```
# 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-r71u3_ha
  Running command git clone --filter=blob:none --quiet https://github.com/afnan47/cuda.git /tmp/pip-req-build-r71u3_ha
  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=98d15ac286cc1c180a866667355185a55206546
  Stored in directory: /tmp/pip-ephem-wheel-cache-_c5a_uro/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;

__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++) {
        vector[i] = rand() % 10;
    }
}

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

int main() {
    int N = 4;
    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, N);
    cout << "Vector B: ";
    print(B, N);

    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;

    add<<<blocksPerGrid, threadsPerBlock>>>(X, Y, Z, N);

    cudaMemcpy(C, Z, vectorBytes, cudaMemcpyDeviceToHost);
```

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

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

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

    return 0;
}
```

```
 Vector A: 3 6 7 5
 Vector B: 3 5 6 2
 Addition: 6 11 13 7
```