THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

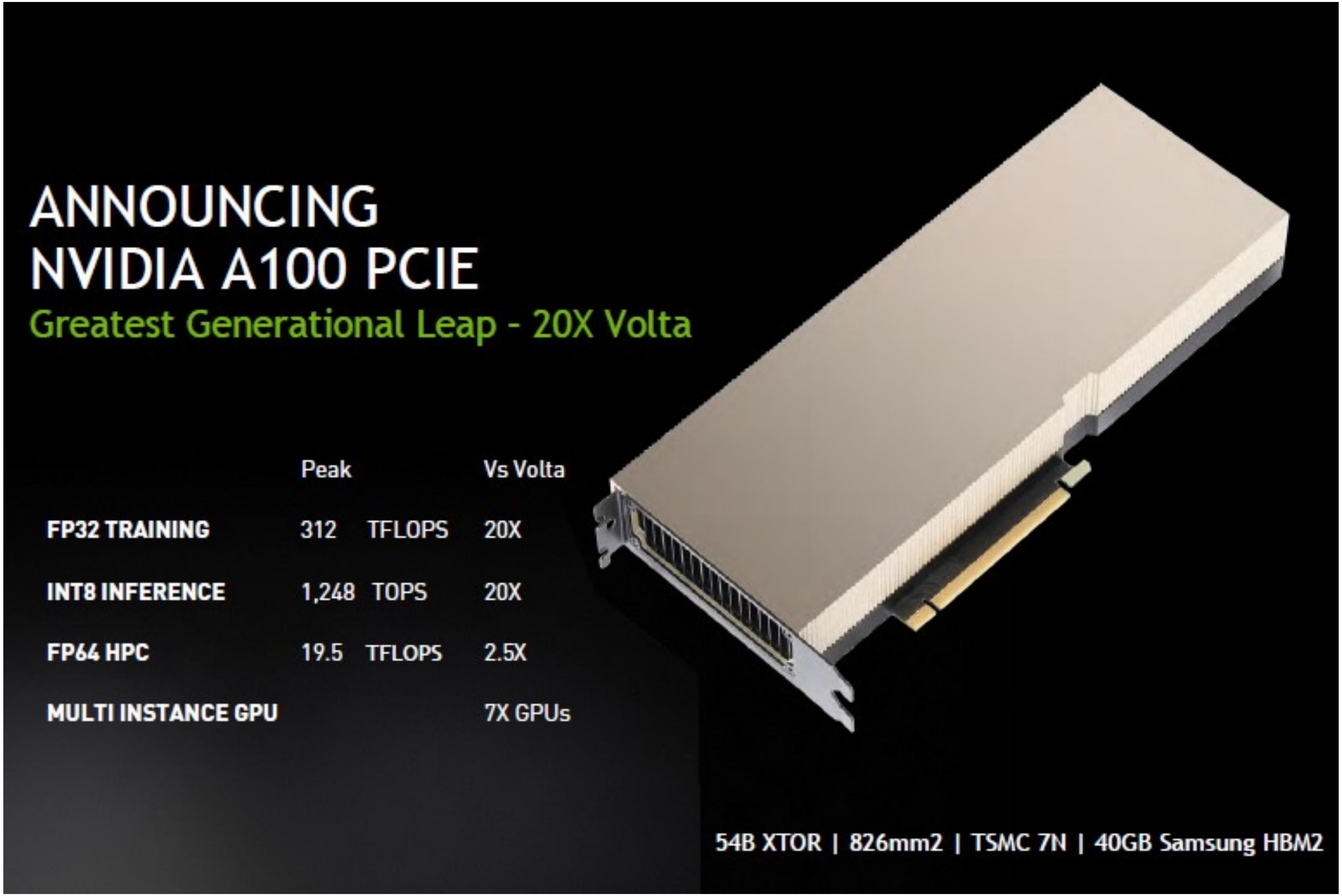
RELAXED SYSTEM LAB

Nvidia GPU Performance

COMP4901Y

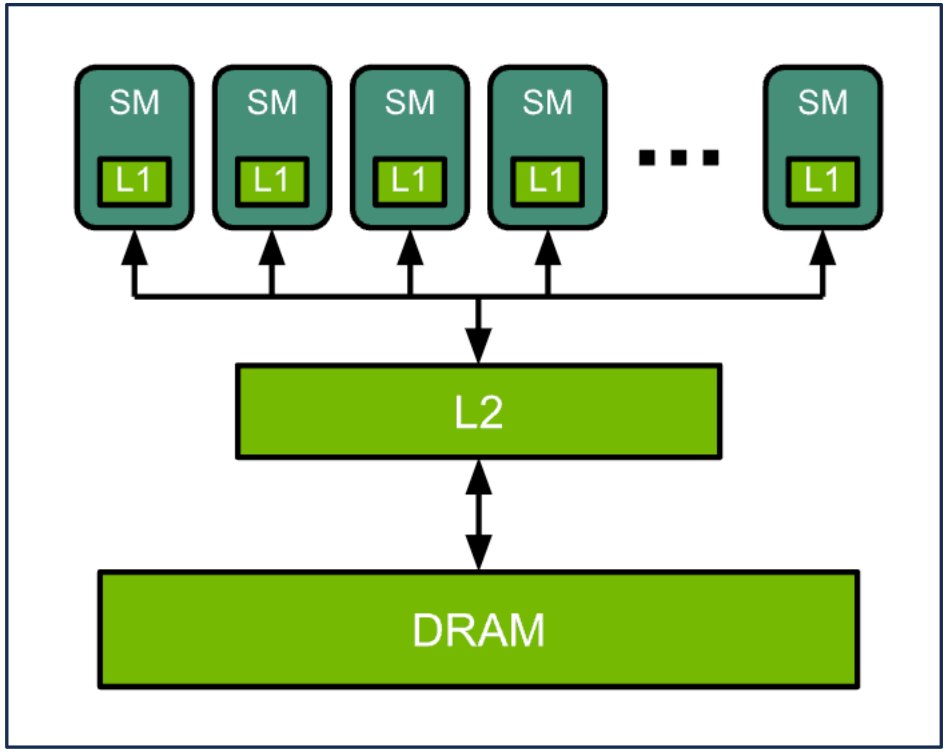
Binhang Yuan

GPU Architecture

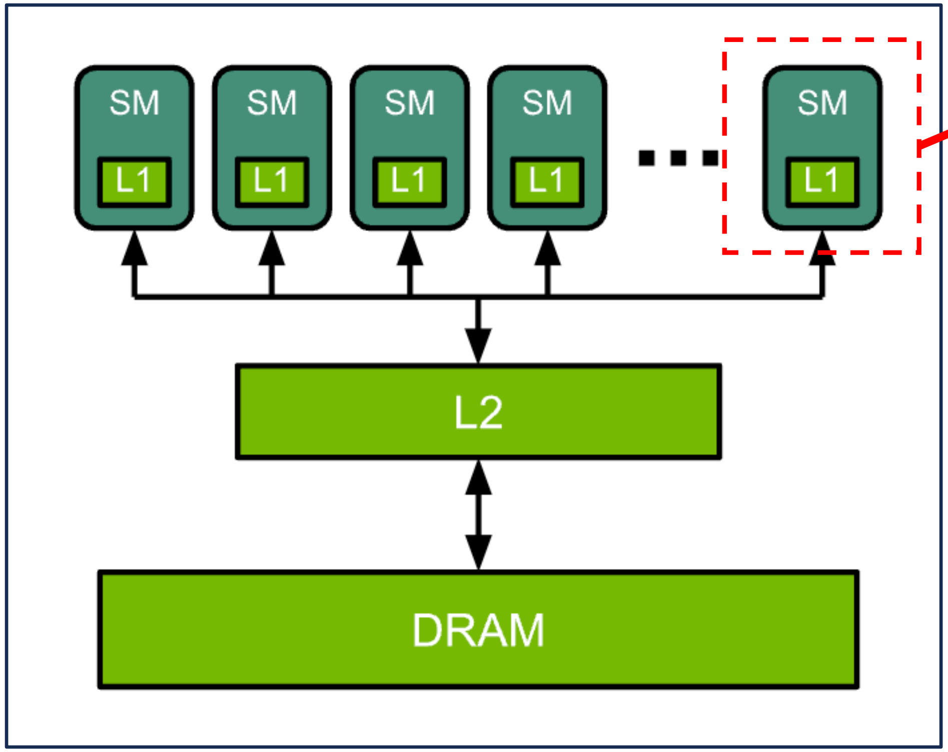

ANNOUNCING
NVIDIA A100 PCIE
Greatest Generational Leap - 20X Volta
Peak
Vs Volta
FP32 TRAINING
312 TFLOPS
20X
INT8 INFERENCE
1,248 TOPS
20X
FP64 HPC
19.5 TFLOPS
2.5X
MULTI INSTANCE GPU
7X GPUs
54B XTOR | 826mm2 | TSMC 7N | 40GB Samsung HBM2

GPU Architecture

- The GPU is a highly parallel processor architecture, including processing elements and a memory hierarchy.
- The memory hierarchy:
  - L0, L1 cache in Streaming Multiprocessors (SMs);
  - On-chip L2 cache;
  - High bandwidth DRAM (HBM).
- Arithmetic and other instructions are executed by the SMs.
- Data and code are accessed from DRAM via the L2 cache.



Ampere GPU Architecture



108 SM in a A100 GPU

Ampere GPU SM

- In Ampere GPU, SM contains **four** processing blocks that share an L1 cache for data caching.
- Each processing block has:
  - 1 Warp scheduler (where the maximum number of thread blocks per SM is 32);
  - 16 INT32 CUDA cores;
  - 16 FP32 CUDA cores;
  - 8 FP64 CUDA cores;
  - 8 Load/Store cores;
  - 1 SFU core (special function units: e.g., sin, cos)
  - 1 <u>**Tensor core**</u> for matrix multiplication;
  - 1 16K 32-bit register file.

A100 GPU Memory Hierarchy

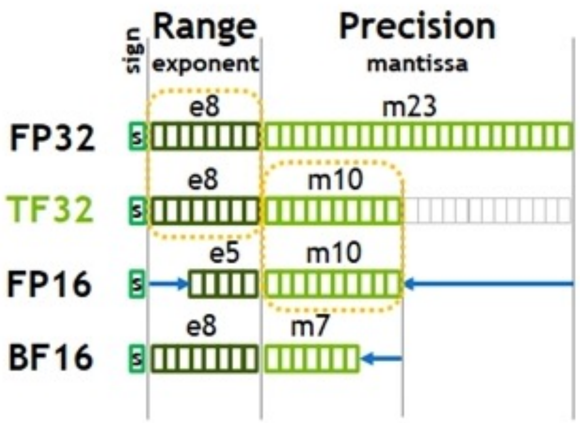
RELAXED SYSTEM LAB

- Size:
  - L1 cache: 192 KB per SM;
  - L2 cache: 40 MB
  - HBM: 80 GB
- Accessibility:
  - The L2 cache is unified, shared by all SMs, and set aside for data and instructions.
  - The L1 instruction cache is private to a single streaming multiprocessor.
  - The L0 instruction cache is private to a single streaming multiprocessor subprocessing block.

https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

A100 GPU Tensor Core Computation

- 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

A100 GPU Specifications

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

A100 GPU Specs	
Tensor core Float 32 (TF32)	156 TFLOPS
Tensor core Float 16 (FP16)	**312 TFLOPS**
Tensor core Int 8 (INT8)	624 TOPS
GPU Memory	80 GB
GPU Memory Bandwidth	2039 GB/s

$$1024 \times 2 \times 1.41 \times 10^9 \times 108 = 312 \times 10^{12}$$

Tensor Cores

- Tensor Cores were introduced in the NVIDIA Volta™ GPU architecture to accelerate matrix multiply and accumulate operations for machine learning and scientific applications.
- These instructions operate on small matrix blocks:
  - For example, 16×16 blocks in A100 GPUs.
- Tensor Cores can compute and accumulate products with higher precision than the inputs:
  - During training with FP16 inputs, Tensor Cores can compute products without loss of precision and accumulate in FP32.

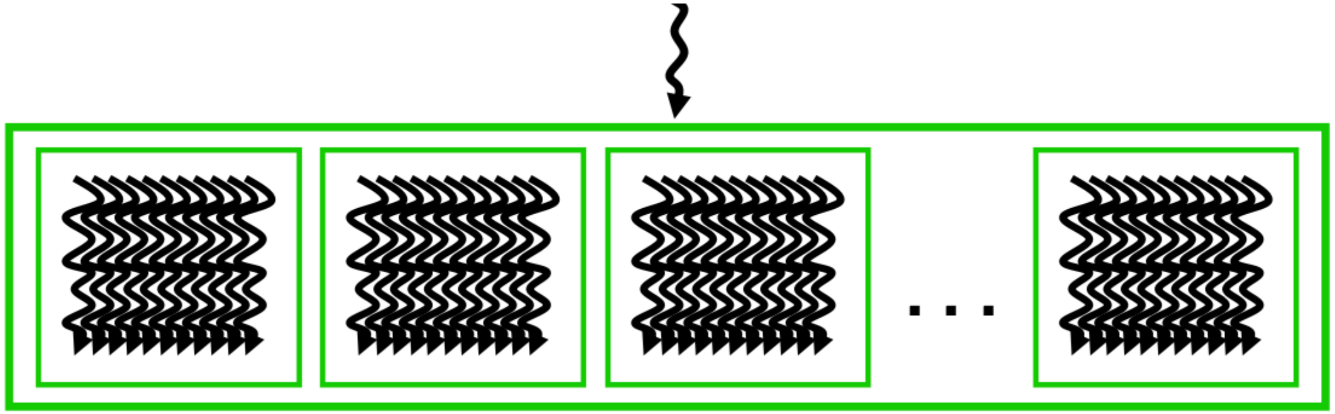
GPU Execution Model

2-level Thread Hierarchy

- GPUs execute functions using a 2-level hierarchy of threads.
  - A given function's threads are grouped into equally sized thread blocks, and a set of thread blocks is launched to execute the function.
- GPUs hide dependent instruction latency by switching to the execution of other threads.
  - The number of threads needed to utilize a GPU effectively is much higher than the number of cores or instruction pipelines.

Host: launch the function

Device: Parallel Kernel



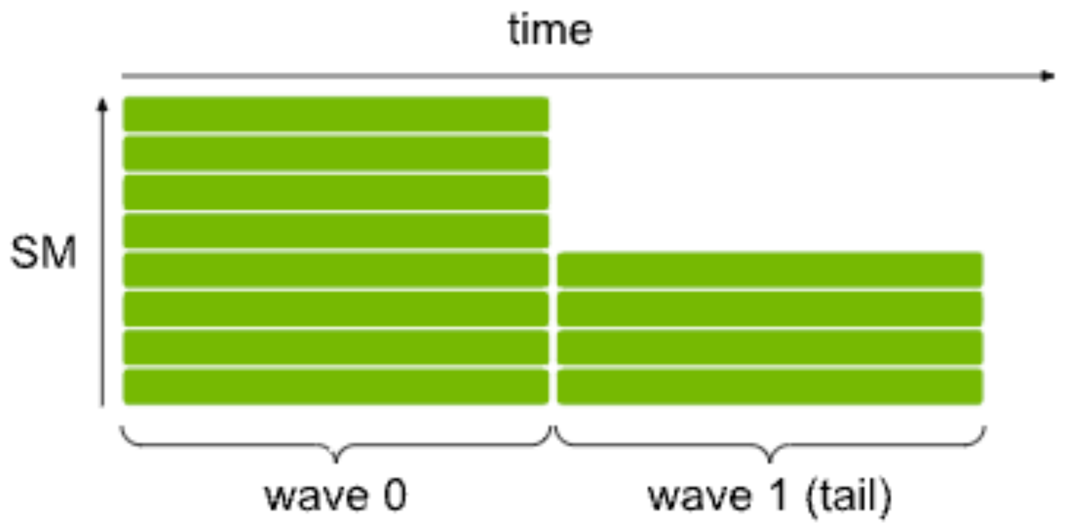
2-level Thread Hierarchy

RELAXED SYSTEM LAB

- GPUs have many SMs, each of which has pipelines for executing many threads and enables its threads to communicate via shared memory and synchronization.
- At runtime, a thread block is placed on an SM for execution, enabling all threads in a thread block to communicate and synchronize efficiently.
- Launching a function with a single thread block would only give work to a single SM; to fully utilize a GPU with multiple SMs, one needs to launch many thread blocks.
- Since an SM can execute multiple thread blocks concurrently, typically, one wants the number of thread blocks to be several times higher than the number of SMs.

2-level Thread Hierarchy

- Minimize the "tail" effect: at the end of a function execution, only a few active thread blocks remain.
- We use the term wave to refer to a set of thread blocks that run concurrently.
- It is most efficient to launch functions that execute in several waves of thread blocks - a smaller percentage of time is spent in the tail wave, minimizing the tail effect and thus the need to do anything about it.
- For the higher-end GPUs, typically only launches with fewer than 300 thread blocks should be examined for tail effects.



Utilization of an 8-SM GPU when 12 thread blocks with an occupancy of 1 block/SM at a time are launched for execution. The blocks execute in 2 waves, the first wave utilizes 100% of the GPU, while the 2nd wave utilizes only 50%.

Understanding Performance

Overview

RELAXED SYSTEM LAB

- The performance of a function on a given processor is limited by one of the following three factors:
  - Memory bandwidth;
  - Math bandwidth;
  - Latency.
- Consider a simplified model where a function:
  - Read its input from memory;
  - Perform math operations;
  - Write its output to memory.

Modeling the Cost

- T_{mem} time is spent in accessing memory;
- T_{math} time is spent performing math operations.
- If we further assume that memory and math portions of different threads can be overlapped;
- The total time for the function is $\max(T_{mem}, T_{math})$.
- The longer of the two times demonstrates what limits performance:
  - If math time is longer, we say that a function is *math limited;*
  - If memory time is longer then it is *memory limited*.

Arithmetic Intensity

RELAXED
SYSTEM LAB

- How much time is spent in memory or math operations depends on both the algorithm and its implementation, as well as the processor's bandwidths.
- Memory time is equal to the number of bytes accessed in memory divided by the processor's memory bandwidth.
- Math time is equal to the number of operations divided by the processor's math bandwidth.

Arithmetic Intensity

- 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*.

Arithmetic Intensity

- Arithmetic intensity: the ratio of algorithm implementation operations and the number of bytes accessed.
- Ops:byte ratio: the ratio of a processor's math and memory bandwidths.
- Thus, an algorithm is math limited on a given processor if the algorithm's arithmetic intensity is higher than the processor's ops:byte ratio.
- Conversely, an algorithm is memory limited if its arithmetic intensity is lower than the processor's ops:byte ratio.

Arithmetic Intensity

RELAXED SYSTEM LAB

- Compare the algorithm's arithmetic intensity to the ops:byte ratio on an NVIDIA Volta V100 GPU.
    - V100 has a peak math rate of 125 FP16 Tensor TFLOPS;
    - An off-chip memory bandwidth of approx. 900 GB/s
    - An on-chip L2 bandwidth of 3.1 TB/s;
- So it has a ops:byte ratio between 40 and 139, depending on the source of an operation's data (on-chip or off-chip memory).

Operation	Arithmetic Intensity	limited by
Linear layer (4096 outputs, 1024 inputs, batch size 512)	315 FLOPS/B	arithmetic
Linear layer (4096 outputs, 1024 inputs, batch size 1)	1 FLOPS/B	memory
Max pooling with 3x3 window and unit stride	2.25 FLOPS/B	memory
ReLU activation	0.25 FLOPS/B	memory
Layer normalization	10 FLOPS/B	memory

Arithmetic Intensity

RELAXED SYSTEM LAB

- Note that this type of analysis is a simplification, as we're counting only the algorithmic operations used.
- In practice, functions also contain instructions for operations not explicitly expressed in the algorithm, such as:
  - Memory access instructions;
  - Address calculation instructions;
  - Control flow instructions, and so on.

Limited by Lantency

RELAXED SYSTEM LAB

- The arithmetic intensity and ops:byte ratio analysis assumes that a workload is sufficiently large to saturate a given processor's math and memory pipelines.
- However, if the workload is not large enough, or does not have sufficient parallelism, the processor will be under-utilized and performance will be limited by latency.
- For example:
  - Consider the launch of a single thread that will access 16 bytes and perform 16000 math operations.
  - While the arithmetic intensity is 1000 FLOPS/B and the execution should be math-limited on a V100 GPU, creating only a single thread grossly under-utilizes the GPU, leaving nearly all of its math pipelines and execution resources idle.

DNN Operation Categories

Elementwise Operations

RELAXED SYSTEM LAB

- Elementwise operations may be unary or binary operations;
- The key is that layers in this category perform mathematical operations on each element independently of all other elements in the tensor.
- For example:
  - A ReLU layer returns max(0, x) for each x in the input tensor.
  - The element-wise addition of two tensors computes each output sum value independently of other sums.
- Layers in this category include most non-linearities (sigmoid, tanh, etc.), scale, bias, add, and others.
- These layers tend to be ***memory-limited***, as they perform few operations per byte accessed.

Reduction Operations

RELAXED SYSTEM LAB

- Reduction operations produce values computed over a range of input tensor values.
- For example:
  - Pooling layers compute values over some neighborhoods in the input tensor.
  - Batch normalization computes the mean and standard deviation over a tensor before using them in operations for each output element.
  - SoftMax also falls into the reduction category.
- Typical reduction operations have a low arithmetic intensity and thus ***are memory limited***.

Dot-Product Operations

- Operations in this category can be expressed as dot-products of elements from two tensors, usually a weight (learned parameter) tensor and an activation tensor.
- Examples:
  - Fully-connected layers are naturally expressed as matrix-vector and matrix-matrix multiplies.
  - Convolutions can also be expressed as collections of dot-products - one vector is the set of parameters for a given filter, and the other is an "unrolled" activation region to which that filter is being applied.
- Operations in the dot-product category can be math-limited if the corresponding matrices are large enough.
- However, for the smaller sizes, these operations end up being memory-limited. For example, a fully-connected layer applied to a single vector (a tensor for a mini-batch of size 1)) is memory limited.

Dot-Product Operations: Matrix Multiplication

- Compute $C = AB$ suppose:
  - A is an $M \times K$ matix; (M rows and K columns)
  - B is an $K \times N$ matix;
  - C is an $M \times N$ matix;
- A total of $M \times N \times K$ fused multiply-adds (FMAs) are needed to compute the product. so a total of $2 \times M \times N \times K$ flops are required.
- The total number of byte scan in FP16: $2(M \times K + K \times N + M \times N)$
- Arithmetic intensity= $\frac{M \times N \times K}{(M \times K + K \times N + M \times N)}$.

References

- https://docs.nvidia.com/deeplearning/performance/dl-performance-gpu-background/index.html#understand-perf
- https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/
- https://www.alcf.anl.gov/sites/default/files/2021-07/ALCF_A100_20210728%5B80%5D.pdf