```
!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-ubu
!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
```

```
!nvcc --version
```

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

```
%load_ext nvcc_plugin
```

```
%%cu

#include<iostream>
#include<math.h>
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++)
      {
          //Converting matrices to 1D array
          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;
      }
    }
}

```

```
33 void init_matrix(int *a, int n, int m) {
34     for(int i=0; i<n; i++) {
35       for(int j=0; j<m; j++) {
36         a[i*m + j] = rand()%10 + 1;
37       }
38     }
39 }
40
41 void print_matrix(int *a, int n, int m) {
42     for(int i=0; i<n; i++) {
43       for(int j=0; j<m; j++) {
44         cout<<"  "<<a[i*m + j];
45       }
46       cout<<endl;
47     }
48     cout<<endl;
49 }
50
51 int main()
52 {
53     cout<<"Matrix Multiplication: "<<endl;
54     int *a,*b,*c;
55     int *a_dev,*b_dev,*c_dev;
56     int m=5, n=4, k=3;
57
58     a = new int[m*n];
59     b = new int[n*k];
60     c = new int[m*k];
61
62     init_matrix(a, m, n);
63     init_matrix(b, n ,k);
64     init_result(c, m, k);
65
66     cout<<"Initial matrix : "<<endl;
67
68     print_matrix(a, m, n);
69     print_matrix(b, n, k);
70
71     cudaMalloc(&a_dev, sizeof(int)*m*n);
72     cudaMalloc(&b_dev, sizeof(int)*n*k);
73     cudaMalloc(&c_dev, sizeof(int)*m*k);
74
75     cudaMemcpy(a_dev, a, sizeof(int)*m*n, cudaMemcpyHostToDevice);
76     cudaMemcpy(b_dev, b, sizeof(int)*n*k, cudaMemcpyHostToDevice);
77
78     //Defining dimensions
79     dim3 dimGrid(1,1);
80     dim3 dimBlock(16,16);
81     matrixMultiplication<<<dimGrid, dimBlock>>>(a_dev,b_dev,c_dev, m, n, k);
82
83     cudaMemcpy(c, c_dev, sizeof(int)*m*k, cudaMemcpyDeviceToHost);
84
85     cout<<"Result : "<<endl;
86     print_matrix(c, m, k);
87
```

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

    delete[] a;
    delete[] b;
    delete[] c;
    return 0;
}
```

```
Matrix Multiplication:
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

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

```
%%cu

#include<iostream>
#include<math.h>
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;
}

```

```
26 void init_matrix(int *a, int n, int m) {
27     for(int i=0; i<n; i++) {
28         for(int j=0; j<m; j++) {
29             a[i*m + j] = rand()%n + 1;
30         }
31     }
32 }
33
34 void print_array(int *a, int n) {
35     for(int i=0; i<n; i++) {
36         cout<<"  "<<a[i];
37     }
38     cout<<endl;
39 }
40
41 void print_matrix(int *a, int n, int m) {
42     for(int i=0; i<n; i++) {
43         for(int j=0; j<m; j++)
44           cout<<"  "<<a[i*m + j];
45         cout<<endl;
46     }
47 }
48
49 int main() {
50     cout<<"Vector and Matrix Multiplication: "<<endl;
51     int *a, *b, *c;
52     int *a_dev, *b_dev, *c_dev;
53
54     int n = 3;
55     int m = 4;
56
57     a = new int[n];
58     b = new int[n*m];
59     c = new int[m];
60
61     init_array(a, n);
62     init_matrix(b, n, m);
63
64     cout<<"Initial array : "<<endl;
65     print_array(a, n);
66     cout<<"Initial matrix : "<<endl;
67     print_matrix(b, n, m);
68     cout<<"Initial resultant array : "<<endl;
69     print_array(c, m);
70     cout<<endl;
71
72     cudaMalloc(&a_dev, sizeof(int)*n);
73     cudaMalloc(&b_dev, sizeof(int)*n*m);
74     cudaMalloc(&c_dev, sizeof(int)*m);
75
76     cudaMemcpy(a_dev, a, sizeof(int)*n, cudaMemcpyHostToDevice);
77     cudaMemcpy(b_dev, b, sizeof(int)*n*m, cudaMemcpyHostToDevice);
78
79     matrixVector<<<m/256+1, 256>>>(a_dev, b_dev, c_dev, n, m);
80
```

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

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

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

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

    return 0;
}
```

```
Vector and Matrix Multiplication:
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

Result :
  8  11  10  13
```

```
%%cu

#include<iostream>
#include<math.h>
using namespace std;

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

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()%100 + 1;
    }
}
```

```
27
28 int main() {
29     cout<<"Addition of two large vectors : "<<endl;
30     int *a, *b, *c;
31     int *a_dev, *b_dev, *c_dev;
32     int n = 100;
33     int threads_per_block = 25;
34     int size = n * sizeof(int);
35
36     a = new int[n];
37     b = new int[n];
38     c = new int[n];
39
40     init_array(a, n);
41     init_array(b, n);
42
43     cudaMalloc(&a_dev, size);
44     cudaMalloc(&b_dev, size);
45     cudaMalloc(&c_dev, size);
46
47     cout<<"Array 1 : "<<endl;
48     print_array(a, n);
49     cout<<"Array 2 : "<<endl;
50     print_array(b, n);
51
52     cudaMemcpy(a_dev, a, size, cudaMemcpyHostToDevice);
53     cudaMemcpy(b_dev, b, size, cudaMemcpyHostToDevice);
54     add<<<n/threads_per_block, threads_per_block>>>(a_dev, b_dev, c_dev, n);
55     cudaMemcpy(c, c_dev, size, cudaMemcpyDeviceToHost);
56
57     cout<<"Result : "<<endl;
58     print_array(c, n);
59
60     cudaFree(a_dev);
61     cudaFree(b_dev);
62     cudaFree(c_dev);
63
64     return 0;
65 }
```

```
Addition of two large vectors :
Array 1 :
  84  87  78  16  94  36  87  93  50  22  63  28  91  60  64  27  41  27  73  37  12
Array 2 :
  96  71  35  79  68  2  98  3  18  93  53  57  2  81  87  42  66  90  45  20  41  36
Result :
  180  158  113  95  162  38  185  96  68  115  116  85  93  141  151  69  107  117
```