```
#Amruta Sool
#BBCO19128
```

```
!/usr/local/cuda/bin/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
```

```
!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-bmxxsg23
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-bmxxsg23
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit aac710a35f52bb78ab34d2e52517237941399eff
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl size=4287 sha256=14ac02382d330fce4ea106136d640a8e5697cc91a62d5a9cfeaf84
  Stored in directory: /tmp/pip-ephem-wheel-cache-_gkvm18c/wheels/a8/b9/18/23f8ef71ceb0f63297dd1903aedd067e6243a68ea756d6feea
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

```
# Commented out IPython magic to ensure Python compatibility.
%load_ext nvcc_plugin
```

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

```
# Commented out IPython magic to ensure Python compatibility.
%%cu

#/*[m,n]*[n,j]= [m,j]
# first take transpose of vector anfd
# then mulriply and add row wisw for each col.
# */
# //parallelism of multiplication only
# //tested ok on 05/12/2018

#include <cuda.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define m 10
__global__ void mul_r(int *a, int *b, int *c){
#
int tid = threadIdx.x;
            if (tid < m){
                        c[tid]= a[tid] * b[tid];
                        }

        }



 int main(){
            int  n, c, d, fst[10][10], snd[10][10], t_snd[10][10];
            int row,col,sum_c, a[10], b[10], ans[10];

  /*
    printf("Enter the number of rows and columns of matrix\n");
    scanf("%d%d", &m, &n);
    */
 n=m;  //square matrix only
    //printf("Enter the elements of first matrix\n");
    //for true randum value of vector
 //srand(time(0));
    for (c = 0; c < m; c++)
    {

       for (d = 0; d < n; d++)
      {
```

```
       //scanf("%d", &first[c][d]);

       fst[c][d]=rand()%10+1;
      }
  }
  printf("display the elements of first matrix\n");
  for (c = 0; c < m; c++) {
   for (d = 0 ; d < n; d++) {

                          printf("%d\t", fst[c][d]);
                              }
                               printf("\n");
                          }

  // take next matrix
  //for true randum value of vector
//srand(time(0));
  for (c = 0; c < m; c++)
  {

     for (d = 0; d < n; d++)
    {
        //scanf("%d", &first[c][d]);

       snd[c][d]=rand()%10+1;
      }
  }
  printf("display the elements of second matrix\n");
  for (c = 0; c < m; c++) {
     for (d = 0 ; d < n; d++) {

                          printf("%d\t", snd[c][d]);
                              }
                               printf("\n");
                          }
  // transpose of  second matrix
  for(c=0; c<m; c++)
       for(d=0; d<n; d++)
       {
           t_snd[d][c] = snd[c][d];
       }

   // Displaying the transpose of matrix a
   printf("\nTranspose of second Matrix:\n");
   for (c = 0; c < n; c++) {
     for (d = 0 ; d < m; d++) {

                      printf("%d\t", t_snd[c][d]);

                              }
                               printf("\n");
                          }

   // now multiply on cuda
  int *dev_a, *dev_b,*dev_ans;
  cudaError_t err=cudaSuccess;
  // allocate memory on GPU
  err=cudaMalloc((void**)&dev_a,m * sizeof(int));


  if (err !=cudaSuccess)
  {   printf("failed to  allocate on device \n");
  printf("error is:\n",cudaGetErrorString(err));
  exit(EXIT_FAILURE);
  }
  //printf("first  ok\n");
  cudaMalloc((void**)&dev_b,m * sizeof(int));
  cudaMalloc((void**)&dev_ans,m * sizeof(int));
  //printf("first finished ok\n");
  row=0;
  col=0;
 cudaEvent_t start, end;
 cudaEventCreate(&start);
 cudaEventCreate(&end);
 cudaEventRecord(start);

  for(row=0; row<m; row++){

   for (d = 0 ; d < m; d++) {

                      a[d]=fst[row][d];

                              }
                          // printf("ok a\n");
      cudaMemcpy(dev_a,a,m*sizeof(int), cudaMemcpyHostToDevice);
  for (col=0; col<m; col++){
   for (d= 0 ; d < m; d++) {
```

```
                    b[d]=t_snd[col][d];
                    ans[d]=0;

                            }
                        // printf("ok b\n");
      cudaMemcpy(dev_b,b,m*sizeof(int), cudaMemcpyHostToDevice);
      cudaMemcpy(dev_ans,ans,m*sizeof(int), cudaMemcpyHostToDevice);
  //  printf("calling GPU\n");
      mul_r<<<1,m>>>(dev_a,dev_b,dev_ans);
      err=cudaMemcpy(ans,dev_ans,m*sizeof(int), cudaMemcpyDeviceToHost);
      if (err !=cudaSuccess)
          {   printf("failed to  copy from device \n");
              exit(EXIT_FAILURE);
          }
  //printf("GPU returned\n");
  //a=fst[0];
  sum_c=0;
   for (d = 0 ; d < m; d++) {

                    //printf("%d\t", ans[d]);
                sum_c+=ans[d];
                            }
  snd[row][col]=sum_c;



  //printf("one element=%d\n",snd[row][col]);
                            // printf("\n");

  }
  }
  //
  cudaEventRecord(end);
 cudaEventSynchronize(end);
 float time = 0;
 cudaEventElapsedTime(&time, start, end);
 printf("execution time=%f\n",time);
//
  printf(" Matrix multipliation ans=:\n");
   for (c = 0; c < n; c++) {
     for (d = 0 ; d < m; d++) {

                    printf("%d\t", snd[c][d]);

                            }
                             printf("\n");
                        }
  //
          return 0;
              }
```

```
 display the elements of first 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       3       9       10      8       4       7       2       3
 10      4       2       10      5       8       9       5       6       1
 4       7       2       1       7       4       3       1       7       2
 6       6       5       8       7       6       7       10      4       8
 5       6       3       6       5       8       5       5       4       1
 8       9       7       9       9       5       4       2       5       10
 3       1       7       9       10      3       7       7       5       10
 display the elements of second matrix
 6       1       5       9       8       2       8       3       8       3
 3       7       2       1       7       2       6       10      5       10
 1       10      2       8       8       2       2       6       10      8
 8       7       8       4       7       6       7       4       10      5
 9       2       3       10      4       10      1       9       9       6
 1       10      7       4       9       6       7       2       2       6
 10      9       5       9       2       1       4       1       5       5
 5       5       8       7       4       2       8       6       10      7
 3       2       8       9       6       8       5       2       9       6
 10      8       6       4       9       9       4       2       9       10

 Transpose of second Matrix:
 6       3       1       8       9       1       10      5       3       10
 1       7       10      7       2       10      9       5       2       8
 5       2       2       8       3       7       5       8       8       6
 9       1       8       4       10      4       9       7       9       4
 8       7       8       7       4       9       2       4       6       9
 2       2       2       6       10      6       1       2       8       9
 8       6       2       7       1       7       4       8       5       4
 3       10      6       4       9       2       1       6       2       2
 8       5       10      10      9       2       5       10      9       9
 3       10      8       5       6       6       5       7       6       10
 execution time=4.661696
  Matrix multipliation ans=:
 278     357     303     377     361     261     288     251     428     372
 290     323     301     264     348     268     300     248     389     355
 289     342     287     316     358     263     274     268     451     385
 353     323     330     400     375     295     327     276     456     348
```

350	344	352	406	369	266	349	236	439	340
198	196	186	254	238	205	183	196	274	253
404	413	374	427	418	319	359	306	526	445
249	295	265	301	303	218	269	223	341	301
405	408	342	418	463	360	336	329	535	463
413	381	345	429	379	345	287	272	525	412