Deep Learning - COL775/COL7375

Homework 3

1 Objective

The goal of this assignment is to implement, benchmark, and analyze the performance of matrix-matrix multiplication using various computational approaches. You will measure the performance in Giga Floating-point Operations Per Second (GFLOPS).

2 System Specification

In your final report, you must provide the details of the hardware you used for this homework.

1. **CPU Details:** Specify the exact model and architecture of your CPU (e.g., Intel Core i7-9750H x64, M4 ARM).
2. **GPU Details:** Specify the exact model of your GPU (e.g., NVIDIA GeForce RTX 3050).
3. **Theoretical Peak FLOPS:** Consult the official manufacturer documentation and report the theoretical single-precision (FP32) peak GFLOPS for your CPU and GPU. For CPUs, this might be based on AVX instructions. Provide a link to the source in your pdf.

3 Tasks

You are required to implement and benchmark matrix multiplication for square matrices ($n \times n$) using the following methods.

3.1 Task 1: Naive Python Implementation

Implement matrix multiplication using standard, nested `for` loops in Python without using any optimization strategies.

3.2 Task 2: Optimized CPU Implementation (NumPy)

Use the optimized matrix multiplication function from the NumPy library.

3.3 Task 3: GPU Implementation (CuPy)

Use the CuPy library to perform matrix multiplication on the GPU. You must benchmark this in two separate modes:

1. **Single Precision:** Ensure your matrices are of type `float32`.
2. **Double Precision:** Ensure your matrices are of type `float64`.

3.4 Task 4: Naive C Implementation

Implement matrix multiplication using standard, nested `for` loops in the C programming language. Compile your code without any optimization flags.

4 Experimental Methodology

For each task, you must follow this procedure to ensure consistent and reliable results.

1. **Matrix Sizes:** Benchmark your implementations for a range of square matrix sizes n (Atleast 5 different values of n). You may need to adjust this range based on the amount of compute you have available (e.g., the Python loop will be too slow for large n). You must keep the chosen values of n consistent across the different tasks. Try to keep n on the higher side to get more pronounced differences.

2. **Statistical Analysis:** To get reliable timing, run each matrix multiplication (for a given implementation and size n) in a loop for **30 iterations** to get statistically meaningful timing results. Record the execution time for each of these 30 runs. Calculate the GFLOPS for each run. From the 30 GFLOPS values you obtain for a given n, calculate the **mean GFLOPS** and the **standard deviation**.

5 Reporting and Visualization

Your final submission should be a PDF report containing your analysis and results.

1. **Graphs:** You must generate a **minimum of 5 separate graphs**.
   - Graph 1: Python Loop Performance
   - Graph 2: NumPy Performance
   - Graph 3: CuPy Single Precision (FP32) Performance
   - Graph 4: CuPy Double Precision (FP64) Performance
   - Graph 5: C-Loop Performance

2. **Graph Specifications:** For each graph:
   - The **X-axis** should be the Matrix Size (n).
   - The **Y-axis** should be the Performance (Mean GFLOPS).
   - Include **error bars** on each data point to show the standard deviation.

3. **Analysis:** For each graph, provide a brief analysis of the results. Compare the achieved performance to the theoretical peak performance of the relevant hardware (CPU for Python/NumPy/C, GPU for CuPy). Discuss any interesting observations, such as performance scaling with matrix size, the difference between single and double precision on the GPU, and the performance gap between naive and optimized libraries. Add a link for the code that you used as well.

6 Bonus Marks

For bonus marks, you are encouraged to implement and benchmark additional strategies. Each bonus implementation must be accompanied by its own graph and analysis, following the same methodology as described above.