```
#define GRID_SIZE 256
#define BLOCK_SIZE 128

__global__
void gpu_dotproduct_stage1(const double *gpu_x, const double *gpu_y, size_t size,
double *gpu_result_stage1){

    size_t thread_id_global = blockIdx.x*blockDim.x + threadIdx.x;
    __shared__ double shared_m[BLOCK_SIZE];

    // I think, this is the right way:
    double thread_dp = 0;
    for (unsigned int i = thread_id_global; i<size; i += blockDim.x * gridDim.x)
        thread_dp += gpu_x[i] * gpu_y[i];
    shared_m[threadIdx.x] = thread_dp;


    // now the reduction
    for(int stride = blockDim.x/2; stride>0; stride/=2){
        __syncthreads();
        if (threadIdx.x < stride){
            shared_m[threadIdx.x] += shared_m[threadIdx.x + stride];
        }
    }

    // thread 0 writes result
    if (threadIdx.x == 0){
        gpu_result_stage1[blockIdx.x] = shared_m[0];
    }
}

__global__
void gpu_dotproduct_stage2(double *gpu_result_stage1, double *gpu_result_stage2){

    // only one block has a job here
    if (blockIdx.x == 0){
        //size_t thread_id_global = blockIdx.x*blockDim.x + threadIdx.x;
        __shared__ double shared_m[BLOCK_SIZE];

        // this time, the lecture is correct
        double thread_sum = 0;
        for (unsigned int i = threadIdx.x; i<GRID_SIZE; i += blockDim.x)
            thread_sum += gpu_result_stage1[i];
        shared_m[threadIdx.x] = thread_sum;

        // now the reduction
        for(int stride = blockDim.x/2; stride>0; stride/=2){
            __syncthreads();
            if (threadIdx.x < stride){
                shared_m[threadIdx.x] += shared_m[threadIdx.x + stride];
            }
        }
        // thread 0 writes result
        if (threadIdx.x == 0){
            //printf("hi, im thread 0 and im now writing %1.5e\n", shared_m[0]);
            *gpu_result_stage2 = shared_m[0];
        }
    }
}


__global__
void gpu_dotproduct_atomicAdd(const double *gpu_x, const double *gpu_y, size_t size,
double *gpu_result){

    size_t thread_id_global = blockIdx.x*blockDim.x + threadIdx.x;
    if (thread_id_global == 0)
        *gpu_result = 0;

    __shared__ double shared_m[BLOCK_SIZE];

    // I think, this is the right way:
    double thread_dp = 0;
    for (unsigned int i = thread_id_global; i<size; i += blockDim.x * gridDim.x)
        thread_dp += gpu_x[i] * gpu_y[i];
    shared_m[threadIdx.x] = thread_dp;


    // now the reduction
    for(int stride = blockDim.x/2; stride>0; stride/=2){
        __syncthreads();
        if (threadIdx.x < stride){
            shared_m[threadIdx.x] += shared_m[threadIdx.x + stride];
        }
    }

    __syncthreads();
    // thread 0 writes result
    if (threadIdx.x == 0){
        atomicAdd(gpu_result, shared_m[0]);
        //gpu_result_stage1[blockIdx.x] = shared_m[0];
    }
}

```