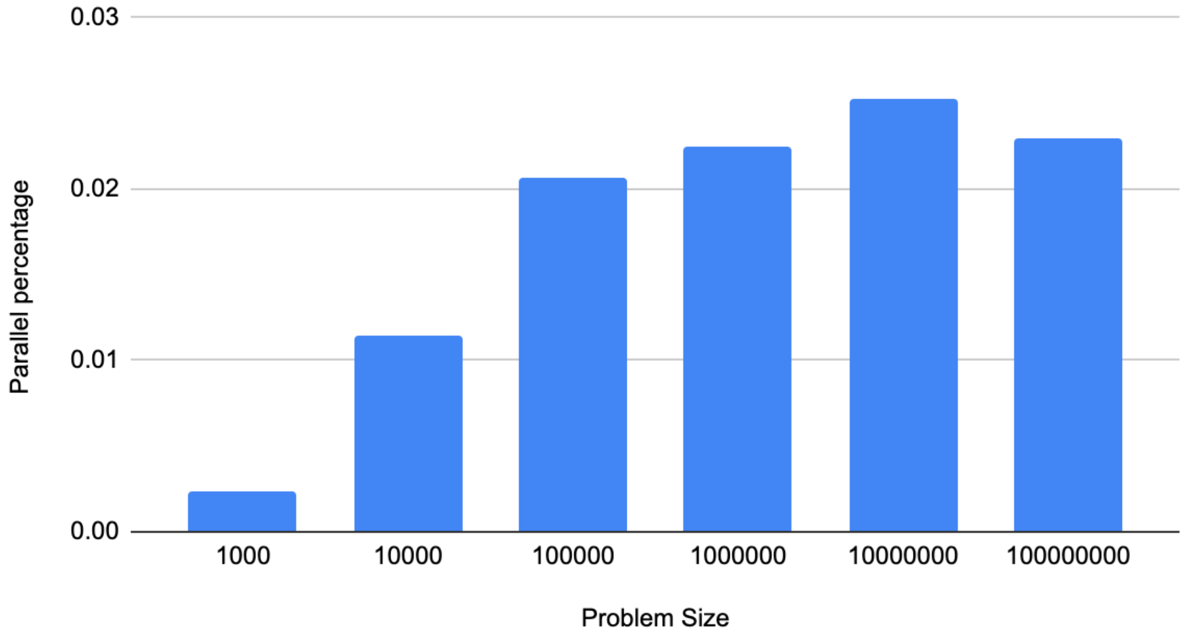
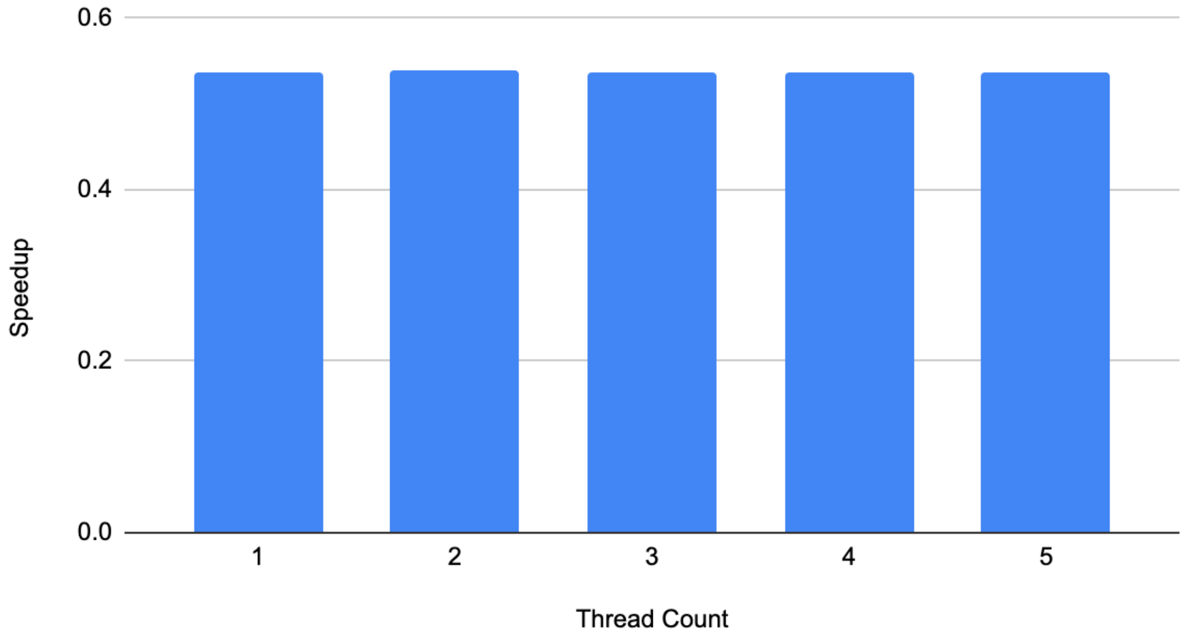
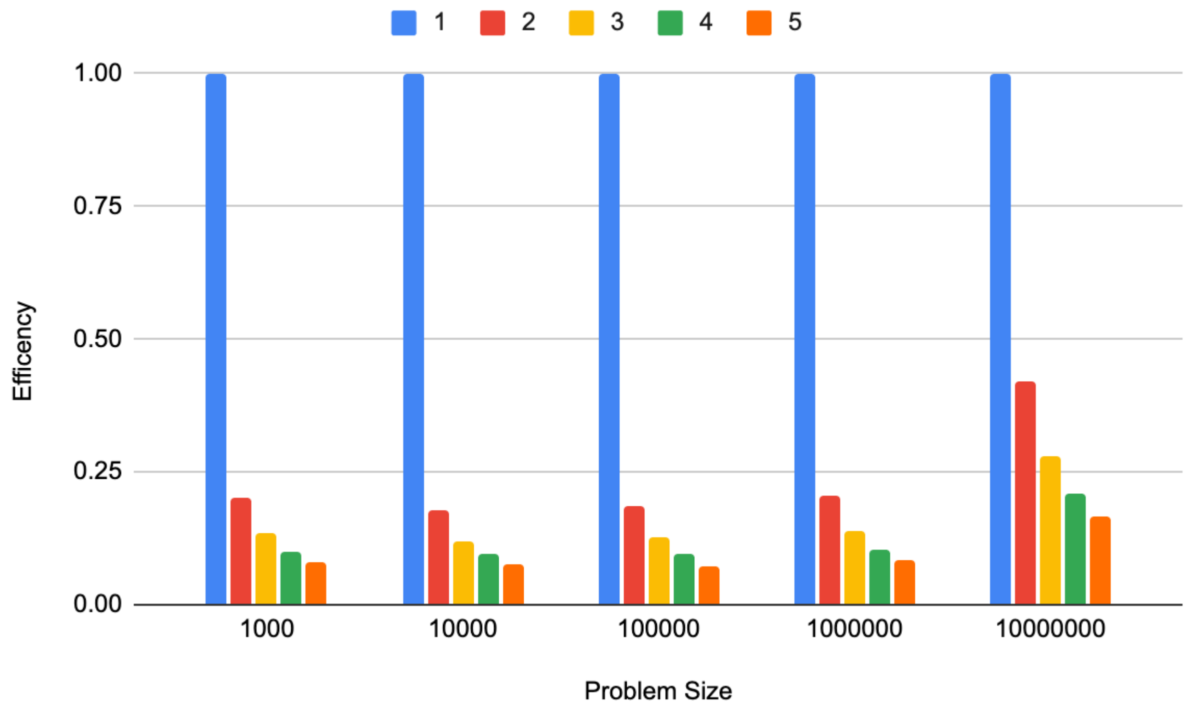
Parallel percentage vs. Problem Size



Speedup vs. Thread Count





The percentage of the program that is parallel very rapidly increases up until a certain point. At around a problem size of 1,000,000 it seems to stagnate and remains constant at around 20-25% of the program being parallelizable. This is a big problem, as because so little of the problem is parallelizable, the potential speedup is very limited, and in my specific implementation, I was never able to achieve a possible speedup, although it did get closer as the problem size increased.

In lab 1, as the number of threads increased, there was a very clear speedup. In my own implementation of the lab, there was a massive spike in speedup when the number of threads was maximized, which I assume was the case for most other programs as well. Unfortunately for this program I was not able to generate any speedup as the thread count increased, but I assume that if more than 5 threads were used, there definitely would have been potential for increased speedup. This makes sense, as with more threads being used, the work each thread does decreases. And because less work has to be done by each individual thread, the time taken to perform that work also decreases.

I was a little surprised by the efficiency of my program when I first looked at it, but looking back it makes a lot of sense. Until the very largest problem size the program is substantially slower than a purely parallel program. At a problem size of 10000000, the largest problem size, there is a very clear boost in efficiency which makes sense. The efficiency is still nowhere near ideal, but the jump does make sense. In each individual problem size, we can see that efficiency decreases as we add threads. This also makes sense, as so much of the program is sequential that for most of the time, the extra threads are sitting idle.