Multi-Core Computer Architecture: Storage and Interconnects Week 4

Aronya Baksy

September 2021

1 Small and simple L1 Cache

- The critical timing path for a cache access is:
  1. Indexing
  2. Tag comparision
  3. MUX control selection (way selection)
- A direct-mapped cache overlaps tag comparision and data transmission, hence hit time is reduced
- Lower associativity reduces power requirement as fewer cache lines are accessed.

2 Way Prediction

- Methods to predict way in a set to reduce the hit time. This gives the reduced hit time of a direct-mapped cache with the benefits of an n-way associative cache
- When the predicted way results to a miss, then the remaining tags in the same set must be compared in parallel.
- Extra bits are used to predict the MRU way (block) within each set.
- Power gating can be done on the unused ways within the set to reduce power consumption

3 Victim Cache

- A buffer near the L1 cache, Any block that is evicted from the L1 cache is stored in the victim cache (L1 and victim are exclusive caches)
- If there is read miss in L1, then search victim cache before searching L2 cache.
- Avoids thrashing problem in L1 cache
- In case of a hit in the victim cache, either the victim cache can itself service the request, or block may transfer to L1 and replace another block which is itself moved to victim cache.

4 Pipelined Cache

- Splitting cache memory access into stages and pipeline these stages (index, tag read, hit/miss check and data transfer)
- Advantages: increased bandwidth, easier to have higher associative caches
- Disadvantages: slow in hits (hit/miss check is only done after 2 cycles), increases branch misprediction penalty

5 Non-blocking Cache

- In case of a miss in a traditional L1 cache, it requires lookup in the higher level caches. During this time, the processor is stalled
- Hit under miss: even if non-blocking L1 cache undergoes a miss for one request, it can still service hits in L1 cache
- Hit under multiple misses: even if non-blocking L1 cache undergoes multiple misses for many requests, it can still service hits in L1 cache
- OOO superscalar processors require non-blocking caches for IPC increase
- L2 cache contains an L1-MSHR (Miss Status Holding Register). ON an L1 Miss, allocate MSHR entry, clear it when L2 responds with the required block.
- L1 miss penalty is hidden, but not L2. Overlapping miss latencies in L1 reduce the effective miss penalty in L1
- Cache controller complexity increases due to > 1 outstanding misses, as well as pipelined memory requirement

6 Multi-Banked Cache

- Divide cache into independent banks that can be accessed in parallel (ARM Cortex A8 supports 1-4 banks for L2, Intel i7 supports 4 banks for L1 and 8 banks for L2
- The n^{th} block is kept in bank $n\%N$ where N is total number of banks
- Storing adjacent cache blocks in different banks (which can be accessed parallely) increases the bandwidth of the cache.

7 Early Restart Technique

- Do not wait for entire block to be loaded for restarting the CPU (during a stall)
- The technique works as follows:
  1. Request words in normal order
  2. Send missed word to the processor as soon as it arrives (don't wait for the entire block to be transferred)
  3. L2 controller is not involved in this technique

8 Critical Word First

- Request the missed word first, and send it to the CPU as soon as it arrives
- The processor continues its work while the rest of the cache block is filled up
- L2 cache controller forwards the words out of order (critical word is forwarded first). L1 cache controller must rearrange them into the block

9 Write Buffer Merging

- In a write-back cache, the processor updates all writes in cache as well as in main memory. The processor writes the data to a write buffer and continues, while the data is written from write buffer to memory
- If the write is being done to a block that is already pending in the write buffer (i.e. the same block is being updated again), then just overwrite that write buffer entry
- Reduces stalls due to full write buffer, improves buffer efficiency

10 Compiler Optimizations

10.1 Loop Interchange

- Swap nested loops in order to access memory in sequential order, maximises cache usage

10.2 Blocking

- Instead of accessing entire rows and columns, subdivide 2D array into blocks
- Requires more accesses, but improves the locality of accesses and cache utilization

10.3 Hardware Pre-Fetching

- Predict and pre-fetch items before the processor demands them. On a miss, fetch more blocks (include next sequential block)
- Prefetch controller looks into memory access patterns
- Requested block is kept in cache, the next sequential block is placed in the stream buffer.
- If the requested block is in the stream buffer, then cache miss is cancelled

10.4 Compiler Pre-Fetching

- The compiler inserts pre-fetching instructions before the data is needed.
- This results in performance gains only if processor reads from caches and executes while pre-fetching is in progress
- Loop unrolling and scheduling instructions allow for pre-fetching of adjacent iterations
- Two types; Register pre-fetch and cache pre-fetch