```
extern "C++" {
#include "stdlib.h"
#include "stdio.h"
}

#include <cuda.h>
#include <cuda_runtime.h>

void print_array(const char *name, float *array, int array_length) {
    printf("%s: [", name);
    for (int i = 0; i < array_length; i++) printf(" %5.2f", array[i]);
    printf(" ]\n");
}

__global__ void gpu__vector_add(int offset, float *v1, float *v2, float *sum) {
    int position = (blockDim.y * threadIdx.x) + threadIdx.y;

    sum[offset + position] = v1[offset + position] + v2[offset + position];
}

int main(int n_arguments, char **arguments) {
    cudaSetDevice( 0 );

    size_t free_memory;
    size_t total_memory;
    cudaMemGetInfo(&free_memory, &total_memory);
    printf("free memory: %u, total memory: %u (before initialize)\n", (unsigned int)free_memory,
(unsigned int)total_memory);


    int array_length = atoi(arguments[1]);

    int max_threads = 512;

    int number_threads_x = 512 / 4, number_threads_y = 4;
    dim3 blockDimensions(number_threads_x, number_threads_y, 1);

    float *cpu__v1;
    float *cpu__v2;
    float *cpu__sum;

    /**
     *  Allocate the arrays locally
     */
    cpu__v1 = (float*)malloc(array_length * sizeof(float));
    cpu__v2 = (float*)malloc(array_length * sizeof(float));
    cpu__sum = (float*)malloc(array_length * sizeof(float));
//    memset(cpu__v1,  0, array_length * sizeof(float));
//    memset(cpu__v2,  0, array_length * sizeof(float));
//    memset(cpu__sum, 0, array_length * sizeof(float));

    float *gpu__v1;
    float *gpu__v2;
    float *gpu__sum;

    /**
     *  Allocate the memory on the GPU
     */
    cudaMalloc((void**) &gpu__v1,  array_length * sizeof(float));
    cudaMalloc((void**) &gpu__v2,  array_length * sizeof(float));
    cudaMalloc((void**) &gpu__sum, array_length * sizeof(float));

//    cudaMemset(gpu__v1,  0, array_length * sizeof(float));
//    cudaMemset(gpu__v2,  0, array_length * sizeof(float));
//    cudaMemset(gpu__sum, 0, array_length * sizeof(float));

    cudaMemGetInfo(&free_memory, &total_memory);
    printf("free memory: %u, total memory: %u (after mallocs)\n", (unsigned int)free_memory,
(unsigned int)total_memory);

    /**
     *  Assign the CPU arrays:
     */
    for (int i = 0; i < array_length; i++) {
        cpu__v1[i] = i;
        cpu__v2[i] = 2 * i;
    }

    print_array("v1", cpu__v1, array_length);
    print_array("v2", cpu__v2, array_length);

    /**
     *  Copy the arrays from the CPU to the GPU (the gpu array goes first)
     */
    cudaMemcpy(gpu__v1, cpu__v1, array_length * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(gpu__v2, cpu__v2, array_length * sizeof(float), cudaMemcpyHostToDevice);

    /**
     *  Run the GPU kernels
     */
    int remaining = array_length;
    while (remaining > 0) {
        if (remaining > max_threads) {
            gpu__vector_add<<<1, max_threads>>>(remaining - max_threads, gpu__v1, gpu__v2, gpu__sum);
        } else {
            gpu__vector_add<<<1, remaining>>>(remaining - max_threads, gpu__v1, gpu__v2, gpu__sum);
        }
        remaining -= max_threads;
    }

    /**
     *  Move the sum from the array on the GPU to the array on the CPU.
     */
    cudaMemcpy(cpu__sum, gpu__sum, array_length * sizeof(float), cudaMemcpyDeviceToHost);

    /**
     *  Print out the sum.
     */
    print_array("sum", cpu__sum, array_length);

    cudaFree(gpu__v1);
    cudaFree(gpu__v2);
    cudaFree(gpu__sum);

    cudaMemGetInfo(&free_memory, &total_memory);
    printf("free memory: %u, total memory: %u (after free)\n", (unsigned int)free_memory, (unsigned
int)total_memory);

    free(cpu__v1);
    free(cpu__v2);
    free(cpu__sum);
}
```