14/Oct/24

GPU's Architecture

Link :- Alfredclord.github.io transformer Inference Optimization Toolkit

→ Streaming Processors (SM) → [SM] like cores in CPU.
- basic CPU block
- contains
  - Instruction Schedulers
  - Instruction execution Pipelines

→ [HBM], high bandwidth memory
- special off chip memory
- data is initially stored and written back
- physically bounded to GPU in Stacked layers with 1000's of pins, provides massively Parallel data throughput by design

[Throughput = number of items and instructions that a GPU can process simultaniously

[Number of Samples processed per second by GPU]

→ SM access data and code from HBM via L2 Cache
- acts as intermediate level between off chip and on chip memories and caches data that be shared among multiple SMs

→ SM has its own → managed by GPU itself

1. L1 Cache

2. Shared Memory (SRAM)
   - faster than HBM, as on chip memory
   - managed by programmer

→ NV Link :- high bandwidth interconnect for GPUs to communicate to each other.

PCIe bus :- helps GPUs to talk to outside world
[common on motherboards to transfer data]

→ Single Node :- 8 GPUs packed together

→ GPU Capability :-

① Compute Performance :- Measured by Number of Trillion Float Operations Per Second [TFLOPS]

② GPU memory need to store model parameter
ex:- GPT : 175 B parameters
Storage :- 350 GB in ~~fp32~~ in fp16

③ Memory Bandwidth :- GB/s → Speed of bytes movement GPU to processing units

T4 Graphics Card :- (2018)

(2018)
- 65 TFLOPs
- 40 SMs
- 64 kB L1 Cache
- 4 MB L2 Cache with 1.3 TB/s bandwidth
- 16 GB HBM with 300 GB/s bandwidth

A100 Graphics Card — 2020
- 312 TFLOPs
- 108 SMs — 192 kB L1 Cache
- 40 MB - L2 Cache
- 80 GB - HBM with 1.55 TB bandwidth

GPU Capabilities Grow Exponentially Fast →

Some bottlenecks are :-

① Compute bound :- time spent on arithmetic operations > time spent on other operations (ex :- memory access)

② Memory Bound :- time taken by memory access > computation time
ex :- element wise operations (activation fn, dropout)
reductions (Sum, Softmax, normalization)

③ Overhead bound :- everything else, such as communication bound, interpreter bound ;

Arithmetic Intensity :- balance measured between ① & ②
ops : byte → ratio : to know if we are in a compute or memory bound

Mathematical Example to Understand this :-

(Q) Find out if "Linear layer forward Pass" on A100 GPU is memory or compute bound?

Ans) Given :- $x \in R^{B \times d}$ input batch

$W \in R^{d \times d}$ weight matrix

B = batch size

d = embedding dimension

Linear layer = matrix multiplication $= xW$

Computation required $= 2Bd^2$ FLOPs

Compute time $= T_{compute} = \frac{\#\ FLOPs}{\text{compute Performance}} = \frac{2Bd^2}{312 \times 10^{12}}$ s

$2d^2$ bytes = need to read this from memory to load weight matrix W

let $B \ll d$

$T_{memory} = \frac{\#\ bytes}{\text{memory bandwidth}} = \frac{2d^2}{1.55 \times 10^{12}}$ s

Arithmetic Intensity $= \frac{\#\ flops}{\#\ bytes} \simeq \frac{\text{compute performance}}{\text{memory bandwidth}}$

$$\frac{T_{compute}}{T_{memory}} = \frac{B}{200}$$

Hence, B is (batch size) smaller than 200
this is memory bounded

Increasing B to greater than 200, increases the computational time which keeping memory transfer time constant then it will be compute bound Scenario

---

High Level Algorithmic Optimization

$$Attention(Q,K,V) = Softmax\left[\frac{QK^T}{\sqrt{d}}\right]\cdot V$$

$Q \in R^{L \times d}$
$K, V \in R^{M \times d}$
L, M = seq length.

$$Multihead\ Attention(Q,K,V) = [head_1, \ldots, head_h]\cdot W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad i = 1, 2, \ldots h$$

W^Q, W^K, W^V, W^O = Learnable Parameters

If $Q = K$: Multihead Self Attention

If $Q \neq K$: Multihead Cross Attention.

Text Generation Stages:

① Prefill

② Auto regressive Decoding

* Fact about GPU :- GPU has computational capacity orders of magnitude higher than memory bandwidth.

KV Cache :-

→ we can Cache K and V values in the process of generation to improve efficiency, reduce redundant computational requirements [for long sequences] is called KV Cache

→ In each time step, the K and V computed for the last token are added to the Cache to reuse later step.

→ [ISSUE] To store this KV Cache we need lot of HBM Capacity, or either load it from CPU [DRAM] which is slower by one or 2 Orders of magnitude compare to loading from HBM

Multi Query Attention | Grouped Query Attention :- [MQA]

→ Shares the cached key value pairs across multiple queries (KV Cache pairs)

which reduces the memory requirements during next token generation.

leading to higher inference throughput

but this leads to cutting model parameters. and Quality degradation

Technique to avoid quality degradation :-
Technique interpolates between MHA and MQA
that is GQA → Grouped Query Attention.

GPU Again

CUDA Kernel Fusion

GPU → performs thousands of operations in parallel
Called as Threads

Each thread has its own memory
Called Registers

threads running on same SM can be grouped together → Thread blocks
to execute at same time [max 1024]

threads within thread blocks can be load data from
Global memory [HBM] into shared memory
[SRAM] used to write results back

Warps :- when SM is given one or more thread blocks
to execute it partition them to warps.

Warp = set of 32 threads

Grid :- multiple thread blocks combined to form

Kernal :- All the computations are defined in kernels

- Small C++ functions executed multiple times in Parallel by different threads
- launched as a grid

Tensor Cores :- Specialized units for matrix multiplies

Kernal Fusion :- Implementation of kernal which performs multiple computations at once without extra memory access

custom CUDA kernals → Triton
Pallas