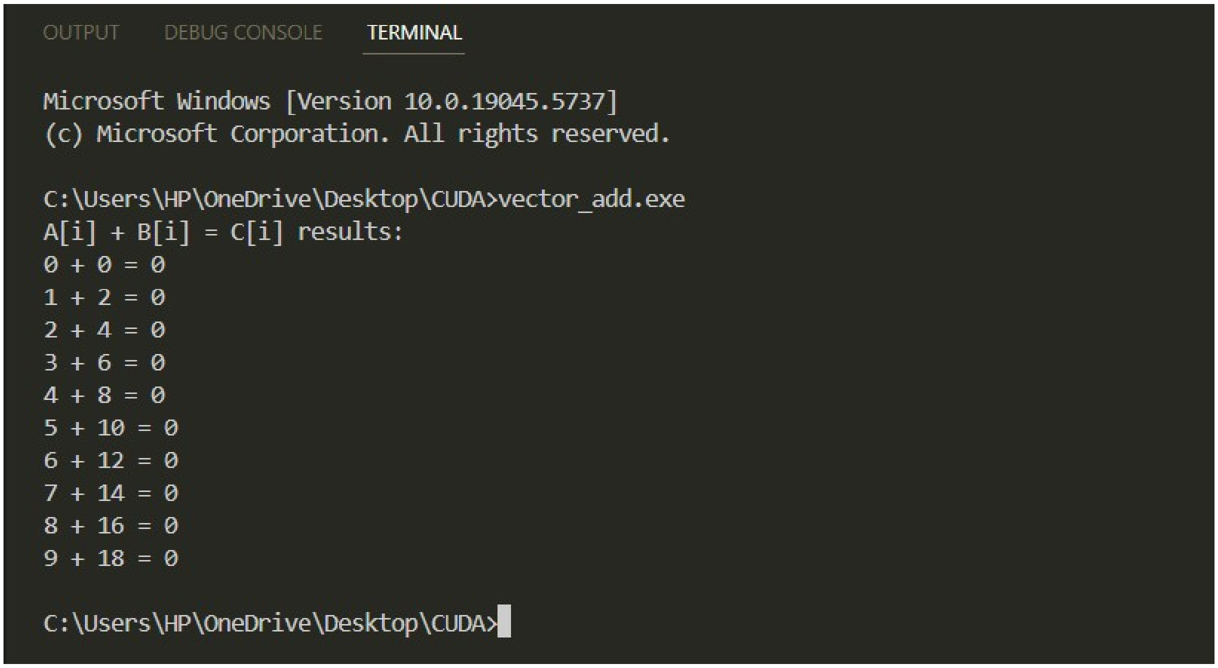
Problem Statement: Write a CUDA Program for:

1. Addition of two large vectors
2. Matrix Multiplication using CUDA C

Output-1 (Addition of two large vectors)

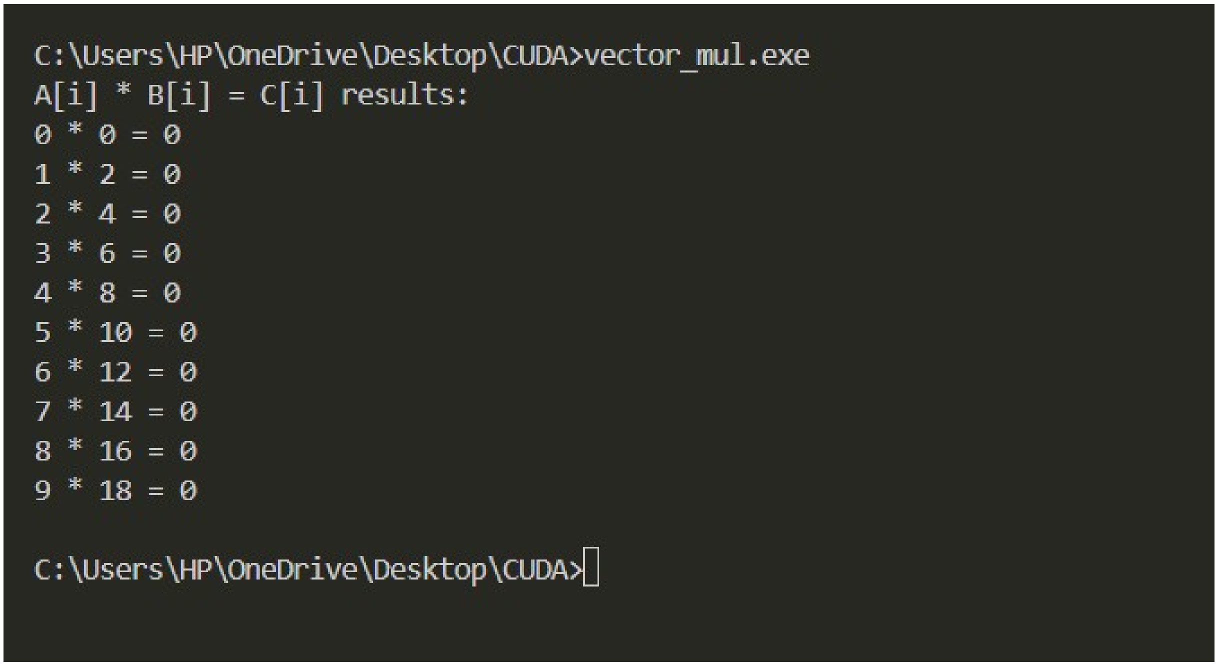
```
#include <iostream>
#include <cuda_runtime.h>

__global__ void vectorAdd(float* A, float* B, float* C, int N) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}

int main() {
    int N = 10;  // Keep it small so we can print results
    size_t size = N * sizeof(float);

    float *h_A = new float[N];
    float *h_B = new float[N];
    float *h_C = new float[N];

    //  Initialize  input  vectors
    for (int i = 0; i < N; i++) {
        h_A[i] = i * 1.0f;
        h_B[i] = i * 2.0f;
    }

    float *d_A, *d_B, *d_C;
    cudaMalloc((void**)&d_A, size);
    cudaMalloc((void**)&d_B, size);
    cudaMalloc((void**)&d_C, size);

    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

    vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
    cudaDeviceSynchronize();
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    // Print full result
    std::cout << “A[i] + B[i] = C[i] results:\n”;
    for (int I = 0; I < N; i++) {
        std::cout << h_A[i] << “ + “ << h_B[i] << “ = “ << h_C[i] << std::endl;
    }

    // Cleanup
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    delete[] h_A;
    delete[] h_B;
    delete[] h_C;

    return 0;
}
```



```
Microsoft Windows [Version 10.0.19045.5737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP\OneDrive\Desktop\CUDA>vector_add.exe
A[i] + B[i] = C[i] results:
0 + 0 = 0
1 + 2 = 0
2 + 4 = 0
3 + 6 = 0
4 + 8 = 0
5 + 10 = 0
6 + 12 = 0
7 + 14 = 0
8 + 16 = 0
9 + 18 = 0

C:\Users\HP\OneDrive\Desktop\CUDA>
```

Output-2 (Matrix Multiplication using CUDA C)

```
#include <iostream>
#include <cuda_runtime.h>

_global___void vectorMul(float* A, float* B, float* C, int N) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] * B[i];
}

int main() {
    int N = 10;  // Keep it small so we can print results
    size_t size = N * sizeof(float);

    float *h_A = new float[N];
    float *h_B = new float[N];
    float *h_C = new float[N];

    //  Initialize  input  vectors
    for (int i = 0; i < N; i++) {
        h_A[i] = i * 1.0f;
        h_B[i] = i * 2.0f;
    }

    float *d_A, *d_B, *d_C;
    cudaMalloc((void**)&d_A, size);
    cudaMalloc((void**)&d_B, size);
    cudaMalloc((void**)&d_C, size);

    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

    vectorMul<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
    cudaDeviceSynchronize();

    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    // Print full result
    std::cout << "A[i] * B[i] = C[i] results:\n";
    for (int i = 0; i < N; i++) {
        std::cout << h_A[i] << " * " << h_B[i] << " = " << h_C[i] << std::endl;
    }

    // Cleanup
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    delete[] h_A;
    delete[] h_B;
    delete[] h_C;

    return 0;
}
```



```
C:\Users\HP\OneDrive\Desktop\CUDA>vector_mul.exe
A[i] * B[i] = C[i] results:
0 * 0 = 0
1 * 2 = 0
2 * 4 = 0
3 * 6 = 0
4 * 8 = 0
5 * 10 = 0
6 * 12 = 0
7 * 14 = 0
8 * 16 = 0
9 * 18 = 0

C:\Users\HP\OneDrive\Desktop\CUDA>
```