PRATAP SHINGANE
2020BTEIT00050
COURSE: PC

Finding the maximum number from given 'n' numbers CUDA code

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

#define BLOCK_SIZE 256 // block size is a constant, set to 256

// kernel function to find the maximum value from an array of floats

__global__ void find_max_kernel(float *input, float *output, int n) {

    // allocate shared memory for each block
    extern __shared__ float sdata[];
    // calculate the thread ID within the block and the index into the input
array
    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x * blockDim.x + threadIdx.x;

    // copy input data into shared memory, padding with -INFINITY if
necessary
    if (i < n) {
        sdata[tid] = input[i];
    } else {
        sdata[tid] = -INFINITY;
    }

    // synchronize threads in the block before performing reduction
```

```
    __syncthreads();

    // perform parallel reduction within the block
    for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1) {
        if (tid < s) {
            sdata[tid] = fmaxf(sdata[tid], sdata[tid + s]);
        }
        __syncthreads();
    }

    //  the first thread in each block writes the result to output array
    if (tid == 0) {
        output[blockIdx.x] = sdata[0];
    }
}

// host function to call the kernel and perform final reduction on host
float find_max(float *input, int n) {
    float *d_input, *d_output, *h_output;
    int num_blocks = (n + BLOCK_SIZE - 1) / BLOCK_SIZE;  //calculate
number of blocks needed

    // allocate device memory
    cudaMalloc(&d_input, n * sizeof(float));
    cudaMalloc(&d_output, num_blocks * sizeof(float));

    // copy input data to device memory
    cudaMemcpy(d_input, input, n * sizeof(float),
cudaMemcpyHostToDevice);



    // To get the elapsed time
    cudaEvent_t start, end;
    cudaEventCreate(&start);
    cudaEventCreate(&end);
```

```
    cudaEventRecord(start);

    // run kernel
    find_max_kernel<<<num_blocks, BLOCK_SIZE, BLOCK_SIZE *
sizeof(float)>>>(d_input, d_output, n);



    cudaEventRecord(end);
    cudaEventSynchronize(end);

    float elapsed_time_ms;
    cudaEventElapsedTime(&elapsed_time_ms, start, end);

    printf("Elapsed time: %f ms\n", elapsed_time_ms);

    cudaEventDestroy(start);
    cudaEventDestroy(end);



    // allocate host memory for output
    h_output = (float *)malloc(num_blocks * sizeof(float));

    // copy output data from device to host
    cudaMemcpy(h_output, d_output, num_blocks * sizeof(float),
cudaMemcpyDeviceToHost);

    // perform final reduction on host
    float max_val = -INFINITY;
    for (int i = 0; i < num_blocks; i++) {
        max_val = fmaxf(max_val, h_output[i]);
    }

    // free memory
    free(h_output);
```

```
    cudaFree(d_input);
    cudaFree(d_output);

    return max_val;
}

int main() {
    int n = 10;
    float *input = (float *)malloc(n * sizeof(float));

    // initialize input data
    for (int i = 0; i < n; i++) {
        if(i%2==0)
          input[i]=i*i;
        else
          input[i] = i*3;

        printf("%f", input[i]);
    }printf("\n");

    float max_val = find_max(input, n);

    printf("Maximum value: %f\n", max_val);

    free(input);

    return 0;
}
```

Output:

```
0.0000003.0000004.0000009.00000016.00000015.00000036.00000021.00000064.00000027.000000
Elapsed time: 0.036480 ms
Maximum value: 64.000000
```