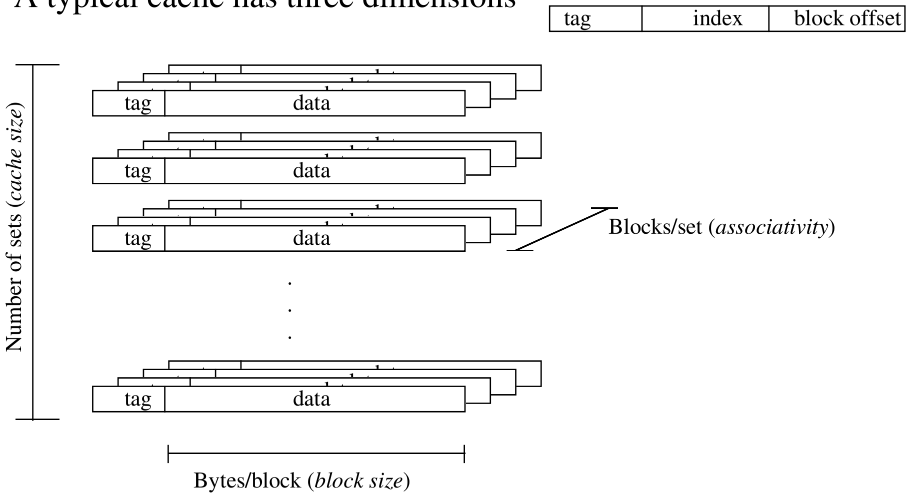
Cache Organization -- Overview

- A typical cache has three dimensions



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. These are also called *cold start misses* or *first reference misses*.
  - *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.
  - *Conflict*—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*.

How To Measure

Misses in infinite cache

Non-compulsory misses in size X fully associative cache

Non-compulsory, non-capacity misses

3Cs Absolute Miss Rate

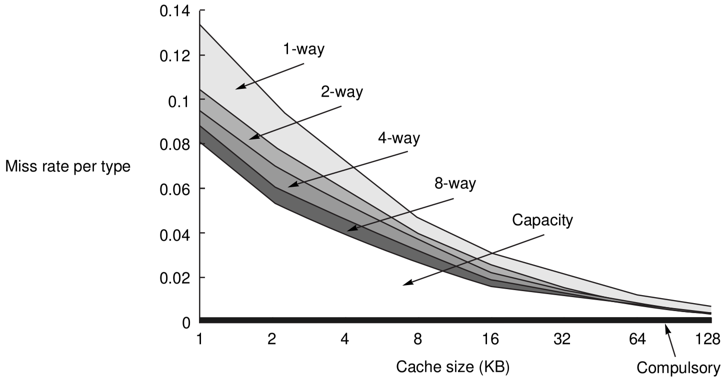


FIGURE 5.10 Total miss rate for each size cache according to three C's for the data in Figure 5.9.

How To Reduce Misses?

- Compulsory Misses?
- Capacity Misses?
- Conflict Misses?
- What can the compiler do?

---

1. Reduce Misses via Larger Block Size

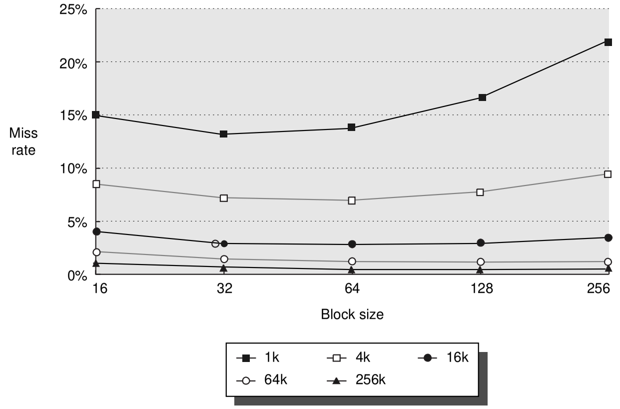


FIGURE 5.11 Miss rate versus block size for five different-sized caches.

- 16K cache, miss penalty for 16-byte block = 42, 32-byte is 44, 64-byte is 48. Miss rates are 3.94, 2.87, and 2.64%?

---

2. Reduce Misses via Higher Associativity

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

---

Example: Avg. Memory Access Time vs. Miss Rate

- Example: assume CT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CT direct mapped

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
16	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	**1.79**
64	1.70	1.60	1.57	**1.59**
128	1.50	1.45	1.42	**1.44**

3. Reducing Misses via Victim Cache

- How to get 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

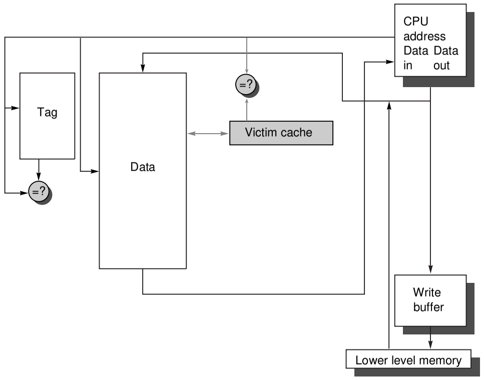
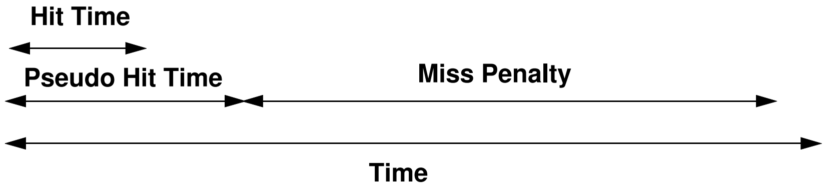


FIGURE 5.15 Placement of victim cache in the memory hierarchy.

---

4. Reducing Misses via Pseudo-Associativity

- Combines fast hit time of Direct Mapped and the lower conflict misses of a 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

---

5. Reducing Misses by HW Prefetching of Instruction & 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 extra memory bandwidth that can be used without penalty

---

6. Reducing Misses by SW 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?

7. Reducing Misses by Various Compiler Optimizations

- Instructions
  - Reorder procedures in memory so as to reduce misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced cache misses by 75% on 8KB direct mapped cache with 4 byte blocks
- 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 */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key

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

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

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:
  - wost case => $2N^3 + N^2$.
- Idea: compute on BxB submatrix that fits in cache

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;
     };
```

- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- B called *Blocking Factor*
- Conflict Misses Are Not As Easy...

Summary of Compiler Optimizations to Reduce Cache Misses

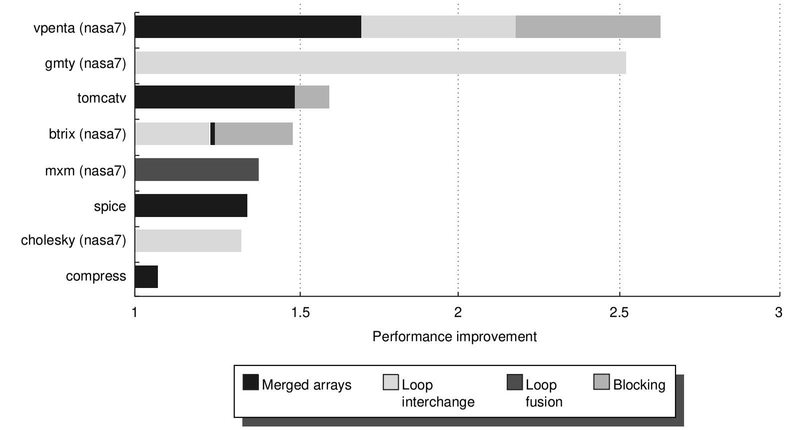


FIGURE 5.17 Lebeck and Wood [1994] performed the four optimizations in this section by hand on three SPEC92 programs and five separate portions of the nasa7 benchmark.

Key Points

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

- 3 Cs: Compulsory, Capacity, Conflict Misses
- 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
- Next: reducing Miss penalty

Improving Cache Performance

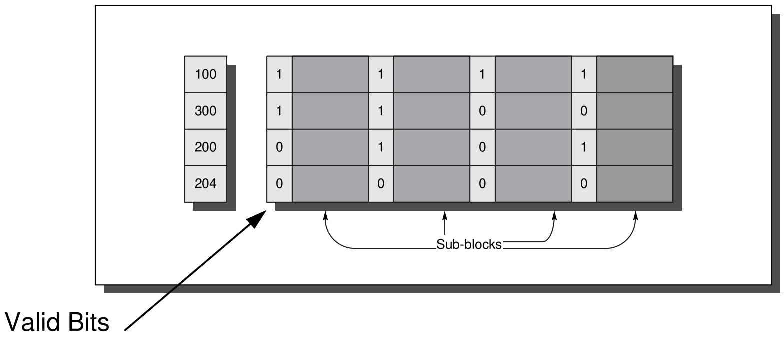
1. Reduce the miss rate,
2. *Reduce the miss penalty*, or
3. Reduce the time to hit in the cache.

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

- 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 by 50%
- Check write buffer contents before read;
  if no conflicts, let the memory access continue
- Write Back?
  - Read miss may require write of 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 stalls less since it can restart as soon as read completes

2. Subblock Placement to Reduce Miss Penalty

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



3. 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*
- Most useful with large blocks,
- Spatial locality a problem; often we next want the next sequential word soon, so not always a benefit (early restart).

4. Non-blocking Caches to reduce stalls on misses

- *Non-blocking cache* or *lockup-free cache* allowing the data cache to continue to supply cache hits during a miss
- "*hit under miss*" reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the CPU
- "*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
- assumes "stall on use" not "stall on miss" which works naturally with dynamic scheduling, but can also work with static.

Value of Hit Under Miss for SPEC

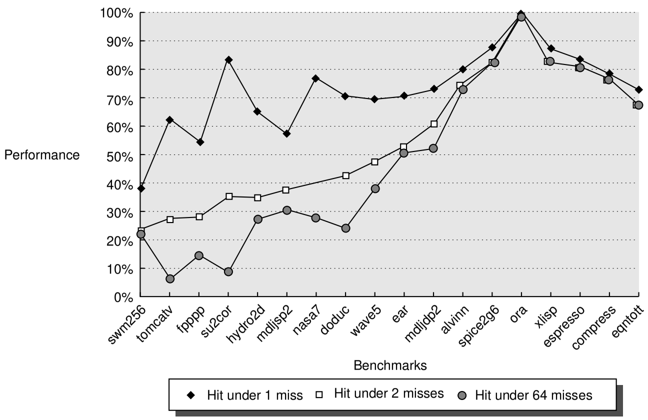
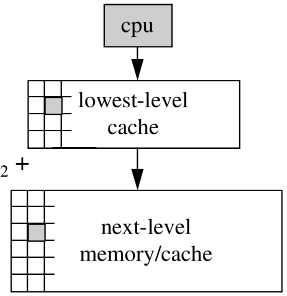


FIGURE 5.22 Ratio of the average memory stall time for a blocking cache to hit-under-miss schemes as the number of outstanding misses is varied for 18 SPEC92 programs.

5th Miss Penalty Reduction: Second Level Cache



- L2 Equations

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

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

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

- 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* ($\text{Miss Rate}_{L1} \text{ x Miss Rate}_{L2}$)

Multi-level Caches, cont.

- L1 cache local miss rate 10%, L2 local miss rate 40%. What are the global miss rates?
- L1 highest priority is fast hit time. L2 typically low miss rate.
- Design L1 and L2 caches in concert.
- Property of inclusion -- if it is in L2 cache, it is guaranteed to be in one of the L1 caches -- simplifies design of consistent caches.
- L2 cache can have different associativity (good idea?) or block size (good idea?) than L1 cache.

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

---

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)
  - Second Level Cache
- Can be applied recursively to Multilevel Caches
  - Danger is that time to DRAM will grow with multiple levels in between

---

Review: Improving Cache Performance

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

---

1. Fast Hit times via Small and Simple Caches

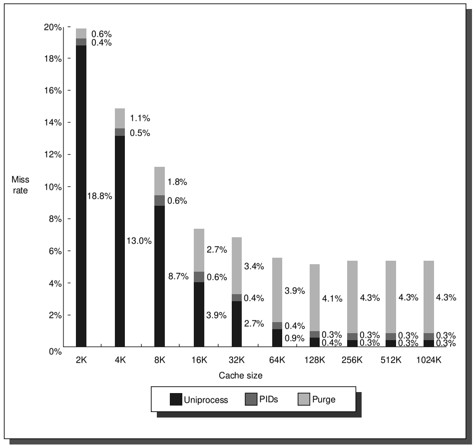
- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache
- Direct Mapped, on chip

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 that 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

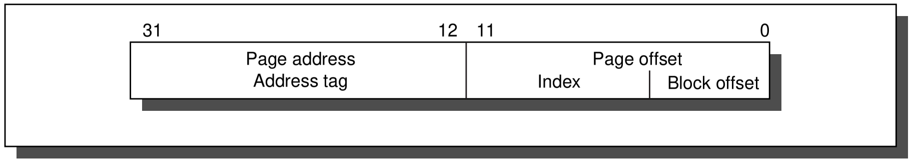
2. Avoiding Translation: Process ID impact

- 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. Avoiding Translation: Index with Physical Portion of Address

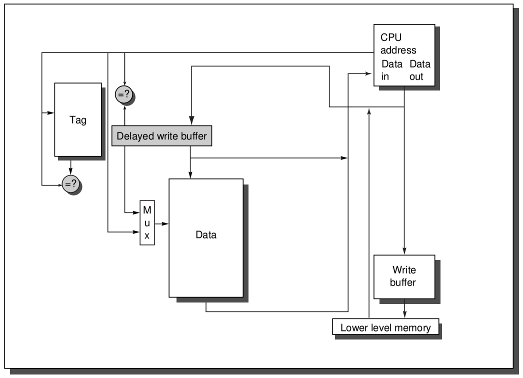
- If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag



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

3. Fast Hit Times Via Pipelined Writes

- Pipeline Tag Check and Update Cache as separate stages
- cycle 1 tag check, cycle 2 update cache, pipelined.
- allows 1 write hit/cycle despite 2-cycle latency



Cache Optimization Summary

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