**#include <iostream>**

**#include <cuda\_runtime.h>**

**using namespace std;**

**// CUDA Kernel for vector addition**

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

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

**if (i < N) {**

**C[i] = A[i] + B[i];**

**}**

**}**

**int main() {**

**int N = 1000000; // Size of vectors**

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

**// Allocate host memory**

**int \*h\_A = (int\*)malloc(size);**

**int \*h\_B = (int\*)malloc(size);**

**int \*h\_C = (int\*)malloc(size);**

**// Initialize vectors**

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

**h\_A[i] = i;**

**h\_B[i] = i \* 2;**

**}**

**// Allocate device memory**

**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 size**

**int threadsPerBlock = 256;**

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

**// Launch kernel**

**vectorAdd<<<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-1] << endl;**

**// Free memory**

**free(h\_A); free(h\_B); free(h\_C);**

**cudaFree(d\_A); cudaFree(d\_B); cudaFree(d\_C);**

**return 0;**

**}**

**1.CUDA Kernel (vectorAdd)**  
\_\_global\_\_ void vectorAdd(int \*A, int \*B, int \*C, int N) { ... }

* + Each thread **computes one element** of the sum.

**2.Memory Allocation**  
int \*d\_A, \*d\_B, \*d\_C;

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

* + Allocates memory on **GPU (device memory)**.

**3.Memory Transfer (Host ↔ Device)**  
cudaMemcpy(d\_A, h\_A, size, cudaMemcpyHostToDevice);

* + Copies vectors from **CPU to GPU**.

**4.Kernel Launch**  
vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d\_A, d\_B, d\_C, N);

* + Uses **CUDA parallel execution**.

**5.Copy Result Back & Free Memory**  
cudaMemcpy(h\_C, d\_C, size, cudaMemcpyDeviceToHost);

* + Transfers result from **GPU to CPU**.

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

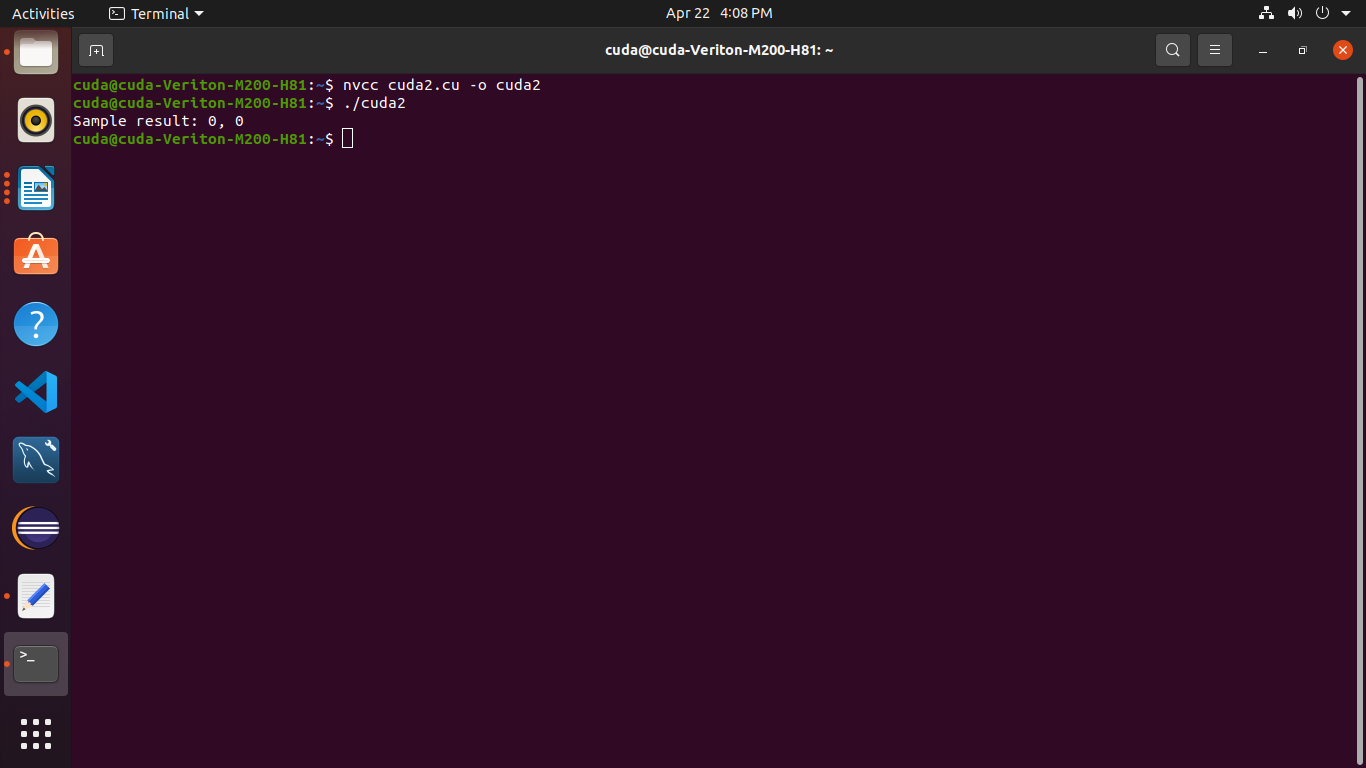
### **Compile:**

nvcc cuda\_vector\_addition.cu -o cuda\_vector\_addition

### **Run:**

./cuda\_vector\_addition

**Output:**

****