```
!apt-get --purge remove cuda nvidia* libnvidia-*
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
!apt-get remove cuda-*
!apt autoremove
!apt-get update
```

```
!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -O cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.d
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
!apt-get update
!apt-get install cuda-9.2
```

```
code = """
#include<iostream>
#include<cstdlib>
using namespace std;

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

void init_array(int *a, int N) {
    for(int i=0; i<N; i++) {
        a[i] = rand()%10 + 1;
    }
}

int main() {
    int *a, *b, *c;
    int *a_dev, *b_dev, *c_dev;
    int n = 8;

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

    int size = n * sizeof(int); // 32

    cudaMalloc(&a_dev, size);
    cudaMalloc(&b_dev, size);
    cudaMalloc(&c_dev, size);

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

    print_array(a, n);
    print_array(b, n);


    cudaMemcpy(a_dev, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev, b, size, cudaMemcpyHostToDevice);


    vectorAdd<<<2,1024>>>(a_dev, b_dev, c_dev, n);


    cudaMemcpy(c, c_dev, size, cudaMemcpyDeviceToHost);

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


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

    return 0;
}
"""
```

```
text_file = open("assign2.cu", "w")
text_file.write(code)
text_file.close()
```

```
!nvcc assign2.cu
```

```
!./a.out
```

```
  4  7  8  6  4  6  7  3
  10  2  3  8  1  10  4  7
Results :
  14  9  11  14  5  16  11  10
```

```
code = """#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 <= n) {
        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 = 4;

    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);
    cout<<"Initial resultant array : "<<endl;
    print_array(c, m);
    cout<<endl;

    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<<<m/256+1, 256>>>(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;
}


"""

text_file = open("matVec.cu", "w")
text_file.write(code)
text_file.close()
```

```
!nvcc matVec.cu
```

```
!./a.out
```

```
Initial array :
  2  2  1
Initial matrix :
  2  3  2  2
  1  1  2  3
```

```
  2 3 2 3
Initial resultant array :
  0 0 0 0

Results :
  8 11 10 13
```

```
code = """
#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 dimGrid(1,1);
    dim3 dimBlock(16,16);

    matrixMultiplication<<<dimGrid, dimBlock>>>(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;
}
"""
```

```
text_file = open("matMulti.cu", "w")
text_file.write(code)
text_file.close()
```

```
!nvcc matMulti.cu
```

```
!./a.out
```

```
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
```