```
# Steps to Run Cuda on Google Colab/ Jupyter Notebook
# Change runtime to GPU and run this cell

!pip install git+https://github.com/Ruturaj-Panditrao/cuda.git
%load_ext nvcc_plugin
```

```
Collecting git+https://github.com/Ruturaj-Panditrao/cuda.git
  Cloning https://github.com/Ruturaj-Panditrao/cuda.git to /tmp/pip-req-build-6p4qsz62
  Running command git clone --filter=blob:none --quiet https://github.com/Ruturaj-Panditrao/cuda.git /tmp/pip-req-build-6p4qsz62
  Resolved https://github.com/Ruturaj-Panditrao/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=4289 sha256=7bcc5d9298463151992f5480ead1b2f8b101f19e57e7
  Stored in directory: /tmp/pip-ephem-wheel-cache-zstilxb8/wheels/5f/0b/4d/5e62392fd6c7c38253be934211530add4ed3f048c93749d487
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
created output directory at /content/src
Out bin /content/result.out
```

```
# Steps to Run Cuda on Linux Command Line (here let us assume file name is abc)
# nvcc --version
# cat>> abc.cu
# Paste the code, then press Ctrl+D
# nvcc abc.cu
# ./a.out


# Note that %%cu is required only for Jupyter Notebooks/Google Colabs
# If you are running the code as a .cu file, you do not need this declaration.
# It basically represents the start of the CUDA code

# To understand the code effectively, start from the Main function

%%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;

    // blockIdx.x = Index of Current Block within the Grid
    // threadIdx.x = Index of Current Thread within each Block
    // blockDim.x = Number of threads per block

    // We are basically trying to assign a unique index (thread id or tid) to each thread for processing

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

    // If the thread assigned index is within the array bounds, let the thread perform addition
}

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

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 = 10;
    // To actually see the difference in CPU and GPU time, set N as 100000, but beware of printing the arrays then :)
    // Number of Elements in Each Vector

    int* A = new int[N];
    int* B = new int[N];
    int* C = new int[N];
    int* D = new int[N];
    // A,B,C,D are allocated memory on the Host (CPU)
    // We will use C for the result of addition on GPU
    // We will use D for the result of addition on CPU

    size_t vectorBytes = N * sizeof(int);
    // Finding the memory size of vector

    initialize(A, N);
    initialize(B, N);
    // Initialize A and B vectors with Random Values

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

    int* X, * Y, * Z;
    // X,Y,Z are allocated memory on the Device (GPU)
    cudaMalloc(&X, vectorBytes);
    cudaMalloc(&Y, vectorBytes);
    cudaMalloc(&Z, vectorBytes);

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

    int threadsPerBlock = 256;
    // This is dependent on the GPU architecture

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

    float gpu_elapsed_time;
    cudaEvent_t gpu_start,gpu_stop;
    cudaEventCreate(&gpu_start);
    cudaEventCreate(&gpu_stop);
    cudaEventRecord(gpu_start);

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

    cudaEventRecord(gpu_stop);
    cudaEventSynchronize(gpu_stop);
    cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
    cudaEventDestroy(gpu_start);
    cudaEventDestroy(gpu_stop);


    cudaMemcpy(C, Z, vectorBytes, cudaMemcpyDeviceToHost);

    cout<<"GPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds"<<endl;

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

    float cpu_elapsed_time;
    cudaEvent_t cpu_start,cpu_stop;
    cudaEventCreate(&cpu_start);
    cudaEventCreate(&cpu_stop);
    cudaEventRecord(cpu_start);

    addCPU(A,B,D,N);

    cudaEventRecord(cpu_stop);
    cudaEventSynchronize(cpu_stop);
    cudaEventElapsedTime(&cpu_elapsed_time, cpu_start, cpu_stop);
    cudaEventDestroy(cpu_start);
    cudaEventDestroy(cpu_stop);

    cout<<"CPU Elapsed time is: "<<cpu_elapsed_time<<" milliseconds"<<endl;
```

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

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

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

    return 0;
}
```

```
Vector A: 3 6 7 5 3 5 6 2 9 1
Vector B: 2 7 0 9 3 6 0 6 2 6
GPU Elapsed time is: 102.828 milliseconds
Addition: 5 13 7 14 6 11 6 8 11 7
CPU Elapsed time is: 0.002496 milliseconds
Addition: 5 13 7 14 6 11 6 8 11 7
```