```
// VDotProduct.cu
//
// driver and kernel call

#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>

#define THREADS_PER_BLOCK 32


__global__ void dot_product (int *a_d, int *b_d, int *c_d, int arraySize)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int t = threadIdx.x;
    __shared__ int temp[THREADS_PER_BLOCK];


    if (x < arraySize/2)
        {
         a_d[x] = x + 1;
        }
    else
       {
        a_d[x] = x -((x - arraySize/2)*2);

        }


        b_d[x] = (x % 10) + 1;



        temp[t] = a_d[x] * b_d[x];
        __syncthreads();

        if(0 == threadIdx.x)
        {

        int sum=0;
        for(int i=0;i<THREADS_PER_BLOCK;i++)
        {
                sum+=temp[i];

        }
        atomicAdd(c_d,sum);
        }




}

extern "C" void gpu_dot_product (int *a, int *b, int *c, int arraySize)
{
        int *a_d, *b_d, *c_d;

        cudaMalloc ((void**) &a_d, sizeof(int) * arraySize);
        cudaMalloc ((void**) &b_d, sizeof(int) * arraySize);
        cudaMalloc ((void**) &c_d, sizeof(int) * arraySize);
        cudaMemcpy (a_d, a, sizeof(int) * arraySize, cudaMemcpyHostToDevice);
        cudaMemcpy (b_d, b, sizeof(int) * arraySize, cudaMemcpyHostToDevice);
```

```
        dot_product <<< ceil((float) arraySize/THREADS_PER_BLOCK), THREADS_PER_BLOCK >>> (a_d, b_d,
c_d, arraySize);

        cudaError_t err = cudaGetLastError();
        if (err != cudaSuccess)
                printf ("CUDA error: %s\n", cudaGetErrorString(err));

        cudaMemcpy (c, c_d, sizeof(int) * arraySize, cudaMemcpyDeviceToHost);
        cudaFree (a_d);
        cudaFree (b_d);
        cudaFree (c_d);


}
```