Michael Laucella
HW2

1. Caches
    a. Write-through:
        i. Pros: when a block is swapped out it doesn't need to update the lower level cache
        ii. Cons: if a value is continuously being changed it needs to write every change to the cache
    b. Write-back:
        i. Pros: better performance it only writes to the lower levels when the value is removed, so if the value is constantly being used it won't write it every time
        ii. Cons: if a block is removed it now needs to write the value before the new block can be loaded

2. When multiple loads to main memory are being called from a thread/task we can swap out the thread with another thread/task to reduce the amount of CPU time that would be wasted waiting.

3. Cache Latency v Cache Hits: As the cache levels descend from 1 to 3 the cache sizes become bigger but slower, because the caches are larger they hold more blocks granting more hits as the cache levels descend. Hence cache hits become more important than the access time as hit rate should increase while speed will decrease.

4. Using Amdahl's law 1/(F+(1-F)/P) we know F=.2 and 1-F=.8 if we use T(INF) we get 1/F and 1/.2=5 so the upper bound for speed we can reach is 5 by using T(INF) processors now if we sub in 4.999 instead of 5 we can calculate a T(20000) if a more concrete number for P is what is desired.

5. Coherence is expensive, it requires that the updated value within the cache be updated in every other cache it's located in as well as in main memory, and with access to main memory alone being expensive we can see that all this transportation on the bus would be expensive.

6. Speedups:
    a. The increase in the size of the data increases the T(s) and since speed is defined as T(s)/T(p) if T(s) increases faster than the rate T(p) speed will continue to get higher so by doubling the data T(s) increases and by using more threads T(p) gets smaller while p gets larger
    b. Eventually for each curve as the number of processes increases eventually we will hit a point where some of the threads created don't even have data to run

and so the speed won't be increasing but rather decreasing as more overhead is being added with each useless thread

7. If the number of threads increases while the amount of data remains stagnant what can occur is that more overhead occurs as well as the threads performing less and less work, because each thread is now doing less work and having a greater percentage of its work dedicated to overhead, so even though there is a speed increase the increase per thread is worse.

8. Synchronization and load balancing both affect the time of the parallel program, in both positive and negative ways, by not being balanced or load imbalance the time for certain threads will be longer, and if there’s increased synchronization then more of the program will have serial attributes or more overhead. However if there is effective load balancing it can reduce overall program time same as if there is less synchronization.

9. The reason for this discrepancy is due to the first and last instruction which are not split among the T(8) these 2 sections would be serial without which is why it is a workload of 50 which divided by 8 returns 6.25 we can see this because .25 of 8 = 2 which we can depict as the 2 non parallel steps in the DAG.

10. No we cannot assume they will finish at the same time, they may not start at exactly the same time plus there are other unknown factors such as cache/memory/other IO times which can differ, as well as OS scheduling which can swap out the threads and resume them at any time.