EE6042: Network & Host-Security

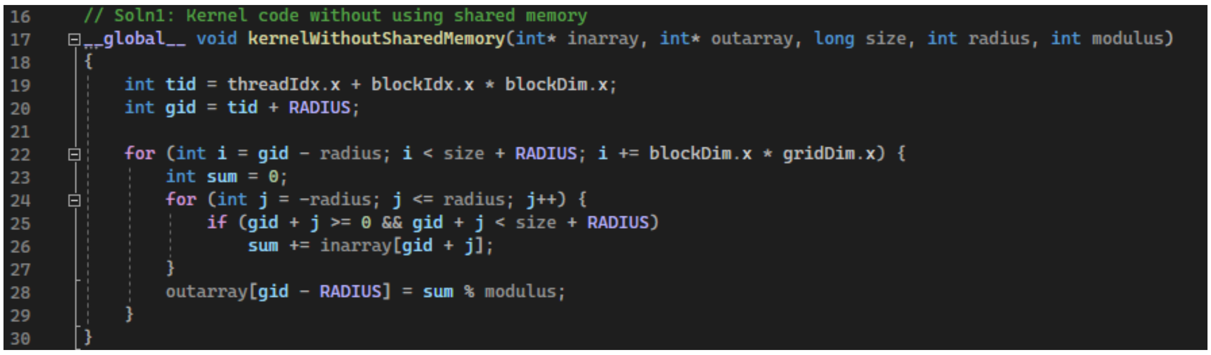
Spring semester 2023
Assignment report

Assignment	1
Assignment title	1-D Sum Case Study
Student name	Fionn Murray
Student ID number	18223451
Report submission date	14/04/2023

Complete Code

```
#include <cuda.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#define RADIUS 128

#define N 10240000L
#define M 65537L

#define BLOCK_SIZE 128

// Soln1: Kernel code without using shared memory
__global__ void kernelWithoutSharedMemory(int* inarray, int* outarray, long 
size, int radius, int modulus)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    int gid = tid + RADIUS;

    for (int i = gid - radius; i < size + RADIUS; i += blockDim.x * 
gridDim.x) {
        int sum = 0;
        for (int j = -radius; j <= radius; j++) {
            if (gid + j >= 0 && gid + j < size + RADIUS)
                sum += inarray[gid + j];
        }
        outarray[gid - RADIUS] = sum % modulus;
    }
}

void fillRandomData(int* a, long size)
{
    int i;
    for (i = 0; i < size; i++) {
        a[i] = rand() % M;
    }
}

void VerifyGPUOperation(int* inarray, int* outarray, long size)
{
    // Verification code
    for (int i = 0; i < size; i++)
    {
        int sum = 0;
        for (int j = i - RADIUS; j <= i + RADIUS; j++)
        {
            if (j >= 0 && j < size)
                sum += inarray[j];
        }
        int expected = sum % M;
        if (outarray[i] != expected)
        {
            printf("Verification failed at index %d: expected %d, got %d\n", 
i, expected, outarray[i]);
            return;
        }
    }
    printf("Verification successful!\n");
}

int main(void)
{
    // Input parameters
    int outputSize[] = { 10240, 1024000, 102400000 };
    int radius[] = { 3, 17, 128};
    int blockSize[] = { 128, 512, 1024};
    int numTests = sizeof(outputSize) / sizeof(int);

    for (int i = 0; i < numTests; i++)
    {
        int size = outputSize[i];
        int rad = radius[i];
        int block = blockSize[i];

        printf("Test %d:\n", i + 1);
        printf("Output Array Size: %d\n", size);
        printf("Radius: %d\n", rad);
        printf("Block Size: %d\n", block);

        // Allocate memory for input and output arrays
        int* h_inarray = (int*)malloc(size * sizeof(int));
        int* h_outarray1 = (int*)malloc(size * sizeof(int));
        fillRandomData(h_inarray, size);

        int* d_inarray;
        int* d_outarray1;
        cudaMalloc((void**)&d_inarray, size * sizeof(int));
        cudaMalloc((void**)&d_outarray1, size * sizeof(int));
        cudaMemcpy(d_inarray, h_inarray, size * sizeof(int),
cudaMemcpyHostToDevice);

        // Create CUDA events for measuring execution time
        cudaEvent_t start, stop;
        cudaEventCreate(&start);
        cudaEventCreate(&stop);

        // Launch kernel without using shared memory
        cudaEventRecord(start, 0);
        kernelWithoutSharedMemory << <(size + block - 1) / block, block >> >
(d_inarray, d_outarray1, size, rad, M);
        cudaEventRecord(stop, 0);
        cudaEventSynchronize(stop);
        float elapsedTimeWithoutSharedMemory;
        cudaEventElapsedTime(&elapsedTimeWithoutSharedMemory, start, stop);
        printf("Kernel Execution Time (without shared memory): %f ms\n",
elapsedTimeWithoutSharedMemory);

        // Copy result back to host
        cudaMemcpy(h_outarray1, d_outarray1, size * sizeof(int),
cudaMemcpyDeviceToHost);

        // Verify the output
        VerifyGPUOperation(h_inarray, h_outarray1, size);

        // Free memory
        free(h_inarray);
        free(h_outarray1);
        cudaFree(d_inarray);
        cudaFree(d_outarray1);

        printf("\n");
    }

    return 0;
}
```

Explanation Of Kernel Function



```
    // Soln1: Kernel code without using shared memory
__global__ void kernelWithoutSharedMemory(int* inarray, int* outarray, long size, int radius, int modulus)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    int gid = tid + RADIUS;

    for (int i = gid - radius; i < size + RADIUS; i += blockDim.x * gridDim.x) {
        int sum = 0;
        for (int j = -radius; j <= radius; j++) {
            if (gid + j >= 0 && gid + j < size + RADIUS)
                sum += inarray[gid + j];
        }
        outarray[gid - RADIUS] = sum % modulus;
    }
}
```

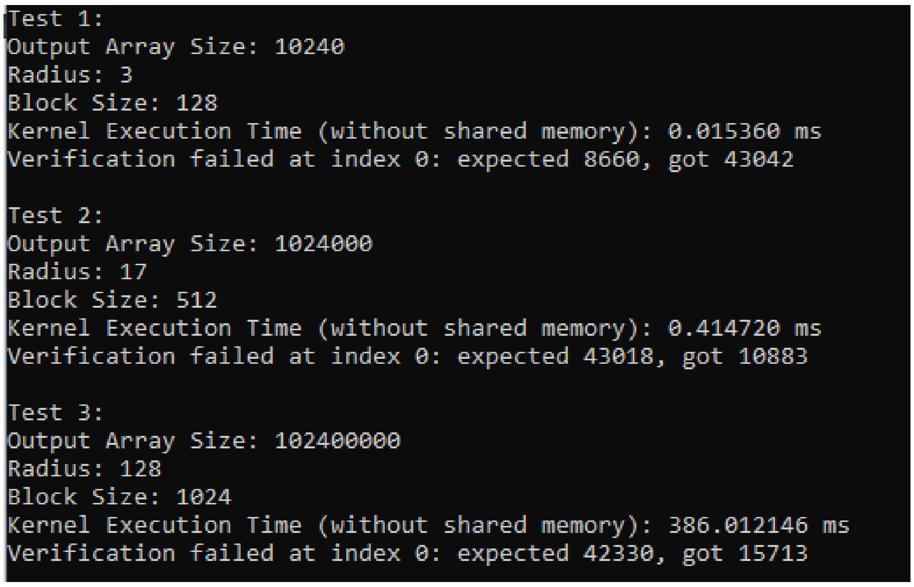
Figure 1, Kernel Function

The kernel function uses the inputs ‘inarray’ and ‘outarray’ as pointers to input and output arrays, ‘size’ as the size of the input array, ‘radius’ as the neighborhood radius, and modulus as the value for modulo operation. The function uses global and local thread indices to compute the neighborhood sum for each element in the input array.

To ensure the thread processes the correct element in the input and output arrays, the thread id ‘tid’ is calculated by adding the thread index within the block, to the block index multiplied by the block size. The radius value is then added to the thread index value to calculate the global index values that are shifted by the radius values number of positions.

The kernel function then computes the modulo value by iterating through chunks of elements in the input and output arrays equal to the number of threads in the grid. The computed modulo sum is then written to the output array once the calculation are completed.

I spent some time troubleshooting to correct the function, as it fails its verification on expected outcome. However, I had to move on before finding the solution in order to test execution times with various parameters. I was able to continue as the function executed correctly but the modulo sum computed incorrectly. I completed 9 tests at a time to find the various changes in execution times.

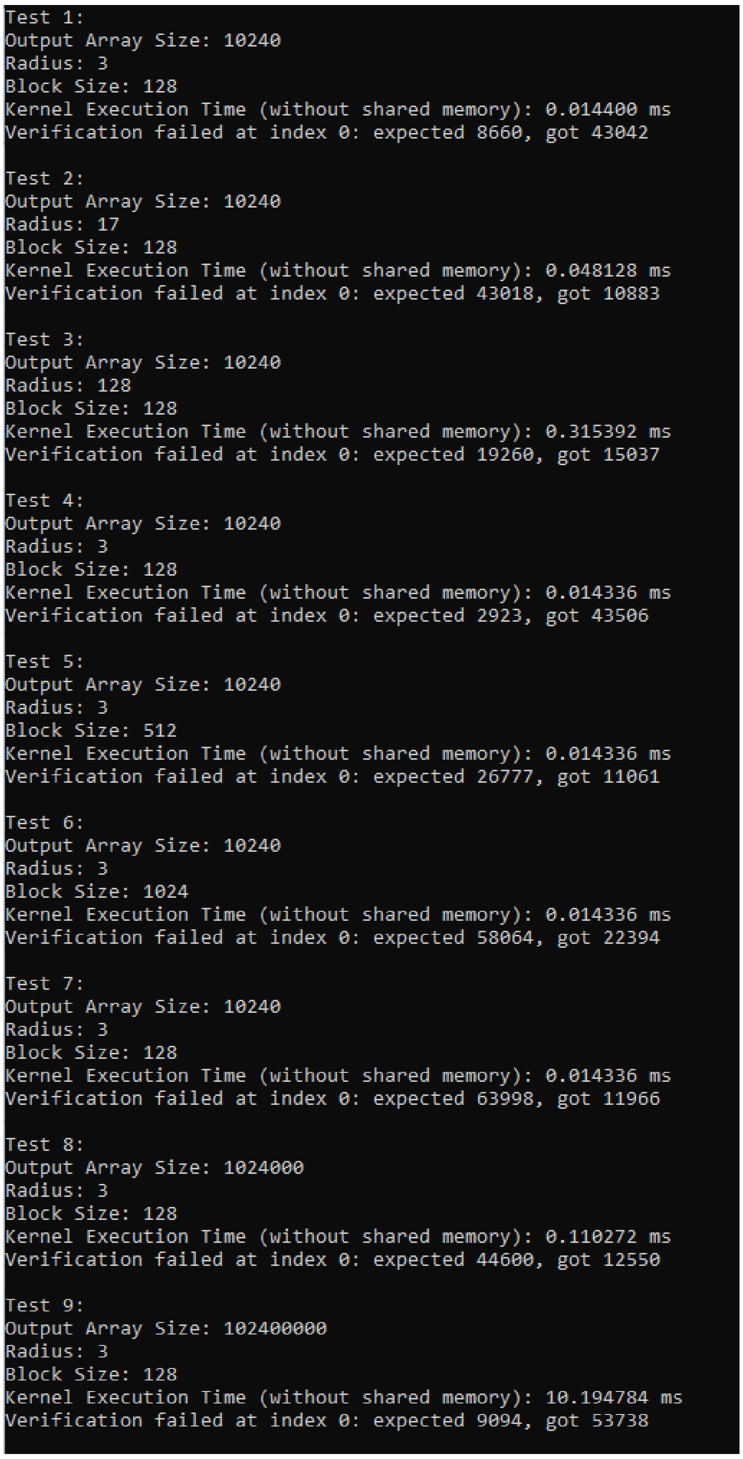


```
Test 1:
Output Array Size: 10240
Radius: 3
Block Size: 128
Kernel Execution Time (without shared memory): 0.015360 ms
Verification failed at index 0: expected 8660, got 43042

Test 2:
Output Array Size: 1024000
Radius: 17
Block Size: 512
Kernel Execution Time (without shared memory): 0.414720 ms
Verification failed at index 0: expected 43018, got 10883

Test 3:
Output Array Size: 102400000
Radius: 128
Block Size: 1024
Kernel Execution Time (without shared memory): 386.012146 ms
Verification failed at index 0: expected 42330, got 15713
```

Figure 2, Segment of Code output debug console

Plot Of Test Case Execution Times

Upon initially testing the execution times of increasing each parameter individually, I noticed that the value of radius and array output size are the only two to impact the kernel execution time. The chosen array along with the chosen radius size have an apparent linear relationship with execution time of the kernel.



```
Test 1:
Output Array Size: 10240
Radius: 3
Block Size: 128
Kernel Execution Time (without shared memory): 0.014400 ms
Verification failed at index 0: expected 8660, got 43042

Test 2:
Output Array Size: 10240
Radius: 17
Block Size: 128
Kernel Execution Time (without shared memory): 0.048128 ms
Verification failed at index 0: expected 43018, got 10883

Test 3:
Output Array Size: 10240
Radius: 128
Block Size: 128
Kernel Execution Time (without shared memory): 0.315392 ms
Verification failed at index 0: expected 19260, got 15037

Test 4:
Output Array Size: 10240
Radius: 3
Block Size: 128
Kernel Execution Time (without shared memory): 0.014336 ms
Verification failed at index 0: expected 2923, got 43506

Test 5:
Output Array Size: 10240
Radius: 3
Block Size: 512
Kernel Execution Time (without shared memory): 0.014336 ms
Verification failed at index 0: expected 26777, got 11061

Test 6:
Output Array Size: 10240
Radius: 3
Block Size: 1024
Kernel Execution Time (without shared memory): 0.014336 ms
Verification failed at index 0: expected 58064, got 22394

Test 7:
Output Array Size: 10240
Radius: 3
Block Size: 128
Kernel Execution Time (without shared memory): 0.014336 ms
Verification failed at index 0: expected 63998, got 11966

Test 8:
Output Array Size: 1024000
Radius: 3
Block Size: 128
Kernel Execution Time (without shared memory): 0.110272 ms
Verification failed at index 0: expected 44600, got 12550

Test 9:
Output Array Size: 102400000
Radius: 3
Block Size: 128
Kernel Execution Time (without shared memory): 10.194784 ms
Verification failed at index 0: expected 9094, got 53738
```

Figure 3, Initial Test

This test first tests the impact of increasing the radius, then the block size, and finally the output array. The increase in radius size has a small impact on the kernel execution time, while the large change in output array size has a much more significant impact on the execution time. The plots below reflect this.

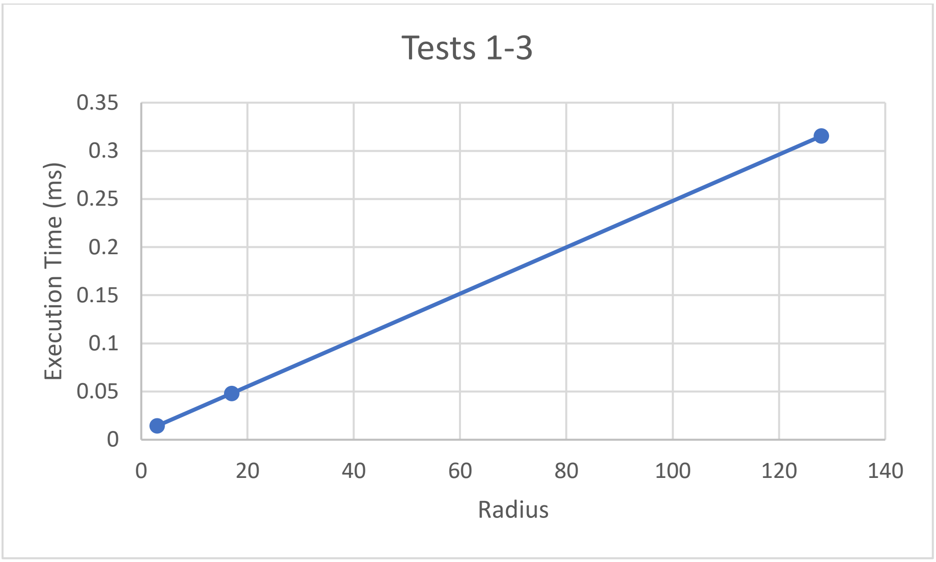


Figure 4, Varying Radius Length

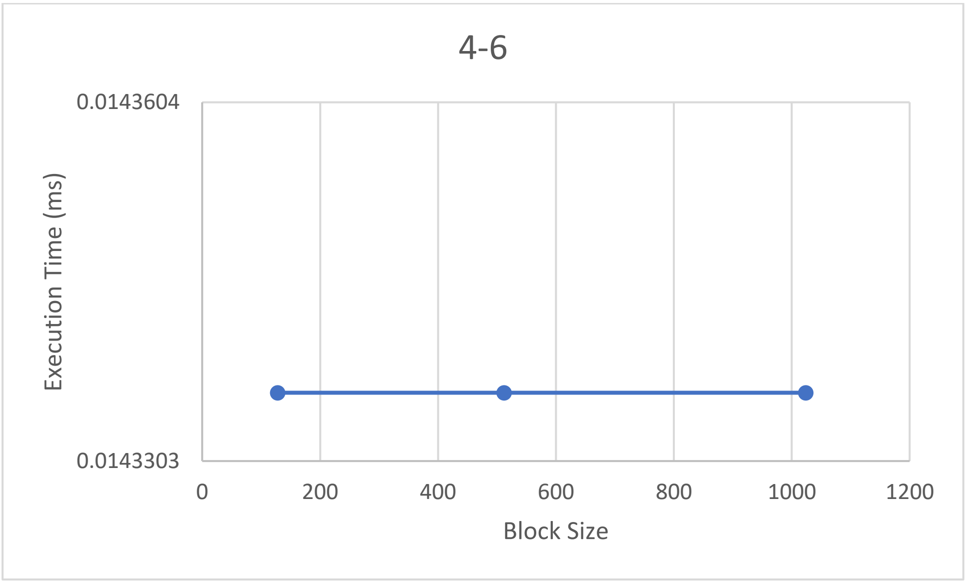


Figure 5, Varying Block Size

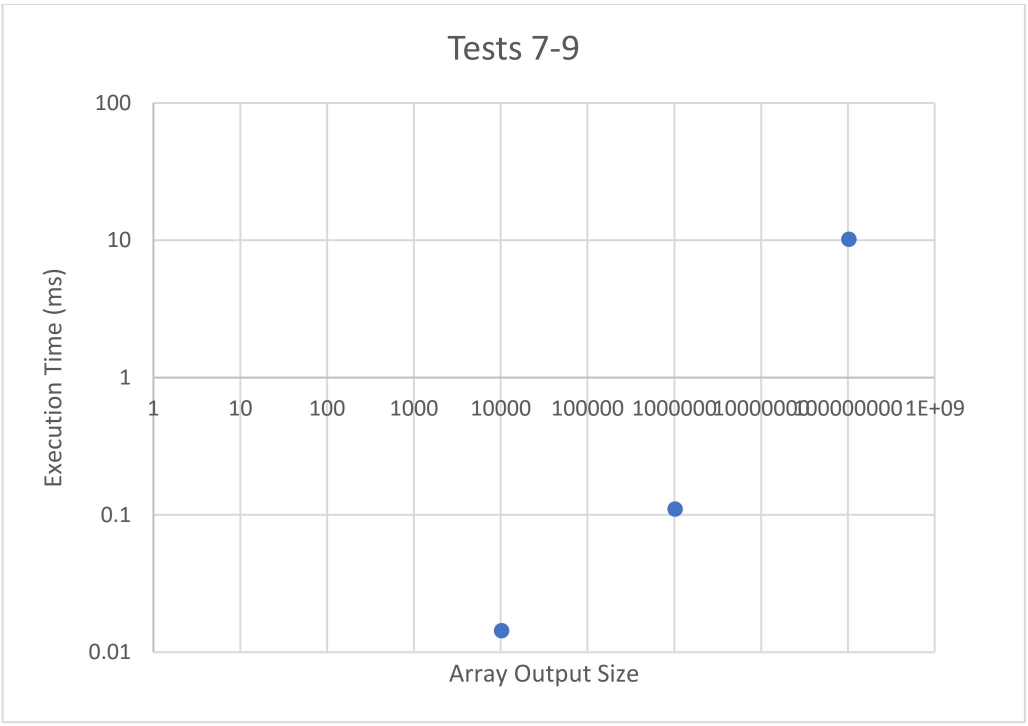


Figure 6, Varying Array Output Size