11
System-Level Performance Effects

18-548/15-548 Memory System Architecture
Philip Koopman
October 7, 1998

Required Reading: Cragon: 3.6-3.6.1
Supplemental Reading: Hennessy & Patterson: 5.9
Mogul paper, 1991 ASPLOS

Assignments
♦ Read about tuning software for speed:
  • Cragon 2.8.3
  
  • Supplemental
  – Re-read Hennessy & Patterson pp. 405-410
  – Lam et al., 1991 ASPLOS paper
  – Uhlig, et al., 1995 ISCA paper
  – Intel architecture reference manual section 3.5

♦ Homework 7 due October 21
♦ Lab 4 due October 23
Where Are We Now?

◆ Where we’ve been:
  • Cache organization, construction & management policies

◆ Where we’re going today:
  • System-level effects on performance
    – Operating system
    – Tasking/Interrupts

◆ Where we’re going next:
  • Tuning software for speed
  • The rest of the memory hierarchy

Preview

◆ Why isn’t SPECmark cache performance the last word?
  • Understand why operating system can degrade cache performance
  • Understand how multitasking can increase conflict misses
  • Understand how code bloat affects cache performance
SPECmarks Give Optimistic Cache Performance

- **Operating system software**
  - SPECmarks only spend 2%-3% of time in operating system
  - GUI and file-intensive software may spend 20%-40% of time in OS
  - OS has different characteristics than SPEC code
    - More branches (poor spatial locality)
    - Fewer loops (poor temporal locality)

- **Multiprogramming**
  - Has effect of periodically flushing some or all of cache, increasing miss rate
    - Multitasking
    - Interrupts
  - “Transaction processing” involves many short program executions

- **Modern software practices and “code bloat”**
  - Complex, layered interfaces
  - Modularization of traditional software
  - Dispersed object-oriented methods

System+User Miss Rates

- **DTMR has a significantly increased miss rate**
  - This is for fully associative cache; might be worse with direct-mapped because OS code is more “scattered” -- increasing conflict misses

<table>
<thead>
<tr>
<th>EFFECTS OF SYSTEM CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISS RATE</td>
</tr>
<tr>
<td>0.30</td>
</tr>
<tr>
<td>0.25</td>
</tr>
<tr>
<td>0.20</td>
</tr>
<tr>
<td>0.15</td>
</tr>
<tr>
<td>0.10</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CACHE SIZE</th>
<th>SYSTEM-USER</th>
<th>DTM USER ONLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Flynn Figure B.1)
User Code Traces Are Optimistic

- **System Code** (Agarwal 1987)
  - System contributes 20% of misses (has poor instruction locality because of large number of branches)
  - Even worse, 10% - 20% added user code misses due to system call interference resulting in conflict misses

(Flynn Figure 5.21)

Concept In Real Life:

- Suppose you’re spending a Saturday repainting your house (with oil-based paint, no less).
  - You get a phone call from your boss saying you have a meeting with an important client in 1 hour, and you’d better get in to work in a suit
  - What is the cost of this context switch beyond simply changing clothes?

- What would it be called if it happens again 15 minutes after you get home and start painting again?
MULTIPROGRAMMING

Q-Factor

- **Q** = average number of instructions executed between task switches

- Small **Q** means frequent task switching
  - Not much time for temporal locality to take effect
  - Increases system responsiveness at cost of higher task switch overhead

- Large **Q** means tasks run a long time between switching
  - Temporal locality can be exploited
  - Reduces responsiveness of system, but lower task switch overhead
Cache “Footprints”

- Multiprogramming causes interference
  - “Footprints” from each task evict cache contents
  - Number of evicted lines depends on:
    - Q factor (large Q has more time to evict lines)
    - Locality of tasks
    - Degree of interference among tasks and how they map to cache
  - Higher associativity can smooth out miss “spikes” due to getting unlucky with cache footprint overlaps

Warm Caches

- Warm cache -- has suffered from little interference
  - Extreme of warm cache is one with no task switching at all (very high Q)

- Cache stays warm if:
  - Cache is big enough to capture working sets of all tasks
  - There are few conflicts
    - Set associativity reduces conflicts
    - Linker/loader or virtual memory mapping may reduce conflicts by biasing each task to a different group of sets
  - Tasks run for “a long time”
Cold Caches

- **Cold cache -- high degree of interference**
  - Extreme of cold cache is one that is entirely invalidated at high frequency
  - But even a cold cache may have some shared system routines still resident

- **Cache gets cold if:**
  - Executions are short (high proportion of compulsory misses)
  - Other tasks evict cache lines before task resumes execution
  - System software must flush cache to maintain coherence with memory
    - Loading software
    - I/O operations

---

Mostly Warm Cache -- 2 Tasks

![Graph](image)

- Multiprogramming level = 2
- Line size = 16 bytes

Average quantum size (memory references)
- 100
- 1K
- 10K
- 20K
- 100K

Single user with systems effects
- DTMR

(From Flynn Figure 5.25)
Slightly Warm Cache -- 10 Tasks

As Q becomes small, inter-task interference results in high miss rate

Transactions -- A Challenge

Transactions
- Very short applications (“transactions”) loaded on demand
- Tend to touch data a small number of times
- Short transactions don’t have much chance to benefit from temporal locality
  - High number of compulsory misses, no loops, touches data only once
  - In contrast to engineering/scientific workload, which has many loops
- Large caches may not help much if most misses are compulsory!

Example: automatic teller transaction
- Read current balance
- Subtract withdrawal
- Write new balance
- Move on to next transaction

Future challenge: “active data”
- Message data carries program with it; executed once when received
Cold Cache (Transaction Environment)

- If tasks end after a short time, large caches don’t help

![Graph showing miss rate versus cache size](image)

- Average quantum size (memory references)
  - 100
  - 1000
  - 10000
  - 20000
  - 100000

(Flynn Figure 5.26)

The True Cost of Context Switching

- Traditionally, cost of context switching is considered to be:
  - Processing timer interrupt
  - OS scheduling
  - Register save
  - Register restore

- BUT, cost of cache conflicts can be significant
  - 0.51% to 7.1% speed penalty measured by Mogul & Borg, 1991

- The TRUE process state that might be lost includes:
  - CPU information (branch prediction, buffers, speculative execution results)
  - Cache contents
  - TLB contents
  - Virtual memory pages resident in memory
  - Disk cache contents
“Thrashing”

- Can be caused by too-frequent context switches
  - More time spent re-loading context than actually executing work
  - Worst case depends on size of combined context working sets --
    the pessimistic case is that everything ends up on disk before task is restarted

- Can be caused by too-small cache sizes for a single task
  - Data set too large for cache -- gets 100% cache misses
  - Data set too large for physical memory -- gets frequent page faults (traditional definition of thrashing)
OS Structure Affects I-Cache Misses

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>USER</th>
<th>OS</th>
<th>I-cache (CPI\textsubscript{inst})</th>
<th>D-cache (CPI\textsubscript{data})</th>
<th>Write (CPI\textsubscript{write})</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS (Mach 3.0)</td>
<td>62%</td>
<td>38%</td>
<td>0.36</td>
<td>0.28</td>
<td>0.16</td>
</tr>
<tr>
<td>IBS (Ultrix 3.1)</td>
<td>76%</td>
<td>24%</td>
<td>0.19</td>
<td>0.30</td>
<td>0.11</td>
</tr>
<tr>
<td>SPECint92</td>
<td>97%</td>
<td>3%</td>
<td>0.05</td>
<td>0.08</td>
<td>0.06</td>
</tr>
<tr>
<td>SPECfp92</td>
<td>98%</td>
<td>2%</td>
<td>0.05</td>
<td>0.44</td>
<td>0.13</td>
</tr>
</tbody>
</table>

[Uhlig et al. 1995]

- Mach 3 is a micro-kernel system -- significantly poorer I-cache performance than Ultrix 3.1
  - Embedded RTOS may use microkernels to reduce required memory image
  - IBS is an instruction-intensive benchmarks (discussed later)

The Challenge of Software Productivity

- Programmer productivity growing slower than software size
  - Partly feature growth
  - Partly inefficient code
- It makes sense to expend hardware resources to save programming costs
  - Especially when the software company isn’t paying for the hardware!
- Disk capacity grows at 60% per year
  - But hard disks are still full...

Modern Software Practices Can Reduce Locality

- Many techniques trade hardware resources for programmer productivity...

- Complex, layered interfaces
  - GUI application program interfaces (instead of writing to video RAM)
  - Protocol stacks (instead of writing bits to network)
  - Binary-to-binary translation (to run “WinTel” executables and legacy code)

- Modularization & layering of traditional software
  - Code re-use; off-the-shelf software components
  - Design for change, information hiding

- Object-oriented methods
  - Organizing software by objects instead of by sequential actions disperses code

Potential Problems With Object-Oriented SW

- Poor spatial locality
  - Methods grouped by object type instead of by flow-of-control
  - Lack of sequentiality -- block size
    - Large blocks may not work well in L-cache if short routines are called (could be many unused bytes in fetched L block)
  - Lack of spatial locality -- TLB entries
    - If every method is in a different page, may need many TLB entries
  - Lack of spatial locality -- conflict misses/associativity
    - Fragmented code may not disperse evenly across the entire cache
    - Could get increase in conflict misses; might need higher degree of associativity

- Potential solutions:
  - Compiler in-lining at cost of increased program size
    - Linker in-lining for libraries?
    - Causes “code bloat”
  - Adjust hardware -- smaller blocks; more TLB entries; increased associativity
Non-SPEC Software & “Code Bloat”

- **I-cache misses increase significantly for some software (IBS benchmark):**
  - mpeg_play
  - jpeg_play
  - gs
  - verilog
  - gcc
  - sdet
  - nroff
  - groff

- **SPEC software isn’t necessarily representative of desktop software**
  - Data accesses also change as multimedia becomes prevalent

[Uhlig et al. 1995]

REVIEW
Review

- **SPECmarks may not be representative of system performance**
  - Operating system and other software has different locality characteristics
  - Multitasking can increase conflict misses by evicting cache blocks
  - Modern software practices may have poor locality

Key Concepts

- **Latency**
  - Latency penalty for context switching can be high

- **Bandwidth**
  - Bandwidth can help for storing and loading state
    - *e.g.*, Save/restore entire cache contents on task switch?

- **Concurrency/Replication**
  - Perhaps have two separate caches for OS & User states?

- **Balance**
  - Need to balance amount of system state for context switching against speedup from caching techniques