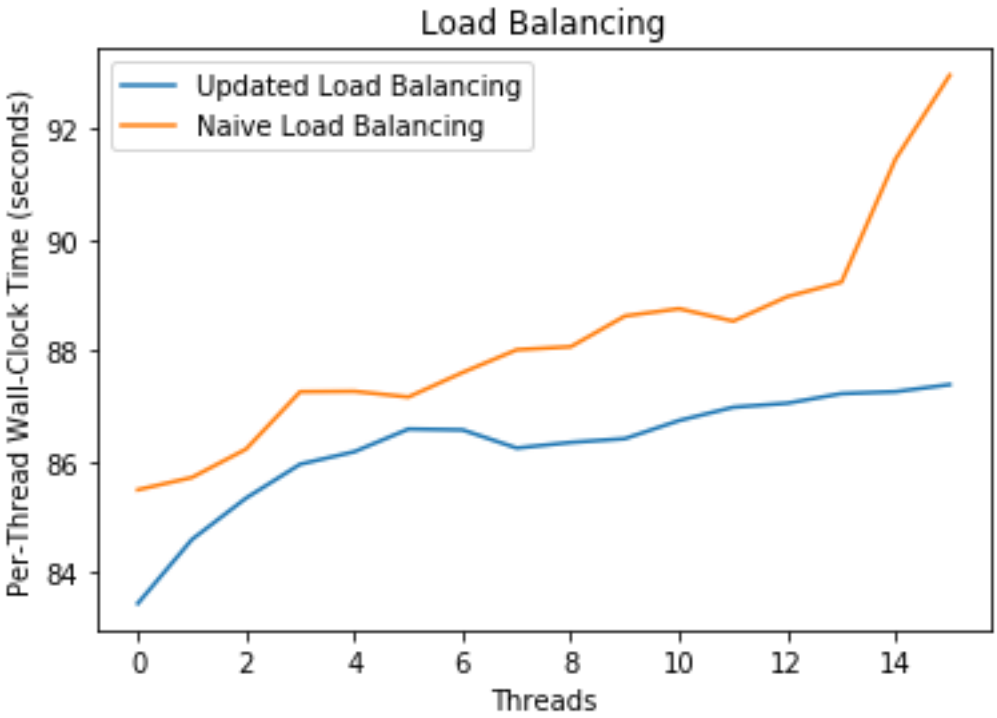
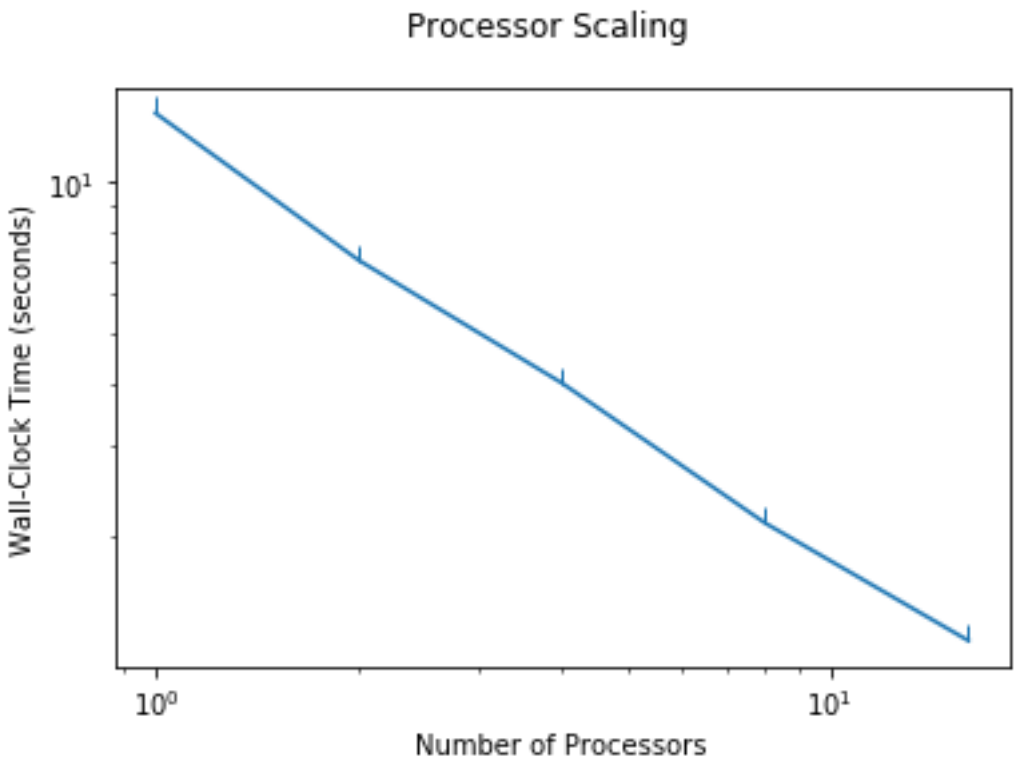
Prelim Problem #10

200256677



I ended up trying two methods of load-balancing. My first idea, labelled in the above plot as the 'naïve' approach, was to iterate through the integers with a stride equal to twice the number of threads (skipping evens). That is, when running on 4 threads, thread 0 would start by checking if the number 3 was prime, then 11, 19, 27… and so on. However, this lead to some load imbalance over a large range of integers, since thread 0 was always working with a smaller number than thread 8 at the same iteration. My improvement, labelled in the above plot as the "updated" method, was to alternate strides each iteration. With this scheme, thread 0 would start with 3 as before, then handle 17 (a stride of 14), then 19 (a stride of 2), 33, 35, and so on. This lead to a slightly flatter curve above, as threads' workloads are better balanced than before.



I found that my code scaled up well, as O(n), so doubling the number of processors halved the total wall-clock time. I should note that each of the wall-clock times plotted here (for N=1, 2, 4, 8, 16 processors) represents the average wall-clock time over 20 repetitions as detailed in the included “run.sh” script.