Architecture and Design of the AlphaServer GS320

Kouroush Gharachorloo, Madhu Sharma, Simon Steely, and Stephen Van Doren

Presented by: Zachary Drillings

Motivation and Goals

- Achieve a low overhead directory protocol optimized for medium scale systems
- Address correctness issues related to rare protocol races without slowing common transaction flow
- Nacks seriously degrade performance and set up situations where livelock/starvation can occur.
  - Even using acks causes unnecessarily high numbers of messages in some cases, as well.

Implementation and Design

- Simple Interconnects, such as a crossbar switch
  - Infeasible for large networks, but fine for this scale
  - Can exploit the ordering properties of the switch!
- Quad-Processor Nodes, using Alpha 21264, up to 32 GB memory per node.
  - Can connect up to 8 nodes for a 32 Processor System using a global switch.
  - Directory and Transactions-in-Transit maintained at each node
- Global Switch buffers packets and sends them out independently allowing for totally-ordered multicast, when needed

Coherence Model

- Invalidation Based. Four Message types:
  - Read
  - Read Exclusive
  - Exclusive
  - Exclusive-without-data
- Requesting Processor doesn’t need to wait for acks, because of ability to enforce total ordering.
- Allow for dirty-sharing
- 3 Virtual lanes
  - Q0 for carrying requests from requester to home directory (totally ordered!- > invalidates are “delivered” as soon as they are scheduled on the switch)
  - Q1 for carrying replies from home directory
  - Q2 for carrying replies to processor from third-party

No Naks

- No need to deal with liveness and starvation as a consequence of naks.
- Guarantee the owner can always service a request
- Fewer Messages
- Figure 3 - Really neat case where this has an advantage over Naking methods.
- Deal with Races by:
  - Late request race: Hold onto valid copy until home directory acknowledges a write-back
  - Early request race: Delay request on Q1, until data arrives on Q2 (total ordering on Q1 prevents deadlocks)

Consistency

- Early Acknowledgement
  - Define commit event for each write, commit only needs to complete before another write occurs.
- Separate data and commit components in replies to data requests.
  - Data is time critical and arrives at requester as fast as possible
  - Commit stays ordered on Q1 line
  - Can’t go past memory barrier until both are received
- For Read and read exclusive, can do Early Commit in certain circumstances. -> Go past barrier if commits are recieved

Discussion/Evaluation

- Not terribly impressive latency improvements when Processors Idle, but better when active.
    - No Snoopy bus
- Somewhat mixed results for various benchmarks against different competitors.
    - Unclear what differing latencies really mean to real-world performance
    - Unclear how valuable benchmarks are, as well as price/performance ratio.
- In general, unconvincing, in my opinion.

Thoughts and Questions

- Some very interesting ideas.
  - Totally ordered requests, no ordering in data replies.
  - No Naking
  - Reduced number of messages and interesting solutions to races, etc.
- However,
  - Is this better than competing designs?
  - How well are these ideas going to scale?
  - How does additional hardware increase cost?
  - How does this system change with a switch to multiple cores on chip? Can it still be useful? Could it, in fact, be an even better model, since nodes could be a single chip and much fast?
  - With 32 cores, is this even a big win over previous generations?