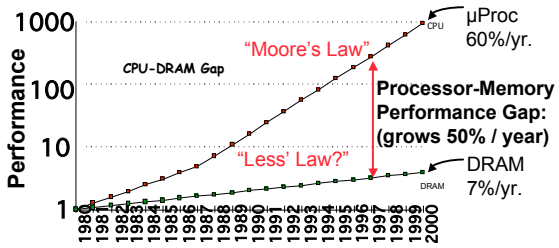
Cache Design

Chapter 5

Roadmap of the chapter

- Basics of Caches
- Reducing Miss penalty, hit time, Miss Rate
- Memory Systems Organization
- Virtual Memory
- Case Studies – Alpha 21264, Sony PlayStation, SunFire

Who Cares About the Memory Hierarchy?



- 1980: no cache in µproc; 1995 2-level cache on chip (1989 first Intel µproc with a cache on chip)

Generations of Microprocessors

- Time of a full cache miss in instructions executed:

1st Alpha:	340 ns/5.0 ns =	68 clks x 2 or		136
2nd Alpha:	266 ns/3.3 ns =	80 clks x 4 or		320
3rd Alpha:	180 ns/1.7 ns =	108 clks x 6 or		648

- 1/2X latency x 3X clock rate x 3X Instr/clock ⇒ -5X

Processor-Memory Performance Gap "Tax"

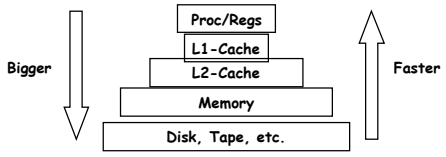
Processor	% Area (-cost)	%Transistors (-power)
• Alpha 21164	37%	77%
• StrongArm SA110	61%	94%
• Pentium Pro	64%	88%

- 2 dies per package: Proc/I$/D$ + L2$

• Caches have no "inherent value", only try to close performance gap

What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spatial and temporal locality
- In computer architecture, almost everything is a cache!
  - Registers "a cache" on variables – software managed
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information?



Traditional Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? *(Block placement)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level? *(Block identification)*
  - Tag/Block
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

What are all the aspects of cache organization that impact performance?

- Cache Parameters – total size, block size, associativity
- Hit time
- Miss Rate
- Miss Penalty
- Bandwidth of the next level of the memory hierarchy

Review: Cache performance

- Miss-oriented Approach to Memory Access:

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

  - $CPI_{Execution}$ includes ALU and Memory instructions

- Separating out Memory component entirely
  - AMAT = Average Memory Access Time
  - CPI_{ALUOps} does not include memory instructions

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$AMAT = HitTime + MissRate \times MissPenalty$$
$$= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) +$$
$$(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$$

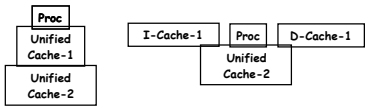
Impact on Performance

- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- CPI = ideal CPI + average stalls per instruction
- 1.1(cycles/ins) +
  [0.30 (DataMops/ins)
  x 0.10 (miss/DataMop) x 50 (cycle/miss)] +
  [1 (InstMop/ins)
  x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1

- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Unified vs Split Caches

- Unified vs Separate I&D



- Example:
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
- Which is better? (ignore L2 cache)
  - Assume 33% data ops ⇒ 75% accesses from instructions (1.0/1.33)
  - hit time=1, miss time=50
  - Note that *data* hit has 1 stall for unified cache (only one port)

$AMAT_{Harvard}$=75%x(1+0.64%x50)+25%x(1+6.47%x50) = 2.05
$AMAT_{Unified}$=75%x(1+1.99%x50)+25%x(!+1.99%x50)= 2.24

How to Improve Cache Performance?

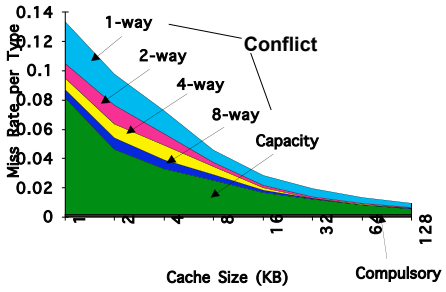
$$AMAT = HitTime + MissRate \times MissPenalty$$

1. Reduce the miss rate,
2. Reduce the miss penalty
3. Reduce the time to hit in the cache.

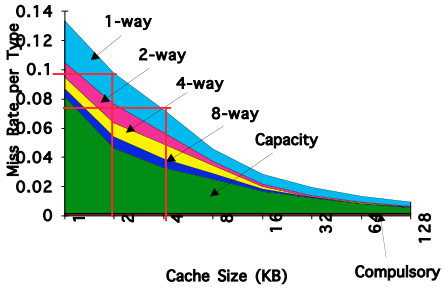
Where do misses come from?

- Classifying Misses: 3 Cs
  - *Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. Also called *cold start misses* or *first reference misses*.
    (Misses in even an Infinite Cache)
  - *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
  - *Conflict*—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called *collision misses* or *interference misses*.
    4th "C":
  - *Coherence* - Misses caused by cache coherence.

3Cs Absolute Miss Rate (SPEC92)



Cache Size



- Old rule of thumb: 2x size => 25% cut in miss rate
- What does it reduce?

Cache Organization

- Assume total cache size not changed:
- What happens if:

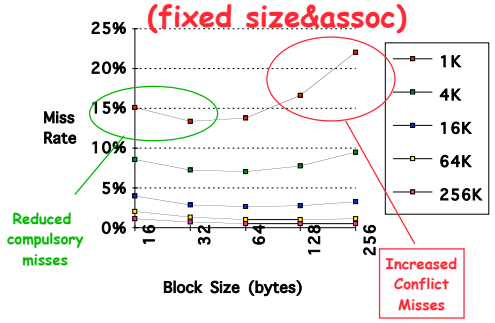
1) Change Block Size:

2) Change Associativity:

3) Change Compiler:

Which of 3Cs is obviously affected?

Larger Block Size (fixed size&assoc)



What else drives up block size?

Associativity vs Cycle Time

- Beware: Execution time is only final measure!
- Why is cycle time tied to hit time?
- Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%
  - suggested big and dumb caches

Example: Avg. Memory Access Time vs. Miss Rate

- Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

Cache Size (KB)	Associativity 1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	2.01
2	1.98	1.86	1.76	1.68
4	1.72	1.67	1.61	1.53
8	1.46	1.48	1.47	1.43
16	1.29	1.32	1.32	1.32
32	1.20	1.24	1.25	1.27
64	1.14	1.20	1.21	1.23
128	1.10	1.17	1.18	1.20

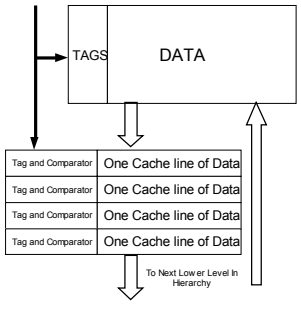
CCT=clock Cycle time

Hit Time of Dm = 1 CCT

(Red means A.M.A.T. not improved by more associativity)

Fast Hit Time + Low Conflict => Victim Cache

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



Reducing Misses via "Pseudo-Associativity"

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)

- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor (L2)
  - Used in MIPS R1000 L2 cache, similar in UltraSPARC

Way Prediction

- How to combine the low miss rate of higher associativity without paying for the increase in hit time?
- 2 way SA cache – have a bit that predicts which set the next block is more likely to be found in, so that you avoid the delay of one large MuX
- Used in alpha 21264
- Useful for power minimization – don't wake-up all the blocks – good for embedded systems.

Reducing Misses by Hardware Prefetching of Instructions & Data

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
  - On miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on having extra memory bandwidth that can be used without penalty

Reducing Misses by Software Prefetching Data

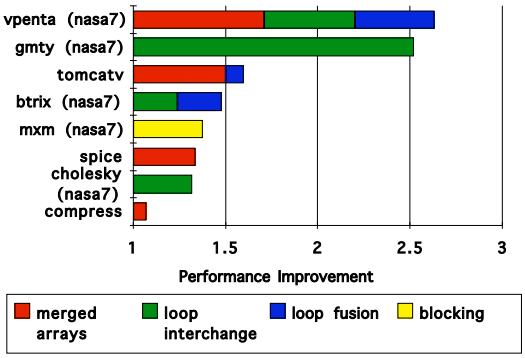
- Why not let the compiler prefetch data – after all it knows the whole program flow? Eg: loops
- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    - » Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    - » Can be incorrect.
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

Reducing Misses by Compiler Optimization

(Favorite Technique of HW Designers)

- Code and data accesses can be rearranged by the compiler without affecting the correctness
- McFarling (1989) showed 50% reduction in instruction misses and 75% reduction in 8KB cache. How?
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
  - Aligning Basic Blocks
- Data Access Reordering to improve spatial/temporal locality
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows

Summary of Compiler Optimizations to Reduce Cache Misses (by hand)



Summary: Miss Rate Reduction

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \text{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

- 3 Cs: Compulsory, Capacity, Conflict
  0. Larger cache
  1. Reduce Misses via Larger Block Size
  2. Reduce Misses via Higher Associativity
  3. Reducing Misses via Victim Cache
  4. Reducing Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Misses by Compiler Optimizations

Improving Cache Performance

1. Reduce the miss rate,
2. *Reduce the miss penalty,* or
3. Reduce the time to hit in the cache.

Write Policy: Write-Through vs Write-Back

- Write-through: all writes update cache and underlying memory/cache
  - Can always discard cached data - most up-to-date data is in memory
  - Cache control bit: only a *valid* bit
- Write-back: all writes simply update cache
  - Can't just discard cached data - may have to write it back to memory
  - Cache control bits: both *valid* and *dirty* bits
- Other Advantages:
  - Write-through:
    - » memory (or other processors) always have latest data
    - » Simpler management of cache
  - Write-back:
    - » much lower bandwidth, since data often overwritten multiple times
    - » Better tolerance to long-latency memory?

Write Policy 2: Write Allocate vs Non-Allocate (What happens on write-miss)

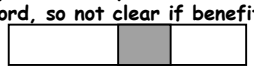
- Write allocate: allocate new cache line in cache
  - Usually means that you have to do a "read miss" to fill in rest of the cache-line!
  - Alternative: per/word valid bits
- Write non-allocate (or "write-around"):
  - Simply send write data through to underlying memory/cache - don't allocate new cache line!

1. Reducing Miss Penalty: Read Priority over Write on Miss

- Write-through w/ write buffers => RAW conflicts with main memory reads on cache misses
  - If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
  - Check write buffer contents before read; if no conflicts, let the memory access continue
- Write-back write buffer to hold displaced blocks
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read
- Merging Writes in a Write Buffer

2. Reduce Miss Penalty: Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - *Early restart*—As soon as the requested word of the block ar rives, send it to the CPU and let the CPU continue execution
  - *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*
- Generally useful only in large blocks,
- Spatial locality => tend to want next sequential word, so not clear if benefit by early restart



3. Reduce Miss Penalty: Non-blocking Caches to reduce stalls on misses

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires muliple memory banks (otherwise cannot support)
  - Penium Pro allows 4 outstanding memory misses

4: Add a second-level cache

- L2 Equations

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$

  $Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} + Miss\ Penalty_{L2})$

- Definitions:
  - *Local miss rate*— misses in this cache divided by the total number of memory accesses *to this cache* ($Miss\ rate_{L2}$)
  - *Global miss rate*—misses in this cache divided by the total number of memory accesses *generated by the CPU*
  - Global Miss Rate is what matters

An Example

- Global Miss rate for L2 = Miss Rate L1 * Miss Rate L2

Suppose 1000 mem reference

40 miss in L1 and 20 miss in L2 What are the local and global miss rates?

Miss Rate L1 = 40/1000 = 0.04 = 4%

Miss Rate L2 = 20/40 = 0.5 = 50%

Global Miss Rate of L2 = 20/1000 = 2%

What is AMAT assuming MissPenalty L2 = 100 and Hit = 1

Therefore, AMAT with two level caches =

Hit Time L1 + Miss Rate L1 *(HitL2+ MissRateL2*MissPenaltyL2)

= 1 + 0.04(1+0.5*100) = 3.4 cycles

Reducing Misses: Which apply to L2 Cache?

- Reducing Miss Rate
  1. Reduce Misses via Larger Block Size
  2. Reduce Conflict Misses via Higher Associativity
  3. Reducing Conflict Misses via Victim Cache
  4. Reducing Conflict Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Capacity/Conf. Misses by Compiler Optimizations

Multilevel Cache Optimization – A Challenge?

- What are the optimal parameters for L1 and L2 w.r.t block size? Associativity? Size?
- Eg: consider L1 block Size vs L2 Block Size

L1BS << L2BS – may increase L1 miss rate? How?

Should L2 include everything that L1 has?

Or only those that L1 does not have? – exclusive cache?

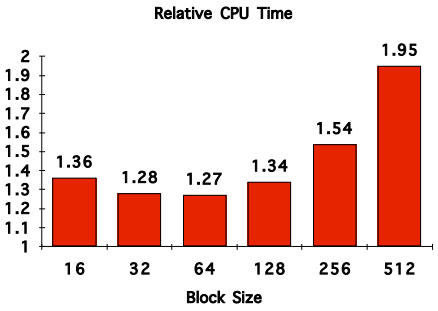
Why waste valuable real-estate with duplicates?

(athlon has two 64kb L1 caches and only 256L2 cache)

Now, L1 and L2 are on-chip, so some trade-offs are different?

Power is still an issue.

L2 cache block size & A.M.A.T.



- 32KB L1, 8 byte path to memory

Reducing Miss Penalty Summary

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times \text{Miss rate} \times Miss\ penalty \right) \times Clock\ cycle\ time$$

- Four techniques
  - Read priority over write on miss
  - Early Restart and Critical Word First on miss
  - Non-blocking Caches (Hit under Miss, Miss under Miss)
  - Second Level Cache
- Can be applied recursively to Multilevel Caches
  - Danger is that time to DRAM will grow with multiple levels in between
  - First attempts at L2 caches can make things worse, since increased worst case is worse

1. Fast Hit times via Small and Simple Caches

- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache?
  - Small data cache and clock rate
- Direct Mapped, on chip

3: Fast Hits by pipelining Cache Case Study: MIPS R4000

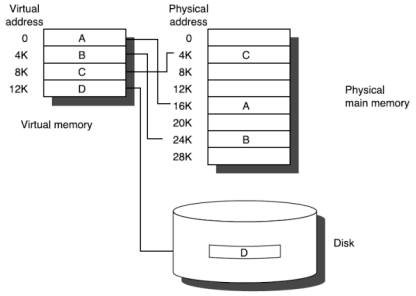
- 8 Stage Pipeline:
  - IF–first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
  - IS–second half of access to instruction cache.
  - RF–instruction decode and register fetch, hazard checking and also instruction cache hit detection.
  - EX–execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.
  - DF–data fetch, first half of access to data cache.
  - DS–second half of access to data cache.
  - TC–tag check, determine whether the data cache access hit.
  - WB–write back for loads and register-register operations.
- What is impact on Load delay?
  - Need 2 instructions between a load and its use!

Virtual Memory

- Permits larger than main memory data sets
- 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
    - applications 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*

Contiguous Virtual Address Space

may map anywhere



Typical Page Parameters

Parameter	L1 Cache	Virtual Memory
Block Size	16 -128 bytes	4KB - 64KB
Hit Time	1-3 cycles	50 - 150 (to main)
Miss Penalty	8-150 cycles	1M to 10M cycles
Access Time	6 - 130 cycles	800K - 8M cycles
Transfer time	2 -20 cycles	200K - 2M clock cycles
Miss Rate	.1 - 10%	.00001% - .001%
Address Mapping	25-45 bits physical to 14-20 bit cache address	32-64 bit virtual address to 24-45 bit physical address

- 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

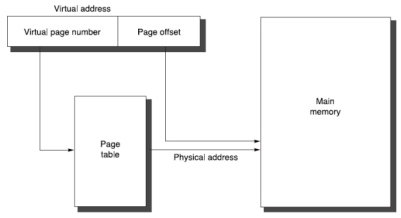
- Replacement
  - cache miss handled by hardware
  - page fault usually handled by the OS
    - OK - since fault penalty is so horrific
    - hence some strategy of what to replace makes sense
- Addresses
  - VM space is determined by the address size of the CPU
  - cache size is independent of the CPU 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

Page Table Address Mapping



Normal Page Tables

- Size
  - number of entries = number of virtual pages
- Role
  - translate VPN to PPN
    - permits ease of page relocation
  - + hold presence bit to indicate if the page is resident
    - and permissions of what types of accesses are allowed
    - privileged, read-only, etc.
- Potential problem
  - consider a 32 bit virtual address and 4K pages
    - 4 GB / 4 KB = 4 MB required for just the page table
  - YIKES
    - either have to have smaller addresses, bigger pages, OR ??
  - problem gets much worse in modern 64-bit machines

Inverted Page Tables

similar to a set associative mechanism

- Idea
  - make the page table reflect the # of physical pages (not virtual)
- Use a hash mechanism
  - virtual page number ==> HPN index into inverted page table
- Compare virtual page number with the tag to make sure it is the one you want
  - if yes
    - check to see that it is in memory - OK if yes - if not page fault
  - if not - miss
    - go to full page table on disk to get new entry
    - implies 2 disk accesses in the worst case
    - trades increased worst case penalty for decrease in capacity induced miss rate since there is now more room for real pages with smaller page table

Back to the 4Q's for VM

- Block Replacement
  - LRU is the best
    - so use it to minimize the impact of the horrific miss penalty
  - however true LRU is a bit pricey - so
    - page table contains a use tag
    - on access the use tag is set
    - OS checks them every so often - records what it sees in a data structure - then clears them all
    - on a miss the OS decides who has been used the least and nukes that one
  - strategy
    - spend a few OS cycles to reduce the miss rate
    - since the miss penalty is huge

Page Size?

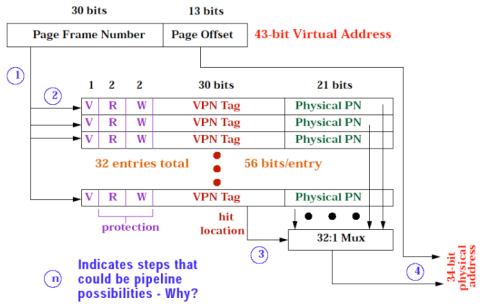
An architectural choice

- Large pages are good:
  - reduces page table size
  - amortizes the long disk access
  - if spatial locality is good then hit rate will improve
- Large pages are bad:
  - more internal fragmentation
    - if everything is random each structure's last page is only half full
    - half of bigger is still bigger
    - if there are 3 structures per process: text, heap, and control stack
    - then 1.5 pages are wasted for each process
  - process start up time takes longer
    - since at least 1 page of each type is required prior to start
    - transfer time penalty aspect is higher

Address Translation

- Page tables are large and paged themselves in some systems
  - double horrific page faults are now possible - BARF!
- If locality applies then cache the references
  - TLB = translation look-aside buffer
    - ENTRY = portion of virtual addr, physical page frame #, protection field, use bit, dirty bit.
    - OS changes entry - conflict issues vary with translation scheme
- TLB and VM conflict somewhat
  - TLB must be checked before the cache access can hit
  - can partially be done in parallel
    - e.g. virtually accessed caches
  - result is cache access times may get stretched a bit
    - this stretch can be hidden by speculation

AXP 21064 TLB



Protection

- Multiprogramming forces us to worry about it
  - think about what happens on your workstation
    - it would be a bummer if your program clobbered the window manager data structures
- Hence lots of processes
  - hence task switch overhead
  - HW must provide savable state
  - OS must promise to save and restore properly
  - most machines task switch every few milliseconds
  - a task switch typically takes several microseconds
  - also implies inter-process communication
    - which implies OS intervention
    - which implies a task switch
    - which implies less of the duty cycle gets spent on the application

Protection Options

- Simplest - base and bound
  - 2 registers - check each address falls between the values
    - these registers must be changed by the OS but not the app
  - ==> need for 2 modes: regular & privileged
    - hence need to privilege-trap and provide mode switch ability
- VM provides another option
  - check as part of the VA --> PA translation process
    - the protection bits reside in the page table & TLB

The Whole Hierarchy

some typical choices

Characteristic	TLB	L1 Cache	L2 Cache	VM (page)
Block Size	4-8 bytes = 1 PTE	4-32 Bytes	32-256 bytes	4K - 16K bytes
Hit Time	1 cycle	1-2 cycles	6-15 cycles	10-100 cycles
Miss Penalty	10-30 cycles	8-66 cycles	30-200 cycles	700K - 6M cycles
Local Miss Rate	.1 - 2%	.5 - 20%	15 - 13%	.00001 - .001%
Size	32 - 8KB	1-128 KB	256KB - 16MB	16MB - 8GB
Backing Store	L1 cache SRAM	L2 cache SRAM	SDRAM or RDRAM	Disk
Q1: block placement	full or set associative	direct mapped or set associative	direct mapped or set associative	fully associative via TLB
Q2: block ID	tag/block	tag/block	tag/block	Page Table
Q3: block replace	Random (but not last?)	NA if direct mapped - random if SA	random if SA	LFU
Q4: write strategy	Flush on PTE write	through or back	write-back	write-back

Alpha 21264 Memory Hierarchy

- 21264 characteristics
  - out of order execution
    - fetch 4 instructions/cycle
    - executes up to 6 instructions/cycle
  - virtual address structure always shows room for growth
    - 48 or 43 bit Vaddr's are specified - 43 is currently in use
    - 44 or 41 bit physical address space (48v==>44p, 43v==>41p)
    - page = 8KB, 64KB, 512KB, or 4MB

Power On

- I$ loaded serially from an external PROM
  - 16K instructions (64KB)
    - executed in PAL (priv. arch. library) mode
    - exceptions are disabled and hence no TLB miss or violations
    - handles future TLB misses as well since it is TLB independen
  - used for system initialization
    - loads the kernel and TLB entries
    - when OS is ready it sets the PC to a seg0 address to begin executing a user level process
- Memory Hierarchy
  - Split I & D TLB's - each with 128 fully associative entries
  - Split I and D L1 caches
    - D$ has a victim buffer which also serves as a write buffer
    - I$ has a prefetcher
  - BSB to unified L2$ offchip; FSB to the main memory array

Instruction Cache

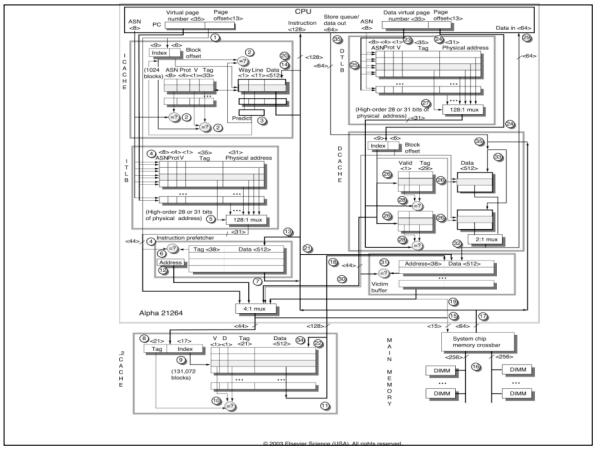
- V addressed and V tagged
  - 8 bit ASN used instead of full PID
  - translation needed only on an Imiss
  - 64 byte line = 4 bundles of 4 instructions (16B)
- 2 way SA with way prediction
  - first hit times like direct mapped but miss rate like 2-way SA
  - 1 bit way predict appended to index
  - cache holds an 11 bit next line prediction and a 1 bit next way prediction
  - predictor is loaded with next sequential group on a I$miss and with whatever the branch predictor says on a branch
  - hence coupled line and way prediction
  - PC (vaddr) checked against ASN and Vtag to confirm the hit
- Imiss
  - check prefetcher and TLB
  - if prefetch hit then intern into L1 and deliver the instruction bundle to the decoder
  - if TLB hit and prefetch miss then access L2

I miss Penalties

- External L2
  - set up for many configurations
    - data here is 667 MHz ES40 server - DDR DIMM's clocked at 222 MHz
  - 15 cycles to access L2
- Other
  - prefetcher
    - blindly gets next 64B I$ line
    - but won't cross a page boundary
  - miss to main
    - 130 cycles
    - critical instruction first return
    - while CPU continues the rest of the 64 byte line comes in 8 bytes per clock
  - write policy
    - none - it's an I$ and unlike the x86 writes to instruction pages aren't allowed

Data Cache

- Basics = 4Q's
  - 2-way set associative
  - virtually indexed and physically tagged
  - hit under 8 misses
  - write-back and LRU
  - on a dirty write miss
    - victim block is moved to the victim buffer (size = 8 lines) while new line is moved in
    - write back happens when it can
  - on an L2 miss
    - block goes to both the L2 and the victim buffer
    - hence vicim buffer combines: victim cache, write buffer, and assist cache duties
  - ECC
    - hence read-modify-write is required on a cache write



Sony PS2 Emotion Engine

- Games require a different memory bias
  - continuous streams dominate
    - high bandwidth is a necessity
    - low temporal locality (data is often touch once)
  - small caches seem to cover the bulk of the miss rate
- Architectural inversion
  - the main processor is a special purpose parallel processor
    - SIMD with vector function units
  - the IO processor is a simple 34MHz MIPS core
    - basically there to export some standard I/O interfaces
    - e.g. USB and 1394 (a.k.a firewire (apple), i-link (sony))
    - 1394 supports *isochronic* data transport

Emotion Memory

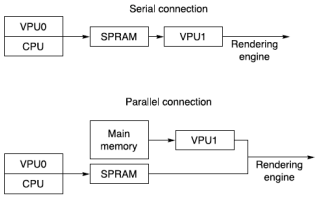
- Bandwidth
  - 10 DMA channels ==> 10 parallel data streams
    - effectively 10 x 16 bits ==> 160 bit wide memory bus
    - running at 400 MHz ==> 8 GB/sec
- Processing
  - graphics processing via SIMD and vector units in EE
    - CPU is a 300MHz MIPS III
    - extended to support 128 bit SIMD instructions
  - display lists passed to Synthesizer which does the rendering
    - 8 MB eDRAM/VRAM with 1024 bit channels in each direction at 150 MHz
    - 16 parallel pixel processors
  - perspective
    - EE: 13.5M T's in 225 mm^2 in .25 micron CMOS
    - Synthesizer: 279 mm^2
    - 21264 15M T's (packing density due to large caches) in 160 mm^2 in .25u

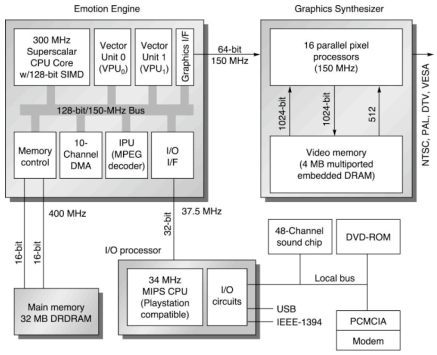
Parallel Processing in the EE

- Game world observation
  - foreground objects change often
  - background object change more slowly
- Result
  - VPU0 - tightly coupled coprocessor
    - foreground duties
    - 128 SIMD instructions coming from the MIPS core
  - VPU1 - independent vector processor
    - background
  - IPU = image processing unit
    - dedicated to MPEG decode
  - memory sizes
    - 16 KB SPRAM - scratch pad RAM
    - 16 KB Icache and 8KB Dcache

Programmer Flow Choice

- parallel - VPU0 filters what to send to VPU1
- serial - simple divide the work pipeline





Servers

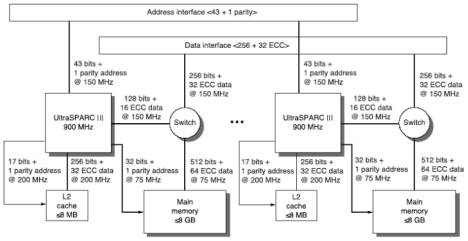
- Different requirements
  - cost-throughput
  - RAS e
    - reliability, availability, serviceability
  - scalability
- Different architecture
  - multiple processors
  - coherent memory (possibly cache coherent)
    - new miss type - *coherence* miss
    - detailed study comes next - we'll abstract here
- Memory architecture differences
  - commercial workloads are quite different than SPEC
    - indicate need for much larger off-chip cache sizes

Sun Fire 6800

- Characteristics
  - 2 - 24 processors UltraSPARC III's
    - 900 MHz - 4-issue
    - 32 KB 4-way SA pseudorandom replacement L1I$ w/ 2 clock latency
    - 64 KB, 4-way SA, write through, no write allocate, pseudorandom replace L1D$ w/ 2 clock latency
    - L2$ unified: up to 8MB external direct mapped, write-back, write allocate
    - L1 I/D miss penalty = 15 to 18 clocks
    - L2 miss penalty = 198 - 252 clocks
    - 2KB write cache & 2 KB prefetch cache
    - 29M T's, 217 mm^2 7-layer CMOS (75% of area is L1$'s)\
    - 70 W at 750 MHz
    - 1368 pin BGA ceramic package
    - in order non-speculative processor
  - redundant interconnect
  - separate back door bus for real time diagnostics

Sun Fire 6800

block diagram



Pitfalls

- cache performance varies with program
  - don't run just one
- small traces lead to bogus measurements
  - need to run large traces
- small address space is death
  - Moore's law ==> need a new address bit every 18 months
- bandwidth is not everything
  - latency may be more important
  - more true of desktop PC than server world
  - caches may hide bandwidth limitations
- application programs are not the real world
  - need to factor OS behavior as well
- page size change is best done by the OS
  - OS manages pages - the HW only accesses them