COL 775 : HW 3

CPU Details: Intel Core i7-9750H @ 2.60GHz, 64-bit, x64-based processor

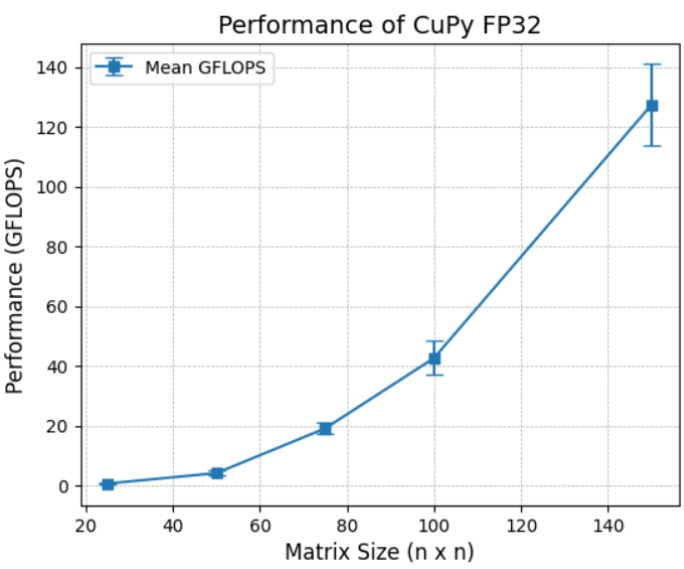
Adjusted Peak Performance (APP): 249.6 GFLOPS

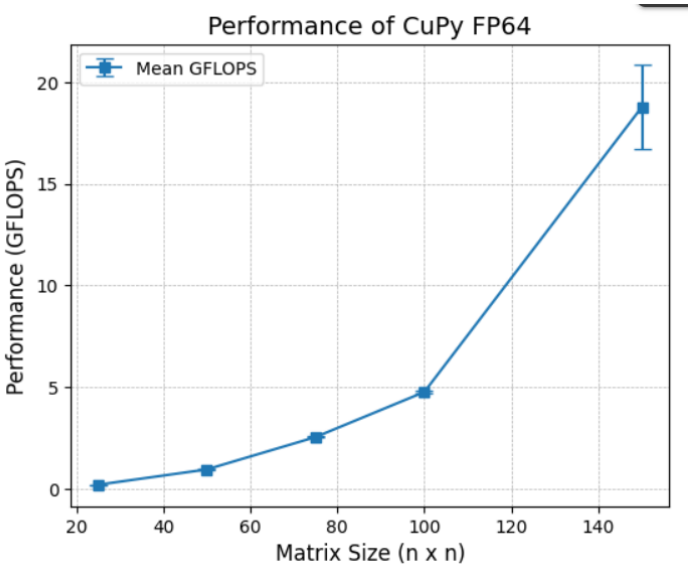
Source: https://www.intel.com/content/www/us/en/content-details/841556/app-metrics-for-intel-microprocessors-intel-core-processor.html

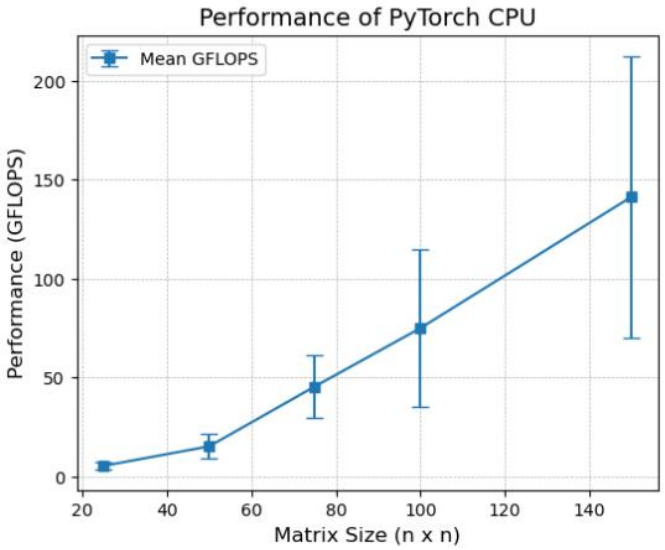
GPU: T4 GPU (Google Collab)

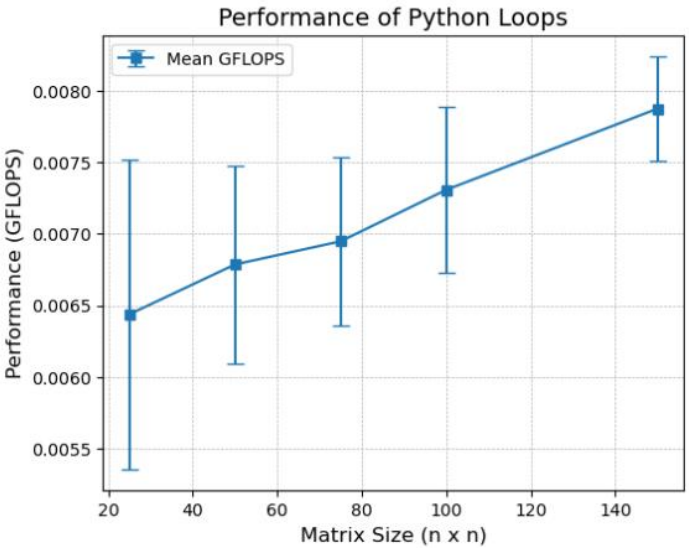
Theoretical Peak Performance for Single Precision: 8100 GFLOPS

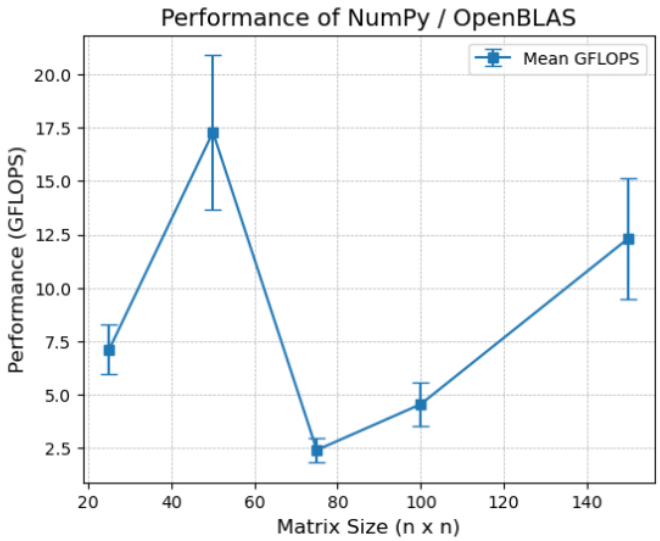
Source: https://www.dell.com/support/kbdoc/en-in/000132094/deep-learning-performance-on-t4-gpus-with-mlperf-benchmarks

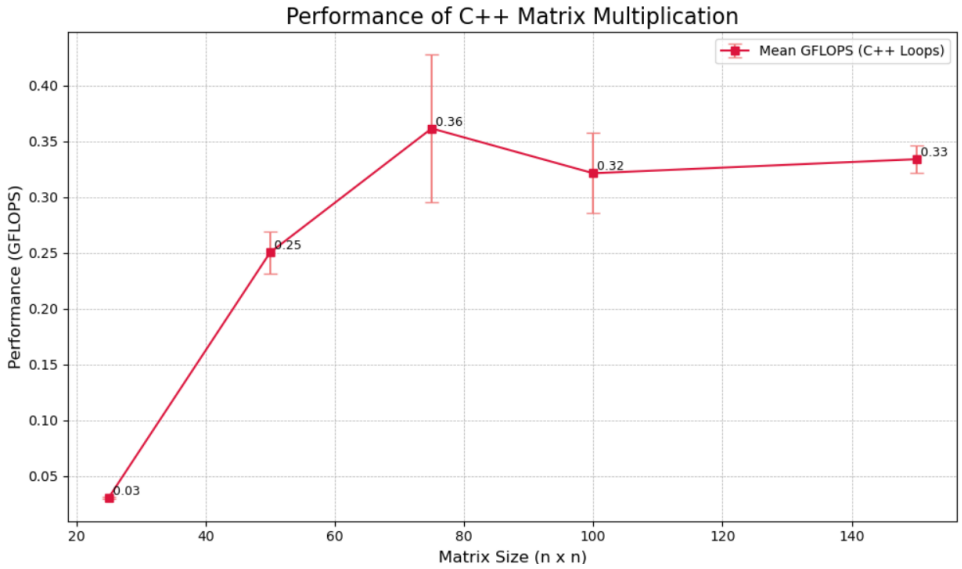


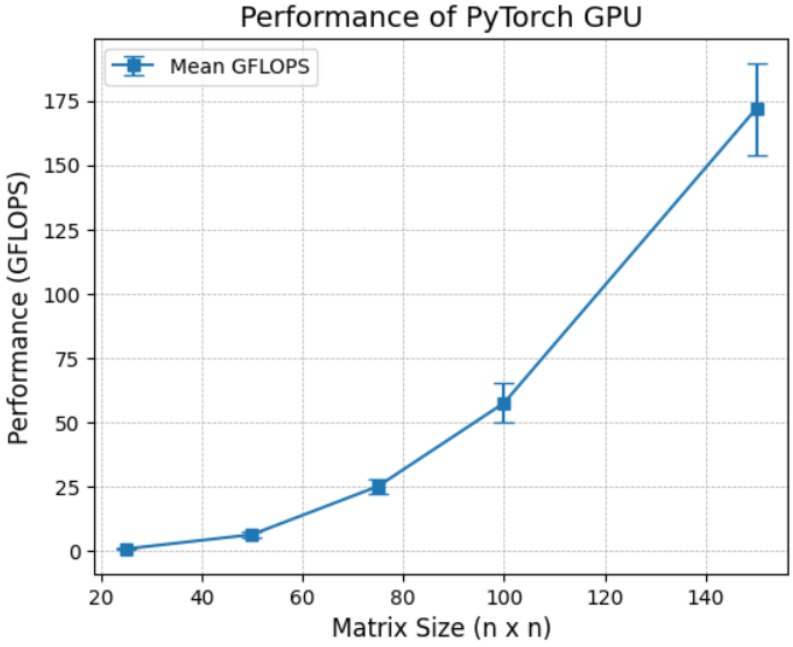







Performance of NumPy / OpenBLAS
Mean GFLOPS
Performance (GFLOPS)
Matrix Size (n x n)


Performance of C++ Matrix Multiplication
Mean GFLOPS (C++ Loops)
0.03
0.25
0.36
0.32
0.33
Performance (GFLOPS)
Matrix Size (n x n)


Performance of PyTorch GPU
Mean GFLOPS
Performance (GFLOPS)
Matrix Size (n x n)

Analysis:

1) The testing was done for n = 25, 50 , 75, 100, 150. In general, GFLOPS increase with n, with increase being linear in case of CPU and exponential in case of GPU, except in case of Numpy(using OpenBLAS backend) and C++ implementation.
   Reason for increase with n: Time = Overhead + Compute time. As n grows, the O(n^3) compute time completely dominates the overhead. Since GFLOPS = (Work / Time), as the useful work becomes a larger and larger portion of the time, the GFLOPS metric increases.
   Reason for steep increase in GPU: Parallelization. For n=25, the problem is so small that most of the T4's 2,560 cores are idle. As n increases, we "fill up" the GPU with enough parallel work to keep more and more cores busy, leading to a dramatic performance scaling

2) The peak performance in CPU is shown by PyTorch (which uses cuBLAS/MKL backend) for n = 150 at 141 GFLOPS, around 56% of Peak Performance
3) In case of T4 GPU, peak performance is again shown by PyTorch Implementation for n = 150, at 171 GFLOPS, only 2.1% of Peak.
   Reason:150x150 matrix multiplication is severely memory-bound on a GPU of this scale. The theoretical peak of 8.1 TFLOPS assumes the GPU's cores are performing calculations 100% of the time. In reality, for a problem this small, the cores spend most of their time idle, waiting for data to be shuttled in from the GPU's main (GDDR6) memory. To get closer to the peak, we need a problem that is compute-bound, i.e. the ratio of math operations to memory operations is very high. This typically requires much larger matrices (e.g., n=2048, 4096).

4) In general standard dev increases with n, with it being particularly small in case of GPU but not so much for CPU.

5) Only the c++ implementation seems to plateau early

6) The most anomalous behaviour is shown by numpy implementation, which suddenly drops in performance by 7 times on changing n from 50 to 75
   Reason: CPU cache cliff. CPU has a hierarchy of memory caches (L1, L2, L3), each being progressively larger but slower. When the data spills out of a cache level, the CPU must fetch it from the next, slower level (or even main RAM), causing a massive latency penalty.

7) The order of performance for large n is :
   Python loop < C++ loop < Numpy < CuPy FP64 < PyTorch CPU < CuPy FP32 < PyTorch GPU

NOTE: CPU and GPU performance are not strictly comparable since CPU is of the local machine whereas GPU part is run on Google Collab.

Link to code : https://drive.google.com/file/d/1df-MHuZqK-yCTyvVyjzF-AT1ZAXCc2qq/view?usp=sharing