Chapter 2: Memory Hierarchy Design

Introduction

Caches
  Review of basics
  Advanced methods

Main Memory

Virtual Memory
Memory Hierarchies: Key Principles

Make the common case fast

Common → Principle of locality

Fast → Smaller is faster
Principle of Locality

Temporal locality

Spatial locality

Examples:
Smaller is Faster

Registers are fastest memory
  Smallest and most expensive

Static RAMs are faster than DRAMs
  10X faster
  10X less dense

DRAMs are faster than disk
  Electrical, not mechanical
  Disk is cheaper (currently)
  Disk is nonvolatile
**Memory Hierarchy Terminology**

**Block**
- Minimum unit that may be present
- Usually fixed length

**Hit** – Block is found in upper level

**Miss** – Not found in upper level

**Miss ratio** – Fraction of references that miss

**Hit Time** – Time to access the upper level

**Miss Penalty**
- Time to replace block in upper level, plus the time to deliver the block to the CPU
  - Access time – Time to get first word
  - Transfer time – Time for remaining words
Memory Hierarchy Terminology

Memory Address

<table>
<thead>
<tr>
<th>Block-frame address</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101010101010101011</td>
<td>01010101</td>
</tr>
</tbody>
</table>

Block Names

- Cache: Line
- VM: Page
Memory Hierarchy Performance

Time is always the ultimate measure

Indirect measures can be misleading
  MIPS can be misleading
  **So can Miss ratio**

Average (effective) access time is better

\[ t_{\text{avg}} = \]

Example:

\[ t_{\text{hit}} = 1 \]
\[ t_{\text{miss}} = 20 \]

miss ratio = .05

\[ t_{\text{avg}} = \]

Effective access time is still an indirect measure
Example

Poor question:

Q: What is a reasonable miss ratio?
A: 1%, 2%, 5%, 10%, 20% ???

A better question

Q: What is a reasonable $t_{avg}$?
   (assume $t_{cache} = 1$ cycle, $t_{memory} = 20$ cycles)
A: 1.2, 1.5, 2.0 cycles

What's a reasonable $t_{avg}$?
Rearranging terms in

\[ t_{\text{avg}} = t_{\text{cache}} + \text{miss ratio} \times t_{\text{memory}} \]

to solve for miss ratios yields

\[ \text{miss} = \frac{(t_{\text{avg}} - t_{\text{cache}})}{t_{\text{memory}}} \]

Reasonable miss ratios (percent) - assume \( t_{\text{cache}} = 1 \)

<table>
<thead>
<tr>
<th>( t_{\text{memory}} ) (cycles)</th>
<th>( t_{\text{avg}} ) (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>10.0</td>
</tr>
<tr>
<td>20</td>
<td>1.0</td>
</tr>
<tr>
<td>200</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Proportional to acceptable \( t_{\text{avg}} \) degradation
Inversely proportional to \( t_{\text{memory}} \)
Basic Cache Questions

Block placement
  Where can a block be placed in the cache?

Block Identification
  How is a block found in the cache?

Block replacement
  Which block should be replaced on a miss?

Write strategy
  What happens on a write?

Cache Type
  What type of information is stored in the cache?
Block Placement

Fully Associative
  Block goes in any block frame

Directmapped
  Block goes in exactly one block frame
  \(( \text{Block frame } \# ) \mod ( \# \text{ of blocks} )\)

Set Associative
  Block goes in exactly one set
  \(( \text{Block frame } \# ) \mod ( \# \text{ of sets} )\)

Example: Consider cache with 8 blocks, where does block 12 go?
**Block Identification**

How to find the block?
- Tag comparisons
- Parallel search to speed lookup
- Check valid bit

Example: Where do we search for block 12?
**Block Replacement**

Which block to replace on a miss?

Least recently used (LRU)
- Optimize based on temporal locality
- Replace block unused for longest time
- State updates on non-MRU misses

Random
- Select victim at random
- Nearly as good as LRU, and easier

First in First out (FIFO)
- Replace block loaded first

Optimal
- ?
Write Policies

Writes are harder
  Reads done in parallel with tag compare; writes are not
  Thus, writes are often slower
  (but processor need not wait)

On hits, update memory?
  Yes  writethrough (storethrough)
  No   writeback (storein, copyback)

On misses, allocate cache block?
  Yes  writeallocate (usually used w/ writeback)
  No   nowriteallocate (usually used w/ writethrough)
Write Policies, cont.

WriteBack
Update memory only on block replacement
Dirty bits used so clean blocks can be replaced without updating memory
Traffic/Reference =
Traffic/Reference =
Less traffic for larger caches

WriteThrough
Update memory on each write
Write buffers can hide write latency (later)
Keeps memory up to date (almost)
Traffic/Reference =
Cache Type

Unified (mixed)
- Less costly
- Dynamic response
- Handles writes into I stream

Separate Instruction & Data (split, Harvard)
- 2x bandwidth
- Place closer to I and D ports
- Can customize
- Poorman's associativity
- No interlocks on simultaneous requests

Caches should be split if simultaneous instruction and data accesses are frequent (e.g., RISCs)
Cache Type Example

Consider building (a) 16K byte I & D caches, or (b) a 32K byte unified cache.
Let $t_{cache}$ is one cycle, $t_{memory}$ is 10 cycles.

(a) $I_{miss}$ is 5 %, $D_{miss}$ is 6 %, 75 % of references are instruction fetches.

$$t_{avg} =$$

(b) miss ratio is 4 %

$$t_{avg} =$$
A Miss Classification (3Cs or 4Cs)

Cache misses can be classified as:

*Compulsory* (a.k.a. cold start)

The first access to a block

*Capacity*

Misses that occur when a replaced block is rereferenced

*Conflict* (a.k.a. collision)

Misses that occur because blocks are discarded because of the setmapping strategy

*Coherence* (shared-memory multiprocessors)

Misses that occur because blocks are invalidated due to references by other processors
Fundamental Cache Parameters

Cache Size
   How large should the cache be?

Block Size
   What is the smallest unit represented in the cache?

Associativity
   How many entries must be searched for a given address?
Cache size is the total capacity of the cache

Bigger caches exploit temporal locality better than smaller caches

But are *not always* better

Why?
Block Size

Block (line) size is the data size that is both
(a) associated with an address tag, and
(b) transferred to/from memory
Advanced caches allow different (a) & (b)

Problem with too small blocks

Problem with large blocks
Block Size Example

Block size that minimizes $t_{\text{avg}}$ is often smaller than the block size that minimizes miss ratio!

Let the main memory take 8 cycles before delivering two words per cycle. Then:

$$t_{\text{memory}} = t_{\text{access}} + B \times t_{\text{transfer}} = 8 + B \times 1/2$$

where $B$ is block size in words

(a) block size 8 words with miss ratio 5 %

$$t_{\text{memory}} =$$

$$t_{\text{avg}} =$$

(b) block size 16 words with miss ratio 4 %

$$t_{\text{memory}} =$$

$$t_{\text{avg}} =$$
SetAssociativity

Partition cache block frames & memory blocks in equivalence classes (usually w/ bit selection)

Number of sets, \( s \), is the number of classes

Associativity (set size), \( n \), is the number of block frames per class

Number of block frames in the cache is \( s \times n \)

Cache Lookup (assuming read hit)

Select set

Associatively compare stored tags to incoming tag

Route data to processor
Associativity, cont.

Typical values for associativity

1 -- directmapped

n = 2, 4, 8, 16 -- nway setassociative

All blocks -- fullyassociative

Larger associativities

Lower miss ratios

Less variance

Intuitively satisfying

Smaller associativities

Lower cost

Faster access (hit) time (perhaps)
Associativity (Cont.)

Associativity that minimizes $t_{avg}$ can be smaller than associativity that minimizes miss ratio!

Consider DM & SA caches w/ same $t_{memory}$.

$$\Delta t_{cache} = t_{cache}(SA) - t_{cache}(DM) > 0$$

$$\Delta miss = miss(SA) - miss(DM) < 0$$

$t_{avg}(SA) < t_{avg}(DM)$ only if

$$t_{cache}(SA) + miss(SA) \times t_{memory} < t_{cache}(DM) + miss(DM) \times t_{memory}$$

$$\Delta t_{cache} + \Delta miss \times t_{memory} < 0$$

E.g.,

(a) Assuming $\Delta t_{cache} = 0 \Rightarrow$ SA better

(b) $\Delta miss = 1/2\%$, $t_{memory} = 20$ cycles $\Rightarrow$ $\Delta t_{cache} < 0.1$ cycle