Barker, Mary

Homework 2

GPU dot product of two vectors with fixed 1024 threads using 2 methods:
(1) Number of blocks determined by vector length
(2) 2 blocks and iterations determined by vector length

```
/*
        Mary Barker
        Homework 2

        Vector addition on GPU that allows for more than 1024 elements
        In particular, it can do 2 different parallel memory setups:
        (1) each element in the vector assigned a thread, and the number of
            blocks assigned accordingly,
        (2) 2 blocks, 1024 threads each, and the algorithm iterates until
            each element in the vector has been reached

        to compile: nvcc BarkerHW2.cu

        OUTPUTS:
        with more than 2 blocks:
        Time in milliseconds= 0.068000000000000
        Last Values are A[4999] = 9998.000000000000000  B[4999] = 4999.000000000000000  C[4999] =
             14997.000000000000000

        with only 2 blocks:
        Time in milliseconds= 0.073000000000000
        Last Values are A[4999] = 9998.000000000000000  B[4999] = 4999.000000000000000  C[4999] =
             14997.000000000000000


*/
#include <sys/time.h>
#include <stdio.h>

//Length of vectors to be added.
#define N 5000

bool two_blocks = true;
int num_iters;
dim3 dimBlock, dimGrid;
float *A_CPU, *B_CPU, *C_CPU; //CPU pointers
float *A_GPU, *B_GPU, *C_GPU; //GPU pointers

void SetupCudaDevices()
{
        dimBlock.x = 1024;
        dimBlock.y = 1;
        dimBlock.z = 1;
        if (two_blocks)
        {
                dimGrid.x = 2;
                dimGrid.y = 1;
                dimGrid.z = 1;
                num_iters = (N - 1) / (dimGrid.x * dimBlock.x) + 1;
        }
        else
        {
                num_iters = 1;
                dimGrid.x = (N - 1) / dimBlock.x + 1;
                dimGrid.y = 1;
                dimGrid.z = 1;
        }
}

void AllocateMemory()
{
        //Allocate Device (GPU) Memory, & allocates the value of the specific pointer/array
        cudaMalloc(&A_GPU,N*sizeof(float));
        cudaMalloc(&B_GPU,N*sizeof(float));
        cudaMalloc(&C_GPU,N*sizeof(float));
```

```
        //Allocate Host (CPU) Memory
        A_CPU = (float*)malloc(N*sizeof(float));
        B_CPU = (float*)malloc(N*sizeof(float));
        C_CPU = (float*)malloc(N*sizeof(float));
}

//Loads values into vectors that we will add.
void Innitialize()
{
        int i;

        for(i = 0; i < N; i++)
        {
                A_CPU[i] = (float)2*i;
                B_CPU[i] = (float)i;
        }
}

//Cleaning up memory after we are finished.
void CleanUp(float *A_CPU,float *B_CPU,float *C_CPU,float *A_GPU,float *B_GPU,float *C_GPU)  //
    free
{
        free(A_CPU); free(B_CPU); free(C_CPU);
        cudaFree(A_GPU); cudaFree(B_GPU); cudaFree(C_GPU);
}

//This is the kernel. It is the function that will run on the GPU.
//It adds vectors A and B then stores result in vector C
__global__ void Addition(float *A, float *B, float *C, int n, int num_iterations_over_blocks)
{
        int id;
        for(int i = 0; i < num_iterations_over_blocks; i++)
        {
                id = i * (blockDim.x * gridDim.x) + blockDim.x * blockIdx.x + threadIdx.x;
                if(id < n) C[id] = A[id] + B[id];
        }
}

int main()
{
        int i;
        timeval start, end;

        //setup parallel structure
        SetupCudaDevices();

        //Partitioning off the memory that you will be using.
        AllocateMemory();

        //Loading up values to be added.
        Innitialize();

        //Starting the timer
        gettimeofday(&start, NULL);

        //Copy Memory from CPU to GPU
        cudaMemcpyAsync(A_GPU, A_CPU, N*sizeof(float), cudaMemcpyHostToDevice);
        cudaMemcpyAsync(B_GPU, B_CPU, N*sizeof(float), cudaMemcpyHostToDevice);

        //Calling the Kernel (GPU) function.
        Addition<<<dimGrid, dimBlock>>>(A_GPU, B_GPU, C_GPU, N, num_iters);

        //Copy Memory from GPU to CPU
        cudaMemcpyAsync(C_CPU, C_GPU, N*sizeof(float), cudaMemcpyDeviceToHost);
```

```
    //Stopping the timer
    gettimeofday(&end, NULL);

    //Calculating the total time used in the addition and converting it to milliseconds.
    float time = (end.tv_sec * 1000000 + end.tv_usec) - (start.tv_sec * 1000000 + start.
        tv_usec);

    //Displaying the time
    printf("Time␣in␣milliseconds=␣%.15f\n", (time/1000.0));

    // Displaying vector info you will want to comment out the vector print line when your
    //vector becomes big. This is just to make sure everything is running correctly.
    for(i = 0; i < N; i++)
    {
        //printf("A[%d] = %.15f  B[%d] = %.15f  C[%d] = %.15f\n", i, A_CPU[i], i, B_CPU[i
            ], i, C_CPU[i]);
    }

    //Displaying the last value of the addition for a check when all vector display has been
        commented out.
    printf("Last␣Values␣are␣A[%d]␣=␣%.15f␣␣B[%d]␣=␣%.15f␣␣C[%d]␣=␣%.15f\n", N-1, A_CPU[N-1],
        N-1, B_CPU[N-1], N-1, C_CPU[N-1]);

    //You're done so cleanup your mess.
    CleanUp(A_CPU,B_CPU,C_CPU,A_GPU,B_GPU,C_GPU);

    return(0);
}
```