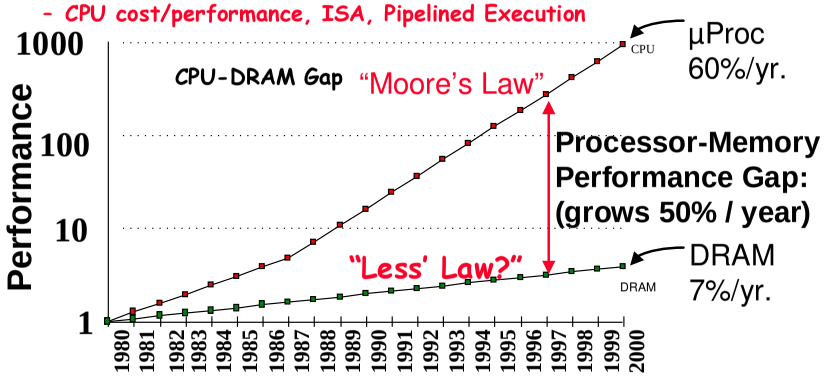
CS252
Graduate Computer Architecture
Lecture 17

Memory Systems
Continued

November 1, 2000
Computer Science 252

Review: Who Cares About the Memory Hierarchy?

- Processor Only Thus Far in Course:
  - CPU cost/performance, ISA, Pipelined Execution



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

Review: Cache performance

- Miss-oriented Approach to Memory Access:

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

- Separating out Memory component entirely
  - AMAT = Average Memory Access Time

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

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

Summary: Miss Rate Reduction

- 3 Cs: Compulsory, Capacity, Conflict
  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
- 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. Frees HW/SW to guess!

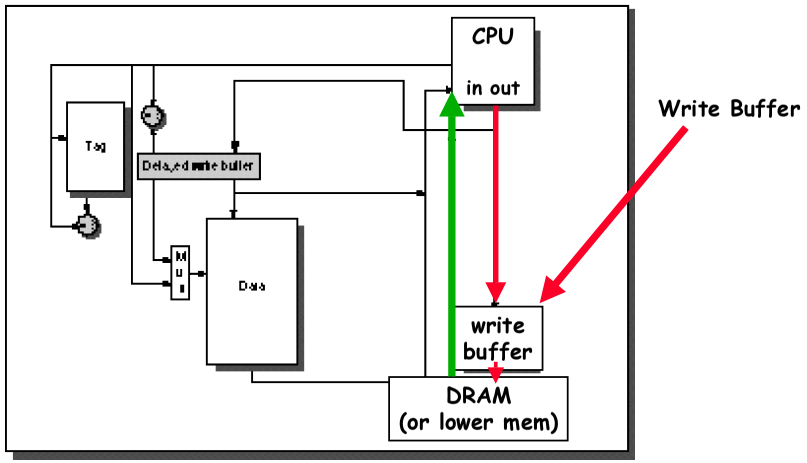
Improving Cache Performance Continued

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?

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

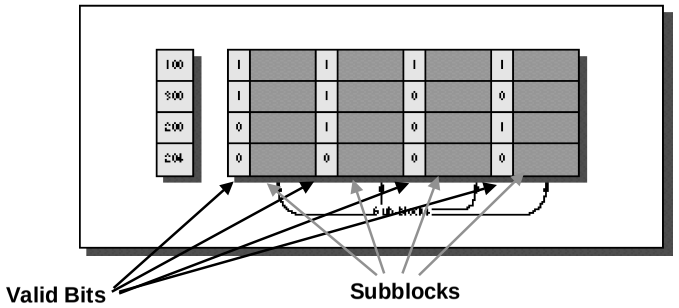


1. Reducing Miss Penalty: Read Priority over Write on Miss

- Write-through with write buffers offer 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 also want buffer to hold misplaced 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

2. Reduce Miss Penalty: Subblock Placement

- Don't have to load full block on a miss
- Have valid bits per subblock to indicate valid
- (Originally invented to reduce tag storage)



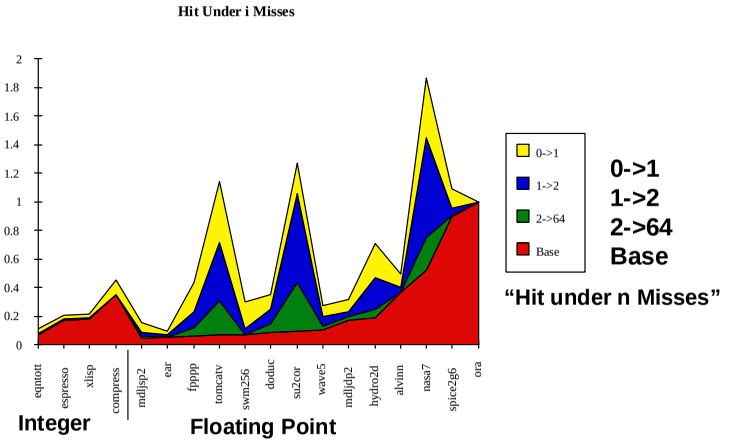
3. Reduce Miss Penalty: Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - *Early restart*—As soon as the requested word of the block arrives, 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 a problem; tend to want next sequential word, so not clear if benefit by early restart

4. 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 *multiple memory banks* (otherwise cannot support)
  - Penium Pro allows 4 outstanding memory misses

Value of Hit Under Miss for SPEC



- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

5. Second level cache

- L2 Equations

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

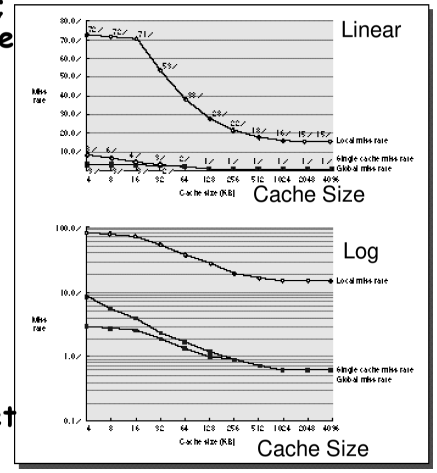
  $Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

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

- 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* ($Miss\ Rate_{L1} \times Miss\ Rate_{L2}$)
  - Global Miss Rate is what matters

Comparing Local and Global Miss Rates

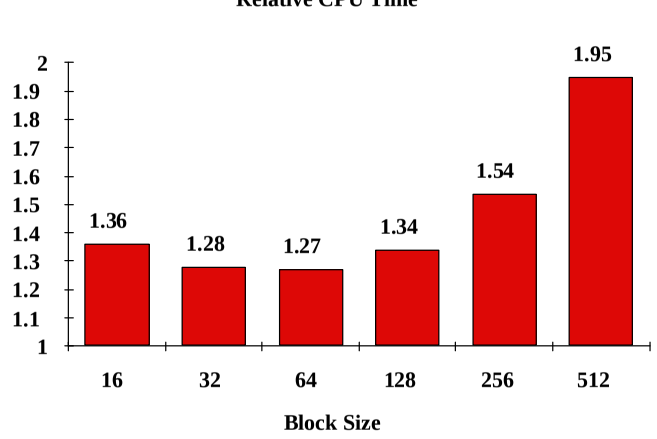
- 32 KByte 1st level cache; Increasing 2nd level cache
- Global miss rate close to single level cache rate provided L2 >> L1
- Don't use local miss rate
- L2 not tied to CPU clock cycle!
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction



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

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



- 32KB L1, 8 byte path to memory

Reducing Miss Penalty Summary

- Five techniques
  - Read priority over write on miss
  - Subblock placement
  - 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

Administrative

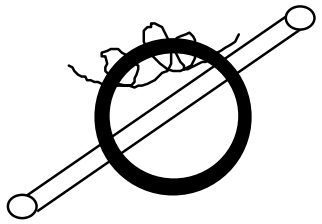
- Got everyone's project descriptions? I will try to comment in next couple of days
- Anyone need resources?
  - NOW: apparently can get account via web site
  - Millenium: can get account via web site
  - SimpleScalar: info on my web page
- Me bad about exam: still grading them!

Main Memory Background

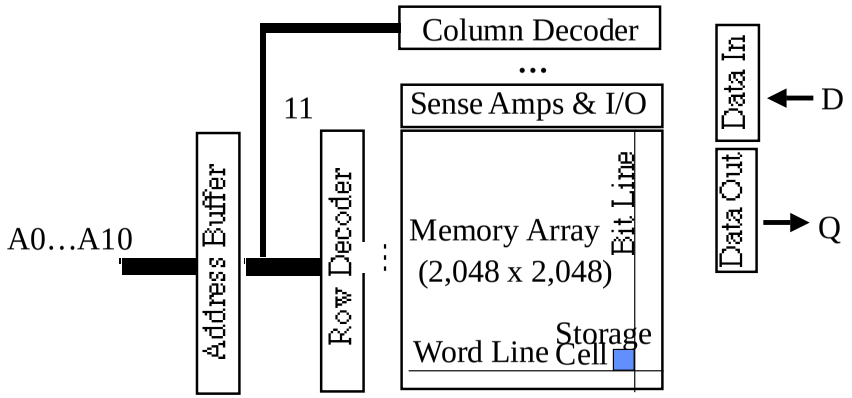
- Performance of Main Memory:
  - Latency: Cache Miss Penalty
    - » *Access Time*: time between request and word arrives
    - » *Cycle Time*: time between requests
  - Bandwidth: I/O & Large Block Miss Penalty (L2)
- Main Memory is *DRAM*: Dynamic Random Access Memory
  - Dynamic since needs to be refreshed periodically (8 ms, 1% time)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - » *RAS* or *Row Access Strobe*
    - » *CAS* or *Column Access Strobe*
- Cache uses *SRAM*: Static Random Access Memory
  - No refresh (6 transistors/bit vs. 1 transistor
    Size: DRAM/SRAM - *4-8*,
    Cost/Cycle time: SRAM/DRAM - *8-16*

Main Memory Deep Background

- "Out-of-Core", "In-Core," "Core Dump"?
- "Core memory"?
- Non-volatile, magnetic
- Lost to 4 Kbit DRAM (today using 64Kbit DRAM)
- Access time 750 ns, cycle time 1500-3000 ns



DRAM logical organization (4 Mbit)



- Square root of bits per RAS/CAS

4 Key DRAM Timing Parameters

- t_{RAC}: minimum time from RAS line falling to the valid data output.
  - Quoted as the speed of a DRAM when buy
  - A typical 4Mb DRAM t_{RAC} = 60 ns
  - Speed of DRAM since on purchase sheet?
- t_{RC}: minimum time from the start of one row access to the start of the next.
  - t_{RC} = 110 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns
- t_{CAC}: minimum time from CAS line falling to valid data output.
  - 15 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns
- t_{PC}: minimum time from the start of one column access to the start of the next.
  - 35 ns for a 4Mbit DRAM with a t_{RAC} of 60 ns

DRAM Performance

- A 60 ns (t_{RAC}) DRAM can
  - perform a row access only every 110 ns (t_{RC})
  - perform column access (t_{CAC}) in 15 ns, but time between column accesses is at least 35 ns (t_{PC}).
    - » In practice, external address delays and turning around buses make it 40 to 50 ns
- These times do not include the time to drive the addresses off the microprocessor nor the memory controller overhead!

DRAM History

- DRAMs: capacity +60%/yr, cost –30%/yr
  - 2.5X cells/area, 1.5X die size in –3 years
- '98 DRAM fab line costs $2B
  - DRAM only: density, leakage v. speed
- Rely on increasing no. of computers & memory per computer (60% market)
  - SIMM or DIMM is replaceable unit => computers use any generation DRAM
- Commodity, second source industry => high volume, low profit, conservative
  - Little organization innovation in 20 years
- Order of importance: 1) Cost/bit 2) Capacity
  - First RAMBUS: 10X BW, +30% cost => little impact

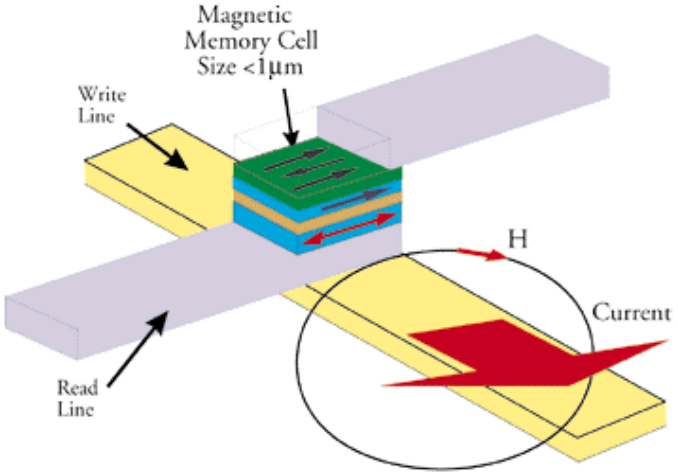
DRAM Future: 1 Gbit DRAM (ISSCC '96; production '02?)

	Mitsubishi	Samsung
• Blocks	512 x 2 Mbit	1024 x 1 Mbit
• Clock	200 MHz	250 MHz
• Data Pins	64	16
• Die Size	24 x 24 mm	31 x 21 mm
– Sizes will be much smaller in production		
• Metal Layers	3	4
• Technology	0.15 micron	0.16 micron

More esoteric Storage Technologies?

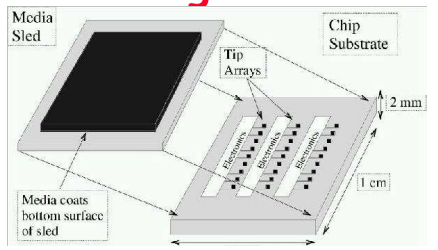
- Tunneling Magnetic Junction RAM (TMJ-RAM):
  - Speed of SRAM, density of DRAM, non-volatile (no refresh)
  - New field called "Spintronics": combination of quantum spin and electronics
  - Same technology used in high-density disk-drives
- MEMs storage devices:
  - Large magnetic "sled" floating on top of lots of little read/write heads
  - Micromechanical actuators move the sled back and forth over the heads

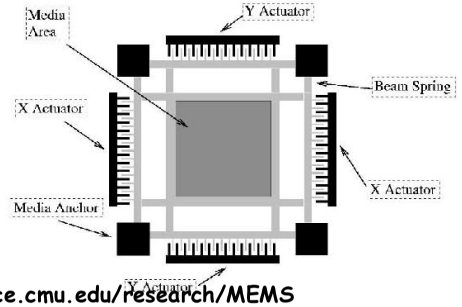
Tunneling Magnetic Junction



MEMS-based Storage

- Magnetic "sled" floats on array of read/write heads
  - Approx 250 Gbit/in²
  - Data rates:
    IBM: 250 MB/s w 1000 heads
    CMU: 3.1 MB/s w 400 heads
- Electrostatic actuators move media around to align it with heads
  - Sweep sled ±50μm in < 0.5μs
- Capacity estimated to be in the 1-10GB in 10cm²

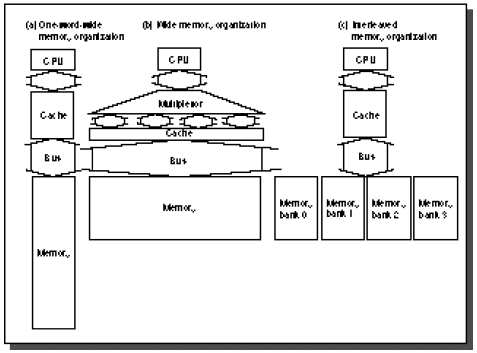




See Ganger et all: http://www.lcs.ece.cmu.edu/research/MEMS

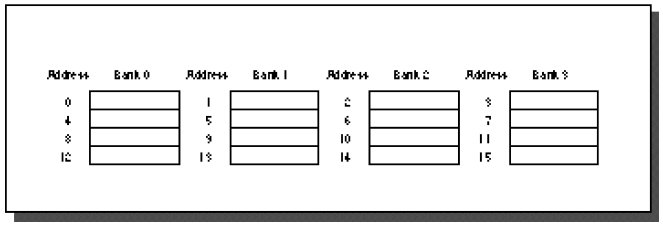
Main Memory Performance

- *Simple*:
  - CPU, Cache, Bus, Memory same width (32 or 64 bits)
- *Wide*:
  - CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits; UtraSPARC 512)
- *Interleaved*:
  - CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*



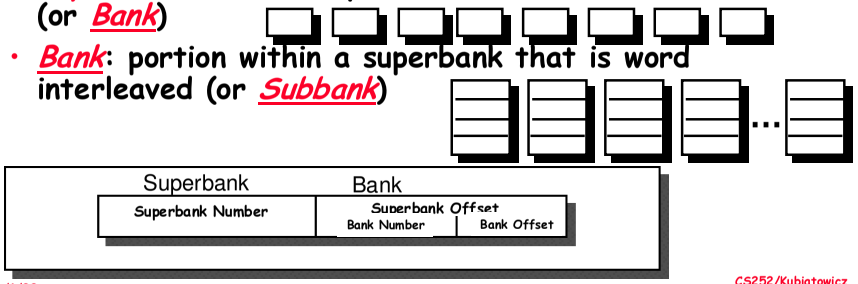
Main Memory Performance

- Timing model (word size is 32 bits)
  - 1 to send address,
  - 6 access time, 1 to send data
  - Cache Block is 4 words
- *Simple M.P.* = 4 x (1+6+1) = 32
- *Wide M.P.* = 1 + 6 + 1 = 8
- *Interleaved M.P.* = 1 + 6 + 4x1 = 11



Independent Memory Banks

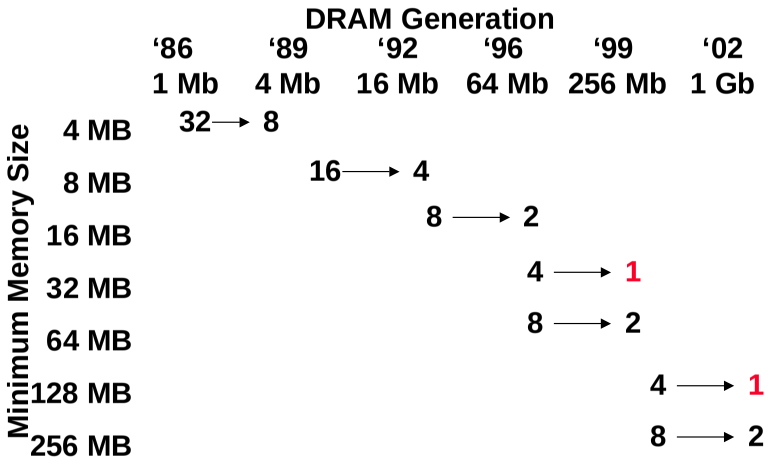
- Memory banks for independent accesses vs. faster sequential accesses
  - Multiprocessor
  - I/O
  - CPU with Hit under n Misses, Non-blocking Cache
- *Superbank*: all memory active on one block transfer (or *Bank*)
- *Bank*: portion within a superbank that is word interleaved (or *Subbank*)



Independent Memory Banks

- How many banks?
  number banks ≤ number clocks to access word in bank
  - For sequential accesses, otherwise will return to original bank before it has next word ready
  - (like in vector case)
- Increasing DRAM => fewer chips => harder to have banks

DRAMs per PC over Time



Avoiding Bank Conflicts

- Lots of banks

```
int x[256][512];
    for (j = 0; j < 512; j = j+1)
        for (i = 0; i < 256; i = i+1)
            x[i][j] = 2 * x[i][j];
```

- Even with 128 banks, since 512 is multiple of 128, conflict on word accesses
- SW: loop interchange or declaring array not power of 2 ("array padding")
- HW: Prime number of banks
  - bank number = address mod number of banks
  - address within bank = address / number of words in bank
  - modulo & divide per memory access with prime no. banks?
  - address within bank = address mod number words in bank
  - bank number? easy if 2^N words per bank

Fast Bank Number

- Chinese Remainder Theorem
  As long as two sets of integers ai and bi follow these rules

  and that ai and aj are co-prime if $i \neq j$, then the integer x has only one solution (unambiguous mapping):
  - bank number = b_0, number of banks = a_0 (= 3 in example)
  - address within bank = b_1, number of words in bank = a_1 (= 8 in example)
  - N word address 0 to N-1, prime no. banks, words power of 2

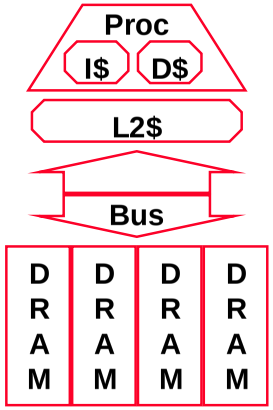
	Seq. Interleaved			Modulo Interleaved		
Bank Number:	0	1	2	0	1	2
Address within Bank: 0	0	1	2	0	16	8
1	3	4	5	9	1	17
2	6	7	8	18	10	2
3	9	10	11	3	19	11
4	12	13	14	12	4	20
5	15	16	17	21	13	5
6	18	19	20	6	22	14
7	21	22	23	15	7	23

Fast Memory Systems: DRAM specific

- Multiple CAS accesses: several names (page mode)
  - *Extended Data Out (EDO)*: 30% faster in page mode
- New DRAMs to address gap; what will they cost, will they survive?
  - *RAMBUS*: startup company; reinvent DRAM interface
    - » Each Chip a module vs. slice of memory
    - » Short bus between CPU and chips
    - » Does own refresh
    - » Variable amount of data returned
    - » 1 byte / 2 ns (500 MB/s per chip)
  - *Synchronous DRAM*: 2 banks on chip, a clock signal to DRAM, transfer synchronous to system clock (66 - 150 MHz)
  - Intel claims RAMBUS Direct (16 b wide) is future PC memory?
    - » Possibly not true! Intel to drop RAMBUS?
- Niche memory or main memory?
  - e.g., Video RAM for frame buffers, DRAM + fast serial output

DRAM Latency >> BW

- More App Bandwidth => Cache misses => DRAM RAS/CAS
- Application BW => Lower DRAM Latency
- RAMBUS, Synch DRAM increase BW but higher latency
- EDO DRAM < 5% in PC



Potential DRAM Crossroads?

- After 20 years of 4X every 3 years, running into wall? (64Mb - 1 Gb)
- How can keep $1B fab lines full if buy fewer DRAMs per computer?
- Cost/bit –30%/yr if stop 4X/3 yr?
- What will happen to $40B/yr DRAM industry?

Main Memory Summary

- Wider Memory
- Interleaved Memory: for sequential or independent accesses
- Avoiding bank conflicts: SW & HW
- DRAM specific optimizations: page mode & Specialty DRAM
- DRAM future less rosy?

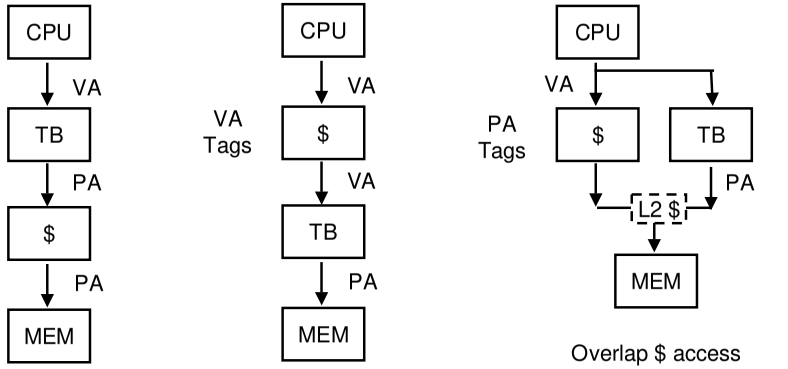
Review: Improving Cache Performance

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

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

2. Fast hits by Avoiding Address Translation



Conventional Organization

Virtually Addressed Cache
Translate only on miss
Synonym Problem

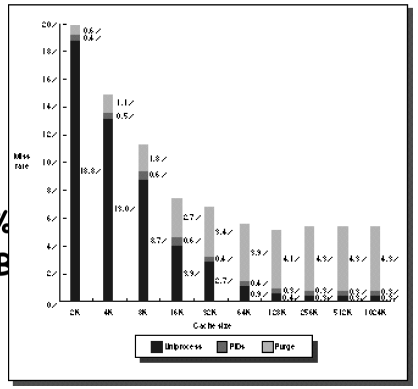
Overlap $ access with VA translation: requires $ index to remain invariant across translation

2. Fast hits by Avoiding Address Translation

- Send virtual address to cache? Called *Virtually Addressed Cache* or just *Virtual Cache* vs. *Physical Cache*
  - Every time process is switched logically must flush the cache; otherwise get false hits
    - » Cost is time to flush + "compulsory" misses from empty cache
  - Dealing with *aliases* (sometimes called *synonyms*); Two different virtual addresses map to same physical address
  - I/O must interact with cache, so need virtual address
- Solution to aliases
  - HW guaranteess covers index field & direct mapped, they must be unique; called *page coloring*
- Solution to cache flush
  - Add *process identifier tag* that identifies process as well as address within process: can't get a hit if wrong process

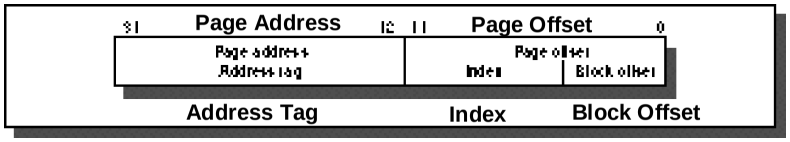
2. Fast Cache Hits by Avoiding Translation: Process ID impact

- Black is uniprocess
- Light Gray is multiprocess when flush cache
- Dark Gray is multiprocess when use Process ID tag
- Y axis: Miss Rates up to 20%
- X axis: Cache size from 2 KB to 1024 KB



2. Fast Cache Hits by Avoiding Translation: Index with Physical Portion of Address

- If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag

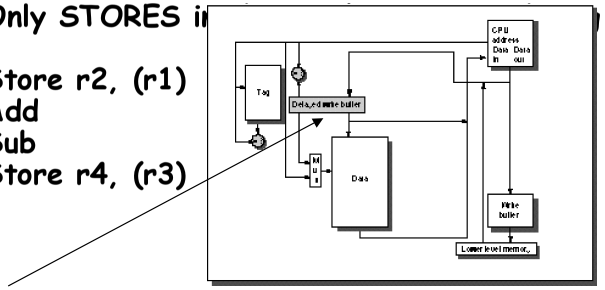


- Limits cache to page size: what if want bigger caches and uses same trick?
  - Higher associativity moves barrier to right
  - Page coloring

3. Fast Hit Times Via Pipelined Writes

- Pipeline Tag Check and Update Cache as separate stages; current write tag check & previous write cache update
- Only STORES i… a miss

```
Store r2, (r1)     Check r1
Add                --
Sub                --
Store r4, (r3)     M[r1]<-r2&
                   check r3
```



- In shade is "Delayed Write Buffer"; must be checked on reads; either complete write or read from buffer

4. Fast Writes on Misses Via Small Subblocks

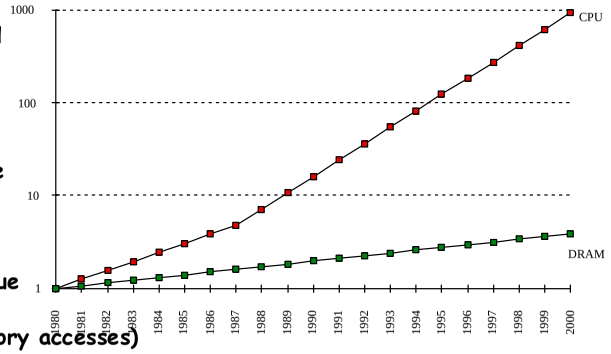
- If most writes are 1 word, subblock size is 1 word, & write through then always write subblock & tag immediately
  - *Tag match and valid bit already set*: Writing the block was proper, & nothing lost by setting valid bit on again.
  - *Tag match and valid bit not set*: The tag match means that this is the proper block; writing the data into the subblock makes it appropriate to turn the valid bit on.
  - *Tag mismatch*: This is a miss and will modify the data portion of the block. Since write-through cache, no harm was done; memory still has an up-to-date copy of the old value. Only the tag to the address of the write and the valid bits of the other subblock need be changed because the valid bit for this subblock has already been set
- Doesn't work with write back due to last case

Cache Optimization Summary

	Technique	*MR*	*MP*	*HT*	*Complexity*
miss rate	Larger Block Size	+	–		0
	Higher Associativity	+		–	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
miss penalty	Priority to Read Misses		+		1
	Subblock Placement		+	+	1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2
hit time	Small & Simple Caches	–		+	0
	Avoiding Address Translation			+	2
	Pipelining Writes			+	1

What is the Impact of What You've Learned About Caches?

- 1960-1985: Speed = *f*(no. operations)
- 1990
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order execution
  - Superscalar Instruction Issue
- 1998: Speed = *f*(non-cached memory accesses)
- What does this mean for
  - Compilers?,Operating Systems?, Algorithms? Data Structures?

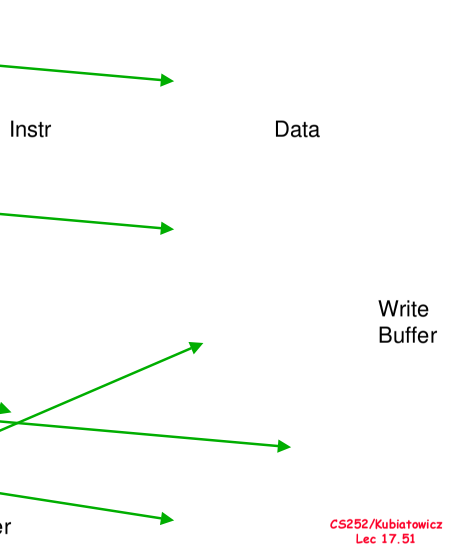


Cache Cross Cutting Issues

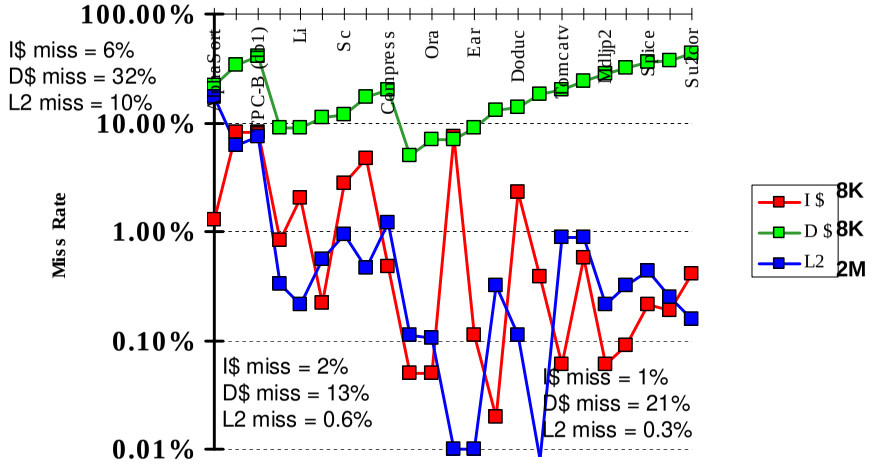
- Superscalar CPU & Number Cache Ports must match: number memory accesses/cycle?
- Speculative Execution and non-faulting option on memory/TLB
- Parallel Execution vs. Cache locality
  - Want far separation to find independent operations vs. want reuse of data accesses to avoid misses
- I/O and consistencyCaches => multiple copies of data
  - Consistency

Alpha 21064

- Separate Instr & Data TLB & Caches
- TLBs fully associative
- TLB updates in SW ("Priv Arch Libr")
- Caches 8KB direct mapped, write thru
- Critical 8 bytes first
- Prefetch instr. stream buffer
- 2 MB L2 cache, direct mapped, WB (off-chip)
- 256 bit path to main memory, 4 x 64-bit modules
- Victim Buffer: to give read priority over write
- 4 entry write buffer between D$ & L2$

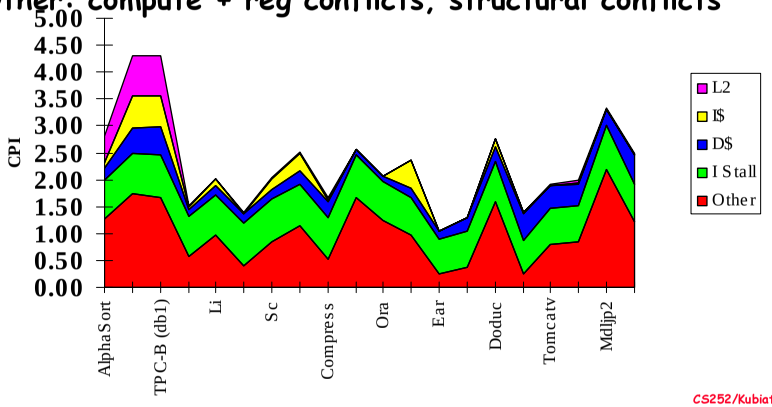


Alpha Memory Performance: Miss Rates of SPEC92



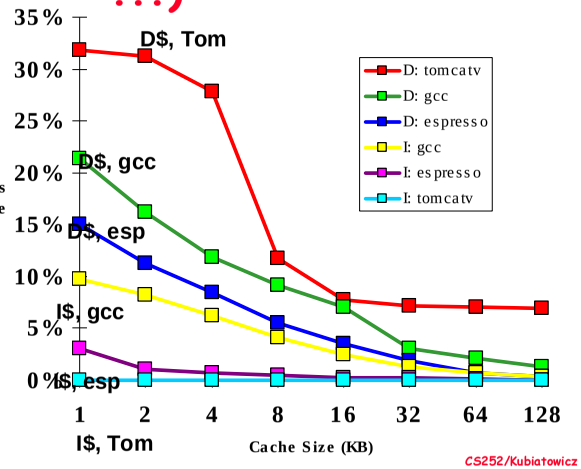
Alpha CPI Components

- Instruction stall: branch mispredict (green);
- Data cache (blue); Instruction cache (yellow); L2$ (pink)
  Other: compute + reg conflicts, structural conflicts

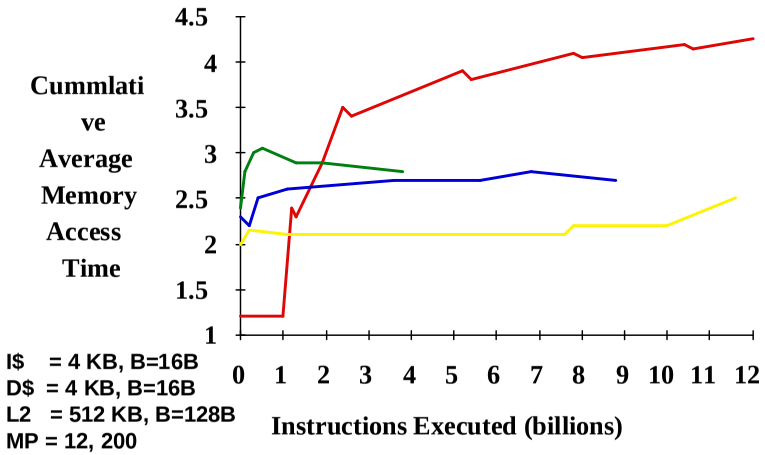


Pitfall: Predicting Cache Performance from Different Prog. (ISA, compiler, ...)

- 4KB Data cache miss rate 8%,12%, or 28%?
- 1KB Instr cache miss rate 0%,3%,or 10%?
- Alpha vs. MIPS for 8KB Data $: 17% vs. 10%
- Why 2X Alpha v. MIPS?



Pitfall: Simulating Too Small an Address Trace



Main Memory Summary

- Wider Memory
- Interleaved Memory: for sequential or independent accesses
- Avoiding bank conflicts: SW & HW
- DRAM specific optimizations: page mode & Specialty DRAM
- DRAM future less rosy?

Cache Optimization Summary

	Technique	MR	MP	HT	Complexity
miss rate	Larger Block Size	+	–		0
	Higher Associativity	+		–	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
miss penalty	Priority to Read Misses		+		1
	Subblock Placement		+	+	1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2
hit time	Small & Simple Caches	–		+	0
	Avoiding Address Translation			+	2
	Pipelining Writes			+	1

Next Time: ECC/Errors

- Next time we will talk about imperfect componets
  - Error correction for handling memory and disk errors
  - Continuous checking for handling pipeline errors
- Coming up: hopefully a talk by a Transmeta person?