Homework 4: Questions

(20 points) What are the strengths and limitations of GPU programing?

GPU programming is about massive parallelism. Any program that requires thousands (or more) iterations of identical tasks is perfect for GPU programming - for example, computing matrices. On the other hand, GPUs are not as efficient with complex calculations especially regarding branching and looping. Furthermore, the memory model greatly limits GPU programming; any data that is processed, must be copied from the CPU memory into the GPU memory, loaded, executed, and then the results are copied back into the CPU memory.

(15 points) Among the four parallel frameworks, multithreaded programming, MPI programming, openMP programming, and GPU programming (e.g., CUDA), we discussed in the class, what will be the strategies you are going to use in general when selecting a parallel framework for your applications?

As I am a visual and tactile learner, I tend to build a pseudo program on draw.io. By creating a visual example of each 'step', I can physically split a task into pieces. Not only does this help me identify the computationally heavy parts of the program, thus which parts should be parallelized, it also helps me identify any communication that may be necessary and handle critical sectioning.

Once I understand the basic framework for the program, I can more easily select a parallel framework. For example, if I require more control over the individual threads, I might avoid openMP and focus on a multithreaded program. On the other hand, if my program requires shared memory, openMP could be a great option. I would also take into account whether I had a machine cluster available for MPI. Lastly, if my program required a massive amount of identical computations, I would consider using GPU programming; while I would avoid it for more complex computations involving looping and/or branching.

(10 points) Benchmarking of a sequential program reveals that 95 percent of the execution time is spent inside functions that are amendable to parallelization. What is the maximum speed up we could expect from executing a parallel version of this program on 10 processors using Amdahl's Law?

If we consider the formula for maximum speed up is $max = 1/((1-P)+P/S)$ where P is the part amendable to parallelization (thus $(1-P)$ is the part not amendable to parallelization) and S is improvement factor. Plugging in the values and solving gets us:

$$max = \frac{1}{(1-P)+\frac{P}{S}} = \frac{1}{(1-0.95)+\frac{0.95}{10}} = \frac{1}{0.05+0.095} = \frac{1}{0.145} = 6.8$$

Therefore, we could expect a maximum speed up of ≈ 6.8x from executing a parallel version of this program on 10 processors using Amdahl's Law.

(5 points) Programming Assignment: A small college wishes to assign unique identification numbers to all of its present and future students. The administration is thinking of using a given various constraints that have been placed on what is considered to be an "acceptable" identifier. The "acceptable" identifier must meet the following constraints:

- The identifier is a six-digit combination of the numerals 0-9
- The first digit may not be a 0.
- Two consecutive digits may not be the same.
- The sum of the digits may not be 7, 11, or 13.

See attached folder for Programs, benchmarks are below.

Problem	*1 process/ thread*	*2 processes / threads*	*3 processes / threads*	*4 processes / threads*
Program1.c (sequential)	0.1058	n/a	n/a	n/a
Program2.c (MPI)	0.0961	0.0580	0.0385	0.0656
Program3.c (openMP or CUDA)	0.0796	0.0780	0.0732	0.0882