Vector Addition

```
#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;
    printf("\n Enter six elements of first array\n");
    for(i=0;i<6;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("\n Enter six elements of second array\n");
        for(i=0;i<6;i++)
        {
            scanf("%d",&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++)
    {
        printf("%d\t",c[i]);
```

```
    }

/* Free the memory allocated to pointers d,e,f */
    cudaFree(d);
    cudaFree(e);
    cudaFree(f);

    return 0;
}
```

Matrix multiplication-

```
#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");
    for(i=0;i<row1;i++)
    {
        for(j=0;j<col1;j++)
            {
                scanf("%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++)
            {
                scanf("%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;
}
```

mnist— and fashon

x_train,x_test= x[:60000], x[60000:]

y_train,y_test= y[:60000], y[60000:]

boston house-

df=pd.read_csv(‘ ‘)

print(df.feature_name)

```
df.columns=df.features_name
df[‘price’]=df.target
x=df.drop[‘PRICE’,axis=1)
y=df[‘price’]
```