**Problem Statement:** Matrix Multiplication using CUDA C

**Google Colab Platform Code:**

**Command to install nvidia gpu on google colab:**  !nvidia-smi

import numpy as np

import cupy as cp

import time

# Size of matrices

N = 1024

# Create random matrices on CPU

A\_cpu = np.random.rand(N, N).astype(np.float32)

B\_cpu = np.random.rand(N, N).astype(np.float32)

# ---------- CPU ----------

start\_cpu = time.time()

C\_cpu = np.matmul(A\_cpu, B\_cpu)

end\_cpu = time.time()

print(f"CPU Matrix Multiplication Time: {end\_cpu - start\_cpu:.4f} seconds")

# ---------- GPU ----------

A\_gpu = cp.asarray(A\_cpu)

B\_gpu = cp.asarray(B\_cpu)

start\_gpu = time.time()

C\_gpu = cp.matmul(A\_gpu, B\_gpu)

cp.cuda.Device(0).synchronize()

end\_gpu = time.time()

print(f"GPU Matrix Multiplication Time: {end\_gpu - start\_gpu:.4f} seconds")

**Command to run the shells in google colab:**

Shift enter

**Output:**  
CPU Matrix Multiplication Time: 0.0516 seconds

GPU Matrix Multiplication Time: 0.6664 seconds

**LeetGPU Platform Code:**

**// matrix\_mul.cu**

#include <stdio.h>

\_\_global\_\_ void matMul(float \*A, float \*B, float \*C, int N) {

int row = blockIdx.y \* blockDim.y + threadIdx.y;

int col = blockIdx.x \* blockDim.x + threadIdx.x;

if (row < N && col < N) {

float sum = 0;

for (int k = 0; k < N; k++)

sum += A[row \* N + k] \* B[k \* N + col];

C[row \* N + col] = sum;

}

}

int main() {

int N = 2; // 2x2 matrix for simplicity

size\_t size = N \* N \* sizeof(float);

float A[] = {1, 2, 3, 4};

float B[] = {5, 6, 7, 8};

float C[4];

float \*d\_A, \*d\_B, \*d\_C;

cudaMalloc(&d\_A, size);

cudaMalloc(&d\_B, size);

cudaMalloc(&d\_C, size);

cudaMemcpy(d\_A, A, size, cudaMemcpyHostToDevice);

cudaMemcpy(d\_B, B, size, cudaMemcpyHostToDevice);

dim3 threads(16, 16);

dim3 blocks((N+15)/16, (N+15)/16);

matMul<<<blocks, threads>>>(d\_A, d\_B, d\_C, N);

cudaMemcpy(C, d\_C, size, cudaMemcpyDeviceToHost);

printf("Result Matrix C:\n");

for (int i = 0; i < N\*N; i++) {

printf("%f ", C[i]);

if ((i+1) % N == 0) printf("\n");

}

cudaFree(d\_A);

cudaFree(d\_B);

cudaFree(d\_C);

return 0;

}

Output:

Running NVIDIA GTX TITAN X in FUNCTIONAL mode...

Compiling...

Executing...

Result Matrix C:

19.000000 22.000000

43.000000 50.000000

Exit status: 0