101803510 Aakash Agarwal COE23

Assignment-4

1. Write a CUDA program for to multiply each element of vector by 5 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;
                     }nv
                     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;
                     }
```

2. Write a CUDA program for 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;
}
```

3. Write a CUDA program for dot product using one 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;
}
```