#include <iostream>

#include <cuda\_runtime.h>

using namespace std;

#define N 16 // Matrix size (N x N)

// CUDA Kernel for Matrix Multiplication

\_\_global\_\_ void matrixMul(int \*A, int \*B, int \*C, int width) {

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

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

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

int sum = 0;

for (int k = 0; k < width; k++) {

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

}

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

}

}

int main() {

int size = N \* N \* sizeof(int);

int h\_A[N \* N], h\_B[N \* N], h\_C[N \* N];

// Initialize matrices A and B

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

h\_A[i] = rand() % 10;

h\_B[i] = rand() % 10;

}

// Allocate memory on GPU

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

cudaMalloc((void\*\*)&d\_A, size);

cudaMalloc((void\*\*)&d\_B, size);

cudaMalloc((void\*\*)&d\_C, size);

// Copy data to device

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

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

// Define block and grid dimensions

dim3 threadsPerBlock(16, 16);

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

// Launch kernel

matrixMul<<<blocksPerGrid, threadsPerBlock>>>(d\_A, d\_B, d\_C, N);

// Copy result back to host

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

// Print sample result

cout << "Sample Result: " << h\_C[0] << " " << h\_C[N \* N - 1] << endl;

// Free memory

cudaFree(d\_A);

cudaFree(d\_B);

cudaFree(d\_C);

return 0;

}

1. **Matrix Initialization**
   * Fills matrices A and B with random numbers.

**2.Memory Allocation on GPU**  
cudaMalloc((void\*\*)&d\_A, size);

* + Allocates space for matrices on the **device (GPU)**.

**3.Kernel Execution**  
matrixMul<<<blocksPerGrid, threadsPerBlock>>>(d\_A, d\_B, d\_C, N);

* + Uses **CUDA threads** for parallel computation

**4. Copying Data Back**  
cudaMemcpy(h\_C, d\_C, size, cudaMemcpyDeviceToHost);

* + Transfers the result matrix **back to CPU memory**.

### **Compiling and Running the Program**

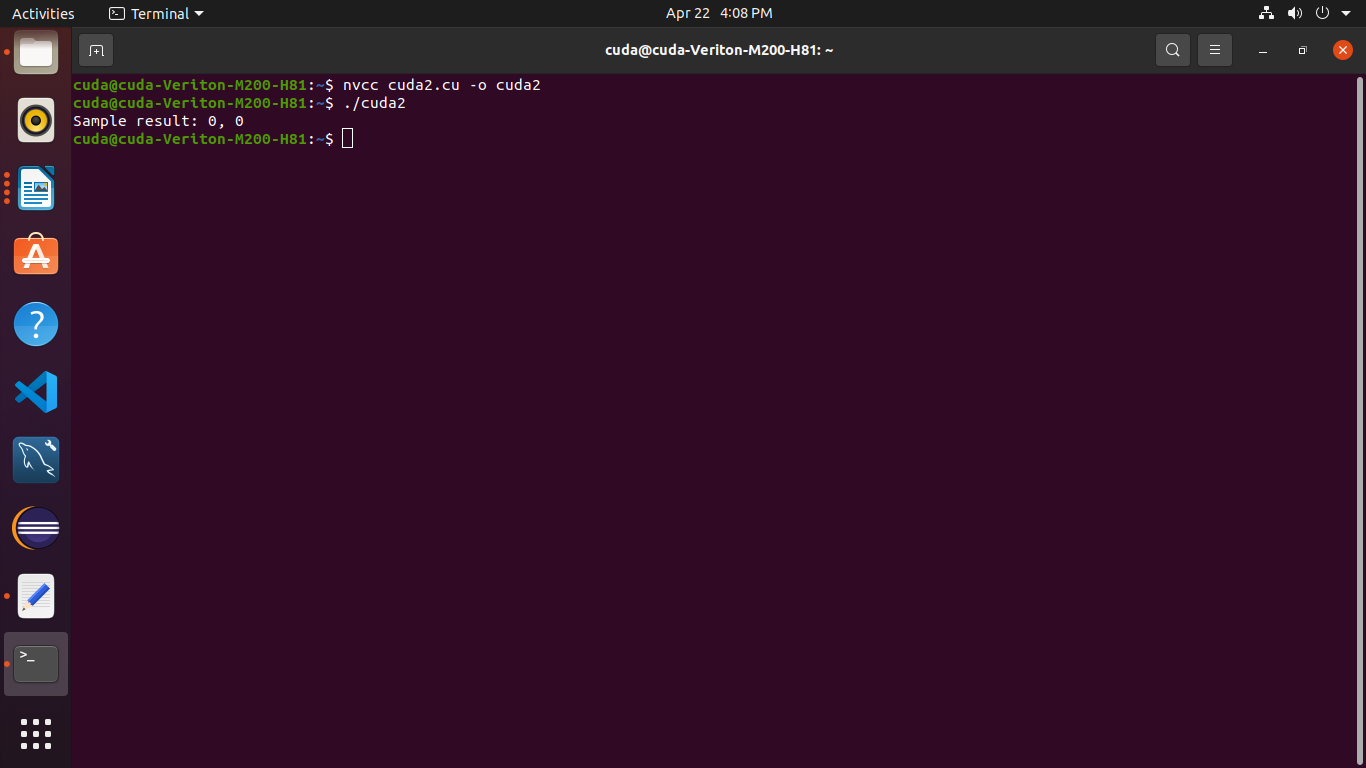
#### **Compile:**

nvcc cuda\_matrix\_multiplication.cu -o cuda\_matrix\_multiplication

#### **Run:**

./cuda\_matrix\_multiplication

**Output:**

****