Advanced Cache Architectures and Virtual Memory

Advanced Cache Architectures

- AMAT = Average Memory Access Time
- AMAT = hit time + miss rate*miss penalty

Several ways to improve performance, then
- Decrease hit time
- Decrease miss rate
- Decrease (observed) miss penalty

Hit Rate vs. Hit Time

- Direct-mapped caches have low hit time, associative caches have low miss rate.
- Ideally, we’d like low miss rate and low hit time together.
  - Way prediction
  - Victim Cache

Victim Cache

- Small, fully associative buffer which holds recently evicted cache lines.
- Targets conflict misses
Reducing Compulsory Misses and Capacity Misses

- Prefetching
  - Brings data into the cache (or a special buffer) based on access patterns or program knowledge.
- Who does the prefetching?
  - Hardware (based on access patterns)
    - Most modern high-performance processors do this
    - Sometimes called stream buffers.
  - Software
    - Most ISAs support some kind of software prefetch
      - Works best for regular computation
    - A separate thread (in a multithreaded processor)
      - We called this speculative precomputation (2001)
      - Typically done by distilling a reduced version of the main thread

Reducing memory stalls

- A non-blocking cache is one that can still handle new requests after a miss.
  - Requires some extra bookkeeping to keep everything straight.
- Hit-under-miss (can have 1 outstanding miss)
  - Can continue to service hits after a miss
  - Stalls on second miss
- Miss-under-miss
  - Can have up to M outstanding misses at once

Tolerating cache misses

- Sometimes you can’t make the miss go away. But that doesn’t mean you have to stall. We tolerate misses by continuing to make progress in the face of cache misses.
- Miss tolerance techniques (increasingly effective)
  - Stall on miss (no tolerance)
  - Stall on use
  - Non-blocking caches
  - Out-of-order execution
  - Multithreaded execution (more about this later)

Cache Optimization Key Points

- Reducing Conflict Misses
  - Way prediction
  - Victim Cache
- Reducing Capacity or Compulsory Misses
  - Prefetching
- Tolerating Misses
  - Non-blocking caches
  - Out-of-order execution
  - Multithreading
Virtual Memory

It’s just another level in the cache/memory hierarchy

Virtual memory is the name of the technique that allows us to view main memory as a cache of a larger memory space (on disk).

Virtual Memory

• is just cacheing, but uses different terminology (and different storage/lookup techniques)

<table>
<thead>
<tr>
<th>cache</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>page</td>
</tr>
<tr>
<td>cache miss</td>
<td>page fault</td>
</tr>
<tr>
<td>address</td>
<td>virtual address</td>
</tr>
<tr>
<td>index</td>
<td>physical address (sort of)</td>
</tr>
</tbody>
</table>

Virtual Memory

• What happens if another program in the processor uses the same addresses that yours does?
• What happens if your program uses addresses that don’t exist in the machine?
• What happens to “holes” in the address space your program uses?

• So, virtual memory provides
  - performance (through the cacheing effect)
  - protection
  - ease of programming/compilation
  - efficient use of memory
Virtual Memory

- is just a mapping function from virtual memory addresses to physical memory locations, which allows caching of virtual pages in physical memory.

What makes VM different than memory caches

- **MUCH** higher miss penalty (millions of cycles)!
- Therefore
  - large pages [equivalent of cache line] (4 KB to MBs)
  - associative mapping of pages (typically fully associative)
  - software handling of misses (but not hits!)
  - write-through not an option, only write-back

Address translation via the page table

- all page mappings are in the page table, so hit/miss is determined solely by the valid bit (i.e., no tag)
- so why is this fully associative???
- Biggest problem – this is slow. Why?
Making Address Translation Fast

- A cache for address translations: translation lookaside buffer (TLB)

Virtual Memory & Caches

- Cache lookup is now a serial process
  1. V->P translation through TLB
  2. Get index
  3. Read tag from cache
  4. Compare
- How can we make this faster?
  1.
  2.

Virtual Caches

- Which addresses are used to lookup data in cache/store in tag?
  - Virtual Addresses?
  - Physical Addresses?
- Pros/Cons?
  - Virtual
  - Physical
### Fast Index Translation

- Can do
  1. V->P translation through TLB
  2. Get index

  in parallel, if the “virtual” index and the “physical” index are the same.

<table>
<thead>
<tr>
<th>virtual page number</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>index</td>
</tr>
</tbody>
</table>

### Virtual Memory Key Points

- How does virtual memory provide:
  - protection?
  - sharing?
  - performance?
  - illusion of large main memory?

- Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.

- Three things can go wrong on a memory access: cache miss, TLB miss, page fault.