Memory Hierarchy—Improving Performance

Professor Alvin R. Lebeck

Computer Science 220

Fall 2006

Admin

- **Homework #4 Due November 2nd**
- **Work on Projects**
- **Midterm**
  - **Max 98**
  - **Min 50**
  - **Mean 80**
- **Read NUCA paper**

Review: ABCs of caches

- Associativity
- Block size
- Capacity
- Number of sets S = C/(BA)
- 1-way (Direct-mapped)
  - A = 1, S = C/B
- N-way set-associative
- Fully associativity
  - S = 1, C = BA
- Know how a specific piece of data is found
  - Index, tag, block offset

Write Policies

- We know about write-through vs. write-back
- Assume: a 16-bit write to memory location 0x00 causes a cache miss.
- Do we change the cache tag and update data in the block?

  Yes: Write Allocate

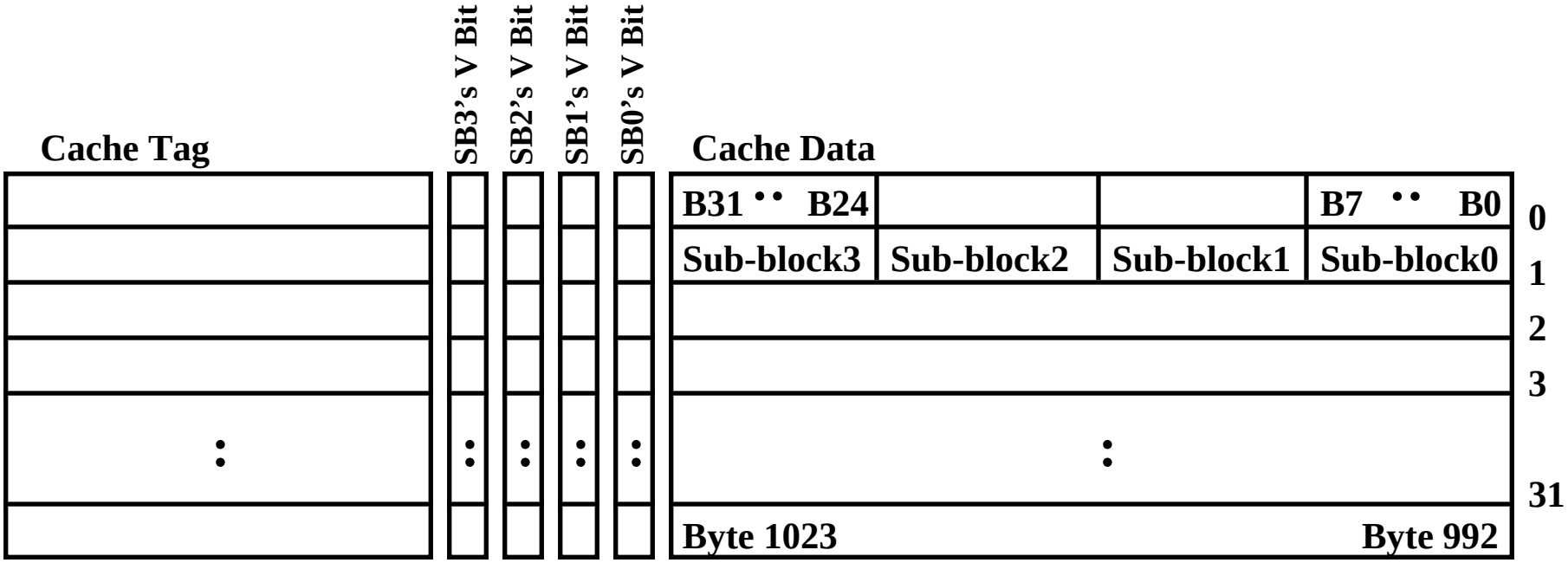
  No: Write No-Allocate
- Do we fetch the other data in the block?

  Yes: Fetch-on-Write (usually do write-allocate)

  No: No-Fetch-on-Write
- Write-around cache
  - Write-through no-write-allocate

Sub-block Cache (Sectored)

- **Sub-block:**
  - Share one cache tag between all sub-blocks in a block
  - Each sub-block within a block has its own valid bit
  - Example: 1 KB Direct Mapped Cache, 32-B Block, 8-B Sub-block
    - » Each cache entry will have: 32/8 = 4 valid bits
- **Miss: only the bytes in that sub-block are brought in.**
  - reduces cache fill bandwidth (penalty).



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)**

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

Cache Performance

CPUtime = IC x ($CPI_{execution}$ + (Mem accesses per instruction x Miss rate x Miss penalty)) x Clock cycle time

hits are included in $CPI_{execution}$

Misses per instruction = Memory accesses per instruction x Miss rate

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

Example

- **Miss penalty 50 clocks**
- **Miss rate 2%**
- **Base CPI 2.0**
- **1.33 references per instruction**
- **Compute the CPUtime**

- **CPUtime = IC x (2.0 + (1.33 x 0.02 x 50)) x Clock**
- **CPUtime = IC x 3.33 x Clock**
- **So CPI increased from 2.0 to 3.33 with a 2% miss rate**

Example 2

- **Two caches: both 64KB, 32 byte blocks, miss penalty 70ns, 1.3 references per instruction, CPI 2.0 w/ perfect cache**
- **direct mapped**
  - **Cycle time 2ns**
  - **Miss rate 1.4%**
- **2-way associative**
  - **Cycle time increases by 10%**
  - **Miss rate 1.0%**
- **Which is better?**
  - **Compute average memory access time**
  - **Compute CPU time**

Example 2 Continued

- **Ave Mem Acc Time =**
  Hit time + (miss rate x miss penalty)
  - **1-way: 2.0 + (0.014 x 70) = 2.98ns**
  - **2-way: 2.2 + (0.010 x 70) = 2.90ns**

- **CPUtime = IC x CPIexec x Cycle**
  - **CPIexec = CPIbase + ((memacc/inst) x Miss rate x miss penalty)**
  - **Note: miss penalty x cycle time = 70ns**

  - **1-way: IC x ((2.0 x 2.0) + (1.3x0.014x70)) = 5.27 x IC**
  - **2-way: IC x ((2.0 x 2.2) + (1.3x0.010x70)) = 5.31 x IC**

Review: Cache Performance

CPUtime = IC x ($CPI_{execution}$ + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time

hits are included in $CPI_{execution}$

Misses per instruction = Memory accesses per instruction x Miss rate

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

Improving Cache Performance

Ave Mem Acc Time =
Hit time + (miss rate x miss penalty)

1. Reduce the miss rate,

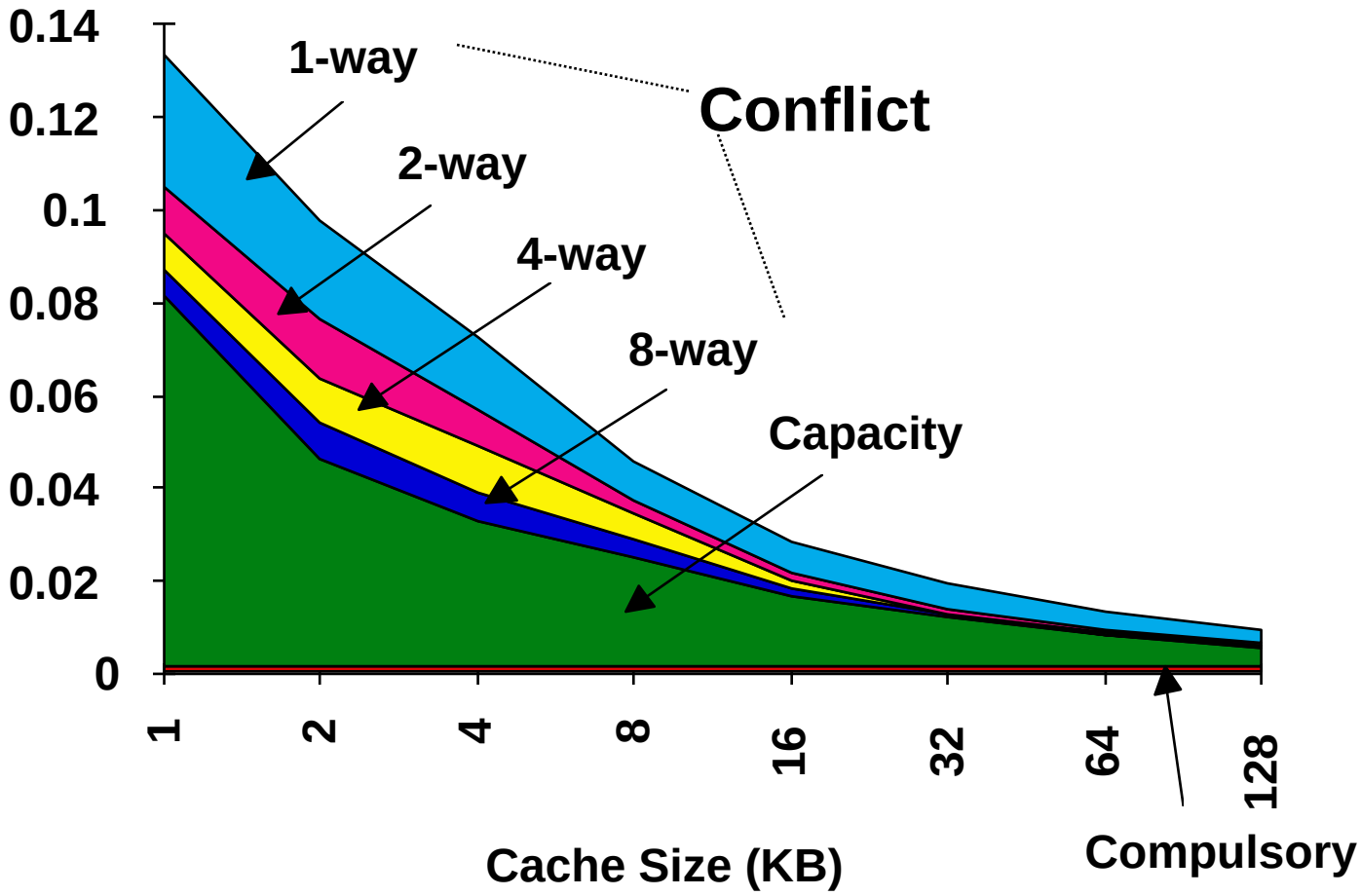
2. Reduce the miss penalty, or

3. Reduce the time to hit in the cache.

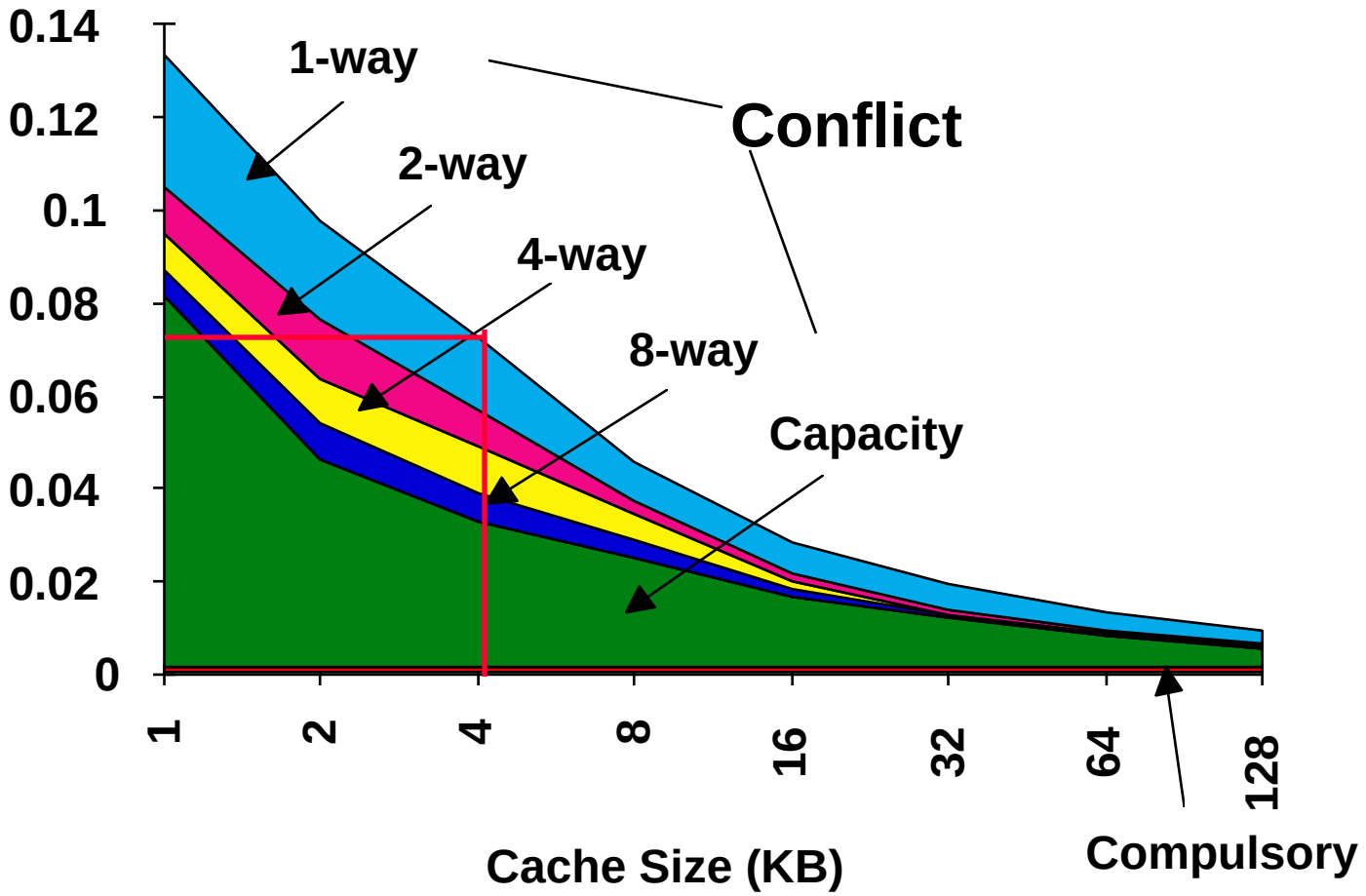
Reducing Misses

- **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)

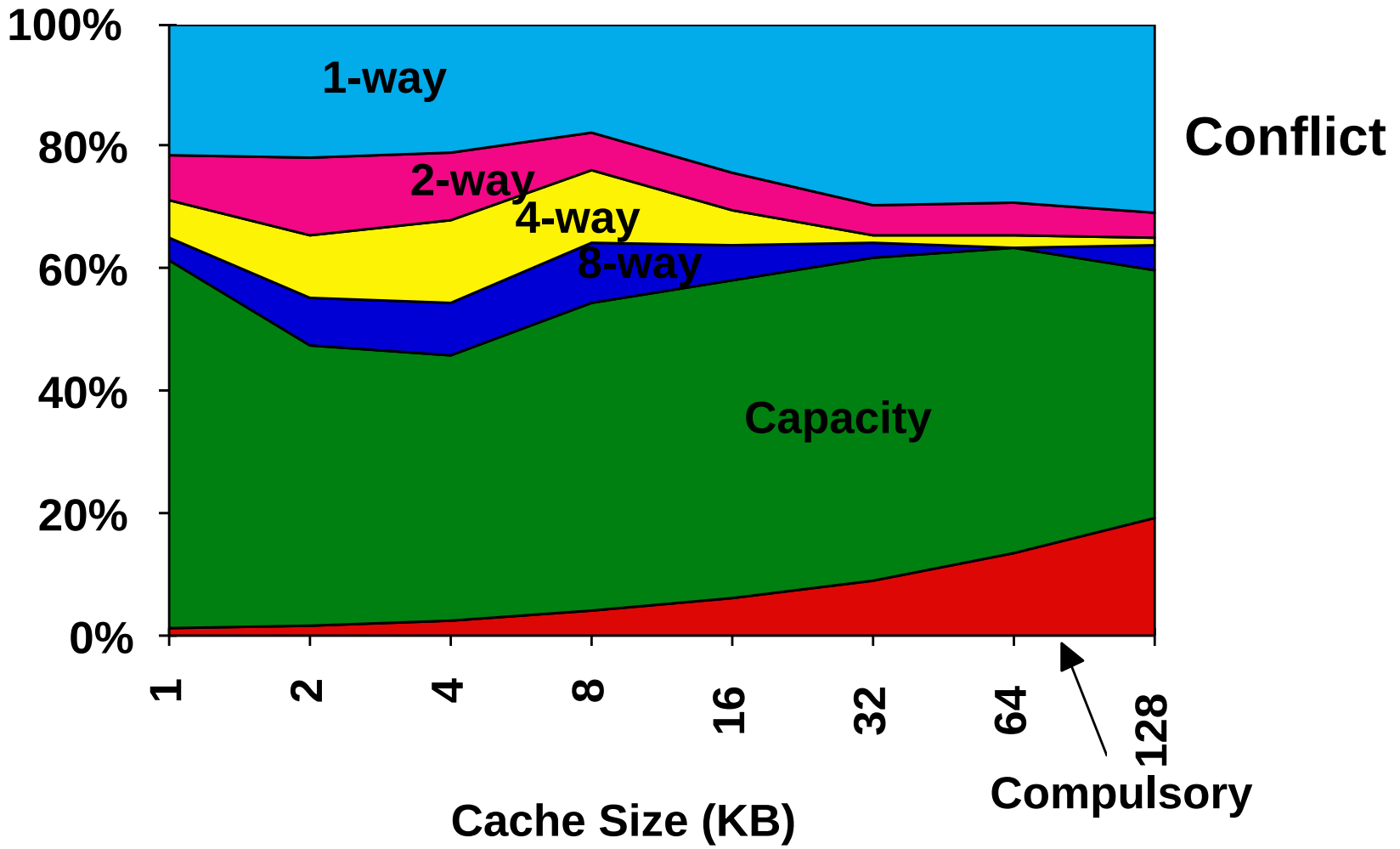
3Cs Absolute Miss Rate



2:1 Cache Rule



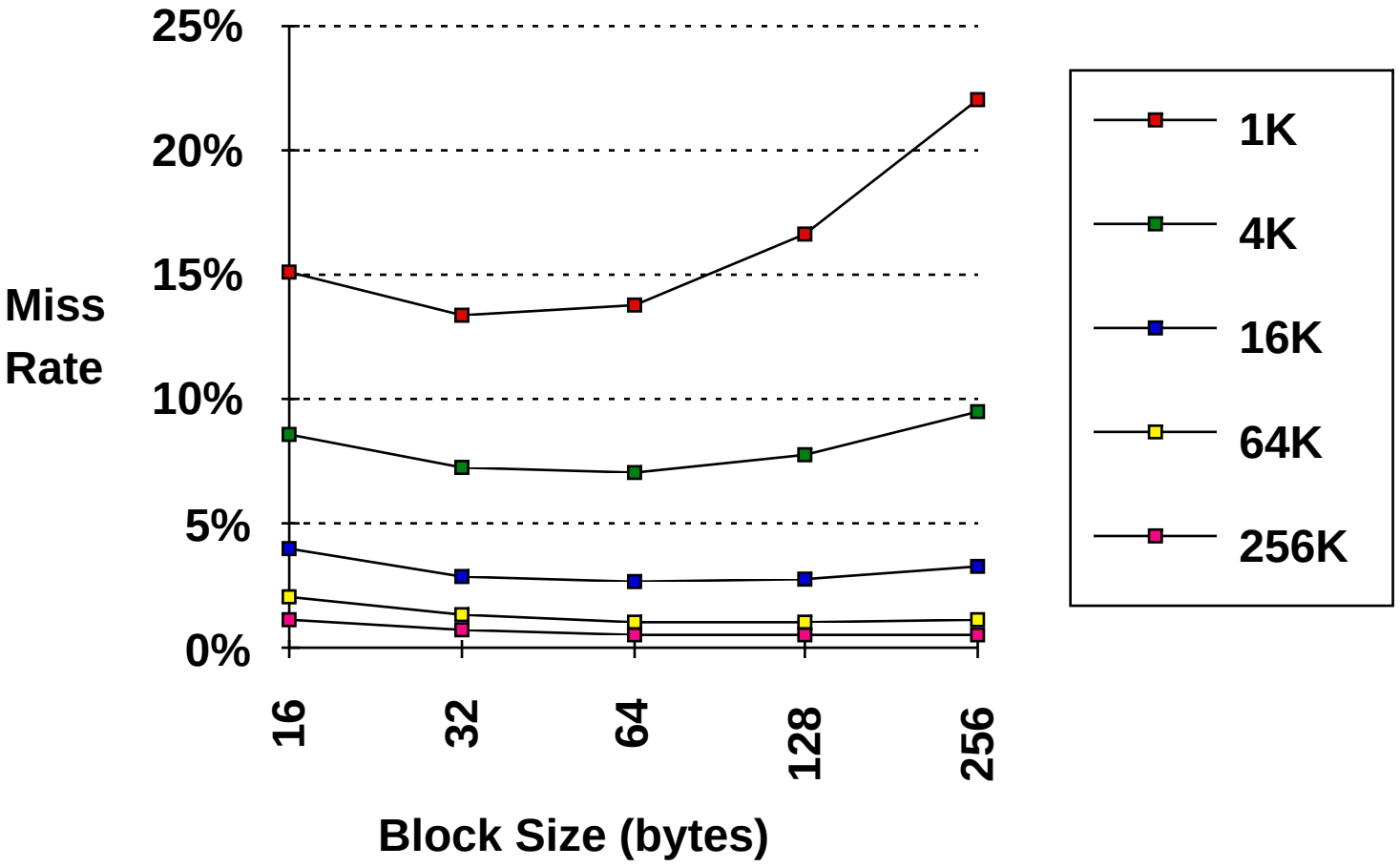
3Cs Relative Miss Rate



How Can We Reduce Misses?

- **Change Block Size? Which of 3Cs affected?**
- **Change Associativity? Which of 3Cs affected?**
- **Change Program/Compiler? Which of 3Cs 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 external cache +10%, internal + 2% for 2-way vs. 1-way**

Example: Avg. Memory Access Time vs. Miss Rate

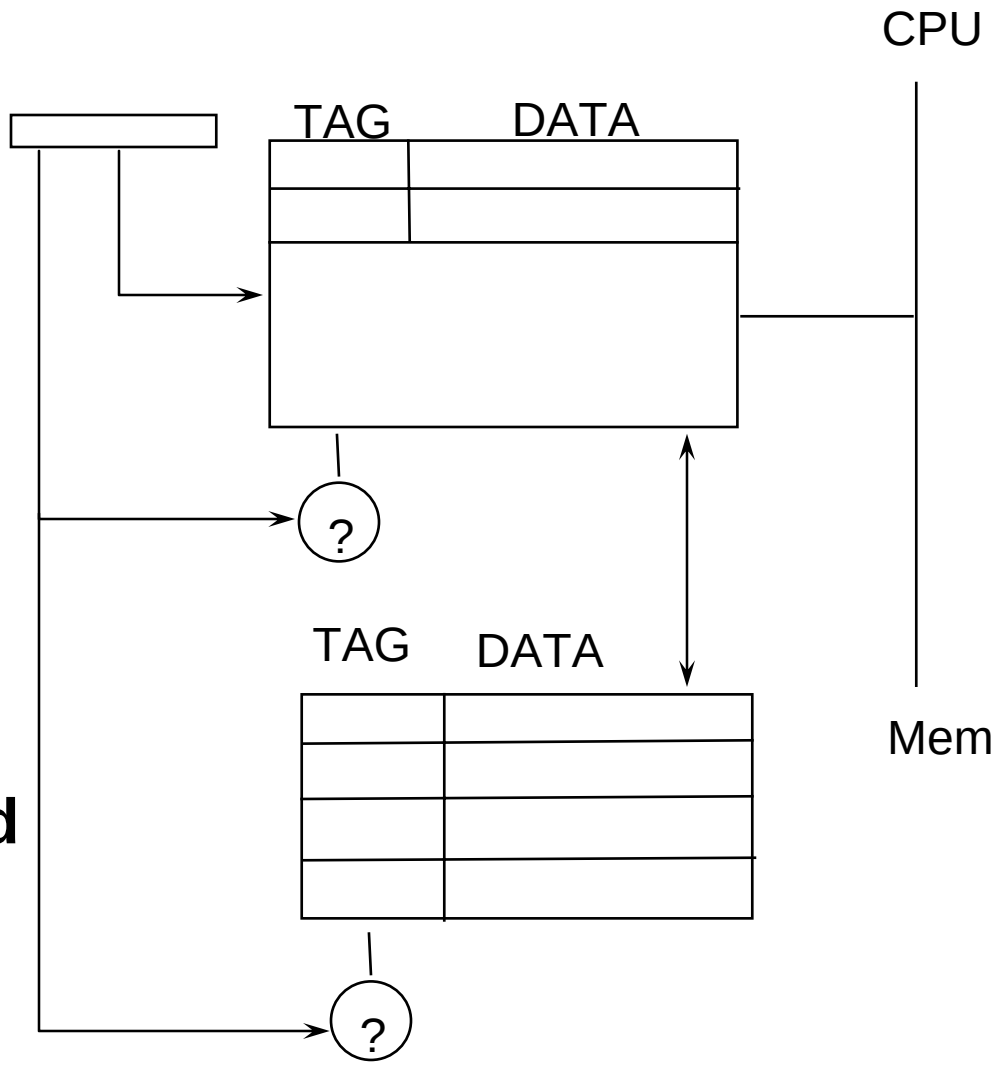
- **Example: assume Clock Cycle = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. Clock cycle for direct mapped**

Cache Size	Associativity			
(KB)	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 Conflict Misses via 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**



4. Reducing Conflict 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**

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**
  - **Pointers vs. arrays**
  - **Kedem: Markov predictor (address correlation)**
- **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) binding**
  - **Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9) non-binding**
  - **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?**

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)

7. Reducing Misses by Program/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
- What is miss rate before and after?

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;
     };
```

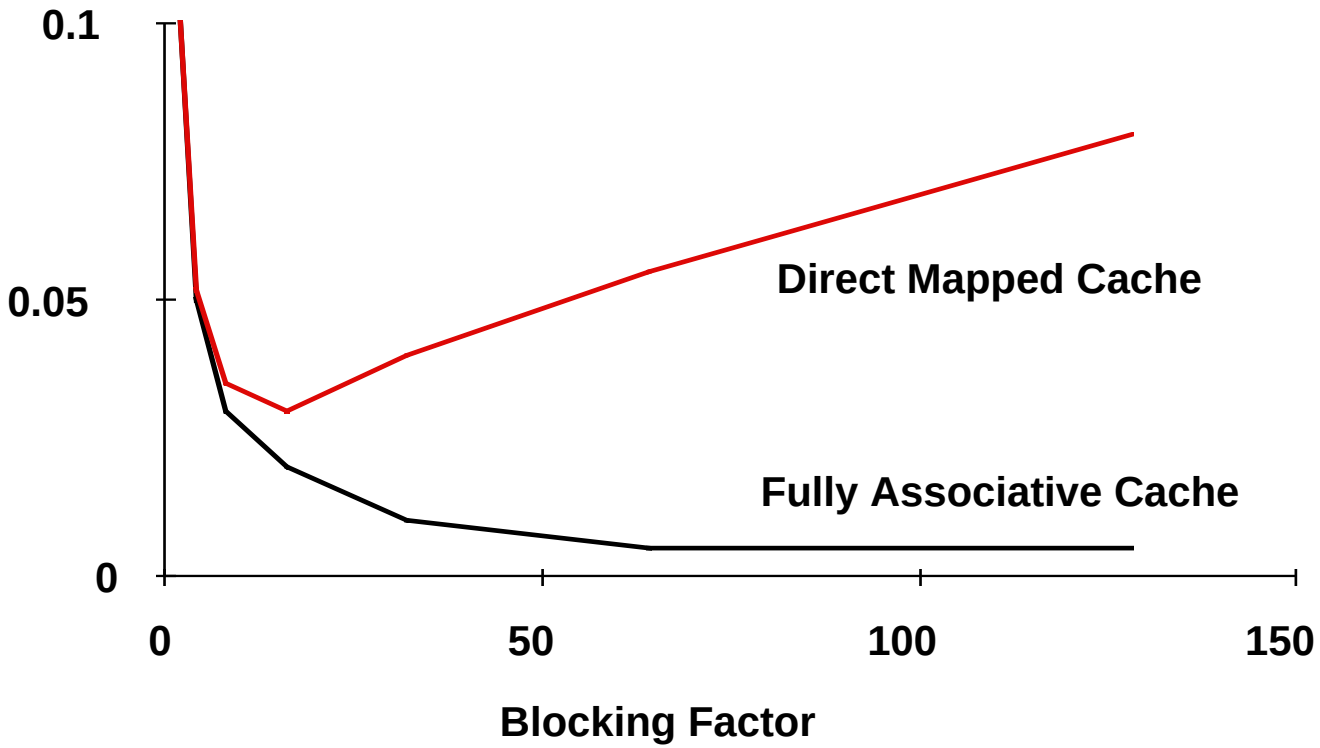
- **Two Inner Loops:**
  - Read all NxN elements of z[ ]
  - Read N elements of 1 row of y[ ] repeatedly
  - Write N elements of 1 row of x[ ]
- **Capacity Misses a function of N & Cache Size:**
  - 3 NxN => no capacity misses; otherwise ...
- **Idea: compute on BxB submatrix that fits**

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;
     };
```

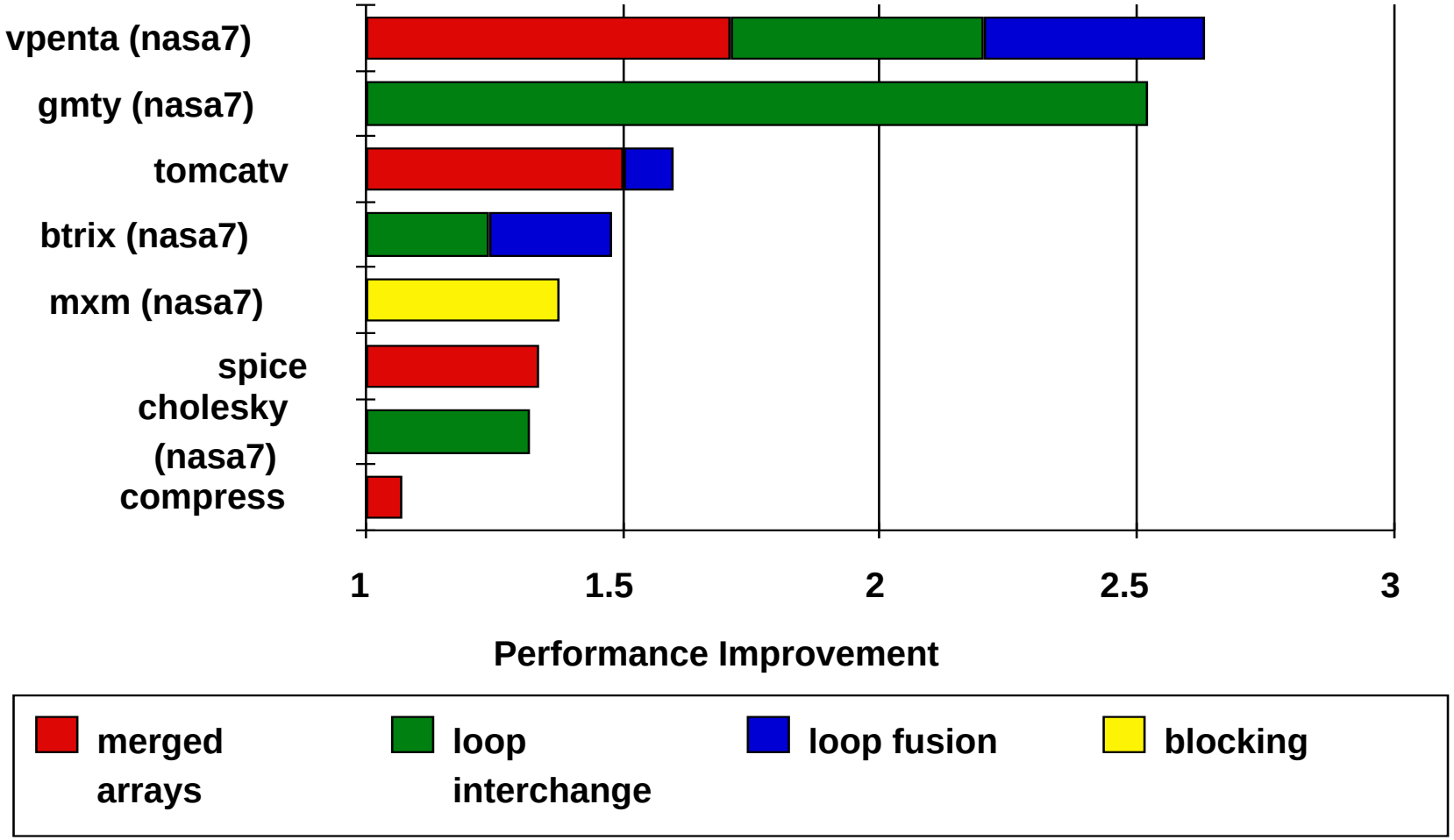
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- B called *Blocking Factor*
- 6 loop variant exists
- Conflict Misses?

Reducing Conflict Misses by Blocking

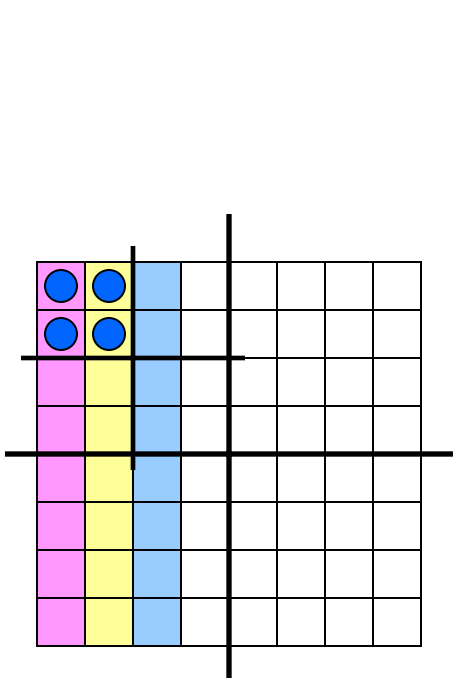


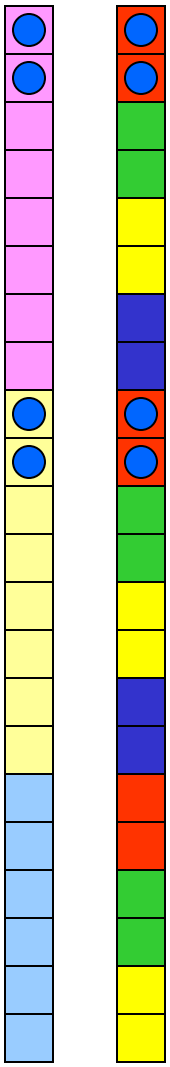
- **Conflict misses in caches not FA vs. Blocking size**
  - **Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache**

Summary of Program/Compiler Optimizations to Reduce Cache Misses



Layout and Cache Behavior



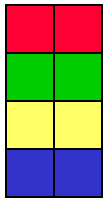


Memory

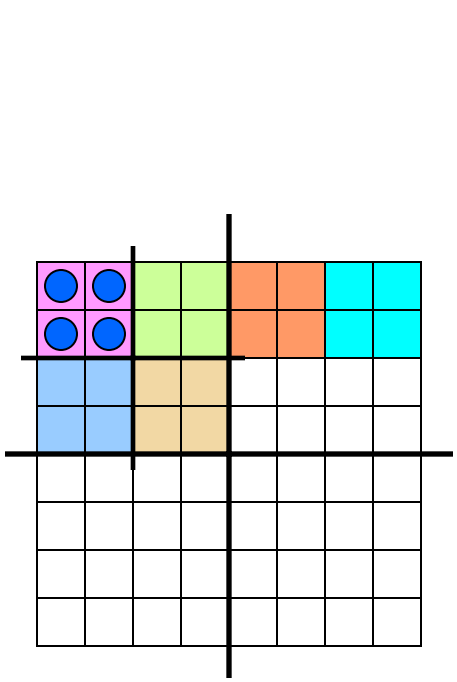
Cache Mapping

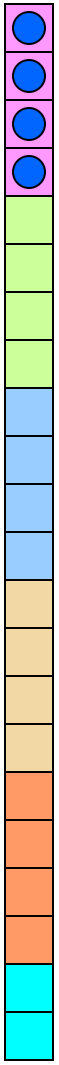
- Tile elements spread out in memory because of column-major mapping
- Fixed mapping into cache
- Self-interference in cache

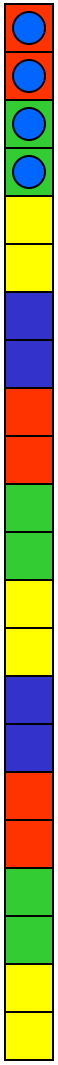
Cache



Making Tiles Contiguous







- **Elements of a quadrant are contiguous**
- **Recursive layout**
- **Elements of a tile are contiguous**
- **No self-interference in cache**

Memory

Cache Mapping

Non-linear Layout Functions

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

4-D blocked

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Morton order

0	3	4	5
1	2	7	6
14	13	8	9
15	12	11	10

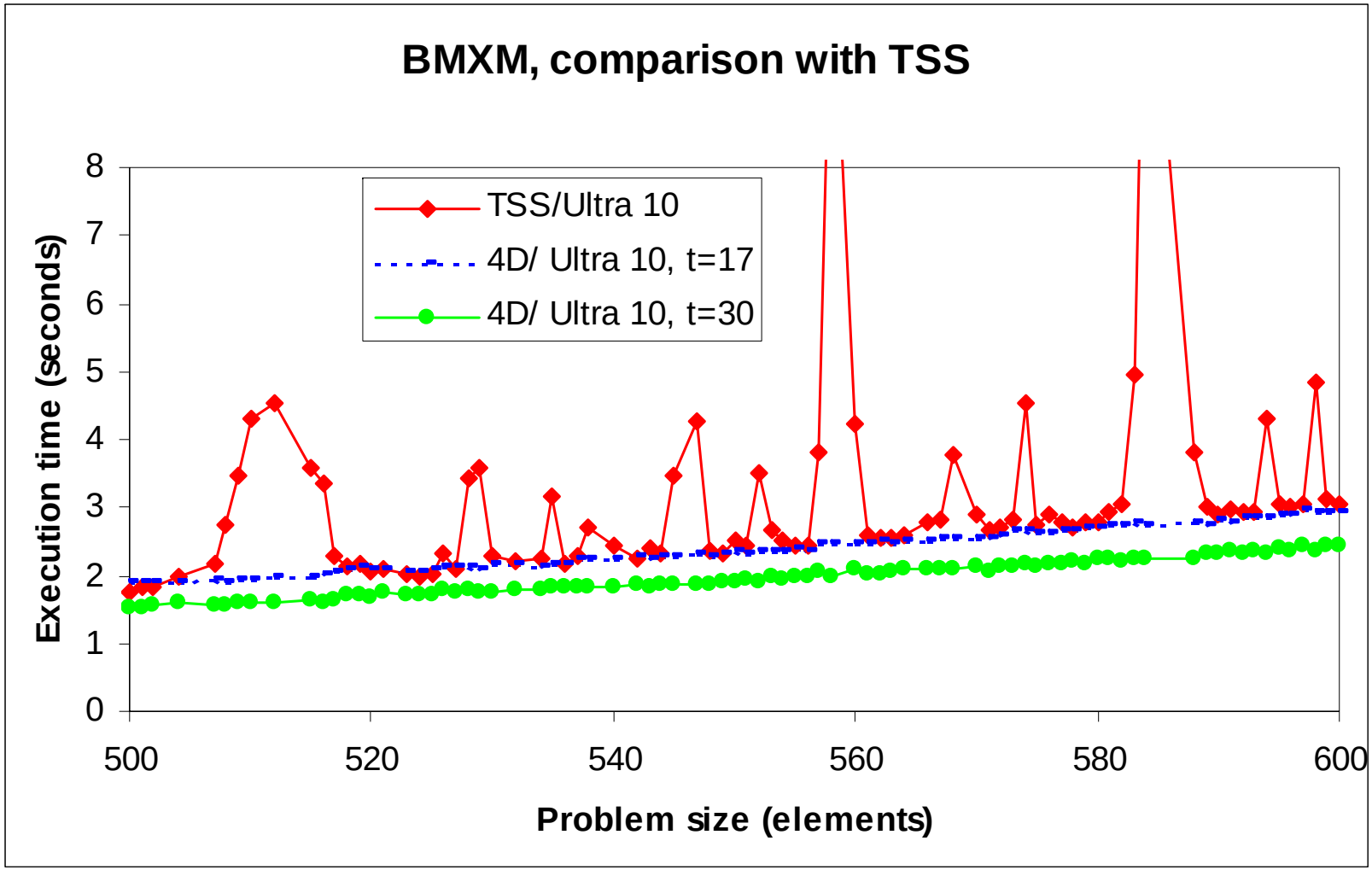
Hilbert order

- Different locality properties
- Different inclusion properties
- Different addressing costs

Performance Improvement

CPU	UltraSPARC 2i		UltraSPARC 2		Alpha 21164	
Clock rate	300MHz		300 MHz		500MHz	
L1 cache	16KB/32B/1		16KB/32B/1		8KB/32B/1	
L2 cache	512KB/64B/1		2MB/64B/1		96KB/64B/3	
L3 cache					2MB/64B/1	
RAM	320MB		512MB		512MB	
TLB entries	64		64		64	
Page size	8KB		8KB		8KB	
	Ultra 10		**Ultra 60**		**Miata**	
	4D	**MO**	**4D**	**MO**	**4D**	**MO**
BLKMXM	0.93	1.06	0.95	1.05	0.97	0.95
RECMXM		0.94		0.94		0.95
STRASSEN		0.87		0.79		0.91
CHOL	0.78		0.85		0.67	
STDHAAR	0.68	0.67	0.64	0.64	0.42	0.43
NONHAAR	0.62	0.61	0.58	0.58	0.40	0.40

Comparison with TSS



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**
  - **How to eliminate them**
- **Program Transformations**
  - **Change Algorithm**
  - **Change Data Layout**
- **Implication: Think about caches if you want high performance!**