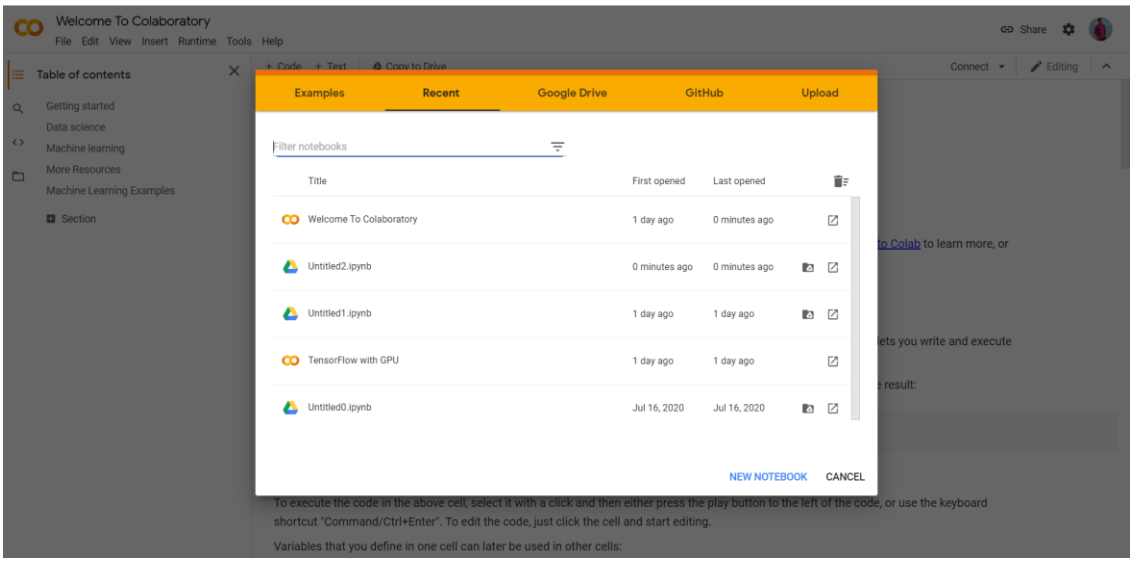
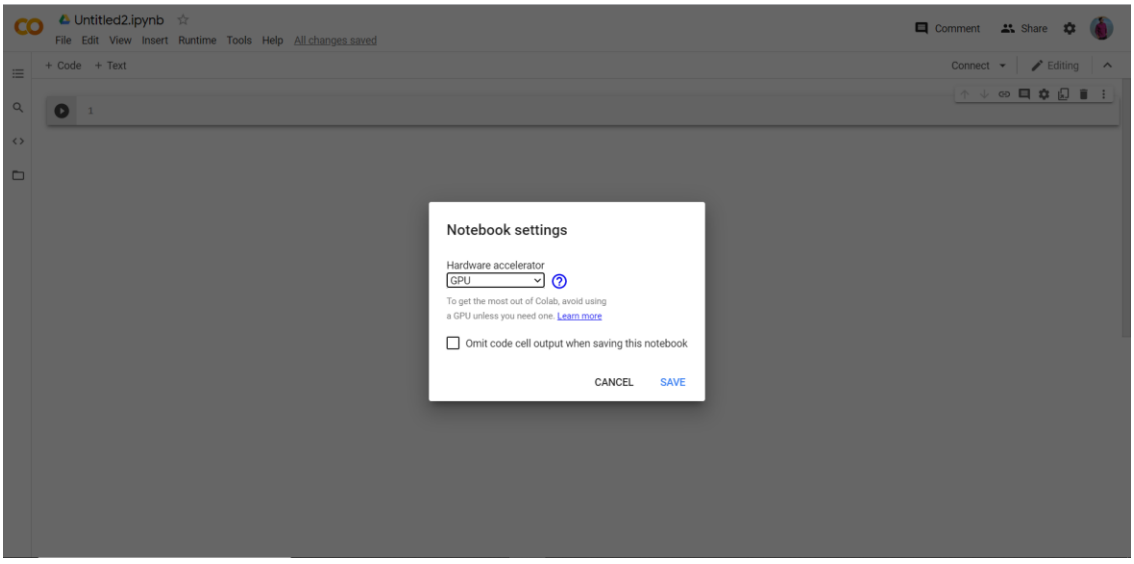
CUDA ASSIGNMENT

Name- KESHAV GARG
Id- 2018UCP1674

Step 1: Go to https://colab.research.google.com in Browser and Click on New Notebook.



Step 2: Click to Runtime > Change > Hardware Accelerator GPU .



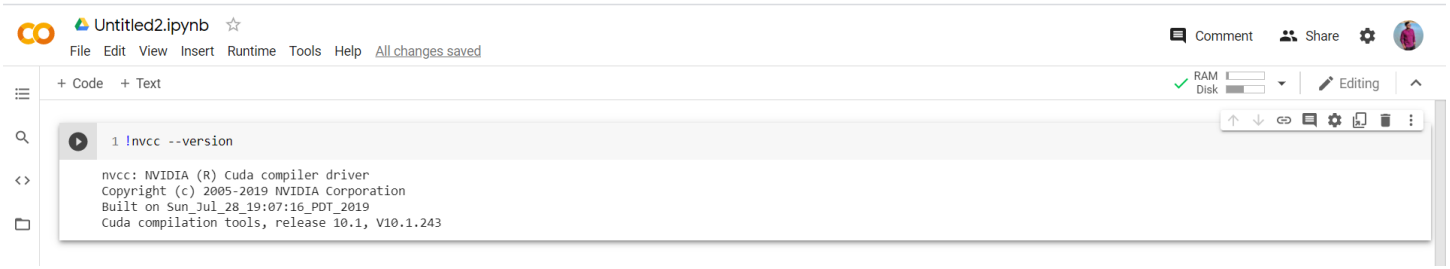
Step 4: Install CUDA Version 9 [write code in a Separate code Block and Run that]

```
!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.deb
!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
```

CUDA maybe be already installed

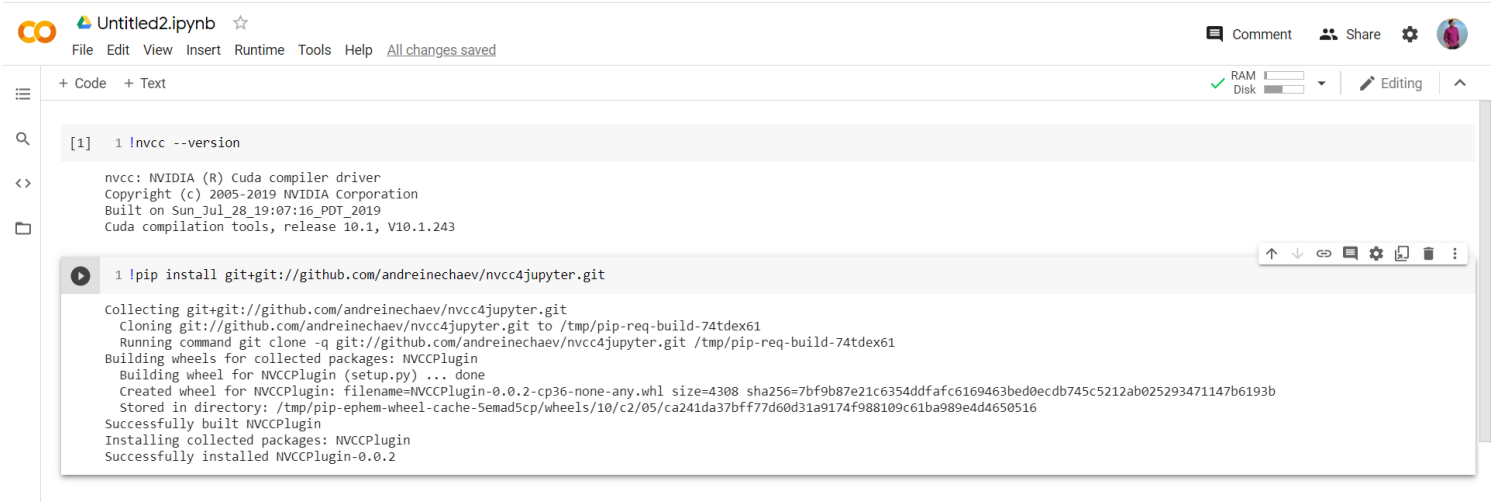
Step 5: Check the Version of CUDA by : running the command below to get the following output :

```
!nvcc -version
```



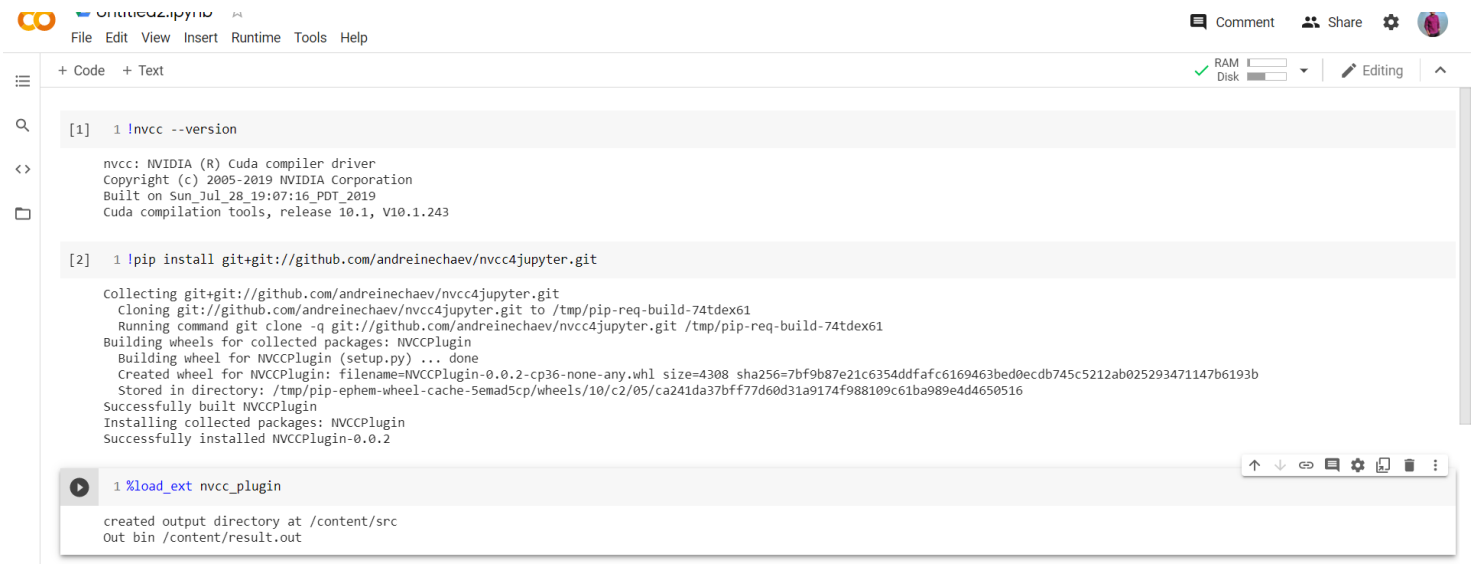
Step 6 : Execute the given command to install a small extension to run nvcc from Notebook cells [write code in a Seprate code Block and Run that]

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



Step 7: Load the extension using this code:[write code in a Seprate code Block and Run that]

```
%load_ext nvcc_plugin
```



Now you are all set to run CUDA program in Google Colab.

Write *%%cu* on the top of the code to execute your c code.

Ques 1. WAP for Vector addition in CUDA C.

Ans:

```
%%cu
#include<stdio.h>
#define MAX 10

//kernel function for addition of 2 Vectors
__global__ void Addition(int *gpu1,int *gpu2,int *res,int N)
{
  int index;
  index = threadIdx.x + blockIdx.x*blockDim.x;
  if(index<N)
  {
    res[index] = gpu1[index] + gpu2[index];
  }
}
```

```
//Function to display Array elements
void Display(int Arr[MAX],int n)
{
  for(int i=0;i<n;i++)
  {
    printf("\n\t Element %d : %d",i+1,Arr[i]);
  }
  printf("\n\t");
}

//Main function
int main()
{
  int Result[MAX],i,N;                                          //Memory Allocation on the device
  int *dev1,*dev2,*dev_Result;       //to declare on CPU & allocate memory on the Device/GPU

  printf("\n\t No. of elements : ");
  N=5;
  printf("%d",N);
  //scanf("%d",&N);

  //First_Array elements
  printf("\n\t First Array elements : ");
  int Array1[]={20,34,23,12,50};

  //Display array1 elements
  printf("\n\t Elements of Array1 : ");
  Display(Array1,N);

 //Second_Array elements
  printf("\n\t Second Array elements : ");
  int Array2[]={51,34,56,23,12};

  //Display array2 elements
  printf("\n\t Elements of Array2 : ");
  Display(Array2,N);

  //Memory Allocation for arrays on the CUDA device
  cudaMalloc((void**)&dev1, sizeof(int)*N);
  cudaMalloc((void**)&dev2, sizeof(int)*N);
  cudaMalloc((void**)&dev_Result, sizeof(int)*N);

  //Data Transfer from CPU to the device for computation
  cudaMemcpy(dev1,Array1, sizeof(int)*N, cudaMemcpyHostToDevice);
  cudaMemcpy(dev2,Array2, sizeof(int)*N, cudaMemcpyHostToDevice);

  //function call with 1 i.e. number of parallel blocks
  // N = Number of threads in each block
  Addition<<<1,N>>>(dev1, dev2, dev_Result, N);

  // Resultant array copy from GPU to CPU.
  printf("\n\t Vector addition : ");
```

```
  cudaMemcpy(Result, dev_Result, sizeof(int)*N, cudaMemcpyDeviceToHost);
  Display(Result, N);
  return 0;
}
```

After writing this code in Separate code Block, Run that cell by [Ctrl+Enter].

Ques 2. WAP for Matrix-matrix multiplication in CUDA C.

Ans.

```
%%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;
```

```
    printf("\n Enter elements of first matrix of size 2*3\n");
    int k=1;
    for(i=0;i<row1;i++)
    {
        for(j=0;j<col1;j++)
            {
                a[i][j]=k++;
                printf(" %d ",a[i][j]);
            }
    }
    printf("\n Enter elements of second matrix of size 3*2\n");
        for(i=0;i<row2;i++)
        {
            for(j=0;j<col2;j++)
                {
                    b[i][j]=k++;
                    printf(" %d ",b[i][j]);
                }
        }

    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<row1;i++)
    {
        for(j=0;j<col2;j++)
        {
              printf("%d\t",c[i][j]);
        }
        printf("\n");
    }

    cudaFree(d);
    cudaFree(e);
    cudaFree(f);

    return 0;
}
```

After writing this code in Separate code Block, Run that cell by [Ctrl+Enter].

```
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<row1;i++)
   {
       for(j=0;j<col2;j++)
       {
            printf("%d\t",c[i][j]);
       }
       printf("\n");
   }

   cudaFree(d);
   cudaFree(e);
   cudaFree(f);

   return 0;
}
```

```
 Enter elements of first matrix of size 2*3
 1  2  3  4  5  6
 Enter elements of second matrix of size 3*2
 7  8  9  10  11  12
Product of two matrices:
 58     64
139     154
```