Parallel programming homework: part 1

# Question 1.

**Amdahl’s law** states that if denotes the time required to solve a problem on processors and if is the fraction of the algorithm that is parallelizable, then

1. Derive a formula for the maximum theoretical speedup possible on a machine with infinite processors. (Note: the speedup for using processors is .)
2. Write a Python function speedup(p) that plots the speedup as a function of the number of processors for an algorithm that has the fraction parallelized (as above). Plot on a semilogx axis (why?) for between 1 and 16384 processors.
3. Use the above function to compare on one graph the speedups possible (for various numbers of processors) for .

# Question 2.

Amplifying and propagating signals is essential for cellular function. The Python code at

<https://github.com/ramcdougal/cbb750/blob/master/propagating-signal.py>

generates snap-shots of a bidirectional propagating signal at various time points. Rewrite the code to run in parallel using Python’s multiprocessing module using shared memory (multiprocessing.Array). Create and use a multiprocessing.Pool to avoid the overhead of repeatedly creating new processes. Test your code to make sure it gives the same results as the original when run with 1, 2, 4, 8, and 16 processes. Measure the runtime (e.g. using time.time) for each of these cases on your hardware and report the results. (Please do not attempt to optimize the code otherwise; it is written the way it is to ensure each advance takes a non-trivial amount of time.)

**Hint:**

By default multiprocessing.Array controls access to the shared memory with a lock; you can override this for better performance (but be careful to avoid race-conditions) via, e.g.:

y = multiprocessing.Array('d', 1000, lock=False)

**Hint:**

You cannot pass a multiprocessing.Array to map for a Pool, but you can send it to the process during Pool creation; e.g.:

process\_pool = multiprocessing.Pool(  
 num\_processes,  
 initializer=init\_process,  
 initargs=(y, new\_y))

where we have elsewhere defined:

def init\_process(y\_to\_share, new\_y\_to\_share):  
 global y, new\_y  
 y = y\_to\_share  
 new\_y = new\_y\_to\_share