

List of programs:-

1- Matrix Addition

single block without shared memory, single block with shared memory

multiple block without shared memory, multiple block with shared memory

2- Matrix Multiplication

single block without shared memory, single block with shared memory

multiple block without shared memory, multiple block with shared memory

3-Square of first 100 integers

single block without shared memory, single block with shared memory

multiple block without shared memory, multiple block with shared memory

4-Matrix Transpose

single block without shared memory, single block with shared memory

multiple block without shared memory, multiple block with shared memory

Matrix Addition

single block without shared memory, single block with shared memory

multiple block without shared memory, multiple block with shared memory

/* single block without shared memory */

```
#include<stdio.h>
#include<conio.h>
#include<cuda.h>
__global__ void addition(int* a,int* b,int* c,int n)
{
    int h;
    h=a[threadIdx.x+n*threadIdx.y]+b[threadIdx.x+n*threadIdx.y];

    c[threadIdx.x+n*threadIdx.y]=h;

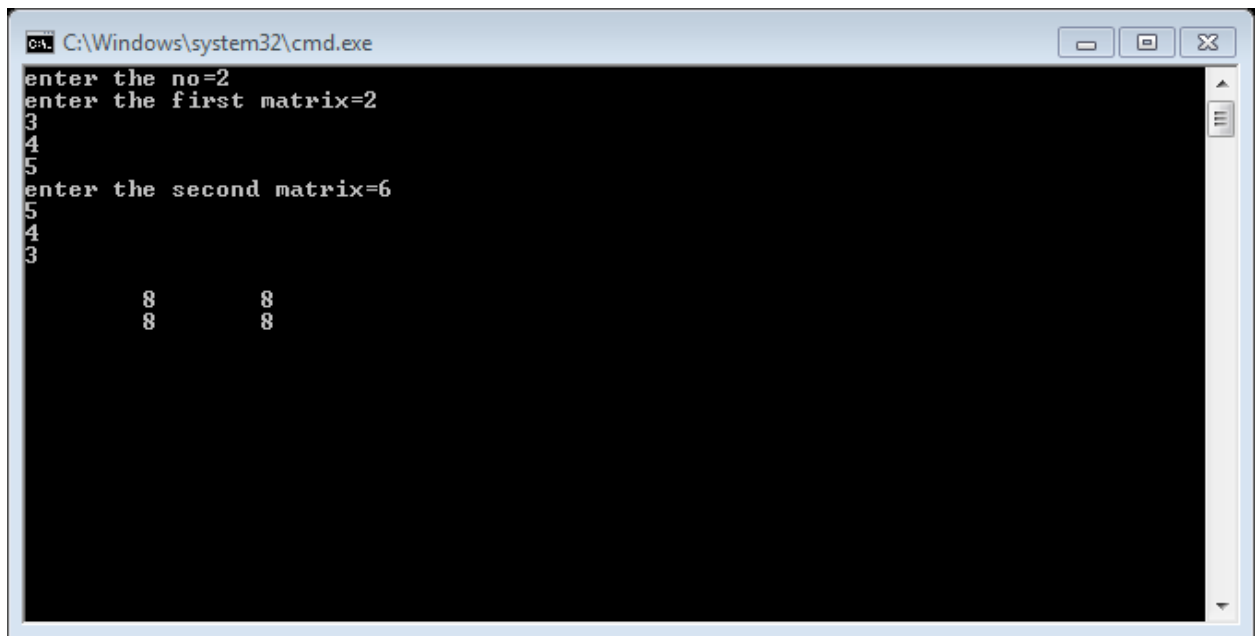
}
main()
{
    int *a_h,*b_h,*c_h,*a_d,*b_d,*c_d;
    int i,n,z;
    printf("enter the no=");
```

```

scanf("%d",&n);
z=n*n;
size_t size=sizeof(int)*z;
a_h=(int*)malloc(size);
b_h=(int*)malloc(size);
c_h=(int*)malloc(size);
cudaMalloc((void**)&a_d,size);
cudaMalloc((void**)&b_d,size);
cudaMalloc((void**)&c_d,size);
printf("enter the first matrix=");
for(i=0;i<z;i++)
{
scanf("%d",&a_h[i]);
}
printf("enter the second matrix=");
for(i=0;i<z;i++)
{
scanf("%d",&b_h[i]);
}
cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);
cudaMemcpy(b_d,b_h,size,cudaMemcpyHostToDevice);
dim3 dimBlock(n,n,1);
dim3 dimGrid(1,1);
addition<<<dimGrid,dimBlock>>>(a_d,b_d,c_d,n);
cudaMemcpy(c_h,c_d,size,cudaMemcpyDeviceToHost);
for(i=0;i<z;i++)
{
if(i%n==0)
{
printf("\n");
}
printf("%d",c_h[i]);
}
free(a_h);
free(b_h);
free(c_h);
cudaFree(a_d);
cudaFree(b_d);
cudaFree(c_d);
getch();
}

```

Output :-



```
C:\Windows\system32\cmd.exe
enter the no=2
enter the first matrix=2
3
4
5
enter the second matrix=6
5
4
3

      8      8
      8      8
```

`/* multiple block without shared memory */`

```
#include<stdio.h>
#include<conio.h>
#include<cuda.h>
__global__ void addition(int* a,int* b,int* c,int n)
{
    int h;
    int x=blockIdx.x*2+threadIdx.x;
    int y=blockIdx.y*2+threadIdx.y;
    h=a[x+n*y]+b[x+n*y];

    c[x+n*y]=h;

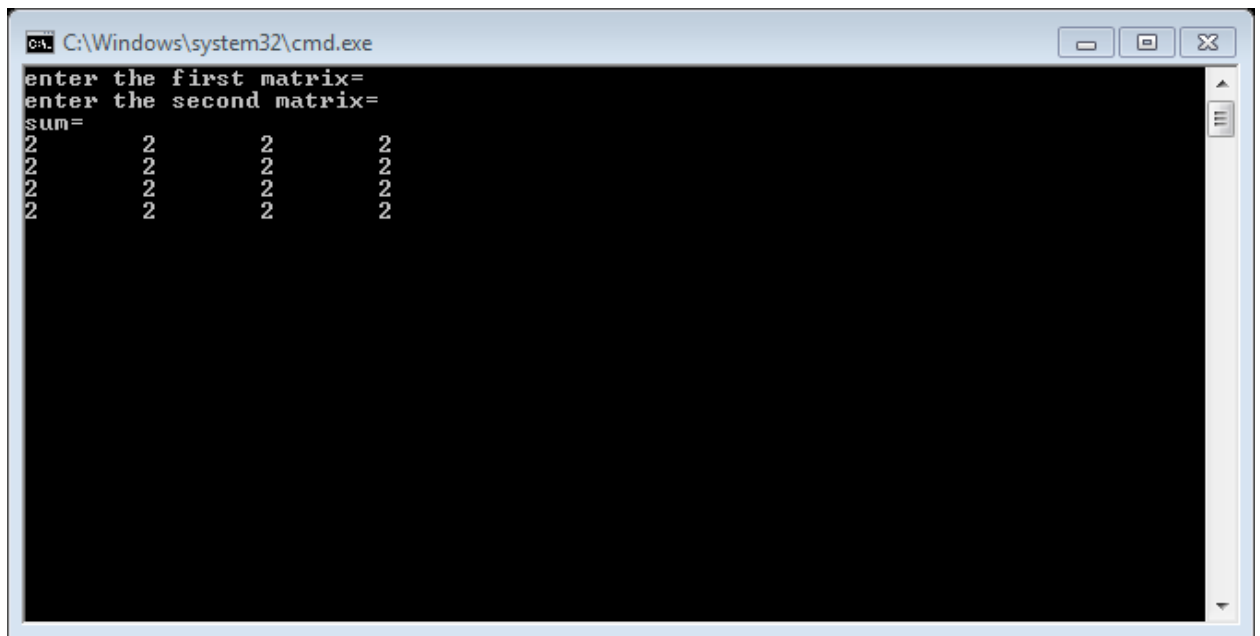
}
main()
{
    int *a_h,*b_h,*c_h,*a_d,*b_d,*c_d;
    int i,n=4,z;

    z=n*n;
    size_t size=sizeof(int)*z;
    a_h=(int*)malloc(size);
    b_h=(int*)malloc(size);
```

```

c_h=(int*)malloc(size);
cudaMalloc((void**)&a_d,size);
cudaMalloc((void**)&b_d,size);
cudaMalloc((void**)&c_d,size);
printf("enter the first matrix=");
for(i=0;i<z;i++)
{
a_h[i]=1;
}
printf("\nenter the second matrix=");
for(i=0;i<z;i++)
{
b_h[i]=1;
}
cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);
cudaMemcpy(b_d,b_h,size,cudaMemcpyHostToDevice);
dim3 dimBlock(2,2,1);
dim3 dimGrid(2,2);
addition<<<dimGrid,dimBlock>>>(a_d,b_d,c_d,n);
cudaMemcpy(c_h,c_d,size,cudaMemcpyDeviceToHost);
printf("\nsum=");
for(i=0;i<z;i++)
{
if(i%n==0)
{
printf("\n");
}
printf("%d\t",c_h[i]);
}
free(a_h);
free(b_h);
free(c_h);
cudaFree(a_d);
cudaFree(b_d);
cudaFree(c_d);
getch();
}
Output:-

```



```
C:\Windows\system32\cmd.exe
enter the first matrix=
enter the second matrix=
sum=
2      2      2      2
2      2      2      2
2      2      2      2
2      2      2      2
```

/* single block with shared memory */

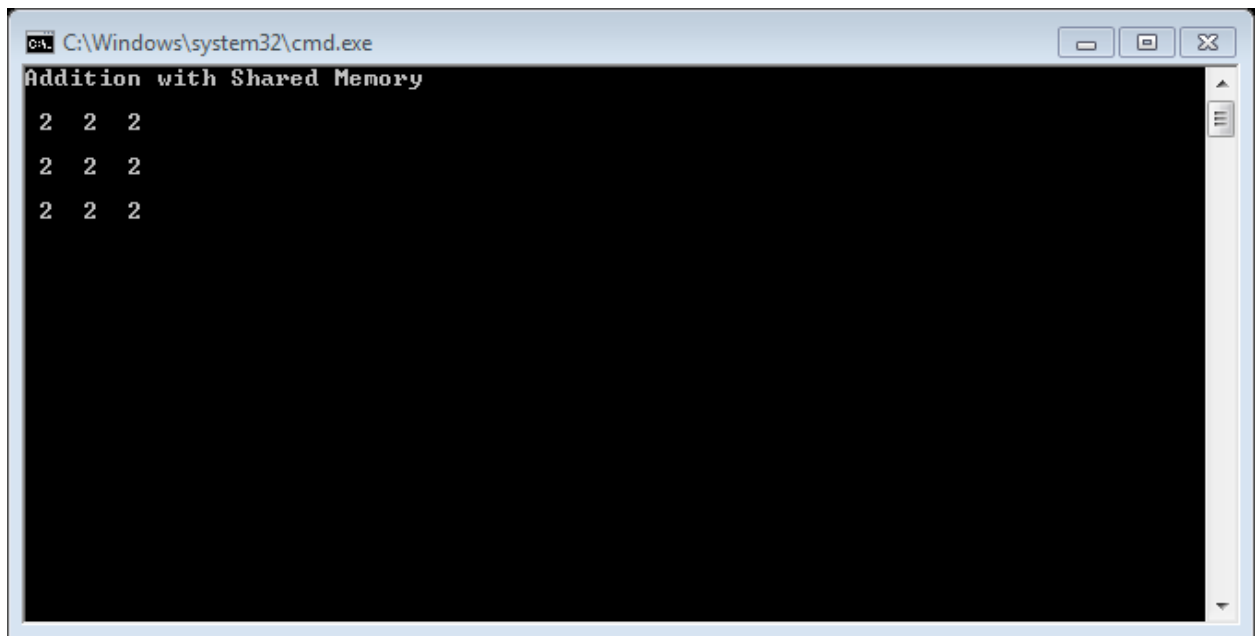
```
#include<stdio.h>
#include<conio.h>
#include<cuda.h>
__global__ void Sharesum(int* a,int* b,int* c,int n)
{
    __shared__ int as[3][3],bs[3][3];
    int s,k,i,sum=0;
    int tx,ty,bx,by;
    tx=threadIdx.x;
    ty=threadIdx.y;
    as[ty][tx]=a[tx+n*ty];
    bs[ty][tx]=b[tx+n*ty];
    sum += as[ty][tx]+bs[ty][tx];
    c[tx*n+ty]=sum;
}
main()
{
    int *ah,*ad,*bh,*bd,*ch,*cd;
    int i,n=3,z;
    z=n*n;
    size_t size=sizeof(int)*z;
```

```

    ah=(int*)malloc(size);
    bh=(int*)malloc(size);
    ch=(int*)malloc(size);
    cudaMalloc((void**)&ad,size);
    cudaMalloc((void**)&bd,size);
    cudaMalloc((void**)&cd,size);
    for(i=0;i<z;i++)
    {
        ah[i]=1;
    }
    for(i=0;i<z;i++)
    {
        bh[i]=1;
    }
    cudaMemcpy(ad,ah,size,cudaMemcpyHostToDevice);
    cudaMemcpy(bd,bh,size,cudaMemcpyHostToDevice);
    dim3 Block(n,n,1);
    dim3 Grid(1,1);
    Sharesum<<<Grid,Block>>>(ad,bd,cd,n);
    cudaMemcpy(ch,cd,size,cudaMemcpyDeviceToHost);
    printf("Addition with Shared Memory");
    for(i=0;i<z;i++)
    {
        if(i%n==0)
        {
            printf("\n\n");
        }
        printf(" %d ",cd[i]);
    }
    printf("\n");
    free(ah);
    free(bh);
    free(ch);
    cudaFree(ad);
    cudaFree(bd);
    cudaFree(cd);
    getch();
}

```

Output:-



```
C:\Windows\system32\cmd.exe
Addition with Shared Memory
2 2 2
2 2 2
2 2 2
```

/* multiple block with shared memory */

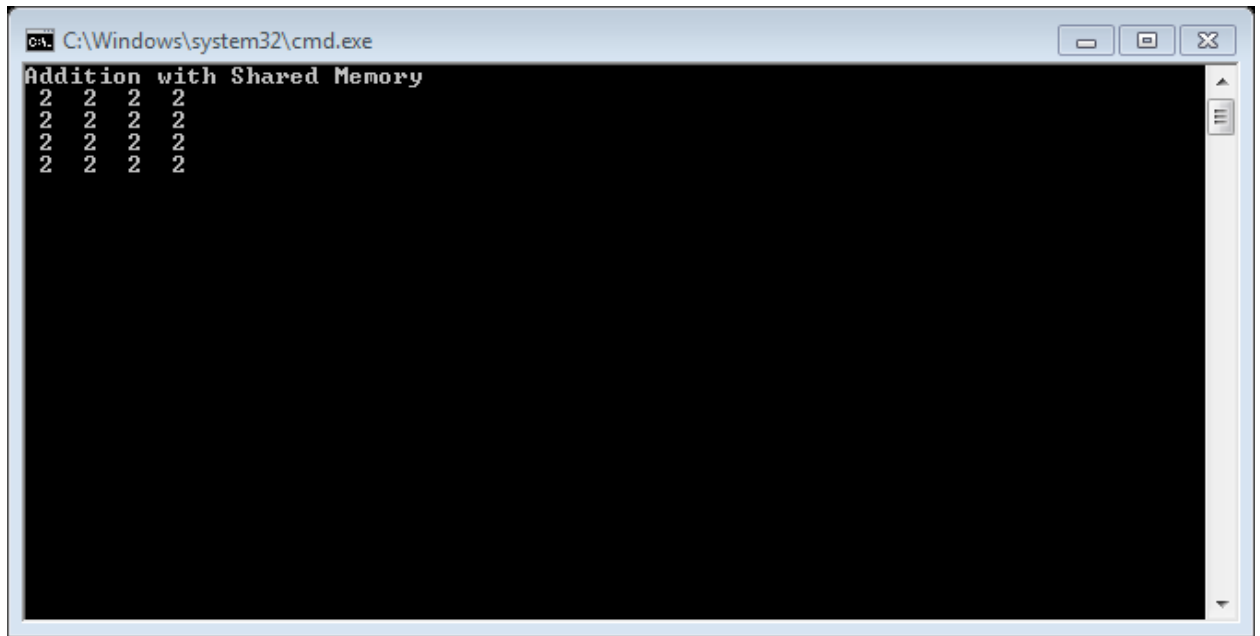
```
#include<stdio.h>
#include<conio.h>
#include<cuda.h>
__global__ void Sharesum(int* a,int* b,int* c,int n)
{
    __shared__ int as[8][8],bs[8][8];
    int s,k,i,sum=0;
    int tx,ty,bx,by;
    bx=blockIdx.x;
    by=blockIdx.y;
    tx=threadIdx.x;
    ty=threadIdx.y;
    s=bx*8+tx;
    k=by*8+ty;
    as[ty][tx]=a[k*n+tx];
    bs[ty][tx]=b[s+ty*n];
    sum += as[ty][tx]+bs[ty][tx];
    c[s*n+k]=sum;
}
main()
{
    int *ah,*ad,*bh,*bd,*ch,*cd;
    int i,n=32,z;
```

```

z=n*n;
size_t size=sizeof(int)*z;
ah=(int*)malloc(size);
bh=(int*)malloc(size);
ch=(int*)malloc(size);
cudaMalloc((void**)&ad,size);
cudaMalloc((void**)&bd,size);
cudaMalloc((void**)&cd,size);
for(i=0;i<z;i++)
{
    ah[i]=1;
}
for(i=0;i<z;i++)
{
    bh[i]=1;
}
cudaMemcpy(ad,ah,size,cudaMemcpyHostToDevice);
cudaMemcpy(bd,bh,size,cudaMemcpyHostToDevice);
dim3 Block(8,8,1);
dim3 Grid(4,4);
Sharesum<<<Grid,Block>>>(ad,bd,cd,n);
cudaMemcpy(ch,cd,size,cudaMemcpyDeviceToHost);
printf("Addition with Shared Memory");
for(i=0;i<z;i++)
{
    if(i%n==0)
    {
        printf("\n\n");
    }
    printf(" %d ",cd[i]);
}
printf("\n");
free(ah);
free(bh);
free(ch);
cudaFree(ad);
cudaFree(bd);
cudaFree(cd);
getch();
}

```

Output:-



```
CA. C:\Windows\system32\cmd.exe
Addition with Shared Memory
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
```

2- Matrix Multiplication

single block without shared memory, single block with shared memory

multiple block without shared memory, multiple block with shared memory

/* single block without shared memory */

```
#include<stdio.h>
#include<conio.h>
#include<cuda.h>

__global__ void mul(int* a,int* b,int* c,int n)
{
    __shared__ int as[4][4],bs[4][4];
    int tx=threadIdx.x;
    int ty=threadIdx.y;
    int multi=0;
    for(int k=0;k<n;k++)
    {
        as[ty][tx]=a[ty*n+k];
        bs[ty][tx]=b[tx+n*k];
        multi +=as[ty][tx]*bs[ty][tx];
    }
    c[ty*n+tx]=multi;
}

main()
{
```

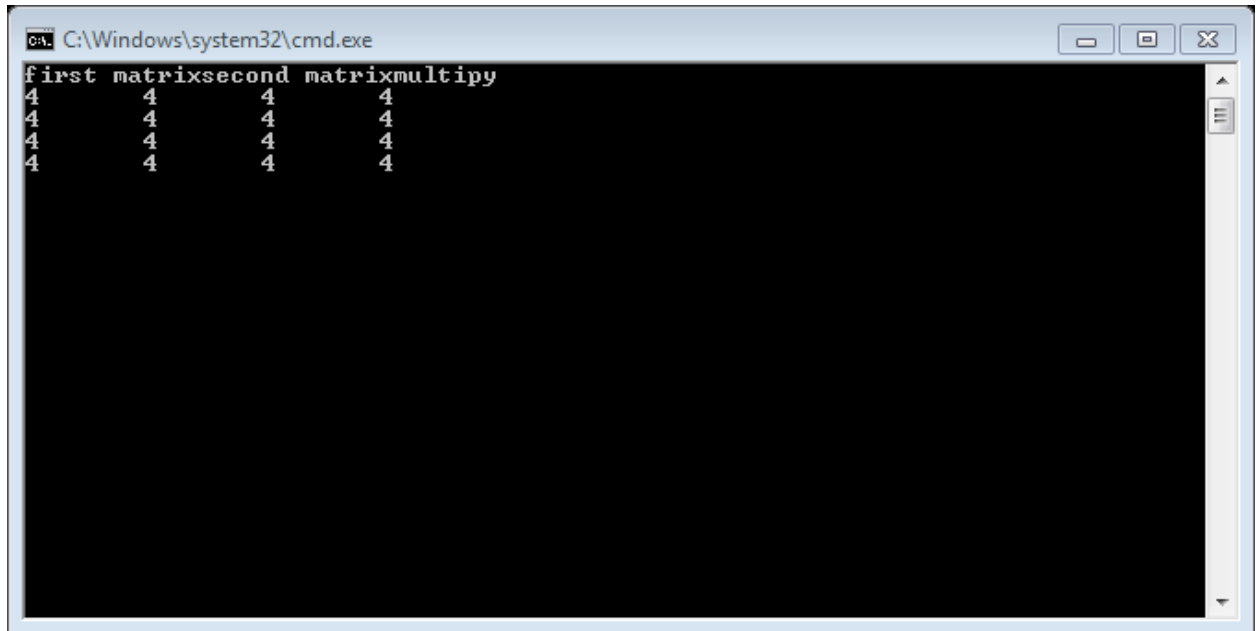
```

int *a_h,*b_h,*c_h,*a_d,*b_d,*c_d;
int i,n=4,z;
z=n*n;
size_t size=sizeof(int)*z;
a_h=(int*)malloc(size);
b_h=(int*)malloc(size);
c_h=(int*)malloc(size);
cudaMalloc((void**)&a_d,size);
cudaMalloc((void**)&b_d,size);
cudaMalloc((void**)&c_d,size);
printf("first matrix");
for(i=0;i<z;i++)
{
a_h[i]=1;
}
printf("second matrix");
for(i=0;i<z;i++)
{
b_h[i]=1;
}
cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);
cudaMemcpy(b_d,b_h,size,cudaMemcpyHostToDevice);
dim3 dimBlock(n,n,1);
dim3 dimGrid(1,1);
mul<<<dimGrid,dimBlock>>>(a_d,b_d,c_d,n);
cudaMemcpy(c_h,c_d,size,cudaMemcpyDeviceToHost);
printf("multiply");
for(i=0;i<z;i++)
{
if(i%n==0)
{
printf("\n");
}
printf("%d\t",c_h[i]);
}
free(a_h);
free(b_h);
free(c_h);
cudaFree(a_d);
cudaFree(b_d);
cudaFree(c_d);
getch();

```

}

Output:-



```
C:\Windows\system32\cmd.exe
first matrixsecond matrixmultiply
4444
4444
4444
4444
```

/* multiple block without shared memory */

```
#include<stdio.h>
#include<conio.h>
#include<cuda.h>

__global__ void mul(int* a,int* b,int* c,int n)
{
    __shared__ int as[2][2],bs[2][2];
    int tx=threadIdx.x;
    int ty=threadIdx.y;
    int bx=blockIdx.x;
    int by=blockIdx.x;
    int multi=0;
    bx=blockIdx.x*2+tx;
    by=blockIdx.y*2+ty;
    for(int k=0;k<n;k++)
    {
        as[ty][tx]=a[by * n + k];
        bs[ty][tx]=b[bx + n * k];
```

```

        multi +=as[ty][tx]*bs[ty][tx];
    }
    c[by*n+bx]=multi;
}
main()
{
    int *a_h,*b_h,*c_h,*a_d,*b_d,*c_d;
    int i,n=4,z;
    z=n*n;
    size_t size=sizeof(int)*z;
    a_h=(int*)malloc(size);
    b_h=(int*)malloc(size);
    c_h=(int*)malloc(size);
    cudaMalloc((void**)&a_d,size);
    cudaMalloc((void**)&b_d,size);
    cudaMalloc((void**)&c_d,size);
    printf("first matrix");
    for(i=0;i<z;i++)
    {
        a_h[i]=1;
    }
    printf("second matrix");
    for(i=0;i<z;i++)
    {
        b_h[i]=1;
    }
    cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);
    cudaMemcpy(b_d,b_h,size,cudaMemcpyHostToDevice);
    dim3 dimBlock(2,2,1);
    dim3 dimGrid(2,2);
    mul<<<dimGrid,dimBlock>>>(a_d,b_d,c_d,n);
    cudaMemcpy(c_h,c_d,size,cudaMemcpyDeviceToHost);
    printf("multiply");
    for(i=0;i<z;i++)
    {
        if(i%n==0)
        {
            printf("\n");
        }
        printf("%d\t",c_h[i]);
    }
    free(a_h);

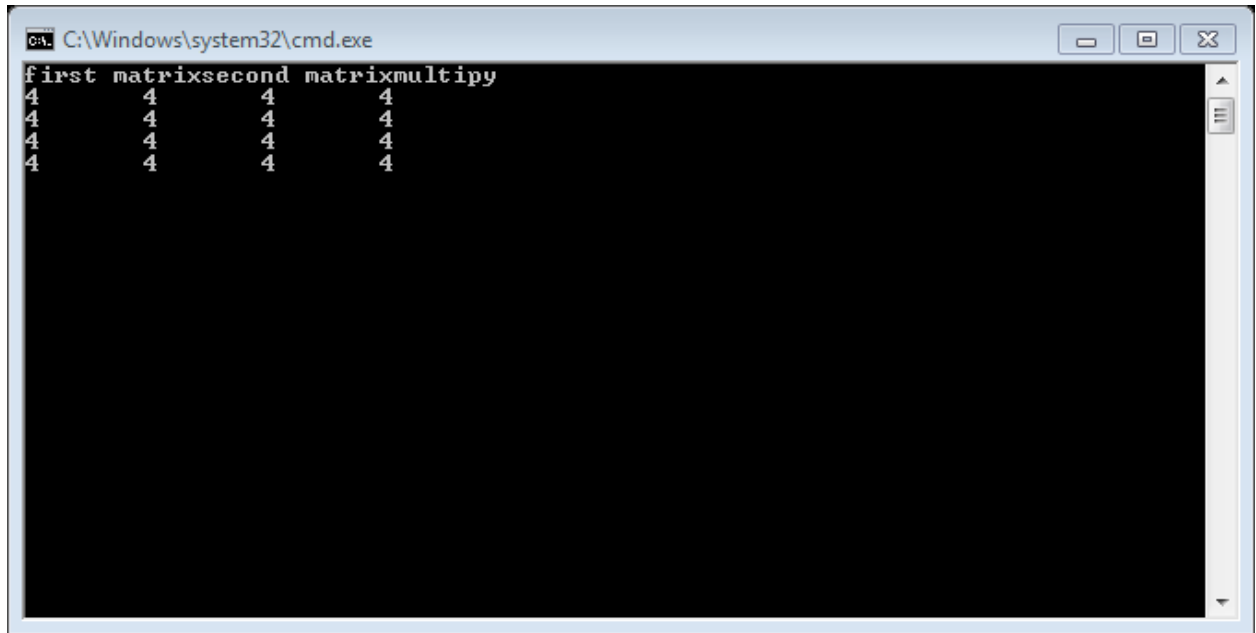
```

```

free(b_h);
free(c_h);
cudaFree(a_d);
cudaFree(b_d);
cudaFree(c_d);
getch();
}

```

Output:-



```

C:\Windows\system32\cmd.exe
first matrixsecond matrixmultiply
4      4      4      4
4      4      4      4
4      4      4      4
4      4      4      4

```

/* single block with shared memory */

```

#include<stdio.h>
#include<conio.h>
#include<cuda.h>

__global__ void mul(int* a,int* b,int* c,int n)
{
    __shared__ int as[4][4],bs[4][4];
    int tx=threadIdx.x;
    int ty=threadIdx.y;
    int multi=0;
    for(int k=0;k<n;k++)
    {
        as[ty][tx]=a[ty*n+k];
        bs[ty][tx]=b[tx+n*k];
        multi +=as[ty][tx]*bs[ty][tx];
    }
}

```

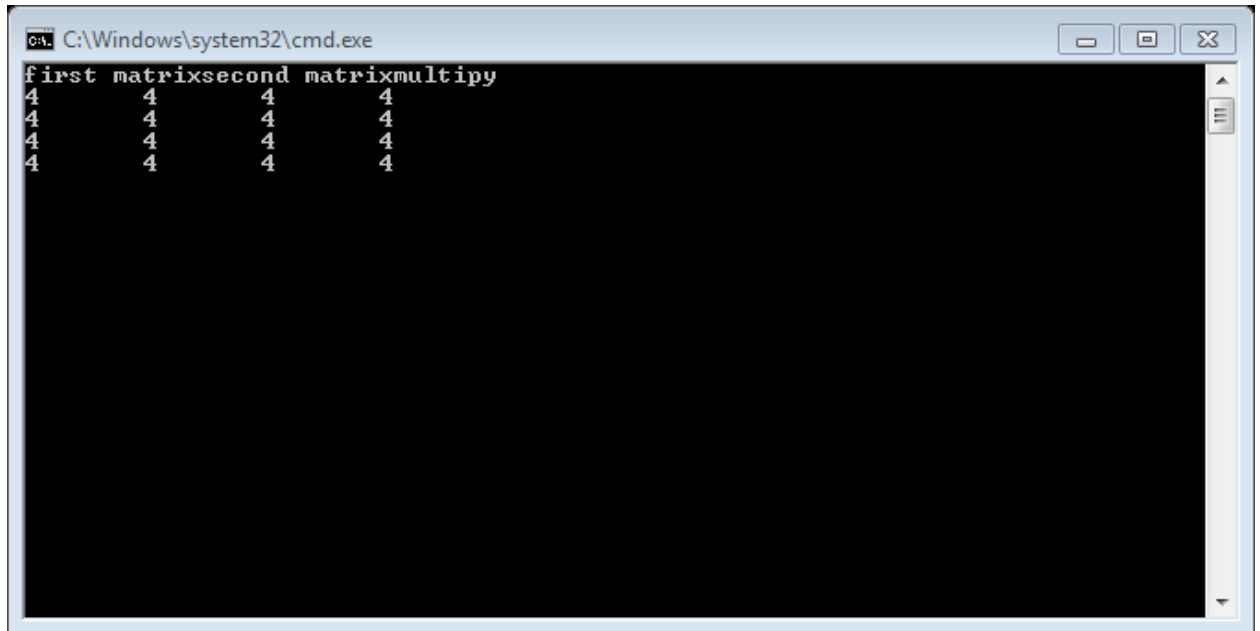
```

        c[ty*n+tx]=multi;
    }
main()
{
    int *a_h,*b_h,*c_h,*a_d,*b_d,*c_d;
    int i,n=4,z;
    z=n*n;
    size_t size=sizeof(int)*z;
    a_h=(int*)malloc(size);
    b_h=(int*)malloc(size);
    c_h=(int*)malloc(size);
    cudaMalloc((void**)&a_d,size);
    cudaMalloc((void**)&b_d,size);
    cudaMalloc((void**)&c_d,size);
    printf("first matrix");
    for(i=0;i<z;i++)
    {
        a_h[i]=1;
    }
    printf("second matrix");
    for(i=0;i<z;i++)
    {
        b_h[i]=1;
    }
    cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);
    cudaMemcpy(b_d,b_h,size,cudaMemcpyHostToDevice);
    dim3 dimBlock(n,n,1);
    dim3 dimGrid(1,1);
    mul<<<dimGrid,dimBlock>>>(a_d,b_d,c_d,n);
    cudaMemcpy(c_h,c_d,size,cudaMemcpyDeviceToHost);
    printf("multiply");
    for(i=0;i<z;i++)
    {
        if(i%n==0)
        {
            printf("\n");
        }
        printf("%d\t",c_h[i]);
    }
    free(a_h);
    free(b_h);
    free(c_h);
}

```

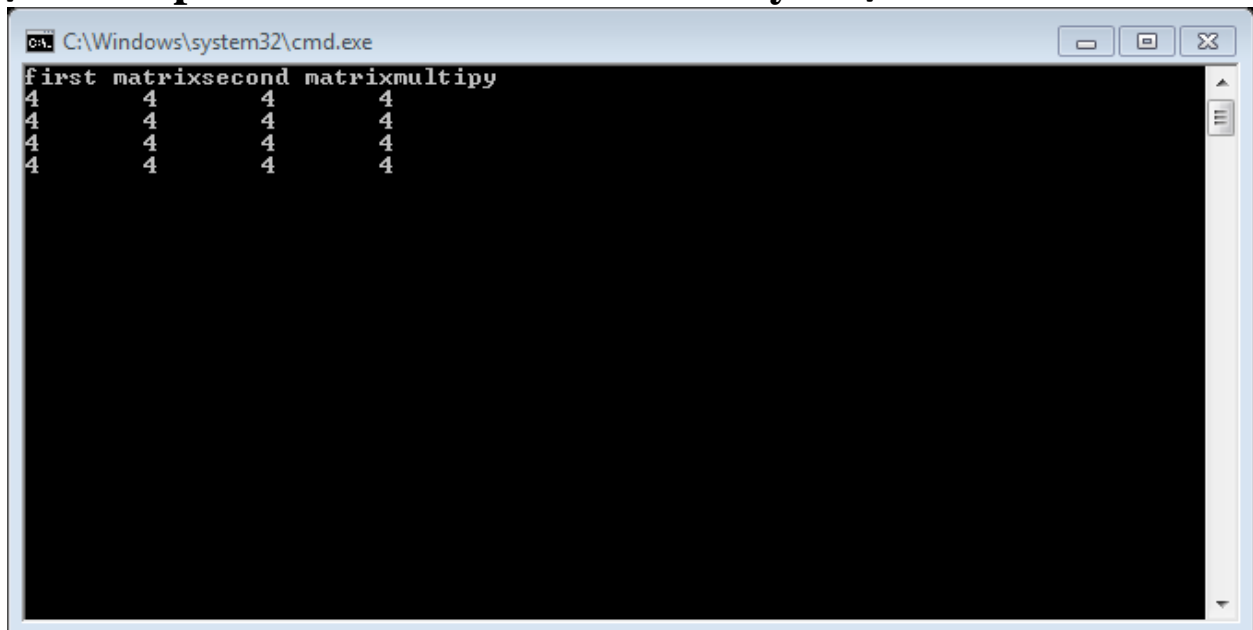
```
cudaFree(a_d);  
cudaFree(b_d);  
cudaFree(c_d);  
getch();  
}
```

Output:-



```
C:\Windows\system32\cmd.exe  
first matrixsecond matrixmultiply  
4444  
4444  
4444  
4444
```

/* multiple block with shared memory */



```
C:\Windows\system32\cmd.exe  
first matrixsecond matrixmultiply  
4444  
4444  
4444  
4444
```

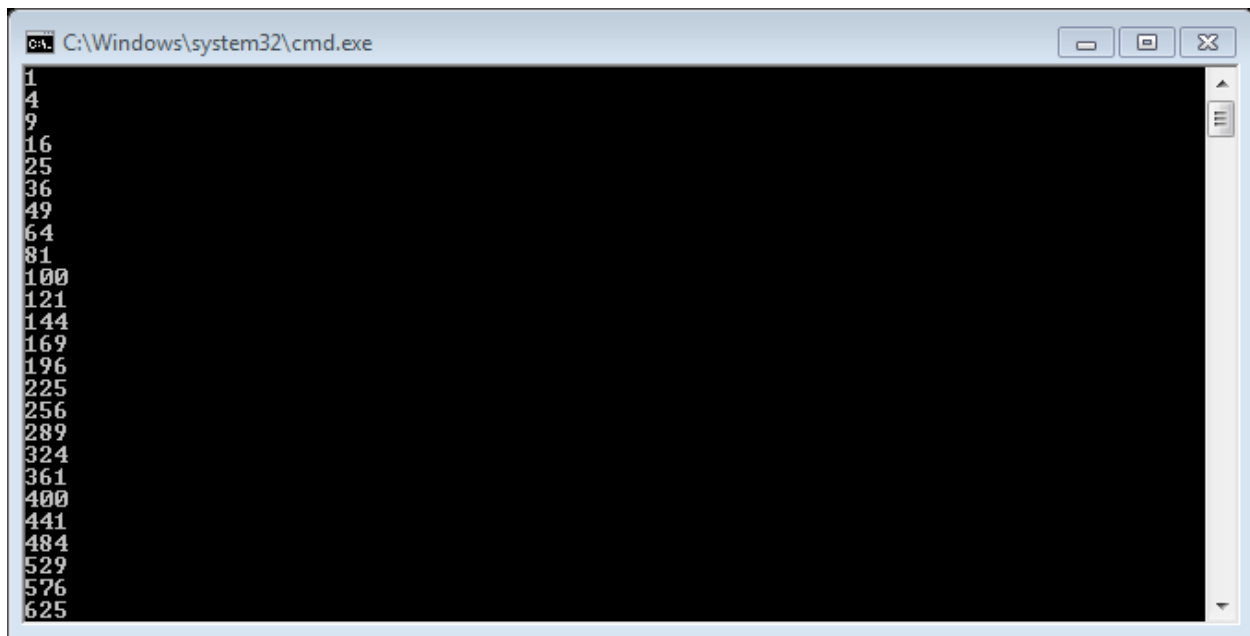
3-Square of first 100 integers

single block without shared memory, single block with shared memory

multiple block without shared memory, multiple block with shared memory

/* single block without shared memory */

```
#include<stdio.h>
#include<conio.h>
#include<cuda.h>
__global__ void seq(int* a,int* b,int n)
{
int id=threadIdx.x;
b[id]=a[id]*a[id];
}
main()
{
int *a_h,*b_h,*a_d,*b_d;
int i,n=100;
size_t size=sizeof(int)*n;
a_h=(int*)malloc(size);
b_h=(int*)malloc(size);
cudaMalloc((void**)&a_d,size);
cudaMalloc((void**)&b_d,size);
for(i=1;i<=100;i++)
{
a_h[i]=i;
}
cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);
dim3 Block(100,1,1);
dim3 Grid(1,1);
seq<<<Grid,Block>>>(a_d,b_d,n);
cudaMemcpy(b_h,b_d,size,cudaMemcpyDeviceToHost);
for(i=1;i<=100;i++)
{
printf("%d\n",b_h[i]);
}
free(a_h);
free(b_h);
cudaFree(a_d);
cudaFree(b_d);
getch();
output:-
```

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. It displays a list of square numbers from 1 to 625, arranged in a single column. The numbers are: 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, and 625. The window includes standard Windows window controls (minimize, maximize, close) in the top right corner.

/* single block with shared memory */

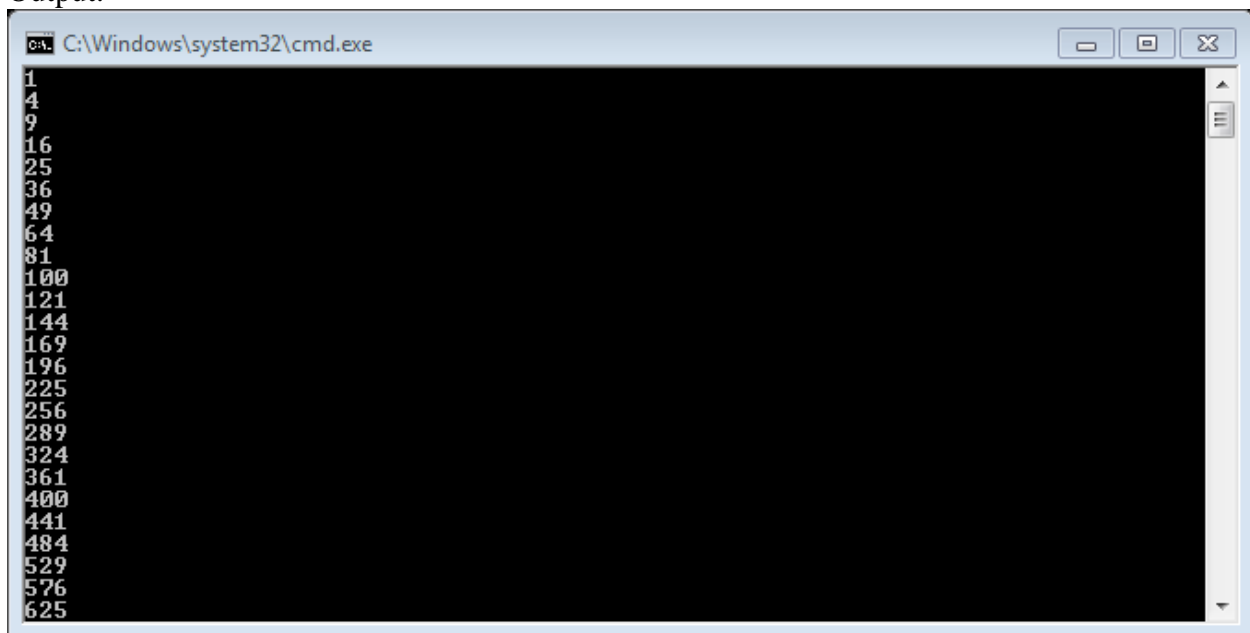
```
#include<stdio.h>
#include<conio.h>
#include<cuda.h>
__global__ void seq(int* a,int* b,int n)
{
    __shared__ int as[100][1];
    int tx=threadIdx.x;
    int ty=threadIdx.y;
    as[ty][tx]=a[tx]*a[tx];
    b[tx]=as[ty][tx];
}
main()
{
    int *a_h,*b_h,*a_d,*b_d;
    int i,n=100;
    size_t size=sizeof(int)*n;
    a_h=(int*)malloc(size);
    b_h=(int*)malloc(size);
    cudaMalloc((void**)&a_d,size);
    cudaMalloc((void**)&b_d,size);
    for(i=1;i<=100;i++)
    {
        a_h[i]=i;
    }
    cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);
    dim3 Block(100,1,1);
```

```

dim3 Grid(1,1);
seq<<<Grid,Block>>>(a_d,b_d,n);
cudaMemcpy(b_h,b_d,size,cudaMemcpyDeviceToHost);
for(i=1;i<=100;i++)
{
printf("%d\n",b_h[i]);
}
free(a_h);
free(b_h);
cudaFree(a_d);
cudaFree(b_d);
getch();
}

```

Output:-



```

C:\Windows\system32\cmd.exe
1
4
9
16
25
36
49
64
81
100
121
144
169
196
225
256
289
324
361
400
441
484
529
576
625

```

/* multiple block with shared memory */

```

#include<stdio.h>
#include<conio.h>
#include<cuda.h>
__global__ void seq(int* a,int* b,int n)
{
__shared__ int as[100][1];
int tx=threadIdx.x;
int ty=threadIdx.y;
as[ty][tx]=a[tx]*a[tx];
b[tx]=as[ty][tx];

}
main()

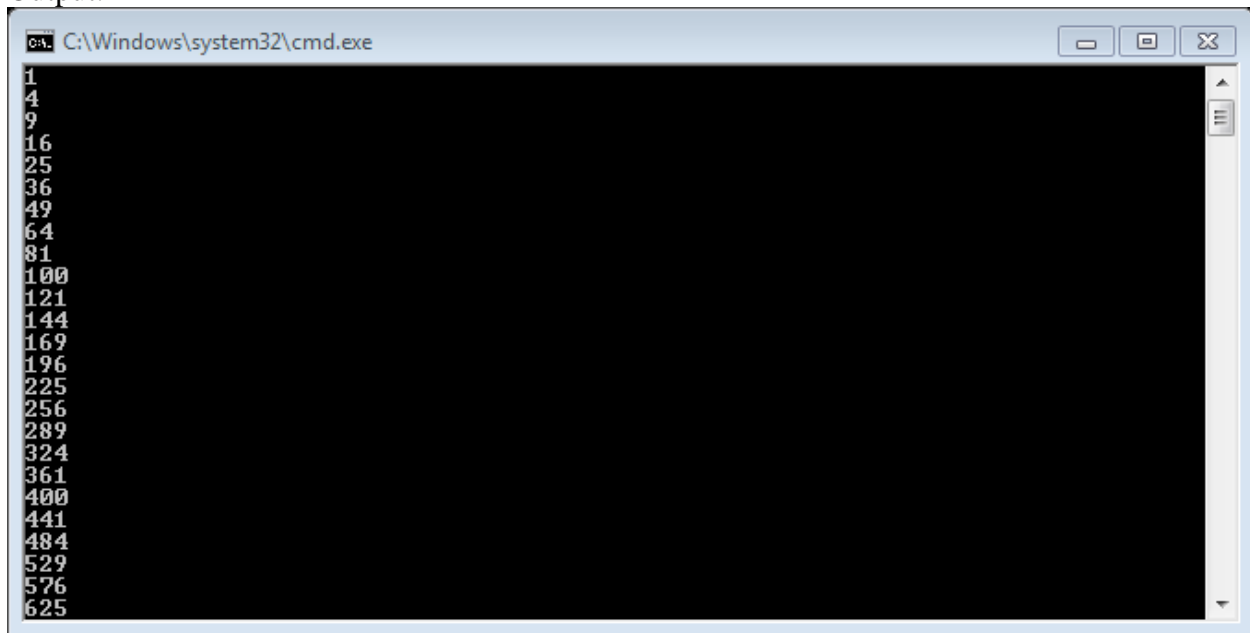
```

```

{
int *a_h,*b_h,*a_d,*b_d;
int i,n=100;
size_t size=sizeof(int)*n;
a_h=(int*)malloc(size);
b_h=(int*)malloc(size);
cudaMalloc((void**)&a_d,size);
cudaMalloc((void**)&b_d,size);
for(i=1;i<=100;i++)
{
a_h[i]=i;
}
cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);
dim3 Block(100,1,1);
dim3 Grid(1,1);
seq<<<Grid,Block>>>(a_d,b_d,n);
cudaMemcpy(b_h,b_d,size,cudaMemcpyDeviceToHost);
for(i=1;i<=100;i++)
{
printf("%d\n",b_h[i]);
}
free(a_h);
free(b_h);
cudaFree(a_d);
cudaFree(b_d);
getch();
}

```

Output:-



```

C:\Windows\system32\cmd.exe
1
4
9
16
25
36
49
64
81
100
121
144
169
196
225
256
289
324
361
400
441
484
529
576
625

```

4-Matrix Transpose

single block without shared memory, single block with shared memory

multiple block without shared memory, multiple block with shared memory

/* single block without shared memory */

```
//Transpose Matrix
#include<stdio.h>
#include<cuda.h>
#include<conio.h>
__global__ void Transport(int* a,int* b,int n)
{
    int h;
    h=a[threadIdx.x + n * threadIdx.y];
    b[threadIdx.y + n *threadIdx.x]=h;
}
main()
{
    int *a_h,*b_h,*a_d,*b_d;
    int i,n,z;
    printf("Enter Any No.:-");
    scanf("%d",&n);
    z=n*n;
    size_t size=sizeof(int)*z;
    a_h=(int*)malloc(size);
    b_h=(int*)malloc(size);
    cudaMalloc((void**)&a_d,size);
    cudaMalloc((void**)&b_d,size);
    printf("Enter Array");
    for(i=0;i<z;i++)
    {
        scanf("%d",&a_h[i]);
    }
    cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);

    dim3 dimBlock(n,n,1);

    dim3 dimGrid(1,1);

    Transport<<<dimGrid,dimBlock>>>(a_d,b_d,n);
    cudaMemcpy(b_h,b_d,size,cudaMemcpyDeviceToHost);
    printf("Transpose");
    for(i=0;i<z;i++)
    {
        if(i%n==0)
        {
            printf("\n");
        }
    }
}
```

```

    }
    printf(" %d ",b_h[i]);
}
free(a_h);
free(b_h);
cudaFree(a_d);
cudaFree(b_d);
getch();
return 0;
}

```

Output:-

```

C:\Windows\system32\cmd.exe
Enter Any No.:-3
Enter Array3
4
5
6
6
7
6
5
4
Transpose
3 6 6
4 6 5
5 7 4

```

/*, single block with shared memory */

```

#include<stdio.h>
#include<cuda.h>
#include<conio.h>
__global__ void transpose(int* a,int* b,int n)

{
    __shared__ int as[3][3];
    int tx=threadIdx.x;
    int ty=threadIdx.y;

    as[ty][tx]=a[ty*n+tx];
    b[ty+tx*n]=as[ty][tx];
}

```

```

int main()
{
    int *a_h,*a_d,*b_h,*b_d;
    int i,n=3,z;

    z=n*n;
    size_t size=sizeof(int)*z;
    a_h=(int*)malloc(size);
    b_h=(int*)malloc(size);

    cudaMalloc((void**)&a_d,size);

    cudaMalloc((void**)&b_d,size);


    printf("Enter 1st Matrix");
    for(i=0;i<z;i++)
    {
        scanf("%d",&a_h[i]);
    }


    cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);


    dim3 dimBlock(n,n,1);

    dim3 dimGrid(1,1);

    transpose<<<dimGrid,dimBlock>>>(a_d,b_d,n);
    cudaMemcpy(b_h,b_d,size,cudaMemcpyDeviceToHost);


    printf("transpose");
    for(i=0;i<z;i++)
    {
        if(i%n==0)
        {
            printf("\n");
        }
        printf(" %d ",b_h[i]);
    }
}

```

```

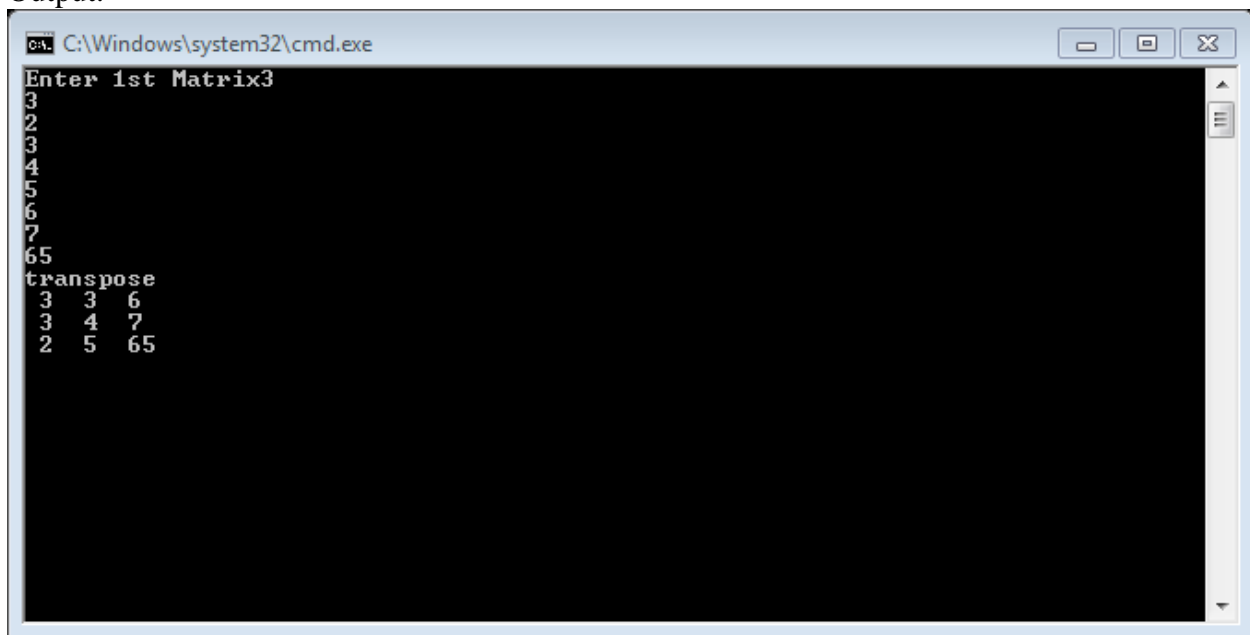
        free(a_h);
        free(b_h);

        cudaFree(b_d);
        cudaFree(a_d);

        getch();
        return 0;
}

```

Output:-



```

C:\Windows\system32\cmd.exe
Enter 1st Matrix3
3
2
3
4
5
6
7
65
transpose
3 3 6
3 4 7
2 5 65

```

/* multiple block with shared memory */

```

#include<stdio.h>
#include<cuda.h>
#include<conio.h>
__global__ void transpose(int* a,int* b,int n)

{
    __shared__ int as[2][2];
    int tx=threadIdx.x;
    int ty=threadIdx.y;

    int bx=blockIdx.x*2+tx;
    int by=blockIdx.y*2+ty;
    as[ty][tx]=a[by*n+bx];
}

```

```

        b[by+bx*n]=as[ty][tx];
    }

int main()
{
    int *a_h,*a_d,*b_h,*b_d;
    int i,n=4,z;

    z=n*n;
    size_t size=sizeof(int)*z;
    a_h=(int*)malloc(size);
    b_h=(int*)malloc(size);

    cudaMalloc((void**)&a_d,size);

    cudaMalloc((void**)&b_d,size);


    printf("Enter 1st Matrix=");
    for(i=0;i<z;i++)
    {
        scanf("%d",&a_h[i]);
    }


    cudaMemcpy(a_d,a_h,size,cudaMemcpyHostToDevice);


    dim3 dimBlock(2,2,1);

    dim3 dimGrid(2,2);

    transpose<<<dimGrid,dimBlock>>>(a_d,b_d,n);
    cudaMemcpy(b_h,b_d,size,cudaMemcpyDeviceToHost);


    printf("transpose");
    for(i=0;i<z;i++)
    {
        if(i%n==0)
        {
            printf("\n");

```



```

        }
        printf(" %d ",b_h[i]);
    }

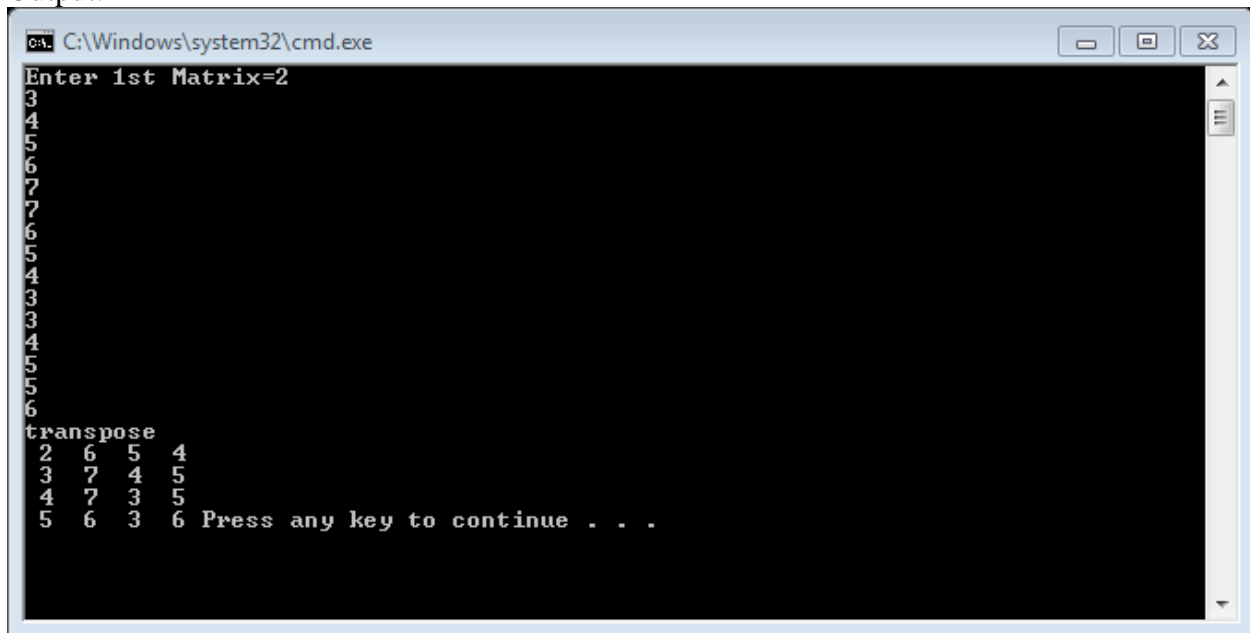
    free(a_h);
    free(b_h);

    cudaFree(b_d);
    cudaFree(a_d);

    getch();
    return 0;
}

```

Output:-



```

C:\Windows\system32\cmd.exe
Enter 1st Matrix=2
3
4
5
6
7
7
6
5
4
3
3
4
5
5
6
transpose
2 6 5 4
3 7 4 5
4 7 3 5
5 6 3 6 Press any key to continue . . .

```