More sample programs

1.Write a CUDA program to print â€œHello Worldâ€ using one thread and one block

2.Write a CUDA program for vector addition of 2 numbers through reference variables.

3.Write a CUDA program Vector addition of 2 numbers using one thread and one block

4.Write a CUDA program vector addition of numbers using 1 thread and multiple blocks

5.Write a CUDA program for vector subtraction of numbers using 1 block and multiple threads

6.Write a CUDA program vector multiplication of numbers using multiple blocks and multiple threads

7.Write a CUDA program pairwise sum of elements of vector to showcase concept of syncthreads

8.Write a CUDA program for dot product using 1 block to showcase concept of shared memory

9.Write a CUDA program to showcase use of 2D threads and 2D blocks

10.
Write a CUDA program to Sort the elements using odd-even Sort

1. Write a CUDA program to print Hello World using one thread and one block

```
      #include<iostream>
using namespace std;
__global__ void printHello(){
}
int main(){
      printHello<<<1,1>>>();
```

```
cout<< Hello World ;
      return 0;
}
```

2. Write a CUDA program for vector addition of 2 numbers through reference variables.

```
#include<iostream>
using namespace std;

void addFun(int *a, int *b){
      int s = *a+*b;
      cout<<s<<endl;
}

int main(){
      int a,b;
      a = 10;
      b = 5;
      addFun(&a,&b);
}
```

3. Write a CUDA program Vector addition of 2 numbers using one thread and one block

```
#include<iostream>
using namespace std;
__global__ void add(int *a, int *b, int *c){
*c = *a + *b;
}
int main(){
      int a,b,c; // host variables
      a = 5;
      b = 10;
      int *da, *db, *dc; // device variables
      // memory allocation
      cudaMalloc(&da, sizeof(int));
      cudaMalloc(&db, sizeof(int));
      cudaMalloc(&dc, sizeof(int));
      // copying memory to destination from source
      cudaMemcpy(da, &a, sizeof(int), cudaMemcpyHostToDevice);
      cudaMemcpy(db, &b, sizeof(int), cudaMemcpyHostToDevice);
      add<<<1,1>>>(da,db,dc);
      cudaMemcpy(&c, dc, sizeof(int), cudaMemcpyDeviceToHost);
      cout<<c<<endl;
      cudaFree(da);
```

```
      cudaFree(db);
      cudaFree(dc);
      return 0;
}
```

4. Write a CUDA program vector addition of numbers using 1 thread and multiple blocks

```
#include<iostream>
using namespace std;
__global__ void add(int *a, int *b, int *c){
      int i = blockIdx.x;
      c[i] = a[i]+b[i];
}
int main(){
      int c[6];
      int a[6] = {1,2,3,4,5,6};
      int b[6] = {11,12,13,14,15,16};
      int *da, *db, *dc;
      cudaMalloc(&da, 6*sizeof(int));
      cudaMalloc(&db, 6*sizeof(int));
      cudaMalloc(&dc, 6*sizeof(int));
      cudaMemcpy(da, &a, 6*sizeof(int), cudaMemcpyHostToDevice);
      cudaMemcpy(db, &b, 6*sizeof(int), cudaMemcpyHostToDevice);
      add<<<6,1>>>(da,db,dc);
      cudaMemcpy(&c, dc, 6*sizeof(int), cudaMemcpyDeviceToHost);
      for (int j=0; j<6; j++){
            cout<<a[j]<<" + "<<b[j]<<" = "<<c[j]<<endl;
      }
      cudaFree(da);
      cudaFree(db);
      cudaFree(dc);
      return 0;
}
```

5. Write a CUDA program for vector subtraction of numbers using 1 block and multiple threads

```
#include<iostream>
using namespace std;
__global__ void sub(int *a, int *b, int *c){
      int i = threadIdx.x;
      c[i] = b[i]-a[i];
}
int main(){
      int a[6],b[6],c[6];
      for(int i=0; i<6; i++){
            a[i] = 3*i+28;
```

```
            b[i] = 5*i+69;
      }
      int *da, *db, *dc;
      cudaMalloc(&da, 6*sizeof(int));
      cudaMalloc(&db, 6*sizeof(int));
      cudaMalloc(&dc, 6*sizeof(int));
      cudaMemcpy(da, &a, 6*sizeof(int), cudaMemcpyHostToDevice);
      cudaMemcpy(db, &b, 6*sizeof(int), cudaMemcpyHostToDevice);
      sub<<<1,6>>>(da,db,dc);
      cudaMemcpy(&c, dc, 6*sizeof(int), cudaMemcpyDeviceToHost);
      for (int j=0; j<6; j++){
            cout<<b[j]<<" - "<<a[j]<<" = "<<c[j]<<endl;
      }
      cudaFree(da);
      cudaFree(db);
      cudaFree(dc);
      return 0;
}
```

6. Write a CUDA program vector multiplication of numbers using multiple blocks and multiple threads

```
      #include<iostream>
using namespace std;
__global__ void mul(int *a, int *b, int *c){
      int j = blockDim.x;
      // blockDim specifies no. of threads in each block
      int i = blockIdx.x*j + threadIdx.x;
      c[i] = b[i]*a[i];
      // c[i] = i;
}
int main(){
      int a[6],b[6],c[6];
      for(int i=0; i<6; i++){
            a[i] = 2*i+11;
            b[i] = 4*i+7;
      }
      int *da, *db, *dc;
      cudaMalloc(&da, 6*sizeof(int));
      cudaMalloc(&db, 6*sizeof(int));
      cudaMalloc(&dc, 6*sizeof(int));
      cudaMemcpy(da, &a, 6*sizeof(int), cudaMemcpyHostToDevice);
      cudaMemcpy(db, &b, 6*sizeof(int), cudaMemcpyHostToDevice);
      mul<<<2,3>>>(da,db,dc);
      cudaMemcpy(&c, dc, 6*sizeof(int), cudaMemcpyDeviceToHost);
      for (int j=0; j<6; j++){
            cout<<b[j]<<" * "<<a[j]<<" = "<<c[j]<<endl;
      }
      cudaFree(da);
      cudaFree(db);
```

```
	cudaFree(dc);
	return 0;
}
```

7.Write a CUDA program pairwise sum of elements of vector to showcase concept of syncthreads.

```
#include<iostream>
using namespace std;
__global__ void fun(int *a, int *b){
	int t = threadIdx.x;
	int n = blockDim.x;
	while(n!=0){
		if (t<n){
	// eg. a[0] += a[0+n], similary for other indices, this
would resuse the array again and again and keep on adding values.
		a[t] += a[t+n];
		}
		__syncthreads();
		n = n/2;
	}
	*b = a[0];
}
int main(){
	int N = 8;
	int a[N], b;
	for(int i=0; i<N; i++){
		a[i] = 2*i+11;
	}
	int *da, *db;
	cudaMalloc(&da, N*sizeof(int));
	cudaMalloc(&db, sizeof(int));
	cudaMemcpy(da, &a, N*sizeof(int), cudaMemcpyHostToDevice);
	cudaMemcpy(db, &b, sizeof(int), cudaMemcpyHostToDevice);
	fun<<<1,N/2>>>(da, db);
	cudaMemcpy(&b, db, sizeof(int), cudaMemcpyDeviceToHost);
	cout<<"Res: "<<b<<endl;
	cudaFree(da);
	cudaFree(db);
	return 0;
}
```

8. Write a CUDA program for dot product using 1 block to showcase concept of shared memory.

```
#include<iostream>
```

```
using namespace std;
__global__ void dot_product(int *a, int *b, int *c){
      int i = threadIdx.x;
      // this allows accessing shared memory of all the threads of
a block
      __shared__ int temp[6];
      temp[i] = b[i] * a[i];
      // this will ensure completion of all threads
      __syncthreads();
      if (threadIdx.x == 0){
            int res = 0;
            for (int i=0; i<6; i++){
                  res += temp[i];
            }
            *c = res;
      }
}

int main(){
      int size = 6;
      int a[size],b[size],c;
      cout<<"Enter elements of a: ";
      for(int i=0; i<size; i++){
            cin>>a[i];
      }
      cout<<"Enter elements of b: ";
      for(int i=0; i<size; i++){
            cin>>b[i];
      }

      int *da, *db, *dc;
      cudaMalloc(&da, size*sizeof(int));
      cudaMalloc(&db, size*sizeof(int));
      cudaMalloc(&dc, sizeof(int)
             cudaMemcpy(da, &a, size*sizeof(int),
cudaMemcpyHostToDevice);
      cudaMemcpy(db, &b, size*sizeof(int),
cudaMemcpyHostToDevice);
      dot_product<<<1,6>>>(da,db,dc);
      cudaMemcpy(&c, dc, sizeof(int), cudaMemcpyDeviceToHost);
      cout<<c<<endl;
      cudaFree(da);
      cudaFree(db);
      cudaFree(dc);
      return 0;
}
```

9. Write a CUDA program to showcase use of 2D threads and 2D blocks

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

```
#define BLOCK_SIZE 16

__global__ static void AddKernel(float *d_Buff1, float *d_Buff2,
float *d_Buff3, size_t pitch, int iMatSizeM, int iMatSizeN)
{

      const int tidx = blockDim.x * blockIdx.x + threadIdx.x;
      const int tidy = blockDim.y * blockIdx.y + threadIdx.y;

      int index = pitch/sizeof(float);

      if(tidx<iMatSizeM && tidy<iMatSizeN)
      {
            d_Buff3[tidx * index  + tidy] = d_Buff1[tidx * index +
tidy] + d_Buff2[tidx * index + tidy];
      }

}

void printMatrix(float *pflMat, int iMatSizeM, int iMatSizeN)
{
       for(int idxM = 0; idxM < iMatSizeM; idxM++)
      {
            for(int idxN = 0; idxN < iMatSizeN; idxN++)
            {
                  printf("%f\t",pflMat[(idxM * iMatSizeN) + idxN]);
            }
            printf("\n");
      }
      printf("\n");
}

int main()
{
       int iMatSizeM=0,iMatSizeN=0;
  printf("Enter size of Matrix(M*N):");
       scanf("%d %d",&iMatSizeM,&iMatSizeN);
  float *h_flMat1 = (float*)malloc(iMatSizeM * iMatSizeN *
sizeof(float));
       float *h_flMat2 = (float*)malloc(iMatSizeM * iMatSizeN *
sizeof(float));
       float *h_flMatSum = (float*)malloc(iMatSizeM * iMatSizeN *
sizeof(float));

       for(int j=0;j<(iMatSizeM*iMatSizeN);j++)
       {
                h_flMat1[j]=(float)rand()/(float)RAND_MAX;
                h_flMat2[j]=(float)rand()/(float)RAND_MAX;
       }

       printf("Matrix 1\n");
       printMatrix(h_flMat1, iMatSizeM, iMatSizeN);
```

```
        printf("Matrix 2\n");
        printMatrix(h_flMat2, iMatSizeM, iMatSizeN);

        float *d_flMat1, *d_flMat2, *d_flMatSum;
        size_t d_MatPitch;

cudaMallocPitch((void**)&d_flMat1,
&d_MatPitch,iMatSizeN*sizeof(float),iMatSizeM);
cudaMallocPitch((void**)&d_flMat2,
&d_MatPitch,iMatSizeN*sizeof(float),iMatSizeM);
     cudaMallocPitch((void**)&d_flMatSum,
&d_MatPitch,iMatSizeN*sizeof(float),iMatSizeM);

        cudaMemcpy2D(d_flMat1,d_MatPitch,h_flMat1,iMatSizeN *
sizeof(float), iMatSizeN      * sizeof(float), iMatSizeM,
cudaMemcpyHostToDevice);
        cudaMemcpy2D(d_flMat2,d_MatPitch,h_flMat2,iMatSizeN *
sizeof(float), iMatSizeN * sizeof(float), iMatSizeM,
cudaMemcpyHostToDevice);

        dim3 blocks(1,1,1);
        dim3 threadsperblock(BLOCK_SIZE,BLOCK_SIZE,1);

        blocks.x=((iMatSizeM/BLOCK_SIZE) + (((iMatSizeM)%
BLOCK_SIZE)==0?0:1));
        blocks.y=((iMatSizeN/BLOCK_SIZE) + (((iMatSizeN)%
BLOCK_SIZE)==0?0:1));

        AddKernel<<<blocks, threadsperblock>>>(d_flMat1, d_flMat2,
d_flMatSum, d_MatPitch, iMatSizeM,iMatSizeN);

        cudaThreadSynchronize();

        cudaMemcpy2D(h_flMatSum, iMatSizeN *
sizeof(float),d_flMatSum, d_MatPitch, iMatSizeN * sizeof(float),
iMatSizeM, cudaMemcpyDeviceToHost);

        cudaFree(d_flMat1);
        cudaFree(d_flMat2);
        cudaFree(d_flMatSum);

        printf("Matrix Sum\n");
        printMatrix(h_flMatSum, iMatSizeM, iMatSizeN);
}
```

10.Write a CUDA program to Sort the elements using odd-even Sort.

```
#include<stdio.h>
#include<cuda.h>
#define N 5
```

```
#define intswap(A,B) {int temp=A;A=B;B=temp;}

__global__ void sort(int *c,int *count)
{
    int l;
    if(*count%2==0)
          l=*count/2;
    else
         l=(*count/2)+1;
    for(int i=0;i<l;i++)
    {
            if((!(threadIdx.x&1)) && (threadIdx.x<(*count-1)))
//even phase(&1 will compare the least significant bit )
            {
                if(c[threadIdx.x]>c[threadIdx.x+1])
                  intswap(c[threadIdx.x], c[threadIdx.x+1]);
            }

            __syncthreads();
            if((threadIdx.x&1) && (threadIdx.x<(*count-1)))
//odd phase
            {
                if(c[threadIdx.x]>c[threadIdx.x+1])
                  intswap(c[threadIdx.x], c[threadIdx.x+1]);
            }
            __syncthreads();
    }

}



int main()
{int a[N],b[N],n;
    printf("enter size of array");
    scanf("%d",&n);
    if (n > N) {printf("too large!\n"); return 1;}
    printf("enter the elements of array");
  for(int i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
  }
  printf("ORIGINAL ARRAY : \n");
  for(int i=0;i<n;i++)
          {

          printf("%d ",a[i]);
          }
  int *c,*count;
  cudaMalloc((void**)&c,sizeof(int)*N);
  cudaMalloc((void**)&count,sizeof(int));
  cudaMemcpy(c,&a,sizeof(int)*N,cudaMemcpyHostToDevice);
```

```
  cudaMemcpy(count,&n,sizeof(int),cudaMemcpyHostToDevice);
  sort<<< 1,n >>>(c,count);
  cudaMemcpy(&b,c,sizeof(int)*N,cudaMemcpyDeviceToHost);
  printf("\nSORTED ARRAY : \n");
  for(int i=0;i<n;i++)
      {
         printf("%d ",b[i]);
      }

  printf("\n");
}
```