficient communication library — oneCCL
- Runtime tunning

Optimization for Distributed Inference

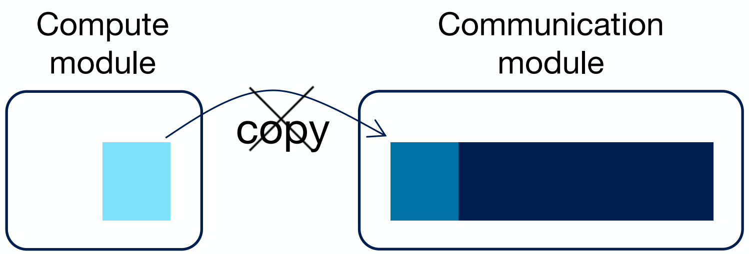
Distributed inference based on oneCCL



Improve scalability by minimizing synchronization.



One time synchronization per layer is enough for some models

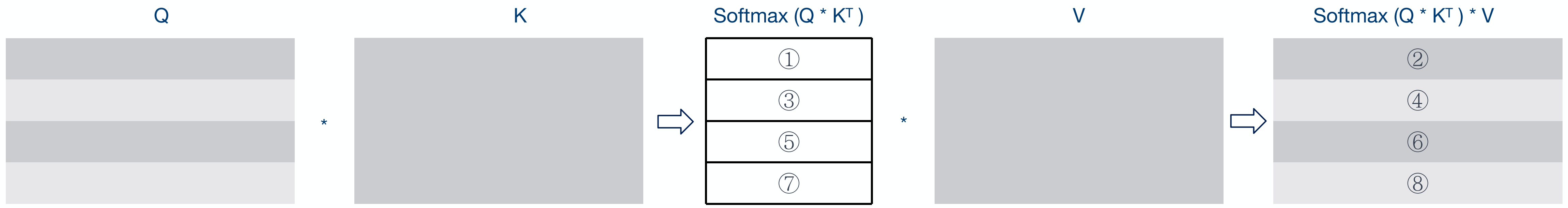


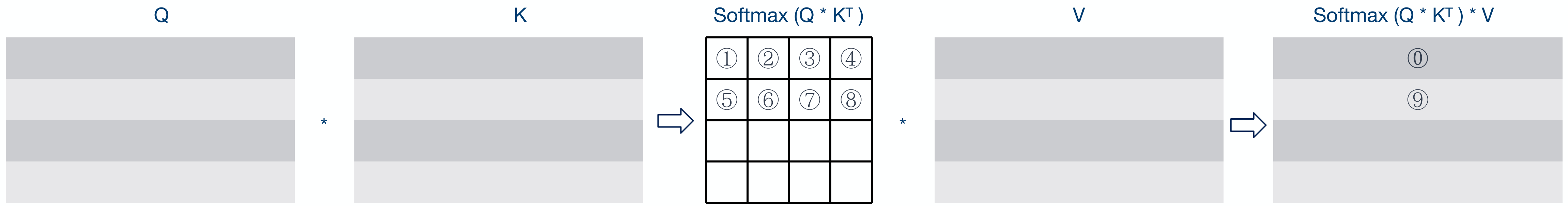
Minimize memory copy with full stack ownership

Attention Optimization

Computing Order: ①②③④⑤⑥⑦⑧
Intermediate score size: Same size with ①
Advantage: less score buffer w/o redundant computing

SlimAttention
(split score in 1 dimesion)

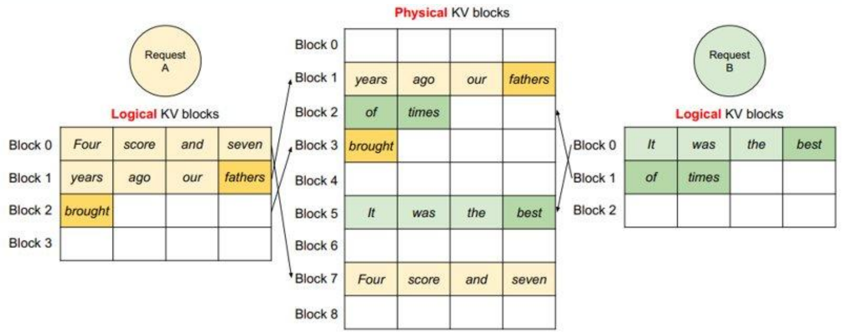


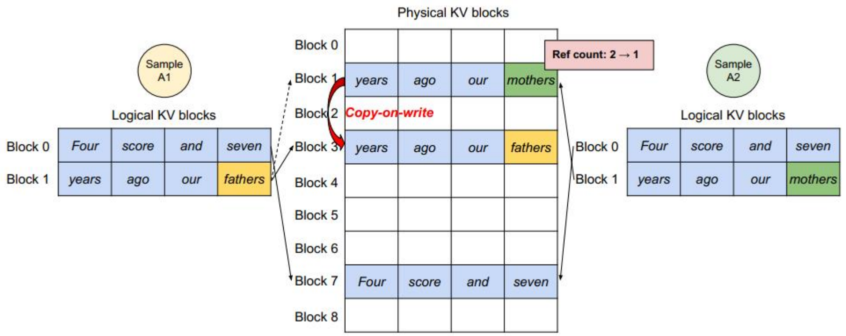


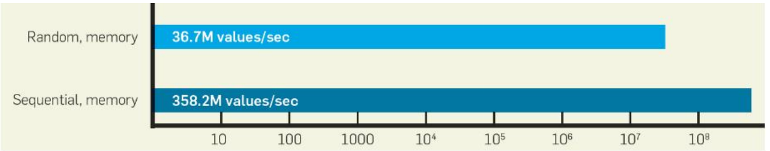
FlashAttention
(split score in 2 dimesion)

Computing Order: ①⓪②⓪③⓪④⓪⑤⑨...
Intermediate score size: Same size with ①
Advantage: minimal intermediate buffer w/ some redundant computing

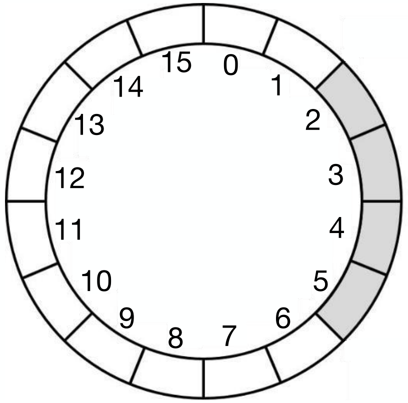
Do We Need Paged Attention on CPU?

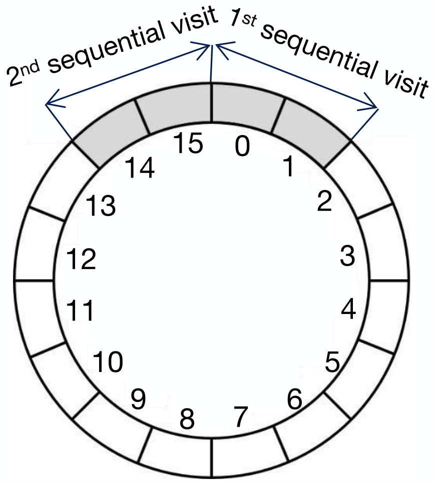




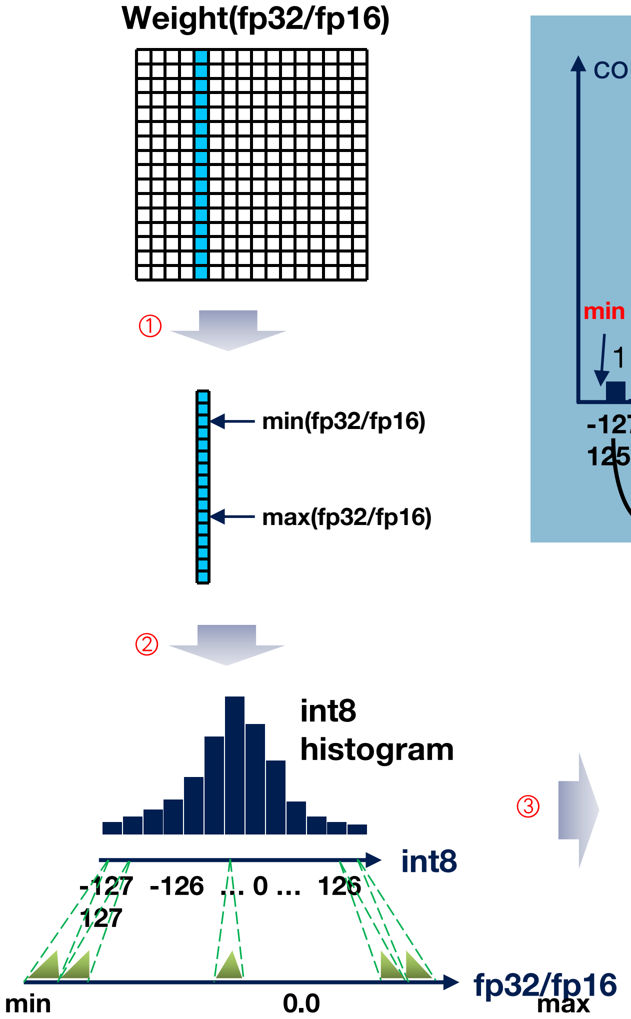


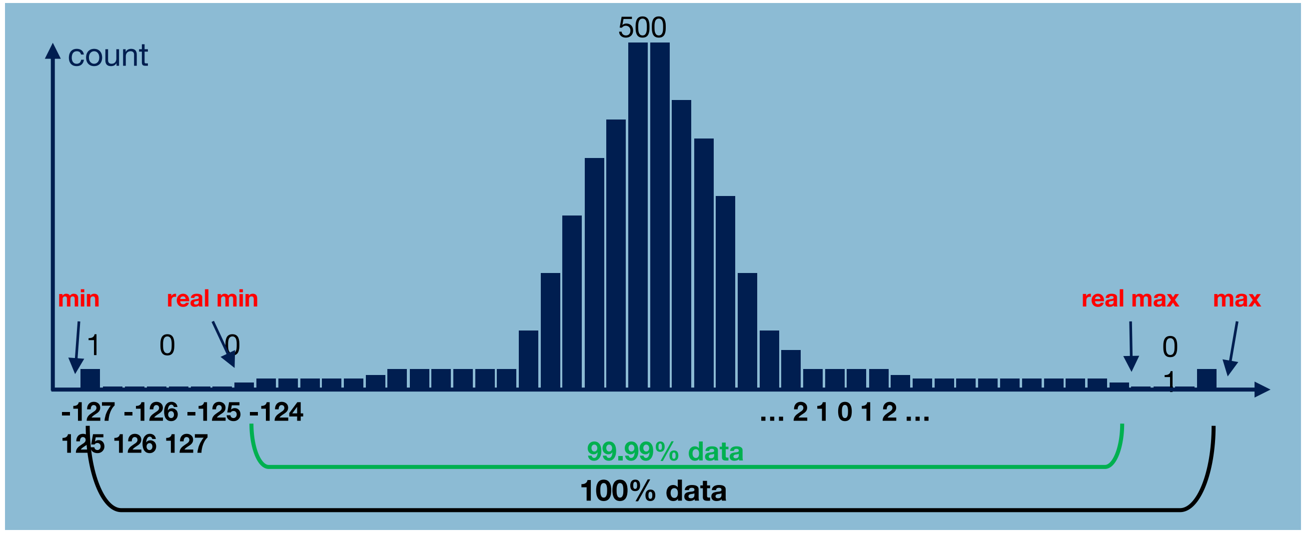
"Pathologies of Big Data" by Adam Jacobs in the ACM Communications, 2009

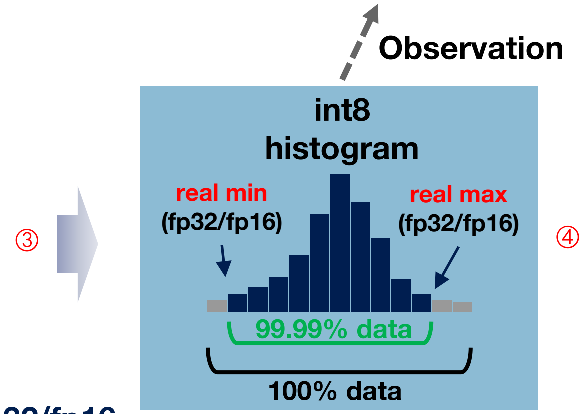


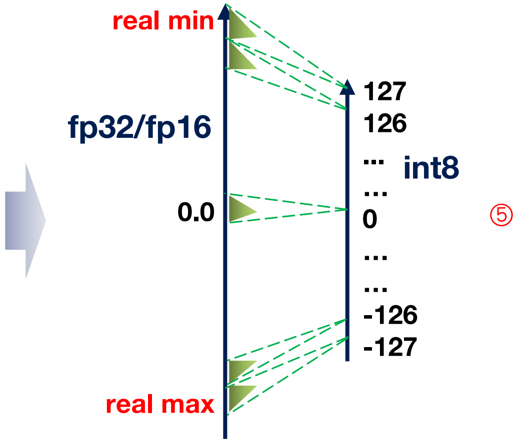


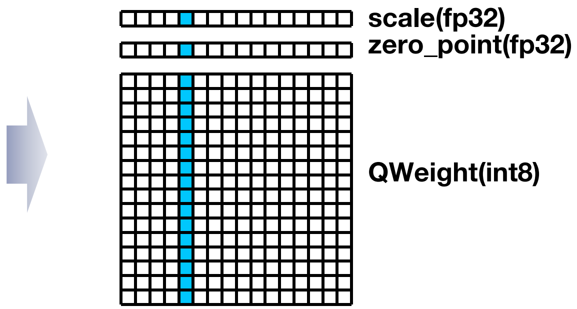
Int8 Weight Only Quantization











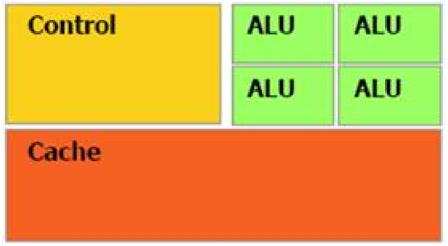
LLaMA2 7B	Accuracy
HF + AutoGPTQ _INT8	73.7046%
xFT + INT8_Convert	73.7629%

With histogram-based quantization, we could get very good accuracy in xFasterTransformer.

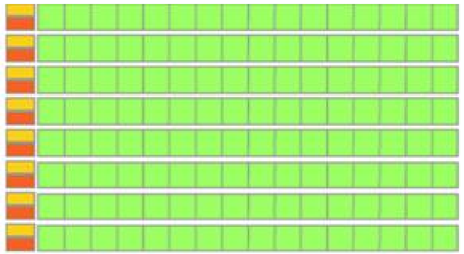
最大化CPU价值

CPU vs. GPU

CPU



GPU



A smaller number of larger cores	A larger number (thousands) of smaller cores
Low latency	High throughput
Performs fewer instructions per clock	Performs more instructions per clock
Designed and optimized for complex programs w/ serial processing	Optimized for parallel processing w/ bulk repetitive calculations
Automatic cache management	Allows for manual memory management
Large memory capacity	Limited memory capacity

Key Challenges in LLM Inference

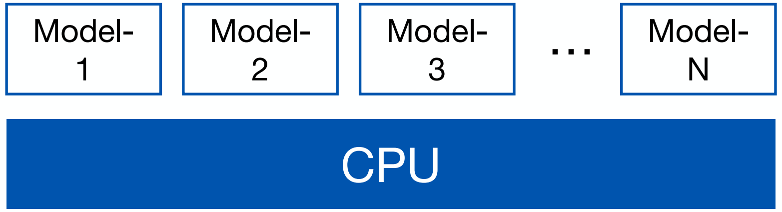
- Autoregressive generation
- Attention (square)
- Model is large

Key HW Factors in LLM Inference

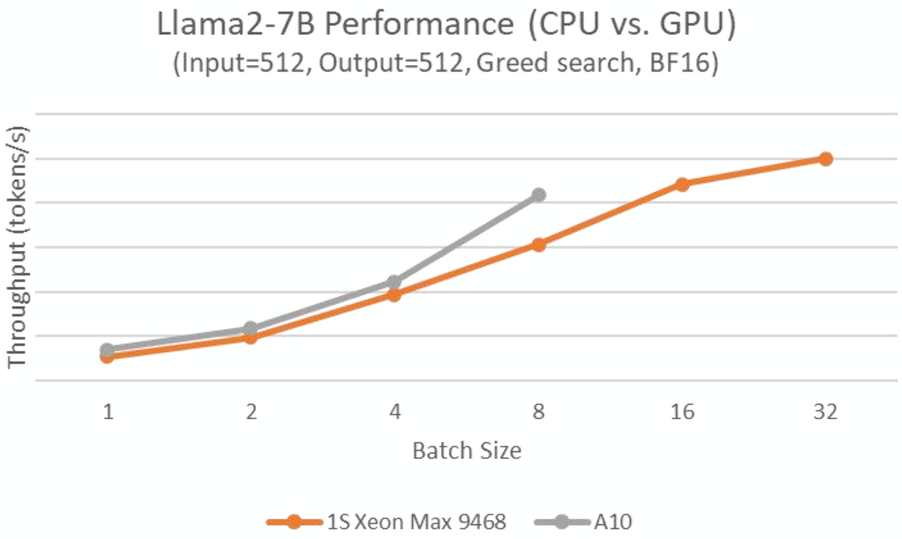
- Memory bandwidth
- Computing
- Memory capacity

Scenarios CPU has values

- Long tail models (many models, few requests)
- Offline mode (to maximize throughput)
- Occasional demand
- Very long prompt token size and no strict latency requirement
- Very large model and no enough GPU
- Hybrid solution (e.g., speculative sampling)



- All models loaded in memory
- Not all models serving together



Speculative Decoding

Token Generation
Draft Model

Proposed Tokens →

Token Verification
Target Model

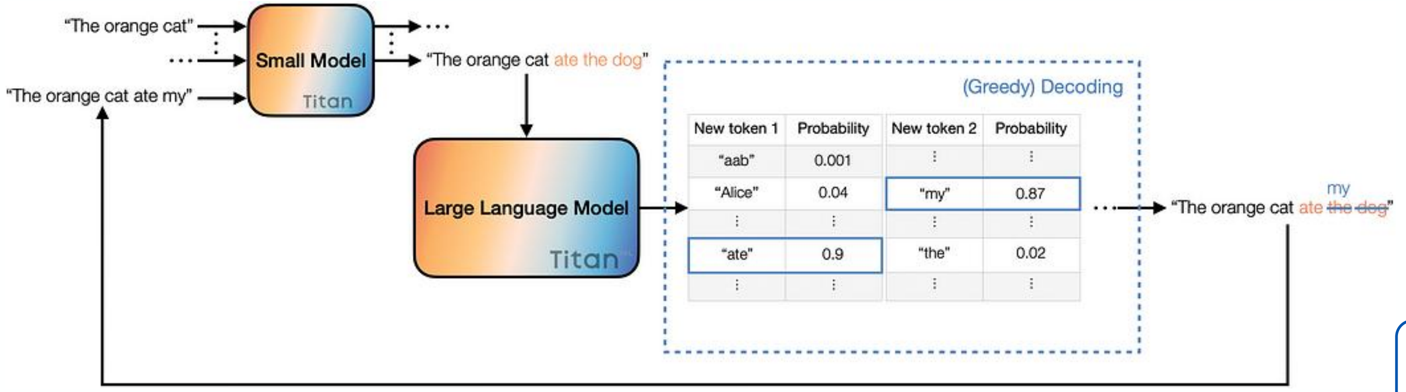
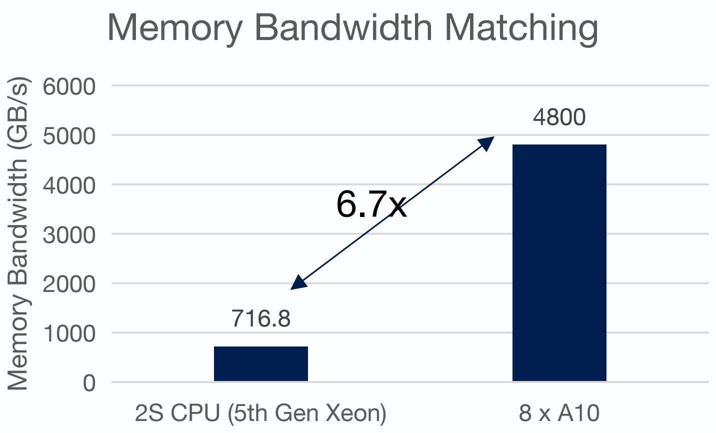


Image from https://medium.com/@TitanML/in-the-fast-lane-speculative-decoding-10x-larger-model-no-extra-cost-f33ea39d065a



In Practice, draft models are about 15-20x smaller than the target model.

Draft² Model → Proposed Tokens → **Draft Model** → Proposed Tokens → **Target Model**

总结

Why Considering CPU? → How to optimize on CPU? → When to use CPU?

Try our solution on Xeon

https://github.com/intel/intel-extension-for-pytorch

Build your own solution or seek ultimate perf

https://github.com/intel/xFasterTransforme
r