Shrikrushna Zirape :

LP5 : Assignment3

41283 (BE2)

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

// Kernel function for Min, Max, Sum, and Average operations
__global__ void reduction(float* input, int n, float* output_min, float* output_max,
float* output_sum, float* output_avg) {
    __shared__ float shared_min;
    __shared__ float shared_max;
    __shared__ float shared_sum;

    int tid = threadIdx.x;
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    // Initialize shared variables
    if (tid == 0) {
        shared_min = input[0];
        shared_max = input[0];
        shared_sum = 0;
    }
    __syncthreads();

    // Reduction loop
    while (i < n) {
        if (input[i] < shared_min) {
            shared_min = input[i];
        }
        if (input[i] > shared_max) {
            shared_max = input[i];
        }
        shared_sum += input[i];
        i += blockDim.x * gridDim.x;
    }
    // Reduce within block
    for (int s = blockDim.x / 2; s > 0; s >>= 1) {
        if (tid < s) {
            if (shared_min > __shfl_down_sync(0xffffffff, shared_min, s)) {
                shared_min = __shfl_down_sync(0xffffffff, shared_min, s);
            }
            if (shared_max < __shfl_down_sync(0xffffffff, shared_max, s)) {
                shared_max = __shfl_down_sync(0xffffffff, shared_max, s);
            }
            shared_sum += __shfl_down_sync(0xffffffff, shared_sum, s);
        }
        __syncthreads();
    }

    // Write output variables
    if (tid == 0) {
        atomicMin(output_min, shared_min);
        atomicMax(output_max, shared_max);
        atomicAdd(output_sum, shared_sum);
        *output_avg = *output_sum / n;
    }
}

int main() {
    // Input array and its size
    float input[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0};
    int n = sizeof(input) / sizeof(float);

    // Allocate memory on the device for the input array and the output variables
    float* d_input;
    cudaMalloc(&d_input, n * sizeof(float));
    cudaMemcpy(d_input, input, n * sizeof(float), cudaMemcpyHostToDevice);

    float* d_output_min;
    cudaMalloc(&d_output_min, sizeof(float));
    cudaMemcpy(d_output_min, &input[0], sizeof(float), cudaMemcpyHostToDevice);
    float* d_output_max;
    cudaMalloc(&d_output_max, sizeof(float));
    cudaMemcpy(d_output_max, &input[0], sizeof(float), cudaMemcpyHostToDevice);

    float* d_output_sum;
    cudaMalloc(&d_output_sum, sizeof(float));
    cudaMemcpy(d_output_sum, &input[0], sizeof(float), cudaMemcpyHostToDevice);

    float* d_output_avg;
    cudaMalloc(&d_output_avg, sizeof(float));

    // Define block size and grid size
    int block_size = 256;
    int grid_size = (n + block_size - 1) / block_size;

    // Launch kernel function
// Copy output variables from device to host
// Pass output variables as arguments to the kernel function
reduction<<<grid_size, block_size>>>(d_input, n, d_output_min, d_output_max,
d_output_sum, d_output_avg);

float output_min;
cudaMemcpy(&output_min, d_output_min, sizeof(float), cudaMemcpyDeviceToHost);

float output_max;
cudaMemcpy(&output_max, d_output_max, sizeof(float),
cudaMemcpyDeviceToHost);

float output_sum;
cudaMemcpy(&output_sum, d_output_sum, sizeof(float),
cudaMemcpyDeviceToHost);

float output_avg;
cudaMemcpy(&output_avg, d_output_avg, sizeof(float), cudaMemcpyDeviceToHost);

// Print output variables
```

```
printf("Min = %f\n", output_min);
printf("Max = %f\n", output_max);
printf("Sum = %f\n", output_sum);
printf("Average = %f\n", output_avg);

// Free memory on the device
cudaFree(d_input);
cudaFree(d_output_min);
cudaFree(d_output_max);
cudaFree(d_output_sum);
cudaFree(d_output_avg);

return 0;
}


/*
nvcc -o reduction reduction.cu


./reduction



Input array: {1, 5, 2, 8, 4, 6, 3, 7}

Block size: 4

Grid size: 2
output :
Min = 1.000000
Max = 8.000000
Sum = 36.000000
Average = 4.500000


*/
```