Computer Science 146
Computer Architecture

Fall 2019

Harvard University

Instructor: Prof. David Brooks
dbrooks@eecs.harvard.edu

Lecture 16: Even more on Caches

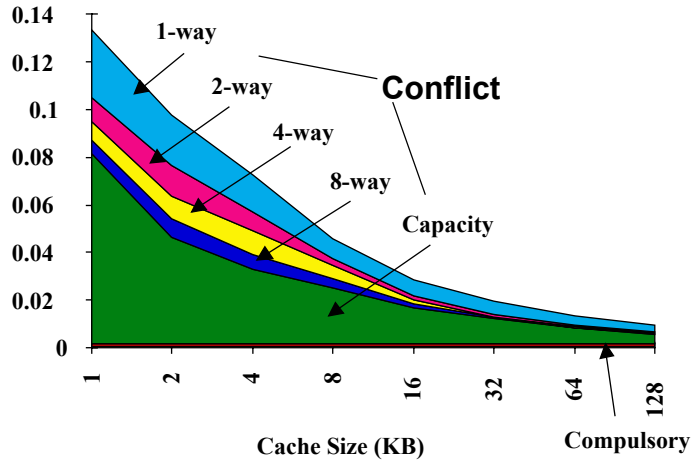
Lecture Outline

- How to improve cache performance?
  - Reducing Cache Miss Penalty
  - Reducing Miss Rate
  - Reducing Hit Time
  - Reducing Miss Penalty/Rate via parallelism

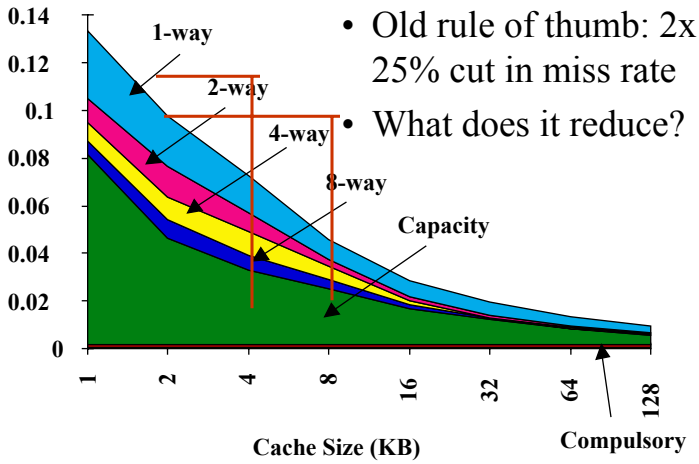
Where to misses come from?

- Classifying Misses: 3 Cs
  - *Compulsory*—First access to a block. 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 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*. *(Remaining Misses)*
- 4th "C": *Coherence* - Misses caused by cache coherence.

3Cs Absolute Miss Rate (SPEC92)

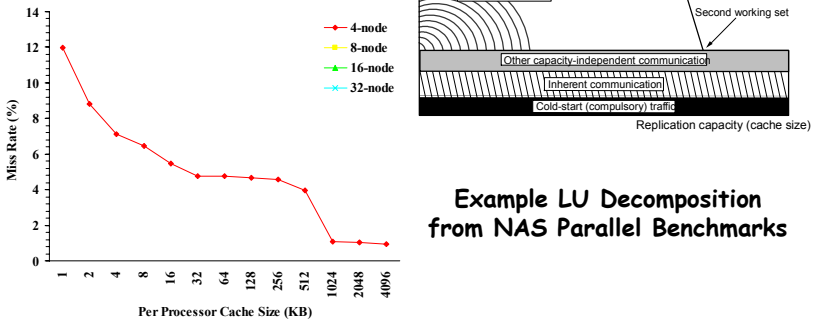


Cache Size



- Old rule of thumb: 2x size => 25% cut in miss rate
- What does it reduce?

Huge Caches => Working Sets



Example LU Decomposition from NAS Parallel Benchmarks

Cache Organization?

- Assume total cache size not changed:
- What happens if:

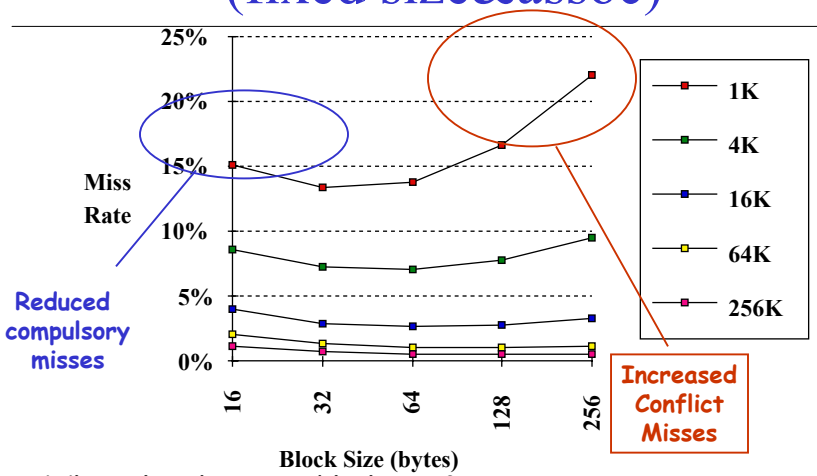
1) Change Block Size:

2) Change Associativity:

3) Change Compiler:

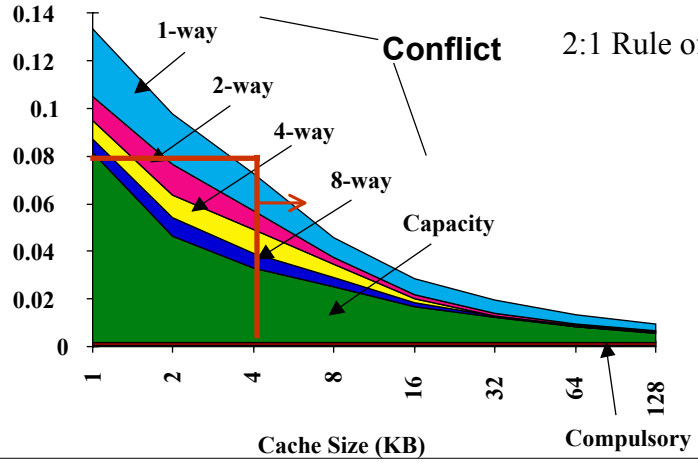
Which of 3Cs is obviously affected?

Larger Block Size (fixed size&assoc)



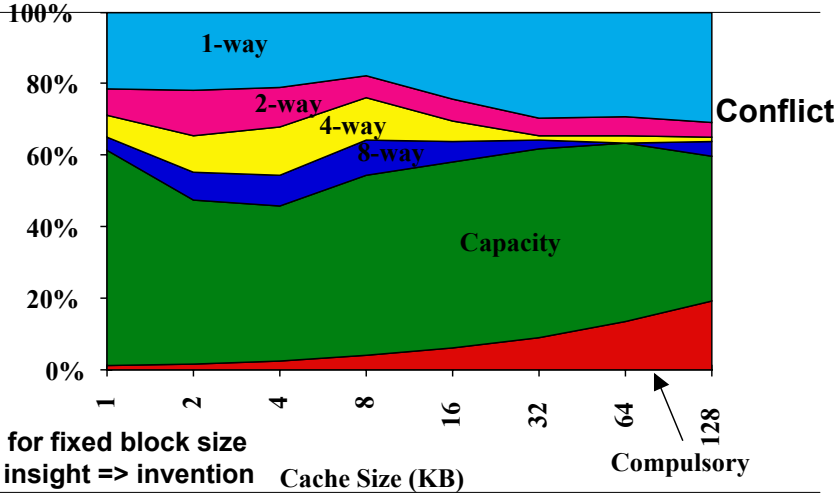
What else drives up block size?

Associativity



2:1 Rule of Thumb

3Cs Relative Miss Rate



Flaws: for fixed block size
Good: insight => invention

Associativity vs Cycle Time

- Beware: Execution time is only final measure!
- Why is cycle time tied to hit time?

- Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%
  - suggested big and dumb caches

Example: Avg. Memory Access Time vs. Miss Rate

- Example: assume Cache Cycle Time = 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)

Way Prediction

- Maintain hit-speed of a direct-mapped cache, get conflict miss reduction of associative cache
- Extra bits are used to predict the way of the next cache access
- Alpha 21264 uses 2-way set-associative I-Cache
  - If predictor is correct, 1-cycle I-cache latency
  - If incorrect, 3-cycle latency
  - SPEC95 => Prediction accuracy ~85%

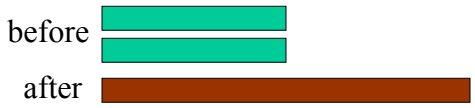
Reducing Misses by Compiler Optimizations

- 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)
- 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 stuctures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```



Reducing conflicts between val & key;
improve spatial locality

Padding and Offset Changes

- Cache size of 8KB

Before:

```
int a[2048];
int b[2048];
int c[2048];

for(i=0;i<2048;i++)
  C[i]=A[i]+B[i]
```

After:

```
int a[2050];
int b[2050];
int c[2050];

for(i=0;i<2048;i++)
  C[i]=A[i]+B[i]
```

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];
```

C: Row-Major Order

X[0][0]
X[0][1]
X[1][0]
X[1][1]

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

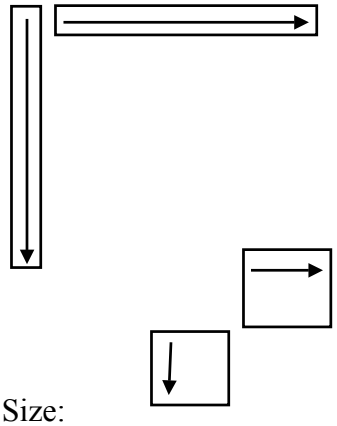
Loop Fusion Example

```
/* 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; improves spatial locality

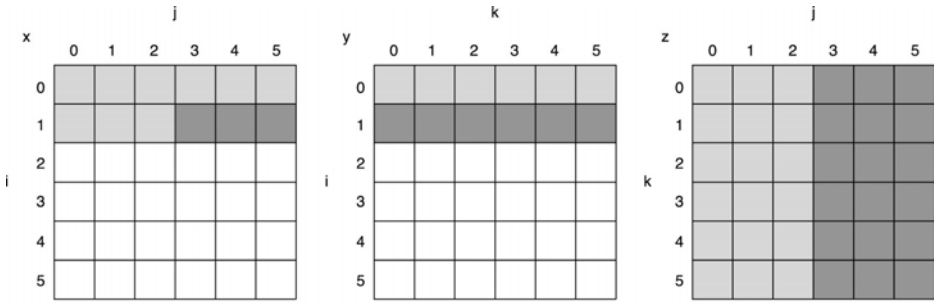
Blocking Example

```
/* 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[]
- Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits in cache

Blocking: Before

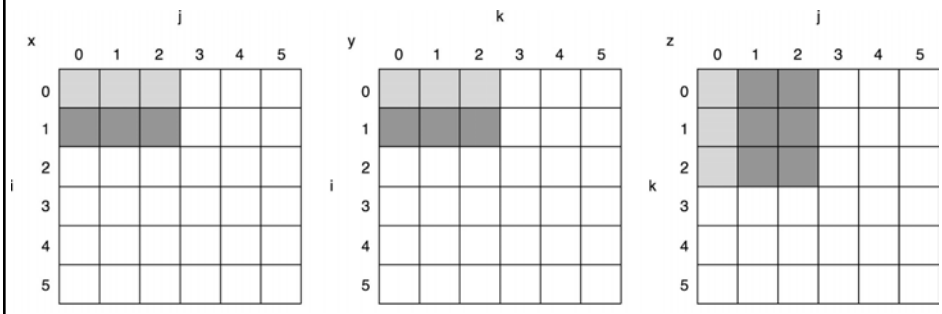


Blocking Example

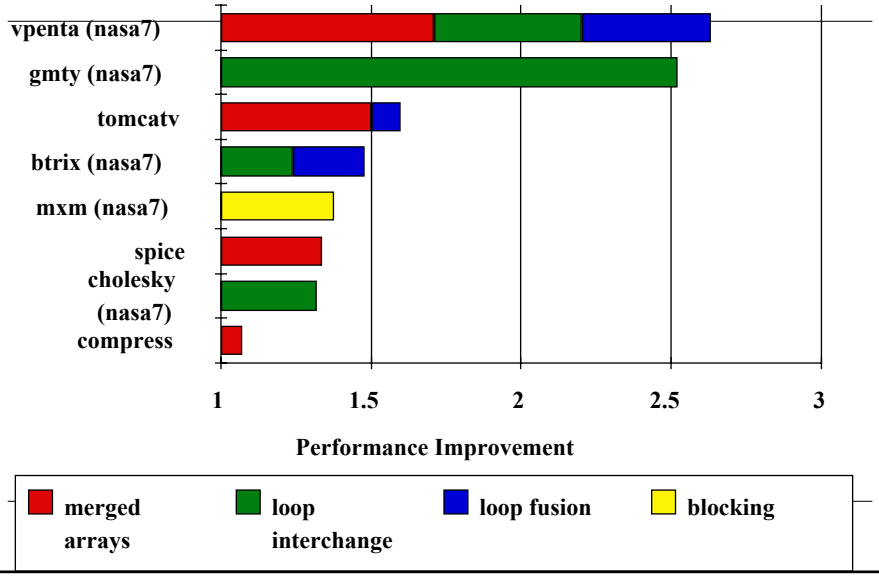
```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
     {r = 0;
      for (k = kk; k < min(kk+B-1,N); k = k+1) {
        r = r + y[i][k]*z[k][j];};
      x[i][j] = x[i][j] + r;   };
```

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

Blocking: After



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



Summary: Miss Rate Reduction

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

- 3 Cs: Compulsory, Capacity, Conflict
  0. Larger cache
  1. Reduce Misses via Larger Block Size
  2. Reduce Misses via Higher Associativity
  3. Reducing Misses via Victim Cache
  4. Reducing Misses via Way-Prediction
  5. Reducing Misses by Compiler Optimization

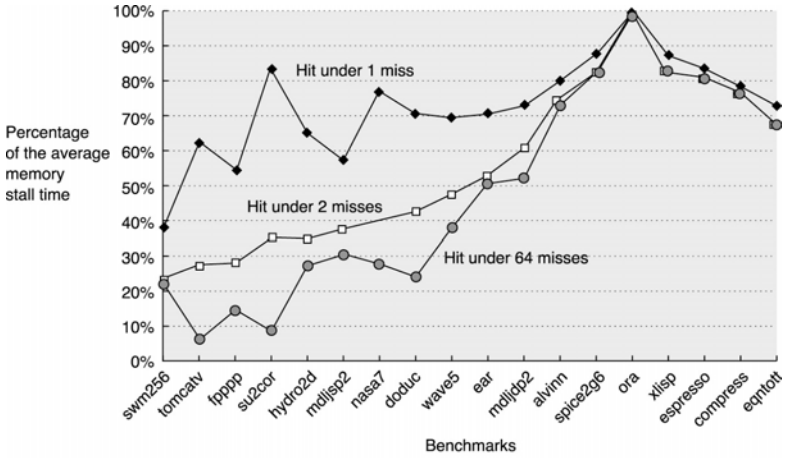
Review: Improving Cache Performance

- How to improve cache performance?
  - Reducing Cache Miss Penalty
  - Reducing Miss Rate
  - Reducing Miss Penalty/Rate via parallelism
  - Reducing Hit Time

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
  - requires multi-bank memories
- "*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)
  - Penium 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

Reducing Misses by Hardware Prefetching of Instructions & Data

- Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer" not the cache
  - On Access: check both cache and stream buffer
  - On SB Hit: move line into cache
  - On SB Miss: Clear and refill SB with successive lines
- 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 having extra memory bandwidth that can be used without penalty

Hardware Prefetching

- What to prefetch?
  - One block ahead (spatially)
    - What will this work well for?
  - Address prediction for non-sequential data
    - Correlated predictors (store miss, next_miss pairs in table)
    - Jump-pointers (augment data structures with prefetch pointers)
- When to prefetch?
  - On every reference
  - On a miss (basically doubles block size!)
  - When resident data becomes “dead” -- how do we know?
    - No one will use it anymore, so it can be kicked out

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
- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    - Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    - Can be incorrect. Faults?
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

Reducing Hit Times

- Some common techniques/trends
  - Small and simple caches
    - Pentium III – 16KB L1
    - Pentium 4 – 8KB L1
  - Pipelined Caches (actually bandwidth increase)
    - Pentium – 1 clock cycle I-Cache
    - Pentium III – 2 clock cycle I-Cache
    - Pentium 4 – 4 clock cycle I-Cache
  - Trace Caches
    - Beyond spatial locality
    - Dynamic sequences of instruction (including taken branches)

Hit Time Reduction

- Translation of Virtual (programmer) to Physical (memory) Addresses is a bottleneck on hit time
- Defer this topic to next time

Cache Optimization Summary

	Technique	*MR*	*MP*	*HT*	*Complexity*
miss rate	Larger Block Size	+	–		0
	Higher Associativity/Larger Cache	+		–	1
	Victim Caches	+			2
	Way-Predicting 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
Hit time	Pipelined Cache			+	1
	Trace Cache			+	3

Cache Bandwidth

- Superscalars need multiple memory access per cycle
- Parallel cache access: more difficult than parallel ALUs
  - Caches have state so multiple accesses will affect each other
- "True Multiporting"
  - Multiple decoders, read/write wordlines per SRAM cell
  - Pipeline a single port by "double pumping" Alpha 21264
  - Multiple cache copies (like clustered register file) POWER4
- Interleaved Multiporting
  - Cache divides into banks – two accesses to same bank => conflict

Next Time

- Virtual Memory
- Main Memory