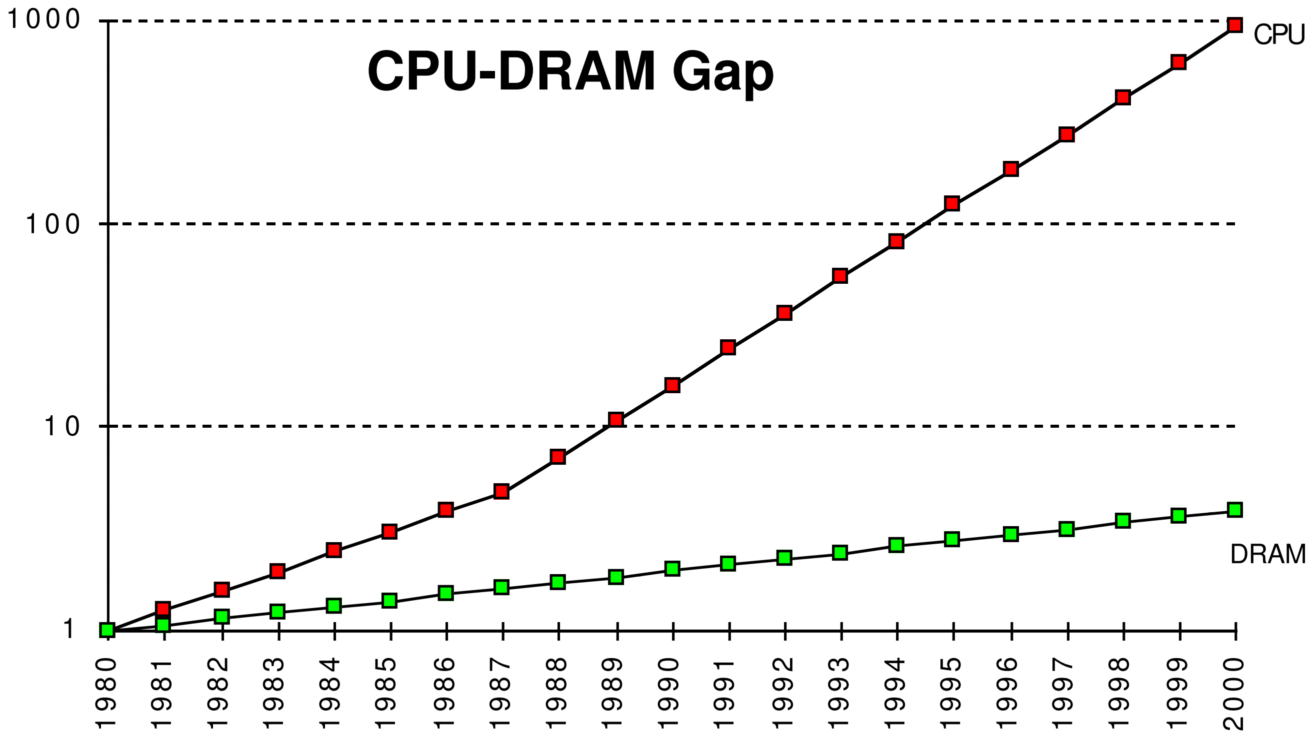
Review: Summary

- CPU-Memory gap is major performance obstacle for performance, HW and SW
- Take advantage of program behavior: locality
- Time of program still only reliable performance measure
- 4Qs of memory hierarchy

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, 60% trans. on Alpha 21164 µproc

Review: 4 Questions for Memory Hierarchy

- **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 Formulas

$$CPUtime = (CPUexecution\ \ clock\ \ cycles + Memory\ \ stall\ \ clock\ \ cycles) \times Clock\ \ cycle\ \ time$$

$$\begin{aligned} Memory\ \ stall\ \ clock\ \ cycles = {} & Reads \times Read\ \ miss\ \ rate \times Read\ \ miss\ \ penalty \\ & + Writes \times Write\ \ miss\ \ rate \times Write\ \ miss\ \ penalty \end{aligned}$$

$$Memory\ \ stall\ \ clock\ \ cycles = Memory\ \ accesses \times Miss\ \ rate \times Miss\ \ penalty$$

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

$$\frac{Misses}{Instruction} = \frac{Memory\ \ accesses}{Instruction} \times Miss\ \ rate$$

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

Review: Improving Cache Performance

1. Reduce the miss rate,

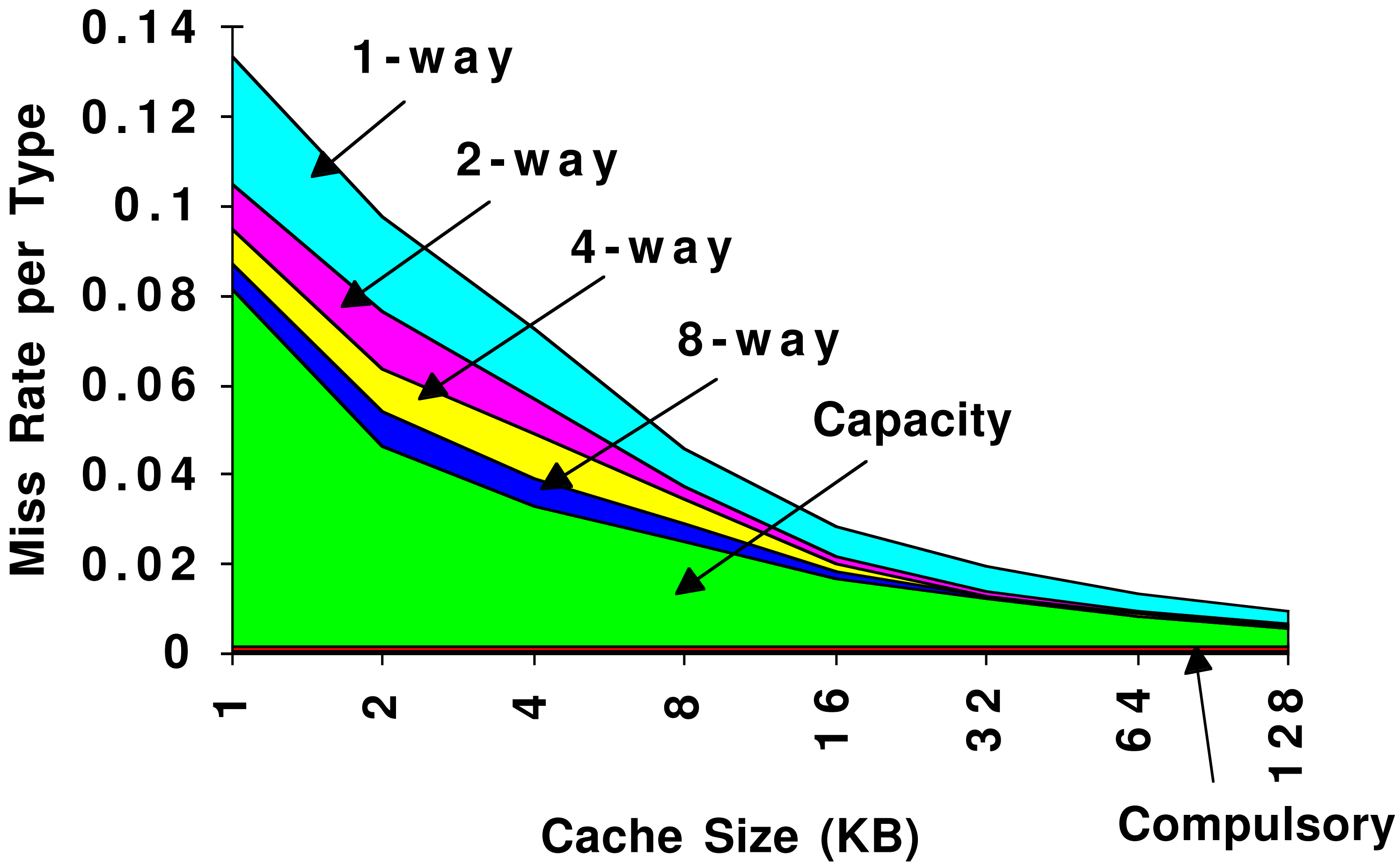
2. Reduce the miss penalty, or

3. Reduce the time to hit in the cache.

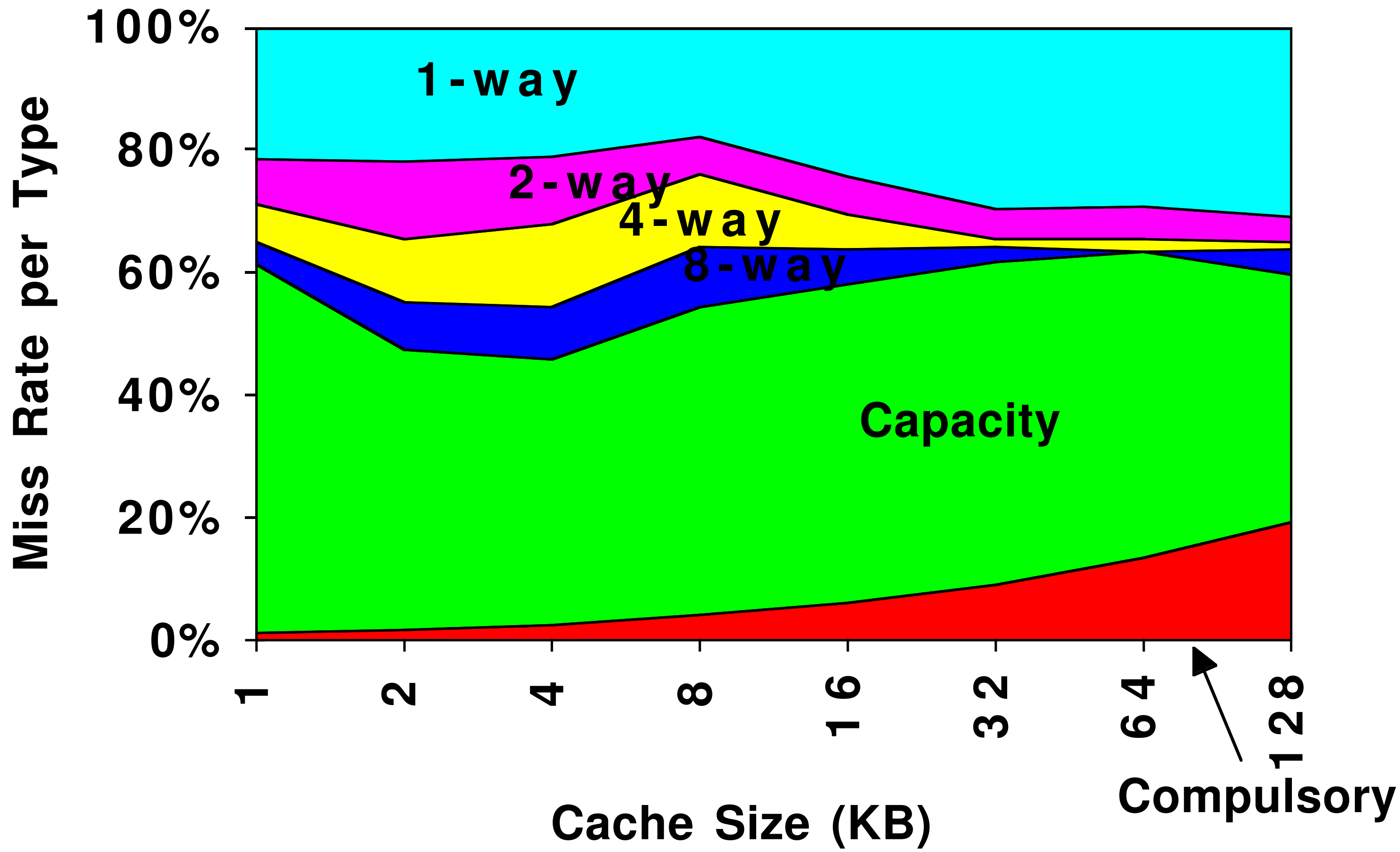
Reducing Misses

- **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. These are also called *cold start misses* or *first reference misses*.**
    (Misses in 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 Size X Cache)
  - ***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*.**
    (Misses in N-way Associative, Size X Cache)

3Cs Absolute Miss Rate *(2:1 Cache Rule)*



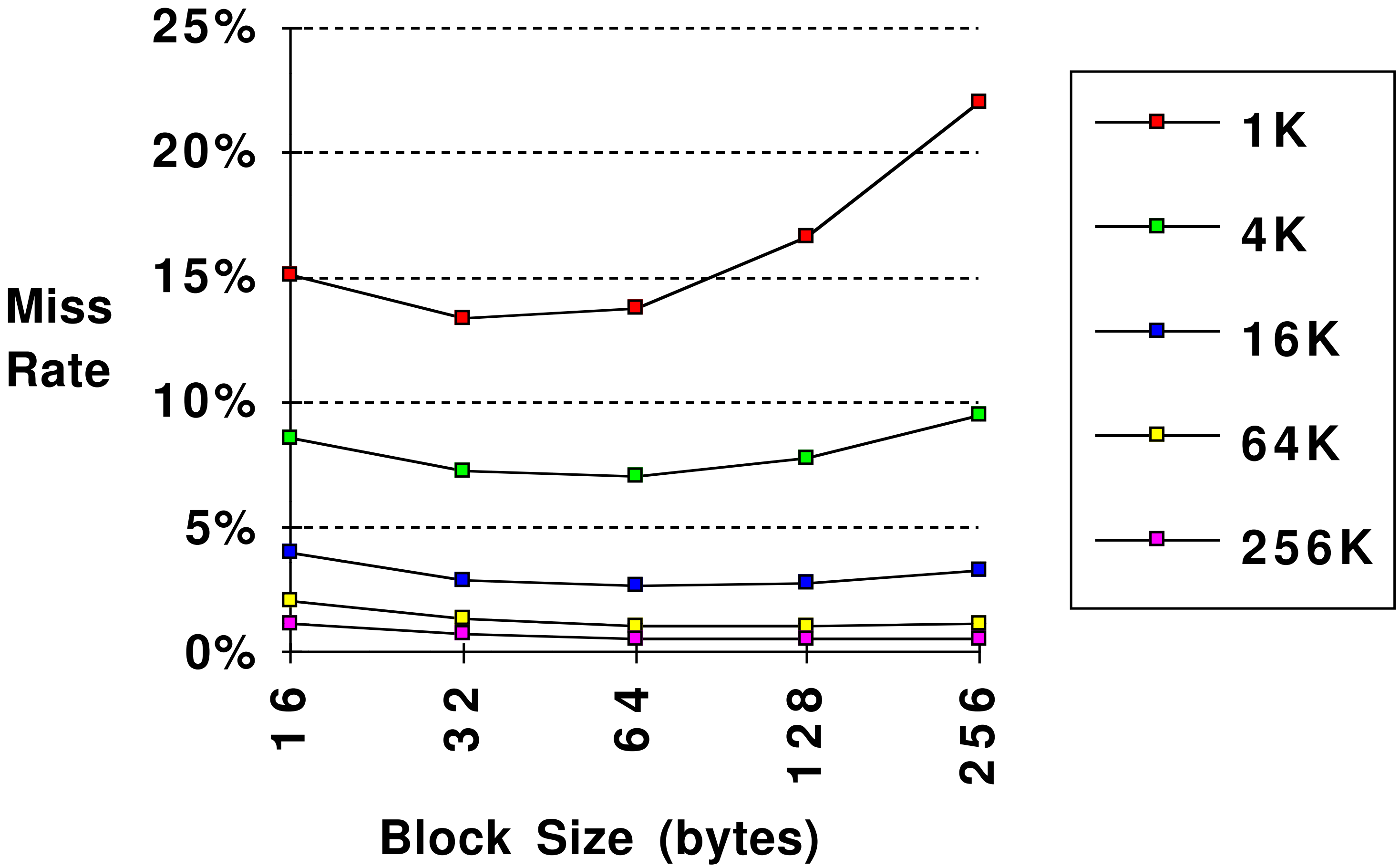
3Cs Relative Miss Rate



How Can Reduce Misses?

- **Change Associativity? Which of 3Cs affect?**
- **Change Block Size? Which of 3Cs affect?**
- **Change Compiler? Which of 3Cs affect?**

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 FA cache size N/2

- **Beware 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 Acc. Time vs. Miss Rate

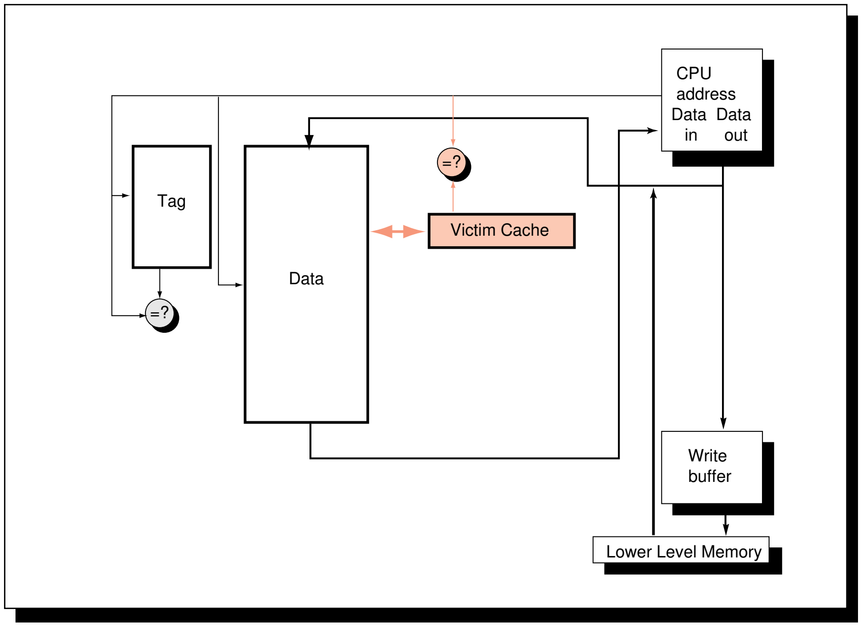
- **Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped**

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)

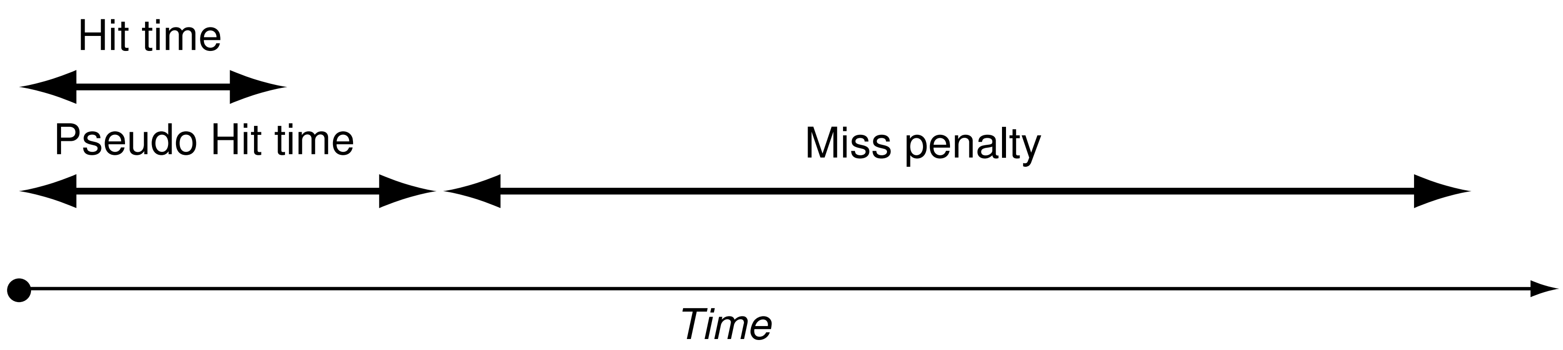
3. Reducing Misses via Victim Cache

- How combine fast hit time of Direct Mapped and 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



4. Reducing Misses via Pseudo-Associativity

- **How combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way Set Associative 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 is hit takes 1 or 2 cycles**
  - **Better for caches not tied directly to processor**

5. Reducing Misses by HW Prefetching Instr, Data

- e.g., Instruction Prefetch
  - Alpha 21064 fetches 2 blocks on a miss
  - extra block placed in stream buffer
  - on miss check stream buffer
- Could do with data blocks as well:
  - 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 associative caches
- Prefetching relies on extra memory bandwidth that can use 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)**
  - **Non-faulting (like speculation)**
- **Issuing Prefetch Instructions takes time**
  - **Is cost of prefetch issues < savings in reduced misses?**

7. Reducing Misses by Compiler Optimizations

- **Instructions**
  - Reorder procedures in memory so as to reduce misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced caches 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;
     };
```

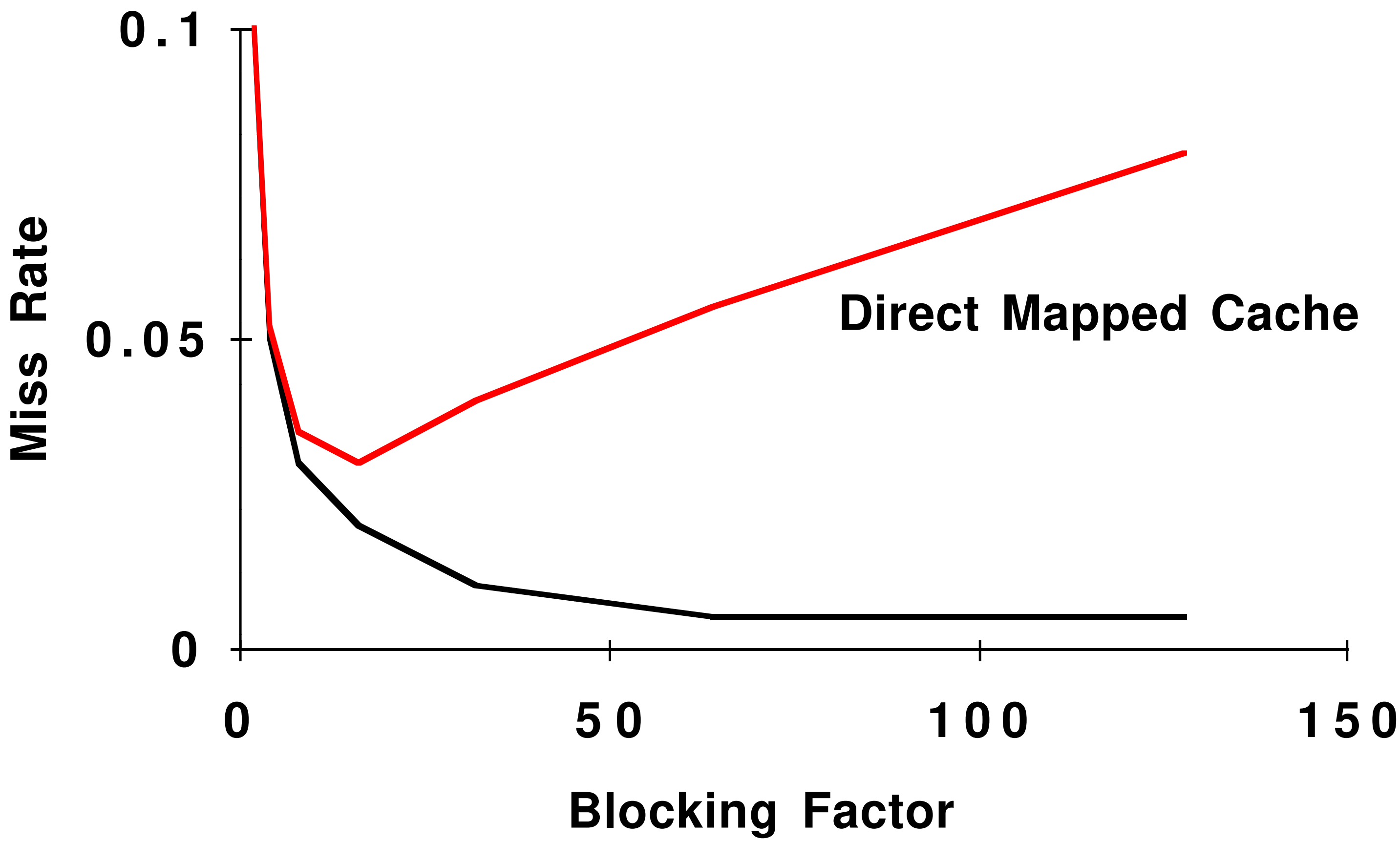
- **2 inner loops:**
  - read all NxN elements of z[]
  - read N elements of 1 row of y[] repeatedly
  - write N elements of 1 column of x[]
- **Capacity Misses a function of N & Cache Size:**
  - 3 NxN => no capacity misses; otherwise ...
- **Idea: compute on BxB submatrix that fits**

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;
      };
```

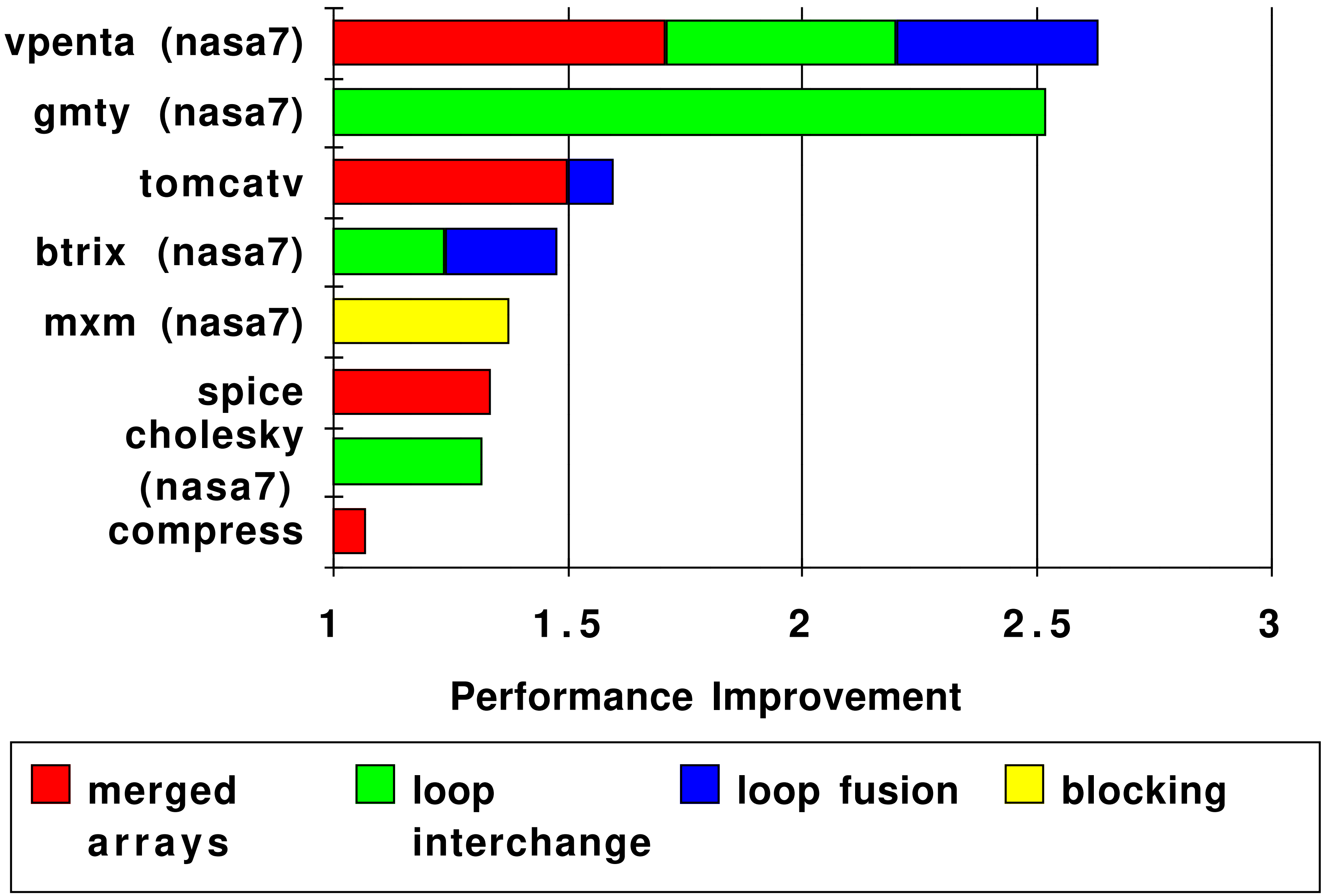
- Capacity Misses from $N^3 + N^2$ to $N^3/B + N^2$
- B called ***Blocking Factor***
- Conflict Misses Too?

Reducing Conflict Misses by Blocking



- Conflict misses in caches not fully associative 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



Summary

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

- **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 lecture: reducing Miss penalty**