```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

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

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to
/tmp/pip-req-build-n8k618qh
  Running command git clone --filter=blob:none --quiet
https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-
n8k618qh
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit
aac710a35f52bb78ab34d2e52517237941399eff
  Preparing metadata (setup.py) ... e=NVCCPlugin-0.0.2-py3-none-
any.whl size=4287
sha256=720ed8e816e2460c133cf49bdc252172da7d8391ede2698b8f76380168ad861
d
  Stored in directory:
/tmp/pip-ephem-wheel-cache-4ihl4jul/wheels/a8/b9/18/23f8ef71ceb0f63297
dd1903aedd067e6243a68ea756d6feea
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

```
%load_ext nvcc_plugin
```

```
created output directory at /content/src
Out bin /content/result.out
```

```
%%cu
#include <iostream>
#include <cuda_runtime.h>

__global__ void addVectors(int* A, int* B, int* C, int n)
{
     int i = blockIdx.x * blockDim.x + threadIdx.x;
     if (i < n)
     {
     C[i] = A[i] + B[i];
     }
}

int main()
{
     int n = 1000000;
     int* A, * B, * C;
     int size = n * sizeof(int);

     // Allocate memory on the host
     cudaMallocHost(&A, size);
     cudaMallocHost(&B, size);
     cudaMallocHost(&C, size);

     // Initialize the vectors
     for (int i = 0; i < n; i++)
     {
     A[i] = i;
     B[i] = i * 2;
     }
     // Allocate memory on the device
     int* dev_A, * dev_B, * dev_C;
     cudaMalloc(&dev_A, size);
     cudaMalloc(&dev_B, size);
     cudaMalloc(&dev_C, size);

     // Copy data from host to device
     cudaMemcpy(dev_A, A, size, cudaMemcpyHostToDevice);
     cudaMemcpy(dev_B, B, size, cudaMemcpyHostToDevice);

     // Launch the kernel
     int blockSize = 256;
     int numBlocks = (n + blockSize - 1) / blockSize;
addVectors<<<numBlocks, blockSize>>>(dev_A, dev_B, dev_C, n);

// Synchronize to make sure the kernel has finished
cudaDeviceSynchronize();

// Copy data from device to host
cudaMemcpy(C, dev_C, size, cudaMemcpyDeviceToHost);

// Print the results
for (int i = 0; i < 10; i++)
{
    std::cout << C[i] << " ";
}
std::cout << std::endl;

     // Free memory
     cudaFree(dev_A);
     cudaFree(dev_B);
     cudaFree(dev_C);
     cudaFreeHost(A);
     cudaFreeHost(B);
     cudaFreeHost(C);
    return 0;
}
```

```
0 3 6 9 12 15 18 21 24 27
```