Caches: What Every OS Designer Must Know

The Memory Wall

- \( \text{Performance} = \frac{\text{Word's Life}}{\text{Time}} \)

- Processor-Memory Performance Gap (grows 3%/year)

- 70% of CPU time

Caching

- Cache is fast (1-5 cycle access time) memory sitting between fast registers and slow RAM (15-100 cycles access time)
- Holds recently-used data or instructions to save memory accesses
- Matches slow RAM access time to CPU speed if hit rate is high (90%)
- Is hardware maintained and (mostly) transparent to software
- Sizes range from few KB to several MB
- Usually a hierarchy of caches (3-5 levels), on- and off-chip
- Good overview of implications of caches for operating systems: [Schimmel 94]

Cache Organization

- Data transfer unit between registers and L1 cache: ≤ 1 word (1-16B)
- Cache line is transfer unit between cache and RAM (or slower cache)
  - Typically 16-32 bytes, sometimes 128 bytes and more
- Line is also unit of storage allocation in cache
- Each line has associated control info:
  - valid bit
  - modified bit
  - tag
- Cache improves memory access by:
  - Absorbing most reads (increases bandwidth, reduces latency)
  - Making writes asynchronous (reduces latency)
  - Clustering reads and writes (reduces latency)

Cache Access

- Virtually indexed: looked up by virtual address
  - Operates concurrently with address translation
- Physically indexed: looked up by physical address
  - Requires result of address translation
The tag is used to distinguish lines of set... 
... consists of the address bits not used for indexing.

Address is hashed to produce index of line set.
Assosciative lookups of line within set
n lines per set; n-way n-way set-associative cache.
- Typically n = 1...5; some embedded processors use 32-64
- n = 1 is called direct mapped.
- n = No. lines is called fully associative (unusual for CPU caches)
Hashing must be simple (complex hardware is slow)
- use least-significant bits of address

Lower bits used to select cache line to be searched 

Locations mapping to cache set i are said to be of colour i

n-way associative cache can hold n lines of the same colour
Types of cache misses:
- Compulsory miss: data cannot be in cache (of infinite size)
  - first access (after flush)
- Capacity miss: all cache entries are in use by other data
- Conflict miss: set of the right colour is full
- misses that would not happen on fully-associative cache
- Coherence miss: miss forced by hardware coherence protocol
- multiprocessors

Note: Lookup hardware for many tags is large and slow: does not scale
Cache Replacement Policy

- Indexing (using address) points to specific line set
- On miss: all lines of set invalid, must replace assisting line
- Replacement strategy must be simple (hardware)
  - Least Recently Used (LRU)
  - Random
  - Least Clean

Cache Write Policy

- Treatment of store operations:
  - Write-back: Store updates data only
  - Write-through: Stores updates to cache and memory immediately
- Cache coherence issues
  - Memory is consistent with cache
  - Memory is always consistent with cache
  - Increased memory traffic
- On store to a line not present in cache, use:
  - Allocate: Allocate a new cache line to the data and store
  - Tag: Requires reading line into cache first
  - Reallocate: Store to memory and bypass cache

Cache Addressing Schemes

- For simplicity, discussion so far assumed cache sees only one kind of address: virtual or physical
- However, indexing and tagging can use different addresses
- There are four possible addressing schemes:
  - Virtually-indexed, virtually-tagged (VVT) cache
  - Physically-indexed, virtually-tagged (PVV) cache
  - Physically-indexed, physically-tagged (PVT) cache
  - Virtually-indexed, physically-tagged (VIP) cache
- PV caches can only make sense with complex and unusual MMU designs
- Not considered here any further

Virtually-Indexed, Virtually-Tagged Cache

- Also called virtual-addressed cache
  - Also (incorrectly) called:
  - virtual cache
  - virtual address cache
- Uses virtual addresses only
  - Can operate concurrently with
  - MMU lookup to determine access rights
  - Used on-chip

Virtually-Indexed, Physically-Tagged Cache

- Virtual address for accessing line
- Physical address for tagging
- Needs address translation completed for retrieving data
- Indexing concurrent with MMU, use MMU output for tag check
- Typically used on-chip

Physically-Indexed, Physically-Tagged Cache

- Only uses physical addresses
- Needs address translation completed before beginning access
- Typically used off-chip
- Notes:
  - Page offset is invariant under virtual-address translation
  - If index bits are "out" of offset, PP cache can be accessed without
  - Result of translation is
  - VP and PP cache become the same in this case
  - Fast and suitable for on-chip use
Address Mismatch Problem: Aliasing

- Page aliased in different address spaces
  - AS1, VA1 = 1, AS2, VA2 = 0
  - One alias gets modified
    - in write-back cache, other alias sees stale data
    - load-update problem

Address Mismatch Problem: Re-Mapping

- Unmap page with a dirty cache line
- Re-use (rename) frame for a different page (in same or different AS)
- Write to new page
  - without mismatch, new write will overwrite old (hits same cache line)
  - with mismatch, order can be reversed: "cache bomb"
Summary: PP Caches

- Slowest
  - requires result of address translation before lookup starts
- No synonym problem
- No homonym problem
- Easy to manage
- If small or highly associative (all index bits come from page offset) indexing can be in parallel with address translation
  - Potentially useful for L1 cache (used on Itanium)
- Cache can use bus snooping to receive/supply DMA data
- Usable as off-chip cache with any architecture
- For an in-depth coverage of caches see [Wiggins 03]
**Write Buffer**

- Store operations can take a long time to complete.
- Can avoid stalling the CPU by buffering writes.
- Write buffer is a FIFO queue of incomplete stores.
- It can also read intermediate values out of buffer.
- Service load of a value that is still in write buffer.
- Avoids unnecessary stalls of load operations.
- Impacts that memory contents are temporarily stale.

**Translation Lookaside Buffer (TLB)**

- TLB is a (V) cache for page-table entries.
- TLB can be:
  - Hardware loaded, transparent to OS, or
  - software loaded, maintained by OS.
- TLB can be:
  - split, instruction and data TLBs, or
  - unified.
- Modern high-performance architectures use a hierarchy of TLBs:
  - Top-level TLB is hardware-loaded from lower levels
  - Transparent to OS.

**Cache Hierarchy**

- Hierarchy of caches to balance memory accesses.
  - Small, fast, virtually indexed L1.
  - Large, slow, physically indexed L2-L5.
- Each level reduces and clusters traffic.
- L1 typically split into instruction and data caches.
- Requirement of peeling.
- Lower level tend to be unified.
- Chip multiprocessors (multicore) often share on-chip L2, L3.

**TLB Issues: Associativity**

- First TLB (VAX-11/780, (Clark, Emer 85)) was 2-way associative
- Most modern architectures have fully associative TLBs
- Exceptions:
  - i486 (4-way)
  - Pentium, P6 (4-way)
  - IBM RS/6000 (2-way)
- Reasons:
  - Modern architectures tend to support multiple page sizes (superpages)
  - Better utilizes TLB entries
  - TLB lookup done without knowing the page’s base address
  - Superpage TLBs are fully-associative.

**TLB Size (i-TLB + D-TLB)**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>TLB Size</th>
<th>Page Size</th>
<th>TLB Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAX 64K</td>
<td>32-256</td>
<td>512B</td>
<td>32-128KB</td>
</tr>
<tr>
<td>x86 32-64</td>
<td>64B+8MB</td>
<td>128-128+256KB</td>
<td></td>
</tr>
<tr>
<td>MIPS 96-128</td>
<td>64B+16MB</td>
<td>384KB-...</td>
<td></td>
</tr>
<tr>
<td>SPARC 64</td>
<td>64B+8MB</td>
<td>512KB-...</td>
<td></td>
</tr>
<tr>
<td>Alpha 32-128+128</td>
<td>64B+8MB</td>
<td>256KB-...</td>
<td></td>
</tr>
<tr>
<td>RS/6000 32+128</td>
<td>64B</td>
<td>128+512KB</td>
<td></td>
</tr>
<tr>
<td>Power4/55 128</td>
<td>64B+16MB</td>
<td>512KB-...</td>
<td></td>
</tr>
<tr>
<td>PA-8000 96+96</td>
<td>64B+64B</td>
<td>4GB</td>
<td></td>
</tr>
</tbody>
</table>

Not much growth in 20 years!
Traditional UNIX process has 2 regions of allocated virtual address space:
- low-end: text, data, heap
- high-end: stack

2-3 PT pages are sufficient to map most address spaces

Superpages can be used to extend TLB coverage

However, difficult to manage in the OS

Modern OS features:
- memory-mapped files
- dynamically loaded libraries
- mapping IPC (server-based systems, etc.)

This problem gets worse for 64-bit address spaces:
- bigger page tables

An in-depth study of such effects can be found in [Uhlig et al. 94]