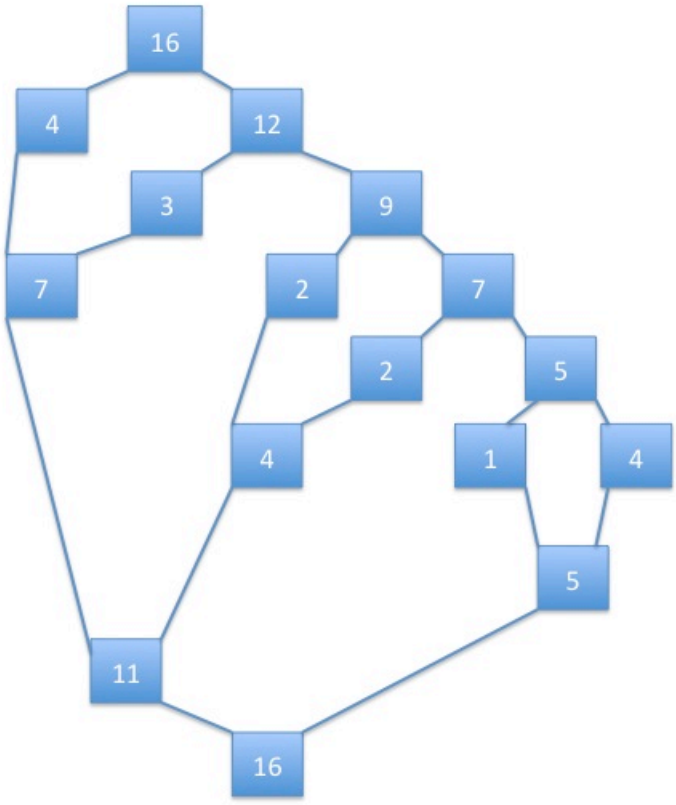
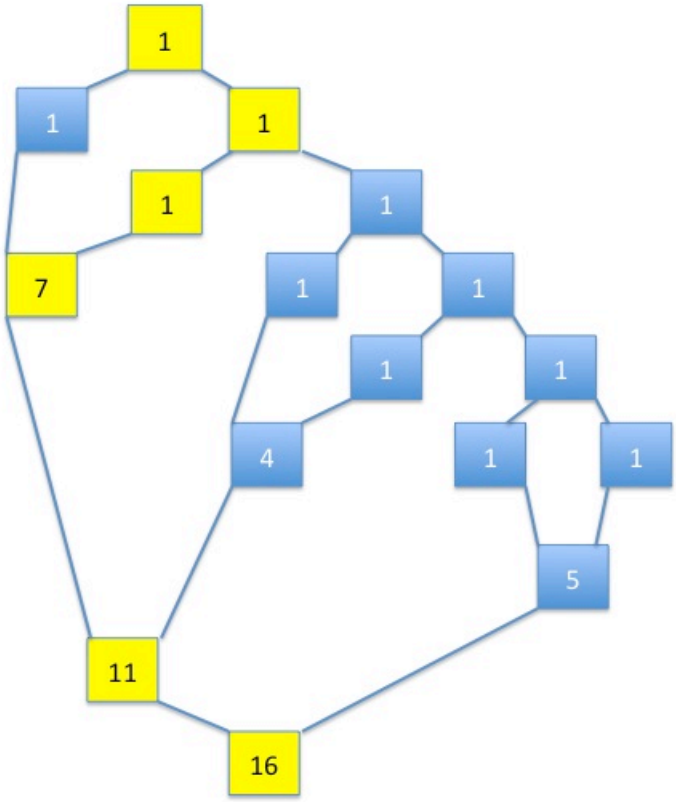
Marc Beitchman
CSE P 506
Assignment 1

Question 1:

a) This DAG shows the number of elements at each task.



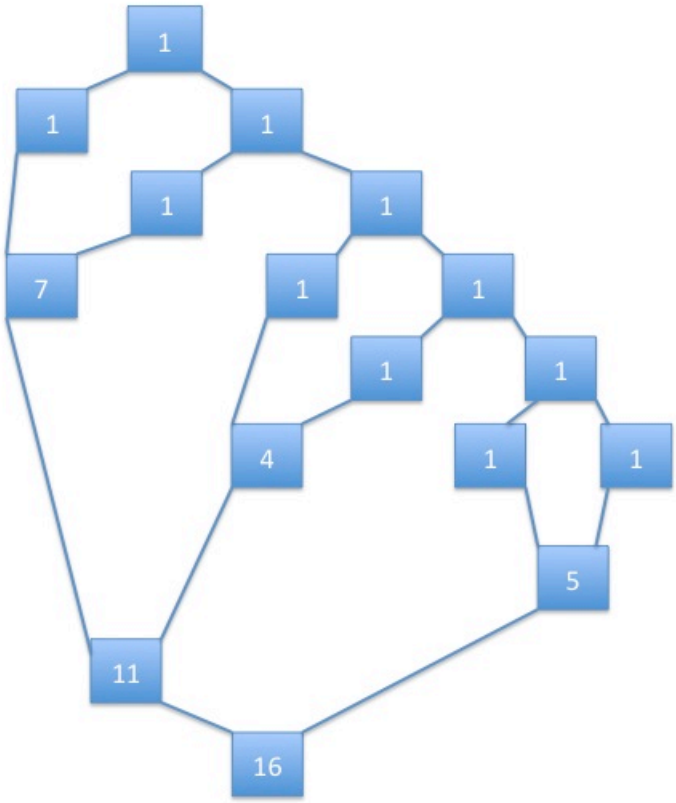
b) $T\infty = 1 + 1 + 1 + 7 + 11 + 16 = 37$



The depth of the DAG presented in class was 27. This splitting distribution in this problem results in a graph with a longer depth than the one shown in class. This distribution will result in a slower execution since execution time has a lower bound of the critical path length given infinite processors.

c) T1 = 11 + 7 + 4 + 5 + 11 + 16 = 54

This DAG shows work done at each node.



T1 for the example in class is 39. The splitting distribution used in this problem creates a DAG that has more overall work (sequential cost) than the splitting distribution shown in class. The increase in work for this problem is mainly due to the increase in merge tasks.

d) T4 for this example is 37 which is T∞. T4 for the example in class is 27.

e) N equal to 4 processors results in Tn equal to T∞. This result follows because the dependencies in the DAG allow for only four processors to ever be used to execute all tasks in parallel given the inherent dependencies in the DAG. Adding more than four processors will result in idle processors that aren't used to execute tasks in the DAG.

f) The splitting distribution is this problem results in worse performance than splitting each task in half since it results in more work and a longer critical path length than splitting each task in half. Splitting each task in half is the best strategy because it results in a balanced DAG that results in the shortest possible critical path, smallest amount of total work and ensures work can be evenly distributed given enough processors.

Question 2:

The proof of $Tp(greedy) \leq \frac{T1}{Tp} + T\infty$ is as follows:

We know there are only *complete* (all processors in system utilized) or *incomplete (not all processors in system utilized)* steps performed by a greedy scheduler.

Consider the *complete* steps.

The work law ($\frac{T1}{p} \leq Tp$) shows that time spent on all processors must be greater than or equal to the total work divided by the total number of processors. This law shows total work must be completed even thought it is parallelized.

A contradiction to the work law would be defined by $Tp < T1/P$.

- Multiply both sides by P results in $PTp < T1$.

This shows that total number of operations performed is less than the total work, which is a contradiction of the work law.

Consider the *incomplete* steps.

Since we have modeled the computation as a DAG, we know the depth (longest-path) of the DAG starts at a vertex with degree 0. Since an incomplete step of a greedy scheduler executes all paths within degree 0 of the DAG, the subgraph after the incomplete step must be one less than the subgraph prior to the incomplete step. Therefore, the incomplete step decreases the depth of the remaining DAG by 1 and the greatest possible number of incomplete steps is at most $T\infty$.

This shows $Tp(greedy) \leq \frac{T1}{Tp} + T\infty$.

Question 3:

For this problem, I implemented two scheduler algorithms. One algorithm (max) tries to steal from the processor that has the most tasks in its queue compared to the other processors. The other algorithm (random) randomly picks a processor to try and steal from. I measured only the max algorithm, as I did not see enough of a performance difference between the two to determine if one scheduler was more efficient than the other.

This graph *(Figure-1)* shows the input size of an array of random integers on the x-axis and speed-up between the sequential mergesort and the parallel mergesort

implementation on the y-axis. The machine used for these measurements had 8-core Intel Xenon 2.67GHz processors with 16GB of RAM. Each point represents the mean of 20 runs. The mean was measured to be in a 95% confidence interval using the R software package. All of the raw data for these measurements is available in the 'rawdata' folder.

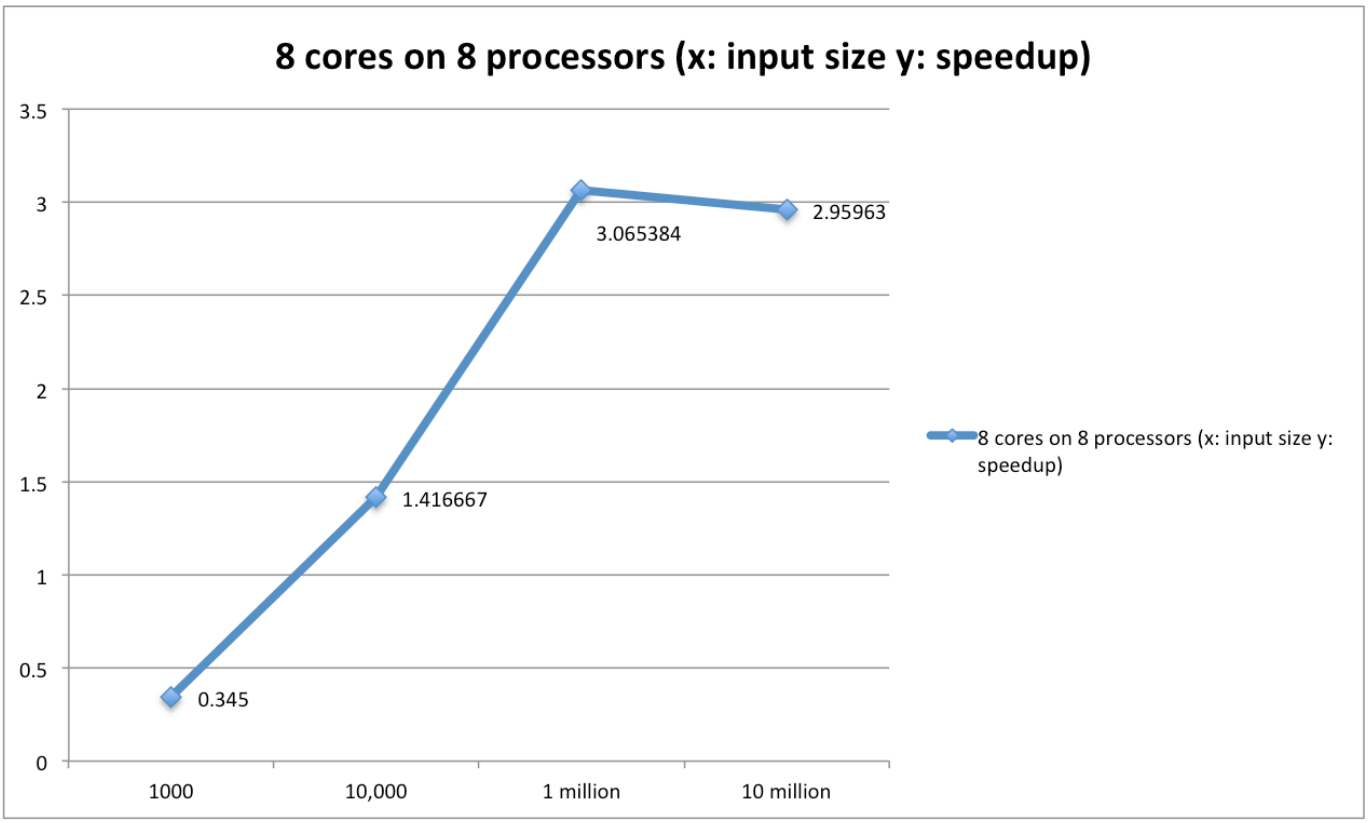


Figure-1

This graph shows the affect of Gustafson's law, which states the parallel portion of execution increases as problem size increases. The algorithm appears to have a maximum speedup of about 3 times.

The next graph *(Figure-2)* shows machine configuration along with number of processors (threads) created on the x-axis and speed-up of the sequential mergesort compared to the parallel mergesort on the y-axis. The dual core machine had AMD Athlon X2 2.6GHz processors with 3GB of RAM, the quad core machine had IntelQ9400 2.67 GHz processors with 8 GB of RAM, and the eight core machine had Intel Xenon 2.67GHz processors with 16GB of RAM. Each point represents the mean of 20 runs. The mean was measured to be in a 95% confidence interval using the R software package. All of the raw data is available in the 'rawdata' folder.

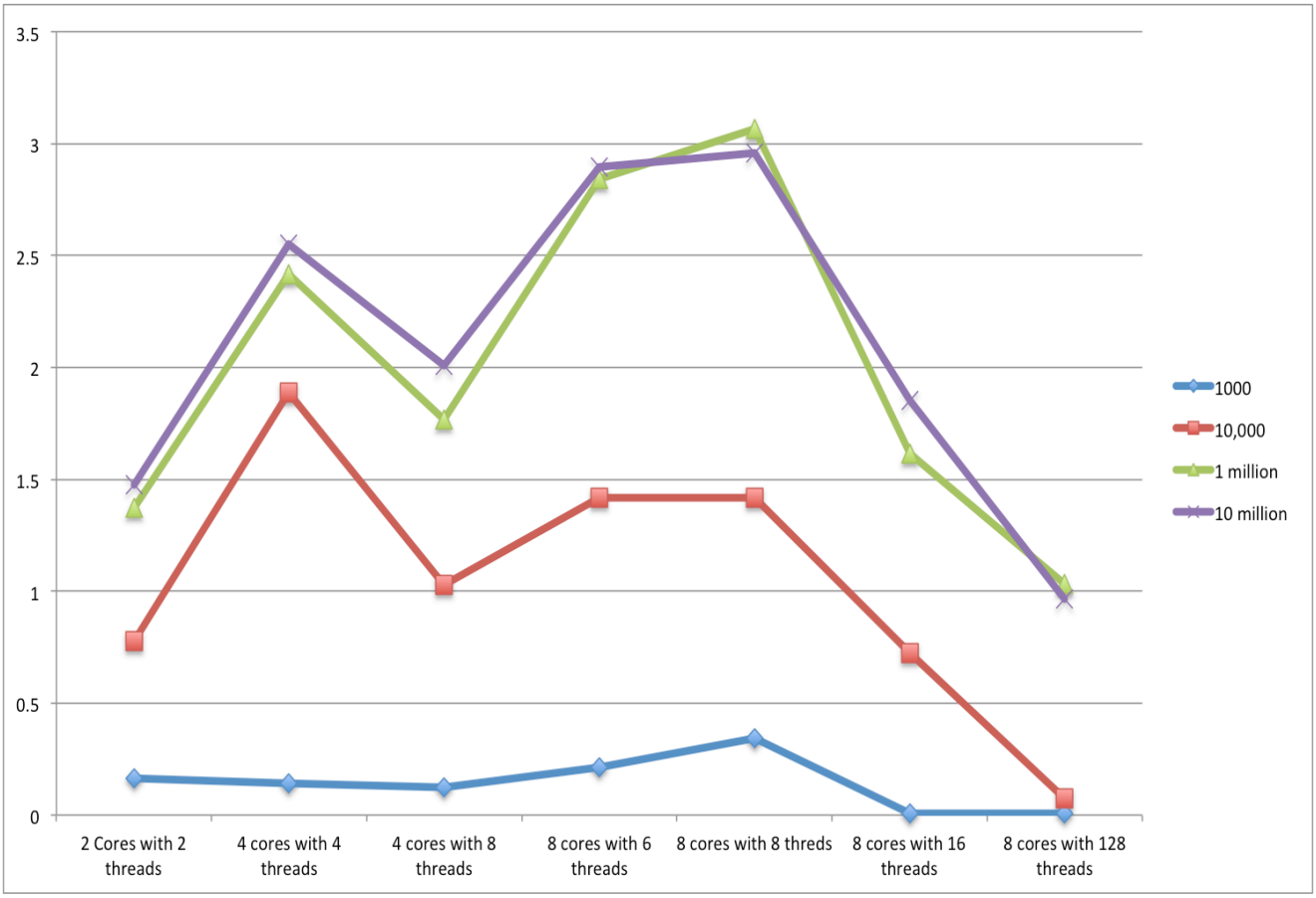


Figure-2

The most interesting pattern in this graph *(Figure-2)* is the decline in speed-up when creating more threads then available cores. This can happen due to lock contention when having multiple threads trying to steal tasks from individual queues of processors. Another possible reason for the slowdown is due to the memory overhead of running a large number of threads. This graph shows why it is generally more efficient to use the task model and only create 1 thread for each processor rather than creating lots of system threads.