Cuda Assignment #2

By: Alex Kazantsev and Hanumant Mahida

Date: March 19th, 2017

Results

Timing data

Number of Elements	Serial Time (s)	CUDA Time (s)	Speed Improvement
10e5	0.001847	0.002176	0.849
10e6	0.017127	0.000471	36.36
10e7	0.142393	0.003285	43.35

Code

Allocate memory and copy data to device for 3 vectors

```
cudaMalloc((void**)&A_on_device, num_elements*sizeof(float));
cudaMemcpy(A_on_device, A_on_host, num_elements*sizeof(float), cudaMemcpyHostToDevice);

cudaMalloc((void**)&B_on_device, num_elements*sizeof(float));
cudaMemcpy(B_on_device, B_on_host, num_elements*sizeof(float), cudaMemcpyHostToDevice);

cudaMalloc((void**)&C_on_device, GRID_SIZE*sizeof(float));
cudaMemset(C_on_device, 0.0, GRID_SIZE*sizeof(float));
```

Performing vector dot product on GPU

```
vector_dot_product_kernel <<< dimGrid, dimBlock >>> (num_elements, A_on_device, B_on_device, C_on_device, n
cudaThreadSynchronize();
```

Vector Dot Product Kernel for handling mutex locks and unlocks

```
__shared__ float runningSums[BLOCK_SIZE];

int tx = threadIdx.x;
int threadID = blockDim.x * blockIdx.x + threadIdx.x;
int stride = blockDim.x * gridDim.x;

float local_thread_sum = 0.0;
unsigned int i = threadID;

while(i < num_elements){
    local_thread_sum += a[i] * b[i];
    i += stride;
}
```

```
runningSums[threadIdx.x] = local_thread_sum;
__syncthreads();

for(int stride = blockDim.x/2; stride > 0; stride /= 2){
    if(tx < stride)
        runningSums[tx] += runningSums[tx+stride];
    __syncthreads();
}

if(threadIdx.x == 0) {
    lock(mutex);
    result[0] += runningSums[0];
    unlock(mutex);
}
```