Reducing Miss Rate

- Approaches 1 - 3:
  - larger block size
  - higher associativity
  - victim caches
- Technique 4 - Pseudo Associative Caches
  - also called column associative
  - Simple hack
    - start with a direct mapped cache
    - on a miss check another entry
    - usual method is to invert the high order index bit to get the next try
  - problem - fast hit and slow hit
    - may have the problem that you mostly need the slow hit
    - in this case it's better to swap the blocks
  - like victim caches - provides selective *on demand* associativity

Miss Reduction 5

Prefetching I & D

- Use hardware other than the cache to prefetch what you expect to need ahead of time
- Question is how to decide
  - AXP 21064 I-fetches 2 blocks on a miss
    - target block goes to the I-cache
    - next block goes to the instruction stream buffer (ISB)
    - if requested block is in the ISB then it moves to the Icache and next block only is promoted from the next lower level.
    - a single block ISB catches 15-25% of the misses
  - Catch-22
    - works well if it can utilize otherwise unused memory bandwidth
    - slows things down if interferes with demand miss memory traffic
  - tends to work well with I but not so well with D

An Interesting Opportunity

Consider

- Superscalar architecture
  - scoreboarded instructions
    - out of order start and finish
  - multiple memory ports
    - e.g. 2 D ports, Single I port
    - with banked memory system multiple ports are possible
    - however there is a pin cost that can't be ignored
  - pipelined memory system
    - SDRAM and RDRAM effectively extends the pipeline into the main memory
    - may return out of order due to hierarchy

Speculative prefetch option

- I-path
  - multiple stream buffers
    - each one looks for branches and jumps
  - always try to have all possible targets in the buffers
  - fetching multiple instructions per reference
    - implies that some available bandwidth is likely to the Icache
- D-path
  - Compiler must provide some help
    - since D-references are hard to predict
  - prefetch blocks become D-speculate blocks in the instruction reorder buffer
  - unused D-reference cycles get filled from D-speculate blocks
- All but the D-path ideas are in the PA-8000

Miss Reduction 6

compiler controlled prefetching

- ❑ Can load either registers or cache
  - unless it's R0 - register prefetch increases register pressure big time
- ❑ Fault Issues
  - can be either faulting or non-faulting
  - non-faulting turns into a NOP if a memory fault occurs on the attempt
  - Only makes sense if CPU can continue normally during the prefetch
    - like a pipelined memory system
    - also called *lockup free* or *non-blocking* cache just to confuse things
- ❑ Tends to work well for loops
  - complements unrolling and SW pipelining
  - relies on predictable stride characteristic

Miss Reduction 7

compiler optimizations

- ❑ Note:
  - compiler assigns I and D locations within a virtual page
  - since miss rates and penalties are key performance limits
    - must schedule things so they are reduced
- ❑ Lots of options
  - array merging
    - allocate arrays so that paired operands show up in same cache block
  - loop interchange
    - exchange inner and outer loop order to improve cache performance
  - loop fusion
    - for independent loops accessing the same data
    - fuse these loops into a single aggregate loop
  - blocking (most useful in capacity miss reduction) - do as much as possible on a sub-block before moving on

Reducing Miss Penalties

- ❑ Particularly important due to the widening CPU/Memory gap
  - in cycle relative terms miss penalties are going up
- ❑ Write buffer (big help with write-through C's)
  - no need for the CPU to wait on a write
  - hence buffer the write and let the CPU proceed
    - needs to be associative so it can respond to a read of a buffered value
  - reads are the more common case so prioritize read miss handling
    - associative write buffer helps
    - also for write back caches - do the read first
    - implies during the read - victimized block moves to a write-back buffer
    - then when the read is satisfied CPU can proceed
    - and then the write-back buffer can access the lower memory system

Other Miss Penalty Improvements

- ❑ Sub-block placement
  - large blocks tend to be a good idea
  - have separate valid tag for a sub-block
  - allocate block but only get sub-blocks as needed
  - reduces transfer time effect
- ❑ Early restart and critical word first
  - as soon as requested word arrives pass it to the CPU
    - finish up with the line fill while processing happens
  - start fetching the critical word
    - then wrap around to fill the rest of the line
- ❑ Non-blocking Caches
  - pipelined memory control with out of order return
  - allows multiple outstanding references
    - e.g. hit under n-miss capability (n=8 for the PA-8000)

Second Level Caches

- Probably the best miss-penalty reduction
- But it does hair things up a bit

$$\text{Average memory access time} = \text{Hit-Time}_{L1} + \text{Miss-Rate}_{L1} \times \text{Miss-Penalty}_{L1}$$

where

$$\text{Miss-Penalty}_{L1} = \text{Hit-Time}_{L2} + \text{Miss-Rate}_{L2} \times \text{Miss-Penalty}_{L2}$$

Advantages:

- little to be done for compulsory misses and the penalty goes up somewhat
- capacity misses in L1 end up with a significant penalty reduction since they likely will get supplied from L2
- conflict misses in L1 similarly will get supplied by L2 unless they also conflict there

Multi-level Cache Issues

- Policy variations ==> added complexity
  - small L1 means N-way set associative likely
  - larger L2 means direct mapped likely
    - L2 must be larger than L1 for it to make any sort of sense
  - write-(through x back) and write-(around x allocate)
    - possible at each level although often different
  - what makes sense? How about write-buffering?
  - block sizes - do they need to be the same?
- Multiple miss types
  - local miss rate - misses/accesses on a level basis
  - global miss rate - miss to main/CPU memory references
    - product of the local level miss rates
    - for 2 levels = Miss-Rate $_{L1}$ x Miss-Rate $_{L2}$

Design Issues for Multi-level Caches

- Consider cache consistent I/O
  - I/O subsystem and CPU both pound on the memory system
  - 2 writers means possible inconsistency
    - which the memory system needs to fix
  - extremely important for multiple processors
  - need to minimize stalls
    - allow both writers to proceed concurrently
    - Cost of everything goes up of course!!!!!

Multi-level Inclusion

inherent problem for multi-level caches

- Write-through L1 - and multi-bank write-back L2
  - accesses to different banks don't conflict
- CPU mostly runs out of L1, I/O runs to Main or L2
  - snooping is required for to maintain consistency
  - snooping may induce an extra cycle in the return
    - since 2 ported memory cells are unlikely in the cache
- L2 tags indicate whether value is also in L1
  - invalidate or update option if it is
  - invalidate is the common choice since it is less intrusive

Reducing Hit Times

critical influence on cycle-time or CPI

- Keep the L1 small and simple
  - small is always faster and maybe even on chip
    - interesting compromise is to keep the tags on chip and the block data off chip
    - possibly over specialized transmission lines (e.g. PA-8000)
  - DEC Alpha - 2 levels of on chip cache
    - small Harvard L0 caches so they can keep up with the processor
    - unified L1 also on chip
- Direct mapped
  - trading hit time for increased miss-rate
    - small direct mapped misses more often than small associative caches
    - but simpler structure makes the hit go faster

Virtual Addressed Caches

- Parallel rather than sequential access
  - physical addressed caches
    - access the TLB to generate the physical address
    - then do the cache access
- Avoid address translation during cache index
  - implies virtual addressed cache
  - address translation proceeds in parallel with cache index
    - if translation indicates that the page is not mapped - then the result of the index is not a hit
    - or if a protection violation occurs - then an exception results
    - all is well when neither happen
- Too good to be true?
  - ideas for problem areas?

Problems with Virtual Caches

- Task switch causes virtual address to refer to different physical address
  - hence cache must be flushed
    - creating huge task switch overhead
    - also creates huge compulsory miss rates for new process (may have happened anyway)
  - OR use PID's as part of the tag to aid discrimination
- Synonyms
  - OS and User code have different virtual addresses which map to same physical address (facilitates copy-free sharing)
  - anti-aliasing mechanisms guarantee single copy
  - SW can help - e.g. SUN's version of UNIX
    - page coloring - aliases must have same low-order 18 bits

Today's Virtual Caches

- DEC and HP
  - both use a common compromise
    - Alpha
    - HP starting with the 7200
  - virtually addressed physically tagged
- Basic idea
  - address tag = physical address tag
  - access the TLB and cache in parallel
    - data return from the cache yields the address tag
    - compare with the result from the TLB's translation
- HP wrinkle
  - to avoid much of the flushing
    - use address space identifiers and a virtual tag as part of the address tag
    - effectively a form of segmentation

Faster Write Hits

- ❑ Write hits are slower
  - write is destructive so can't be done speculatively
  - requires tag match completion
- ❑ Hence pipeline the writes (e.g. AXP)
  - stage 1: do the tag check
  - stage 2: finish the write
- ❑ Result
  - looks like a write happens on every cycle
  - cycle-time can stay short since real write is spread over several cycles (here it was 2 but it easily can be more in practice)
  - mostly works since CPU is not dependent on data from a write
    - spot any problems if read and write ordering is not preserved by the memory system?

Cache Improvement Summary

Technique	Miss Rate	Miss Penalty	Hit Time	HW Complexity	Comments
Larger Block Size	win	lose		easy	trivial engineering effort
Higher Associativity	win		lose	1	associative match isn't free
Victim Caches	win			2	e.g. HP 7200
Pseudo-Associative	win			2	Used in L2 of MIPS R10000
HW Prefetch of I and D	win			2	D fetch hard to do
Compiler controlled prefetch	win			3	Needs non-blocking cache
Compiler cache scheduling (blocking, ...)	win			0	Too bad it's hard to do for all applications - loop focus for now

Summary Cont'd

Technique	Miss Rate	Miss Penalty	Hit Time	HW Complexity	Comments
Prioritizing Read Misses		win		1	write buffer - simple
Subblock placement		win		1	good at reducing tag overhead
Early restart + critical word first		win		2	MIPS R10K, IBM 620
Nonblocking Caches		win		3	MIPS R10K, AXP 21064
Second Level Caches		big win		2	big additional cost for all that SRAM
Small Simple Caches	lose		win	0	trivial so it's widely used
Avoid translation effects			win	2	AXP 21064 several PA-whatevers
Pipelining Writes			win	1	AXP 21064, PA-8000, UltraSPARC

Main Memory

bottom rung of the memory hierarchy

- ❑ 3 important issues
  - capacity
    - Bell's law - 1 MB per MIP needed for system balance
    - real key here is to avoid those costly page faults
  - latency
    - how long does it take to get the data back
    - by addressing big chunks - like an entire cache block (amortize the latency)
    - critical to cache performance when the miss is to main
  - bandwidth
    - affects the time it takes to transfer the block
    - a key issue when DMA service from an I/O device is considered
    - also a key issue with the very large block sizes in lower level caches

Memory T echnology

- SRAM's and DRAM' s are different
  - DRAM density is 16x SRAM at same feature size
    - hence cost is muc h c heaper since it trac ks area
  - SRAM speed is 8 to 16x that of DRAM
- Main memory toda y means DRAM
  - Multiplexed address lines - RAS then C AS
    - 2 dimensional address - rows go to a buffer
    - subsequent C AS selects subrow
  - Refresh needed every few milliseconds - another parasite
- Where are we toda y
  - 256 Mbit - 1 Gb c hips
    - RAS access time between 40 and 65ns
    - C AS access 10 ns
    - cyc le time - 90 ns (separation between subsequent accesses)

Example

- Consider
  - 1 cyc le to send the address
  - 6 cyc les per word of access
  - 1 cyc le to transmit the data
- Hence if main memory is organized by word
  - 8 cyc les for every word get' s spent
- Given a cac he line of 4 words
  - 32 cyc les is the miss penalty

Clearly we need a better organizational model

Memory Organization Improvements

- Wider memory
  - make a word of main memory look like a cac he line
    - easy to do - need 4 words - just parallel 4 c hips and decode the c hip selects
  - problem is the cost of the wider bus
    - especially true if this contributes to the pin count on the CPU
- Interlea ved or phased memory
  - keep the memory bus the same but make it run faster
  - still n-banks
    - but split phase delivery makes the bus bandwidth go up by the interlea ve factor .
- Both are optimized for sequential memory accesses
  - e .g. capitalizes on spatial locality principle just like cac hes do

DRAM Specific Interleaving Options

- Since 1 MB DRAM da ys other options exist
- Row is usually the square root of the DRAM size
  - 1K for 1M and so on
- Nibble Mode - RAS then C AS, next, next, etc .
- Page Mode - P age Mode - RAS then any C AS
  - hence after C AS the row is random addressable
- Static Column (SCRAM)
  - similar to page mode
  - no need to do a C AS - can also supply C AS next quic kly
- Sync hronous DRAM' s and RAMBUS
  - internally pipelined - lots of activity here presently

Virtual Memory

- ❑ Permits applications to grow bigger than main memory size
- ❑ Helps with multiple process management
  - each process gets its own chunk of memory
  - permits protection of 1 process' chunk from another
  - mapping of multiple chunks onto shared physical memory
  - mapping also facilitates relocation
    - Application and CPU run in virtual space
    - Mapping onto physical space is invisible to the application
- ❑ Management applies between main memory and secondary (disk) hierarchy levels
  - Miss becomes a *page* or *address fault*
  - Block becomes a *page* or *segment*

Typical Page Parameters

Parameter	Value
Page Size	4KB - 64KB
L1 Cache Hit Time	1 - 2 clock cycles
Virtual Hit (e.g. in Main Memory)	40-100 clock cycles
Miss Penalty - all the way to disk	700K - 6M clock cycles
Disk access time	500K - 4M cycles
Page Transfer time	200K - 2M clock cycles
Page Fault Rate	.00001% - .001%
Main Memory Size	4MB - 4 GB

- ❑ It's a lot like what happens in Cache
  - but all the numbers are even worse
    - with the exception of the miss rate

Cache vs. VM Differences

- ❑ Replacement
  - cache miss handled by hardware
  - page fault usually handled by the OS
    - OK - since fault penalty is so horrific
    - hence some strategy of what to replace makes sense
- ❑ Addresses
  - VM space is determined by the address size of the CPU
  - cache size is independent of the CPU address size
- ❑ Lower level memory
  - for caches - the main memory is not shared by something else
  - for VM - most of the disk contains the file system
    - file system addressed differently - usually in I/O space
    - the VM lower level is usually called *SWAP* space

2 VM Styles - Paged or Segmented?

- ❑ Pages are fixed size blocks
- ❑ Segments vary from 1 byte to 2**32

Aspect	Page	Segment
Words/Address	One - contains page and offset	Two - possible large max-size hence need Seg and offset address words
Programmer visible	No	Sometimes yes
Replacement	Trivial - due to fixed size	Hard - need to find contiguous space ==> GC necessary or wasted memory
Memory Inefficiency	Internal fragmentation - wasted part of a page	External fragmentation - due to variable size blocks
Disk Efficiency	Yes - adjust page size to balance access and transfer time	Not always - segment size varies