EDAN26 example questions 2021

1. What is Amdahl's law? Explain what it means and why it is important.
2. What can Helgrind and the Google thread sanitizer help you with in multicore programming?
3. What is an actor in Scala/Akka?
4. What happens to an actor when it receives a message? (is it interrupted, for instance?)
5. Which guarantees about message arrival order does Akka give?
6. How can you avoid data-races in Scala/Akka programs?
7. What is the difference between `self` and `this` in Akka?
8. When an Akka actor receives a message, how can it know the identity of the actor that sent the message so it can make a reply?
9. If now Scala/Akka is much slower than Java and C, why would you consider using it for an Internet company that is intended to grow to millions of users?
10. Explain the terms
    - *Decomposition*
    - *Assignment*
    - *Orchestration*
    - *Mapping*
11. Which two of the terms in the previous question are commonly together called *partitioning*?
12. What are the main goals in partitioning?
13. What is the difference between static and dynamic assignment of tasks to threads and when should each be used?
14. What is meant by systolic array? Why *can* it give very high performance and why is it difficult to achieve that performance for many programs?
15. The Tera architecture was a parallel computer with no cache memories. What was the architectural feature that would give high performance anyway?
16. What does it mean that a multicore is a *cache coherent shared memory multiprocessor*?
17. What is a *directory* of a memory block in a cache coherent shared memory multiprocessor?
18. What is the idea with a *cache-only memory architecture?* and why would that for some programs be more efficient than a normal cache coherent shared memory multiprocessor with a fixed home location of each memory block?
19. What happens in Java if the main thread calls `run` on a thread?
20. What is the semantics of a synchronized block in Java, and what is an object's *entry set* and *wait set* in Java?
21. What is meant by a lock being *reentrant*?
22. What is the semantics of `volatile` in Java (that is, what happens conceptually) ?
23. Is `volatile` in C the same as in Java? (yes/no answer is sufficient)
24. What is the purpose of using a condition variable? Give an example.

25. Is the operating system kernel always involved when a thread waits for a Pthreads mutex in Linux? Why or why not or when?

26. What is Sequential Consistency? How is it defined? (not word for word but the essence)

27. Give one example of a compiler optimization that can break Sequential Consistency, and motivate why it can break it.

28. What is a write buffer?

29. Why is it, usually, a good idea to let reads ”bypass” writes, but why can that break Sequential Consistency?

30. Why can overlapping writes break Sequential Consistency?

31. Why can non-blocking reads break Sequential Consistency?

32. What it is a cache coherence protocol and which types of messages can it have and what would their purpose be?

33. Why could it be useful to have both an *exclusive state* and a *modified state* of a cache block?

34. Why can relaxed memory models improve performance?

35. What is the difference between Weak Ordering and Release Consistency?

36. What is Release Consistency and what happens at an *acquire* and at a *release*?

37. When can `memory_order_acquire` and `memory_order_release` be useful?

38. When can `memory_order_consume` be useful?

39. When can `memory_order_relaxed` be useful?

40. What is the main difference between the two lines that increment a variable?

```
atomic_int      a = 0;

a += 1;
a = a + 1;
```

41. Explain the C11 function `atomic_compare_exchange_weak`. Which parameters does it take and what does it do with them? Why is it called *weak*?

42. The C11 function `atomic_compare_exchange_weak_explicit` has two additional parameters. What are they used for?

43. Explain what the pair of instructions *load-and-reserve* and *store-conditional* are used for and what they do?

44. Which memory consistency model is used for non-atomic variables in C/C++?

45. If two threads access different element from the same array in C, is that a data-race? The same question for different struct-members?

46. Why do you think very few compilers implement `memory_order_consume` and for compilers which don’t implement it, how can libraries define it and still have working code?

47. What does *synchronizes with* mean in C/C++ and how can it be achieved?

48. What does *dependency ordered* mean in C/C++ and how can it be achieved?

49. What does *happens before* mean in C/C++ and how can it be achieved?

50. Consider a store with memory order release on Power. Should the `lwsync` instruction be before or after the store, and why? (the answer would be similar for other architectures including ARM although, of course, the corresponding instruction would be called something else).

51. Explain what is meant by a pointer *owning* an object in Rust.

52. Explain what is meant by *moving* an object in Rust.

53. Suppose a function wants to pass a pointer to a function and then continue using the pointer after the called function has returned. How can that be done?

54. How do you need to modify the answer to the previous question if you want the called function to modify the pointed to object?

55. What does the `move` in the following code mean?

    ```
    thread::spawn(move || {
            let val = String::from("hi");
            tx.send(val).unwrap();
    });
    ```

56. What is an `Arc` in Rust and what can it be used for?

57. What is the key idea with transactional memory?

58. Software transactional memory for Clojure is much more reasonable than for some other languages. What makes C/C++/Java less suitable for software transactional memory compared to Clojure?

59. Power has the following instructions for hardware transactional memory. What are they used for?

    - `tbegin.`
    - `tend.`
    - `tabort.`
    - `tsuspend.`
    - `tresume.`

60. Data partitioning is important in multicore programming. Is there a big difference in what to consider when you use locking or transactional memory? Why or why not?

61. What are usually the two main issues to consider with hardware transactional memory on Power (at least if we want to avoid to many aborted transactions)? Were they a problem in your preflow-push program and why or why not do you think?

62. What is the meaning of `dosync` and `ref-set` in Clojure?

63. Even if a thread that has taken a mutex only intends to use it to update a few pointers, other threads may have to wait a long time. How can that be? Give some example of ”unexpected delays”.

64. What is the ABA-problem and how can your C/C++ program crash due to it (unless you avoid it)?

65. How can the ABA-problem be avoided in C/C++ in codes where objects are allocated and deallocated with the heap manager (`malloc`/`free` and `new`/`delete`).

66. Under which circumstances can you have the ABA-problem also in Java?

67. Explain the terms:

    - *blocking* vs *non-blocking*
    - *lock-free*
    - *wait-free*

68. Describe the main ideas of how a lock-free stack can be implemented in C. Which variable(s) need to be atomic and why?

69. Describe the main ideas of how a lock-free queue can be implemented in C. Which variable(s) need to be atomic and why? In a queue, two variables need to be modified but the atomic operations can only modify one at a time. How can that be solved?

70. Explain the terms:

    - *true dependence*
    - *output dependence*
    - *anti dependence*

71. Suppose you have a large (millions lines of code) C program with matrix computations that needs to be parallelized, and that both IBM's and Nvidias parallelizing compilers fail to improve the performance. What would then the main strength of OpenMP be, and how can you know where to start in the code?

72. What is a so called *perfect loop nest* and why is that a useful concept for a compiler that wants to execute loop iterations in a different order than specified in the source code?

73. What is meant by *data dependence equation* and what does the existence of a solution tell the compiler?

74. If the compiler has found a solution to a data dependence equation, how can it check if that solution is within the loop bounds?

75. What is meant by *dependence distance* and what is the *dependence matrix*?

76. If needed, the compiler can (sometimes) compute a loop transformation matrix U. Why should U be unimodular?

77. Given U and the loop bounds of the original loop nest, how can it compute the new loop bounds, in principle?

78. For each of the cache miss types listed below, first explain what it means, and then what you can do to try to reduce their number.

    - *true sharing miss*
    - *false sharing miss*

79. Why is it much easier to do *data prefetching* for arrays than for linked-lists and trees?

80. Why can software initiated data prefetching (created by the compiler or manually by the programmer) sometimes slow down the program instead of making it faster?

81. How can hardware quite easily do data prefetching and normally for which kind of data structure?

82. What can go wrong if you prefetch data and request ownership?

83. How can hardware transactional memory help (help as in make it better) a parallelizing compiler?

84. If you are implementing a new lock-free data-structure and are trying to understand why it sometimes crashes (very rarely), how can C11Tester be helpful? What does it do?