Create New Notebook in google colab then go in runtime section in menu bar select change runtime and select any gpu runtime and save it.

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

Add this command to check cuda is install or it already installed in colab notebook

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-8tx7yadu
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-8tx7yadu
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit 5741c522547756ac4bb7a16df32106a15efb8a57
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: nvcc4jupyter
  Building wheel for nvcc4jupyter (pyproject.toml) ... done
  Created wheel for nvcc4jupyter: filename=nvcc4jupyter-1.2.1-py3-none-any.whl size=10741 sha256=1c43b610d84440f376c57bb0b3d20f87e2433fdb6f06e
  Stored in directory: /tmp/pip-ephem-wheel-cache-c63664xa/wheels/a8/b9/18/23f8ef71ceb0f63297dd1903aedd067e6243a68ea756d6feea
Successfully built nvcc4jupyter
Installing collected packages: nvcc4jupyter
Successfully installed nvcc4jupyter-1.2.1
```

add this command to install nvcc4jupyter to run cuda program

```
%load_ext nvcc4jupyter
```

```
Detected platform "Colab". Running its setup...
Source files will be saved in "/tmp/tmpwh2qeq7h".
```

now setup is done almost just run this command to run cuda program in your colab

```
%%cuda
#include <stdio.h>

__global__ void hello(){
    printf("Hello from block: %u, thread: %u\n", blockIdx.x, threadIdx.x);
}

int main(){
    hello<<<2, 2>>>();
    cudaDeviceSynchronize();
}
```

```
Hello from block: 0, thread: 0
Hello from block: 0, thread: 1
Hello from block: 1, thread: 0
Hello from block: 1, thread: 1
```

Program is run successfully and Output is displayed

```
%%cuda
#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
```

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


// CUDA code to multiply matrices
__global__ void multiply(int* A, int* B, int* C, int size) {
    // Uses thread indices 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
```