**Source Code:**

#include<iostream>

#include<bits/stdc++.h>

#include<cuda.h> #define BLOCK\_SIZE 16 using namespace std;

void initialize\_matrix(int \*array, int rows, int cols){ for(int i = 0 ; i < rows; i++){ for(int j = 0; j < cols; j++){ array[i\*cols + j] = rand() % 10;

}

} }

void print\_matrix(int \*array, int rows, int cols){ for(int i = 0 ; i < rows; i++){ for(int j = 0; j < cols; j++){ cout << array[i\*cols + j] << " ";

}

cout << endl;

} }

void matrix\_multiplication\_cpu(int \*a, int \*b, int \*c, int common, int c\_rows,int c\_cols){ for(int i = 0; i < c\_rows; i++){ for(int j = 0; j < c\_cols; j++){ int sum = 0;

for(int k = 0; k < common; k++){ sum += a[i\*common + k] \* b[k\*c\_cols + j];

}

c[i\*c\_cols + j] = sum;

}

}

}

\_\_global\_\_ void matrix\_multiply(int \*a, int \*b, int \*c, int c\_rows, int common, int c\_cols) {

int row = blockIdx.y\*blockDim.y + threadIdx.y; int col = blockIdx.x\*blockDim.x + threadIdx.x; int sum=0;

if(col < c\_cols && row < c\_rows) {

for(int j = 0 ;j < common;j++)

{

sum += a[row\*common+j] \* b[j\*c\_cols+col];

}

c[c\_cols\*row+col]=sum;

} } int main(){

int A\_rows, A\_cols, B\_rows, B\_cols, C\_rows, C\_cols; cout << "Dimensions of matrix 1:\n"; cout << "Rows: "; cin >> A\_rows; cout << "Columns: "; cin >> A\_cols;

cout << "Dimensions of matrix 2:\n"; cout << "Rows: " << A\_cols << endl << "Columns: "; cin >> B\_cols; B\_rows = A\_cols;

C\_rows = A\_rows; C\_cols = B\_cols; int A\_size = A\_rows \* A\_cols; int B\_size = B\_rows \* B\_cols; int C\_size = C\_rows \* C\_cols; int \*A, \*B, \*C; int \*m1,\*m2,\*result; A = new int[A\_size];

B = new int[B\_size]; C = new int[C\_size]; initialize\_matrix(A,A\_rows,A\_cols); cout << "Matrix 1\n"; print\_matrix(A,A\_rows,A\_cols); initialize\_matrix(B,B\_rows,B\_cols); cout << "Matrix 2\n"; print\_matrix(B,B\_rows,B\_cols); cudaMallocManaged(&m1, A\_size \* sizeof(int)); cudaMallocManaged(&m2, B\_size \* sizeof(int)); cudaMallocManaged(&result, C\_size \* sizeof(int)); cudaMemcpy(m1,A,A\_size \* sizeof(int), cudaMemcpyHostToDevice); cudaMemcpy(m2,B,B\_size \* sizeof(int), cudaMemcpyHostToDevice); dim3 dimGrid(A\_rows + BLOCK\_SIZE - 1 / BLOCK\_SIZE, B\_cols + BLOCK\_SIZE - 1 / BLOCK\_SIZE); dim3 dimBlock(BLOCK\_SIZE,BLOCK\_SIZE); float gpu\_elapsed\_time; cudaEvent\_t gpu\_start,gpu\_stop; cudaEventCreate(&gpu\_start); cudaEventCreate(&gpu\_stop); cudaEventRecord(gpu\_start);

matrix\_multiply<<<dimGrid,dimBlock>>>(m1,m2,result,C\_rows,A\_cols,C\_cols

);

cudaEventRecord(gpu\_stop); cudaEventSynchronize(gpu\_stop);

cudaEventElapsedTime(&gpu\_elapsed\_time, gpu\_start, gpu\_stop); cudaEventDestroy(gpu\_start); cudaEventDestroy(gpu\_stop);

cudaMemcpy(C,result,C\_size\*sizeof(int),cudaMemcpyDeviceToHost); cout << "GPU result:\n"; print\_matrix(C,C\_rows,C\_cols);

cout<<"GPU Elapsed time is: "<<gpu\_elapsed\_time<<" milliseconds"<<endl; cudaEventCreate(&gpu\_start); cudaEventCreate(&gpu\_stop); cudaEventRecord(gpu\_start);

matrix\_multiplication\_cpu(A,B,C,A\_cols,C\_rows,C\_cols); cudaEventRecord(gpu\_stop); cudaEventSynchronize(gpu\_stop);

cudaEventElapsedTime(&gpu\_elapsed\_time, gpu\_start, gpu\_stop); cudaEventDestroy(gpu\_start); cudaEventDestroy(gpu\_stop); cout << "CPU result:\n"; print\_matrix(C,C\_rows,C\_cols);

cout<<"CPU Elapsed time is: "<<gpu\_elapsed\_time<<" milliseconds"<<endl; cudaFree(m1); cudaFree(m2); cudaFree(result); return 0; }