Announcements

Homework #2 & Reader #2 handed out today

Computer Architecture Lab (CALCM)
- Seminars
- Distinguished lectures
- Research
- Fun

Send mail to babak@cmu.edu

www.ece.cmu.edu/CALCM
Memory Systems

Basic caches
- introduction
- fundamental questions
- cache size, block size, associativity

Advanced caches

Main memory

Virtual memory

Motivation

CPU wants
- memory reference/inst x bytes-per-reference x IPC/cycle
- $1.2 \times 4 \times 1/2 \text{ns} = 2.4 \text{ GB/s}$

CPU can only go as fast as memory!
**Motivation for Hierarchy (†)**

Want storage:
- as fast as CPU
- as large as the application requires

Exploit reference locality in a memory hierarchy:
- Locality in time (temporal locality)
  - if a datum is recently referenced, it is likely to be referenced again soon
- Locality in space (spatial locality)
  - if a datum is recently referenced, neighboring data is likely to be referenced soon

---

**Memory Hierarchy**

Make common case fast:
- common: temporal & spatial locality
- fast: smaller more expensive memory

![Memory Hierarchy Diagram]
**Processor/Memory Boundaries**

<table>
<thead>
<tr>
<th>Processor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I-Unit</td>
</tr>
<tr>
<td></td>
<td>E-Unit</td>
</tr>
<tr>
<td></td>
<td>Regs</td>
</tr>
<tr>
<td>L1 I-Cache</td>
<td>I-TLB</td>
</tr>
<tr>
<td>L1 D-Cache</td>
<td>D-TLB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td></td>
</tr>
</tbody>
</table>

Main Memory (not to scale!)

---

**Cache**

Cache managed by hardware
- Keep recently accessed block
  - temporal locality
- Break memory into blocks (several bytes)
  - spatial locality
- Transfer data to/from cache in blocks
Cache (Cont.)

Put block in “block frame”
- state (e.g., valid)
- address tag
- data

[Diagram of address state data]

On memory access
- if incoming tag == stored tag then HIT
- else MISS
  - replace old block
  - get new (referenced) block from memory
  - put block in the appropriate block frame in cache
  - return appropriate word within block to processor
Cache Example

Memory words:
0x11c 0xe0e0e0e0e0
0x120 0xffffffff
0x124 0x00000001
0x128 0x00000007
0x12c 0x00000003
0x130 0xabababab

Cache Example (Cont.)

A 16-byte cache block frame:
☐ tag state data
☐ 0x?? Invalid ???

Id R1, 0x128
Is tag 0x120 in cache?
☐ 0x128 mod 16 = 0x128 & 0xffffffff
No, get block
☐ tag state data
☐ 0x129 Valid 0xffffffff, 0x1, 0x7, 0x3
Cache Example (Cont.)

Return 0x7 to CPU to put in R1
Id R2, 0x124
Is tag 0x120 in cache?
Yes, return 0x1 to CPU

Cache Performance

Assume
- Cache access time is equal to 1 cycle
- Cache miss ratio is 0.1
- Cache miss penalty is 20 cycles

Mean access time

\[ \text{Mean access time} = \text{Cache access time} + \text{miss ratio} \times \text{miss penalty} \]
\[ = 1 + 0.01 \times 20 = 1.2 \]

Typically
- level-1 is 16K-64K, level-2 is 512K-4M, memory is 128M-4G
- level-1 as fast of processor
- level-1 is 1/10000 capacity but contains 98% of references
Cache Circuitry

Terminology

block (line, page) — minimum unit that may be present
hit — block is found in the cache
miss — block is not found in the cache
miss ratio — fraction of references that miss
hit time — time to access the cache
miss penalty
  □ time to replace block in the cache + deliver to upper level
  □ access time — time to get first word
  □ transfer time — time for remaining words
**Memory Hierarchy Performance**

Time is always the ultimate measure
Indirect measures can be misleading
   - like MIPS, miss ratio can be misleading
Average access time is better
   - $t_{avg} = t_{hit} + \text{miss ratio} \times t_{miss}$
   - e.g., $t_{hit} = 1$, miss ratio = 5%, $t_{miss} = 20$
   - $t_{avg} = 2$

**Fundamental Questions about Caches**

Where can a block be placed? Block placement
How is a block found? Block identification
Which block is replaced on a miss? Block replacement
What happens on a write? Write strategy (skip for now)
What is kept? Cache type
Block Placement

Fully-associative — block goes in any frame
Direct-mapped — block goes in exactly one frame
Set-associative — block goes in exactly one set

Block Placement (Cont.)

Where does block 12 (1100) go?

Fully-associative    Set-associative    Direct-mapped
**Block Identification**

How to find the block:
- compare tags
- search in parallel to find lookup
- check valid bit

**Block Placement (Cont.)**

E.g., where do we find block 12?

```
Block  | Set/Block | Set
-------|-----------|-----
0      | 0 1 12    | 0 1
1      | 0 1       | 2
2      | 0         | 3
3      | 1         | 4
4      | 2         | 5
5      | 1         | 6
6      | 12        | 7
7      |           |
```

- Fully-associative
- Set-associative
- Direct-mapped
Block Replacement

Which block to replace on a miss?
Least recently used — LRU
  □ optimized for temporal locality, complicated (why?)
Random
  □ pseudo-random for testing, nearly as good as LRU, simpler
Not most recently used — NMRU
  □ track MRU, random select from others, good compromise
Optimal — Belady’s algorithm, replace block furthest in time
  □ How do you implement it?

Cache Type

Unified
  □ less costly, dynamic response, handles writes to instructions
Split I and D
  □ 2x bandwidth, place close to I/D ports
  □ can customize, poor-man’s associativity, no conflict between I/D
  □ self-modifying code can cause problems
Caches should be split for frequent simultaneous I & D access
  □ counter example is PowerPC 601
Unified cache

Physical path = 8 words wide
- 1 I-fetch per 4-8 instructions
- 1 ld/st per 3-5 instructions
- => few I/D access conflicts

Cache Type (Cont.)

I and D split cache
- \( I_{miss} \) is 5% and \( D_{miss} \) is 6%
- 75% references are instruction fetches
- \( t_{avg} = (1 + 0.05*10)*0.75 + (1 + 0.06*10)*0.25 = 1.5 \)

Unified cache
- \( t_{avg} = 1 + 0.04*10 = 1.4 \)?
- \( t_{avg} = 1.4 + \) cycles lost to interference
- What will cycles lost to interference be typically?
Mark Hill's Miss Classification (4 C's)

- compulsory — miss in infinite cache
  - first access to a block
- capacity — miss in fully-associative cache
  - misses occur because cache not large enough
- conflict — miss in set-associative or direct-mapped cache
  - misses occur because of mapping strategy
- coherence — miss in multiprocessor cache
  - misses occur because of sharing among procs

Fundamental Cache Parameters

- Cache size
- Block size
- Cache associativity
**Cache Size**

Cache size in the total data (not including tag) capacity
- bigger can exploit temporal locality better
- not ALWAYS better

Too large a cache
- smaller is faster => bigger is slower
- access time may degrade critical path

Too small a cache
- don’t exploit temporal locality well
- useful data permanently replaced

---

**Block Size**

Block size is the data that is both
- associated with an address tag + transferred from memory
- advanced caches allow different

Too small blocks
- don’t exploit spatial locality well
- have inordinate tag overhead

Too large blocks
- useless data transferred
- useful data permanently replaced — too few total # blocks
Associativity

Partition cache frames into
  □ equivalence classes of frames called sets

Typical values for associativity
  □ 1-way=direct-mapped, 2-, 4-, 8-way associative

Larger associativity
  □ lower miss rate, less variation among programs

Smaller associativity
  □ lower cost, faster hit time