



Caches

- How to improve performance -

Donatella Sciuto

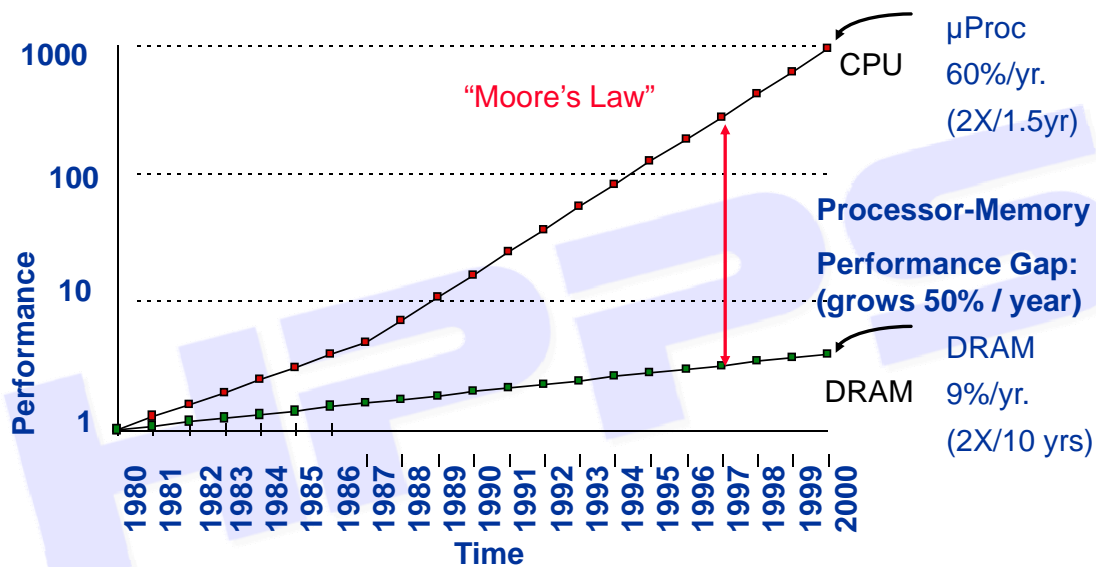
HPPS



Outline

- Introduction
- How to reduce cache misses
- How to reduce miss penalty
- How to reduce hit time

Who Cares About the Memory Hierarchy?



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

3

μ-LAB

POLITECNICO DI MILANO

Cache performance

- Miss-oriented Approach to Memory Access:

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

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

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

4

μ-LAB

POLITECNICO DI MILANO

Cache performance

Separating out Memory component entirely

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

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

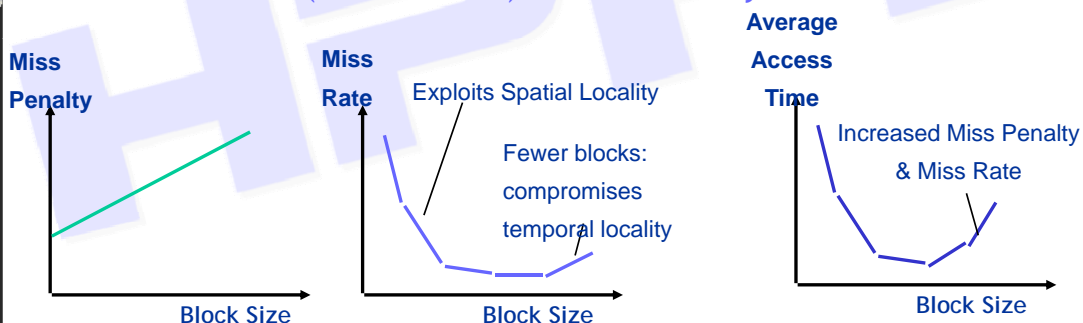
$$AMAT = HitTime + MissRate \times MissPenalty$$

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

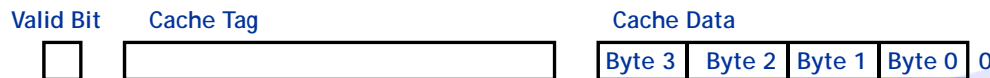
Block Size Tradeoff

- In general, larger block size take advantage of spatial locality **BUT**:
 - ▶ Larger block size means larger miss penalty:
 - Takes longer time to fill up the block
 - ▶ If block size is too big relative to cache size, miss rate will go up
 - Too few cache blocks
- In general, Average Access Time:

$$= Hit\ Time \times (1 - Miss\ Rate) + Miss\ Penalty \times Miss\ Rate$$



Extreme Example: single line



- Cache Size = 4 bytes Block Size = 4 bytes
 - ▶ Only ONE entry in the cache
- If an item is accessed, likely that it will be accessed again soon
 - ▶ But it is unlikely that it will be accessed again immediately!!!
 - ▶ The next access will likely to be a miss again
 - Continually loading data into the cache but discard (force out) them before they are used again
 - Worst nightmare of a cache designer: **Ping Pong Effect**

Extreme Example: single line

- **Conflict Misses** are misses caused by:
 - ▶ Different memory locations mapped to the same cache index
 - Solution 1: make the cache size bigger
 - Solution 2: Multiple entries for the same Cache Index

Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference): first access to a block
 - ▶ “Cold” fact of life: not a whole lot you can do about it
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.
 - ▶ Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant
 - ▶ *(Misses in even an Infinite Cache)*

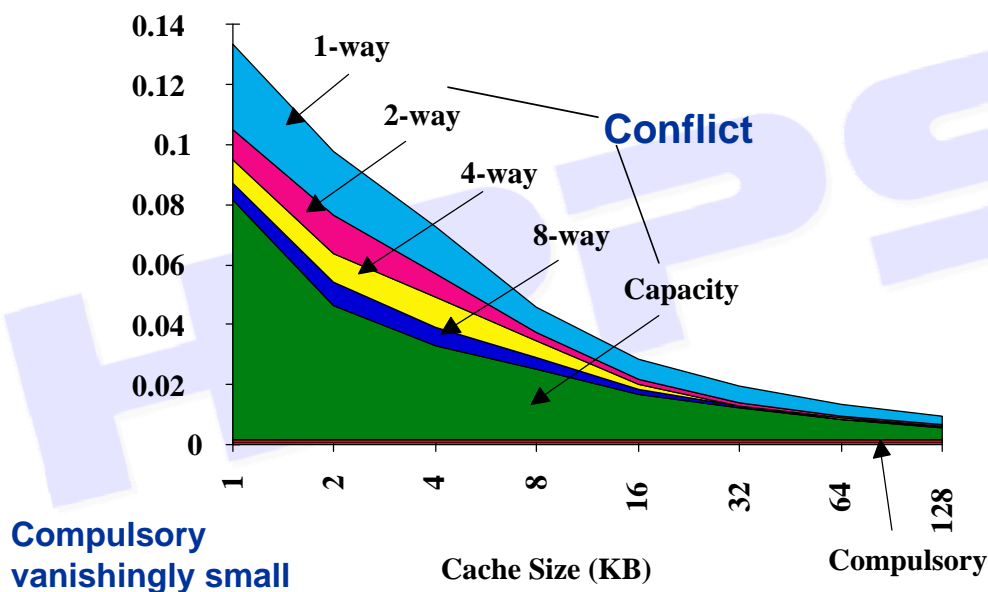
Sources of cache misses

- **Conflict** (collision):
 - ▶ Multiple memory locations mapped to the same cache location
 - ▶ 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)*
 - ▶ Solution 1: increase cache size
 - ▶ Solution 2: increase associativity

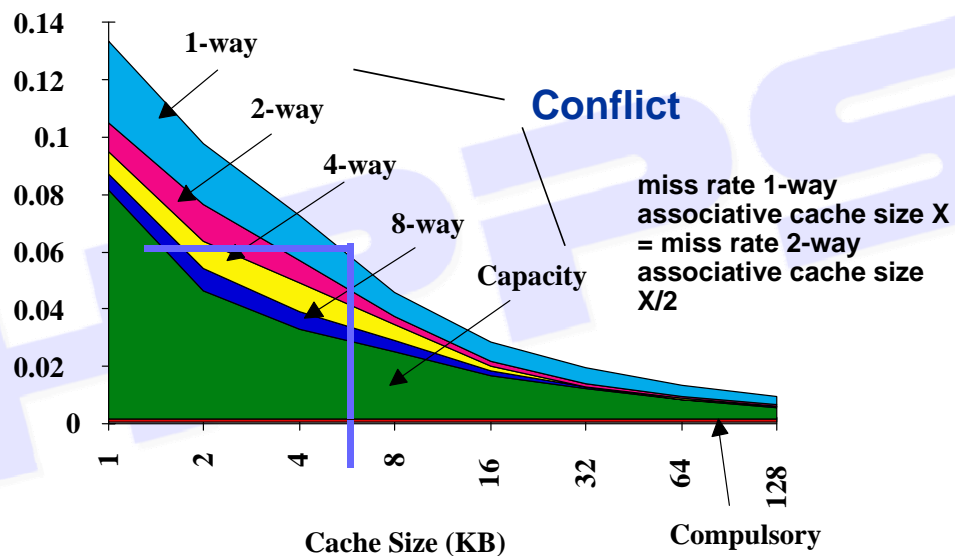
Sources of cache misses

- **Capacity:**
 - ▶ Cache cannot contain all blocks access by the program
 - ▶ 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)*
 - ▶ Solution: increase cache size
- **Coherence (Invalidation):** other process (e.g., I/O) updates memory
 - ▶ Misses caused by cache coherence

3Cs Absolute Miss Rate (SPEC92)

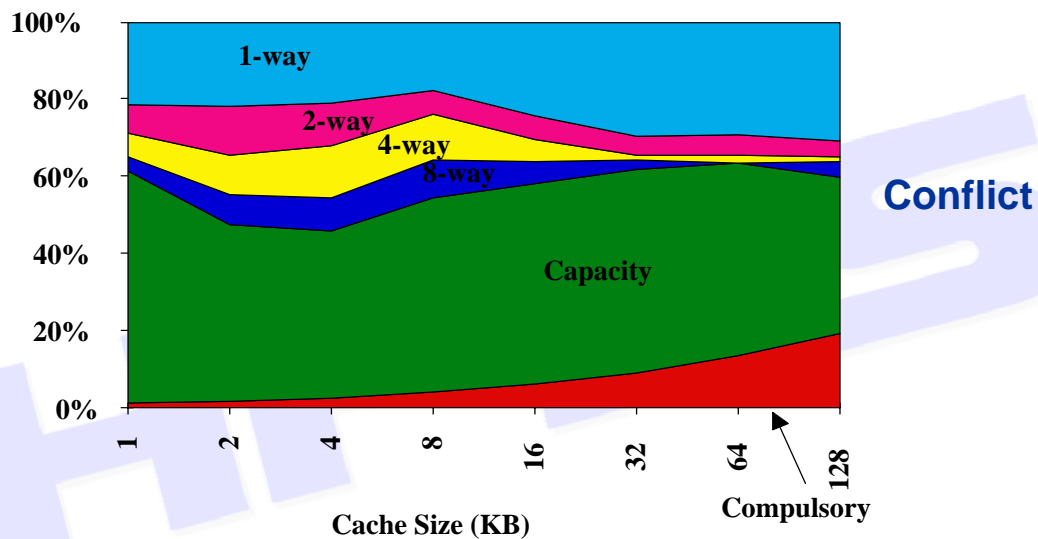


2:1 Cache Rule



13

3Cs Relative Miss Rate



Flaws: for fixed block size

Good: insight => invention

14

Source of Cache Misses Quiz

Assume constant cost.

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size: Small, Medium, Big?			
Compulsory Miss			
Conflict Miss			
Capacity Miss			
Coherence Miss			

Choices: Zero, Low, Medium, High, Same

Sources of Cache Misses Answer

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Coherence Miss	Same	Same	Same

Note:

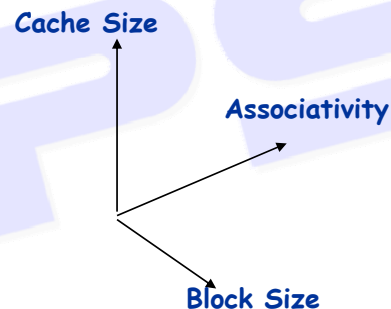
If you are going to run “billions” of instruction,
Compulsory Misses are insignificant.

Summary: caches

- Write Policy:
 - ▶ Write Through: needs a write buffer.
 - ▶ Write Back: control can be complex
- Today CPU time is a function of (ops, cache misses) vs. just $f(\text{ops})$: What does this mean to Compilers, Data structures, Algorithms?

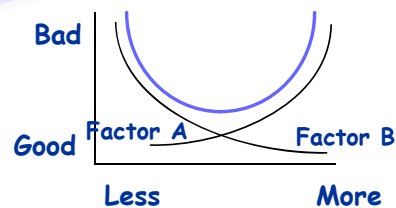
Summary: the cache design space

- Several interacting dimensions
 - ▶ cache size
 - ▶ block size
 - ▶ associativity
 - ▶ replacement policy
 - ▶ write-through vs write-back



Summary: the cache design space

- The optimal choice is a compromise
 - ▶ depends on access characteristics
 - workload
 - use (I-cache, D-cache, TLB)
 - ▶ depends on technology / cost
- Simplicity often wins



Improving Cache Performance

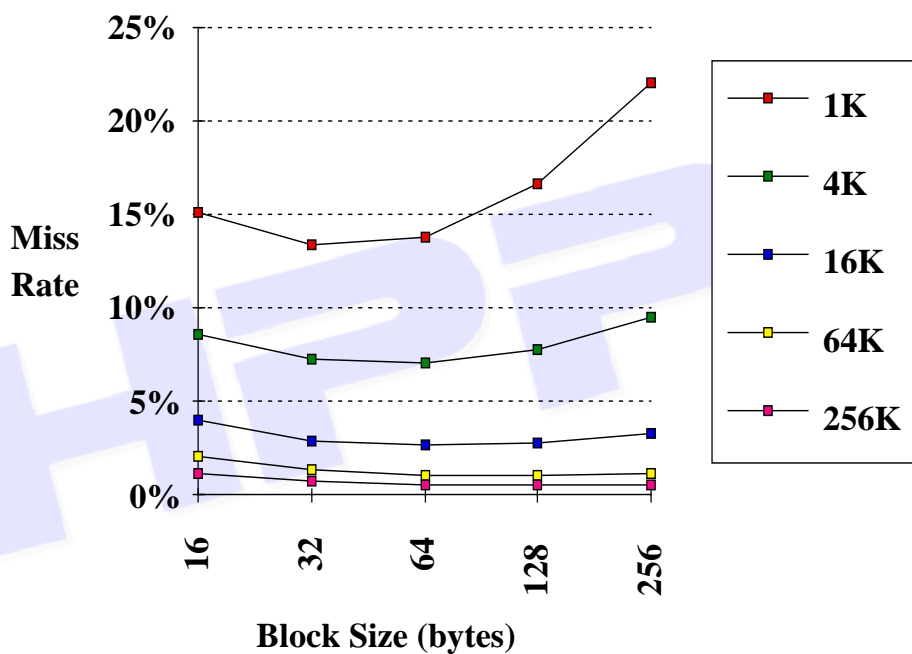
$$\text{AMAT} = \text{Hit time} + \text{miss rate} \times \text{miss penalty}$$

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

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?

1. Reduce Misses via Larger Block Size



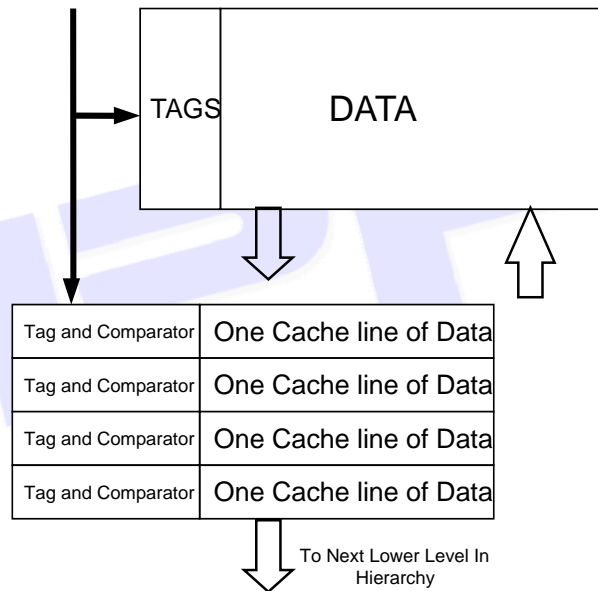
2. Reduce Misses via Higher Associativity

- 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%

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

Victim cache



25

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?



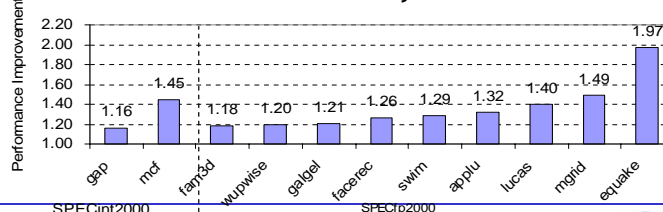
26

4. Reducing Misses via "Pseudo-Associativity"

- 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

5. Reducing Misses by Hardware Prefetching of Instructions & Data

- Prefetching relies on having extra memory bandwidth that can be used without penalty
- Instruction Prefetching
 - ▶ Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
 - ▶ Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer
 - ▶ On miss check stream buffer
- Data Prefetching
 - ▶ Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
 - ▶ Prefetching invoked if 2 successive L2 cache misses to a page, if distance between those cache blocks is < 256 bytes



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)

7. Reducing Misses by Compiler Optimizations

- 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 structures */
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[]

Blocking Example

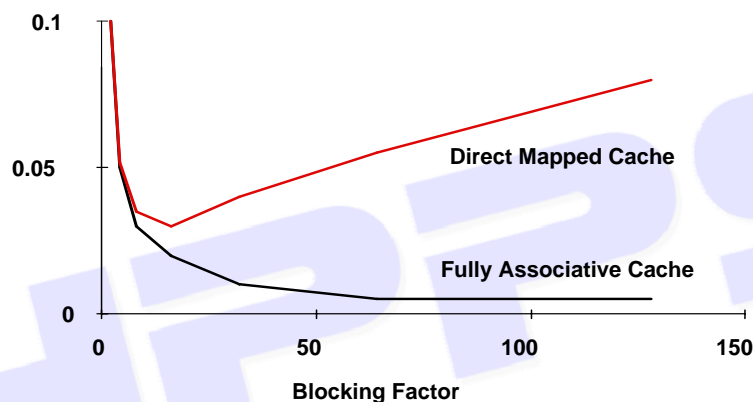
- Capacity Misses a function of N & Cache Size:
 - ▶ 3 NxN_{x4} => no capacity misses; otherwise ...
- Idea: compute on BxB submatrix that fits

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;
        };
```

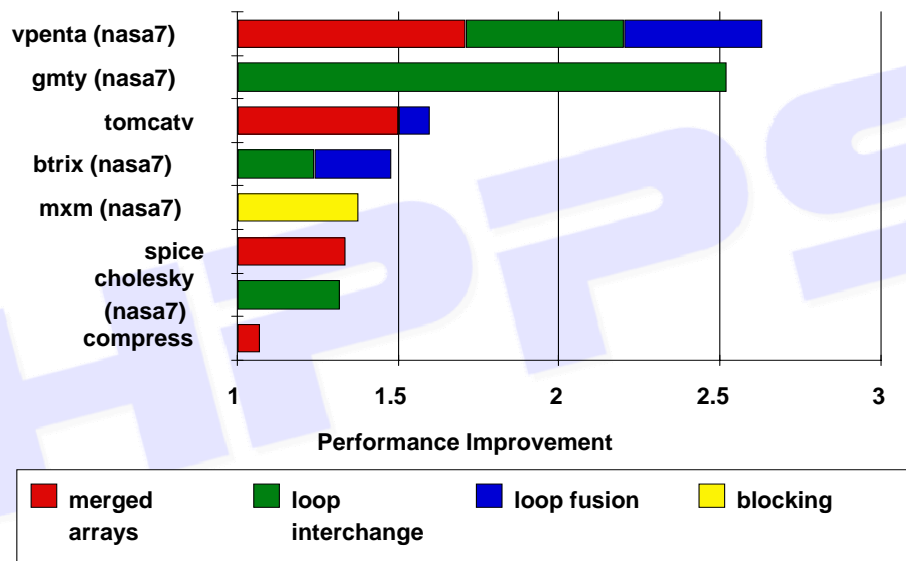
- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- Conflict Misses Too?

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

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



Summary

Reducing Miss Rate

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

Improving Cache Performance

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

Write Policy: Write-Through vs Write-Back

- Write-through: all writes update cache and underlying memory/cache
 - ▶ Can always discard cached data - most up-to-date data is in memory
 - ▶ Cache control bit: only a *valid* bit
- Write-back: all writes simply update cache
 - ▶ Can't just discard cached data - may have to write it back to memory
 - ▶ Cache control bits: both *valid* and *dirty* bits

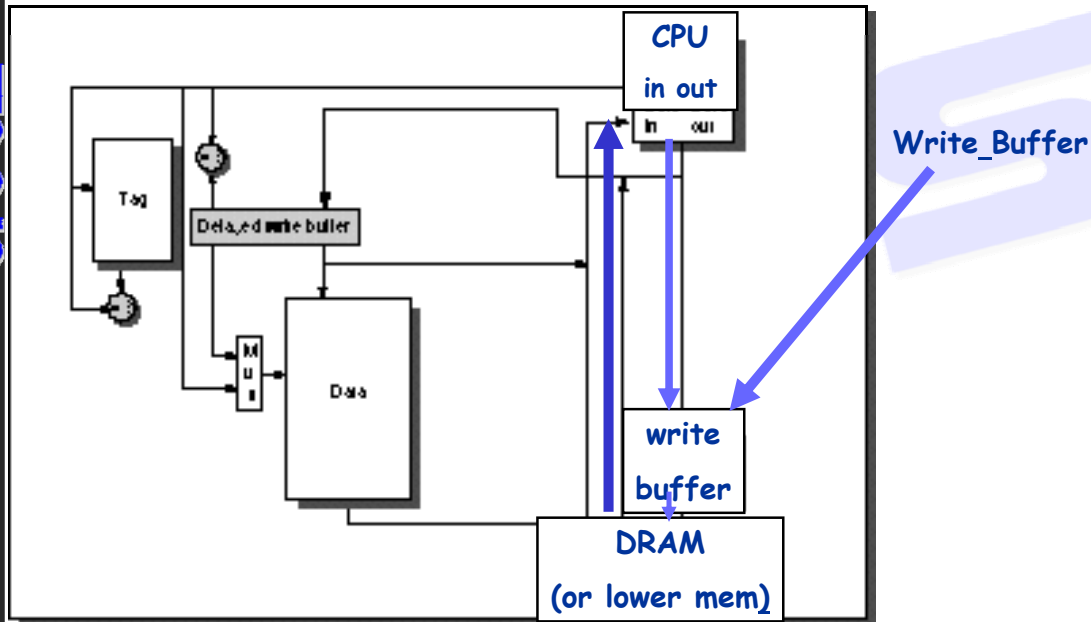
Write Policy: Write-Through vs Write-Back

- Other Advantages:
 - ▶ Write-through:
 - memory (or other processors) always have latest data
 - Simpler management of cache
 - ▶ Write-back:
 - much lower bandwidth, since data often overwritten multiple times
 - Better tolerance to long-latency memory?

Write Policy 2: Write Allocate vs Non-Allocate

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

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



45

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

46

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

- 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 arrives, send it to the CPU and let the CPU continue execution
 - Spatial locality \Rightarrow tend to want next sequential word, so not clear size of benefit of just early restart
 - ▶ *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*
 - Long blocks more popular today \Rightarrow Critical Word 1st Widely used

Increasing Cache Bandwidth by Pipelining

- Pipeline cache access to maintain bandwidth, but higher latency
- Instruction cache access pipeline stages:
 - 1: Pentium
 - 2: Pentium Pro through Pentium III
 - 4: Pentium 4
 - \Rightarrow greater penalty on mispredicted branches
 - \Rightarrow more clock cycles between the issue of the load and the use of the data

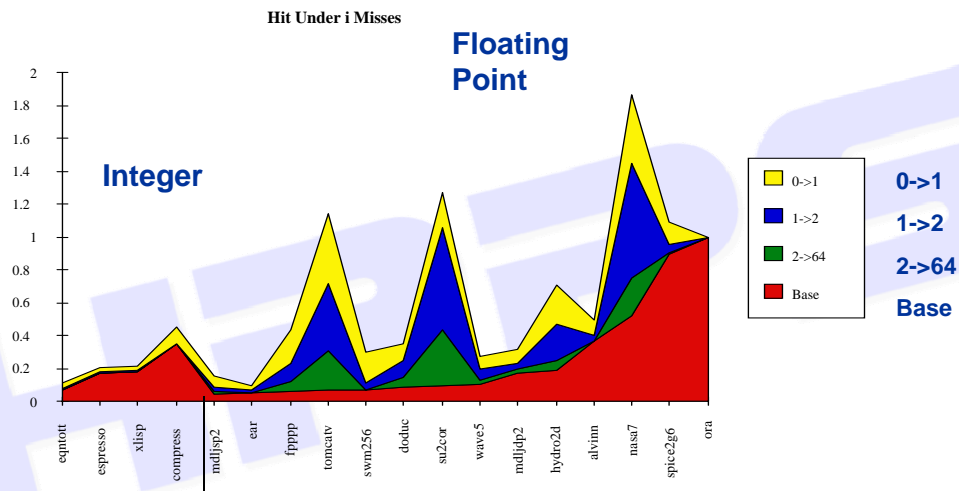
4. Reduce Miss Penalty: Non-blocking Caches to reduce stalls on misses

- Non-blocking cache or lockup-free cache allow data cache to continue to supply cache hits during a miss
 - ▶ requires out-of-order execution CPU
 - ▶ requires multi-bank memories
- “hit under miss” reduces the effective miss penalty by working during miss vs. ignoring CPU requests

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

- “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

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, SPEC 92

53

μ -LAB

POLITECNICO DI MILANO

5th Miss Penalty Reduction: Second Level Cache

• L2 Equations

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

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

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

54

μ -LAB

POLITECNICO DI MILANO

5th Miss Penalty Reduction: Second Level Cache

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} x Miss Rate_{L2})
- ▶ Global Miss Rate is what matters

Comparing Local and Global Miss Rates

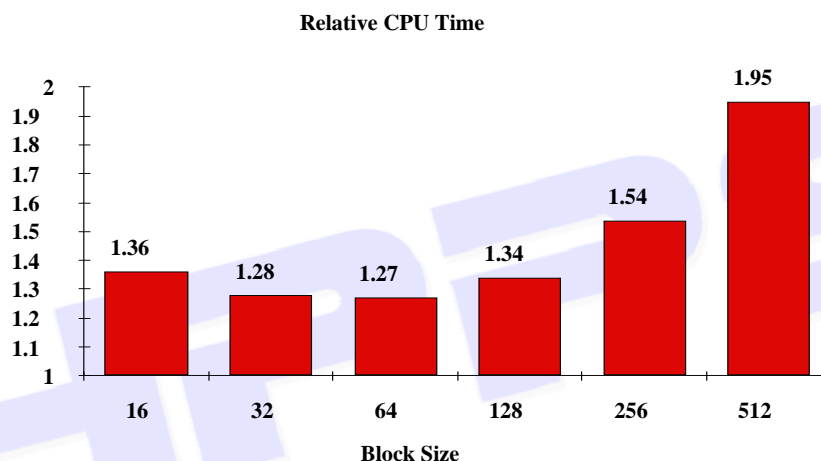
- 32 KByte 1st level cache;
Increasing 2nd level cache
- Global miss rate close to single level cache rate
provided L2 >> L1
- Don't use local miss rate
- L2 not tied to CPU clock cycle!
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction

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

59



Cache Optimization Summary

	<i>MR</i>	<i>MP</i>	<i>HT</i>	<i>Complexity</i>
<i>Technique</i>				
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
Priority to Read Misses		+		1
Subblock Placement		+	+	1
Early Restart & Critical Word 1st		+		2
Non-Blocking Caches		+		3
Second Level Caches		+		2

miss rate

miss penalty

60



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

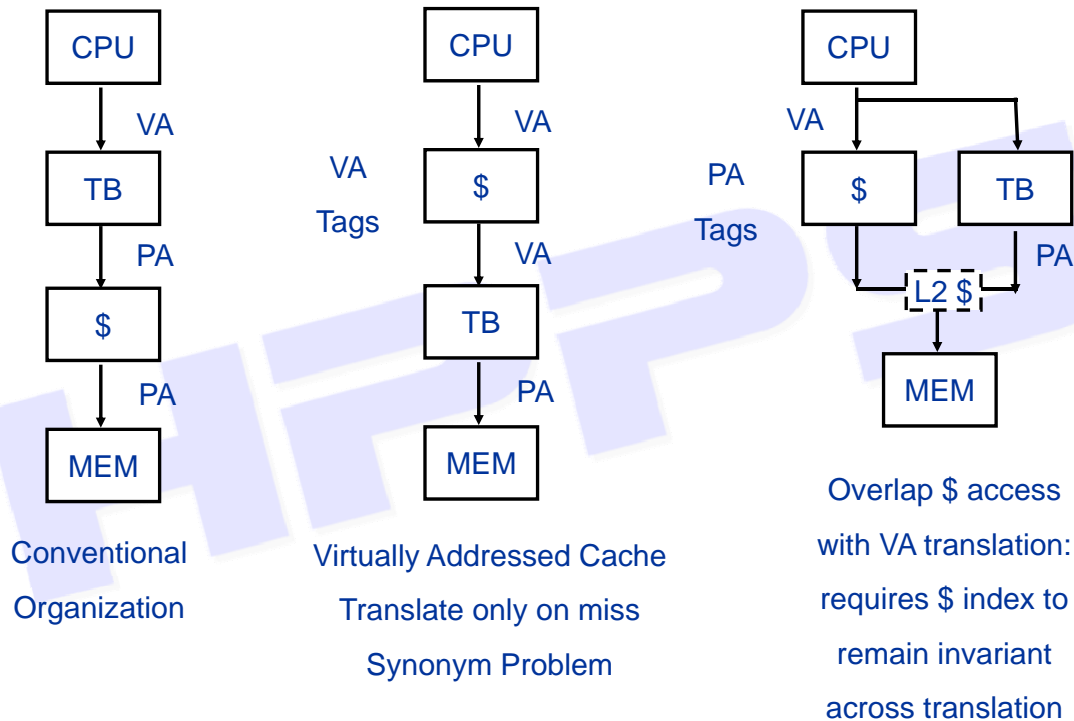
2. Fast hits by Avoiding Address Translation

- 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

2. Fast hits by Avoiding Address Translation

- Solution to aliases
 - ▶ HW guarantees that every cache block has unique physical address
 - ▶ SW guarantee : lower n bits must have same address; as long as 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



65

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

66

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)
Add
Sub
Store r4, (r3)

Check r1
--
--
M[r1] ← r2 &
check r3

- The “[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.

4. Fast Writes on Misses Via Small Subblocks

- ▶ *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

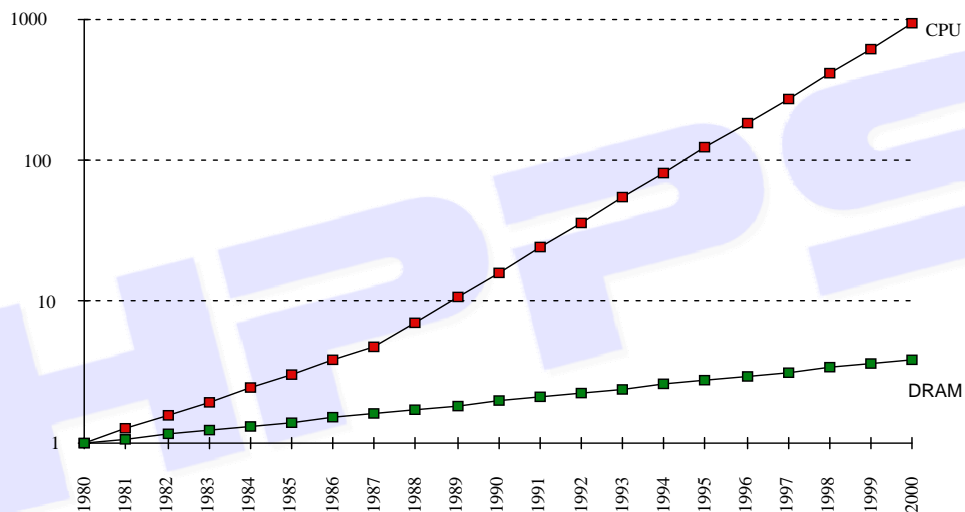
Fast Hit times via Trace Cache (Pentium 4 only; and last time?)

- Find more instruction level parallelism?
How avoid translation from x86 to microops?
- Trace cache in Pentium 4
 1. Dynamic traces of the executed instructions vs. static sequences of instructions as determined by layout in memory
 - ▶ Built-in branch predictor
 2. Cache the micro-ops vs. x86 instructions
 - ▶ Decode/translate from x86 to micro-ops on trace cache miss
- + ⇒ better utilize long blocks (don't exit in middle of block, don't enter at label in middle of block)
- ⇒ complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size
- ⇒ instructions may appear multiple times in multiple dynamic traces due to different branch outcomes

Cache Optimization Summary

Technique	MR	MP	HT	Complexity
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
Priority to Read Misses		+		1
Subblock Placement		+	+	1
Early Restart & Critical Word 1st		+		2
Non-Blocking Caches		+		3
Second Level Caches		+		2
Small & Simple Caches	-		+	0
Avoiding Address Translation			+	2
Pipelining Writes			+	1

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



Technique	Hit Time	Band width	Miss penalt y	Miss rate	HW cost/ complexit y	Comment
Small and simple caches	+			-	0	Trivial; widely used
Way-predicting caches	+				1	Used in Pentium 4
Trace caches	+				3	Used in Pentium 4
Pipelined cache access	-	+			1	Widely used
Nonblocking caches		+	+		3	Widely used
Banked caches		+			1	Used in L2 of Opteron and Niagara
Critical word first and early restart			+		2	Widely used
Merging write buffer			+		1	Widely used with write through
Victim Caches			-	+	1	Fairly Simple and common
Compiler techniques to reduce cache misses				+	0	Software is a challenge; some computers have compiler option
Hardware prefetching of instructions and data			+	+	2 instr., 3 data	Many prefetch instructions; AMD Opteron prefetches data
Compiler-controlled prefetching			+	+	3	Needs nonblocking cache; in many CPUs

Concluding remarks

- Memory wall inspires optimizations since so much performance lost there
 - Reducing hit time: Small and simple caches, Way prediction, Trace caches
 - Increasing cache bandwidth: Pipelined caches, Multibanked caches, Nonblocking caches
 - Reducing Miss Penalty: Critical word first, Merging write buffers
 - Reducing Miss Rate: Compiler optimizations
 - Reducing miss penalty or miss rate via parallelism: Hardware prefetching, Compiler prefetching

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)?
- What does this mean for
 - ▶ Compilers?, Operating Systems?, Algorithms? Data Structures?