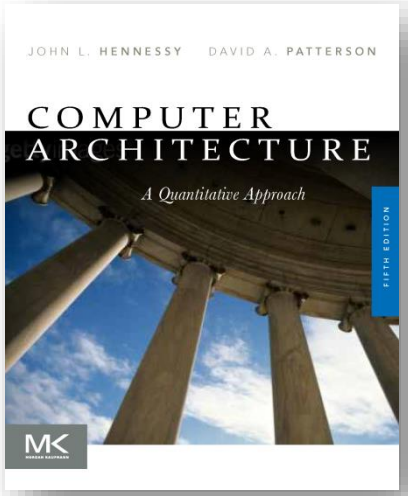


Chapter 2

Memory Hierarchy Design

Outline

- Introduction
- Ten Advanced Optimizations of Cache Performance
- Memory Technology and Optimizations
- Protection: Virtual Memory and Virtual Machines
- Crosscutting Issues: The Design of Memory Hierarchies
- Fallacies and Pitfalls

Introduction

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

Principle of Locality

資料或是指令在被存取時有區域性特性

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

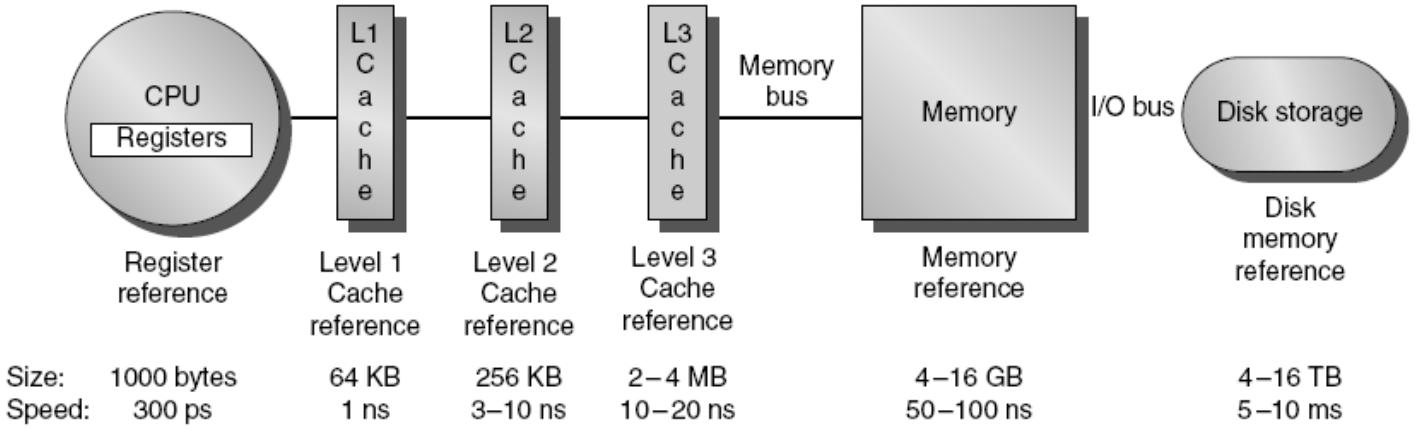
a variable that gets increased or decreased by a fixed amount on every iteration of a loop

Taking Advantage of Locality

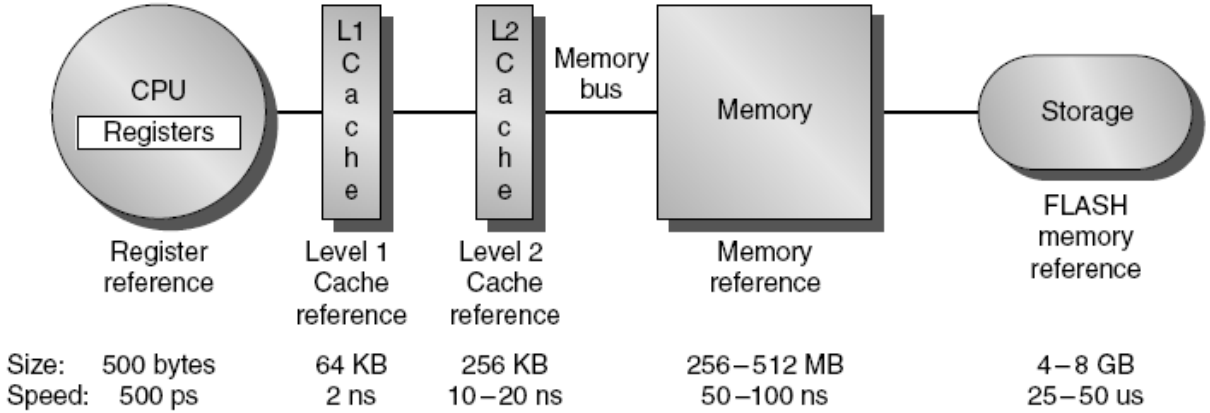
- Memory hierarchy
  - Store everything on disk
  - Copy recently accessed (and nearby) items from disk to smaller DRAM memory
    - Main memory
  - Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
    - Cache memory attached to CPU

Disk→Memory→Cache→CPU

Memory Hierarchy



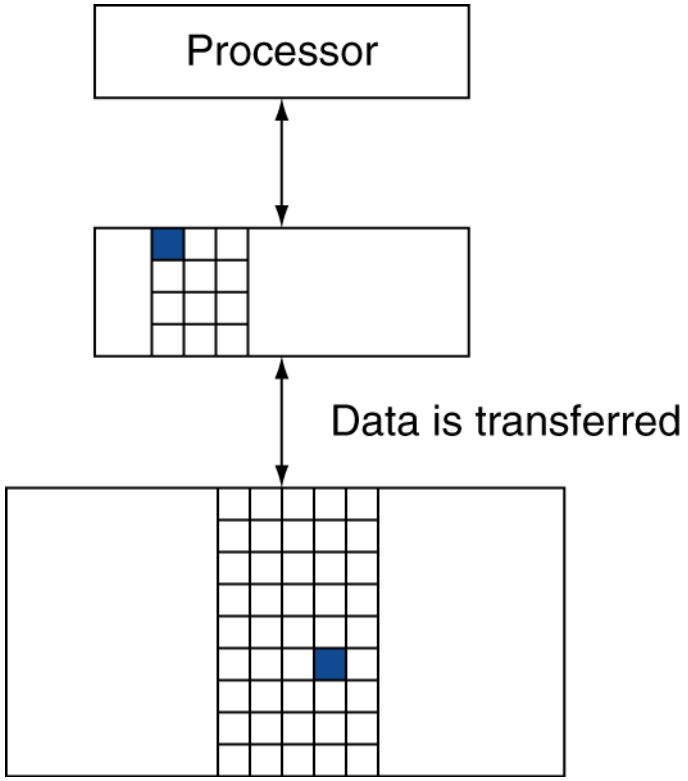
(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Memory Hierarchy Levels

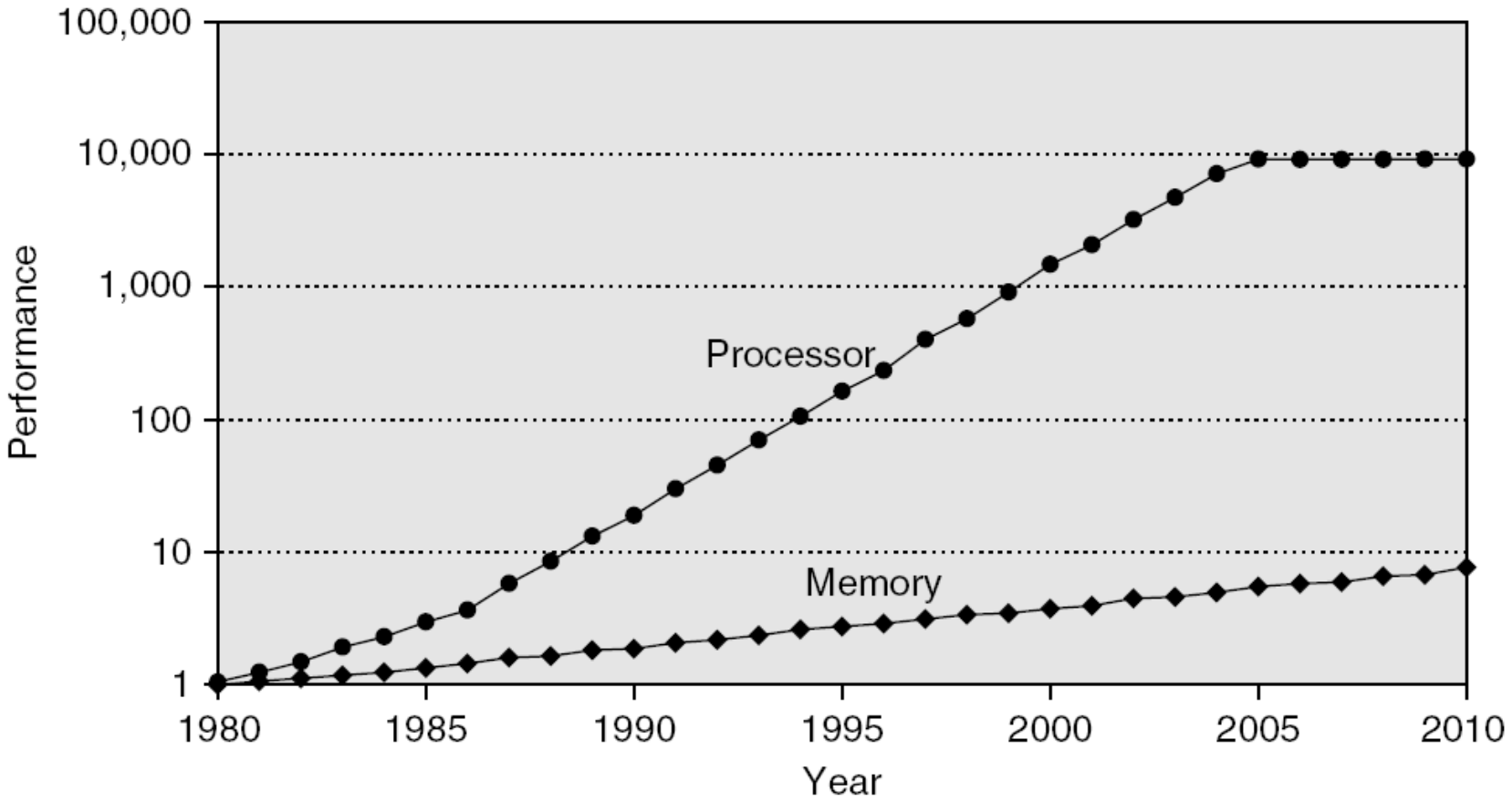
Cache與Memory之間的傳送基本單位



- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

當欲存取資料不在cache內時，需要先到memory搬到cache後，再從cache存取

Memory Performance Gap



Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references
      - = 409.6 GB/s!
    - DRAM bandwidth is only 6% of this (25 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Performance and Power

- High-end microprocessors have >10 MB on-chip cache
  - Consumes large amount of area and power budget

Memory Hierarchy Basics

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

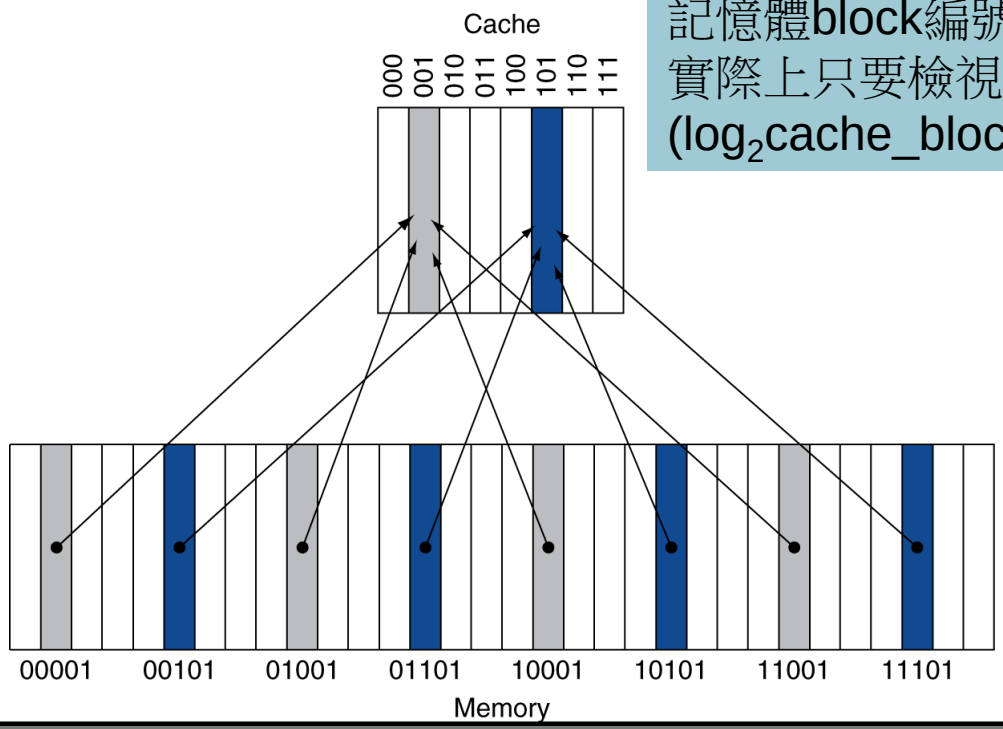
Review of Caches

- When a word is not found in the cache, the word must be fetched from the memory and placed in the cache before continuing.
  - Multiple words, called a block (or line), are moved for efficiency reasons.
  - Each cache block including a tag to see which memory address it corresponds to.
- Where blocks (or lines) can be placed in a cache?
  - Set associative
  - Direct-mapped
  - Fully associative

Direct Mapped Cache

在已確定memory block編號與cache block個數下，串起兩者間的關係。

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



記憶體block編號 mod cache中block個數，其實實際上只要檢視記憶體block編號的後幾個bit ($\log_2$cache_block個數)

- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

維護成本

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

由於low-order bits已被用來指引位於哪個cache block，但由於不同的memory block會有相同的low-order bits，為辨別是哪個memory block的資料，因此還要在紀錄memory block的high-order bits(tag)

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Access words: 22,26,22,26,16,3,16,18

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

因為一個block只有一個word，所以第22個word存在第22個memory block。22%8=6，代表該memory block如果存在cache中，所對應的是第6個cache block。mod 8即為只看二進位的後3個bit。

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

22

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data	
000	N			
001	N			
010	**Y**	**11**	**Mem[11010]**	26
011	N			
100	N			
101	N			
110	Y	10	Mem[10110]	22
111	N			

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data	
000	N			
001	N			
010	Y	11	Mem[11010]	26
011	N			
100	N			
101	N			
110	Y	10	Mem[10110]	22
111	N			

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data	
000	Y	10	Mem[10000]	16
001	N			
010	Y	11	Mem[11010]	26
011	Y	00	Mem[00011]	3
100	N			
101	N			
110	Y	10	Mem[10110]	22
111	N			

Cache Example

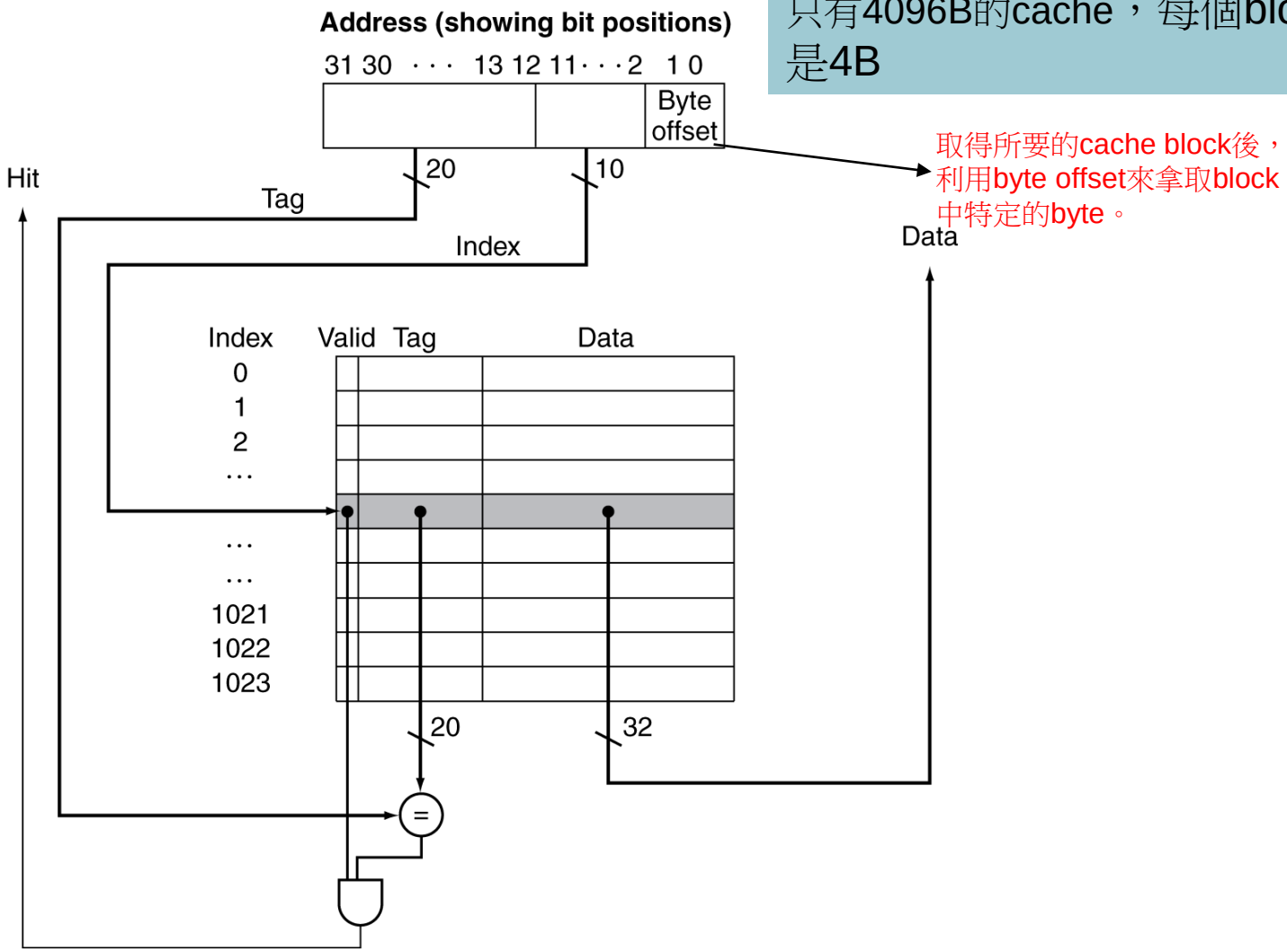
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

因為第26個memory block對應的是第2個cache block(26%8=4) ，而第18個memory block對應的也是第2個cache block(18%8=2) ，原有Cache block中的資料將會被覆蓋。

Index	V	Tag	Data	
000	Y	10	Mem[10000]	16
001	N			
010	**Y**	**10**	**Mem[10010]**	18
011	Y	00	Mem[00011]	3
100	N			
101	N			
110	Y	10	Mem[10110]	22
111	N			

Address Subdivision

此例子中有4GB的記憶體，但是只有4096B的cache，每個block是4B

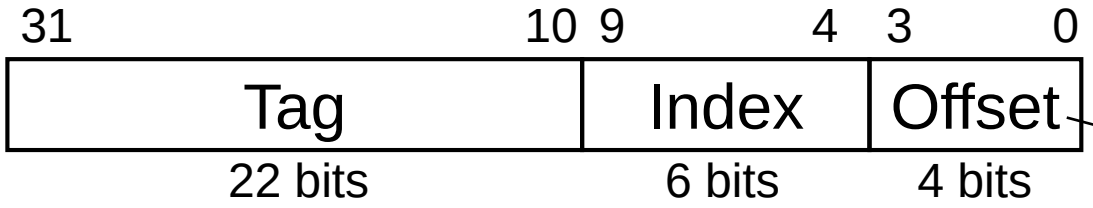


Example: Larger Block Size

- 64 blocks, 16 bytes/block

Byte address

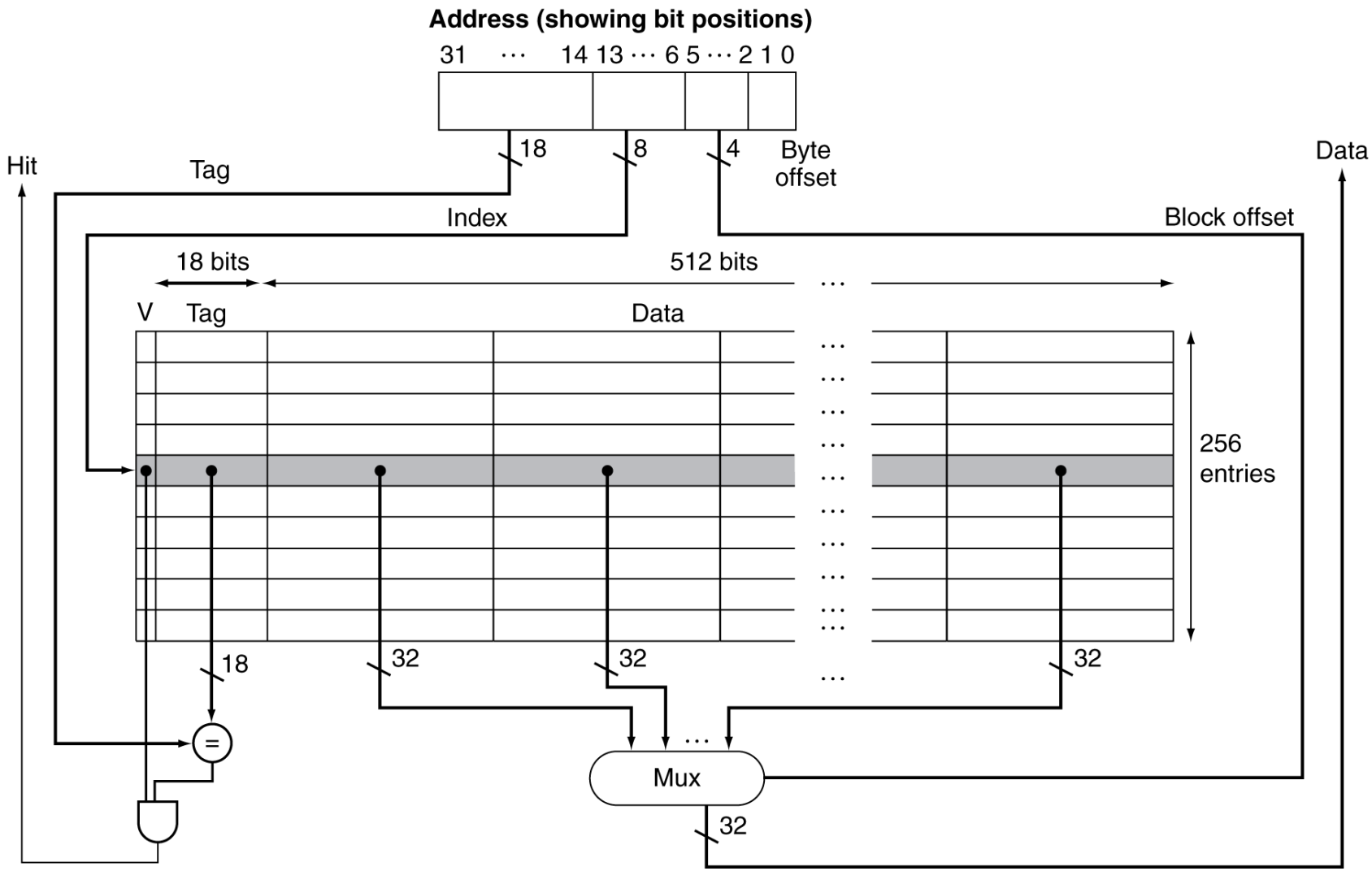
  - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$ 所在的memory block編號
- Block number = 75 modulo 64 = 11



代表一個block中有2^4個byte，該欄位會再細分成block offset與byte offset

Byte address 1200~1215的這16個byte都是在第75個memory block中

Example: Intrinsity FastMATH



Cache Placement Schemes

- **Set associative:**
  - **A set is a group of blocks in the cache**
  - **A block is first mapped onto a set, and then the block can be placed anywhere within that set.**
  - **The set is chosen by the address of the data:**

    (*Block address*) MOD (*Number of sets in cache*)
  - **If there are n blocks in a set, the cache placement is called n-way set associative.**

Associative Caches

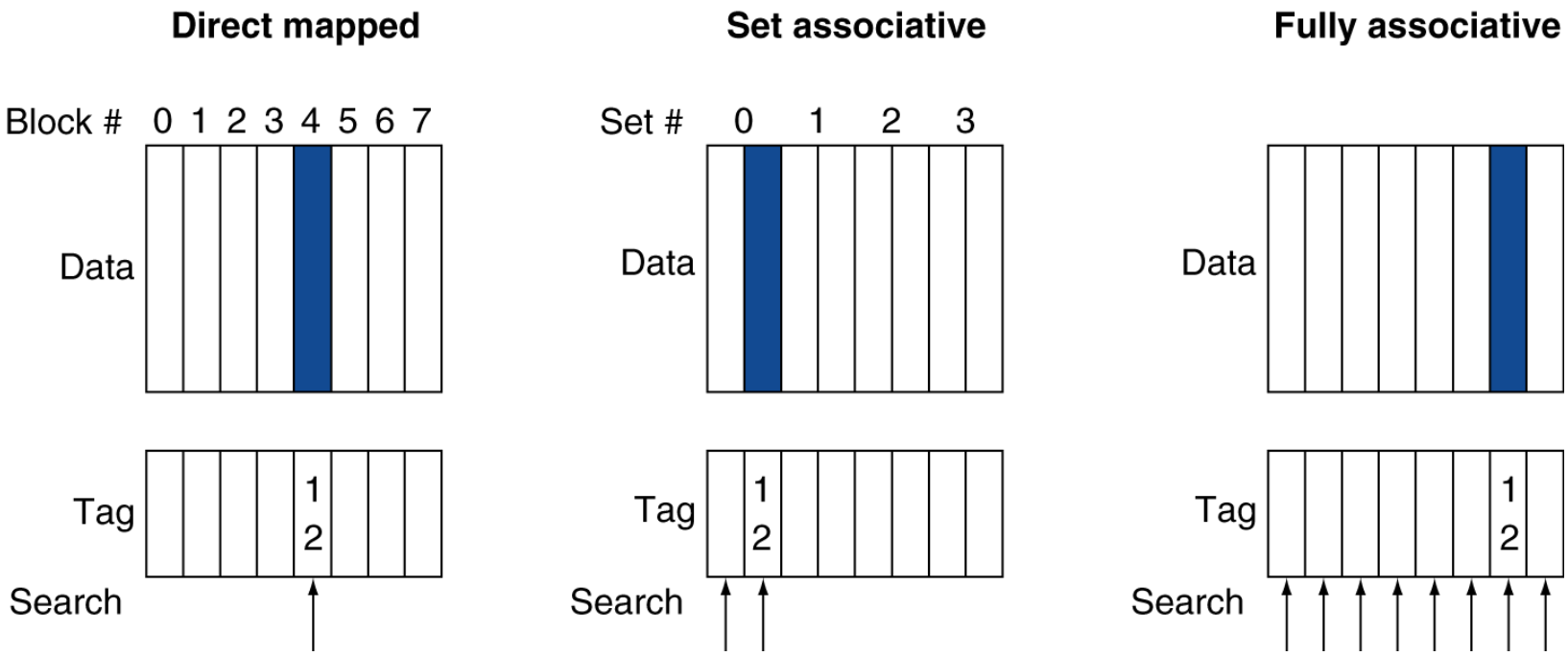
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)

Memory Hierarchy Basics

- *n* sets => *n-way set associative*
  - *Direct-mapped cache* => one block per set
  - *Fully associative* => one set

- Writing to cache: two strategies
  - *Write-through*
    - Update the item in the cache and write through to update main memory
  - *Write-back*
    - Only update the copy in the cache
  - Both strategies use *write buffer* to make writes asynchronous

Associative Cache Example



Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - The very first access to a block cannot be in the cache, so the block must be brought into the cache.
  - Capacity
    - If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur because of blocks being discarded and later retrieved.
  - Conflict
    - If the block placement strategy is not fully associative, conflict misses will occur because a block may be discarded and later retrieved if multiple blocks map to its set and access to the different blocks are intermingled.

Memory Hierarchy Basics

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

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

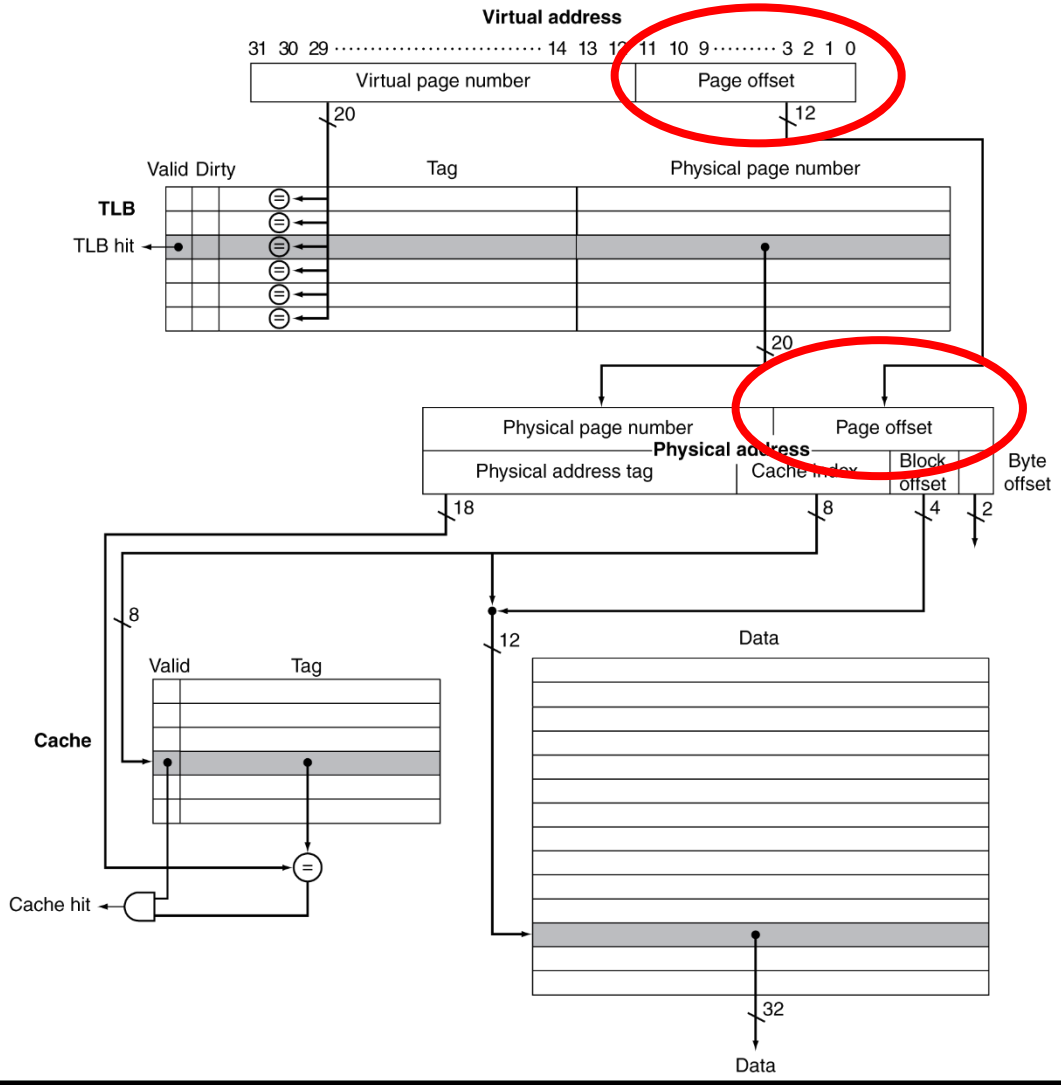
- Note that speculative and multithreaded processors may execute other instructions during a miss
  - Reduces performance impact of misses

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Advantage:
      - Take advantage of **spatial locality.** Reduces **compulsory misses**
    - Disadvantage:
      - Increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Advantage:
      - Reduce **capacity misses**
    - Disadvantage:
      - Increases hit time, increases power consumption
  - Higher associativity
    - Advantage:
      - Reduces conflict misses
    - Disadvantage:
      - Increases hit time, increases power consumption

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Higher number of cache levels
    - Hit time$_{L1}$+Miss rate$_{L1}$*(Hit Time$_{L2}$+Miss rate$_{L2}$*Miss penalty$_{L2}$)
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
    - **This optimization serves reads before writes have been completed.**
    - **Suppose a read miss will replace a dirty memory block. Instead of writing the dirty block to memory, and then reading memory, we could copy the dirty block to a buffer, then read memory, and then write memory.**
  - Avoiding address translation in cache indexing
    - Cache must cope with the translation of a virtual address from the processor to a physical address to access memory
      - A common optimization is to use the page offset
        - It is identical in both virtual and physical addresses.
    - Reduces hit time


Virtual address
31 30 29 ····· 14 13 12 11 10 9 ····· 3 2 1 0
Virtual page number
Page offset
20
12
Valid Dirty
Tag
Physical page number
TLB
TLB hit
20
Physical page number
Page offset
Physical address
Physical address tag
Cache index
Block offset
Byte offset
18
8
4
2
8
12
Data
Valid
Tag
Cache
Cache hit
32
Data

Outline

- Introduction
- Ten Advanced Optimizations of Cache Performance
- Memory Technology and Optimizations
- Protection: Virtual Memory and Virtual Machines
- Crosscutting Issues: The Design of Memory Hierarchies
- Fallacies and Pitfalls

Ten Advanced Optimizations

- Classify ten advanced cache optimizations into five categories based on these metrics
  - Reducing the hit time
    - Small and simple caches
    - Way prediction
  - Increase cache bandwidth
    - Pipelined caches
    - Multibanked caches
    - Nonblocking caches

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

Ten Advanced Optimizations

- Classify ten advanced cache optimizations into five categories based on these metrics
  - Reducing the miss penalty
    - Critical word first
    - Merging write buffers
  - Reducing the miss rate
    - Compiler optimizations
  - Reducing the miss penalty or miss rate via parallelism
    - Hardware prefetching
    - Compiler prefetching

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

Ten Advanced Optimizations

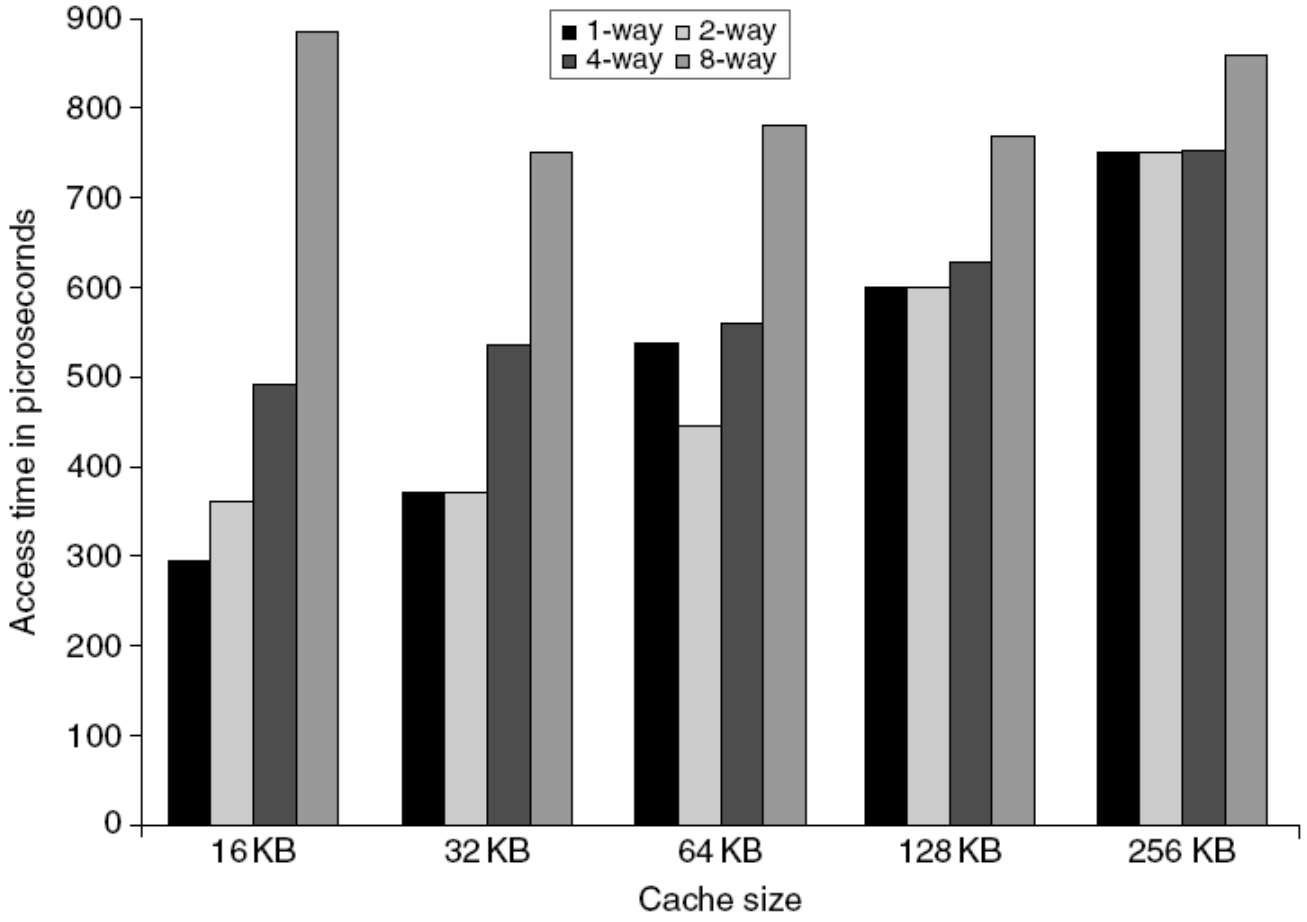
- Small and simple first level caches
  - Critical timing path:
    - addressing tag memory using the index potion of the address, then
    - comparing tags, then
    - Setting the multiplexor to choose the correct data
  - Direct-mapped caches can overlap tag compare and transmission of data
  - Lower associativity reduces power because fewer cache lines are accessed

Reducing the hit time

Small and simple first level caches

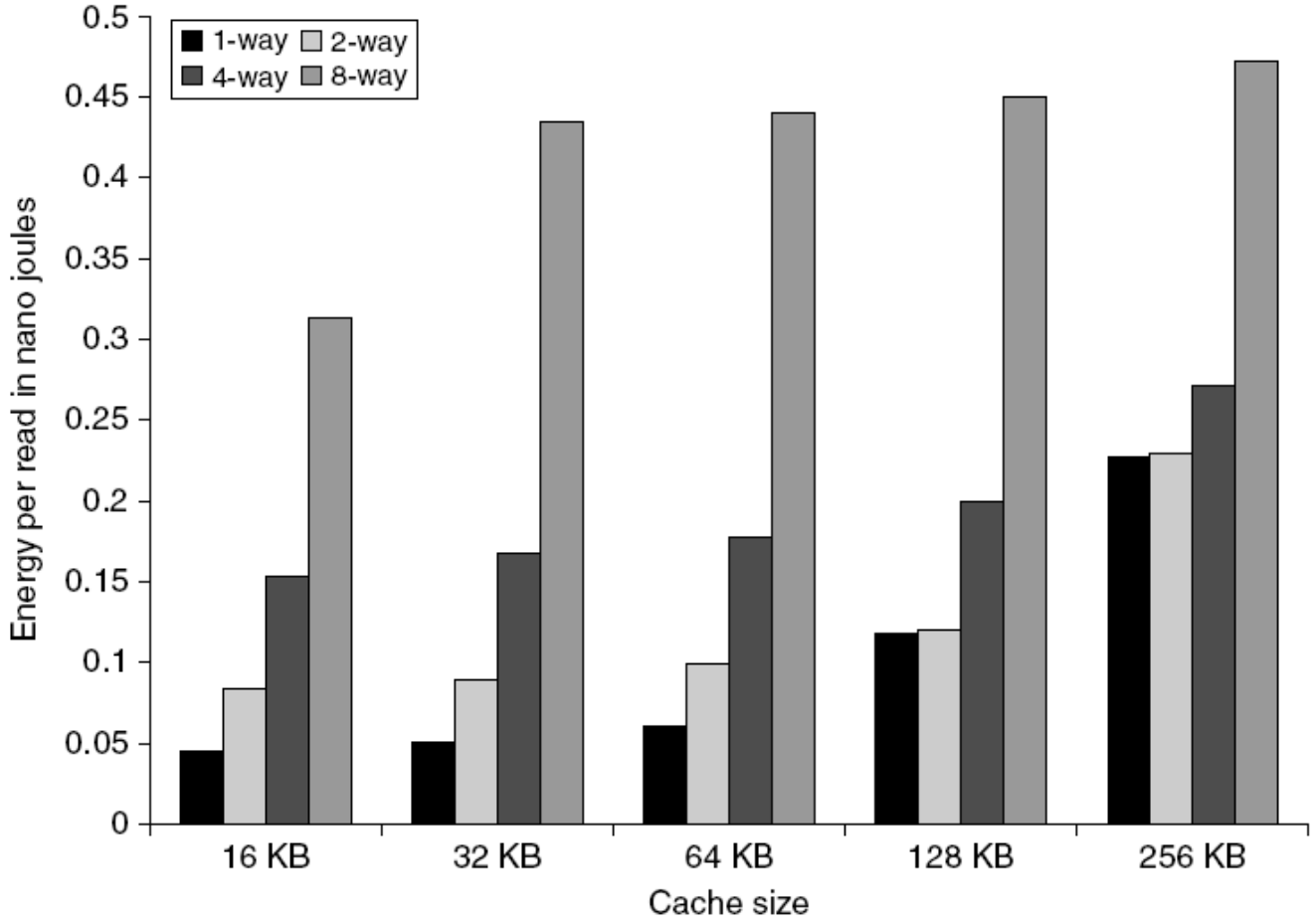
- Although the total amount of on-chip cache has increased dramatically with new generations of microprocessors, due to the clock rate impact arising from a larger L1 cache, the size of the L1 caches has recently increased wither slightly or not at all.
- One approach to determining the impact on hit time and power consumption in advance of building a chip is to use CAD tools.

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

Way Prediction

- 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
  - A miss results in checking the other blocks for matches in the next clock cycle.
- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - “Way selection”
  - Increases mis-prediction penalty

Predict set
Predict block

Reducing the hit time

Pipelining Cache

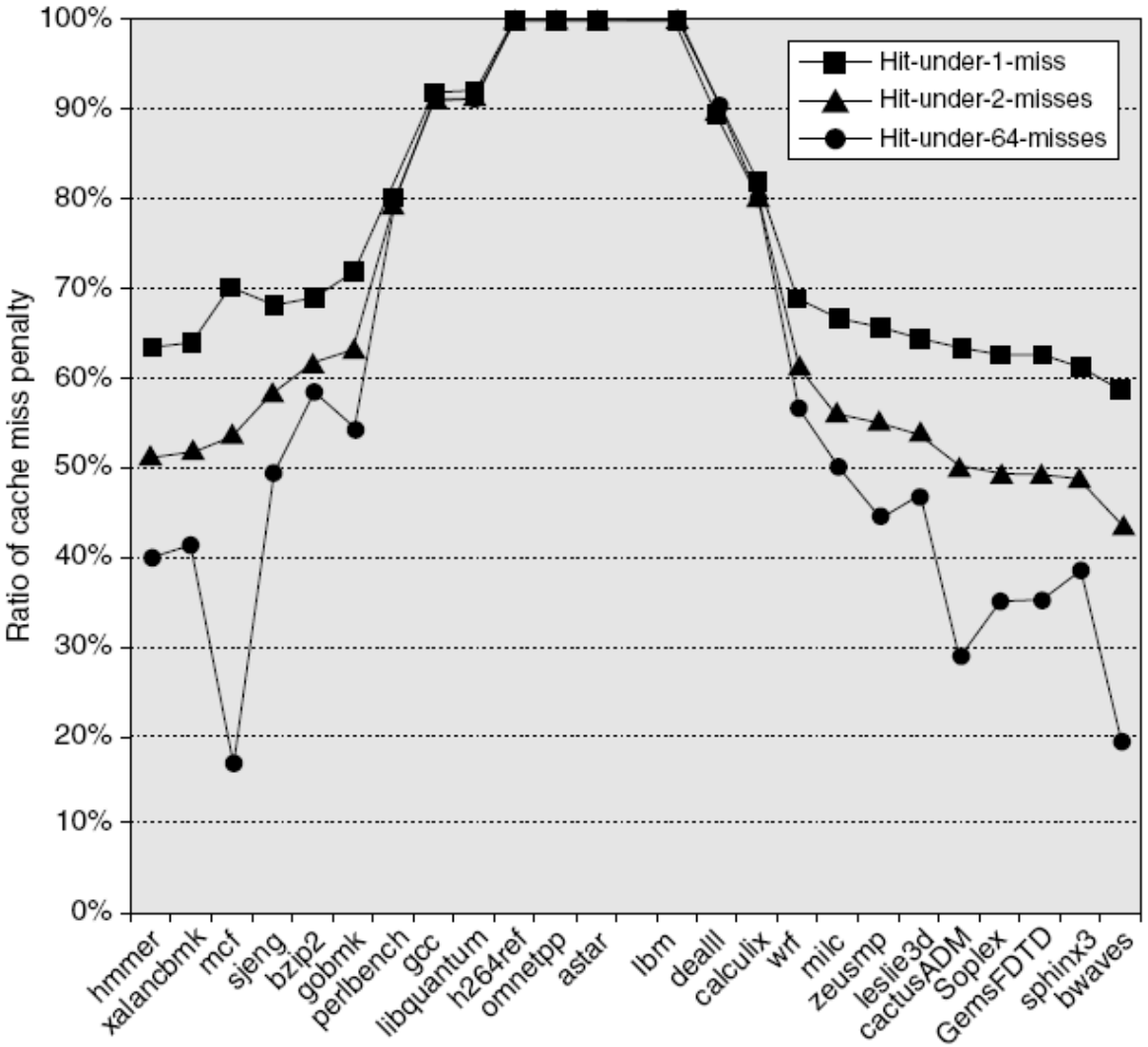
- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Increase cache bandwidth

Nonblocking Caches

- Allow the data cache to continue to supply cache hits during misses
  - "Hit under miss"
    - Reduce the effective miss penalty by being helpful during a miss instead of ignoring the requests of the processor
  - "Hit under multiple miss"
    - The memory system can serve multiple misses
  - Intel Core i7 support both
  - ARM A8 provides only limited nonblocking support in L2
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty

Increase cache bandwidth


Ratio of cache miss penalty
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Hit-under-1-miss
Hit-under-2-misses
Hit-under-64-misses
hmmer
xalancbmk
mcf
sjeng
bzip2
gobmk
perlbench
gcc
libquantum
h264ref
omnetpp
astar
lbm
dealII
calculix
wrf
milc
zeusmp
leslie3d
cactusADM
Soplex
GemsFDTD
sphinx3
bwaves

Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Interleave banks according to block address
  - Bank 0 has all blocks whose address modulo 4 is 0

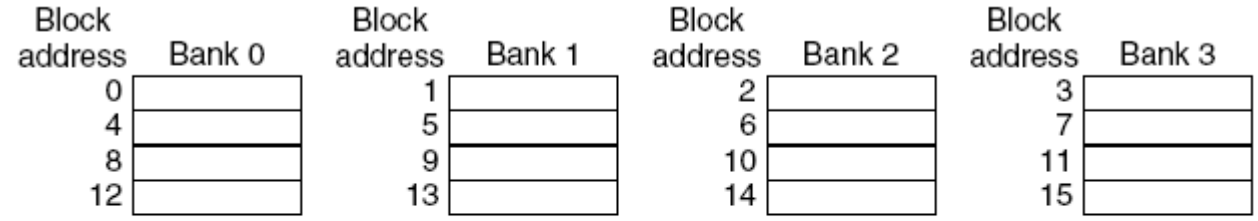


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

Increase cache bandwidth

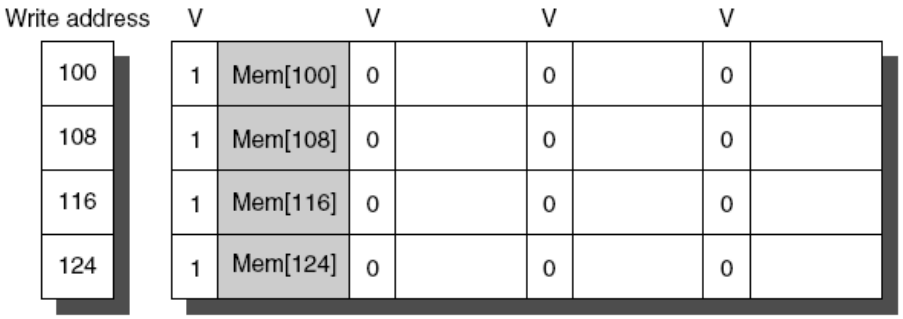
Critical Word First, Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
  - Let the processor continue execution while filling the rest of the words in the block
- Early restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched
  - Only benefit designs with large cache blocks

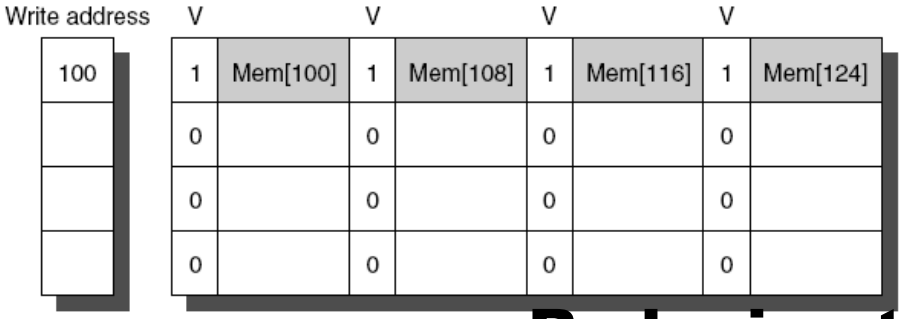
Reducing the miss penalty

Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



No write buffering merging



當某個**write buffer**中已存放未來將更新的資料時，如果又有新的資料要被更新，新的位置將被檢查是否可以和已存資料的合併，減少**write penalty**

Write buffering merging

Reducing the miss penalty

Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order

```
/* Before */
   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 (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; improved spatial locality

跳躍式讀取

Compiler Optimizations

- *Merging Arrays*
  - improve spatial locality by single array of compound elements vs. 2 arrays

```
/* 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

Compiler Optimizations

- *Loop Fusion*
  - Combine 2 independent loops that have same looping and some variables overlap

```
/* 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; improve spatial locality

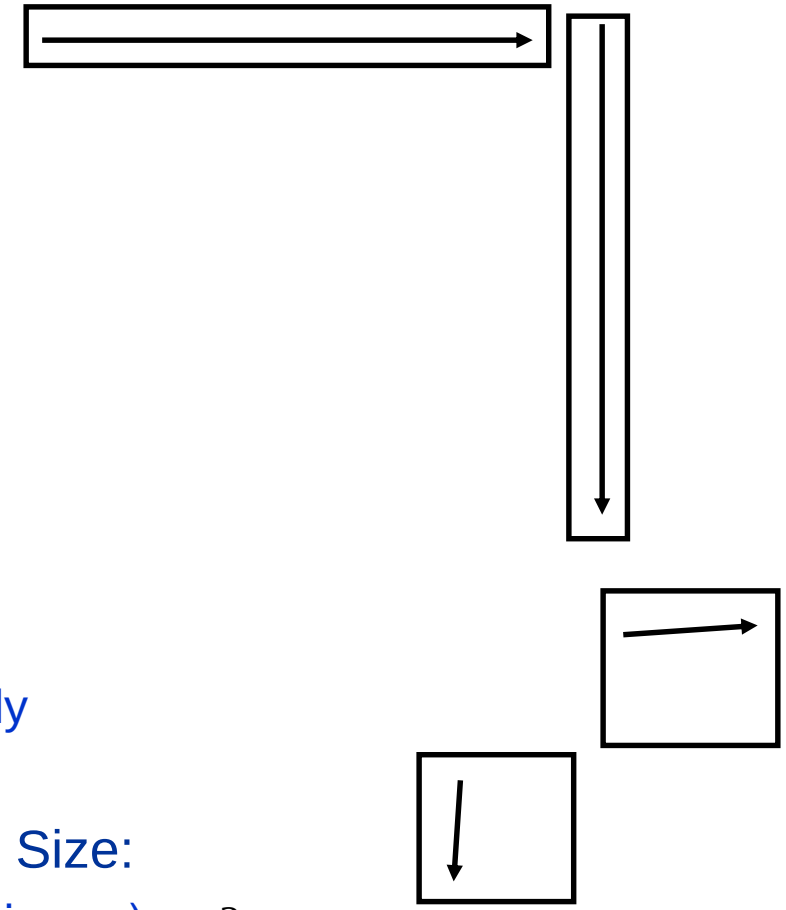
Compiler Optimizations

- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

Blocking Example

X=Y*Z

```
/* 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: 為計算**X**的一個**row**
  - 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

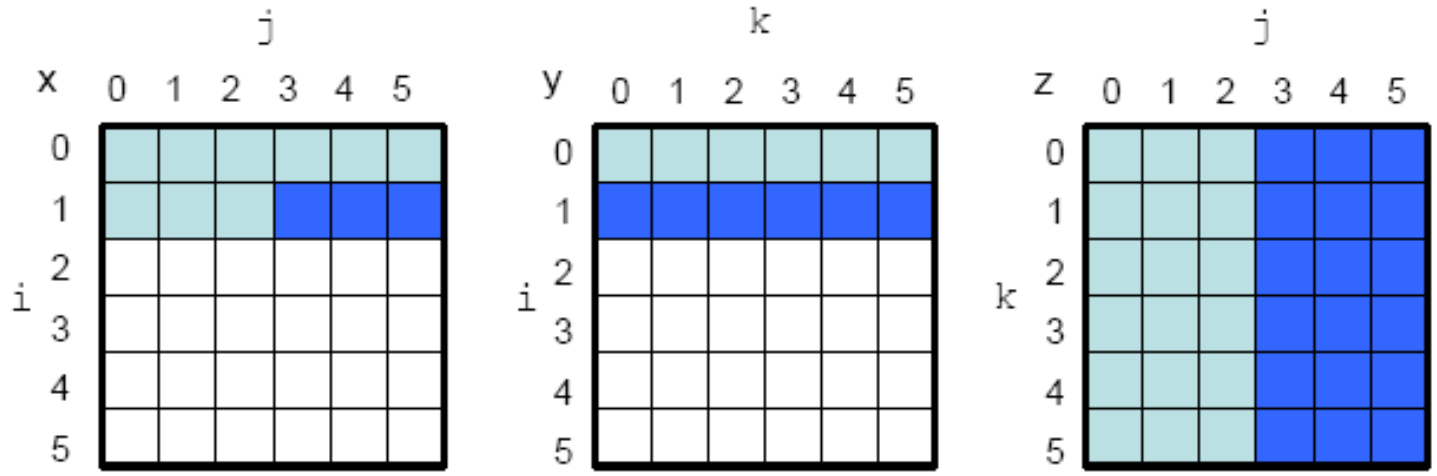
$N^2(2N+1)$

$N^2(N+N+1)$ for each X element

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;
     };
```

Y的row或是Z的column之資料經常性地被讀取後又被取代



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;
     };
```

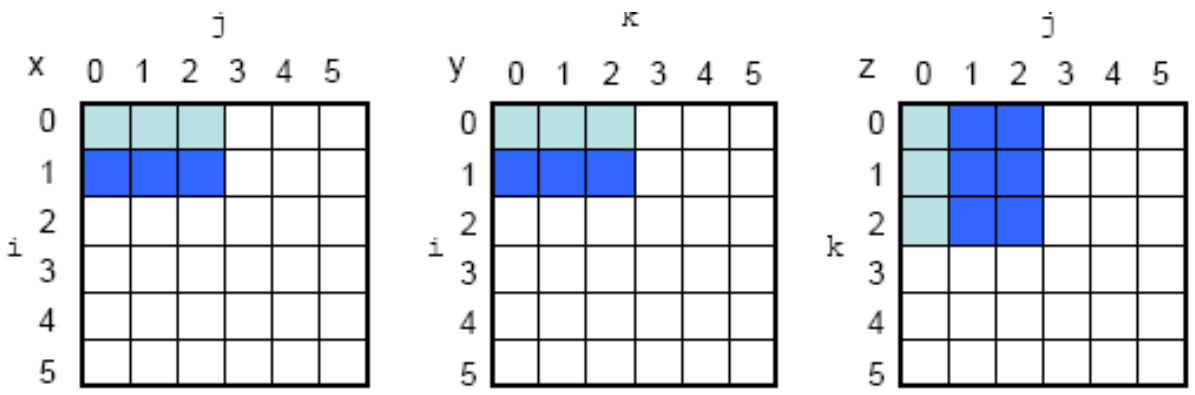
- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- Conflict Misses Too?

$N^2(N/B+N/B+1)$ for each X element

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;
     };
```

Improve **temporal and spatial locality**



Y的row或是Z的column之資料子集合被讀取後將被使用到不再使用為止，相同資料之後將不會再出現**miss**

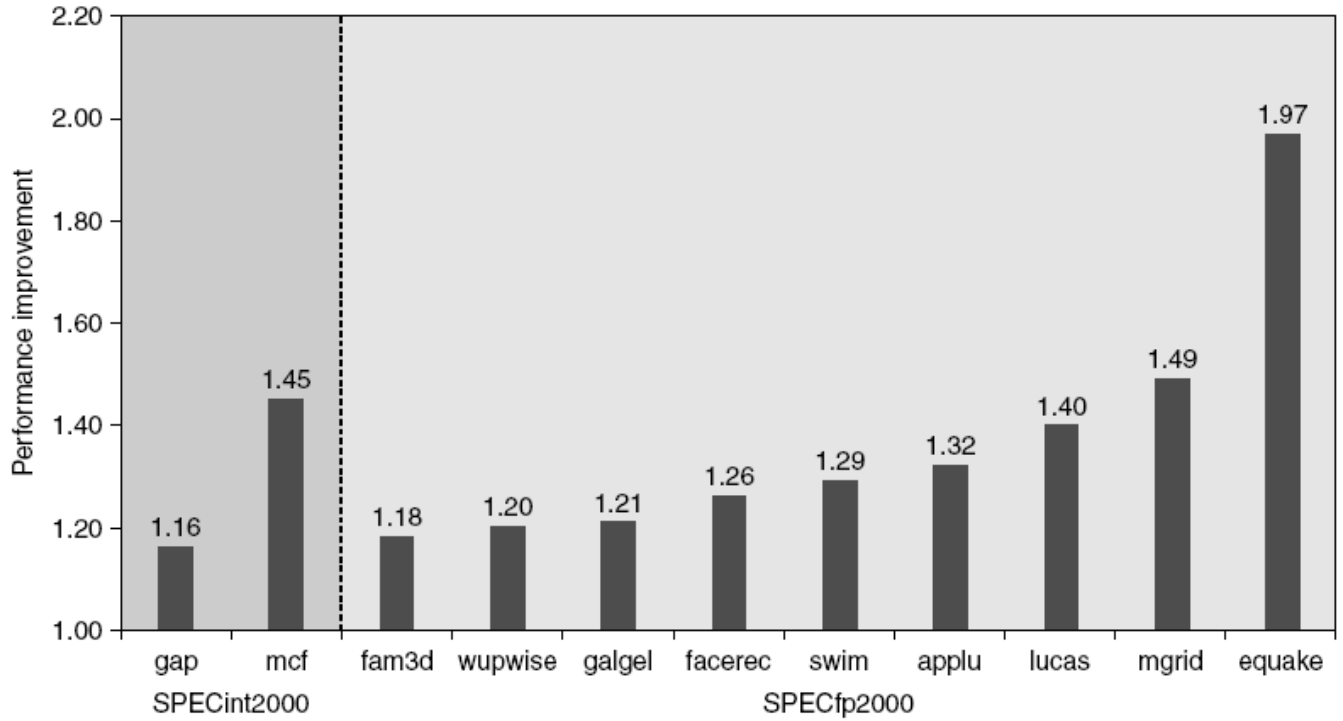
Hardware Prefetching

- Both instructions and data can be prefetched, either directly into the cache or into an external buffer that can be more quickly accessed than main memory.
- Fetch two blocks on miss
  - The required block is placed in the instruction cache
  - The prefetched next sequential block is placed in the instruction stream buffer
    - If the required block is in the instruction stream buffer, the original cache request is canceled, the block is read from the stream buffer, and the next prefetch request is issued.

Hardware Prefetching

- Prefetching relies on utilizing memory bandwidth that otherwise would be unused, but if it interferes with demand misses it can actually lower performance.

Hardware Prefetching



Pentium 4 Pre-fetching

Compiler Prefetching

- Insert prefetch instructions to request data before data is needed
- Two flavors of prefetch
  - Register prefetch
    - Loads data into register
  - Cache prefetch
    - Loads data into cache
- Non-faulting: prefetch doesn't cause exceptions
  - Virtual address faults
  - Protection violations
- If the miss penalty is large, it uses software pipelining or unrolls many times to prefetch data for a future iteration.

Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Outline

- Introduction
- Ten Advanced Optimizations of Cache Performance
- Memory Technology and Optimizations
- Protection: Virtual Memory and Virtual Machines
- Crosscutting Issues: The Design of Memory Hierarchies
- Fallacies and Pitfalls

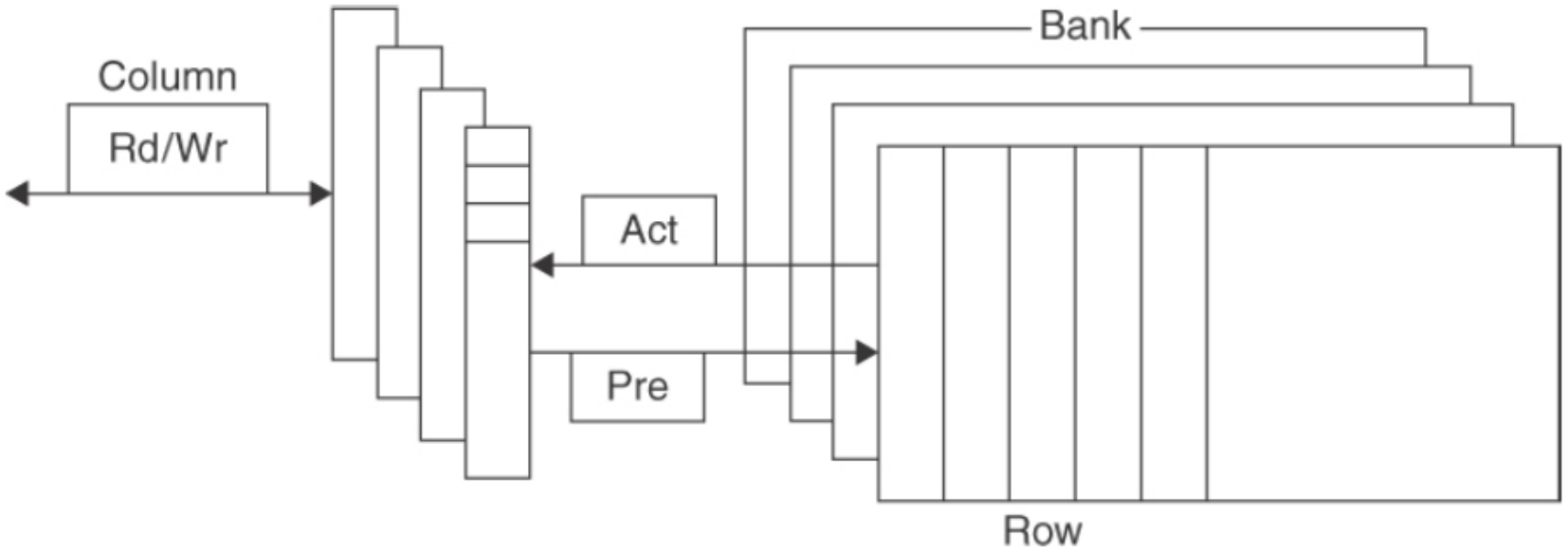
Memory Technology

- Performance metrics
  - Main memory latency is the primary concern of the cache
  - Main memory bandwidth is the primary concern of multiprocessors and I/O
  - With the instruction of burst transfer memory, memory latency is quoted using two measures
    - Access time
      - Time between when a read is requested and when the desired word arrives
    - Cycle time
      - Minimum time between unrelated requests to memory
      - between the start of one memory access to the time when the next access can be started
- DRAM used for main memory, SRAM used for cache

Memory Technology

- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit
- DRAM
  - One transistor/bit
  - Must be re-written after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)
- The memory is organized as a rectangular matrix address by rows and columns.

Internal organization of a DRAM



Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors
- DRAMs must buffer a row of bits inside the DRAM for the column access, and the row is usually the square root of the DRAM size.

Memory Technology

- Some optimizations:
  - Multiple accesses to same row
    - Add timing signals to allow repeated access to the row buffer without another row access time
  - Synchronous DRAM
    - Add clock signals to DRAM interface so that the repeated transfers would not bear that overhead.
    - Burst mode with critical word first
  - Wider interfaces
    - Overcome the problem of getting a wide stream of bits from the memory without having to making the memory too large.
  - Double data rate (DDR)
    - Transfer data on both the rising edge and falling edge of the clock signal
  - Multiple banks on each DRAM device
    - Provide the advantages of interleaving

Memory Optimizations

Production year	Chip size	DRAM Type	Row access strobe (RAS)		Column access strobe (CAS)/ data transfer time (ns)	Cycle time (ns)
			Slowest DRAM (ns)	Fastest DRAM (ns)		
1980	64K bit	DRAM	180	150	75	250
1983	256K bit	DRAM	150	120	50	220
1986	1M bit	DRAM	120	100	25	190
1989	4M bit	DRAM	100	80	20	165
1992	16M bit	DRAM	80	60	15	120
1996	64M bit	SDRAM	70	50	12	110
1998	128M bit	SDRAM	70	50	10	100
2000	256M bit	DDR1	65	45	7	90
2002	512M bit	DDR1	60	40	5	80
2004	1G bit	DDR2	55	35	5	70
2006	2G bit	DDR2	50	30	2.5	60
2010	4G bit	DDR3	36	28	1	37
2012	8G bit	DDR3	30	24	0.5	31

Figure 2.13 Times of fast and slow DRAMs vary with each generation. (Cycle time is defined on page 95.) Performance improvement of row access time is about 5% per year. The improvement by a factor of 2 in column access in 1986 accompanied the switch from NMOS DRAMs to CMOS DRAMs. The introduction of various burst transfer modes in the mid-1990s and SDRAMs in the late 1990s has significantly complicated the calculation of access time for blocks of data; we discuss this later in this section when we talk about SDRAM access time and power. The DDR4 designs are due for introduction in mid- to late 2012. We discuss these various forms of DRAMs in the next few pages.

Memory Optimizations

Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1066–1600	2133–3200	DDR4-3200	17,056–25,600	PC25600

Figure 2.14 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2010. Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM. Although not shown in this figure, DDRs also specify latency in clock cycles as four numbers, which are specified by the DDR standard. For example, DDR3-2000 CL 9 has latencies of 9-9-9-28. What does this mean? With a 1 ns clock (clock cycle is one-half the transfer rate), this indicate 9 ns for row to columns address (RAS time), 9 ns for column access to data (CAS time), and a minimum read time of 28 ns. Closing the row takes 9 ns for precharge but happens only when the reads from that row are finished. In burst mode, transfers occur on every clock on both edges, when the first RAS and CAS times have elapsed. Furthermore, the precharge in not needed until the entire row is read. DDR4 will be produced in 2012 and is expected to reach clock rates of 1600 MHz in 2014, when DDR5 is expected to take over. The exercises explore these details further.

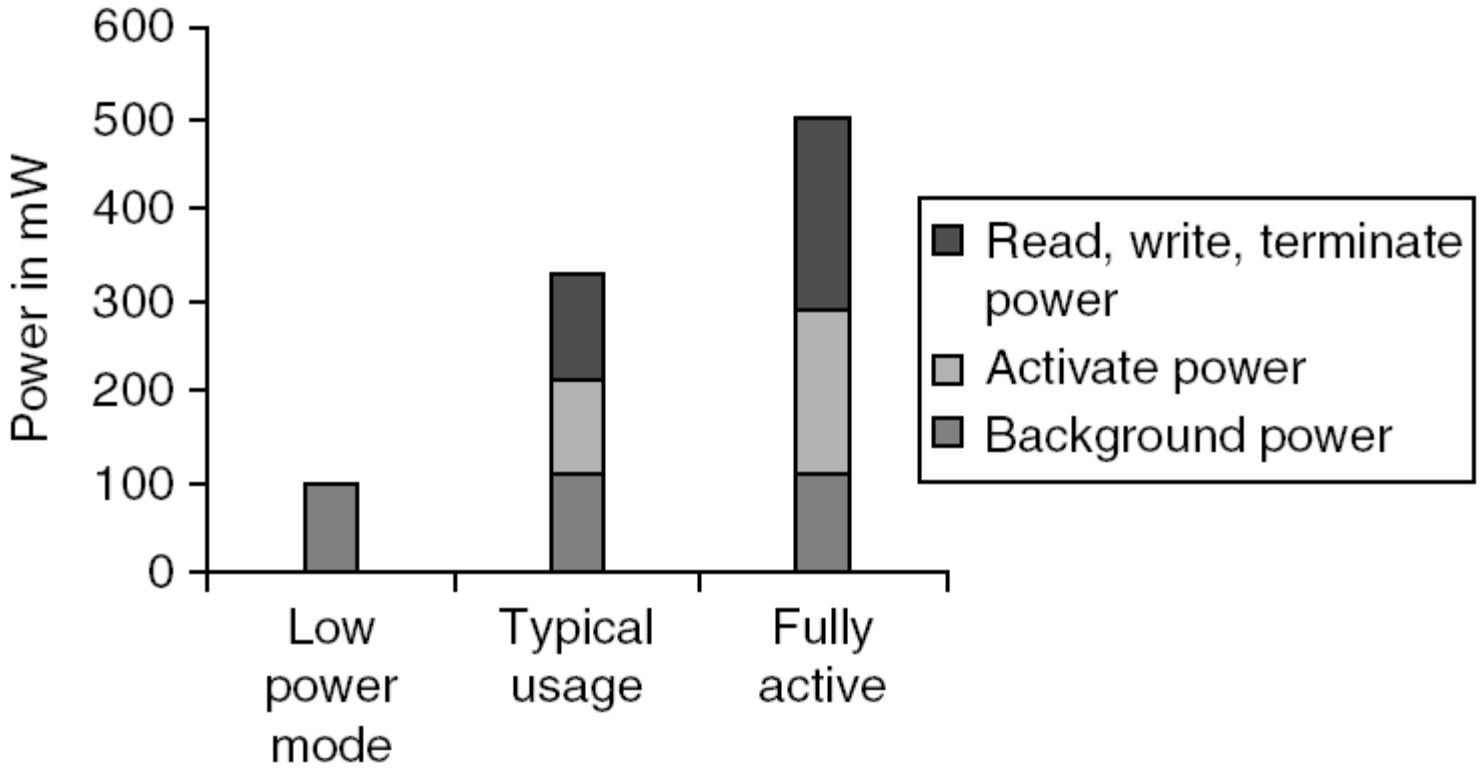
Memory Optimizations

- DDR:
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz
- GDDR5 is graphics memory based on DDR3

Memory Optimizations

- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules
- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

Memory Power Consumption



Flash Memory

- Type of Electrically-Erasable Programmable Read-Only Memory (EEPROM)
- Must be erased (in blocks) before being overwritten
- Non volatile
- Limited number of write cycles
- Cheaper than SDRAM, more expensive than disk
- Slower than SRAM, faster than disk

Memory Dependability

- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic errors
  - Are changes to a cell's contents, not a change in the circuity
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use sparse rows to replace defective rows

- Chipkill(IBM): a RAID-like error recovery technique
  - Distribute the data and ECC information

Virtual Memory

- Protection via virtual memory
  - Keeps processes in their own memory space
- Role of architecture:
  - Provide user mode and supervisor mode
  - Protect certain aspects of CPU state
  - Provide mechanisms for switching between user mode and supervisor mode
  - Provide mechanisms to limit memory accesses
  - Provide TLB to translate addresses

Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

- Allows different ISAs and operating systems to be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Impact of VMs on Virtual Memory

- Each guest OS maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called "real memory"
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - Requires VMM to detect guest's changes to its own page table
    - Occurs naturally if accessing the page table pointer is a privileged operation
- The basic requirements of system Virtual Machines are almost identical to those for Virtual Memory.
  - At least two processor modes: user and system
  - A privileged subset of instructions that is available only in system mode, resulting in a trap if executed in user mode.

Fallacies and Pitfalls

- Fallacies
  - Predicting cache performance of one program from another