

Lecture 12: Caches I

Prof. John Kubiawicz
Computer Science 252
Fall 1998

JDK.F98
Slide 1

Review: Genetic Programming for Design

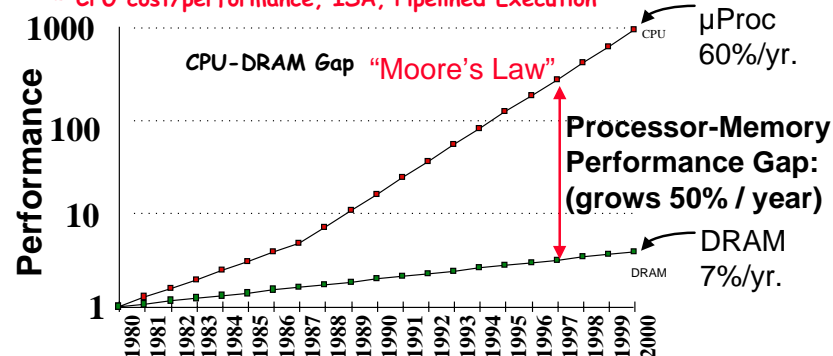
- Genetic programming has two key aspects:
 - An *Encoding* of the design space.
 - » This is a symbolic representation of the result space (genome).
 - » Much of the domain-specific knowledge and "art" involved here.
 - A *Reproduction* strategy
 - » Includes a method for generating offspring from parents
 - Mutation: Changing random portions of an individual**
 - Crossover: Merging aspects of two individuals**
 - » Includes a method for evaluating the effectiveness ("fitness") of individual solutions.
- Generation of new branch predictors via genetic programming:
 - Everything derived from a "basic" predictor (table) + simple operators.
 - Expressions arranged in a tree
 - **Mutation: random modification of node/replacement of subtree.**
 - **Crossover: swapping the subtrees of two parents.**

JDK.F98
Slide 2

Review: Who Cares About the Memory Hierarchy?

- Processor Only Thus Far in Course:

- CPU cost/performance, ISA, Pipelined Execution



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

JDK.F98
Slide 3

Processor-Memory Performance Gap "Tax"

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

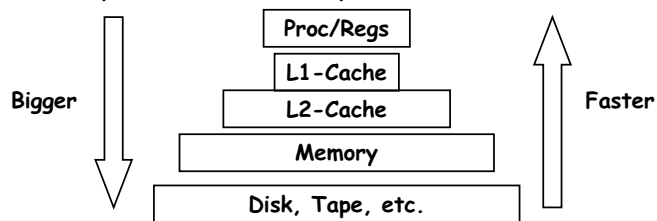
- 2 dies per package: Proc/I\$/D\$ + L2\$

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

JDK.F98
Slide 4

What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spacial and temporal locality
- In computer architecture, almost everything is a cache!
 - Registers a cache on variables
 - 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?



JDK.F98
Slide 5

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 \times $3X$ clock rate \times $3X$ Instr/clock \Rightarrow $-5X$

JDK.F98
Slide 6

What happens on a Cache miss?

- For in-order pipeline, 2 options:
 - Freeze pipeline in Mem stage (popular early on: Sparc, R4000)

IF	ID	EX	Mem	stall	stall	stall	...	stall	Mem	Wr
IF	ID	EX	stall	stall	stall	...	stall	stall	Ex	Wr
 - Use Full/Empty bits in registers + MSHR queue
 - » MSHR = "Miss Status/Handler Registers" (Kroft)
 - Each entry in this queue keeps track of status of outstanding memory requests to one complete memory line.
 - Per cache-line: keep info about memory address.
 - For each word: register (if any) that is waiting for result.
 - Used to "merge" multiple requests to one memory line
 - » New load creates MSHR entry and sets destination register to "Empty". Load is "released" from pipeline.
 - » Attempt to use register before result returns causes instruction to block in decode stage.
 - » Limited "out-of-order" execution with respect to loads.
 - Popular with in-order superscalar architectures.
- Out-of-order pipelines already have this functionality built in... (load queues, etc).

JDK.F98
Slide 7

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)

JDK.F98
Slide 8

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

Note: **memory hit time is included in execution cycles.**

JDK.F98
Slide 9

Review: Cache Performance

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

Misses per instruction =
Memory accesses/inst x Miss rate

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

JDK.F98
Slide 10

Review: Improving Cache Performance

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

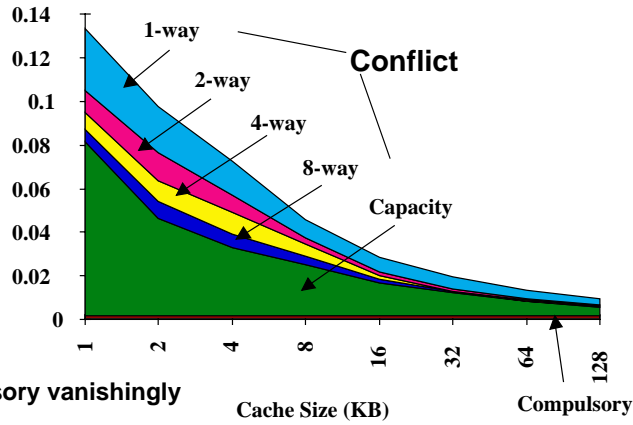
JDK.F98
Slide 11

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)
- More recent, 4th "C":
 - Coherence - Misses caused by cache coherence.

JDK.F98
Slide 12

3Cs Absolute Miss Rate (SPEC92)

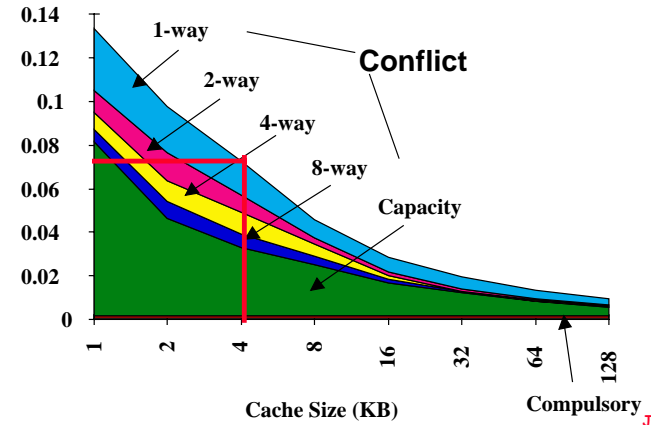


Compulsory vanishingly small

JDK.F98
Slide 13

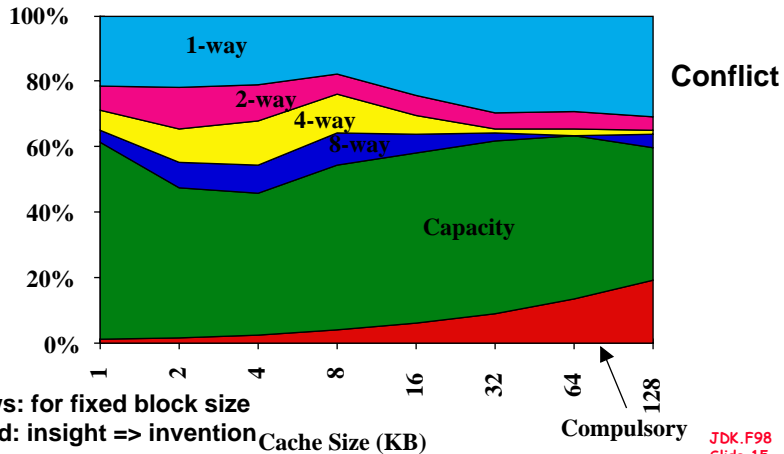
2:1 Cache Rule

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



JDK.F98
Slide 14

3Cs Relative Miss Rate



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

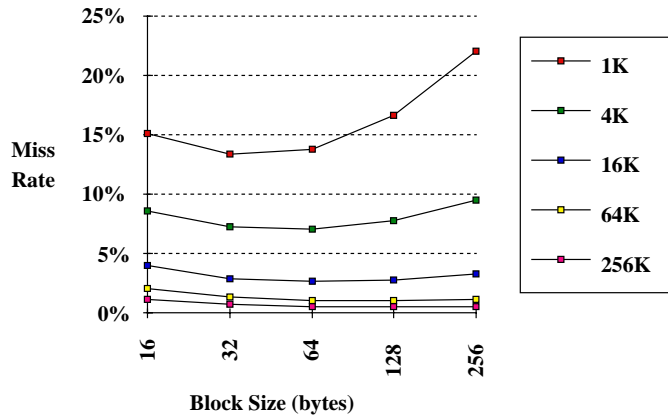
JDK.F98
Slide 15

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?

JDK.F98
Slide 16

1. Reduce Misses via Larger Block Size



JDK.F98
Slide 17

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%

JDK.F98
Slide 18

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

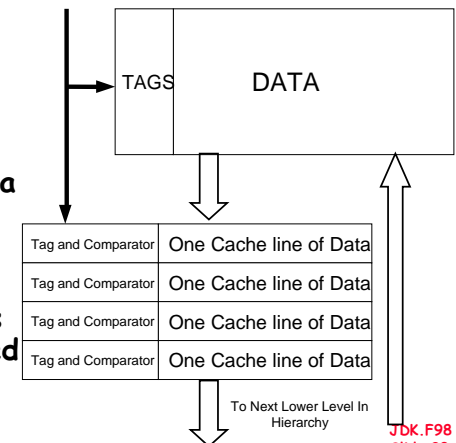
Cache Size (KB)	Associativity				8-way
	1-way	2-way	4-way	8-way	
1	2.33	2.15	2.07	2.01	
2	1.98	1.86	1.76	1.68	
4	1.72	1.67	1.61	1.53	
8	1.46	<u>1.48</u>	<u>1.47</u>	1.43	
<u>16</u>	<u>1.29</u>	<u>1.32</u>	<u>1.32</u>	<u>1.32</u>	
<u>32</u>	<u>1.20</u>	<u>1.24</u>	<u>1.25</u>	<u>1.27</u>	
<u>64</u>	<u>1.14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

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

JDK.F98
Slide 19

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



JDK.F98
Slide 20

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

JDK.F98
Slide 21

CS 252 Administrivia

- Upcoming events in CS 252
 - 19-Oct Project Proposal due (Monday)
 - » Submission in postscript, as link from project-specific web page. Due by 5:00. See course-page for information
 - 20-Oct 8 minute Proj Meetings: 10-11:30, 2-3 (Tue)
- Reading assignment for Wednesday:
 - "Pseudo-Randomly Interleaved Memory"
B. Ramakrishna Rau
 - "The Multics Virtual Memory: Concepts and Design"
Bensoussan, Clingen, Daley
 - "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes"
M Hsiao
 - "A Class of Odd-Weight-Column SEC-DED-SbED Codes for Memory System Applications"
Shigeo Kaneda
 - » One paragraph on paper (1) and, one on paper (2), and one on papers 3 and 4.
 - Papers 3 and 4 very short...

JDK.F98
Slide 22

5. Reducing Misses by Hardware Prefetching of Instructions & Datals

- 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

JDK.F98
Slide 23

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

JDK.F98
Slide 24

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

JDK.F98
Slide 25

Merging Arrays Example

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

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

Reducing conflicts between val & key;
improve spatial locality

JDK.F98
Slide 26

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

JDK.F98
Slide 27

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

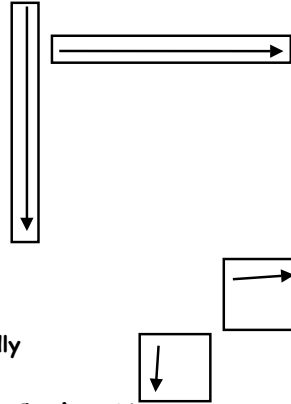
JDK.F98
Slide 28

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:**
 - $2N^3 + N^2 \Rightarrow$ (assuming no conflict; otherwise ...)
- **Idea: compute on BxB submatrix that fits**

JDK.F98
Slide 29

Blocking Example

```

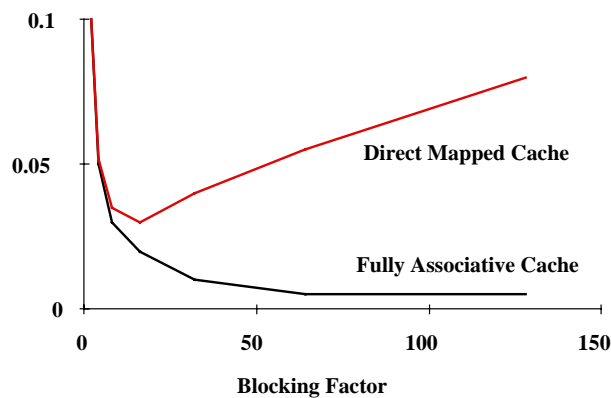
/* After */
for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
    for (i = 0; i < N; i = i+1)
      for (j = jj; j < min(jj+B-1,N); j = j+1)
        {r = 0;
         for (k = kk; k < min(kk+B-1,N); k = k+1) {
           r = r + y[i][k]*z[k][j];
         }
        x[i][j] = x[i][j] + r;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

```

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

JDK.F98
Slide 30

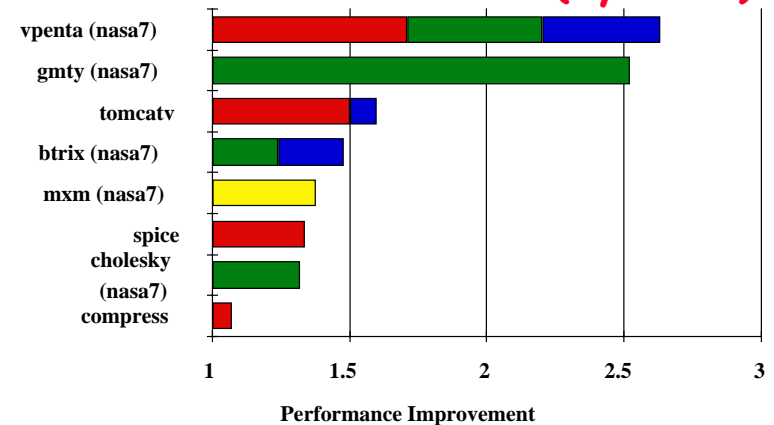
Reducing Conflict Misses by Blocking



- **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

JDK.F98
Slide 31

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



■ merged arrays
 ■ loop interchange
 ■ loop fusion
 ■ blocking

F98
32

Summary

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

- **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

JDK.F98
Slide 33

Review: Improving Cache Performance

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

JDK.F98
Slide 34

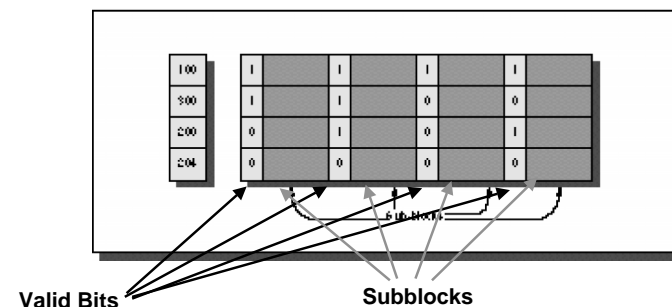
1. Reducing Miss Penalty: Read Priority over Write on Miss

- 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

JDK.F98
Slide 35

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)



JDK.F98
Slide 36

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



block

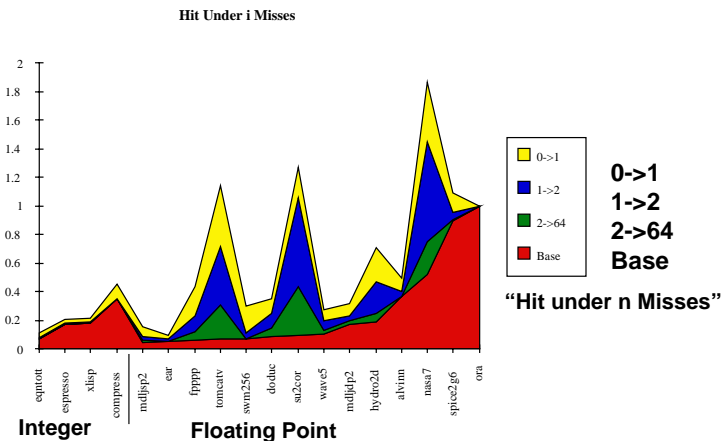
JDK.F98
Slide 37

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)
 - Pentium Pro allows 4 outstanding memory misses

JDK.F98
Slide 38

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

JDK.F98
Slide 39

5th Miss Penalty

• L2 Equations

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

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

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

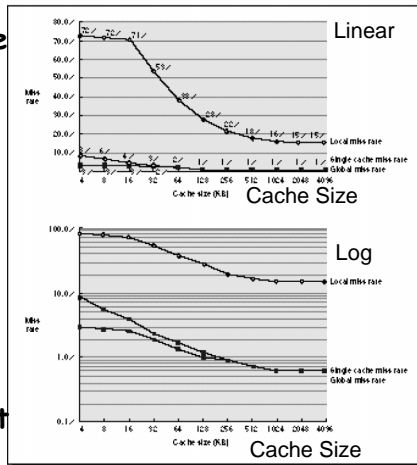
• Definitions:

- **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} × Miss Rate_{L2})
- Global Miss Rate is what matters

JDK.F98
Slide 40

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



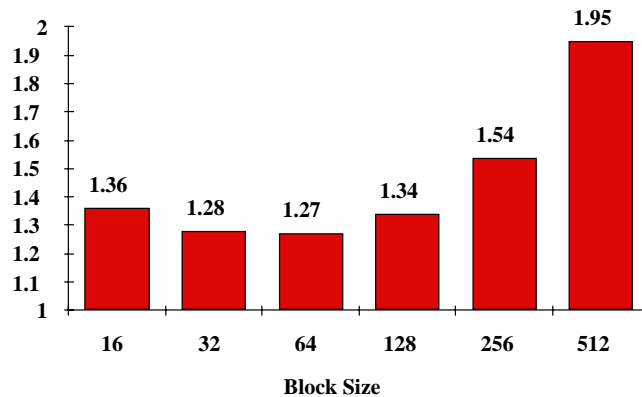
JDK.F98
Slide 41

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

JDK.F98
Slide 42

L2 cache block size & A.M.A.T. Relative CPU Time



- 32KB L1, 8 byte path to memory

JDK.F98
Slide 43

Reducing Miss Penalty Summary

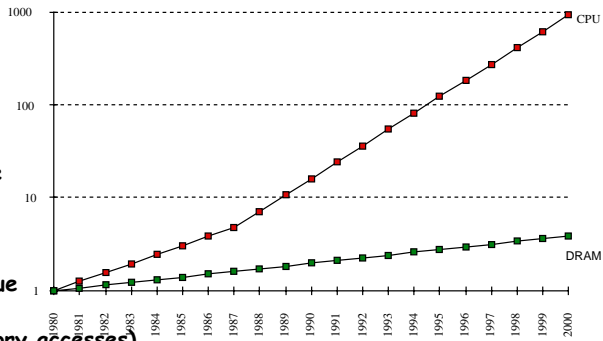
$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times Miss\ rate \times Miss\ penalty \right) \times Clock\ cycle\ time$$

- 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

JDK.F98
Slide 44

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

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



JDK.F98
Slide 45

Cache Optimization Summary

	<i>Technique</i>	<i>MR</i>	<i>MP</i>	<i>HT</i>	<i>Complexity</i>
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

JDK.F98
Slide 46