Cache Optimizations III

Critical word first, reads over writes, merging write buffer, non-blocking cache, stream buffer, and software prefetching.

Reducing Miss Penalty Summary

$$\text{CPU time} = IC \times CPI + \text{Memory accesses} \times \text{Miss rate} \times \text{Miss penalty} \times \text{Clock cycle time}$$

- Four techniques:
  - Multi-level cache
  - Early Restart and Critical Word First on miss
  - Read priority over write
  - Merging write buffer

- Can be applied recursively to Multilevel Caches:
  - Danger is that time to DRAM will grow with multiple levels in between
  - First attempts at L2 caches can make things worse, since increased worst case is worse.

Early Restart and Critical Word First

- Don’t wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first
  - Generally useful only in large blocks (relative to bandwidth)
  - Good spatial locality may reduce the benefits of early restart, as the next sequential word may be needed anyway

Read Priority over Write on Miss

- Write-through with write buffers offer RAW conflicts with main memory reads on cache misses:
  - If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
  - Check write buffer contents before read; if no conflicts, let the memory access continue
  - Usually used with no-write allocate and a write buffer

- Write-back also want buffer to hold misplaced blocks:
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read
  - Usually used with write allocate and a writeback buffer
Merging Write Buffer

- Write merging: new written data into an existing block are merged
- Reduce stall for write (writeback) buffer being full
- Improve memory efficiency

Improving Cache Performance

1. Reducing miss rates
   - Larger block size
   - Larger cache size
   - Higher associativity
   - Victim caches
   - Way prediction and
     Pseudosassociativity
   - Compiler optimization
2. Reducing miss penalty
   - Multilevel caches
   - Critical word first
   - Read miss first
   - Merging write buffers
3. Reducing miss penalty or miss rates via parallelism
   - Reduce miss penalty or miss rate by parallelism
   - Non-blocking caches
   - Hardware prefetching
   - Compiler prefetching
4. Reducing cache hit time
   - Small and simple caches
   - Avoiding address translation
   - Pipelined cache access
   - Trace caches

Non-blocking Caches to reduce stalls on misses

- Non-blocking cache or lockup-free cache allow data cache to continue to supply cache hits during a miss
  - Usually works with out-of-order execution
- "hit under miss" reduces the effective miss penalty by allowing one cache miss; processor keeps running until another miss happens
  - Sequential memory access is enough
  - Relative simple implementation
- "hit under multiple miss" or "miss under miss" may further lower the effective miss penalty by overlapping multiple misses
  - Implies memories support concurrency (parallel or pipelined)
  - Significantly increased the complexity of the cache controller
  - Requires multiple memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses

Value of Hit Under Miss for SPEC

- FP programs on average: AMAT= 0.68 → 0.52 → 0.34 → 0.26
- Int programs on average: AMAT= 0.24 → 0.20 → 0.19 → 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

Reducing Misses by Hardware Prefetching of Instructions & Data

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
- On-miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on having extra memory bandwidth that can be used without penalty

Stream Buffer Diagram

- Tags
- Data
- Direct mapped cache
- Stream buffer
- Source: Jouppi /ICS'90
**Demand Page Faults**

1. Reducing miss rates
   - Larger block size
   - Larger cache size
   - Higher associativity
   - Victim caches
   - Way prediction and pseudoassociativity
   - Compiler optimization

2. Reducing miss penalty
   - Multilevel caches
   - Critical word first
   - Read miss first
   - Merging write buffers

**Reducing Miss Penalty or Miss Rates**

- Non-blocking cache
- Hardware prefetching
- Compiler prefetching
- Small and simple caches
- Avoiding address translation
- Pipelined cache access
- Trace caches

---

**Fast hits by Avoiding Address Translation**

- Conventional Organization
- Virtually Addressed Cache
  - Translate only on miss
  - Synonym Problem

- Antialiasing solutions
  - HW guarantees covers index field & direct mapped, they must be unique; called page coloring

- Solution to cache flush
  - Add process identifier tag that identifies process as well as address within process: can't get a hit if wrong process

---

**Fast Cache Hits by Avoiding Translation: Process ID impact**

- Black is uniprocess
- Light Gray is multiprocess when flush cache
- Dark Gray is multiprocess when use Process ID tag
- Y axis: Miss Rates up to 20%
- X axis: Cache size from 2 KB to 1024 KB

---

**Fast Cache Hits by Avoiding Translation: Index with Physical Portion of Address**

- If a direct mapped cache is no larger than a page, then the index is physical part of address
- Can start tag access in parallel with translation so that can compare to physical tag

- Limits cache to page size: what if want bigger caches and uses some trick?
  - Higher associativity moves barrier to right
  - Page coloring

- Compared with virtual cache used with page coloring?
Pipelined Cache Access

Alpha 21264 Data cache design
- The cache is 64KB, 2-way associative; cannot be accessed within one-cycle
- One-cycle used for address transfer and data transfer, pipelined with data array access
- Cache clock frequency doubles processor frequency; wave pipelined to achieve the speed

Trace Cache

- Trace: a dynamic sequence of instructions including taken branches
- Traces are dynamically constructed by processor hardware and frequently used traces are stored into trace cache
- Example: Intel P4 processor, storing about 12K mops

What is the Impact of What We’ve Learned About Caches?

1960-1985: Speed = f(no. operations)
1990
- Pipelined Execution & Fast Clock Rate
- Out-of-Order execution
- Superscalar Instruction Issue
1998: Speed = f(non-cached memory accesses)
What does this mean for

Cache Optimization Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>MP</th>
<th>MR</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multilevel cache</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Critical work first</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Read first</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Merging write buffer</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Victim caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Larger block</td>
<td>-</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Larger cache</td>
<td>-</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Higher associativity</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Way prediction</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pseudoassociative</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Compiler techniques</td>
<td>+</td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>