ECE 4822: Engineering Computation IV

Homework No. 4: My First GPU Program

P01:

```
float *a, *b, *out;
float *d_a, *d_b, *d_out;
// Allocate Memory in the CPU domain
a =   (float*)malloc(sizeof(float) * N );
b =   (float*)malloc(sizeof(float) * N );
out =   (float*)malloc(sizeof(float) * N );


// Allocate memory in the GPU domain
cudaMalloc((void**)&d_a, sizeof(float) *N);
cudaMalloc((void**)&d_b, sizeof(float) *N);
cudaMalloc((void**)&d_out, sizeof(float) *N);

// Populate array
for(int i = 0; i < N; i++){
  a[i] = 1.0f; b[i] = 2.0f;
}
```

```
// Transfer the data from CPU to GPU

cudaMemcpy(d_a, a, sizeof(float) * N, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, sizeof(float) * N, cudaMemcpyHostToDevice);

vector_add<<<1, 1>>>(d_out, d_a, d_b, N);
//  vector_add(out, a, b, N);
cudaMemcpy(out, d_out, sizeof(float) * N, cudaMemcpyDeviceToHost);

// Check first 10 results
for (int i = 0; i < 10; i++) {
  printf("out[%d] = %f\n", i, out[i]);
}

//Free GPU
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_out);
```

So we see with the code above we, simply have a for loop computing our addition.
And we are doing it 1 at a time. So even though we are sending it to the GPU we aren't using the full power of it.

After running the program using 2 nice -19 sbatch --partition=gpu run.sh where run.sh is

```
nedc_130_[1]: more run.sh

#!/bin/bash

./hello.exe

nedc_130_[1]:
```

We get a slurm.out file that contains our print statements

```
./hello.exe

nedc_130_[1]: more slurm-228413.out

3.000000

3.000000

3.000000
```

P02:

This program allows us to run our program in parallel using the gpus cores and blocks using multi threads. So now we can use some of the power of gpu parallelization

```
__global__ void vector_add(float* out, float* a, float* b, int n) {

 int index = threadIdx.x;

 int stride = blockDim.x;


 for(int i = index; i < n; i += stride){

  out[i] = a[i] + b[i];

  printf("%f\n", out[i]);

 }
```

Unfortunately I could not time the program so we can’t really see the differences

```
nedc_130_[1]:   nvprof nice -19 sbatch --partition=gpu run.sh
======== Warning: unable to locate cuda driver library, GPU profiling
skipped
Submitted batch job 228451
======== Warning: No profile data collected.
nedc_130_[1]:
```

But from the website we can see that there is a clear speed difference between the two programs

Version	**Execution Time**	**Speedup**
Single-threaded GPU	1425 ms	1x
1 block (256 threads)	22.78 ms	62x
Multiple blocks (full grid)	1.13 ms	1261x

Overall the homework demonstrated the process of running C/C++ code on a GPU using CUDA. In the first version, the program computed vector addition serially, with only one thread, which underutilized the GPU's capabilities. By parallelizing the kernel using multiple threads and blocks, we were able to distribute the computation across the GPU cores efficiently. While direct timing was not measured on the gpu, the tutorial results clearly show that parallelization provides a dramatic speedup. This exercise highlights the importance of thread and block organization in CUDA programming and illustrates how GPUs achieve high-performance computation through massive parallelism.