Impact of Chip-Level Integration on Performance of OLTP Workloads

Summary

The paper investigated the performance benefits integrating an L2 cache and memory controller on-chip, and found that chip-level integration can improve performance by 1.4 to 1.5 for online transaction processing (OLTP) workloads. The authors identified the main reasons to be:

1. For uniprocessors, lower L2 cache latency.
2. For multiprocessors, lower L2 cache latency and lower dirty remote latency (communication misses).
3. Higher L2 associativity leads to less misses than larger off-chip cache

Interesting Points

1. The primary benefits from chip-level integration is more efficient communication interfaces.
2. OLTP workloads are very sensitive to memory system performance, especially cache-to-cache communication.
3. Many OLTP cache misses are conflict misses.
4. Kernel component is approximately 25% of the total execution time (user and kernel).
5. OLTP workloads have low processors utilization (< 20%).

In-class Unanswered Questions

What market are we interested in?

1. High-end servers
2. Volume servers
3. High-end desktop
4. Game consoles (multimedia)
5. Mobile devices
6. High-performance computing (HPC)

What are the most important workloads for the market?

What are good benchmarks for the workloads?

Example of a bad methodology:

There is a guy standing a underneath a street lamp searching for something. You ask, "did you lose something?" He replies, "yes, my keys." You ask, "where did you lose them?" He replies, "somewhere over there", as he points across the street. You ask, "why are looking over here?" He replies, "because the light is better."

Are there programming models that change the behavior of a workload?

For example, threaded vs. staged programming models for online transaction processing (OLTP).

How does the parallelism between workloads differ?

For example, matrix operations for game consoles and HPC workloads vs. transactions for high-end and volume servers workloads.

For a specific workload, what are good tradeoffs between computation, caching and communication?

Should computational resources be heterogeneous or homogeneous?