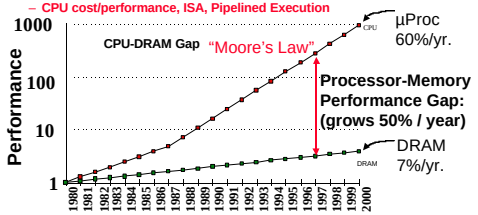
Lecture 4: Memory Hierarchy

Prof. Fred Chong
ECS 250A Computer Architecture
Winter 1999

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)

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

Generations of Microprocessors

- Time of a full cache miss in instructions executed:

1st Alpha (7000): 340 ns/5.0 ns = 68 clks x 2 or 136
2nd Alpha (8400): 266 ns/3.3 ns = 80 clks x 4 or 320
3rd Alpha (t.b.d.): 180 ns/1.7 ns =108 clks x 6 or 648

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

Review: 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)

Review: Cache Performance

CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time

Memory stall clock cycles =
(Reads x Read miss rate x Read miss penalty + Writes x Write miss rate x Write miss penalty)

Memory stall clock cycles =
Memory accesses x Miss rate x Miss penalty

Review: Cache Performance

CPUtime = Instruction Count x ($CPI_{execution}$ + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time

Misses per instruction = Memory accesses per instruction x Miss rate

CPUtime = IC x ($CPI_{execution}$ + Misses per instruction x Miss penalty) x Clock cycle time

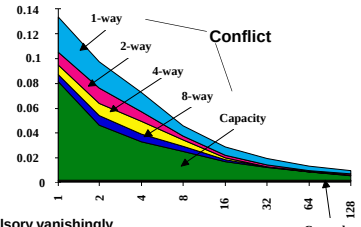
Review: Improving Cache Performance

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

Reducing Misses

- 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.
    (Misses in Fully Associative Size X Cache)
  - *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*.
    (Misses in N-way Associative, Size X Cache)

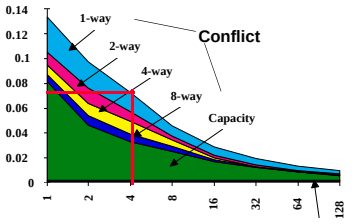
3Cs Absolute Miss Rate (SPEC92)



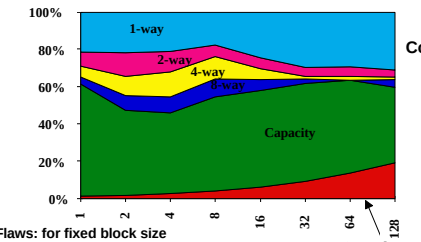
Compulsory vanishingly small

2:1 Cache Rule

miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2



3Cs Relative Miss Rate



Flaws: for fixed block size
Good: insight => invention

How Can Reduce Misses?

- 3 Cs: Compulsory, Capacity, Conflict
- In all cases, assume total cache size not changed:
- What happens if:

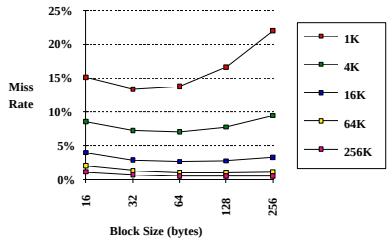
1) Change Block Size:
   Which of 3Cs is obviously affected?

2) Change Associativity:
   Which of 3Cs is obviously affected?

3) Change Compiler:
   Which of 3Cs is obviously affected?

FTC.W99 13

1. Reduce Misses via Larger Block Size



FTC.W99 14

2. Reduce Misses via Higher Associativity

- 2:1 Cache Rule:
  - Miss Rate DM cache size N - Miss Rate 2-way cache size N/2
- Beware: Execution time is only final measure!
  - Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%

FTC.W99 15

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

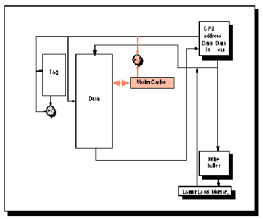
Cache Size (KB)	Associativity 1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	2.01
2	1.98	1.86	1.76	1.68
4	1.72	1.67	1.61	1.53
8	1.46	1.48	1.47	1.43
16	1.29	1.32	1.32	1.32
32	1.20	1.24	1.25	1.27
64	1.14	1.20	1.21	1.23
128	1.10	1.17	1.18	1.20

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

FTC.W99 16

3. Reducing Misses via a "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



FTC.W99 17

4. 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

FTC.W99 18

5. 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

6. Reducing Misses by Software Prefetching Data

- 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
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

7. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - *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

Merging Arrays Example

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key; improve spatial locality

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
      for (i = 0; i < 5000; i = i+1)
            x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
      for (j = 0; j < 100; j = j+1)
            x[i][j] = 2 * x[i][j];
```

Sequential accesses instead of striding through memory every 100 words; improved spatial locality

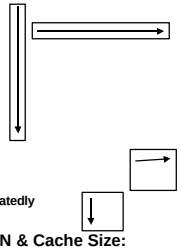
Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
      a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
      d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {   a[i][j] = 1/b[i][j] * c[i][j];
      d[i][j] = a[i][j] + c[i][j];}
```

2 misses per access to a & c vs. one miss per access; improve spatial locality

Blocking Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     {r = 0;
      for (k = 0; k < N; k = k+1){
        r = r + y[i][k]*z[k][j];};
      x[i][j] = r;
     };
```



- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity Misses a function of N & Cache Size:
  - 3 NxNx4 => no capacity misses; otherwise ...
- Idea: compute on BxB submatrix that fits

FTC.W99 25

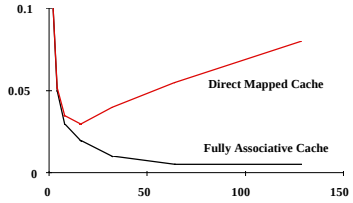
Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
     {r = 0;
      for (k = kk; k < min(kk+B-1,N); k = k+1) {
        r = r + y[i][k]*z[k][j];};
      x[i][j] = x[i][j] + r;
     };
```

- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- Conflict Misses Too?

FTC.W99 26

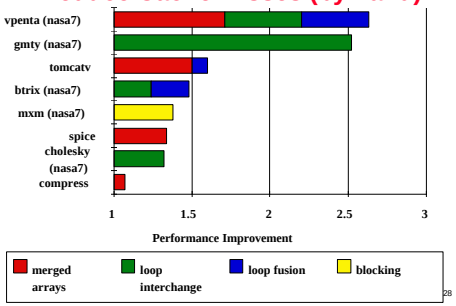
Reducing Conflict Misses by Blocking



- Conflict misses in caches not FA vs. Blocking size
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

FTC.W99 27

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



28

Summary

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

- 3 Cs: Compulsory, Capacity, Conflict
  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
- Remember danger of concentrating on just one parameter when evaluating performance

FTC.W99 29

Review: Improving Cache Performance

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

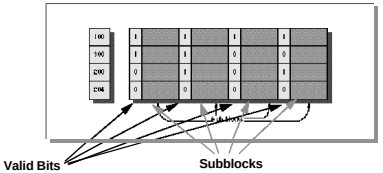
FTC.W99 30

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?
  - 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)



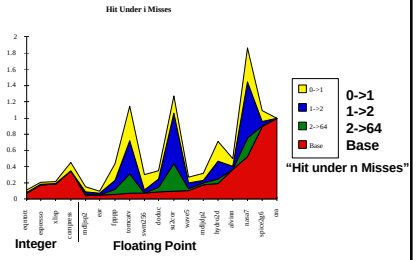
3. Reduce Miss Penalty: Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - *Early restart*—As soon as the requested word of the block 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 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 out-of-order executuion CPU
- "*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

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

5th Miss Penalty

- L2 Equations

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

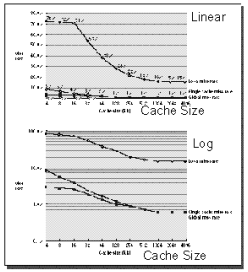
  $Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

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

- 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

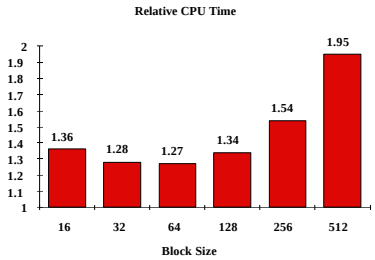
- 32 KByte 1st level cache; Increasing 2nd level cache
- Global miss rate close to single level cache rate provided L2 >> L1
- Don't use local miss rate
- L2 not tied to CPU clock cycle!
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction



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

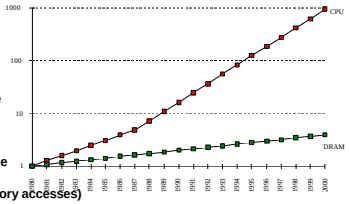
Reducing Miss Penalty Summary

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

- 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

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

- 1960-1985: Speed = ƒ(no. operations)
- 1990
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order execution
  - Superscalar Instruction Issue
- 1998: Speed = ƒ(non-cached memory accesses)
- Superscalar, Out-of-Order machines hide L1 data cache miss (-5 clocks) but not L2 cache miss (-50 clocks)?



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

Improving Cache Performance

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

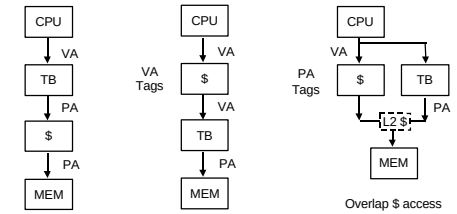
1. Fast Hit times via Small and Simple Caches

- 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

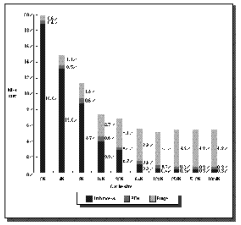
- Send virtual address to cache? Called *Virtually Addressed Cache* or just *Virtual Cache* vs. *Physical Cache*
  - Every time process is switched logically must flush the cache; otherwise get false hits
    - » Cost is time to flush + "compulsory" misses from empty cache
  - Dealing with *aliases* (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

Virtually Addressed Caches



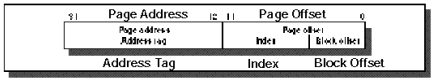
2. Fast Cache Hits by Avoiding Translation: Process ID impact

- 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 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

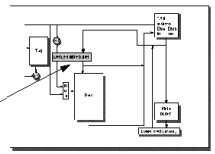


- Limits cache to page size: what if want bigger caches and uses same trick?
  - Higher associativity moves barrier to right
  - Page coloring

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 in the pipeline; empty during 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

FTC.W99 49

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

FTC.W99 50

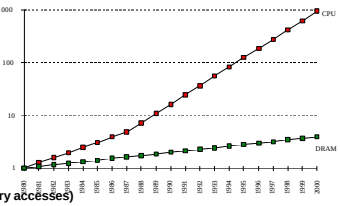
Cache Optimization Summary

	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

FTC.W99 51

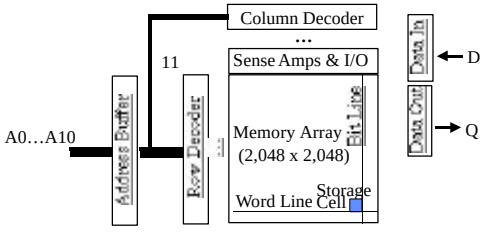
What is the Impact of What You've Learned About Caches?

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



FTC.W99 52

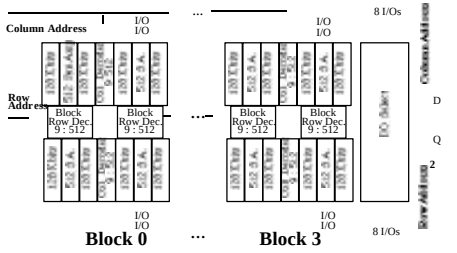
DRAM logical organization (4 Mbit)



- Square root of bits per RAS/CAS

FTC.W99 53

DRAM physical organization (4 Mbit)



FTC.W99 54

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

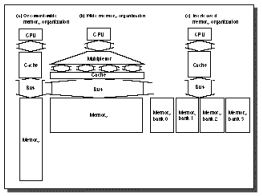
DRAM Future: 1 Gbit DRAM (ISSCC '96; production '02?)

	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

- Wish could do this for Microprocessors!

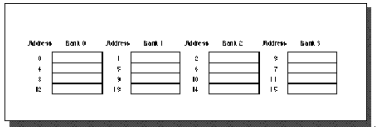
Main Memory Performance

- *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; UltraSPARC 512)
- *Interleaved*:
  - CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*



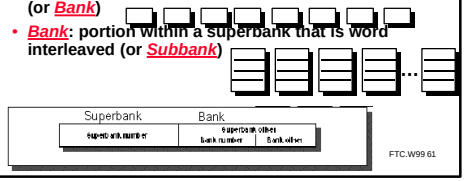
Main Memory Performance

- 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

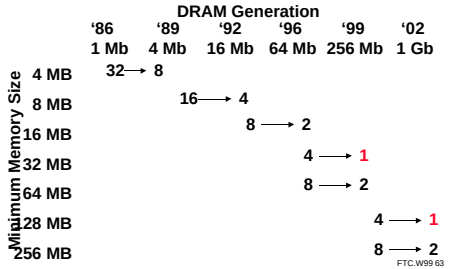
- Memory banks for independent accesses vs. faster sequential accesses
  - Multiprocessor
  - I/O
  - CPU with Hit under n Misses, Non-blocking Cache
- *Superbank*: all memory active on one block transfer (or *Bank*)
- *Bank*: portion within a superbank that is word interleaved (or *Subbank*)



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
  $b_i = x \bmod a_i, 0 \le b_i < a_i, 0 \le x < a_0 \times a_1 \times a_2 \times \ldots$
  and that ai and aj are co-prime if i ° 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

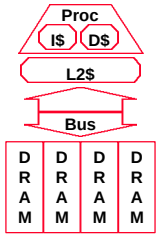
		Seq. Interleaved			Modulo Interleaved		
Bank Number:		0	1	2	0	1	2
Address within Bank:	0	0	1	2	0	16	8
	1	3	4	5	9	1	17
	2	6	7	8	18	10	2
	3	9	10	11	3	19	11
	4	12	13	14	12	4	20
	5	15	16	17	21	13	5
	6	18	19	20	6	22	14
	7	21	22	23	15	7	23

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
- 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



FTC.W99 67

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?

FTC.W99 68

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?

FTC.W99 69

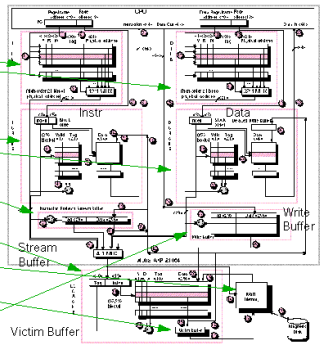
Cache Cross Cutting Issues

- 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

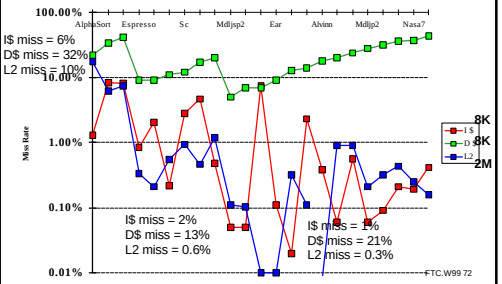
FTC.W99 70

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$



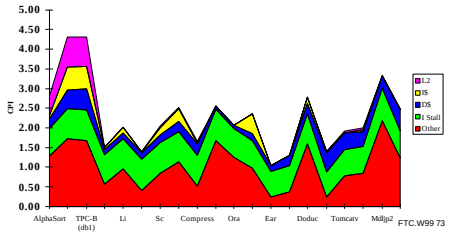
Alpha Memory Performance: Miss Rates of SPEC92



FTC.W99 72

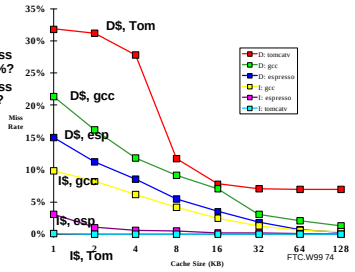
Alpha CPI Components

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

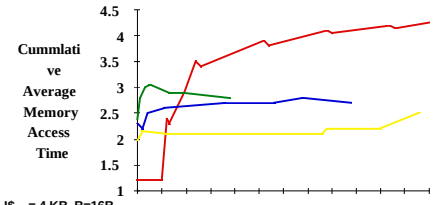


Pitfall: Predicting Cache Performance from Different Prog. (ISA, compiler, ...)

- 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



I$ = 4 KB, B=16B
D$ = 4 KB, B=16B
L2 = 512 KB, B=128B
MP = 12, 200

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?

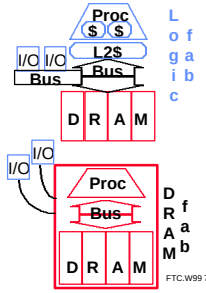
Cache Optimization Summary

	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

IRAM Vision Statement

Microprocessor & DRAM on a single chip:

- on-chip memory latency 5-10X, bandwidth 50-100X
- improve energy efficiency 2X-4X (no off-chip bus)
- serial I/O 5-10X v. buses
- smaller board area/volume
- adjustable memory size/width

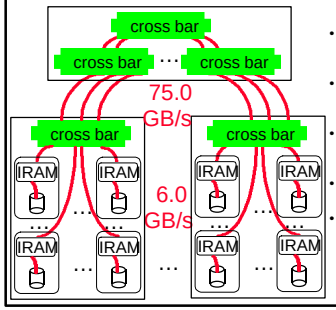


App #1: Intelligent PDA (2003?)

- Pilot PDA (todo,calendar, calculator, addresses,...)
- \+ Gameboy (Tetris, ...)
- \+ Nikon Coolpix (camera)
- \+ Cell Phone, Pager, GPS, tape recorder, TV remote, am/fm radio, garage door opener, ...
- \+ Wireless data (WWW)
- \+ Speech, vision recog. – Speech control of all devices
- \+ Speech output for conversations – Vision to see surroundings, scan documents, read bar codes, measure room

FTC.W99 79

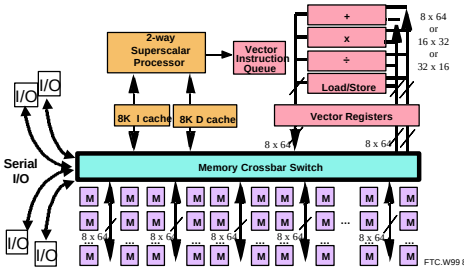
App #2: "Intelligent Disk"(IDISK): Scaleable Decision Support?



- 1 IRAM/disk + xbar + fast serial link v. conventional SMP
- Move function to data v. data to CPU (scan, sort, join,...)
- Network latency = f(SW overhead), not link distance
- Avoid I/O bus bottleneck of SMP
- Cheaper, faster, more scalable (-1/3 $, 3X perf)

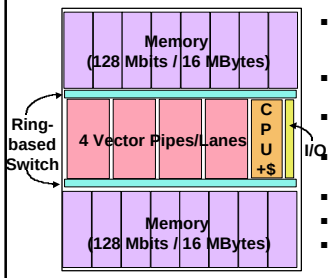
FTC.W99 80

V-IRAM-2: 0.13 µm, Fast Logic, 1GHz 16 GFLOPS(64b)/64 GOPS(16b)/128MB



FTC.W99 81

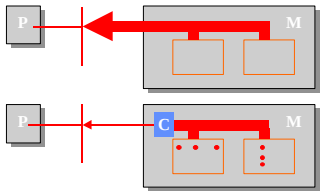
Tentative VIRAM-1 Floorplan



- 0.18 µm DRAM 32 MB in 16 banks x 256b, 128 subbanks
- 0.25 µm, 5 Metal Logic
- - 200 MHz MIPS, 16K I$, 16K D$
- - 4 200 MHz FP/int. vector units
- die: - 16x16 mm
- xtors: - 270M
- power: -2 Watts

FTC.W99 82

Processor-Memory Bandwidth



- Up to 1600X speedup!

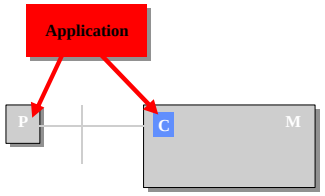
FTC.W99 83

Active-Page Interface

- Memory Operations
  - *write(address,data)* and *read(address)*
- Active-Page functions (*AP_funcs*)
- Allocation
  - *AP_alloc(group_id, AP_funcs, vaddr)*
- Shared Variables
- Synchronization Variables

FTC.W99 84

Partitioning



- Processor-centric vs Memory-centric

FTC.W99 85

Processor-Centric Applications

Matrix	Matrix multiply for Simplex and finite element
MPEG-MMX	MPEG decoder using MMX instructions

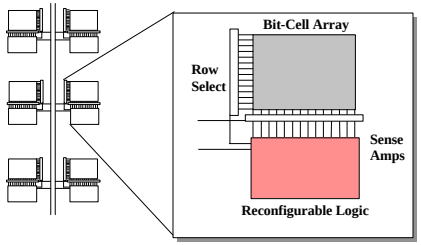
FTC.W99 86

Memory-Centric Applications

Array	C++ std template library class
Database	Unindexed search of an address database
Median	Median filter for images
Dynamic Programming	Protein sequence matching

FTC.W99 87

Reconfigurable Architecture DRAM (RADram)



FTC.W99 88

RADram vs IRAM

- Higher *yield* through redundancy
  - IRAMs will fabricate at processor costs
  - RADrams will be closer to DRAMs
- RADram exploits parallelism
- RADram can be application-specific
- RADram supports commodity processors

FTC.W99 89

RADram Technology

- 1 Gbit DRAM in 2001 (SIA Roadmap)
  - 50% for logic = 32M transistors
  - 32K LEs (1K transistors / LE)
  - 512Mbit = 128 x 512K superpages
  - 256 LEs / superpage

FTC.W99 90

RADram Logic

Applicaton	*CLBs*	*Speed*
Array	115	29.2 ns
Database	142	35.4 ns
Dyn Prog	179	39.2 ns
MPEG-MMX	131	34.6 ns

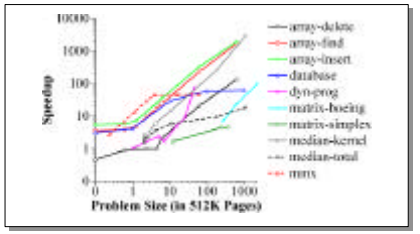
FTC.W99 91

RADram Parameters

Parameter	*Reference*	*Variation*
CPU Clock	1 GHz	--
L1 I-Cache	64K	--
L1 D-Cache	64K	32K-256K
L2 Cache	1M	256K-4M
Reconf Logic	100 MHz	10-500 MHz
Cache Miss	50 ns	0-600 ns

FTC.W99 92

Speedup vs Data Size



FTC.W99 93