Problem 1

Writethrough: Pros: Simplier architecture, and no need for a modified or dirty state. Cashe will always be up to date. Cons: Slightly slower write i/o but promotes faster reading
Writeback: Pros: Slightly faster since it does not need to write to main memory. Promotes write heavy use. Cons: Needs to stop memory from reading in order to check of cached version is up to date. Needs to check the dirty block. Difficulty implementing.

Problem 2

We know that Message transmission time is dependent on latency, the length of the message, and bandwidth. Because the latency is fixed, (no cache), we can either shrink the word size, or increase the bandwidth.

Problem 3

As we get closer to the bus, the latency time becomes smaller and smaller, and so the cache hit rate becomes more important to make sure that most of the cache being read are right.

Problem 4

The percentage time of the problem can be modeled by $T = \frac{80}{c} + 20$ where c is the number of cores. 6 cores results in a 3x speedup, while an infinite amount of cores is needed for a 5x speedup.

Problem 5

Coherence Protocol changes the way cache lookup and writes are implemented and both Snooping and Directory-based implementations have their drawbacks and advantages. While snooping tends to be faster by looking on the bus for changes, it does not scale well if the bus increases. The directory is where shared variables must be placed and a lookup is required to make sure there is coherence.

Problem 6

The reason the larger problem is bigger is because the Speedup constant has a larger value. This is because both the serial component and the parallel component grow linearly as the problem grows and the more effectively more cores can be parallel. If the graph were to be extended, both small and medium problems with have $\log x$ style growth while the double problem grows linearly.

Problem 7

Effectively, because the number of processes is a term on the bottom, as it grows, efficiency decreases. Intuitively, this means that as more relatively expensive calls to spawn a process (system call) grows, it becomes more resourceful to solve the given problem at hand.

Problem 8

As the number of synchronization tasks grow, the load balancing becomes more imperfect because more tasks become dependent on the results of another, resulting in threads having to wait on the the task that takes the longest time to reach the sync point. This results in idle time increasing. Optimally, the number of sync point is minimized.

Problem 9

This is calculated by the fact that the total amount of work $T_1 = 50$ and $T_\infty = 8$ from the path of the longest dependent task. The reason why parallelism isn't 8 is because the first task and the last task are both dependent and cannot be parallelized. This explains the the drop to 6.25

Problem 10

No, because the data they are operating on are different, and having essentially one sync point at the end, even though the results are not dependent on each other, this does not guarantee that they will complete at the same time even with the same operations.