Part 1 – Overview

1. What is the von Neumann model?
   What does the vN model abstract? That is, what information in the target architecture is ignored?
   What is a good performance model for the vN model? That is, what is a good way to express potential application performance on an abstract vN machine?

2. We examined performance change over time (1970s to 2010s) and found that it had multiple components, e.g., those performance gains due to improvements in architecture and those not.
   What improvement(s) is(are) independent of architecture?
   Give an example of an architectural improvement.

3. What is Moore's Law?
   What are the implications of Moore's Law on architecture improvements?
   On non-architecture performance?
   How is it varying with time? (Is it dead?) This may take a little internet searching.

4. What is Dennard Scaling?
   What are the effects of Dennard Scaling on how computers have evolved (1990s-2010s)?
   How have the effects of Dennard Scaling varyied during that time?
   How is Dennard Scaling different from Moore's Law?

5. In general, performance can be improved by decreasing latency and improving bandwidth. We claimed that this translates into parallelism and locality. Explain. Or dispute if you have a good argument.
   What does efficiency have to do with this?
   Give an example (an application) that includes all three components (parallelism, locality, efficiency). That is show how performance of that application can be improved (with parallelism and locality) and how that improvement is limited (by lack of efficiency).

6. What is the power wall?
   How has it affected computer architecture? Give at least two examples of this.

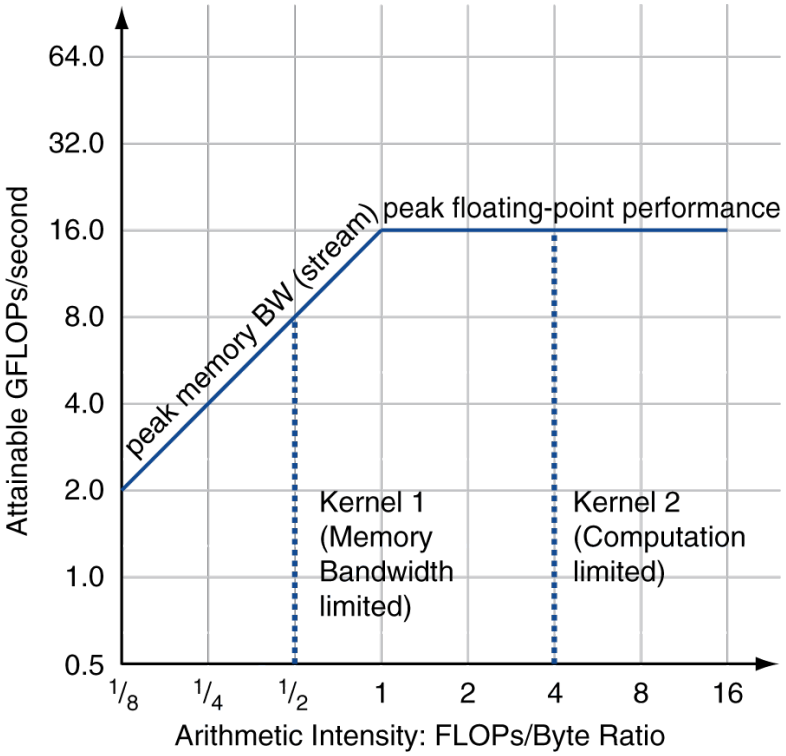
7. The most basic way to evaluate performance improvement is to measure wall clock time. But a crucial question is knowing how well you are doing, that is what fraction of potential improvements you have actually achieved. Explain how you could know this. Include how do you account for the characteristics of your computer.

8. What applications lend themselves to high performance programming? Why?

9. What is arithmetic intensity?
   What types of applications have high and low computational intensity?
   Why is knowing the applications' AI critical for obtaining high performance?

10. What is CPE? How do you measure performance using CPE?
    What is the benefit of using CPE?

11. Explain what a roofline model is. Then answer the questions below.



What is the peak memory BW? Assume 4 bytes per FP operand.

What is the peak floating point performance?

What limits Kernel 1 performance?

What limits Kernel 2 performance?

Can you move the ridge point (where the diagonal and horizontal roofs meet) by writing better code?

Part 2a – Review memory hierarchy

12. What are the major levels of a memory hierarchy? (at least four)
    What are their access latencies?

13. What types of locality are there? For each type, why is it important?
    What are the 0'th order ways to take advantage of them (through hardware support)?

14. For each type of locality, what kinds of programs do or do not have it?

15. What does the cache controller do? On a read hit? Read miss?

16. What is the "simple" model of memory hierarchy performance?
    What does this say about desired fraction of hits and misses? That is, what fraction hits should you target?
    Note that this is also a simple application of Amdahl's Law.

17. What is a cache block?
    Why are larger cache blocks generally helpful?
    What limits cache block size?

18. Why are multi-level caches essential?

19. What kinds of cache misses are there? (4 Cs)
    When do they occur?
    How are they likely to relate to Arithmetic Intensity?

Part 2b – Programming with memory

20. What is the programmer's view of memory (using a typical programmer's model)?
    How do arrays get mapped to this model? For references into a two dimensional array, how do you compute the memory locations that are accessed?

21. Given a simple cache model (fully associative, block size = something typical like 64B), what is the cache behavior (hit rate) for the following two code snippets. Start by describing the memory references traces.

```
for (i = 0; i < N; i++)
        sum += a[0][i];

for (i = 0; i < N; i++)
        sum += a[i][0];
```

22. For MMM, why does loop order matter?
    For each loop order, what is the miss rate? How does this change with matrix/cache size?

23. What is blocking? How do you block MMM code? Why does it work? How is performance related to matrix/cache size? To block size?

24. In the lecture notes we assume that all memory references are independent. We know, however, that modern memory hierarchies have many optimizations, like high bandwidth streaming. Let's consider how this affects the optimization of MMM. You may use roofline plots if that helps.
    Show that, with a memory that has *very* high bandwidth, loop interchange does not affect performance.
    Given kji and kij implementations of MMM with large N. For a single core of a CPU you used in this lab, what is the minimum memory bandwidth needed to make MMM CPU bound.

Part 3a – Intel Assembly Language

25. How many registers are there? How big are they?

26. What are the addressing modes?
    How can they be used together in an instruction?

27. How are parameters passed in function calls?

28. How is flow control implemented?
    What are condition codes?

29. What are conditional moves?
    When are they useful or not useful?

Part 3b – Optimizing program performance

30. What is asymptotic performance (Big-O)? What are its limitations?

31. What are some generally useful optimizations for all programs?

32. What is an "optimization blocker"?
    Give two examples (with code) and show how do you can remove them.

33. Modern CPUs and ILP:
    What function units are there?
    What is the pipeline depth of the function units?
    What is the latency of data transfer from one unit to another?
    How is control flow implemented?
    Put all this together: for the "combine" code example, what is the critical path? Explain.

34. Loop unrolling. Why is it necessary for many optimizations?
    What are the limitations? (This is important to understand – loop unrolling by itself may not have much benefit on modern systems, e.g., in previous question, without other CPU features.)
    What is code scheduling? How does loop unrolling help?

35. What is the accumulator problem?
    What are two ways to deal with it?
    How does this problem relate to loop unrolling?

36. What's hard about branches? (Improving performance of code with conditional branches)
    Why can they kill program performance?
    What can you do about it? (from the programming assignment)
    What are the limitations of this method?

Part 4a – Intro to SIMD and Vector processing

37. What is Flynn's computer architecture taxonomy (SIMD, MIMD, etc.)?
    For each of the four quadrants give an example of a computer (of computer feature) that fits.

38. What is standalone SIMD architecture?
    What does the controller do?
    The PEs?
    How is conditional execution done?

39. What is dataparallel programming? In your favorite data parallel language (or pseudocode) write SOR.

40. Dataparallel checklist. For each of the following explain what it is: (ia) Map vector-vector, (ib) Map scalar-vector, (ii) Reduce, (iii) broadcast, (iv) conditional execution, (v) in-vector data movement, (vi) non-local memory references - scatter/gather.

41. What is a vector architecture?
    How is it different from the standalone SIMD architecture?
    How is it different from our standard CPU core? (with ILP as described above)

42. What are standard vector AL primitives?
    What are the different types of memory references? Describe scenarios where they would be used.
    How are conditional operations implemented? Give a code example.

43. How do memory banks work?
    How is this architecture different from standard interleaved memory?
    What are the benefits?

44. Start with the DAXPY code in the notes. Extend it to arbitrary size vectors.

45. For the example "Vector example with dependency" what is the problem? Explain the solution

Part 4b – Programming vector units on Intel CPUs (AVX)

46. There are many ways to access AVX hardware in executing a program. Name five.

47. How big are AVX registers?
    How can they be used? What data types?
    Why is alignment important?

48. Give examples of the major instructions types (Vector-vector, Vector-scalar, Data movement, Loads/Stores, Conditional operations, "Weird" instructions (add-sub), Macros)

Part 5a – Parallelizing compilers

49. When can instructions be executed parallel?

50. When is code vectorizable?

51. When is code parallelizable through data decomposition?

52. What's the difference between vectorizability and parallelizability?
    Give an example of code that is vectorizable but not parallelizable.

53. What is a loop carried dependence?
    What's the difference between an ITG and and LDG?

54. What is a valid transformation?
    Why is this stronger than needed?

55. What is SOR? What is the complexity of the serial version?
    In the serial implementation, what are the dependences?

56. What is the "soap bubble" problem? How does this relate to SOR?

57. How can SOR be parallelized given all the serial dependences that must be preserved?
    What is the algorithmic complexity of this mapping?

58. How else can SOR be parallelized? (Red-black) How does this improve complexity?
    Explain how Red-Black and serial implementations change the physics of the application.

59. What does all this say about the limits of parallelizing/vectorizing compilers?

Part 5b – Intro to parallel programming

60. What kinds of parallelism are there? What are some examples of each?

61. What are the four steps in creating a parallel program (given the constraint of a serial reference code)?
    For MMM, describe what you would do in each of these steps.

62. How can SOR be decomposed? How can SOR tasks be assigned?

63. For the SAS version of SOR, how does each thread know what to work on? What is the purpose each barrier? Of the lock?

65. Show how you could remove one of the barriers without affecting correctness.

64. What is the communication to computation ratio?
    Why is it important?
    How does it depend on P and N (in general)?
    What does this have to do with weak and strong scaling?

65. In SOR, we looked at partitioning by strips versus squares. How does the partitioning affect the communication to computation ratio?

66. In SOR, how does partitioning affect inherent communication? (In this case through coherence misses)

Part 6 – Threads programming

67. What is the programming model assumed in threads programming? (title page)

68. What are the essential attributes of a thread?
    How do they differ from those of a process?
    Why must each thread have its own stack?
    How do threads differ from processes in efficiency?

69. How do you pass arguments to a thread?
    A single variable?
    An arbitratrary parameter list?

70. What is "join"?
    Why do you do it?

71. What is a MUTEX?
    When must they be used?

72. How do you use MUTEX's for synchronization (e.g., among threads/processes)?

73. What is a barrier?
    How do you use it?
    When do you use a barrier instead of a MUTEX?

Part 7 – Review Concurrency

74. What are dependences?
    Why are they important?
    Why do we ignore WAW and WAR and not RAW?

75. What is the critical section problem?