```
// headers
#include <stdio.h>

#include <cuda.h> // for CUDA

#include "helper_timer.h"

// global variables
// odd number 11444777 is deliberate illustration ( Nvidia OpenCL Samples )
int iNumberOfArrayElements=5;

float *hostInput1=NULL;
float *hostInput2=NULL;
float *hostOutput=NULL;
float *gold=NULL;

float *deviceInput1=NULL;
float *deviceInput2=NULL;
float *deviceOutput=NULL;

float timeOnCPU;
float timeOnGPU;

// *** CUDA KERNEL DEFINITION ***
// global kernel function definition
__global__ void vecAdd(float *in1,float *in2,float *out,int len)
{
    // variable declarations
    int i=blockIdx.x * blockDim.x + threadIdx.x;
    // code
    if(i < len)
    {
        out[i]=in1[i]+in2[i];
    }
}

int main(int argc,char *argv[])
{
    // function declarations
    void fillFloatArrayWithRandomNumbers(float *, int);
    void vecAddHost(const float *, const float *, float *, int);
    void cleanup(void);

    // code
    // allocate host-memory
    hostInput1=(float *)malloc(sizeof(float) * iNumberOfArrayElements);
    if(hostInput1== NULL)
    {
        printf("CPU Memory Fatal Error = Can Not Allocate Enough Memory For Host Input Array 1.\nExitting ...\n");
        cleanup();
        exit(EXIT_FAILURE);
    }

    hostInput2=(float *)malloc(sizeof(float) * iNumberOfArrayElements);
    if(hostInput2== NULL)
    {
        printf("CPU Memory Fatal Error = Can Not Allocate Enough Memory For Host Input Array 2.\nExitting ...\n");
        cleanup();
        exit(EXIT_FAILURE);
    }

    hostOutput=(float *)malloc(sizeof(float) * iNumberOfArrayElements);
    if(hostOutput== NULL)
    {
        printf("CPU Memory Fatal Error = Can Not Allocate Enough Memory For Host Output Array.\nExitting ...\n");
        cleanup();
        exit(EXIT_FAILURE);
    }

    gold=(float *)malloc(sizeof(float) * iNumberOfArrayElements);
    if(gold== NULL)
    {
        printf("CPU Memory Fatal Error = Can Not Allocate Enough Memory For Gold Output Array.\nExitting ...\n");
        cleanup();
        exit(EXIT_FAILURE);
    }

    // fill above input host vectors with arbitary but hard-coded data
    fillFloatArrayWithRandomNumbers(hostInput1,iNumberOfArrayElements);
    fillFloatArrayWithRandomNumbers(hostInput2,iNumberOfArrayElements);

    // allocate device-memory
    cudaError_t err=cudaSuccess;
    err=cudaMalloc((void **)&deviceInput1,sizeof(float) * iNumberOfArrayElements);
    if(err!=cudaSuccess)
    {
        printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.\nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
        cleanup();
        exit(EXIT_FAILURE);
    }

    err=cudaMalloc((void **)&deviceInput2,sizeof(float) * iNumberOfArrayElements);
    if(err!=cudaSuccess)
    {
        printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.\nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
        cleanup();
        exit(EXIT_FAILURE);
    }

    err=cudaMalloc((void **)&deviceOutput,sizeof(float) * iNumberOfArrayElements);
    if(err!=cudaSuccess)
    {
        printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.\nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
        cleanup();
        exit(EXIT_FAILURE);
    }

    // copy host memory contents to device memory
    err=cudaMemcpy(deviceInput1,hostInput1,sizeof(float) * iNumberOfArrayElements,cudaMemcpyHostToDevice);
    if(err!=cudaSuccess)
    {
        printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.\nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
        cleanup();
        exit(EXIT_FAILURE);
    }

    err=cudaMemcpy(deviceInput2,hostInput2,sizeof(float) * iNumberOfArrayElements,cudaMemcpyHostToDevice);
    if(err!=cudaSuccess)
    {
        printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.\nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
        cleanup();
        exit(EXIT_FAILURE);
    }

    // cuda kernel configuration
    dim3 DimGrid=dim3(ceil(iNumberOfArrayElements/256.0),1,1);
    dim3 DimBlock=dim3(256,1,1);

    // start timer
    StopWatchInterface *timer = NULL;
    sdkCreateTimer(&timer);
    sdkStartTimer(&timer);

    vecAdd<<<DimGrid,DimBlock>>>(deviceInput1,deviceInput2,deviceOutput,iNumberOfArrayElements);

    // stop timer
    sdkStopTimer(&timer);
    timeOnGPU = sdkGetTimerValue(&timer);
    sdkDeleteTimer(&timer);

    // copy device memory to host memory
    err=cudaMemcpy(hostOutput,deviceOutput,sizeof(float) * iNumberOfArrayElements,cudaMemcpyDeviceToHost);
    if(err!=cudaSuccess)
    {
        printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.\nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
        cleanup();
        exit(EXIT_FAILURE);
    }

    // results
    vecAddHost(hostInput1, hostInput2, gold, iNumberOfArrayElements);

    // compare results for golden-host
    const float epsilon = 0.000001f;
    bool bAccuracy=true;
    int breakValue=0;
    int i;
    for(i=0;i<iNumberOfArrayElements;i++)
    {
        float val1 = gold[i];
        float val2 = hostOutput[i];
        if(fabs(val1-val2) > epsilon)
        {
            bAccuracy = false;
            breakValue=i;
            break;
        }
    }

    if(bAccuracy==false)
    {
        printf("Break Value = %d\n",breakValue);
    }

    char str[125];
    if(bAccuracy==true)
        sprintf(str,"%s","Comparison Of Output Arrays On CPU And GPU Are Accurate Within The Limit Of 0.000001");
    else
        sprintf(str,"%s","Not All Comparison Of Output Arrays On CPU And GPU Are Accurate Within The Limit Of 0.000001");

    printf("1st Array Is From 0th Element %.6f To %dth Element %.6f\n",hostInput1[0], iNumberOfArrayElements-1, hostInput1[iNumberOfArrayElements-1]);
    printf("2nd Array Is From 0th Element %.6f To %dth Element %.6f\n",hostInput2[0], iNumberOfArrayElements-1, hostInput2[iNumberOfArrayElements-1]);
    printf("Grid Dimension = (%d,1,1) And Block Dimension = (%d,1,1)\n",DimGrid.x,DimBlock.x);
    printf("Sum Of Each Element From Above 2 Arrays Creates 3rd Array As :\n");
    printf("3nd Array Is From 0th Element %.6f To %dth Element %.6f\n",hostOutput[0], iNumberOfArrayElements-1, hostOutput[iNumberOfArrayElements-1]);
    printf("The Time Taken To Do Above Addition On CPU = %.6f (ms)\n",timeOnCPU);
    printf("The Time Taken To Do Above Addition On GPU = %.6f (ms)\n",timeOnGPU);
    printf("%s\n",str);

    // total cleanup
    cleanup();
    return(0);
}

void cleanup(void)
{
    // code

    // free allocated device-memory
    if(deviceInput1)
    {
        cudaFree(deviceInput1);
        deviceInput1=NULL;
    }

    if(deviceInput2)
    {
        cudaFree(deviceInput2);
        deviceInput2=NULL;
    }

    if(deviceOutput)
    {
        cudaFree(deviceOutput);
        deviceOutput=NULL;
    }

    // free allocated host-memory
    if(hostInput1)
    {
        free(hostInput1);
        hostInput1=NULL;
    }

    if(hostInput2)
    {
        free(hostInput2);
        hostInput2=NULL;
    }

    if(hostOutput)
    {
        free(hostOutput);
        hostOutput=NULL;
    }

    if(gold)
    {
        free(gold);
        gold=NULL;
    }
}

void fillFloatArrayWithRandomNumbers(float *pFloatArray, int iSize)
{
    // code
    int i;
    const float fScale = 1.0f / (float)RAND_MAX;
    for (i = 0; i < iSize; ++i)
    {
        pFloatArray[i] = fScale * rand();
    }
}

// "Golden" Host processing vector addition function for comparison purposes
void vecAddHost(const float* pFloatData1, const float* pFloatData2, float* pFloatResult, int iNumElements)
{
    int i;

    StopWatchInterface *timer = NULL;
    sdkCreateTimer(&timer);
    sdkStartTimer(&timer);

    for (i = 0; i < iNumElements; i++)
    {
        pFloatResult[i] = pFloatData1[i] + pFloatData2[i];
    }

    sdkStopTimer(&timer);
    timeOnCPU = sdkGetTimerValue(&timer);
    sdkDeleteTimer(&timer);
}

```