```c
/*
Program: addVector

This is a modification of the addVectorCUDA.cu from the class folder.
The modification was done to complete number 3 on Homework #4.
Changes made to the original program inclues function calculations, location of code statments,
and adding/removing comments. This was done so to complete the assignment as well as to understand
the logic behing parallel coding using CUDA GPU.

Author: Inanc Senocak
Editor: Dustin (Ting-Hsuan) Ma

Compile: nvcc -O2 addVectorCUDA.cu -o run.exe
Execute: ./run.exe

*/

#include "timer.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>

#define NX 1000000000
#define RADIUS 5
#define BLOCK_SIZE 256
#define SIZE BLOCK_SIZE + 2 * RADIUS

typedef float REAL;

__global__ void GPU_stencil(REAL *in, REAL *out)
{
	__shared__ REAL
		   tmp[SIZE]; // This is the correct way to dynamically allocate memory for each thr
ead

	// defining the index used by global and local array
	int gindex = blockIdx.x * blockDim.x + threadIdx.x;
	int lindex = threadIdx.x + RADIUS;

	// setting array elemtns into tmp array
	tmp[lindex] = in[gindex];

	__syncthreads();

	// Applying the stencil
	REAL sum = 0.0f;
	for (int j = -RADIUS; j <= RADIUS; j++) {
		sum += tmp[lindex + j];
	}

	// Store the result
	out[gindex] = sum;
}

void CPU_stencil(REAL *in, REAL *out)
{
	// CPU stencil done in class
	for (int i = RADIUS; i < NX; i++) {
		REAL sum = 0.0f;
		for (int j = -RADIUS; j <= RADIUS; j++) {
			sum += in[i + j];
		}
		out[i] = sum;
	}
}

int main(void)
{
	// Allocating memory for CPU
	REAL *a = (REAL *) malloc(NX * sizeof(*a));
	REAL *b = (REAL *) malloc(NX * sizeof(*b));

	// Allocating memory for GPU
	REAL *d_a, *d_b;
	cudaMallocManaged(&d_a, NX * sizeof(REAL));
	cudaMallocManaged(&d_b, NX * sizeof(REAL));

	REAL *c = (REAL *) malloc(NX * sizeof(*c)); // created to store values from Device to Host

	// Let’s fill the arrays with some numbers
	for (int i = 0; i < NX; i++) {
		a[i] = 0.0f;
		b[i] = 2.0f;
		c[i] = 0.0f;
	}
```



```c
	// ********************CPU************************
	double start, finish; // time for CPU
	REAL   elapsedTime;   // in float because it is recorded in ms

	GET_TIME(start);

	CPU_stencil(b, a); // calling CPU function

	GET_TIME(finish);

	// Outputting answer for CPU calculation
	printf("|===========================CPU============================|\n");
	printf("a[%d] = %4f, elapsed wall time (host) = %.6f seconds \n", RADIUS, a[RADIUS],
	       finish - start);
	printf("\n");

	// ********************GPU************************
	int nBlocks = (NX + BLOCK_SIZE - 1) / BLOCK_SIZE; // allows n to round up

	// Copying array memory from host to device
	cudaMemcpy(d_b, b, NX * sizeof(REAL), cudaMemcpyHostToDevice);

	cudaEvent_t timeStart, timeStop; // cudaEvent_t initializes variable used in event time
	cudaEventCreate(&timeStart);
	cudaEventCreate(&timeStop);
	cudaEventRecord(timeStart, 0);

	GPU_stencil<<<nBlocks, BLOCK_SIZE>>>(d_b, d_a); // replaced <<<1,1>>> with current

	cudaEventRecord(timeStop, 0);
	cudaEventSynchronize(timeStop);
	cudaEventElapsedTime(&elapsedTime, timeStart, timeStop);

	// Copying result array from device back to memory
	cudaMemcpy(c, d_a, NX * sizeof(REAL), cudaMemcpyDeviceToHost);

	// Outputting answer for GPU calculation
	printf("|===========================GPU============================|\n");
	printf("c[%d] = %4f, elapsed wall time (device) = %3.1f ms\n", RADIUS, c[RADIUS], elapsedTime);

	// Removing event created for timing the calculation
	cudaEventDestroy(timeStart);
	cudaEventDestroy(timeStop);

	// Deallocating memory used for host and device
	free(a);
	free(b);
	free(c);
	cudaFree(d_a);
	cudaFree(d_b);

	return EXIT_SUCCESS;
}
```