```
!nvcc --version
!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
%load_ext nvcc_plugin
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Sun_Jul_28_19:07:16_PDT_2019
Cuda compilation tools, release 10.1, V10.1.243
Collecting git+git://github.com/andreinechaev/nvcc4jupyter.git
  Cloning git://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-rhmbb
  Running command git clone -q git://github.com/andreinechaev/nvcc4jupyter.git /tmp/p
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-cp36-none-any.whl size=4307
  Stored in directory: /tmp/pip-ephem-wheel-cache-43ir_xdi/wheels/10/c2/05/ca241da37b
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<stdio.h>
#include<stdlib.h>

__global__ void vectorAdd(int *a, int *b, int *result, int n) {
    int tid = blockIdx.x*blockDim.x + threadIdx.x;
    if(tid <= n) {
        result[tid] = a[tid] + b[tid];
    }
}

void printVec(int *a,int n) {
    for(int i=0;i<n;i++) {
        printf("%d ",a[i]);
    }
    printf("\n");
}

void initVec(int *a,int n) {
    for(int i=0;i<n;i++) {
        a[i]=rand()%n+1;
    }
}

int main() {
    int *a,*b,*c;
    int *a_dev,*b_dev,*c_dev;
    int n=10;

    a=(int*)malloc(n*sizeof(int));
```

```
    b=(int*)malloc(n*sizeof(int));
    c=(int*)malloc(n*sizeof(int));

    cudaMalloc(&a_dev,n*sizeof(int));
    cudaMalloc(&b_dev,n*sizeof(int));
    cudaMalloc(&c_dev,n*sizeof(int));

    initVec(a,n);
    initVec(b,n);
    printVec(a,n);
    printVec(b,n);

    cudaMemcpy(a_dev,a,n*sizeof(int),cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev,b,n*sizeof(int),cudaMemcpyHostToDevice);

    vectorAdd<<<1,n>>>(a_dev,b_dev,c_dev,n);

    cudaMemcpy(c,c_dev,n*sizeof(int),cudaMemcpyDeviceToHost);

    printf("Sum : \n");
    printVec(c,n);

    cudaFree(a_dev);
    cudaFree(b_dev);
    cudaFree(c_dev);

    return 0;
}
```

```
 4 7 8 6 4 6 7 3 10 2
 3 8 1 10 4 7 1 7 3 7
 Sum :
 7 15 9 16 8 13 8 10 13 9
```

```
%%cu
#include<iostream>

using namespace std;

__global__
void matrixVector(int *vec, int *mat, int *result, int n, int m)
{
    int tid = blockIdx.x*blockDim.x + threadIdx.x;
    int sum=0;

    if(tid < m) {
        for(int i=0; i<n; i++) {
            sum += vec[i]*mat[(i*m) + tid];
        }
        result[tid] = sum;
    }
}
```

```
}

void init_array(int *a, int n) {
    for(int i=0; i<n; i++)
      a[i] = rand()%n + 1;
}

void init_matrix(int *a, int n, int m) {
    for(int i=0; i<n; i++) {
        for(int j=0; j<m; j++) {
            a[i*m + j] = rand()%n + 1;
        }
    }
}

void print_array(int *a, int n) {
    for(int i=0; i<n; i++) {
        cout<<"  "<<a[i];
    }
    cout<<endl;
}

void print_matrix(int *a, int n, int m) {
    for(int i=0; i<n; i++) {
        for(int j=0; j<m; j++)
          cout<<"  "<<a[i*m + j];
        cout<<endl;
    }
}

int main() {
    int *a, *b, *c;
    int *a_dev, *b_dev, *c_dev;

    int n = 3;
    int m = 5;

    a = new int[n];
    b = new int[n*m];
    c = new int[m];

    init_array(a, n);
    init_matrix(b, n, m);

    cout<<"Initial array : "<<endl;
    print_array(a, n);
    cout<<"Initial matrix : "<<endl;
    print_matrix(b, n, m);


    cudaMalloc(&a_dev, sizeof(int)*n);
    cudaMalloc(&b_dev, sizeof(int)*n*m);
    cudaMalloc(&c_dev, sizeof(int)*m);
```

```
    cudaMemcpy(a_dev, a, sizeof(int)*n, cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev, b, sizeof(int)*n*m, cudaMemcpyHostToDevice);

    matrixVector<<<1, m>>>(a_dev, b_dev, c_dev, n, m);

    cudaMemcpy(c, c_dev, sizeof(int)*m, cudaMemcpyDeviceToHost);

    cout<<"Results : "<<endl;
    print_array(c, m);

    cudaFree(a_dev);
    cudaFree(b_dev);
    cudaFree(c_dev);

    delete[] a;
    delete[] b;
    delete[] c;

    return 0;
}
```

```
 Initial array :
   2  2  1
  Initial matrix :
   2  3  2  2  1
   1  2  3  2  3
   2  3  2  1  1
  Results :
   8  13  12  9  9
```

```
%%cu
#include<iostream>

using namespace std;

__global__
void matrixMultiplication(int *a, int *b, int *c, int m, int n, int k)
{
    int row = blockIdx.y*blockDim.y + threadIdx.y;
    int col = blockIdx.x*blockDim.x + threadIdx.x;
    int sum=0;

    if(col<k && row<m) {
      for(int j=0;j<n;j++)
      {
          sum += a[row*n+j] * b[j*k+col];
      }
      c[k*row+col]=sum;
    }
```

```
}

void init_result(int *a, int m, int k) {
    for(int i=0; i<m; i++) {
      for(int j=0; j<k; j++) {
        a[i*k + j] = 0;
      }
    }
}

void init_matrix(int *a, int n, int m) {
    for(int i=0; i<n; i++) {
      for(int j=0; j<m; j++) {
        a[i*m + j] = rand()%10 + 1;
      }
    }
}

void print_matrix(int *a, int n, int m) {
    for(int i=0; i<n; i++) {
      for(int j=0; j<m; j++) {
        cout<<"  "<<a[i*m + j];
      }
      cout<<endl;
    }
    cout<<endl;
}

int main()
{

    int *a,*b,*c;
    int *a_dev,*b_dev,*c_dev;
    int m=5, n=4, k=3;

    a = new int[m*n];
    b = new int[n*k];
    c = new int[m*k];

    init_matrix(a, m, n);
    init_matrix(b, n ,k);
    init_result(c, m, k);

    cout<<"Initial matrix : "<<endl;

    print_matrix(a, m, n);
    print_matrix(b, n, k);
    print_matrix(c, m, k);

    cudaMalloc(&a_dev, sizeof(int)*m*n);
    cudaMalloc(&b_dev, sizeof(int)*n*k);
    cudaMalloc(&c_dev, sizeof(int)*m*k);
```

```
    cudaMemcpy(a_dev, a, sizeof(int)*m*n, cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev, b, sizeof(int)*n*k, cudaMemcpyHostToDevice);

    dim3 threads(16,16);
    dim3 blocks(1,1);

    matrixMultiplication<<<blocks, threads>>>(a_dev,b_dev,c_dev, m, n, k);

    cudaMemcpy(c, c_dev, sizeof(int)*m*k, cudaMemcpyDeviceToHost);

    cout<<"Result : "<<endl;
    print_matrix(c, m, k);

    cudaFree(a_dev);
    cudaFree(b_dev);
    cudaFree(c_dev);

    delete[] a;
    delete[] b;
    delete[] c;

    return 0;
}
```

```
Initial matrix :
  4  7  8  6
  4  6  7  3
  10  2  3  8
  1  10  4  7
  1  7  3  7

  2  9  8
  10  3  1
  3  4  8
  6  10  3

  0  0  0
  0  0  0
  0  0  0
  0  0  0
  0  0  0

Result :
  138  149  121
  107  112  103
  97  188  130
  156  125  71
  123  112  60
```