1. Differentiate single-core CPU and multi-core CPU: A single-core CPU has only one processing unit, while a multi-core CPU has multiple processing units. Multi-core CPUs can perform multiple tasks simultaneously, while single-core CPUs can only perform one task at a time[1].

2. What are the issues available in handling the performance?: The following are some of the issues that can affect performance:

- CPU-bound tasks: Tasks that require a lot of processing power can slow down the system.
- I/O-bound tasks: Tasks that require a lot of input/output operations can slow down the system.
- Memory-bound tasks: Tasks that require a lot of memory can slow down the system.
- Contention: When multiple threads or processes compete for the same resources, it can lead to contention and slow down the system[1].

3. Write a mathematical formula for speedup and efficiency of parallel program: The speedup of a parallel program is defined as the ratio of the time taken by the best sequential algorithm to the time taken by the parallel algorithm. The efficiency of a parallel program is defined as the ratio of the speedup to the number of processors used. Therefore, the mathematical formula for speedup is:

`Speedup = T(1) / T(p)`

where `T(1)` is the time taken by the best sequential algorithm and `T(p)` is the time taken by the parallel algorithm using `p` processors. The mathematical formula for efficiency is:

`Efficiency = Speedup / p`

where `p` is the number of processors used[1].

4. How do you define the term false Sharing?: False sharing occurs when two or more threads update different variables that happen to reside on the same cache line. This can lead to unnecessary cache invalidations and slow down the system. False sharing can be avoided by ensuring that variables that are frequently updated by different threads are placed on different cache lines[1].

5. Analyze whether the complexity of an algorithm dominates run time: The complexity of an algorithm does not always dominate the run time. Other factors such as the size of the input, the hardware configuration, and the implementation details can also affect the run time. Therefore, it is important to analyze the performance of an algorithm under different conditions[1].

6. What is meant by message queue?: A message queue is a mechanism that allows different processes or threads to communicate with each other by sending and receiving messages. A message queue can be used to implement various communication patterns such as request-reply, publish-subscribe, and work queues[1].

7. Mention the types of wakeup calls in producer thread: The following are the types of wakeup calls in a producer thread:

- Empty wakeup: When a consumer thread wakes up a producer thread because the buffer is empty.

- Full wakeup: When a consumer thread wakes up a producer thread because the buffer is full.

- Timeout wakeup: When a producer thread wakes up after a certain amount of time has elapsed[1].

8. Write down the process of identifying the Synchronization: The following are the steps involved in identifying the synchronization points in a program:

- Identify shared data: Identify the data that is shared between different threads or processes.

- Identify critical sections: Identify the sections of the code that access the shared data.

- Identify synchronization primitives: Identify the synchronization primitives such as locks, semaphores, and barriers that are used to protect the critical sections.

- Analyze the code: Analyze the code to ensure that the synchronization primitives are used correctly and that deadlocks and race conditions are avoided[1].

9. Point out the effect of cancel construct: The cancel construct is used to terminate a parallel region prematurely. The effect of the cancel construct is that all the threads in the parallel region are terminated immediately, and the program continues execution outside the parallel region. However, the cancel construct can lead to unexpected behavior if not used carefully, and it is recommended to avoid using it unless absolutely necessary[1].

10. What is the process of flush operation in OpenMP?: The flush operation in OpenMP is used to ensure that all the updates made by a thread to shared variables are visible to other threads. The process of flush operation involves the following steps:

- Identify shared variables: Identify the variables that are shared between different threads.

- Insert flush directive: Insert a flush directive before the point where the shared variables are accessed.

- Execute the program: Execute the program and ensure that the updates made by one thread are visible to other threads[1].