Lecture 7: EITF20 Computer Architecture

Anders Ardö

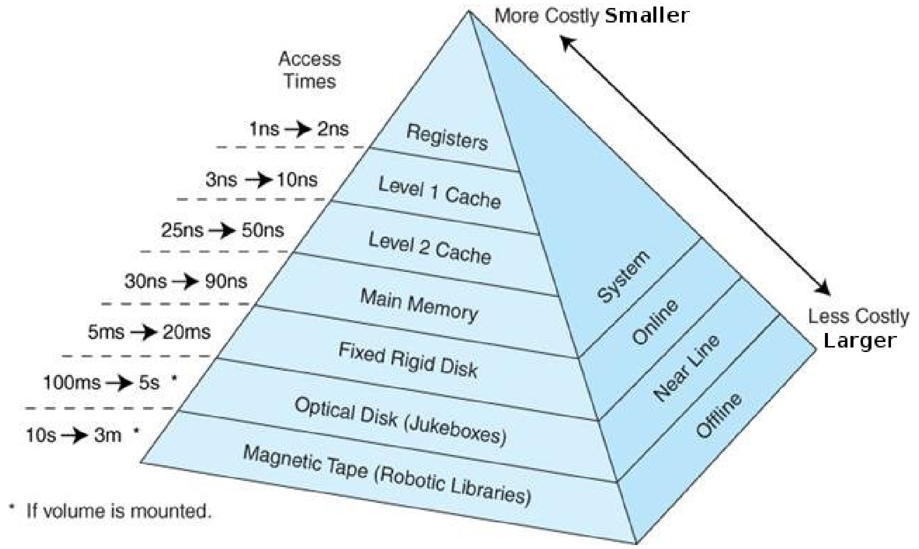
EIT – Electrical and Information Technology, Lund University

November 21, 2012

Outline

1 Reiteration
2 Cache performance optimization
3 Bandwidth increase
4 Reduce hit time
5 Reduce miss penalty
6 Reduce miss rate
7 Summary

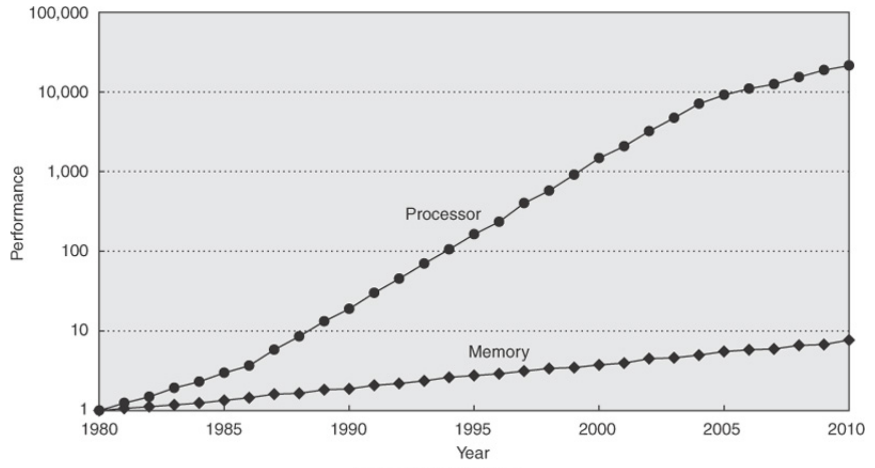
Memory hierarchy



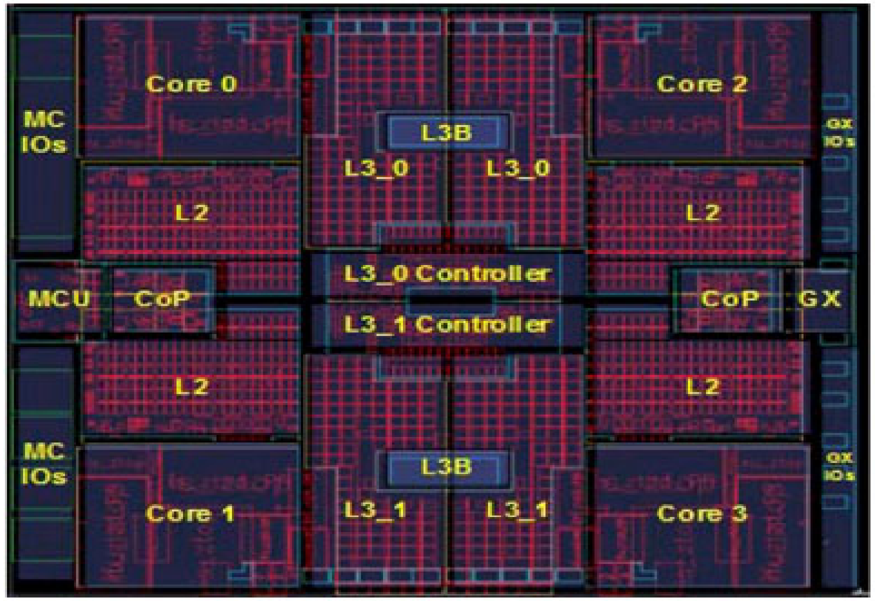
AIM: Fast as cache; Large as disk; Cheap as possible

Who cares?

- 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



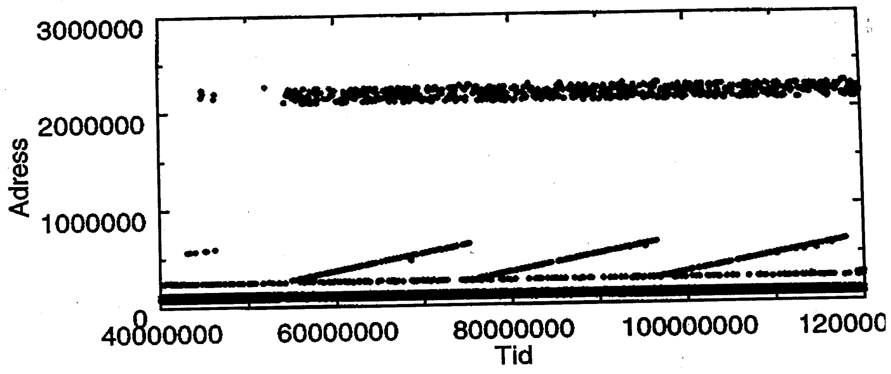
IBM z196 5.2GHz microprocessor chip



Why does caching work?

The Principle 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 $= 1.0 - \textit{hit rate}$
- 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

  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
  $= \textit{hit time} + \textit{miss rate} * \textit{miss penalty}$

4 questions for the Memory Hierarchy

- 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)***

Lecture 7 agenda

Chapter 5.2, (App C.2-C.3) in "Computer Architecture"
Cache optimizations

1. Reiteration
2. Cache performance optimization
3. Bandwidth increase
4. Reduce hit time
5. Reduce miss penalty
6. Reduce miss rate
7. Summary

Outline

1. Reiteration
2. Cache performance optimization
3. Bandwidth increase
4. Reduce hit time
5. Reduce miss penalty
6. Reduce miss rate
7. 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 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

1 Reiteration

2 Cache performance optimization

3 Bandwidth increase

4 Reduce hit time

5 Reduce miss penalty

6 Reduce miss rate

7 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

Bandwidth increase

- Pipelined cache
  - greater penalty on misspredicted branches
- Multibanked caches
  - works best when even spread of accesses across banks
  - sequential interleaving

Outline

1 Reiteration

2 Cache performance optimization

3 Bandwidth increase

4 Reduce hit time

5 Reduce miss penalty

6 Reduce miss rate

7 Summary

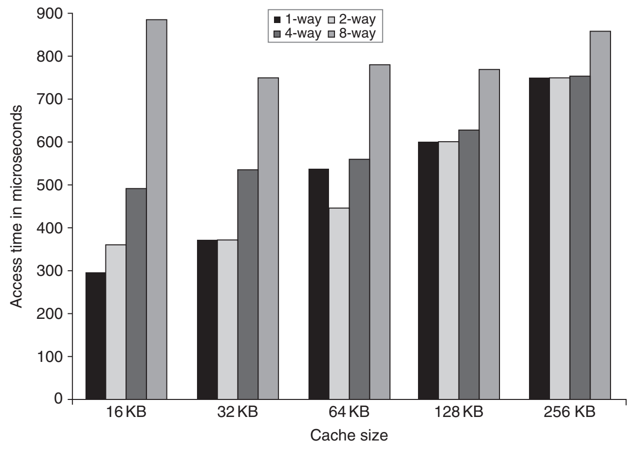
Cache optimizations

	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)

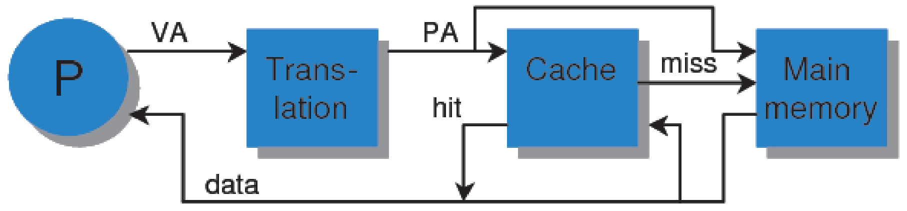
Hit time critical since it affects clock rate.

- Smaller and simpler is faster:
  - Fits on-chip ($\rightarrow$ avoids off-chip time penalties)
  - A direct mapped cache allows data fetch and tag check to proceed in parallel

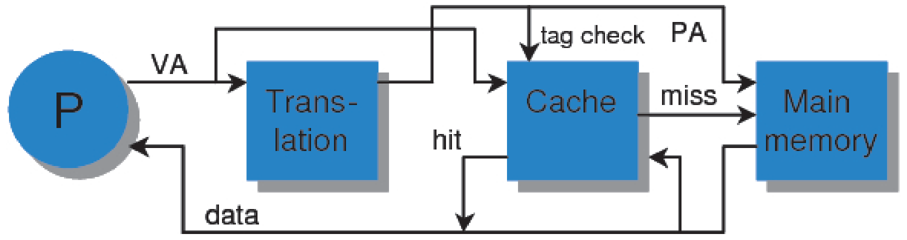


Reduce hit time 2: Address translation

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

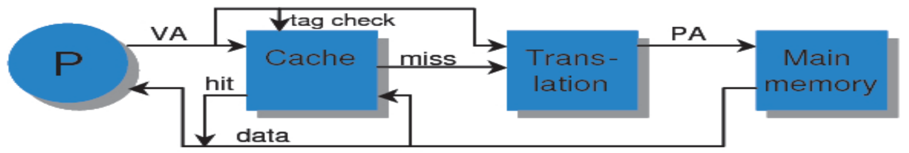


Use the virtual address to index the cache in parallel



Reduce hit time 3: Address translation

- Use virtual addresses to both index cache and tag check



Processes have different virtual address spaces

- Flush cache between context switches or
- Extend tag with process-identifier tag (PID)

Two virtual addresses may map to the same physical address – synonyms or aliases

- HW guarantees that every cache block have a unique physical address
- Some OS uses "page colouring" to ensure that the cache index is unique

Reduce hit time 4: trace caches

- Trace caches
  - Dynamically find a sequence of executed instructions (including taken branches) to make up a cache block
  - Complicated address mapping
  - Avoids header and trailer overhead of normal cache schemes
  - The same instruction can be stored several times in the cache (different traces)
  - Used in Intel NetBurst (P4)

Reduce hit time: various tricks

- **Way prediction** combines the speed of a direct mapped cache with the conflict miss reductions of set-associative caches
  - Hardware prediction of which block in the set that will be accessed next (Compare branch prediction)
  - If prediction correct low latency otherwise longer latency
  - Prediction accurancy $\approx$ 85 %
- ***Pseudo associativity*** **–** (i) check first a direct mapped entry, if miss (ii) check a second entry
  - Two hit times (fast, slow)
  - If hit rate in the fast entry is low, the performance will be dictated by the slow hit time

Outline

1 Reiteration

2 Cache performance optimization

3 Bandwidth increase

4 Reduce hit time

5 Reduce miss penalty

6 Reduce miss rate

7 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

The more the merrier

Use several levels of cache memory:

- The 1st level cache fast and small $\Longrightarrow$ on-chip
- 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?

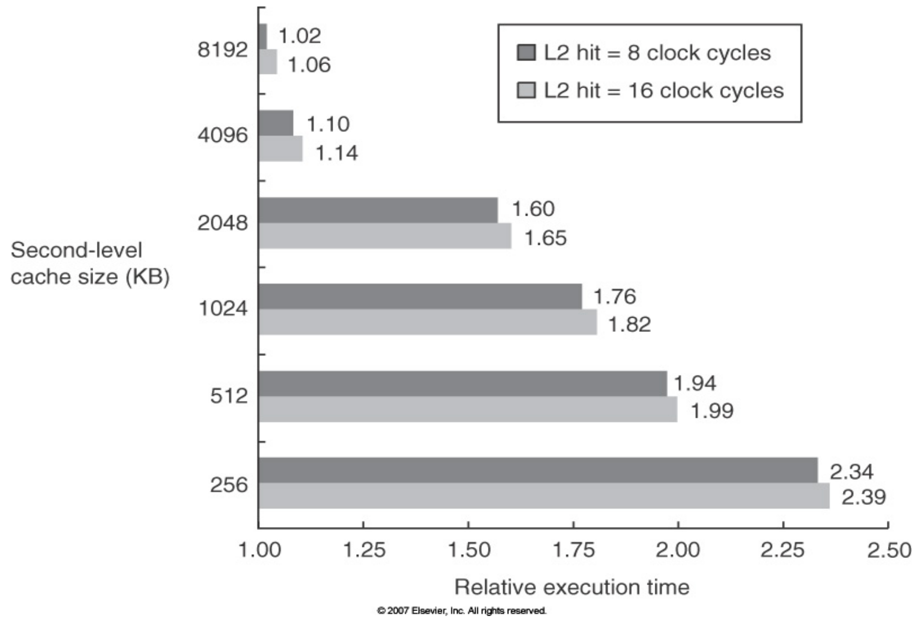
Today 3 levels on-chip; 4th comming

Performance:

- Average Access time$_{L1}$ $=$ Hit$_{L1}$ $+$ Miss rate$_{L1}$ $*$ **Miss penalty$_{L1}$**
- **Miss penalty$_{L1}$** $=$ Hit_block$_{L2}$ $+$ Miss rate$_{L2}$ $*$ Miss penalty$_{L2}$

Design issue: multilevel inclusion or exclusion?

Multilevel caches: execution time



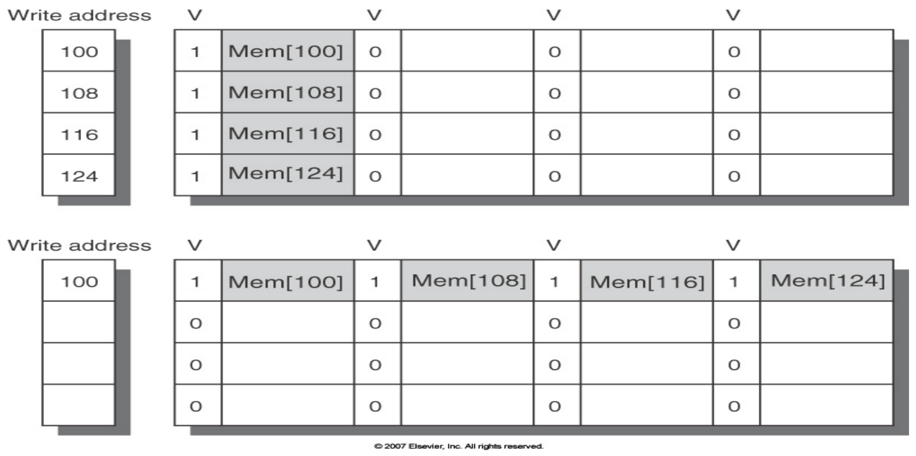
Multilevel caches: examples

		Cache			SPEC	
CPU	**CP**	**L1**	**L2**	**L3**	**Int**	**FP**
	GHz	KB	KB	MB		
FX-51	2.2	64+64	1024	-	1376	1371
Itanium 2	1.5	16+16	256	6	1322	2119
Pentium 4	3.2	12+8	512	-	1221	1271
(Pentium 4 EE)	3.2	12+8	512	2	1583	1475
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		

Reduce miss penalty 2: Write buffers, Read priority

Preference, Companionship

- Giving priority to read misses over writes
- Merging write buffer



Reduce miss penalty 3: other tricks

Impatience

- Early restart and critical word first
  - *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 perfomance mainly with large block sizes.

Reduce miss penalty 4: Nonblocking caches

Nonblocking cache ≡ lockup-free cache

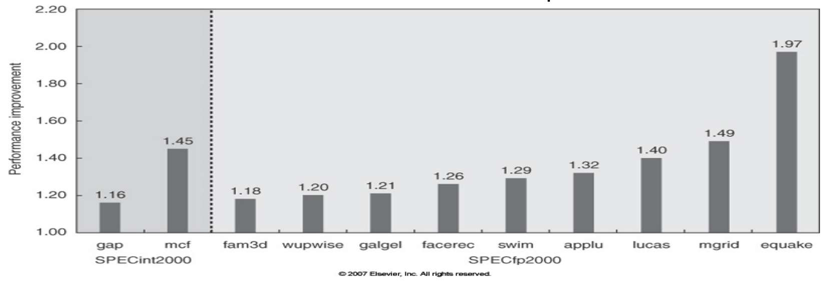
- (+) Permit other cache operations to proceed when a miss has occurred
- (+) May further lower the effective miss penalty if multiple misses can overlap
- (-) The cache has to book-keep all outstanding references – Increases cache controler complexity

Good for out-of-order pipelined CPUs
The presence of true data dependencies may limit performance

Reduce miss rate/penalty: 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
  - d=1 $\Longrightarrow$ *one-block lookahead*. Used in Alpha 21064, Pentium 4



- ***Software prefetching*** – load data before it is needed:
  - ***Register prefetching***
  - ***Cache prefetching*** – requires *special instructions* (Used in Itanium)
  - Both requires *lockup free (nonblocking) caches*
  - Can be faulting or nonfaulting (virtual memory)

Outline

1 Reiteration
2 Cache performance optimization
3 Bandwidth increase
4 Reduce hit time
5 Reduce miss penalty
6 Reduce miss rate
7 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?

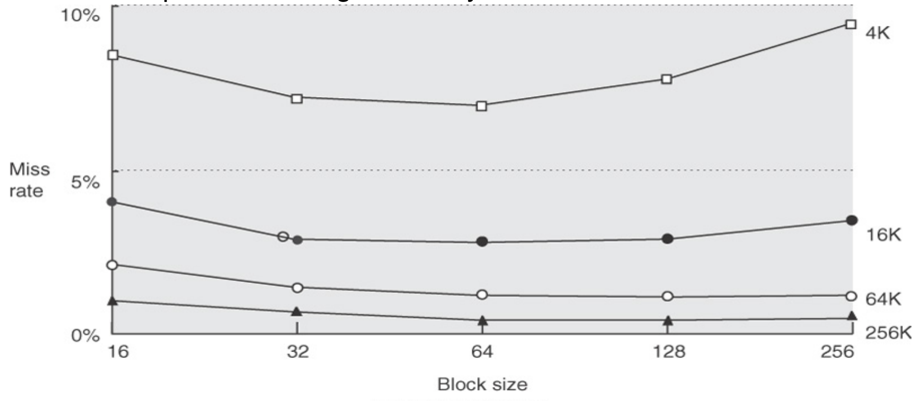
- Change cache size?
- Change block size?
- Change associativity?
- Change compiler?
- Other tricks!

Which of the three C's are affected?

Reduce misses 1: increase block size

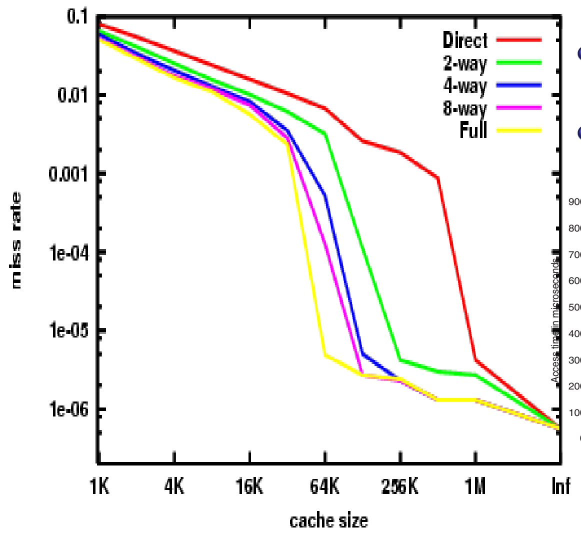
- Increased block size utilises the spatial locality
- Too big blocks increases capacity miss rate
- Big blocks also increases miss penalty

Beware - impact on average memory access time



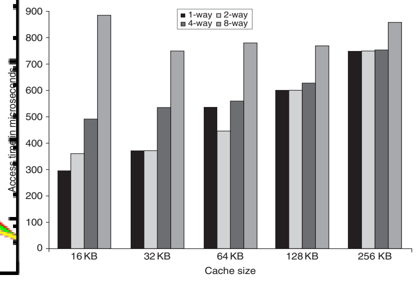
Reduce misses 2: change associativity

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



Beware

- Hit time increases with increasing associativity
- Will clock cycle time increse?



Reduce misses 3: Compiler optimizations

Basic idea: Reorganize code to increase locality!

- Merging arrays:

Before	After

int val[SIZE];
int key[SIZE];
	Struct merge { int val;
	int key; }
	struct merge newarr[SIZE];

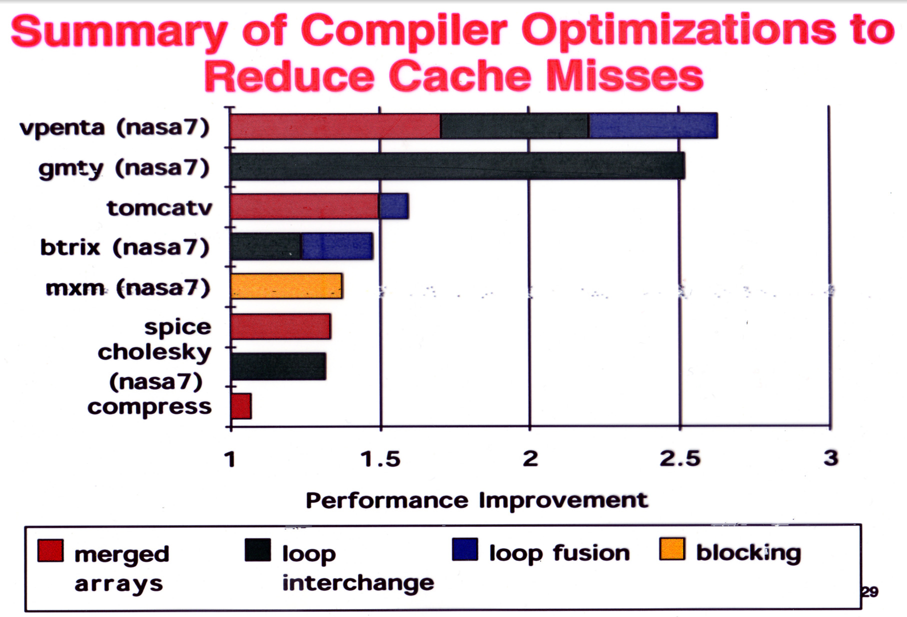
- Loop interchange:

Before	After

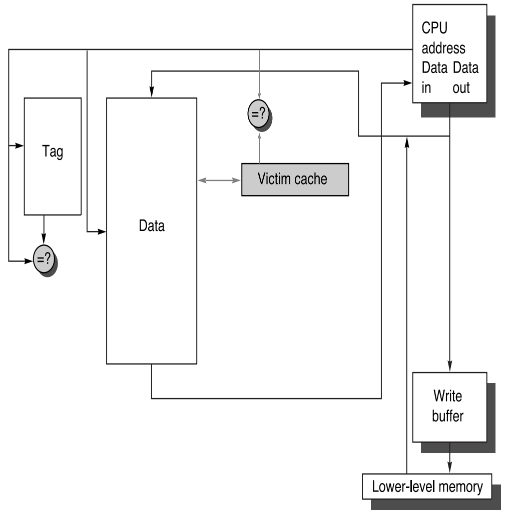
for (j=0; j<N; j++)
for (i=0; i<N; i++)
A[i,j] = ...
	for (i=0; i<N; i++)
	for (j=0; j<N; j++)
	A[i,j] = ...

- Blocking

Reduce misses 3: Compiler optimizations



Reduce misses 4: Victim cache



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

Recycling

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi: a 4-entry victim cache removed 25% of conflict misses for a 4 Kbyte direct mapped data cache
- Used in AMD Athlon, HP and Alpha machines

Outline

1 Reiteration
2 Cache performance optimization
3 Bandwidth increase
4 Reduce hit time
5 Reduce miss penalty
6 Reduce miss rate
7 Summary

Cache Performance

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

However, remember:

Execution time is the only true measure!

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