Azmani Sultana

Id: 22201949

Section 11

① What do you understand by 'Performance Via Prediction' in terms of computer architecture? Give a proper example of it.

– 'Performance via Prediction' is a design principle in computer architecture where systems use speculation to guess the outcome of operations to enhance performance. Instead of waiting for a condition to be resolved, the processor makes a guess and continuous execution based on that guess. Statistically, the guesses are often correct, so the performance improves because the system avoids idle cycles. If incorrect, the processor discards the work and rolls back. Here, the cost of recovering from a wrong guess is relatively low.

Example: In computing, this concept is similar to speculative execution in CPUs. The processor guesses the likely path of a program and starts executing instructions ahead of time. If the guess is correct, the program runs faster. If the guess is wrong, the processor discards the work done on the wrong path and starts over from the correct path. As long as the prediction

is usually right and the cost of recovering from a wrong guess is low, this approach speeds up the overall performance.

(2) Why is it important to keep redundancy while designing a system? Explain a scenario where this redundancy will be useful.

- Redundancy is crucial in system design for reliability, fault tolerance, and data integrity. Computers need to be both fast and reliable. Since any hardware can break, we make systems reliable by adding extra components that can take over if something breaks and to help spot any problem.

Example: Dual power supplies.

Servers often have two power supplies. If one fails, the other takes over instantly and the system keeps running without interruption.

③ a) Explain Amdahl's Law in your own words.

Amdahl's law: This law helps us to understand the overall performance improvement gained by optimizing a single part of a system.

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

The law describes the potential speedup of a system when only a portion of it is improved. It states that the overall performance gain is limited by the part of the system that remains unimproved.

b) Amdahl's law strongly relates to Design Principle 3: Make the common case faster. Since the overall speedup is most influenced by the part of the system used most often, optimizing the common case (frequently executed parts) yields the best performance improvement.

Example: If 90% of a task is in floating-point computation, improving only that portion will significantly boost performance. Enhancing a rarely used portion will not help much, as Amdahl's Law shows diminishing returns on less-used improvements.

④ Narrate a scenario where increasing the throughput could also improve the response time. Justify your answer.

Scenario: In a web server handling thousands of requests, increasing throughput means more requests processed per second. If the server processes more requests simultaneously, it reduces the time each request waits in the queue.

Justification: Although throughput measures system productivity and response time measures the delay for a single task, reducing queuing delays via higher throughput can indirectly reduce individual response times, especially in high-load systems like cloud services or database servers.

⑤ To calculate the benchmark of a system, why do we take the geometric mean instead of only taking the average of the individual spec ratios?

- To calculate the benchmark of a system, we use geometric mean to summarize SPEC ration because it correctly handles multiplicative performance ratios, gives a more balanced and fair average

and is less affected by extreme values, providing a more reliable system performance metric.

Example: If a system runs 3 programs with speedups of 2X, 1X and 0.5X, the geometric mean is:

$$\sqrt[3]{2 \times 1 \times 0.5} = \sqrt[3]{1} \approx 1.587$$

This gives a meaningful average speedup, unlike the arithmetic mean which would be misleading.

⑥ From those factors, Instruction count is directly affected by the programming language used

i) High level languages like Python or Java usually produce more instructions due to abstraction.

ii) Low level languages (like C or assembly) can be more optimized, producing fewer instructions for the same task.

CPI may be influenced indirectly by the instruction mix. Here, Language influences instruction types, which can affect average CPI. But clock rate is not affected at all. Because, it is mainly hardware dependent and compiler influence not directly tied to programming language.