**Javairia Rehman**

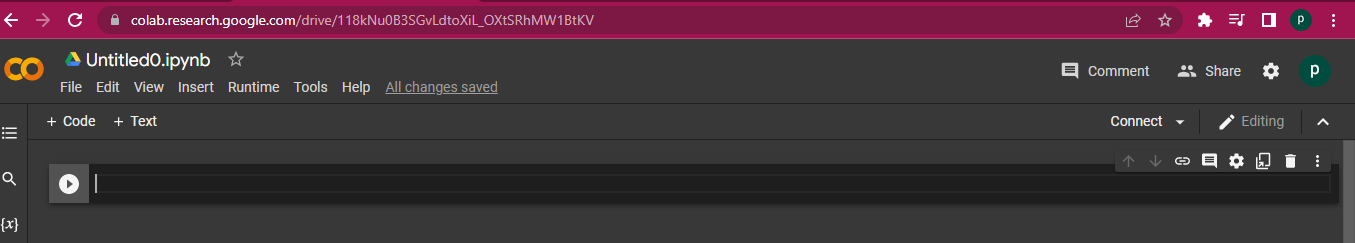
**19P-0020**

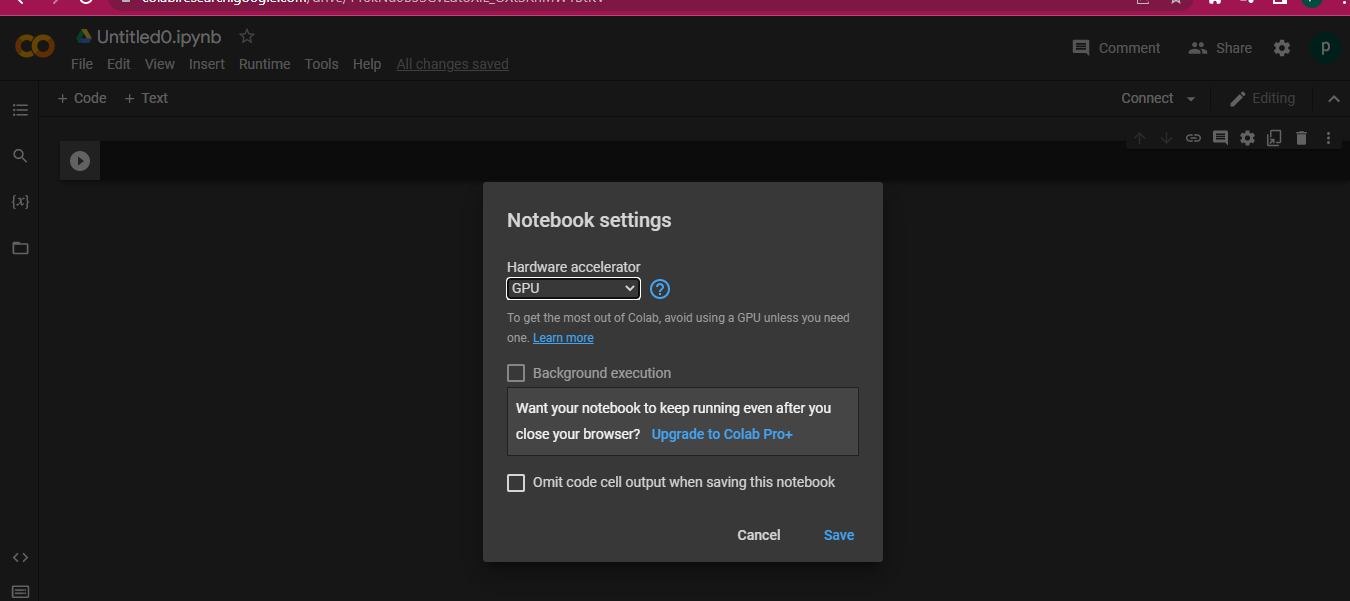
**BS CS 6-A**

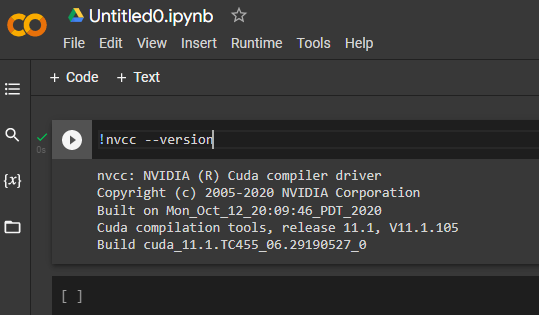
**CUDA(COMPUTER UNIFIED DEVICE ARCHITECTURE)**

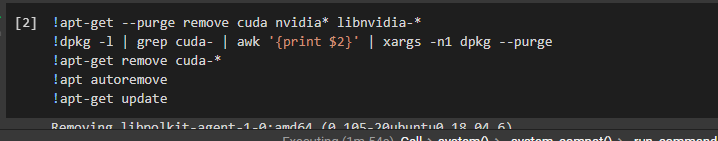
CUDA (or **Compute Unified Device Architecture**) is a parallel computing platform and application programming interface (API) that allows software to use certain types of graphics processing units (GPUs) for general purpose processing, an approach called general-purpose computing on GPUs (GPGPU).

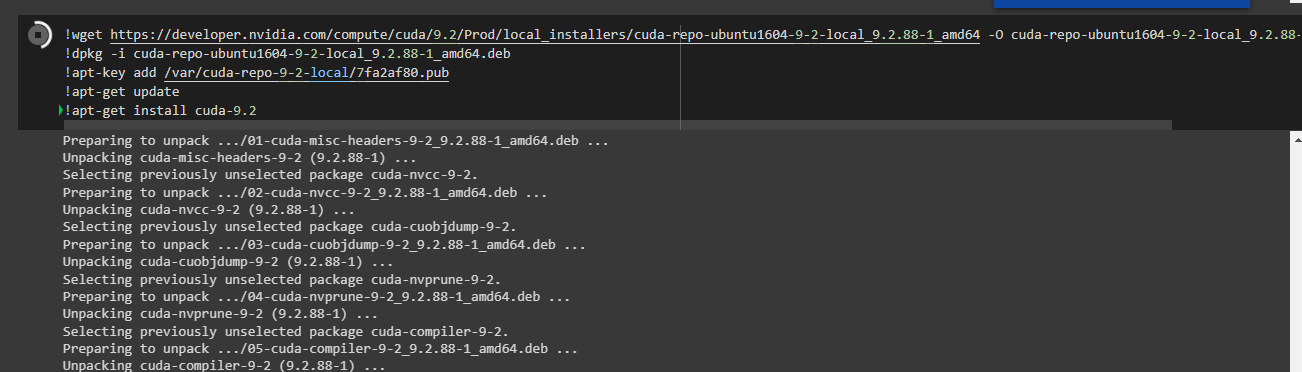
**SETTING ENVIRNMENT**

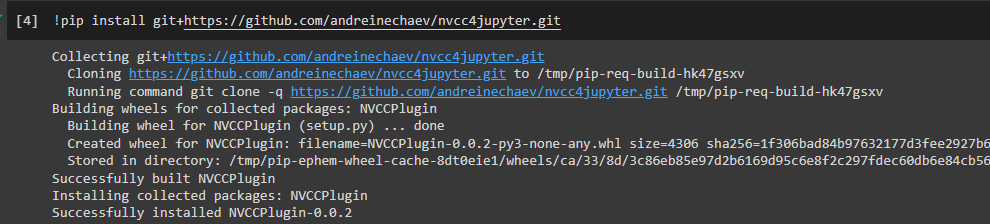
****

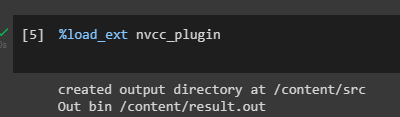












**Code of vector addition on cuda**

%%cu

#include<stdio.h>

#include<cuda.h>

\_\_global\_\_ void arradd(int \*x,int \*y, int \*z)    //kernel definition

{

  int id=blockIdx.x;

/\* blockIdx.x gives the respective block id which starts from 0 \*/

  z[id]=x[id]+y[id];

}

int main()

{

    int a[6];

    int b[6];

    int c[6];

    int \*d,\*e,\*f;

    int i;

  a[0]={1};

 a[1]={2};

 a[2]={3};

 a[3]={4};

 a[4]={5};

 a[5]={6};

    printf("\n six elements of first array\n");

    for(i=0;i<6;i++)

    {

        printf("%d\t",a[i]);

    }

 b[0]={1};

 b[1]={2};

 b[2]={3};

 b[3]={4};

 b[4]={5};

 b[5]={6};

    printf("\n  six elements of second array\n");

        for(i=0;i<6;i++)

        {

            printf("%d\t",b[i]);

        }

/\* cudaMalloc() allocates memory from Global memory on GPU \*/

    cudaMalloc((void \*\*)&d,6\*sizeof(int));

    cudaMalloc((void \*\*)&e,6\*sizeof(int));

    cudaMalloc((void \*\*)&f,6\*sizeof(int));

/\* cudaMemcpy() copies the contents from destination to source. Here destination is GPU(d,e) and source is CPU(a,b) \*/

 cudaMemcpy(d,a,6\*sizeof(int),cudaMemcpyHostToDevice);

 cudaMemcpy(e,b,6\*sizeof(int),cudaMemcpyHostToDevice);

/\* call to kernel. Here 6 is number of blocks, 1 is the number of threads per block and d,e,f are the arguments \*/

arradd<<<6,1>>>(d,e,f);

/\* Here we are copying content from GPU(Device) to CPU(Host) \*/

 cudaMemcpy(c,f,6\*sizeof(int),cudaMemcpyDeviceToHost);

printf("\nSum of two arrays:\n ");

    for(i=0;i<6;i++)

    {

        c[i]=a[i]+b[i];

        printf("%d\t",c[i]);

    }

/\* Free the memory allocated to pointers d,e,f \*/

    cudaFree(d);

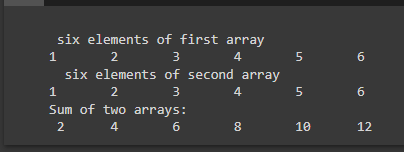
    cudaFree(e);

    cudaFree(f);

    return 0;

}

**Output**



**MATRIX MULTIPLICATION**

%%cu

#include<stdio.h>

#include<cuda.h>

#define row1 2 /\* Number of rows of first matrix \*/

#define col1 3 /\* Number of columns of first matrix \*/

#define row2 3 /\* Number of rows of second matrix \*/

#define col2 2 /\* Number of columns of second matrix \*/

\_\_global\_\_ void matproduct(int \*l,int \*m, int \*n)

{

    int x=blockIdx.x;

    int y=blockIdx.y;

    int k;

n[col2\*y+x]=0;

for(k=0;k<col1;k++)

   {

    n[col2\*y+x]=n[col2\*y+x]+l[col1\*y+k]\*m[col2\*k+x];

   }

}

int main()

{

    int a[row1][col1];

    int b[row2][col2];

    int c[row1][col2];

    int \*d,\*e,\*f;

    int i,j;

 a[0][0]={1};

 a[0][1]={2};

a[1][0]={3};

a[1][1]={4};

    printf("\n elements of first matrix of size 2\*2\n");

    for(i=0;i<2;i++)

    {

        for(j=0;j<2;j++)

            {

                printf("%d\t",a[i][j]);

            }

     printf("\n");

    }

  b[0][0]={2};

 b[0][1]={4};

b[1][0]={6};

b[1][1]={8};

    printf("\n  elements of second matrix of size 2\*2\n");

        for(i=0;i<2;i++)

        {

            for(j=0;j<2;j++)

                {

                    printf("%d\t",b[i][j]);

                }

         printf("\n");

        }

    cudaMalloc((void \*\*)&d,row1\*col1\*sizeof(int));

    cudaMalloc((void \*\*)&e,row2\*col2\*sizeof(int));

    cudaMalloc((void \*\*)&f,row1\*col2\*sizeof(int));

 cudaMemcpy(d,a,row1\*col1\*sizeof(int),cudaMemcpyHostToDevice);

 cudaMemcpy(e,b,row2\*col2\*sizeof(int),cudaMemcpyHostToDevice);

dim3 grid(col2,row1);

/\* Here we are defining two dimensional Grid(collection of blocks) structure. Syntax is dim3 grid(no. of columns,no. of rows) \*/

    matproduct<<<grid,1>>>(d,e,f);

 cudaMemcpy(c,f,row1\*col2\*sizeof(int),cudaMemcpyDeviceToHost);

    printf("\nProduct of two matrices:\n ");

 for(i = 0; i < 1; ++i)

        for(j = 0; j < 2; ++j)

        {

            c[i][j]=0;

        }

  for(i = 0; i < 2; ++i)

   {

        for(j = 0; j < 2; ++j)

    {

        c[i][j]=0;

            for(int k = 0; k < 2; ++k)

            {

                c[i][j] += a[i][k] \* b[k][j];

           }}       }

  for (int i = 0; i < 2; ++i) {

      for (int j = 0; j < 2; ++j) {

         printf("%d  ", c[i][j]);

      }

      printf("\t\n");

   }

    cudaFree(d);

    cudaFree(e);

    cudaFree(f);

    return 0;

}

**Output**

