Reducing Cache Miss Penalty

- Multilevel caches, critical work first, reads priority over writes, merging write buffer

Improving Cache Performance

1. Reducing miss rates
   - Larger block size
   - Larger cache size
   - Higher associativity
   - Pseudoassociativity
   - Compiler optimization

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

3. 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

Multi-level Cache

- Add a second-level cache
- L2 Equations
  
  \[ \text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1} \]
  
  \[ \text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2} \]
  
  \[ \text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}) \]

- Definitions:
  - Local miss rate—misses in this cache divided by the total number of memory accesses to this cache (Miss rate_{L1})
  - Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU (Miss Rate_{L1} \times Miss Rate_{L2})
  - Global miss rate is what matters to overall performance
  - Local miss rate is factor in evaluating the effectiveness of L2 cache

Local vs. Global Miss Rates

Example:

- For 1000 inst., 40 misses in L1, 20 misses in L2
- L1 hit 1 cycle, L2 hit 10 cycles, miss 100
- 1.5 memory references per instruction

Local miss rate, AMAT, stall cycles per instruction, and those without L2 cache

With L2 cache
- Local miss rate = 50%
- AMAT=1+4\%\times100=5
- Average Memory Stalls per instruction=(3.4-1.0)\times1.5=3.6

Without L2 cache
- AMAT=1+4\%\times100=5
- Average Memory Stalls per Instruction=(5-1.0)\times1.5=6

Assume ideal CPI=1.0, performance improvement = \((6+1)/(3.6+1)=52\%\)
Comparing Local and Global Miss Rates

- First-level cache: split 64K+64K 2-way
- Second-level cache: 4K to 4M
- In practice: caches are inclusive

- Global miss rate approaches single cache miss rate provided that the second-level cache is much larger than the first-level cache
- Global miss rate is what matters

Compare Execution Times

- Performance is not sensitive to L2 latency
- Larger cache size makes a big difference

Impacts of L2 Associativity

L2 cache:
- Direct mapped hit time = 10 cycles
- 2-way 10.1 cycles
- Local miss rate for direct mapped=25%
- Local miss rate for two-way=20%
- L2 Miss penalty=100

Answer:
- Miss penalty 1-way = 10+25%x100 = 35
- Miss penalty 2-way = 10.1 + 20%x100 = 30.1
- If treat 10.1 as 11 cycles:
  - Miss penalty 2-way = 11 + 20%x100 = 31

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 buffer being full
- Improve memory efficiency

Reducing Miss Penalty Summary

\[
\text{CPU time} = \text{IC} \times (\text{CPI} \times \text{Miss rate} \times \text{Miss penalty}) \times \text{Clock time}
\]

- Four techniques
  - Multi-level cache
  - Early Restart and Critical Word First on miss
  - Read priority over write
  - Merging write buffer
Reducing Cache Miss Penalty by Exploiting Memory Parallelism

- Non-blocking cache, stream buffer, and software prefetching

Improving Cache Performance

1. Reducing miss rates
   - Larger block size
   - Larger cache size
   - Higher associativity
   - Pseudoassociativity
   - Compiler optimization

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

3. 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 increases 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

Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution

- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    » Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    » Can be incorrect. Frees HW/SW to guess!

- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

Improving Cache Performance

1. Reducing miss rates
   - Larger block size
   - Larger cache size
   - Higher associativity
   - Pseudoassociativity
   - Compiler optimization

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

3. 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

Reducing Cache Hit Time

Virtual Cache, pipelined cache, cache summary
Fast Cache Hits by Avoiding Translation: Process ID impact

- Black is uniprocess
- Dark Gray is multiprocess when use Process ID tag
- Light Gray is multiprocess when flush cache
- Y axis: Miss Rates up to 20%

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 same trick?
  - Higher associativity moves barrier to right

Pipelined Cache Access

For multi-issue, cache bandwidth affects effective cache hit time
- Queuing delay adds up if cache does not have enough read/write ports

Pipelined cache accesses: reduce cache cycle time and improve bandwidth

Cache organization for high bandwidth
- Duplicate cache
- Banked cache
- Double clocked cache

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

Summary of Reducing Cache Hit Time

- Small and simple caches: used for L1 inst/data cache
  - Most L1 caches today are small but set-associative and pipelined
  - Used with large L2 cache or L2/L3 caches
- Avoiding address translation during indexing cache
  - Avoid additional delay for TLB access

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>

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>Nonblocking caches</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Hardware prefetching</td>
<td>+</td>
<td>+</td>
<td></td>
<td>2/3</td>
</tr>
<tr>
<td>Software prefetching</td>
<td>+</td>
<td>+</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Small and simple cache</td>
<td>-</td>
<td>+</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Avoiding address translation</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pipeline cache access</td>
<td>+</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Trace cache</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>