Computer Architecture Week 7

Khaula Molapo

Scenario Analysis

In a database server, cache design is very important for making data access faster. The cache stores frequently used data in memory so that the system doesn't always have to go to the hard drive. I would design the cache using multiple levels — L1, L2, and L3. L1 would be the fastest but smallest, while L3 would be larger but slower. The data would move between levels depending on how often it's used. The cache would use a Least Recently Used (LRU) method to remove old data when space is full. This design helps reduce delays and improves performance when many users are trying to access the same data. It also reduces the workload on the main storage and makes the overall system much faster and more efficient.

Concept Research

Cache coherence is about making sure that all the copies of data stored in different caches stay the same. When multiple processors use their own caches, they might change data at the same time. Without cache coherence, one processor might see old data while another sees new data, which causes errors. Cache coherence keeps everything updated and synchronized by using rules or protocols like MESI (Modified, Exclusive, Shared, Invalid). This ensures that if one processor updates data, others automatically get the correct version. It's mainly used in multiprocessor systems to make sure data stays accurate and consistent across all cores.

Tool Practice

In Visual Studio Code, I simulated a simple cache program using Python. The program stored frequently accessed values in memory and replaced older ones when the cache was full. At first, I didn't understand how caching worked, but testing with sample data made it clear. I used a dictionary to store items and counted hits and misses. Watching how cache hits reduced processing time showed me why caching is so useful. It saves time and system power because it avoids fetching data repeatedly. The simulation also helped me understand how replacement strategies like FIFO and LRU work in real systems.

Application Practice

A cache optimization strategy helps improve how data is stored and accessed in a system. I designed one where frequently used data is stored closer to the CPU and updated automatically. The strategy uses predictive algorithms that guess what data will be needed next based on past activity. It also

clears unused data to make room for new information. By managing cache sizes and timing properly, the system becomes faster and more efficient. This reduces processing delays, improves user experience, and saves energy. Overall, the purpose is to balance speed, storage space, and performance for smooth operations.