```
# Set up CUDA
#First Change runtime to GPU and run this cell
!pip install git+https://github.com/afnan47/cuda.git
%load_ext nvcc_plugin
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/afnan47/cuda.git
  Cloning https://github.com/afnan47/cuda.git to /tmp/pip-req-build-x8c7w1kb
  Running command git clone --filter=blob:none --quiet https://github.com/afnan47/cuda.git /tmp/pip-req-build-x8c7w1kb
  Resolved https://github.com/afnan47/cuda.git to commit aac710a35f52bb78ab34d2e52517237941399eff
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl size=4287 sha256=7ae816c6d26de3803b0dea478621f201ed9d2f2
  Stored in directory: /tmp/pip-ephem-wheel-cache-5m4zwhnw/wheels/aa/f3/44/e10c1d226ec561d971fcd4b0463f6bff08602afa928a3e7bc7
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
created output directory at /content/src
Out bin /content/result.out
```

```
%%cu
#include <iostream>
using namespace std;


// CUDA code to multiply matrices
__global__ void multiply(int* A, int* B, int* C, int size) {
    // Uses thread idices and block indices to compute each element
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

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


void initialize(int* matrix, int size) {
    for (int i = 0; i < size * size; i++) {
        matrix[i] = rand() % 10;
    }
}


void print(int* matrix, int size) {
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            cout << matrix[row * size + col] << " ";
        }
        cout << '\n';
    }
    cout << '\n';
}


int main() {
    int* A, * B, * C;

    int N = 2;
    int blockSize =  16;

    int matrixSize = N * N;
    size_t matrixBytes = matrixSize * sizeof(int);

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

    initialize(A, N);
    initialize(B, N);
    cout << "Matrix A: \n";
    print(A, N);

    cout << "Matrix B: \n";
    print(B, N);
    int* X, * Y, * Z;
    // Allocate space
    cudaMalloc(&X, matrixBytes);
    cudaMalloc(&Y, matrixBytes);
    cudaMalloc(&Z, matrixBytes);

    // Copy values from A to X
    cudaMemcpy(X, A, matrixBytes, cudaMemcpyHostToDevice);

    // Copy values from A to X and B to Y
    cudaMemcpy(Y, B, matrixBytes, cudaMemcpyHostToDevice);

    // Threads per CTA dimension
    int THREADS = 2;

    // Blocks per grid dimension (assumes THREADS divides N evenly)
    int BLOCKS = N / THREADS;

    // Use dim3 structs for block  and grid dimensions
    dim3 threads(THREADS, THREADS);
    dim3 blocks(BLOCKS, BLOCKS);

    // Launch kernel
    multiply<<<blocks, threads>>>(X, Y, Z, N);

    cudaMemcpy(C, Z, matrixBytes, cudaMemcpyDeviceToHost);
    cout << "Multiplication of matrix A and B: \n";
    print(C, N);

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

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

    return 0;
}
```

```
Matrix A:
3 6
7 5

Matrix B:
3 5
6 2

Multiplication of matrix A and B:
45 27
51 45
```

✓ 2s completed at 16:46