1. Warmup: Why do we use caches? Please relate this to the von Neumann bottleneck.

Because in von Neumann's model for computer architecture, the processor and memory are connected by a bus which has limited capacity and the speed of processors is significantly faster than that of main memory. The processor would be waiting for a long time for data to be transferred from the main memory to the processor, thus wasting lots of resources. However, after introducing cache as a middle man, the processor can get recently used data way much faster than before, by simply fetching it from the cache without having to send a request back to the main memory and wait. The load of the bus could be decreased as well. Moreover, the cache can fill in the gap of performance difference between main memory and processors, CPU can now ask for data in chunks and take advantage of locality and preload data without having to wait for data delivery every single time.

2. Apply Amdahl’s law to compute the speedup for the following program if you have (a) 1, (b) 2,(c) 4, (d) 8, (e) 12, (f) 16, and (g) ∞ CPUs. In the following diagram, S portions are sequential and P are parallelizable.

The formula for Amdahl's law is 1/(F + (1 - F) / P)
Every = sign here means "around" truncating except the first digit after the decimal.
a) 1 because it is single core, therefor parallelism doesn't work. Everything is sequential
b) 1 / (0.33 + (1 - 0.33) / 2) = 1.5
c) 1 / (0.33 + (1 - 0.33) / 4) = 2.0
d) 1 / (0.33 + (1 - 0.33) / 8) = 2.4
e) 1 / (0.33 + (1 - 0.33) / 12) = 2.6
f) 1 / (0.33 + (1 - 0.33) / 16) = 2.7
g) 1 / (0.33 + (1 - 0.33) / ∞) = 3.0

3. Explain the difference between concurrency and parallelism with an example: if an operating
system is executing three long-running programs, how would its scheduler execute the programs
concurrently on one core, concurrently on three cores, or in parallel on three cores? Comment on
running the programs in parallel on one core.

The main difference between concurrency and parallelism is that the former concept relates to the ability of running several tasks in a small interval of time on a single core and the later concept refers to the ability of running several tasks simultaneously on more than one core. Assume that there are three long-running programs on a single computer C. If C has one core and concurrently runs these jobs, then one program might run before others or some part of these programs would be scheduled to run first and stop to wait for some other parts of these programs to run and maybe resume later. No two programs are run at the exact same time but they would be finished together in a small interval of time. To the viewer, it might feel like all of these three programs are running together because C actually runs some of one task and some of other tasks and keep executing instructions of these programs alternately to produce the illusion that they are running together. If C has three cores, then it wouldn't make a big difference from having only one core running concurrently. If C has three cores and supports parallelism, then in best cases, all three programs could run literally together, at exactly same time, and instructions by each of these programs could be handled by each core. If C has only one core and supports parallelism, then it is not possible to have the actual simultaneous actions of multiple programs.

4. If we can have fast caches on the same die as a CPU core, why bother having main memory at all? Why not just have bigger caches? (There are multiple right answers here; see if you can think of one or more reasons, and explain each one. Guessing one correct answer and several wrong ones is worse than a single correct answer.)

The point here is the tradeoff between speed and space. If a memory storage device is far away from cpu then it would be inevitably slow as electrons take times to run across. And if a device is too big then it is not feasible to be integrated into the same die as the cores. If we have a really big cache, could store as many data as modern RAM, then there would be lots of transistors in a small area (the die), which would produce too much heat to be dissipate.

5. Relatedly, what’s the point of cache? Does its purpose differ between single- and multi-core
processors (if so, how)? Does its implementation differ between single- and multi-core
processors (if so, how and why)?

The purpose of cache memory is to compensate the speed difference between processors and main memory. With it, processors could fetch recently used data way faster then sending a request to the main memory and next time when a specific data is delivered from the main memory, a chunk of other potentially useful data are also packed up and sent to the cache. This would be a speed-up if processors use them in a short period of time. Cache memory serves as a middle man with a good balance of speed and size in the memory hierarchy. The main purpose of cache in single and multi core machines are not the same as besides what cache do in single core machine, L2 and L3 caches in multicore machines are shared across cores, which makes them very good medium for messaging passing and synchronization. In multicore machines, each core would have their own private L1 cache that other cores wouldn't be able to access but have L2 and L3 shared across cores. Because of these, the cache coherence requirements become even harder to achieve than in single core machine.

6. You are implementing a server that sends responses to requests, and requires very small
amounts of computation to handle each request. Your application requires many clients and
many servers to work together. Which of the programming models discussed in class are you
likely to use to model this application, and why?

I would choose LogP model. Because we need several servers that might geographically be separate from each other, the network communication latency becomes an important factor, and having a shared memory (either implemented using a huge memory or enforced by a consensus protocol) is not necessary. LogP assumes a distributed memory which would best correspond to the real implementation of this application. Since most requests are very lightweight, each server could handle most of the quests fairly easily which need more frequent synchronization across the servers. Whereas having a single barrier for synchronization at the end of every super-step would make the overall system less consistent and introduce overhead, which makes BSP model not a good candidate. BSP model also has drawbacks like not being able to distinguish well between sending a long message versus sending multiple short messages of the same length in total and achieving good locality. PRAM is not as a fit as LogP because it has assumed shared memory, which is not a precise assumption for the real hardware that this system is likely to build upon.