EITF20: Computer Architecture

Part4.1.1: Cache - 2

Liang Liu
liang.liu@eit.lth.se

Outline

- **Reiteration**
- **Cache performance optimization**
  - Bandwidth increase
  - Reduce hit time
  - Reduce miss penalty
  - Reduce miss rate
- **Summary**

Memory hierachy

More Costly **Smaller**

Access Times

Access Times	Level
1ns → 2ns	Registers
3ns → 10ns	Level 1 Cache
25ns → 50ns	Level 2 Cache
30ns → 90ns	Main Memory
5ms → 20ms	Fixed Rigid Disk
100ms → 5s *	Optical Disk (Jukeboxes)
10s → 3m *	Magnetic Tape (Robotic Libraries)

System

Online

Near Line

Offline

Less Costly **Larger**

* If volume is mounted.

AIM: Fast as cache; Large as disk; Cheap as possible

Why?

- 1980: no cache in microprocessors
- 1995: 2-level caches in a processor package
- 2000: 2-level caches on a processor die
- 2003: 3-level caches on a processor die

Performance

100,000
10,000
1,000
100
10
1

Processor

Memory

1980 1985 1990 1995 2000 2005 2010

Why does caching work?

- A program access a relatively small portion of the address space at any instant of time
- Two different types of locality:
  - **Temporal locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - **Spatial locality** (Locality in space): If an item is referenced, items whose addresses are close, tend to be referenced soon

Cache measures

- **hit rate = no of accesses that hit/no of accesses**
  - close to 1, more convenient with
- **miss rate = 1.0 − hit rate**
- **hit time: cache access time plus time to determine hit/miss**
- **miss penalty: time to replace a block**
  - measured in ns or number of clock cycles and depends on:
  - latency: time to get first word
  - bandwidth: time to transfer block
- **out-of-order execution can hide some of the miss penalty**
- **Average memory access time = hit time + miss rate * miss penalty**

Four memory hierarchy questions

- Q1: Where can a block be placed in the upper level?
(Block placement)
- Q2: How is a block found if it is in the upper level?
(Block identification)
- Q3: Which block should be replaced on a miss?
(Block replacement)
- Q4: What happens on a write?
(Write strategy)

Outline

- Reiteration
- Cache performance optimization
- Bandwidth increase
- Reduce hit time
- Reduce miss penalty
- Reduce miss rate
- Summary

Cache performance

$$\text{Execution Time} =$$

$$IC * (CPI_{execution} + \frac{\text{mem accesses}}{\text{instruction}} * \textbf{miss rate} * \textbf{miss penalty}) * \mathbf{T_C}$$

Three ways to increase performance:

- Reduce miss rate
- Reduce miss penalty
- Reduce hit time

However, remember:

Execution time is the only true measure!

Cache performance, example

$$\text{CPU execution Time} = \\ IC * (CPI_{execution} + \frac{mem\ accesses}{instruction} * miss\ rate * miss\ penalty) * T_C$$

Example:

miss rate (%)	1
miss penalty (cycles)	50
$\frac{mem\ accesses}{instruction}$	k
CPI increase	$k * 0.01 * 50$

Sources of Cache miss

- **A cache miss can be classified as a:**
  - **Compulsory miss**: The first reference is always a miss
  - **Capacity miss**: If the cache memory is to small it will fill up and subsequent references will miss
  - **Conflict miss**: Two memory blocks may be mapped to the same cache block with a direct or set-associative address mapping

3 C's

Miss rate components – 3 C's

Miss Rate per Type

0.14
0.12
0.1
0.08
0.06
0.04
0.02
0

1-way
2-way
4-way
8-way
Conflict
Capacity
Compulsory

1
2
4
8
16
32
64
128

Cache Size (KB)

Miss rate (relative) components – 3 C's

100%
80%
60%
40%
20%
0%

Miss Rate per Type

1-way
Conflict
2-way
4-way
8-way
Capacity

1 2 4 8 16 32 64 128

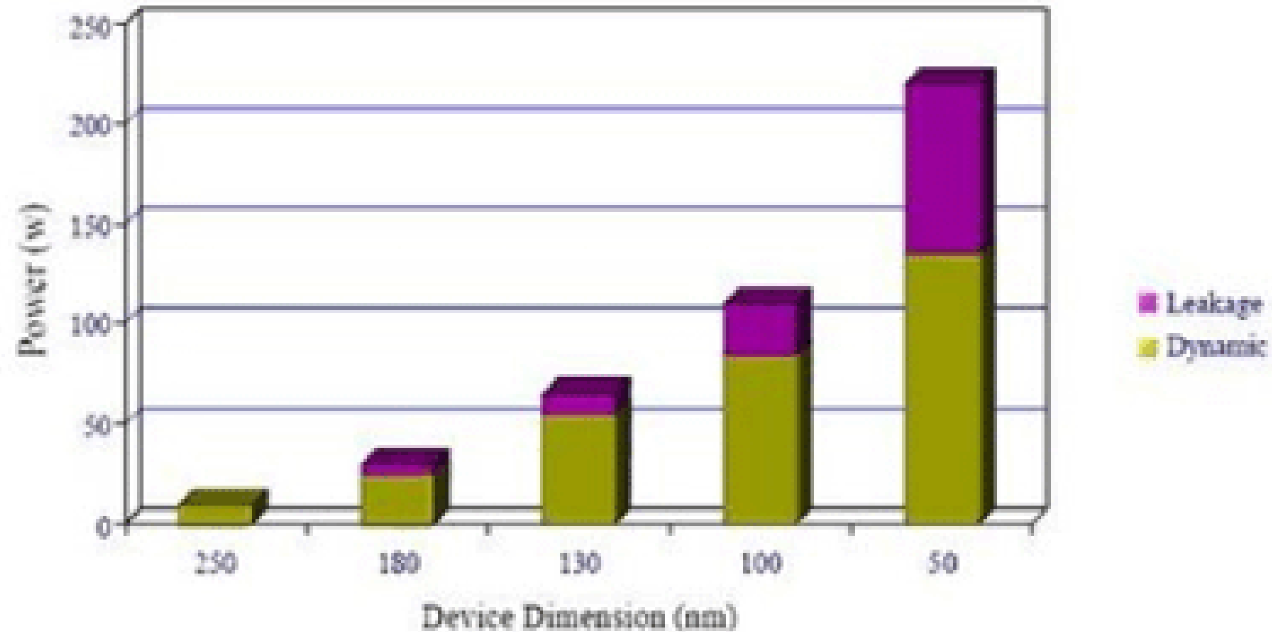
Cache Size (KB)

Compulsory

Miss rate components

	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(er)	Medium	High

Cache size: power



Size	Leakage	Dynamic
8M Byte	76mW	30mW

Miss rate components – 3 C's

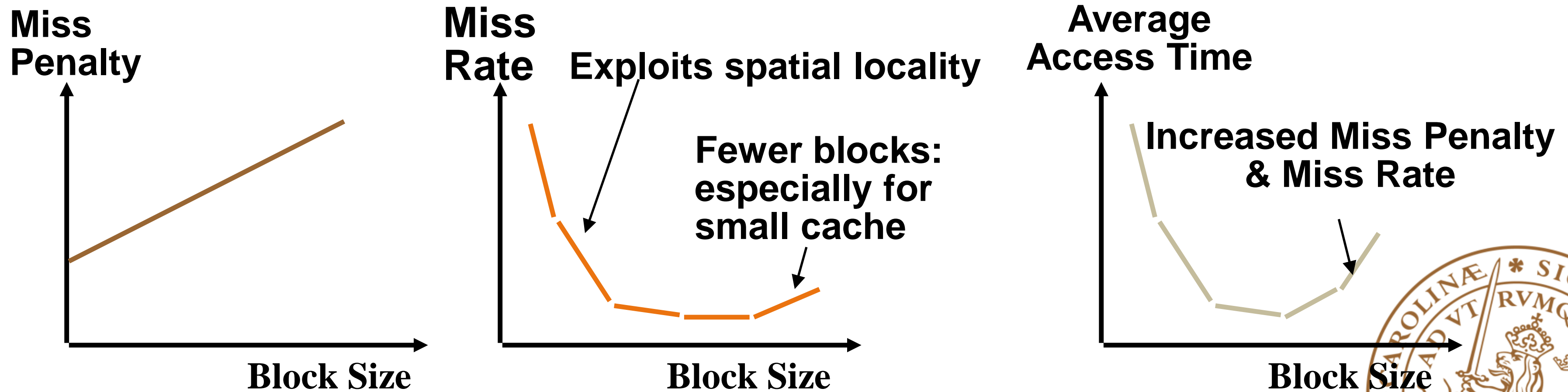
- Small percentage of compulsory misses
- Capacity misses are reduced by larger caches
- Full associativity avoids all conflict misses
- Conflict misses are relatively more important for small set-associative caches

Miss may move from one to another!

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

$$\text{Average memory access time} = \textit{hit time} + \textit{miss rate} * \textit{miss penalty}$$



Cache optimizations

	Hit time	Band-width	Miss penalty	Miss rate	HW complexity
Simple	+			-	0
Addr. transl.	+				1
Way-predict	+				1
Trace	+				3
Pipelined	-	+			1
Banked		+			1
Nonblocking		+	+		3
Early start			+		2
Merging write			+		1
Multilevel			+		2
Read priority			+		1
Prefetch			+	+	2-3
Victim			+	+	2
Compiler				+	0
Larger block			-	+	0
Larger cache	-			+	1
Associativity	-			+	1

Outline

- Reiteration
- Cache performance optimization
- **Bandwidth increase**
- Reduce hit time
- Reduce miss penalty
- Reduce miss rate
- Summary

Cache optimizations

	Hit time	Band-width	Miss penalty	Miss rate	HW complexity
Simple	+			-	0
Addr. transl.	+				1
Way-predict	+				1
Trace	+				3
Pipelined	-	+			1
Banked		+			1
Nonblocking		+	+		3
Early start			+		2
Merging write			+		1
Multilevel			+		2
Read priority			+		1
Prefetch			+	+	2-3
Victim			+	+	2
Compiler				+	0
Larger block			-	+	0
Larger cache	-			+	1
Associativity	-			+	1

Pipelined Cache

Address and Store Data From CPU

Tag

Store Data

Delayed Write Addr.

Delayed Write Data

Load/Store

=?

S

L

Tags

Data

=?

1

0

Load Data to CPU

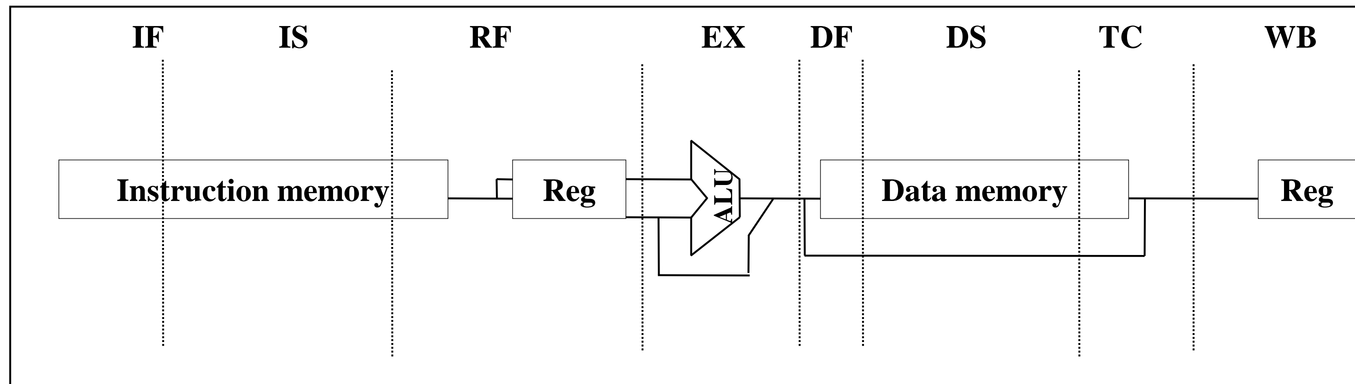
Hit?

Greater penalty on mispredicted branches and data hazard

The MIPS R4000

8 Stage Pipeline:

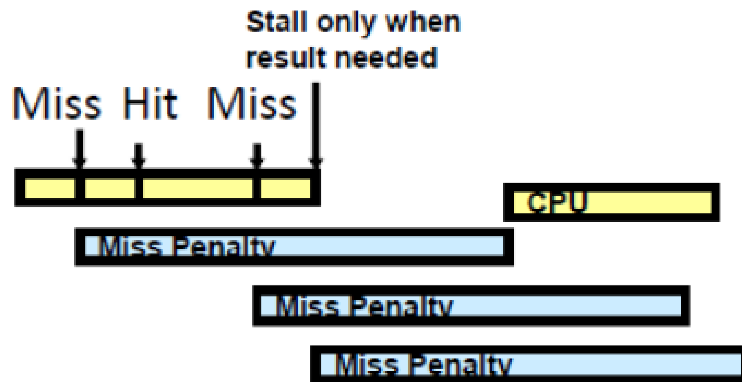
- IF – first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access
- IS – second half of access to instruction cache
- RF – instruction decode and register fetch, hazard checking and also instruction cache hit detection
- EX – execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation
- DF – data fetch, first half of access to data cache
- DS – second half of access to data cache
- TC – tag check, determine whether the data cache access hit
- WB – write back for loads and register-register operations



Non-blocking Cache

Stall CPU on miss

Hit under miss



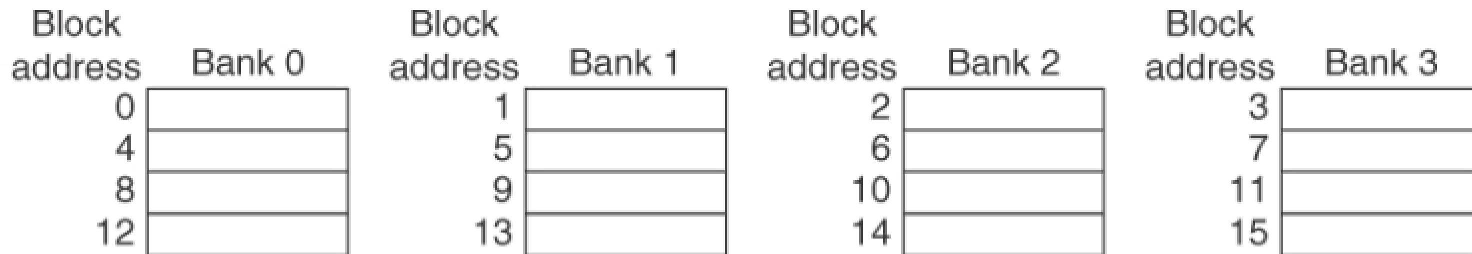
Multiple out-standing misses

Significantly increases the complexity of the cache controller

Non-blocking Cache

- **Non-blocking cache or lockup-free**
  - Allow data cache to continue to supply cache hits during a miss
- **“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
  - Pentium Pro allows 4 outstanding memory misses
- **Hardware (comparing to OOO exe)?**
  - Registers and queues for tracking multiple memory requests
  - Memory that supports multiple request: like multi-band memory (or structure harzard)
  - Control logic to keep track of dependencies and ensure precise exceptions

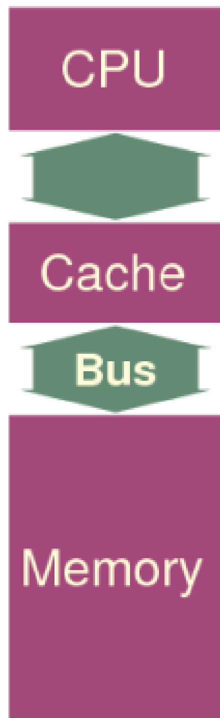
Multi-bank

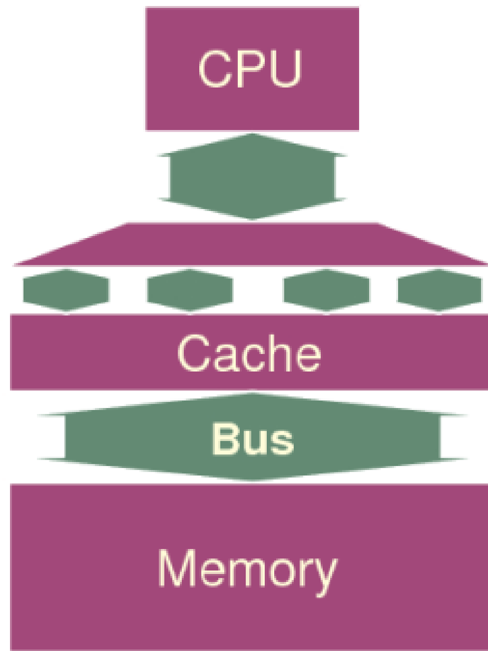


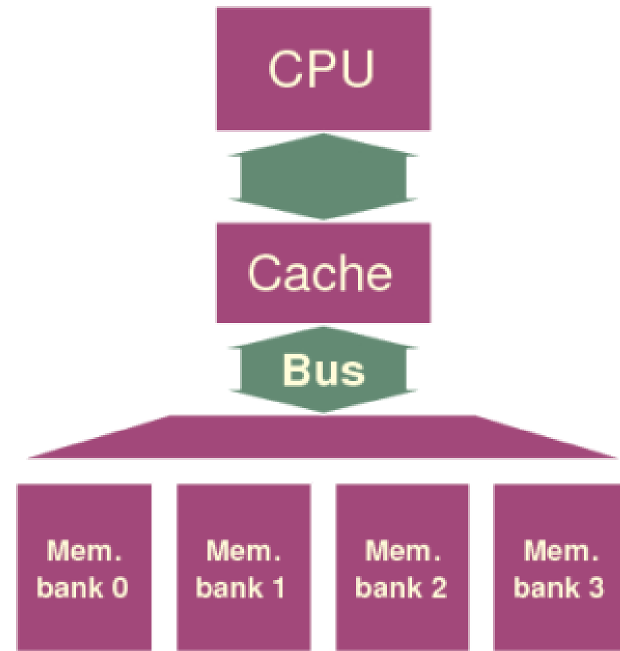
Multi-banked caches

- Divide into independent banks that can support simultaneous accesses (e.g. vector processor/SIMD)
- 4 in L1 and 8 in L2 for Intel core i7
- Works best when even spread of accesses across banks (can simultaneous access or interleaving)
- Sequential interleaving

Improving main memory performance







normal memory

wide memory

interleaving

Improves bandwidth.

Interleaving

Normal

addr	access + cycle	data

addr	access + cycle	data

addr	access + cycle	data

addr	access + cycle	data

Interleaving 4–way

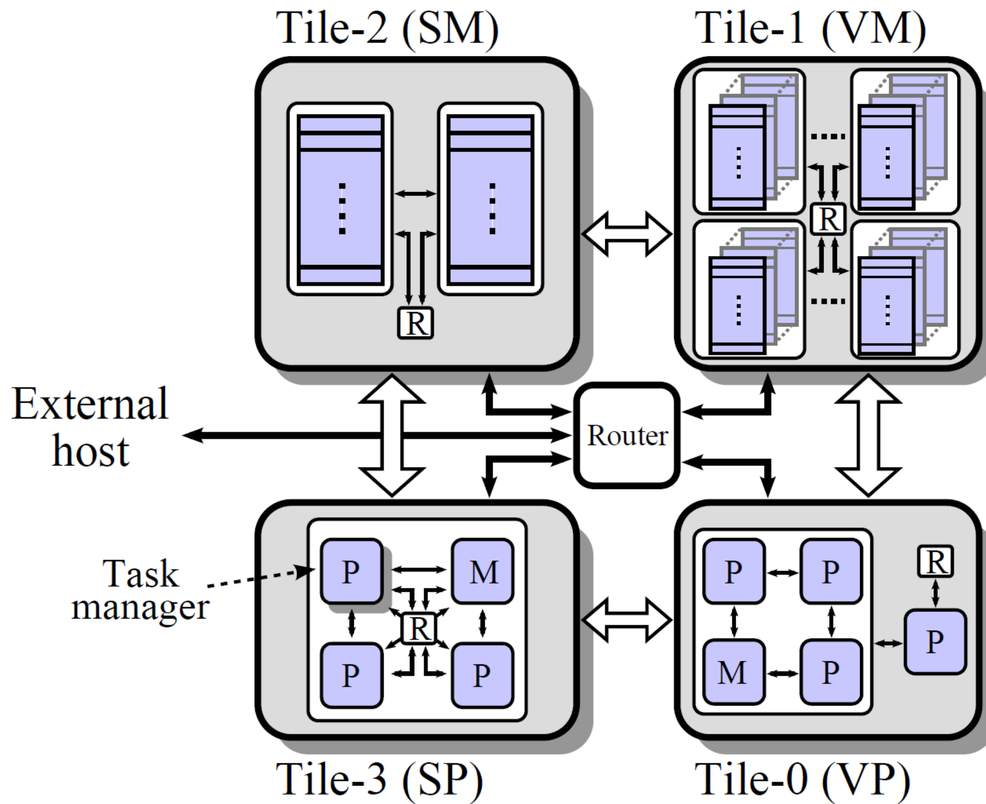
addr	access + cycle	data

addr	access + cycle	data

addr	access + cycle	data

addr	access + cycle	data

Excel-programmed vector processor



Bandwidth

- Access two 4*4 matrix/clk
- Each element is 4Byte
- Clock rate, 500MHz
- 64GB/s

Excel-programmed vector processor

Vector partition

Memory cell

Matrix partition

Memory page

Access index reg.

N

Reg. bank

Crossbar

IO bank

Flexible access

- **Matrix storage**
  - Accesss row-wise or colunm-wise in one clock cycle

1	2	3
4	5	6
7	8	9

Multi-bank memory

1	2	3
4	5	6
7	8	9

Interleaved storage

1	2	3
5	6	4
9	7	8

Outline

- Reiteration
- Cache performance optimization
- Bandwidth increase
- **Reduce hit time**
- Reduce miss penalty
- Reduce miss rate
- Summary

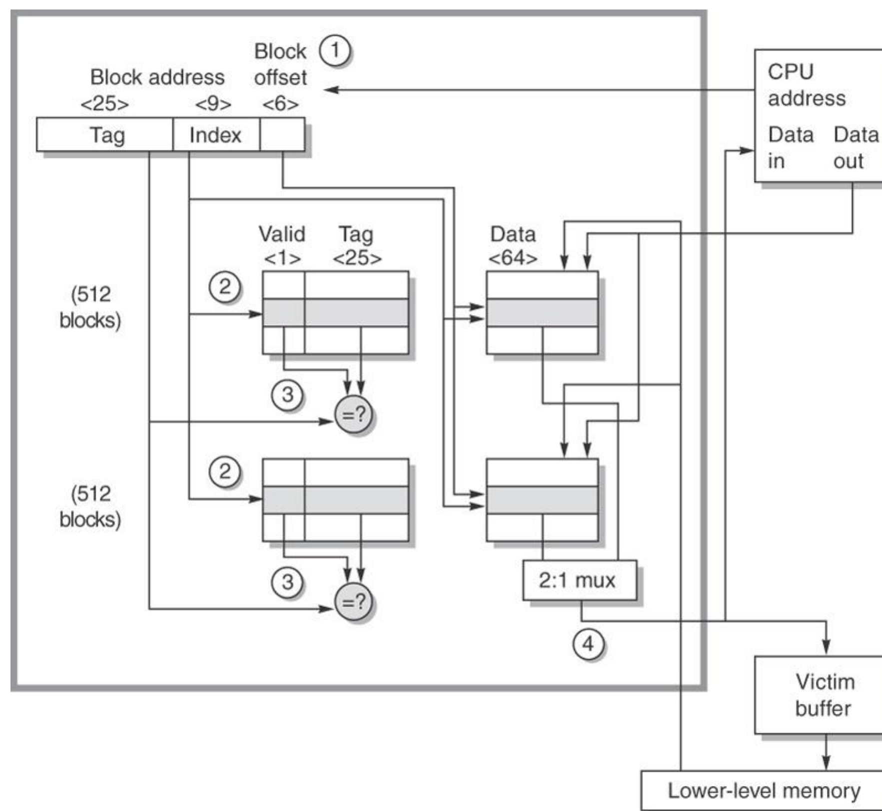
Cache optimizations

	Hit time	Band-width	Miss penalty	Miss rate	HW complexity
Simple	+			-	0
Addr. transl.	+				1
Way-predict	+				1
Trace	+				3
Pipelined	-	+			1
Banked		+			1
Nonblocking		+	+		3
Early start			+		2
Merging write			+		1
Multilevel			+		2
Read priority			+		1
Prefetch			+	+	2-3
Victim			+	+	2
Compiler				+	0
Larger block			-	+	0
Larger cache	-			+	1
Associativity	-			+	1

Reduce hit time 1: KISS (Keep It Simple, Stupid)

Hit time critical since it affects clock rate.

- **Smaller and simpler is faster:**
  - Fits on-chip (game changing by technology evolution)
  - Simple cache allows data fetch and tag check to proceed in parallel

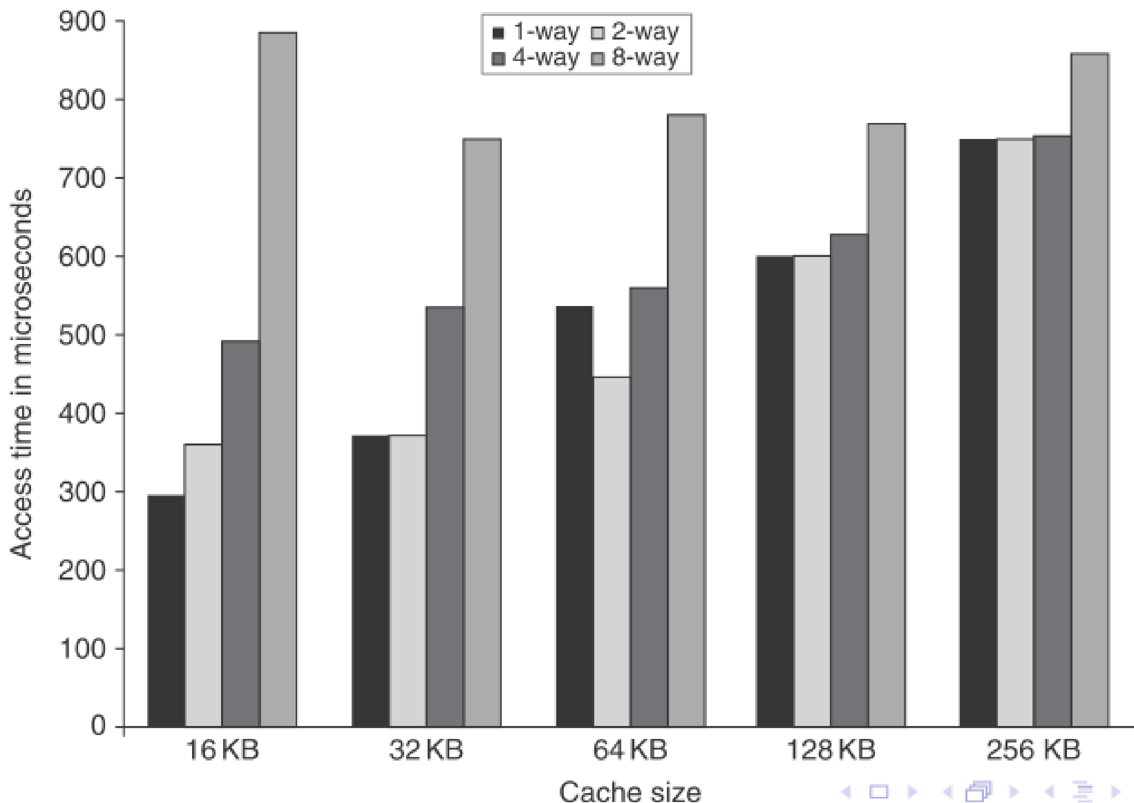


© 2007 Elsevier, Inc. All rights reserved.

Reduce hit time 1: KISS (Keep It Simple, Stupid)

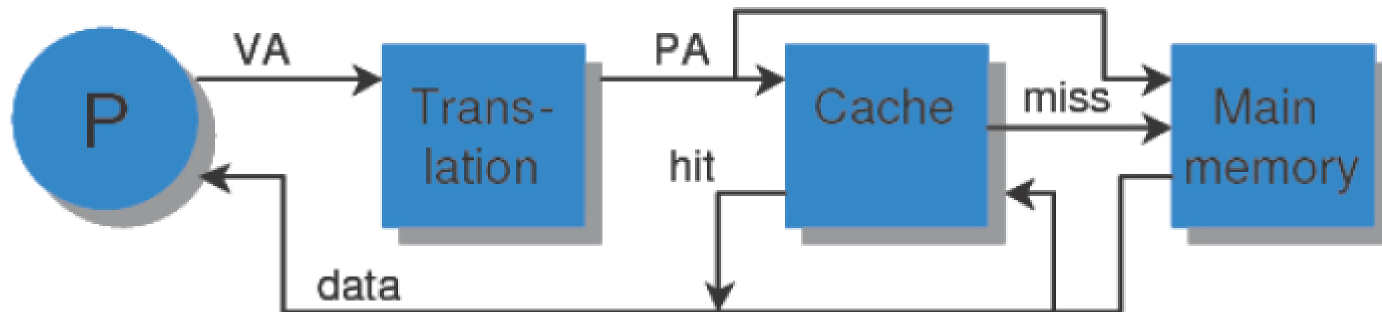
Hit time critical since it affects clock rate.

- **Smaller and simpler is faster:**
  - Fits on-chip (game changing by technology evolution)
  - Simple cache allows data fetch and tag check to proceed in parallel

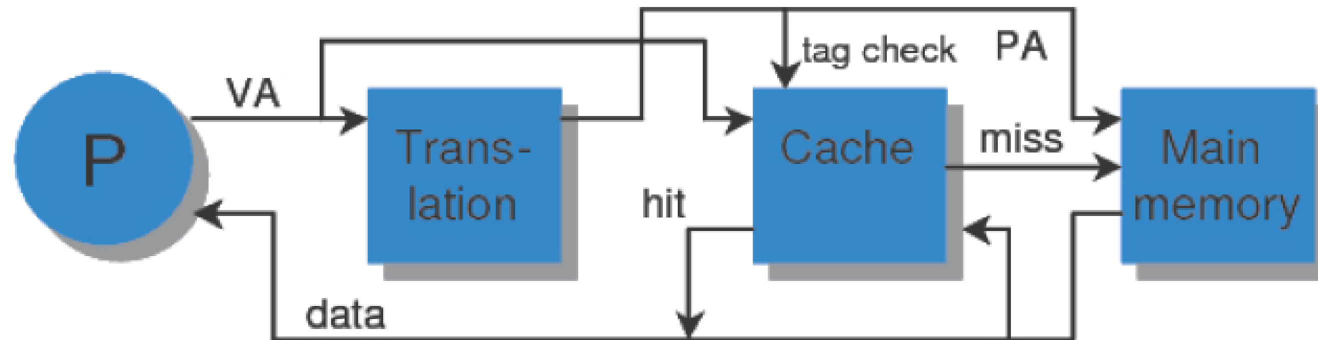


Reduce hit time 2: Address translation

- Processor uses virtual addresses (VA) while caches and main memory use physical addresses (PA)

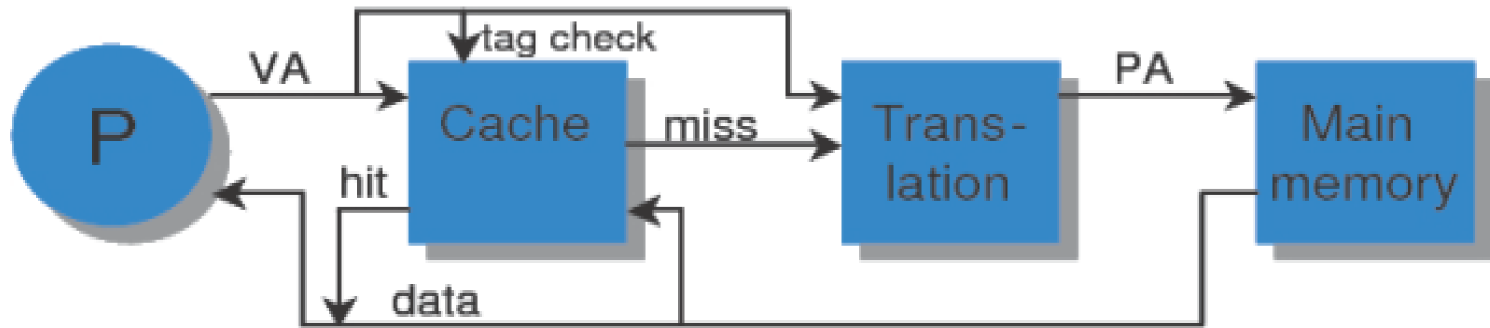


- Use the virtual address to index the cache in parallel



Reduce hit time 2: Address translation

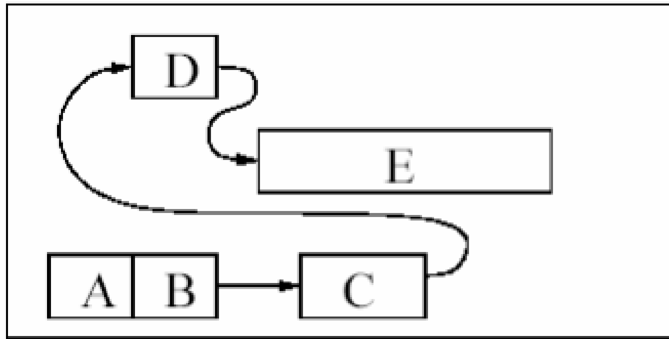
- Use virtual addresses to both index cache and tag check



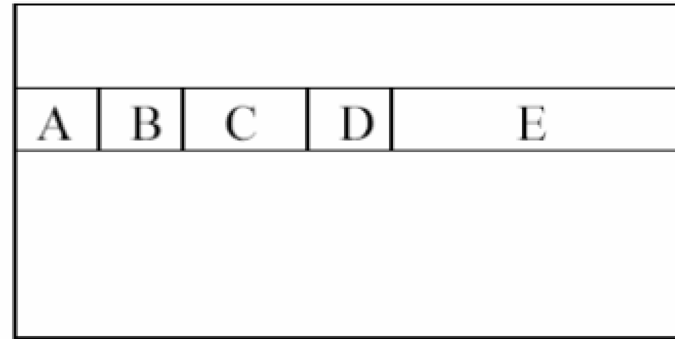
- Processes have different virtual address spaces
- Two virtual addresses may map to the same physical address – synonyms or aliases

Reduce hit time 3: trace caches

- Dynamically find a sequence of executed instructions (including taken branches) to make up a cache block



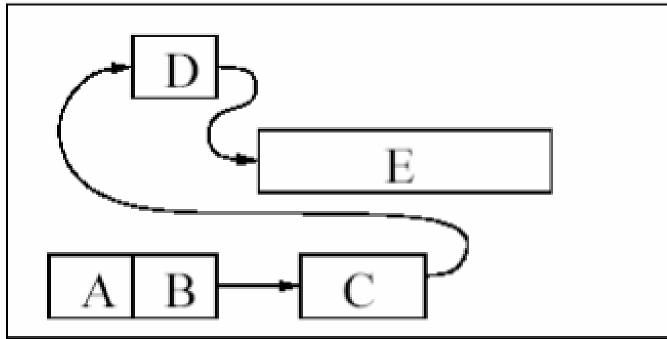
(a) Instruction cache.



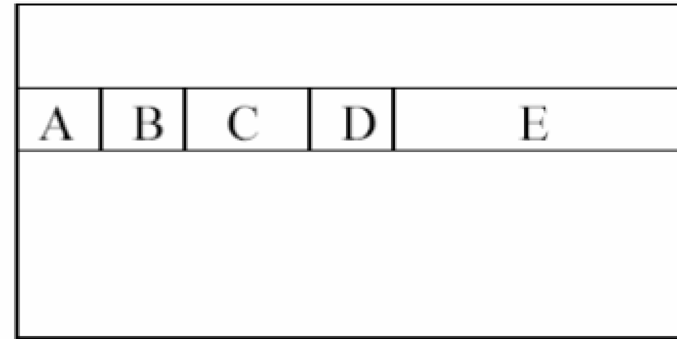
(b) Trace cache.

- A trace is a sequence of instructions starting at any point in a dynamic instruction stream
- It is specified by a start address and the branch outcomes of control transfer instructions

Reduce hit time 3: trace caches



(a) Instruction cache.



(b) Trace cache.

- **Trace cache is accessed in parallel with instruction cache**
  - Hit ->Trace read into issue buffer
  - Miss -> from instruction cache
- **Trace cache hit if**
  - Fetch address match
  - Branch predictions match
- **Trace cache is NOT on the critical path of instruction fetch**

Reduce hit time 3: trace caches

Instruction Cache

2nd BB

1st BB

3rd BB

A

n

Instruction Latch

To Instruction Buffers

Fetch Address A

0

1

n

Trace Cache

n

1st BB 2nd BB 3rd BB

A

hit?

Take output from trace cache if trace cache hit; otherwise, take output from instruction cache.

Line-Fill Buffer

n

Reduce hit time 3: trace caches

Integer Fetch IPC as a Function of Trace Cache Size

Fetch IPC

13.5
12.5
11.5
10.5
9.5
8.5
7.5
6.5

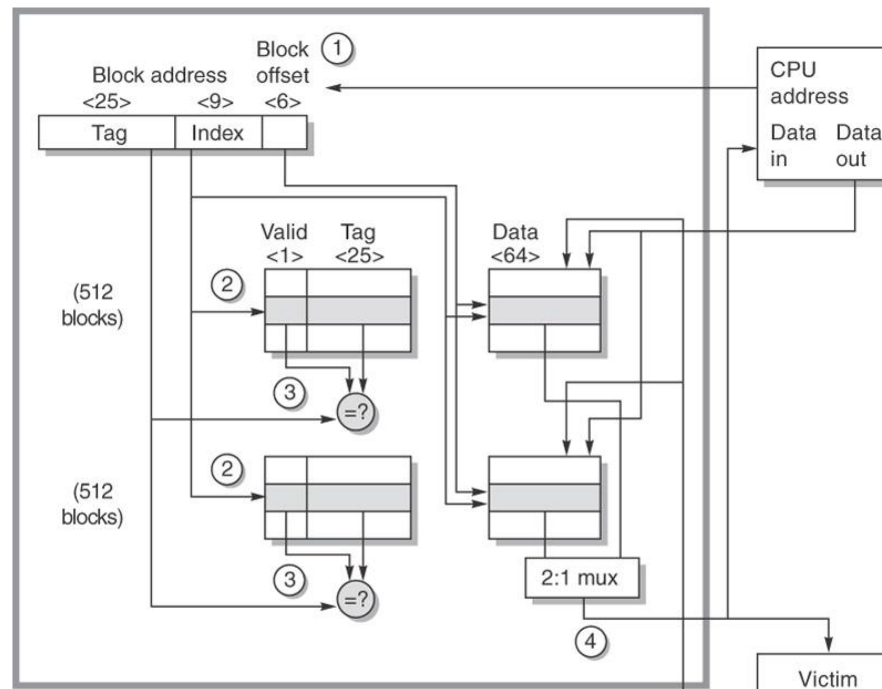
0 500 1000 1500 2000 2500 3000 3500 4000

Trace Cache Entries

go
m88ksim
gcc
compress
li
ijpeg
perl
vortex

Reduce hit time 4: way prediction

- **How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?**
- **Way prediction: keep extra bits in cache to predict the "way" or block within the set, of next cache access.**
  - Multiplexor is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data
  - Miss -> check other blocks for matches in next clock cycle



Reduce hit time 4: way prediction

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Way prediction: keep extra bits in cache to predict the "way" or block within the set, of next cache access.
  - Multiplexor is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data
  - Miss -> check other blocks for matches in next clock cycle

- Accuracy: 90% for 2-way and 80% for 4-way (ARM Cortex-A8)
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles

Outline

- Reiteration
- Cache performance optimization
- Bandwidth increase
- Reduce hit time
- **Reduce miss penalty**
- Reduce miss rate
- Summary

Cache optimizations

	Hit time	Band-width	Miss penalty	Miss rate	HW complexity
Simple	+			-	0
Addr. transl.	+				1
Way-predict	+				1
Trace	+				3
Pipelined	-	+			1
Banked		+			1
Nonblocking		+	+		3
Early start			+		2
Merging write			+		1
Multilevel			+		2
Read priority			+		1
Prefetch			+	+	2-3
Victim			+	+	2
Compiler				+	0
Larger block			-	+	0
Larger cache	-			+	1
Associativity	-			+	1

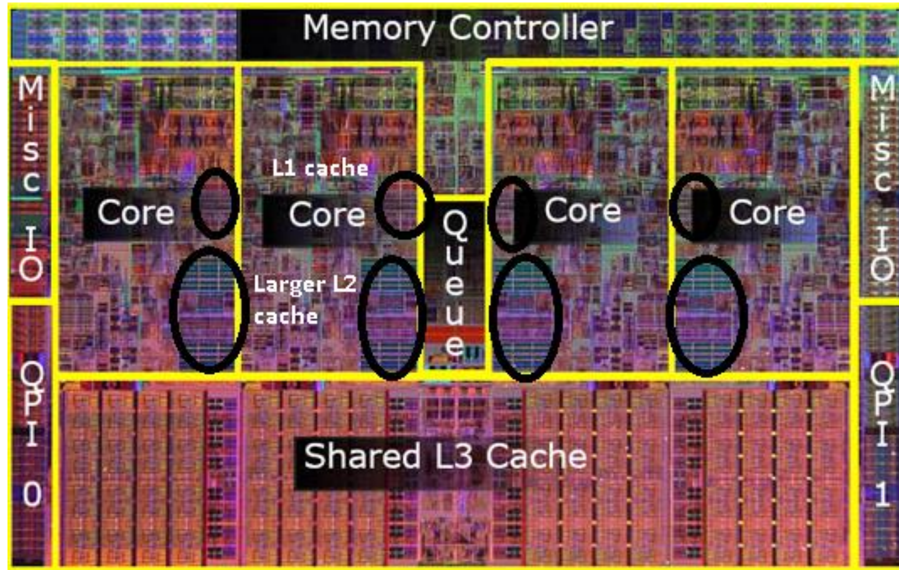
Reduce miss penalty 1: Multilevel caches

Use several levels of cache memory:

- The 1st level cache **fast and small** ⇒ match processing speed
- 2nd level cache can be made much **larger and set-associative** to reduce capacity and conflict misses
- ... and so on for 3rd and 4th level caches

On-chip or Off-chip?

- Today 4 levels on-chip



Broadwell

CPUID code	000306D4
Product code	80658
L1 cache	64 KB per core
L2 cache	256 KB per core
L3 cache	2–6 MB (shared)
L4 cache	128 MB of eDRAM (Iris Pro models only)
Created	2014
Transistors	14 nm transistors

Reduce miss penalty 1: Multilevel caches

- **Use several levels of cache memory:**
  - The 1st level cache fast and small ⇒ match processing speed
  - 2nd level cache can be made much larger and set-associative to reduce capacity and conflict misses
  - ... and so on for 3rd and 4th level caches
- **On-chip or Off-chip?**
  - Today 4 levels on-chip

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

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

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

Multilevel caches: execution time

Second-level cache size (KB)	L2 hit = 8 clock cycles	L2 hit = 16 clock cycles
8192	1.02	1.06
4096	1.10	1.14
2048	1.60	1.65
1024	1.76	1.82
512	1.94	1.99
256	2.34	2.39

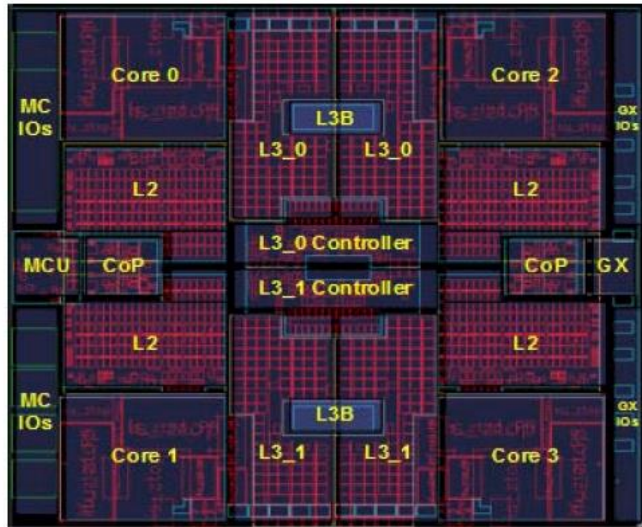
1.00 1.25 1.50 1.75 2.00 2.25 2.50

Relative execution time

Multilevel caches: examples

CPU	CP	Cache		
	CP GHz	L1 KB	L2 KB	L3 MB
FX-51	2.2	64+64	1024	-
Itanium 2	1.5	16+16	256	6
Pentium 4	3.2	12+8	512	-
(Pentium 4 EE)	3.2	12+8	512	2
Core i7	3.5	32+32	256	8
Phenom II	3	128	512	8
AMD Bulldozer	4	16+64	2048	8
IBM z196	5.2	64+128	1536	24

IBM z196



zEnterprise 196

Reduce miss penalty 2: Write buffers, Read priority

Write through:

- Using write buffers: RAW conflicts with reads on cache misses (first write is still in the buffer when the LW needs the value)

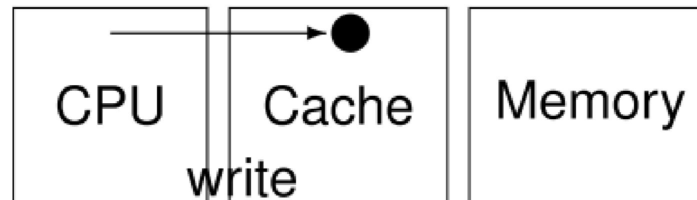
```
SW R3, 512(R0)      ;M[512] ← R3      (cache index 0)
LW R1, 1024(R0)     ;R1 ← M[1024]     (cache index 0)
LW R2, 512(R0)      ;R2 ← M[512]      (cache index 0)
```

- If simply wait for write buffer to empty might increase read miss penalty by 50% (old MIPS 1000)
- Check write buffer contents before read; if no conflicts, let the memory access continue
- Complicated cache control

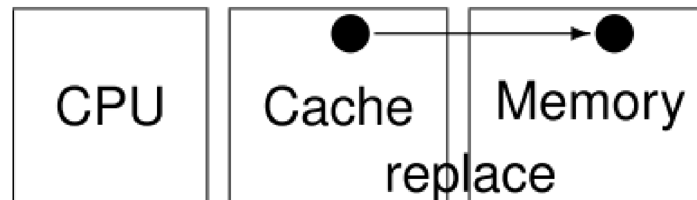
Reduce miss penalty 2: Write buffers, Read priority

Write Back:

- Read miss replacing dirty block
- Normal: Write dirty block to memory, and then do the read (very long latency and stalls the processor)
- 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 read completes



...



Reduce miss penalty 2: Write buffers, Read priority

Merging write buffers

- Multi-word writes more efficient to memory
- The Sun T1 (Niagara) processor, among many others, uses write merging

Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Reduce miss penalty 3: other tricks

Impatience

Don't wait for full block before restarting CPU

- **Early restart –** fetch words in normal order but restart processor as soon as requested word has arrived
- **Critical word first –** fetch the requested word first. Overlap CPU execution with filling the rest of the cache block

Increases performance mainly with large block sizes.

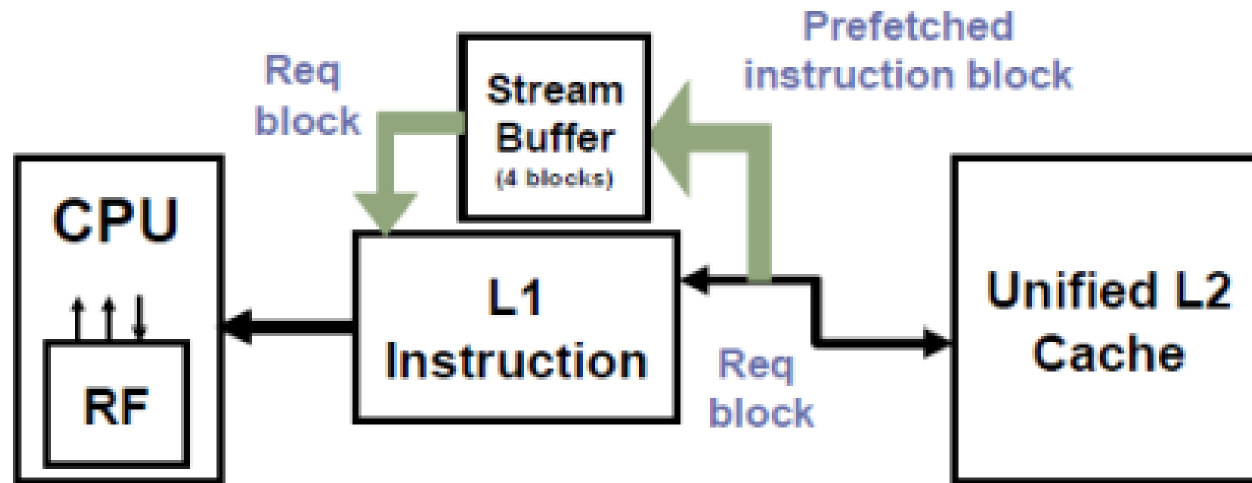
Reduce miss penalty 4: Non-blocking caches

- **Non-blocking cache ≡ lockup-free cache**
  - (+) Permit other cache operations to proceed when a miss has occurred
  - (+) May further lower the effective miss penalty if multiple misses can overlap
  - (-) The cache has to book-keep all outstanding references –Increases cache controller complexity
- **Good for out-of-order pipelined CPUs**
  - The presence of true data dependencies may limit performance
  - Requires pipelined or banked memory system (otherwise cannot support)

Reduce miss rate/penalty: hardware prefetching

Goal: overlap execution with speculative prefetching to cache

- **Hardware prefetching -** If there is a miss for block X, fetch also block X+1, X+2,... X+d
  - Instruction prefetching
    - **Alpha 21064 fetches 2 blocks on a miss (Intel i7 on L1 and L2)**
    - **Extra block placed in stream buffer or caches**
    - **On miss check stream buffer (highly possible is there)**
  - Works with data blocks too (generally better with I-Cache but depending on application)



Reduce miss rate/penalty: hardware prefetching

Goal: overlap execution with speculative prefetching to cache

Potential issue

- Complicated cache control
- Relies on extra memory bandwidth that can be used without penalty
- Only useful if produce hit for next reference
- May polute cache (useful data is replaced)

Performance improvement

Suite	Benchmark	Performance improvement
SPECint2000	gap	1.16
SPECint2000	mcf	1.45
SPECfp2000	fam3d	1.18
SPECfp2000	wupwise	1.20
SPECfp2000	galgel	1.21
SPECfp2000	facerec	1.26
SPECfp2000	swim	1.29
SPECfp2000	applu	1.32
SPECfp2000	lucas	1.40
SPECfp2000	mgrid	1.49
SPECfp2000	equake	1.97

© 2007 Elsevier, Inc. All rights reserved.

Outline

- Reiteration
- Cache performance optimization
- Bandwidth increase
- Reduce hit time
- Reduce miss penalty
- **Reduce miss rate**
- Summary

Cache optimizations

	Hit time	Band-width	Miss penalty	Miss rate	HW complexity
Simple	+			-	0
Addr. transl.	+				1
Way-predict	+				1
Trace	+				3
Pipelined	-	+			1
Banked		+			1
Nonblocking		+	+		3
Early start			+		2
Merging write			+		1
Multilevel			+		2
Read priority			+		1
Prefetch			+	+	2-3
Victim			+	+	2
Compiler				+	0
Larger block			-	+	0
Larger cache	-			+	1
Associativity	-			+	1

Reduce miss rate

- **The three C's:**
  - Compulsory – misses in an infinite cache
  - Capacity – misses in a fully associative cache
  - Conflict – misses in an N-way associative cache
- **How do we reduce the number of misses?**
  - Change cache size?
  - Change block size?
  - Change associativity?
  - Change compiler?
  - Other tricks!

Which of the three C's are affected?

Reduce misses 1: increase block size

- Increased block size utilizes the spatial locality
- Too big blocks increases miss rate
- Big blocks also increases miss penalty

Miss rate

10%

5%

0%

4K

16K

64K

256K

16 32 64 128 256

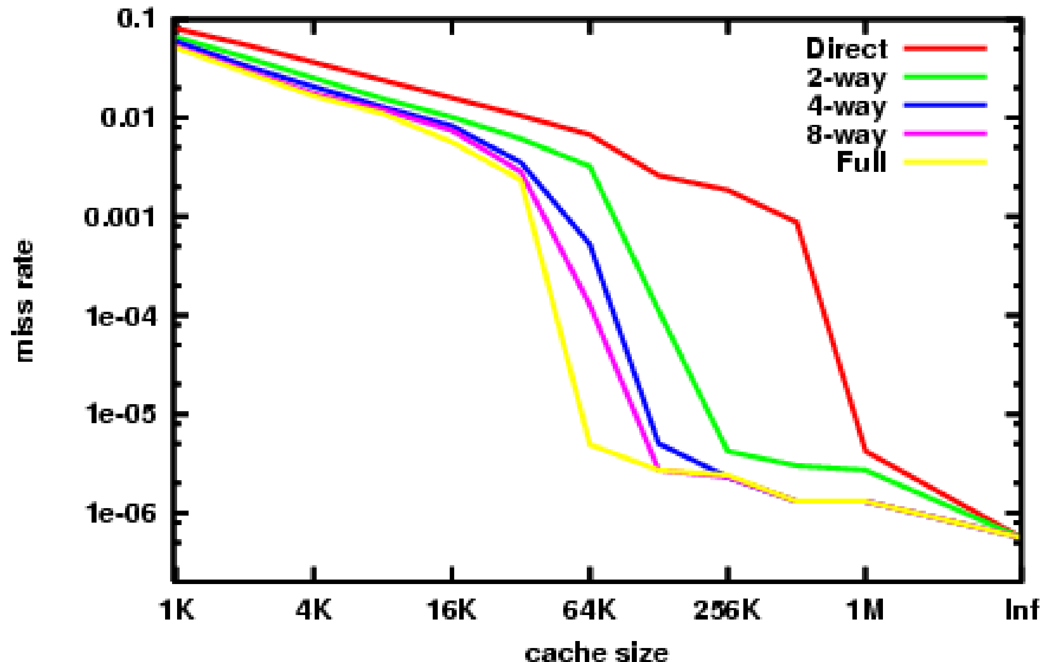
Block size

© 2007 Elsevier, Inc. All rights reserved.

Beware - impact on average memory access time

Reduce misses 2: change associativity

Rule of thumb: A direct mapped cache of size N has the same miss rate as a 2-way set associative cache of size N/2



- **Hit time increases with increasing associativity**

Beware - impact on average memory access time

Reduce misses 3: Compiler optimizations

Basic idea: Reorganize code to improve locality

- **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 two 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

Reduce misses 3: Compiler optimizations

Merging Arrays

- Improve spatial locality by single array of compound elements vs. 2 arrays

```
/* Before */
int val[SIZE];
int key[SIZE];
```

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

Reduces conflicts between `val` and `key`

Reduce misses 3: Compiler optimizations

Loop Interchange

- Change nesting of loops to access data in order stored in memory
- If x[i][j] and x[i][j+1] are adjacent (row major)

```
/* Before */
for (k = 0; k < 100; k++)
  for (j = 0; j < 100; j++)
    for (i = 0; i < 5000; i++)
      x[i][j] = 2 * x[i][j];
```

```
/* After */
for (k = 0; k < 100; k++)
  for (i = 0; i < 5000; i++)
    for (j = 0; j < 100; j++)
      x[i][j] = 2 * x[i][j];
```

Depending on the storage of the matrix
Sequential accesses instead of striding through memory every 100 words

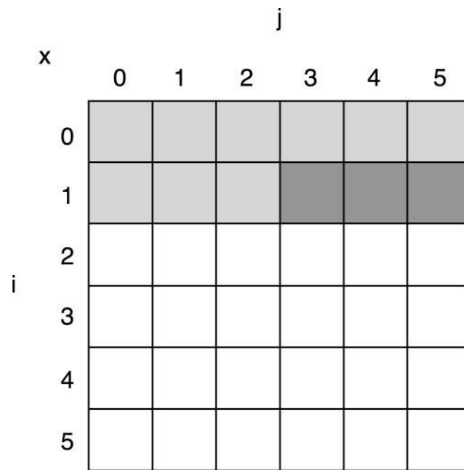
Reduce misses 3: Compiler optimizations

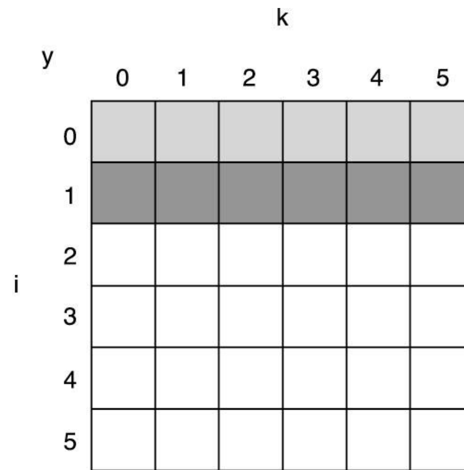
- Block (matrix multiplication)

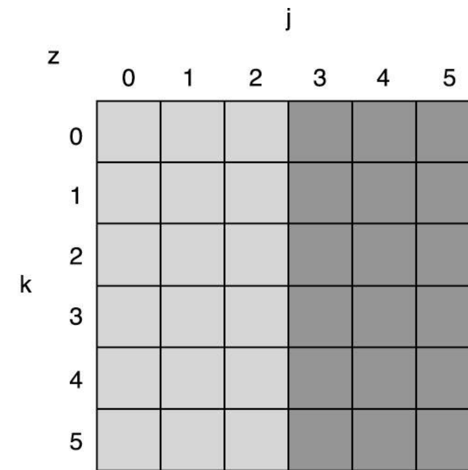
```
/* Before */
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++) {
     r = 0;
     for (k = 0; k < N; k++)
       r = r + y[i][k]*z[k][j];
     x[i][j] = r;
  }
```

Reduce misses 3: Compiler optimizations

- White means not touched yet
- Light gray means touched a while ago
- Dark gray means newer accesses





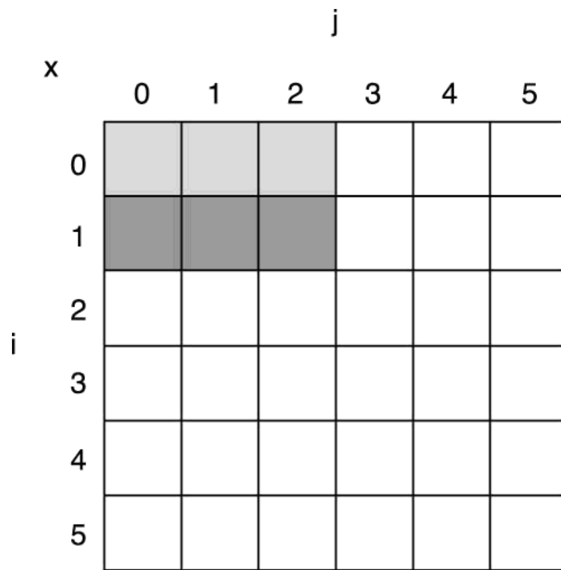


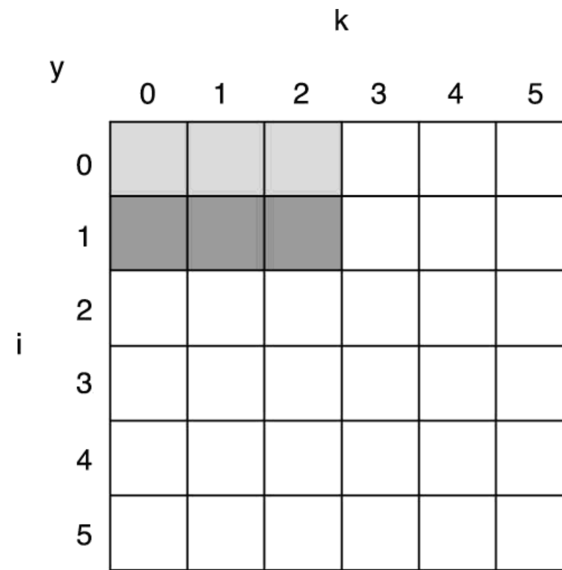
© 2003 Elsevier Science (USA). All rights reserved.

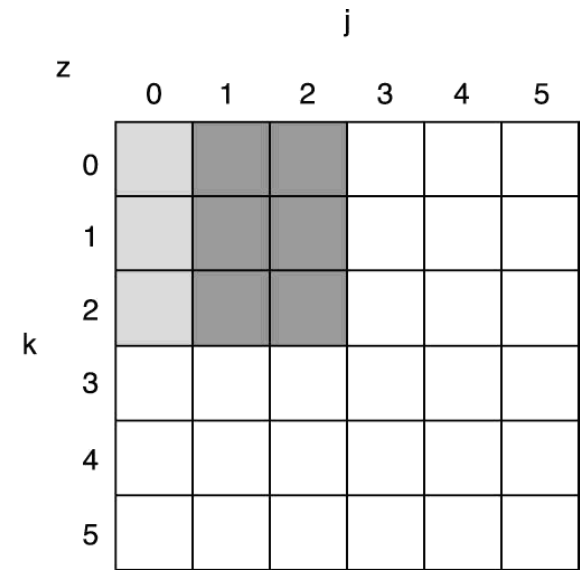
Reduce misses 3: Compiler optimizations

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

Reduce misses 3: Compiler optimizations







Reduce misses 3: Compiler optimizations

Summary of Compiler Optimizations to Reduce Cache Misses

vpenta (nasa7)
gmty (nasa7)
tomcatv
btrix (nasa7)
mxm (nasa7)
spice
cholesky (nasa7)
compress

1 1.5 2 2.5 3

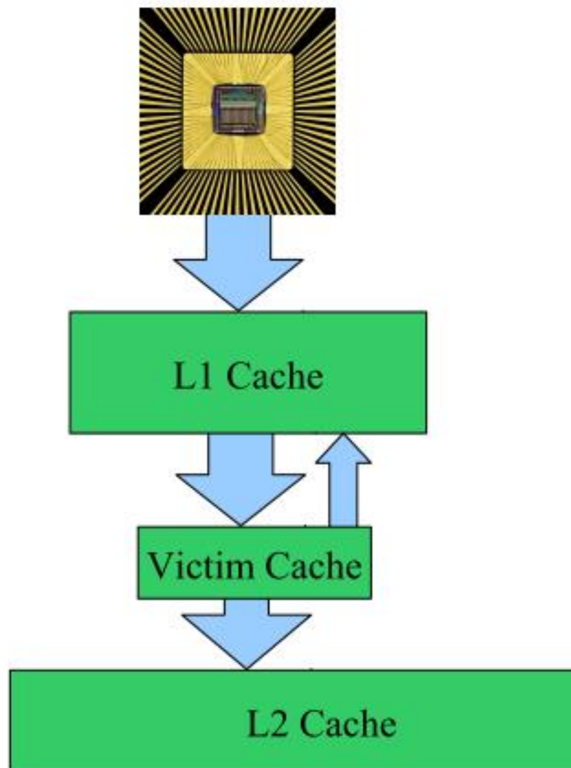
Performance Improvement

merged arrays
loop interchange
loop fusion
blocking

29

Reduce misses 4: Victim cache

How to combine fast hit time of direct mapped yet still avoid conflict misses?



- **Victim cache operation**
  - On a miss in L1, we check the Victim Cache
  - If the block is there, then bring it into L1 and swap the ejected value into the victim cache
  - If not, fetch the block from the lower levels
- **Norman Jouppi,1990**
  - a 4-entry victim cache removed **25%** of conflict misses for a 4 Kbyte direct mapped cache
- **Used in AMD Athlon, HP and Alpha machines**

Outline

- Reiteration
- Cache performance optimization
- Bandwidth increase
- Reduce hit time
- Reduce miss penalty
- Reduce miss rate
- **Summary**

Cache performance

$$\text{Execution Time} =$$

$$IC * (CPI_{execution} + \frac{\text{mem accesses}}{\text{instruction}} * \textbf{miss rate} * \textbf{miss penalty}) * \mathbf{T_C}$$

Three ways to increase performance:

- Reduce miss rate
- Reduce miss penalty
- Reduce hit time
- ... and increase bandwidth

remember:

Execution time is the only true measure!

Cache optimization

	Hit time	Band-width	Miss penalty	Miss rate	HW complexity
Simple	+			-	0
Addr. transl.	+				1
Way-predict	+				1
Trace	+				3
Pipelined	-	+			1
Banked		+			1
Nonblocking		+	+		3
Early start			+		2
Merging write			+		1
Multilevel			+		2
Read priority			+		1
Prefetch			+	+	2-3
Victim			+	+	2
Compiler				+	0
Larger block			-	+	0
Larger cache	-			+	1
Associativity	-			+	1