

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Computer Architecture
ECE 568

Part 13

Cache Performance and Improvement Techniques

Israel Koren
Fall 2011

ECE568/Koren Part.13 .1

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Cache Measures

- ◆ **Hit rate**: fraction found in that level
 - So high that usually we focus on **Miss rate**
- ◆ **Average memory-access time**
 $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- ◆ **Miss penalty**: time to replace a block from lower level, including time to forward to CPU
 - **access time**: time to lower level
= f(latency to lower level)
 - **transfer time**: time to transfer block
= f(BW between upper & lower levels)

ECE568/Koren Part.13 .2

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Cache performance - Impact on CPU

◆ Miss-oriented Approach to Memory Access:

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

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

◆ Separating out Memory component entirely

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

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

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

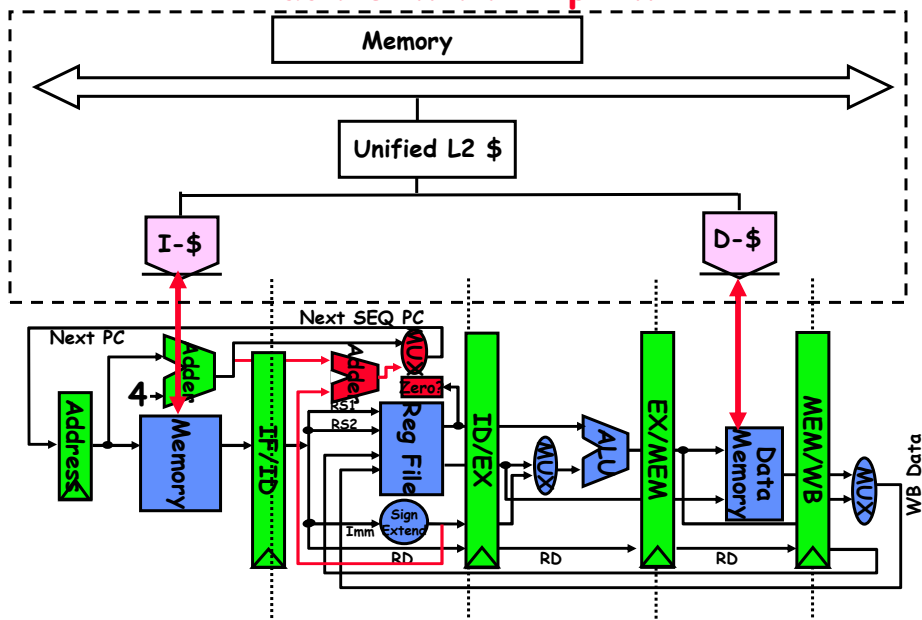
$$= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) \times Freq_{Inst_F} + (HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data}) \times Freq_{data_op}$$

ECE568/Koren Part.13.3

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Caches in the Pipeline



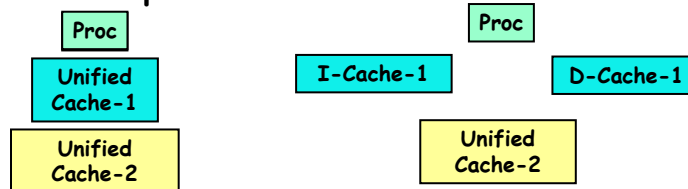
ECE568/Koren Part.13.4

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

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 misses)?

- Assume 33% data ops \Rightarrow 75% accesses for instructions (1.0/1.33)
- hit_time=1, miss_time=50
- Note that *data* hit has 1 stall for unified cache (only one port)

$$AMAT_{\text{Harvard}} = 75\% \times (1 + 0.64\% \times 50) + 25\% \times (1 + 6.47\% \times 50) = 2.05$$

$$AMAT_{\text{Unified}} = 75\% \times (1 + 1.99\% \times 50) + 25\% \times (1 + 1 + 1.99\% \times 50) = 2.24$$

ECE568/Koren Part.13 .5

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

How to Improve Cache Performance?

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

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

ECE568/Koren Part.13 .6

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

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 (*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*)

◆ 4th "C":

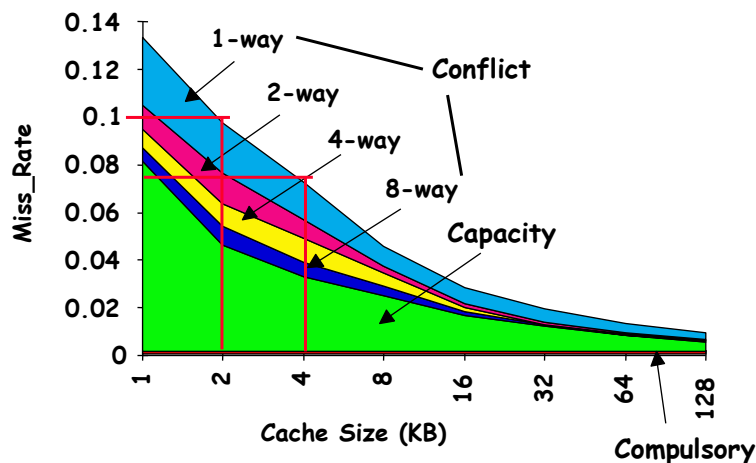
- **Coherence** - Misses caused by cache coherence

ECE568/Koren Part.13.7

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

3Cs Miss Rate (SPEC92) vs. Cache size



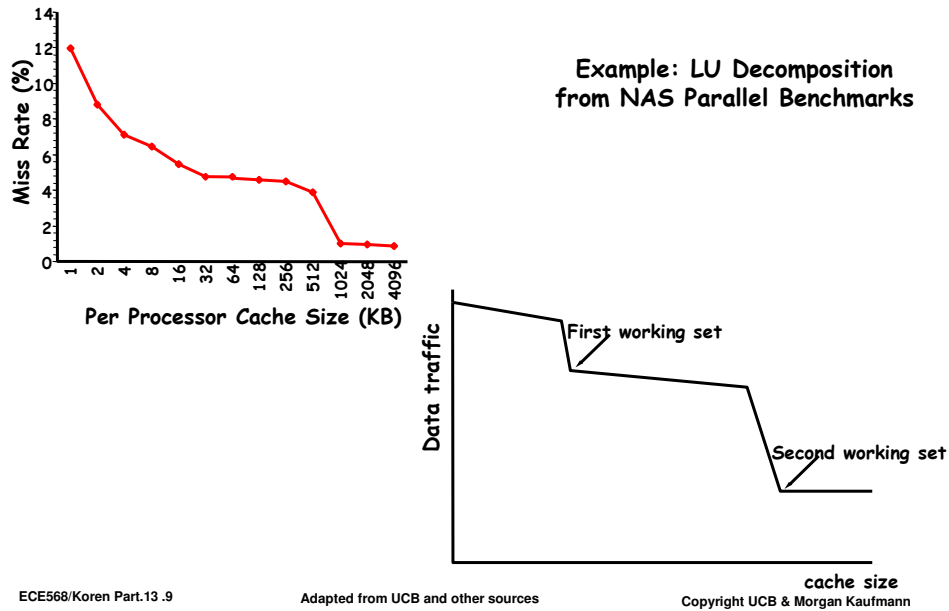
◆ Old rule of thumb: 2x size => 25% cut in miss rate

ECE568/Koren Part.13.8

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

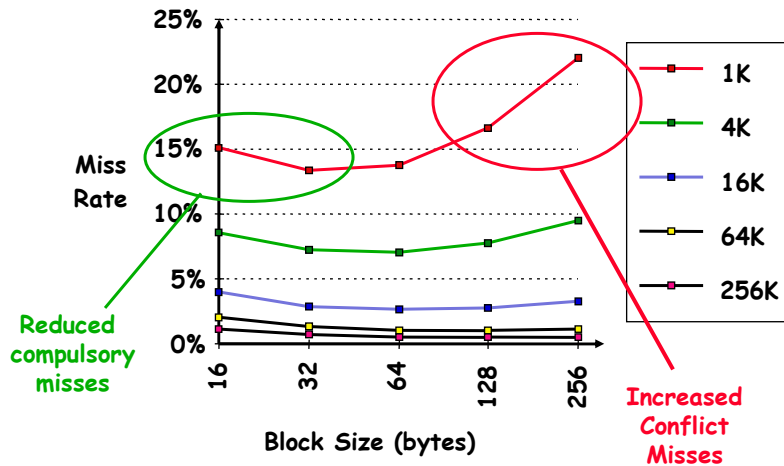
Huge Caches - Working Sets



Cache Organization impact

- ◆ Assume total cache size not changed
- ◆ Which of 3Cs is obviously affected?
- ◆ Change Block Size:
- ◆ Change Associativity:

Larger Block Size (fixed size & assoc)



ECE568/Koren Part.13 .11

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Conflict misses reduce with higher associativity but will AMAT go down?

- ◆ Example: ClockCycleTime 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 higher associativity)

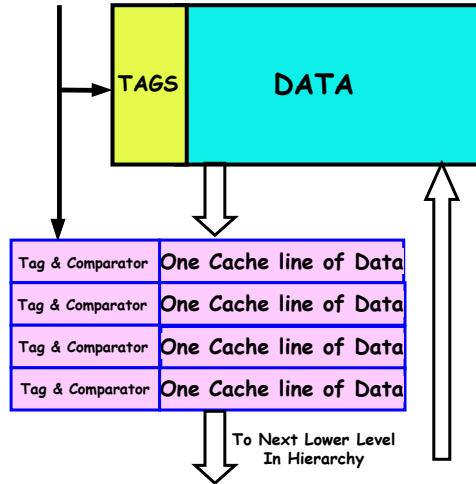
ECE568/Koren Part.13 .12

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

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
- ◆ 4-entry **victim cache** removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- ◆ Used in Alpha, HP machines



ECE568/Koren Part.13 .13

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

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 complicated if hit takes 1 or 2 cycles
 - Better for caches not tied directly to processor (i.e., L2)
 - Used in MIPS R1000 L2 cache, similar in UltraSPARC

ECE568/Koren Part.13 .14

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

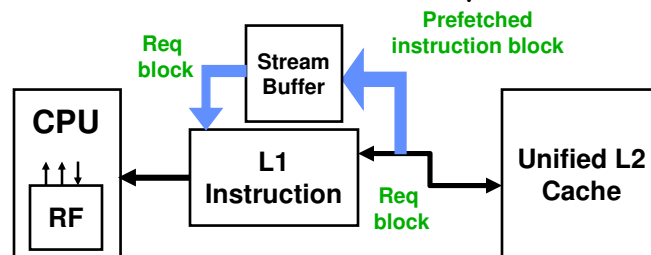
Reducing Misses by Hardware Prefetching of Instructions & Data

◆ Instruction Prefetching

- Alpha 21064 fetches 2 blocks on a miss
- Extra block placed in "stream buffer" - check upon miss
- If hit in stream buffer, move stream buffer block into cache and prefetch next block (i+2)

◆ Works with data blocks too:

- 1 data stream buffer reduced by 25% misses from 4KB cache; 4 streams reduced by 43%; 8 streams reduced by 50% to 70% misses from 64KB, 4-way set associative cache



ECE568/Koren Part.13 .15

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Hardware Prefetching

◆ Strided prefetch

- If observe sequence of accesses to block b , $b+N$, $b+2N$, then prefetch $b+3N$ etc.
- IBM Power 5 [2003] supports eight independent streams of strided prefetch per processor, prefetching 12 lines ahead of current access

◆ Intel processors use

- Adjacent cache line prefetch from L2 to L1
- HW prefetchers: instructions from L2 to L1 based on Branch prediction unit (BTB); Data and instruction from memory to L2 - triggered by successive cache misses and detection of a stride in accesses

◆ Prefetching relies on having extra memory bandwidth

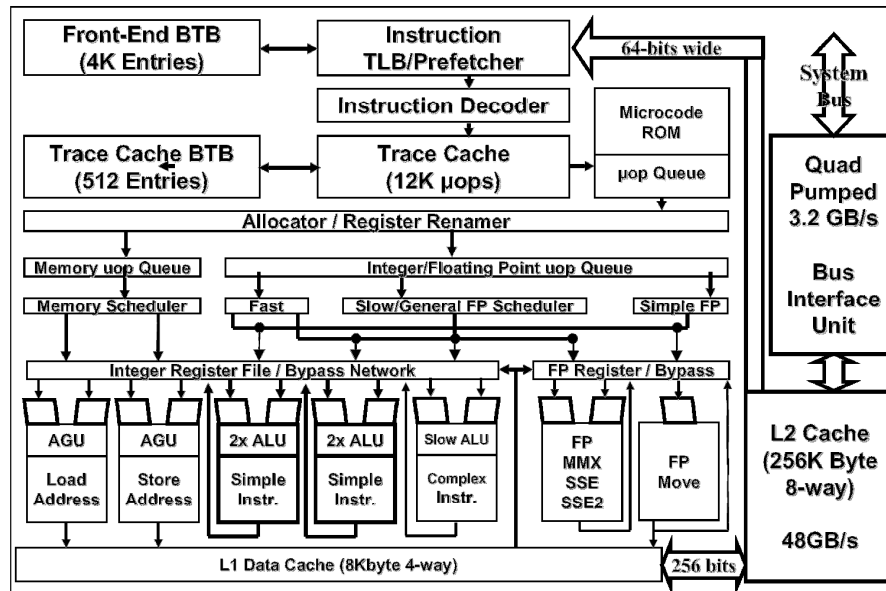
- Intel processors allow disabling of HW prefetch

ECE568/Koren Part.13 .16

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Pentium 4 Microarchitecture



ECE568/Koren Part.13.17

Source: G. Hinton et al. "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal, Q1, 2001.

Reducing Misses by Software Prefetching Data

◆ Data Prefetching comes in two flavors:

- Load data into register (HP PA-RISC loads)
 - » Binding prefetch: Must be correct address and register
- Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
 - » Non-Binding prefetch: Can be incorrect

```

for(i=0; i < N; i++) {
    prefetch( &a[i + 1] );
    prefetch( &b[i + 1] );
    SUM = SUM + a[i] * b[i];
}
    
```

◆ Special prefetching instructions cannot cause faults; a form of speculative execution

◆ Issuing Prefetch Instructions takes time

- Is cost of issuing prefetch < savings in reduced misses ?
- Higher superscalar reduces difficulty of issue bandwidth

ECE568/Koren Part.13.18

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Reducing Misses by Compiler Optimizations

- ◆ Cache misses reduced by 75% on 8KB direct mapped cache using software
- ◆ Instructions
 - Reorder procedures in memory so as to reduce conflict misses
 - Through profiling to detect conflicts
- ◆ 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

ECE568/Koren Part.13 .19

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Merging Arrays Example

```
/* Before: 2 sequential arrays */      /* After: 1 array */
int val[SIZE];                          struct merge {
int key[SIZE];                          int val;
                                          int key;
                                          };
```

Reduce conflicts between val & key; improve spatial locality

Loop Interchange Example

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

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

ECE568/Koren Part.13 .20

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Loop Fusion Example

```
for(i=0; i < N; i++)  
    a[i] = b[i] * c[i];
```

```
for(i=0; i < N; i++)  
    d[i] = a[i] * c[i];
```

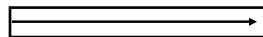


```
for(i=0; i < N; i++)  
{  
    a[i] = b[i] * c[i];  
    d[i] = a[i] * c[i];  
}
```

Improve spatial locality

Blocking Example - Matrix multiplication

```
/* 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 $N \times N$ elements of $z[]$
- Read N elements of 1 row of $y[]$ repeatedly
- Write N elements of 1 row of $x[]$



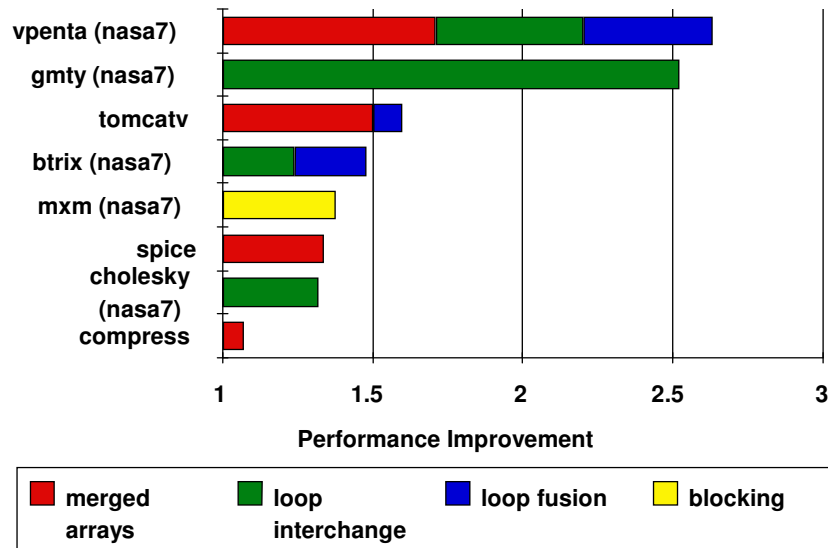
◆ Capacity Misses a function of N & Cache Size:

- $2N^3 + N^2 \Rightarrow$ (assuming no conflict; otherwise ...)

◆ Idea: compute on $B \times B$ submatrix that fits

- Capacity Misses from $2N^3 + N^2$ to $N^3/B + 2N^2$

Summary of Compiler Optimizations to Reduce Cache Misses



ECE568/Koren Part.13 .23

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Reducing Miss Penalty: (1) Read Priority over Write on Miss

- ◆ **Write-through w/ write buffers => RAW hazards with main memory reads on cache misses**
 - If simply wait for write buffer to empty, might increase read miss penalty (MIPS 1000 by 50%)
 - Check write buffer contents before read; if no conflicts, let the memory access continue
- ◆ **Write-back: add 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

ECE568/Koren Part.13 .24

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Reduce Miss Penalty: (2) 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 => tend to want next sequential word, so not clear if benefit by early restart



ECE568/Koren Part.13 .25

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

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

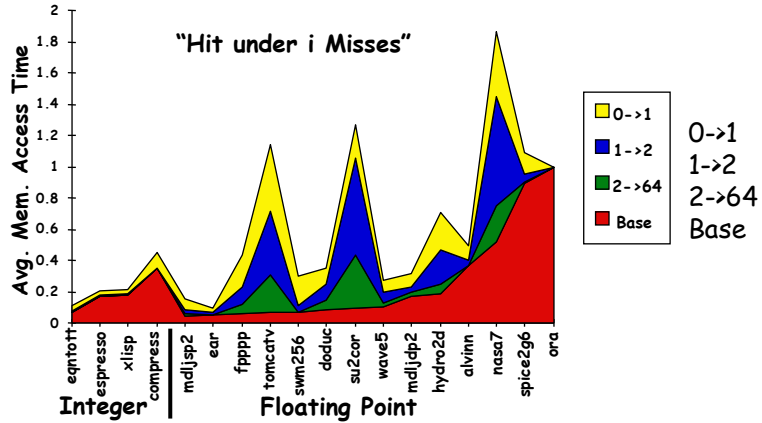
- ◆ *Non-blocking cache* or *lockup-free cache*: allow data cache to continue to supply cache hits during a miss
 - requires out-of-order execution
- ◆ "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- ◆ "*hit under multiple miss*" or "*miss under 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

ECE568/Koren Part.13 .26

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Benefits 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

ECE568/Koren Part.13 .27

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

(4): Add a second-level cache

◆ 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*
- Global Miss Rate is what matters

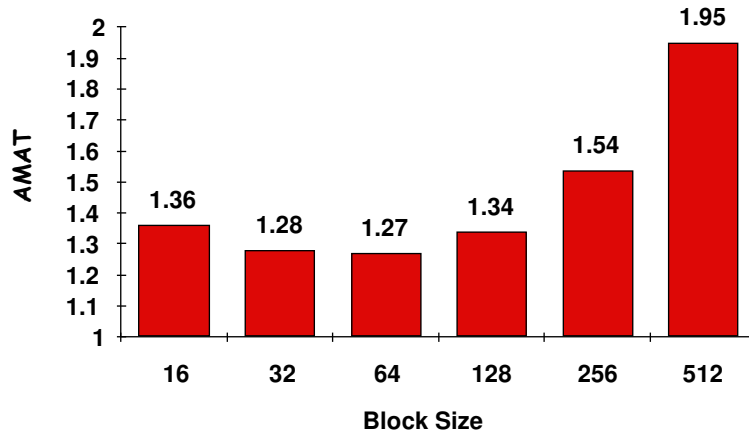
ECE568/Koren Part.13 .28

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

L2 cache block size & AMAT

Relative to CPU Time



◆ 32KB L1, 8 byte path to memory

ECE568/Koren Part.13 .29

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Cache Optimization Summary

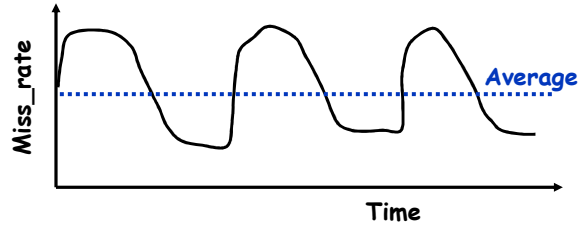
	<i>Technique</i>	<i>MRate</i>	<i>MP</i>	<i>HT</i>	<i>Complexity</i>
miss rate	Larger Block Size	+	-		0
	Higher Associativity	+		-	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
miss penalty	Priority to Read Misses		+		1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2

ECE568/Koren Part.13 .30

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Cache Misses vs. Time



- ◆ Cache misses are not distributed uniformly over time but tend to cluster
- ◆ Reload transients can be time consuming
- ◆ Each process (when run in isolation) has its own **footprint** in the cache

ECE568/Koren Part.13 .31

Copyright 2011 Koren UMass

Cache Footprint of Process A

	1	2	3	4				
Set 0	A	A	A					---
Set 1	A	A	A	A				---
Set 2	A	A	A	A	A			---
Set 3	A	A						---
Set 4	A	A	A	A	A	A		---
Set 5	A							---
Set 6	A	A	A					---
Set 7	A	A						---

- ◆ Four-way set associative, 8 sets
- ◆ Size of A's footprint =

ECE568/Koren Part.13 .32

H. Stone, "High Performance Computer Architecture," AW 1993

Footprint Distribution for k-way S.A. Cache

- ◆ Find the probability of having more than k blocks per set (total of N sets)
- ◆ Assume blocks distributed uniformly to sets
 - Prob {a block referenced by A falls into a given set} = $p = 1/N$
 - S_A is the size of A's footprint
 - Use Binomial model with p and $(1-p)$
 - Prob {exactly i blocks of A in a single set} =

$$\text{Prob}\{X=i\} = \binom{S_A}{i} p^i (1-p)^{S_A - i} \quad \text{for } i < k$$

$$\sum_{j=k}^{S_A} \binom{S_A}{j} p^j (1-p)^{S_A - j} \quad \text{for } i=k$$

Similar distribution for process B with footprint S_B

A & B compete for the cache

A then B

	1	2	3	4				
Set 0	B	B	A	A	A			----
Set 1	B	B	B	A	A	A	A	----
Set 2	B	A	A	A	A	A		----
Set 3	B	B	B	B	A	A		----
Set 4	B	A	A	A	A	A	A	----
Set 5	B	B	B	A				----
Set 6	B	B	A	A	A			----
Set 7	B	B	B	B	A	A		----

Most recently used items appear on the left

B then A

	1	2	3	4				
Set 0	A	A	A	B	B			----
Set 1	A	A	A	A	B	B	B	----
Set 2	A	A	A	A	A	B		----
Set 3	A	A	B	B	B	B		----
Set 4	A	A	A	A	A	A	B	----
Set 5	A	B	B	B				----
Set 6	A	A	A	B	B			----
Set 7	A	A	B	B	B	B		----

Size of Reload Transient for Set 1

- ◆ X (Y) denotes # of blocks in A 's (B 's) footprint in Set 1
- ◆ $Z=X+Y$ total number of blocks in set 1
- ◆ If $Z \leq k$ set 1 contributes nothing to reload transient:
- ◆ $\text{Prob} \{ Z \leq k \} = \sum_{i=0}^k \left(\text{Prob}\{x=i\} \sum_{j=0}^{k-i} \text{Prob}\{Y=j\} \right)$
- ◆ $X=i; Y=0,1,\dots, k-i$
- ◆ If X and Y Binomially distributed ($p=1/N$), $Z=X+Y$ is too; this is the prob. of having k or less among $S_A + S_B$
- ◆ A reload transient occurs when $Z > k$; let W blocks of A be overwritten by B
- ◆ $\text{Prob} \{ W=i \} = \sum_{j=i}^k \text{Prob} \{ X=j \} \text{Prob} \{ Y=k+i-j \}; 1 \leq i \leq k$

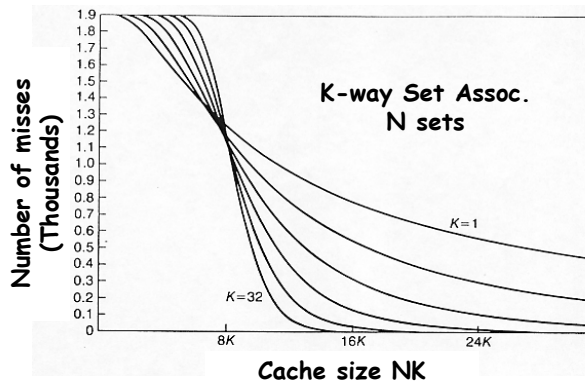
ECE568/Koren Part.13 .35

Copyright 2011 Koren UMass

Cache Reload Transient

- ◆ Transient to reload A is: $S_A - \{ \# \text{ of } A\text{'s blocks that remain in cache when } A \text{ resumes} \} = S_A - N (\text{Avg}\{X\} - \text{Avg}\{W\}) = S_A - N (E\{X\} - E\{W\})$
 where $E\{X\} = \sum_{i=0}^k i \text{Prob} \{ X=i \}$

$S_A=1900$
 $S_B=7900$



ECE568/Koren Part.13 .36

H. Stone, "High Performance Computer Architecture," AW 1993