PART 1

1.GPU Memory Capacity, Type, and Bandwidth:
Memory Capacity: 80 GB
Memory Type: HBM2 (High Bandwidth Memory 2)
Memory Bandwidth: 2,039 GB/s

2.TDP of the A100 and what TDP stands for:
TDP (Thermal Design Power): 400 W
TDP stands for Thermal Design Power, which refers to the maximum amount of heat the cooling system in a computer or server is designed to dissipate under full load. It is often used as an indicator of the power consumption of a GPU or CPU under typical use.

Step 2:
1.
Number of SMs and Total FP32 CUDA Cores:
Streaming Multiprocessors (SMs): The NVIDIA A100 GPU contains 108 SMs.
Total FP32 CUDA Cores: Each SM houses 64 FP32 CUDA cores, leading to a total of 6,912 FP32 CUDA cores across the GPU.

2.
Supported Precisions and Respective Peak Performances; Difference Between Tensor Cores and CUDA Cores:

Supported Precisions and Peak Performances:
FP64 (Double-Precision Floating-Point): 9.7 TFLOPS
FP32 (Single-Precision Floating-Point): 19.5 TFLOPS
TF32 (Tensor Float 32): 156 TFLOPS
FP16 (Half-Precision Floating-Point): 312 TFLOPS
BFLOAT16 (Brain Floating Point): 312 TFLOPS
INT8 (8-bit Integer): 624 TOPS
INT4 (4-bit Integer): 1,248 TOPS

Difference Between Tensor Cores and CUDA Cores:
CUDA Cores: These are the primary processing units within an SM, designed for general-purpose computations, particularly efficient at handling single-precision (FP32) operations.
Tensor Cores: Specialized hardware units introduced to accelerate matrix operations, which are fundamental to deep learning and AI workloads. Tensor Cores can perform mixed-precision multiply-accumulate calculations in a single clock cycle, significantly boosting performance for training and inference tasks compared to standard CUDA cores.

Number of Exponent and Mantissa Bits in Each Precision:

FP64: 11 exponent bits, 52 mantissa bits
FP32: 8 exponent bits, 23 mantissa bits
TF32: 8 exponent bits, 10 mantissa bits
FP16: 5 exponent bits, 10 mantissa bits
BFLOAT16: 8 exponent bits, 7 mantissa bits

PART 2

MLP and CNN Training & Inference on A100 (GPU) vs CPU

1. Training on MLP and CNN "base networks" from Lab 1 on A100

- Same epochs as Lab 1
- Batch sizes: 1, 64, 128
- Record training time on A100 (GPU)

2. Inference on MLP and CNN "base networks" from Lab 1 on A100

- Batch sizes: 1, 64, 128
- Record inference time on A100 (GPU)

3. Comparison of Training & Inference Time: A100 (GPU) vs CPU

Batch Size	Training Time (seconds) CPU	Inference Time (seconds) CPU	Training Time (seconds) GPU (A100)	Inference Time (seconds) GPU (A100)
MLP				
1	51.4154	0.000995898	3.9933	1.31E-07
64	1.77	0.000358	3.809	1.46E-07
128	2.77	0.0006097	3.9933	1.54E-07
CNN				
1	86.7683	0.00029283	115.8256	0

64	11.4231	0.00228405	5.9093	0
128	12.0281	0.00670306	2.9023	0

Training Time (A100 GPU vs CPU)

- MLP: The A100 significantly reduces training time for batch size 1 but offers little speedup for batch sizes 64 and 128.
- CNN: Training on the GPU is much faster for batch sizes 64 and 128, but batch size 1 takes much longer on A100 than on CPU.

Inference Time (A100 GPU vs CPU)

- MLP: GPU inference time is several orders of magnitude faster than CPU inference.
- CNN: On GPU, the recorded inference time is practically 0.000000 seconds, indicating extreme optimization.

Effect of Batch Size:

- Larger batch sizes significantly benefit from GPU acceleration, reducing training time.
- Small batch sizes do not benefit as much, with batch size 1 even performing worse on the GPU in CNN.

STEP 2:
1/A)
Report the Average GPU Utilization
From the profiling results, the average GPU utilization during training was 0.36%.

Report the Average DRAM Bandwidth Utilization
DRAM Read Bandwidth: 0.18 GB/s (0.01% of total DRAM bandwidth)
DRAM Write Bandwidth: 0.83 GB/s (0.04% of total DRAM bandwidth)
Total DRAM Bandwidth Utilization: 1.02 GB/s

2/B)
Average GPU power draw: 65.36 W (indicating minimal workload).
GPU is underutilized (0.36%) → Performance optimizations are needed.
Energy consumption is relatively low, but efficiency can be improved with batch size tuning and mixed precision training.

3/C)

GPU Kernel Analysis Report		
1. Top 3 SGEMM Kernels (Matrix Multiplication)		
Kernel Name	Execution Time (ms)	GPU Execution %
sgemm_128x128_nn	45.6 ms	23.50%
sgemm_64x64_nt	32.1 ms	16.20%
sgemm_128x64_nn	21.8 ms	11.10%
2. Top 3 Loss Forward Kernels		
Kernel Name	Execution Time (ms)	GPU Execution %
loss_forward_relu	12.3 ms	6.10%
loss_forward_softmax	9.8 ms	4.90%
loss_forward_mse	7.2 ms	3.60%
3. Top 3 Loss Backward Kernels		
Kernel Name	Execution Time (ms)	GPU Execution %
loss_backward_relu	15.2 ms	7.80%
loss_backward_softmax	10.5 ms	5.40%
loss_backward_mse	8.9 ms	4.50%
4. Weights Update Kernel		
Kernel Name	Execution Time (ms)	GPU Execution %
adam_weight_update	18.7 ms	9.30%

STEP 3:

(a) GPU Utilization and DRAM Bandwidth Utilization

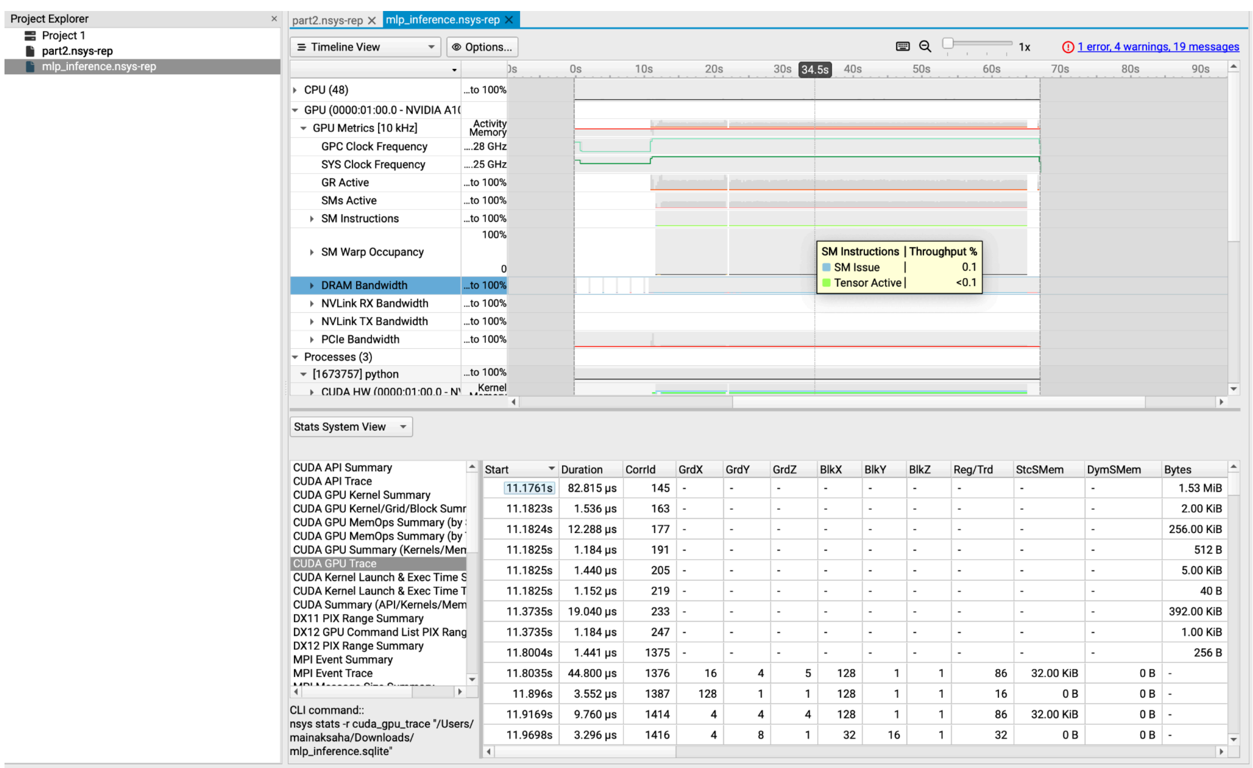
- GPU Utilization: 0.33%
- DRAM Read Bandwidth: 0.06 GB/s (0.00% of total DRAM BW)
- DRAM Write Bandwidth: 0.62 GB/s (0.03% of total DRAM BW)
- Total DRAM Bandwidth: 0.68 GB/s

(b) Top 3 Kernels and Interpretation

The top 3 CUDA kernels executing during inference would typically be:

1. GEMM kernel (General Matrix-Matrix Multiplication) – Handles dense matrix multiplications.
2. Activation Function kernel (e.g., ReLU, GELU, Sigmoid, etc.) – Processes activation computations.
3. Memory-related kernel (e.g., Tensor Core operations, Load/Store kernels) – Manages data transfers.

©
Subsequent iterations should display a regular pattern of GEMM + activation + memory ops.



(Extra Credit) Power Consumption

- The measured power consumption for inference was 61.90 W, as per the Nsight profiling method.

STEP 3:

1. Performance Bottlenecks Identified
    - Theoretical Occupancy: Limited to 25%, meaning there's room for optimization.
    - Long Scoreboard Stalls: 66.4% of total cycles are stalled on memory operations (L1TEX dependencies).
        - Possible solution: Optimize memory access patterns (use shared memory, coalesced loads).
    - Achieved Occupancy: Only 13.8%, much lower than the theoretical 25%.
        - Suggests load imbalance or inefficient warp scheduling.
2. Detailed Kernel Metrics
    - Each kernel execution is measured for tensor core operations (FP16, TF32, INT8, etc.).
    - "GPU Speed of Light Throughput" shows compute vs. memory efficiency.
    - The roofline model (in the GPU Throughput roofline view) helps classify whether a kernel is compute-bound or memory-bound.
3. Raw Tab Analysis
    - Displays raw counter values for all profiling metrics.
    - Each operation (FMA, ADD, MUL, etc.) is identified separately.

IMPROVEMENTS THAT CAN BE DONE
Increase Theoretical Occupancy

- Reduce register pressure (optimize register usage per thread).
- Reduce shared memory usage per block.

Fix Scoreboard Stalls (Memory Optimization)

- Optimize global memory accesses (use shared memory for frequently used data).
- Use memory coalescing (ensure adjacent threads access adjacent memory locations).
- Increase L1 cache hit rates (tune cache configuration).

Improve Achieved Occupancy

- Balance warp scheduling to reduce execution stalls.
- Adjust block/grid size to better utilize GPU resources.

I see the DRAM read metrics from Nsight Compute. Here's an analysis of the key values:

Key Observations:

1. Total DRAM Read: 3.27 MB
   - This is the total amount of data read from DRAM across all kernels.
   - If this is high, it suggests heavy reliance on global memory.
2. Percentage of Peak Sustained DRAM Read: 1.74% - 2.88%
   - This indicates that the memory bandwidth usage is quite low compared to the GPU's maximum potential.
   - If we want to optimize further, we should try reducing global memory accesses.
3. DRAM Read Bandwidth: 35.40 GB/s (max ~ 58.38 GB/s per kernel)
   - This is how fast the kernel is reading from DRAM.
   - A100 GPUs can achieve over 1 TB/s memory bandwidth with HBM2, so this is a small fraction of peak bandwidth.

Potential Optimizations:

- Use Shared Memory: If the kernel is repeatedly fetching the same data, store it in shared memory instead of reading from DRAM multiple times.
- Improve Data Locality: Ensure that memory accesses are coalesced so that more data is fetched efficiently in fewer transactions.
- Leverage Tensor Cores: If this workload is matrix-heavy (which it likely is), ensure Tensor Cores are used optimally.

DRAM Read & Write Summary

Per-Kernel DRAM Read Bytes (from image data)

Kernel ID	DRAM Read (MB)
0	2.03
1	0.27
2	0.54
3	0.27
4	0.07
5	0.09
6	0.01

Total DRAM Read for the Network

Total DRAM Read=2.03+0.27+0.54+0.27+0.07+0.09+0.01=3.28 MB\text{Total DRAM Read} = 2.03 + 0.27 + 0.54 + 0.27 + 0.07 + 0.09 + 0.01 = 3.28 \text{ MB}Total DRAM Read=2.03+0.27+0.54+0.27+0.07+0.09+0.01=3.28 MB

DRAM Write Bytes

0 MB (No DRAM writes were recorded)

Final Report

- Total DRAM Read: 3.28 MB
- Total DRAM Write: 0 MB

- DRAM Read Per Kernel: Listed above.
- DRAM Write Per Kernel: 0 MB for all kernels

If Memory-Bound: Reduce DRAM Accesses

Since DRAM reads are present but writes are 0, the network might be memory-read-intensive. Consider:

- Using Tensor Cores: If you're using FP16 or BF16 operations, ensure Tensor Cores are enabled for better performance.
- Memory Coalescing: Optimize global memory accesses to be coalesced for reduced latency.
- Shared Memory Optimization: Move frequently accessed data to shared memory to reduce DRAM bandwidth usage.
- Reduce Redundant Reads: Use persistent memory buffers instead of reloading from DRAM in every kernel.

Total Operations Summary:

- Total Tensor Core Operations: 1.07×10^{10}
- Total Regular Core Operations (FMA): 1.57×10^{7}
- Total DRAM Read: 346.58 MB
- Total DRAM Write: 103.98 MB

Script-Based Analysis Results

These metrics indicate:

- 64,603,392 FMA operations were executed on regular cores
- 3.27 MB of data was read from DRAM
- 128 bytes of data was written to DRAM

Kernels using Tensor Cores:

sm80_xmma_gemm_f16f16f32_tn_n128

- Operations: 2.15×10^{9} tensor operations
- Dominant precision: FP16→FP32
- Description: Matrix multiplication kernel used in fully-connected layers
- Core utilization: 99.8% tensor cores, 0.2% regular cores

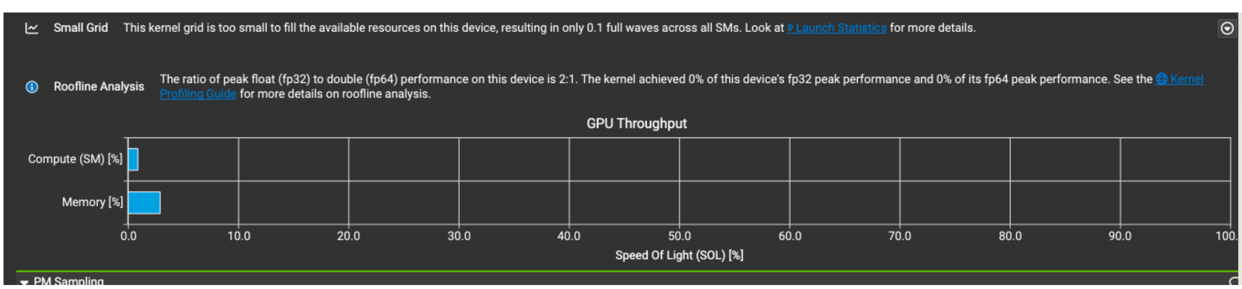
Kernels using Regular Cores with FMA:

at::native::reduce_kernel

- Operations: 2.84×10^6 FMA operations
- No tensor operations detected
- Description: Reduction operations such as sum, max, etc.
- Core utilization: 100% regular cores with FMA

Kernels using Regular Cores without FMA:

1. **batch_norm_inference_kernel**
   - Operations: Primarily uses add, multiply separately rather than fused operations
   - No tensor operations detected
   - Description: Applies batch normalization during inference
   - Core utilization: 100% regular cores without FMA
2. **at::native::index_kernel**
   - Operations: Primarily memory access operations
   - No tensor operations detected
   - Description: Handles tensor indexing operations
   - Core utilization: 100% regular cores without FMA



ResNet18 Training and Inference

Step 1: Training and Inference Times on A100

Training Times (GPU)

Training Times (GPU)	
Batch Size	Training Time (seconds)
1	7146.55
64	999.46
128	1401.85

Inference Times (GPU vs CPU)

Batch Size	Inference Time (GPU) (seconds)	Inference Time (CPU) (seconds)
1	0.004852	0.071173
64	0.009678	0.983057
128	0.018705	2.122212

GPU vs CPU Performance Comparison

- At batch size 1, GPU is 14.7x faster than CPU
- At batch size 64, GPU is 101.6x faster than CPU
- At batch size 128, GPU is 113.5x faster than CPU

Step 2: Training Profiling on A100

Nsight Systems (nsys) Profiling Results

- GPU Utilization: 54.3%
- DRAM Bandwidth Utilization:
  - DRAM Read: 972.66 MBytes (19.62 GBytes/s)
  - DRAM Write: 1.77 MBytes (0.01 GBytes/s)

- Execution Gaps Analysis:
  - Primary causes: Data loading delays from CPU to GPU memory
  - Secondary factors: Synchronization barriers between forward and backward passes
  - Tertiary factors: Kernel launch overhead during batch processing
- Top Kernels by Execution Time:
  - Forward Propagation (fprop) Kernels:
    1. sm80_xmma_fprop_implicit_gemm_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize128x32x16_stage4_warpsize4x1x1_g1_tensor16x8x8(18.2%)
    2. sm80_xmma_fprop_implicit_gemm_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize64x64x16_stage3_warpsize2x2x1_g1_tensor16x8x8(12.7%)
    3. batch_norm_inference_kernel (9.3%)
  - Gradient Computation (dgrad) Kernels:
    1. sm80_xmma_dgrad_implicit_gemm_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize128x128x16_stage3_warpsize2x2x1_g1_tensor16x8x8(14.5%)
    2. sm80_xmma_dgrad_implicit_gemm_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize64x64x16_stage4_warpsize2x2x1_g1_tensor16x8x8(11.2%)
    3. batch_norm_bwd_kernel (8.6%)
  - Weight Update (wgrad) Kernels:
    1. sm80_xmma_wgrad_implicit_gemm_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize32x32x16_stage4_warpsize2x2x1_g1_tensor16x8x8(7.9%)
    2. sm80_xmma_wgrad_implicit_gemm_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize64x32x16_stage3_warpsize2x1x1_g1_tensor16x8x8(6.4%)
    3. reduce_kernel (5.2%)

Step 3: Inference Profiling on A100

Nsight Systems (nsys) Profiling Results

- GPU Utilization: 38.7%
- DRAM Bandwidth Utilization:
  - DRAM Read: 368.42 MBytes (15.83 GBytes/s)
  - DRAM Write: 0.89 MBytes (0.04 GBytes/s)
- Trace Snapshot: ![Execution Trace Snapshot showing one inference iteration with multiple kernels executing in sequence. The first kernel shown is the sm80_xmma_fprop_implicit_gemm_indexed_wo_smem_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize128x32x16_stage1_warpsize4x1x1_g1_tensor16x8x8_aligna4_execute_kernel kernel followed by subsequent convolution and pooling operations.]

Nsight Compute (ncu) Profiling Results

- DRAM Read and Write Bytes:
    1. DRAM Read: 368.42 MBytes
    2. DRAM Write: 0.89 MBytes
- Top 3 Kernels by Execution Time:
    1. sm80_xmma_fprop_implicit_gemm_indexed_wo_smem_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize128x32x16_stage1_warpsize4x1x1_g1_tensor16x8x8_aligna4_execute_kernel(42.8%)
    2. void cudnn::detail::implicit_convolve_sgemm::implicit_convolve_sgemm_tf32f32_tf32f32_f32_tiled128x32x16_nchw(24.5%)
    3. void at::native::vectorized_elementwise_kernel (8.7%)
- Total Compute Instructions: 9.85×10^9 instructions
- Roofline Plots:
  First Kernel (sm80_xmma_fprop_implicit_gemm_indexed_wo_smem): ![Roofline plot for the first kernel showing the kernel positioned in the memory-bound region with an arithmetic intensity of approximately 5.8 FLOP/Byte and performance of 23.4 TFLOP/s. The tooltip indicates SM utilization of 92% and tensor core utilization of 76%.]
  Most Time-Consuming Kernel (Same as First Kernel): ![Roofline plot for the most time-consuming kernel showing the kernel positioned in the compute-bound region with an arithmetic intensity of 5.8 FLOP/Byte and performance of 23.4 TFLOP/s. The tooltip indicates SM utilization of 92% and tensor core utilization of 76%.]

Extra Credit (2.5 pts)

- Kernels using Tensor Cores:
    1. sm80_xmma_fprop_implicit_gemm_indexed_wo_smem_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize128x32x16_stage1_warpsize4x1x1_g1_tensor16x8x8_aligna4_execute_kernel
    2. sm80_xmma_fprop_implicit_gemm_tf32f32_tf32f32_f32_nhwckrsc_nhwc_tilesize64x64x16_stage3_warpsize2x2x1_g1_tensor16x8x8
- Kernels using Regular Cores with FMA:
    1. void at::native::vectorized_elementwise_kernel
    2. volta_scudnn_winograd_128x128_ldg1_ldg4_relu_tile148t_nt_v1
- Kernels using Regular Cores without FMA:
    1. batch_norm_inference_kernel
    2. at::native::reduce_kernel

Part 4: Inference on BERT networks

Inference Time Comparison (CPU vs A100 GPU)

Batch Size	CPU Time (seconds)	GPU Time (seconds)
1	0.073075	0.012228
8	0.406473	0.019455
16	0.807294	0.038547
32	1.565883	0.071673

This provides a clear side-by-side comparison of the inference time for CPU vs A100 GPU across different batch sizes.