```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Thu_Jun__6_02:18:23_PDT_2024
Cuda compilation tools, release 12.5, V12.5.82
Build cuda_12.5.r12.5/compiler.34385749_0
```

```
!nvidia-smi
```

```
Tue May  6 13:43:53 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   40C    P8              9W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

```
%%writefile vector.cu

#include <bits/stdc++.h>
#include <cuda_runtime.h>
using namespace std;
using namespace std::chrono;

__global__ void add(int* A, int* B, int* C, int size){
  int tid = blockIdx.x * blockDim.x + threadIdx.x;

  if(tid < size){
    C[tid] = A[tid] + B[tid];
  }
}

void initialize(int* vector, int size){
  for(int i=0; i<size; i++){
    cout << "Enter element " << i+1 << " of the vector: ";
    cin >> vector[i];
  }
  cout << endl;
}

void print(int* vector, int size){
  for(int i=0; i<size; i++){
    cout << vector[i] << " ";
  }
  cout << endl;
}

void sequentialAddition(int* A, int* B, int* C, int size){
  for(int i=0; i<size; i++){
    C[i] = A[i] + B[i];
  }
}

int main(){
  int N;
  cout << "Enter the size of vectors: ";
  cin >> N;

  int *A, *B, *C;

  int vectorSize = N;
  size_t vectorBytes = vectorSize * sizeof(int);

  A = new int[vectorSize];
  B = new int[vectorSize];
  C = new int[vectorSize];

  initialize(A, vectorSize);
```

```
  initialize(B, vectorSize);

  cout << "Vector A: "; print(A, vectorSize);
  cout << "Vector B: "; print(B, vectorSize);

  int *X, *Y, *Z;
  cudaMalloc(&X, vectorBytes);
  cudaMalloc(&Y, vectorBytes);
  cudaMalloc(&Z, vectorBytes);

  cudaMemcpy(X, A, vectorBytes, cudaMemcpyHostToDevice);
  cudaMemcpy(Y, B, vectorBytes, cudaMemcpyHostToDevice);

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

  // Sequential Addition

  auto start = high_resolution_clock::now();
  sequentialAddition(A, B, C, N);
  auto stop = high_resolution_clock::now();
  auto seq_duration = duration_cast<microseconds>(stop-start);

  cout << "Sequential Addition: "; print(C, N);

  // Parallel Addition

  start = high_resolution_clock::now();
  add<<<blocksPerGrid, threadsPerBlock>>>(X,Y,Z,N);
  cudaDeviceSynchronize();
  cudaMemcpy(C, Z, vectorBytes, cudaMemcpyDeviceToHost);
  stop = high_resolution_clock::now();
  auto par_duration = duration_cast<microseconds>(stop-start);

  cout << "Parallel Addition: "; print(C, N);

  cout << "Sequential Addition Time: "<< seq_duration.count() <<" microseconds."<<endl;
  cout << "Parallel Addition Time: "<< par_duration.count() <<" microseconds."<<endl;

  delete []A;
  delete []B;
  delete []C;

  cudaFree(X);
  cudaFree(Y);
  cudaFree(Z);

  return 0;
}
```

```
Writing vector.cu
```

```
!nvcc -arch=sm_75 vector.cu -o vec
```

```
!./vec
```

```
Enter the size of vectors: 3
    Enter element 1 of the vector: 1
    Enter element 2 of the vector: 2
    Enter element 3 of the vector: 3

    Enter element 1 of the vector: 4
    Enter element 2 of the vector: 5
    Enter element 3 of the vector: 6

    Vector A: 1 2 3
    Vector B: 4 5 6
    Sequential Addition: 5 7 9
    Parallel Addition: 5 7 9
    Sequential Addition Time: 0 microseconds.
    Parallel Addition Time: 165 microseconds.
```

```
%%writefile matrix.cu

#include <bits/stdc++.h>
#include <cuda_runtime.h>

using namespace std;
using namespace std::chrono;

__global__ void multiply(int *A, int *B, int *C, int M, int N, int K){
  int row = blockIdx.y * blockDim.y + threadIdx.y;
```

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

    if(row<M && col<K){
      int sum = 0;
      for(int i=0; i<N; i++){
        sum += A[row * N + i] * B[K * i + col];
      }
      C[row * K + col] = sum;
    }
  }

  void initialize(int *matrix, int rows, int cols){
    for(int i=0; i< rows*cols; i++){
      cout << "Enter element " << i+1 << " : ";
      cin >> matrix[i];
    }
  }

  void print(int *matrix, int rows, int cols){
    for(int row=0; row<rows; row++){
      for(int col=0; col<cols; col++){
        cout << matrix[row * cols + col] << " ";
      }
      cout << endl;
    }
    cout << endl;
  }

  void sequentialMultiply(int *A, int *B, int *C, int M, int N, int K){
    for(int i=0; i<M; i++){
      for(int j=0; j<K; j++){
        int sum = 0;
        for(int k=0; k<N; k++){
          sum += A[i*N+k] * B[k*K+j];
        }
        C[i*K+j] = sum;
      }
    }
  }

  int main(){
    int M,N,K;
    cout << "Enter number of rows and columns of first matrix: ";
    cin >> M >> N;
    cout << "Enter number of columns of second matrix: ";
    cin >> K;

    int *A, *B, *C;

    A = new int[M*N];
    B = new int[N*K];
    C = new int[M*K];

    initialize(A, M, N);
    initialize(B, N, K);
    cout<<"Matrix A:"<<endl; print(A, M, N);
    cout<<"Matrix B:"<<endl; print(B, N, K);

    int *X, *Y, *Z;

    cudaMalloc(&X, M*N*sizeof(int));
    cudaMalloc(&Y, N*K*sizeof(int));
    cudaMalloc(&Z, M*K*sizeof(int));

    cudaMemcpy(X,A,M*N*sizeof(int),cudaMemcpyHostToDevice);
    cudaMemcpy(Y,B,N*K*sizeof(int),cudaMemcpyHostToDevice);

    int THREADS = 16;
    int BLOCKS = (M + THREADS - 1) / THREADS;

    dim3 threads(THREADS, THREADS);
    dim3 blocks(BLOCKS, BLOCKS);

    // Sequential multiplication
      auto start = high_resolution_clock::now();
      sequentialMultiply(A, B, C, M, N, K);
      auto stop = high_resolution_clock::now();
      auto seq_duration = duration_cast<microseconds>(stop - start);

      cout << "Sequential Multiplication of matrix A and B: \n";
      print(C, M, K);

       // Parallel multiplication
```

```
    start = high_resolution_clock::now();
    multiply<<<blocks, threads>>>(X, Y, Z, M, N, K);
    cudaMemcpy(C, Z, M * K * sizeof(int), cudaMemcpyDeviceToHost);
    stop = high_resolution_clock::now();
    auto par_duration = duration_cast<microseconds>(stop - start);

    cout << "Parallel Multiplication of matrix A and B: \n";
    print(C, M, K);

    cout << "Sequential Multiplication Time: " << seq_duration.count() << " microseconds" << endl;
    cout << "Parallel Multiplication Time: " << par_duration.count() << " microseconds" << endl;

    delete[] A;
    delete[] B;
    delete[] C;

    cudaFree(X);
    cudaFree(Y);
    cudaFree(Z);

    return 0;
}
```

```
Overwriting matrix.cu
```

```
!nvcc -arch=sm_75 matrix.cu -o mat
```

```
!./mat
```

```
Enter number of rows and columns of first matrix: 2
3
Enter number of columns of second matrix: 1
Enter element 1 : 2
Enter element 2 : 3
Enter element 3 : 4
Enter element 4 : 5
Enter element 5 : 6
Enter element 6 : 7
Enter element 1 : 1
Enter element 2 : 2
Enter element 3 : 3
Matrix A:
2 3 4
5 6 7

Matrix B:
1
2
3

Sequential Multiplication of matrix A and B:
20
38

Parallel Multiplication of matrix A and B:
20
38

Sequential Multiplication Time: 0 microseconds
Parallel Multiplication Time: 130 microseconds
```