Assignment 3

Varaprasad Kurra Panther ID: 002430487

Sravanthi Malepati Panther ID: 002538438

Olusesi Balogun Panther ID: 002541505

Question 1:

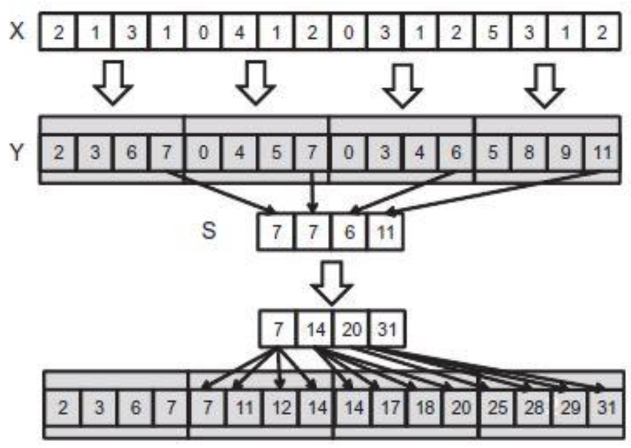
Write a CUDA program for calculating the pre_x sums of an arbitrary-length array as described in Section 8.6 of the Kirk and Hwu book. You should follow their code snippets and input-output conventions.

Solution: In this problem, we have an arbitrary size array of length N. We have implemented our solution in 3 steps which includes Kogge – Stone implementation as one of its important steps. We will look into each step below

1. A total of 16 input elements are divided into four scan blocks.

2. We can use the Kogge–Stone kernel, the Brent–Kung kernel, or the three-phase kernel to process the individual scan blocks. The kernel treats the four scan blocks as independent input data sets.

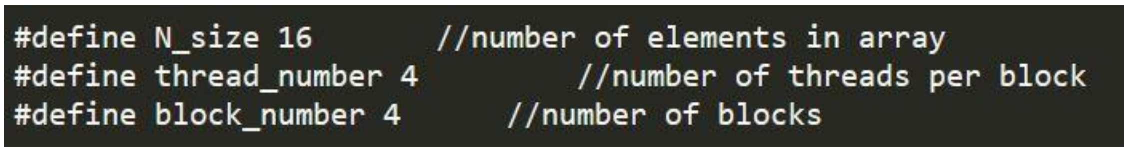
3. After the scan kernel terminates operation, each Y element contains the scan result within its scan block.



An example specifying the illustration of given example.

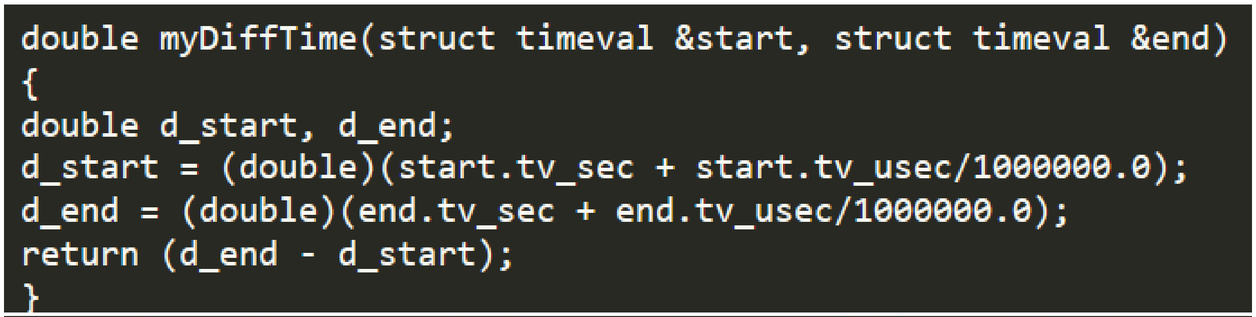
Implementation:

We have defined the constant's in our program as follows

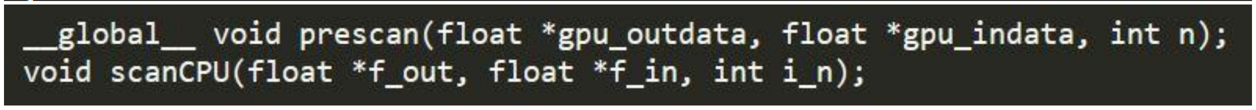


```
#define N_size 16       //number of elements in array
#define thread_number 4         //number of threads per block
#define block_number 4      //number of blocks
```

We have defined the following functions to implement the Algorithm

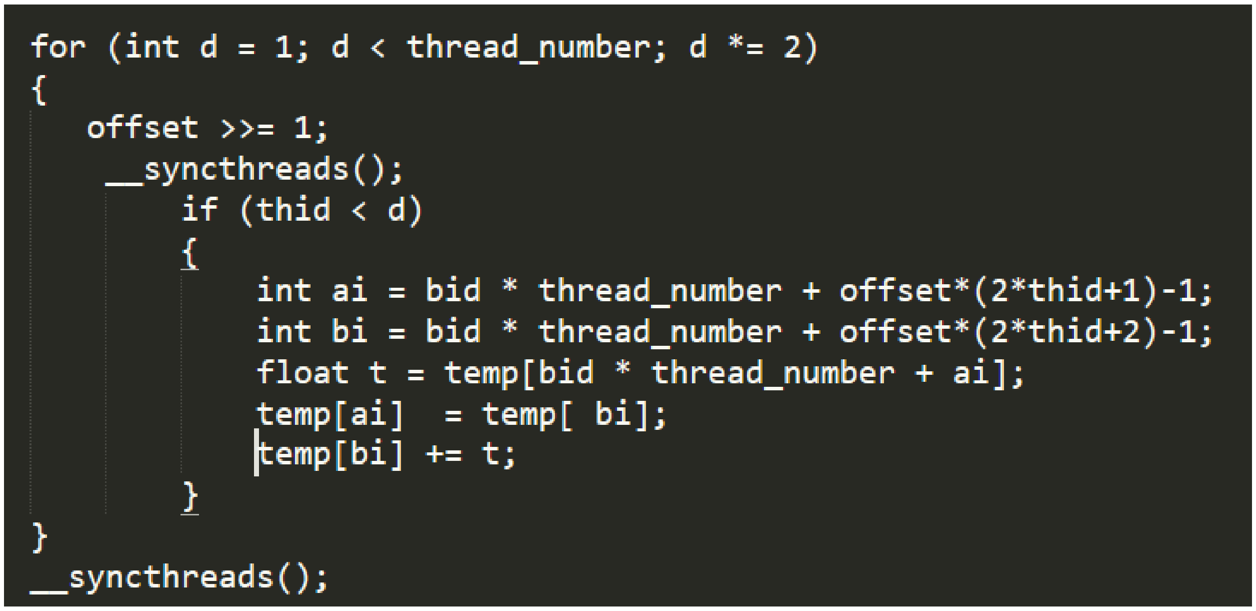


```
double myDiffTime(struct timeval &start, struct timeval &end)
{
double d_start, d_end;
d_start = (double)(start.tv_sec + start.tv_usec/1000000.0);
d_end = (double)(end.tv_sec + end.tv_usec/1000000.0);
return (d_end - d_start);
}
```



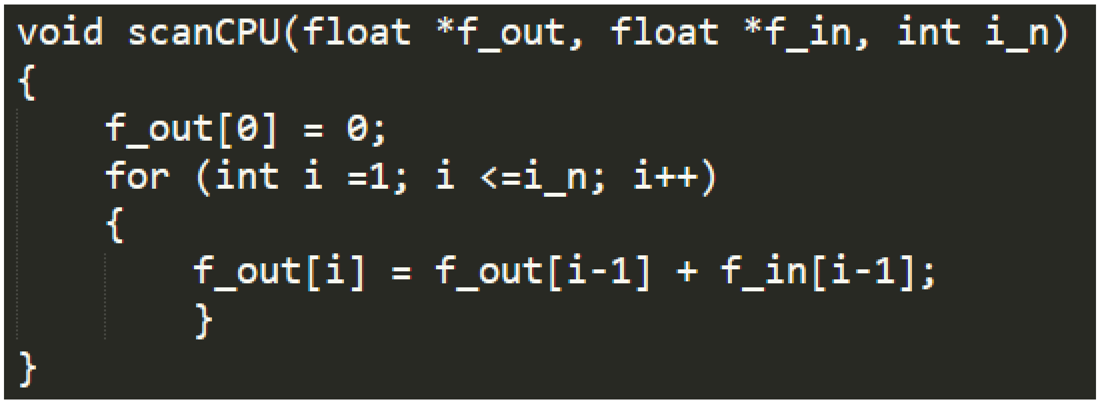
```
__global__ void prescan(float *gpu_outdata, float *gpu_indata, int n);
void scanCPU(float *f_out, float *f_in, int i_n);
```

Using the threads for carrying out the functionality.



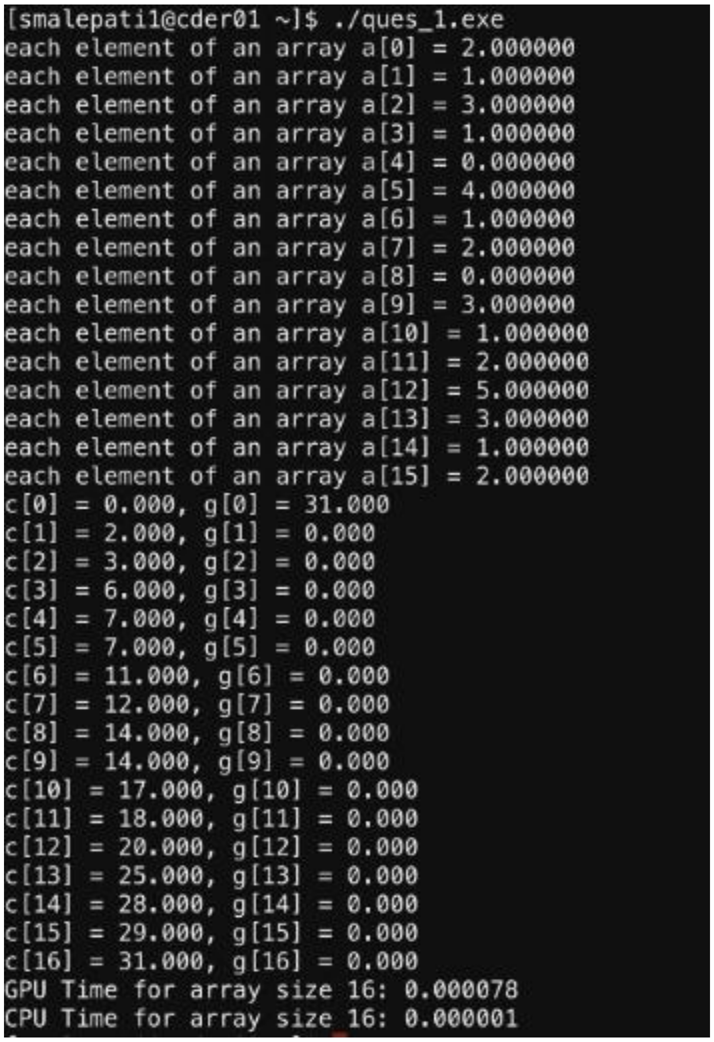
```
for (int d = 1; d < thread_number; d *= 2)
{
   offset >>= 1;
    __syncthreads();
        if (thid < d)
        {
            int ai = bid * thread_number + offset*(2*thid+1)-1;
            int bi = bid * thread_number + offset*(2*thid+2)-1;
            float t = temp[bid * thread_number + ai];
            temp[ai]  = temp[ bi];
            temp[bi] += t;
        }
}
__syncthreads();
```

Scanning the array for carryout the Prefix Sums.



```
void scanCPU(float *f_out, float *f_in, int i_n)
{
    f_out[0] = 0;
    for (int i =1; i <=i_n; i++)
    {
        f_out[i] = f_out[i-1] + f_in[i-1];
        }
}
```

Finally Obtaining the output.



```
[smalepati1@cder01 ~]$ ./ques_1.exe
each element of an array a[0] = 2.000000
each element of an array a[1] = 1.000000
each element of an array a[2] = 3.000000
each element of an array a[3] = 1.000000
each element of an array a[4] = 0.000000
each element of an array a[5] = 4.000000
each element of an array a[6] = 1.000000
each element of an array a[7] = 2.000000
each element of an array a[8] = 0.000000
each element of an array a[9] = 3.000000
each element of an array a[10] = 1.000000
each element of an array a[11] = 2.000000
each element of an array a[12] = 5.000000
each element of an array a[13] = 3.000000
each element of an array a[14] = 1.000000
each element of an array a[15] = 2.000000
c[0] = 0.000, g[0] = 31.000
c[1] = 2.000, g[1] = 0.000
c[2] = 3.000, g[2] = 0.000
c[3] = 6.000, g[3] = 0.000
c[4] = 7.000, g[4] = 0.000
c[5] = 7.000, g[5] = 0.000
c[6] = 11.000, g[6] = 0.000
c[7] = 12.000, g[7] = 0.000
c[8] = 14.000, g[8] = 0.000
c[9] = 14.000, g[9] = 0.000
c[10] = 17.000, g[10] = 0.000
c[11] = 18.000, g[11] = 0.000
c[12] = 20.000, g[12] = 0.000
c[13] = 25.000, g[13] = 0.000
c[14] = 28.000, g[14] = 0.000
c[15] = 29.000, g[15] = 0.000
c[16] = 31.000, g[16] = 0.000
GPU Time for array size 16: 0.000078
CPU Time for array size 16: 0.000001
```

Question 2:

In this scan, a single kernel can be written to perform all three step of the hierarchical scan algorithm. Thread block i first performs a scan on its scan block, using one of the three parallel algorithms of the above mentioned Question 1.

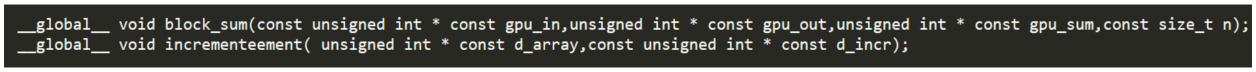
The block then waits for its left neighbor block i–1 to pass the sum value. Once the sum from block i–1 is received, the block generates and passes its sum value to its right neighbor block i+1.

The block then moves on to add the sum value received from block i–1 in order to complete all the output values of the scan block.

So the theme of this problem is to complete all the 3 steps performed by Kogge Stone Algorithm.

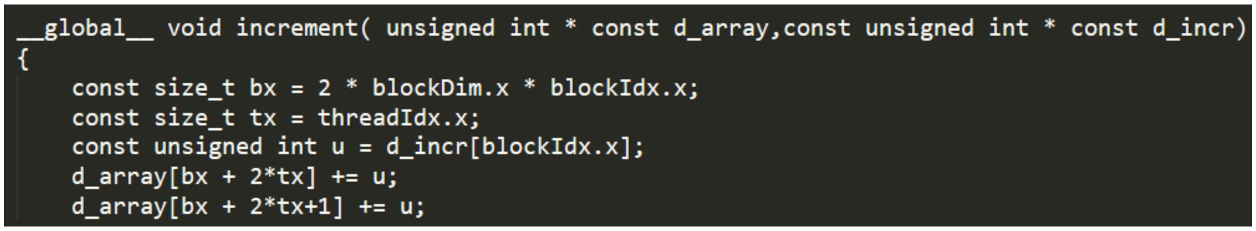
Code Snippet:

We have defined the following fucntions to execute the single pass scan.



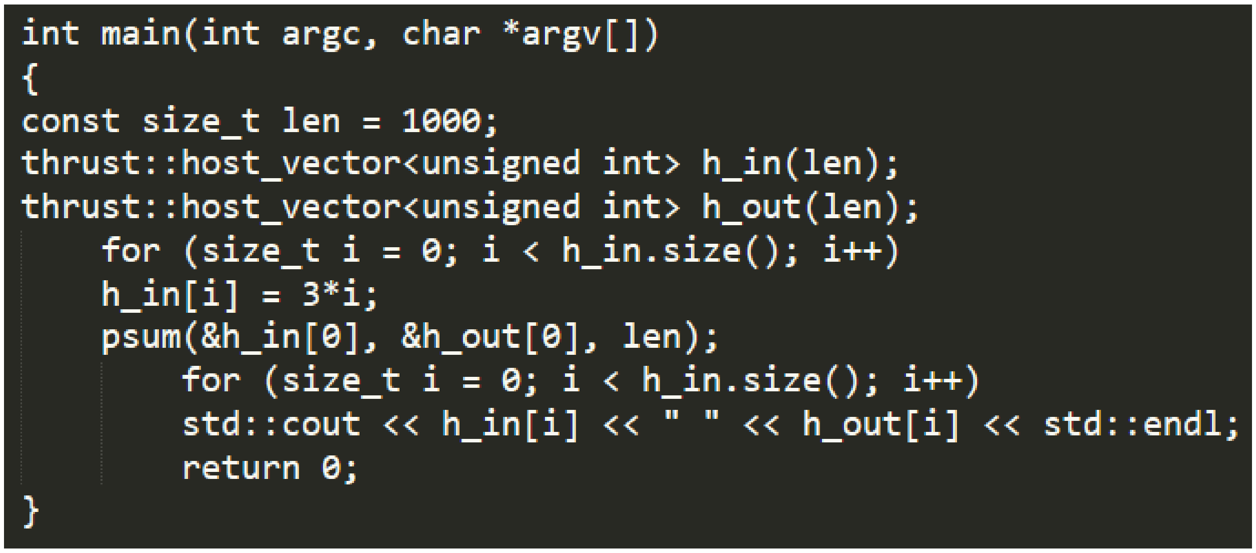
```
__global__ void block_sum(const unsigned int * const gpu_in,unsigned int * const gpu_out,unsigned int * const gpu_sum,const size_t n);
__global__ void incrementeement( unsigned int * const d_array,const unsigned int * const d_incr);
```

Performing the single scan pass



```
__global__ void increment( unsigned int * const d_array,const unsigned int * const d_incr)
{
    const size_t bx = 2 * blockDim.x * blockIdx.x;
    const size_t tx = threadIdx.x;
    const unsigned int u = d_incr[blockIdx.x];
    d_array[bx + 2*tx] += u;
    d_array[bx + 2*tx+1] += u;
```

Main function to implement the Single pass scan



```
int main(int argc, char *argv[])
{
const size_t len = 1000;
thrust::host_vector<unsigned int> h_in(len);
thrust::host_vector<unsigned int> h_out(len);
    for (size_t i = 0; i < h_in.size(); i++)
    h_in[i] = 3*i;
    psum(&h_in[0], &h_out[0], len);
        for (size_t i = 0; i < h_in.size(); i++)
        std::cout << h_in[i] << " " << h_out[i] << std::endl;
        return 0;
}
```