Computer Architectures

Chapter 5

Tien-Fu Chen

National Chung Cheng Univ.

Topics in Memory Hierachy

- **Memory Hierachy**

  Features: temporal & spatial locality

  Common: Faster -> more expensive -> smaller

- **Cache**
  - organization
  - fetch algorithm
  - replacement
  - write policy
  - virtual addr space
  - multi-level cache
  - unified / split cache
  - multiprocessor caches
- **Virtual memory**
  - paged memory paged segment
  - segment memory segmented page
  - TLB
  - protection
- **Main memory**
  - interleaved memory
  - access mode
- **Multiprocessor cache**
  - coherence protocols
  - consistency models

Four Questions about Caches

Q1: Where can a block be placed in the upper level? (Block placement)

- Fully associative, direct mapped, 2-way set associative
- index = (block num) % (# of set)

Q2: How is a block found if it is in the upper level? (Block identification)

- address = tag + index + block offset

Q3: Which block should be replaced on a miss? (Block replacement)

- Random (large associativities)
- LRU (smaller associativities)

Q4: What happens on a write? (Write strategy)

- Write through: written to both cache and mem
- Write back: written only to the cache. The modified cache block is written to main memory only when it is replaced.

Terms in Memory Hierarchy

- **Block (page, line)**

  Units transferred between , fixed length

- **Hit/Miss**

- **Miss Ratio**

$$miss\ ratio(m) = \frac{\#\ of\ \ hits}{\#\ \ of\ \ misses}$$

- **Hit time**

  time to access upper level

- **Miss penalty**

  access time of lower level
  + replace time + transfer time

- **average (effective) access time**

$$t_{avg} = t_{hit} + miss\ ratio \times t_{miss}$$

$$CPI_{w/cache} = CPI_{ideal} + \frac{mem\ \ ref}{instruns} \times miss\ ratio \times C_{miss}$$

Decisions on Caches

- **Organization (Placement)**
  - Direct-mapped
  - Fully-associative
  - Set-associative
  - sector caches
  - multi-level caches
- **Fetch Algorithm**
  - fetch on miss
  - non-blocking
  - fetch by passing
  - prefetch
- **Replacement Algorithm**
  - LRU, random, FIFO
- **Write policy**
  - Write-back
  - Write-through
- **Cache Parameters**
  - Cache size (C)
  - Block (line) size (b)
  - set size (s)
  - Associativity (a)

$$C = (\#\ of\ \ set\ s) \times a \times b$$

Strategies for Line Replacement

- **Fetching a line**
  - access and fill - starts on line boundary
  - **fetch bypass** (wrap-around load)
    access faulted word first and load the remainder line on following
  - non-blocking cache
    processor continues execution when exection not depend on
  - prefetching
- **Line Replacements**
  - LRU: Least-recently used
    optimize based on temporal locality
    a counter associated with each line
    modify counter on each read/write
  - FIFO:
    longest line is to be replaced
  - Random:
    select a vitctim at random
- **Victim Caches**

  a fully-assoicative buffer holding replaced lines

Write Policies

- **write back (copy back)**

  cached block is updated to memory only when replaced
  - dirty bits used to avoid update clean blocks
  - Less traffic for larger caches

  $$Traffic / inst = fract\_dirty \times miss \times Block$$

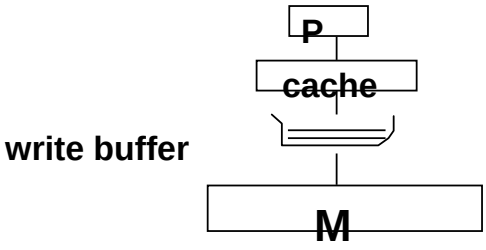
- **write through**

  block written both to cache and memory
  adv: retain a consistent copy between cache and mem
  disadv: high memory bandwidth required

  $$Traffic / inst = fract\_writes$$

- **on a write miss**
  - write allocate
  - No write allocate

- **Write Buffers**
  allow reads to proceed without stalling



Other Types of Caches

- **Split Data / Instruction caches**

Separate	Unified
1.bandwidth increased	1.less costly
2.take spactial locality form instr. cache	2. overall miss ratio increase
3. no interlock on simultaneous requests	3.self-modifying code and execute are possible

- **Cache Performance**

  CPU time = IC x (CPI execution + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time

  Misses per instruction = Memory accesses per instruction x Miss rate

  CPU time = IC x (CPI execution + Misses per instruction x Miss penalty) x Clock cycle time

Improving Cache Performance

- **Cache Size**

$$cache\ size = \#of\ set \times set\ associativity \times block\ size$$

- **Effects of parameters**
  - Cache size
    larger cache exploit better temporal locality
    large access time
  - Block size
    larger block size has better spatial locality
    cause unused data transferred & replacement
  - Associativity
    larger associaivity get lower miss ratio
    high cost
- **Average memory-access time =**

  Hit time + Miss rate x Miss penalty (ns or clocks)
- **Improve performance by:**
  1. Reduce the miss rate,
  2. Reduce the miss penalty, or
  3. Reduce the time to hit in the cache.

Sources of cache miss

3C/4C

- **Compulsory :**
  first access to a block
  cold-start miss, first reference miss
- **Capacity :**
  due to blocks being discarded, limited by size
- **Conflict :**
  due to conflicts in set-associative caches
- **Coherence :**
  Misses because blocks are invalidated due to references by other processors

Miss rate reduction

- I-1: larger block size
  - take advantage of spatial locality
  - will increase miss penalty
- I-2: higher associativity => increased hit time
- I-3: victim cache
  - A small fully associative cache
  - contains blocks that are replaced on a miss
  - useful for small and direct-mapped caches

Miss Rate Reduction (Cont)

- **I-4: Pseudo-associative caches**
  - proceed like direct-mapped
  - on a miss, check another entry in set
  - fast hit time and slow hit time
  - can reduce miss rate

- **I-5: Prefetching**
  - Prefetch instructions
  - hardware based stream buffer
  - compiler-controlled prefetch
    - Register prefetch
    - Cache prefetch
  - Issues
    - faulting vs. non faulting (nonbinding)
    - blocking vs nonblocking (lockup-free)
- **Prefetching relies on extra memory bandwidth that can be used without penalty**

More miss rate reduction (prefetching)

- **Hardware prefetching of instructions**
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in stream buffer
  - On miss check stream buffer
  - Works with data blocks too
- **Data prefetching**
  - Preload 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
  - Issuing Prefetch Instructions takes time

  Is cost of prefetch issues < savings in reduced misses?
- **Prefetching policy**
  - OBL (one block lookahead)
  - always prefetch
  - prefetch on miss
  - tagged prefetch
  - stride prefetching

More miss rate reduction (compiler optimizations)

- **Instructions**
  - Reorder procedures in memory so as to reduce misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks
- **I-7: Compiler optimization on 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 locks?of data repeatedly vs. going down whole columns or rows

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
   for (j = 0; j < 100; j = j+1)
      for (i = 0; i < 5000; i = i+1)
            x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
   for (i = 0; i < 5000; i = i+1)
      for (j = 0; j < 100; j = j+1)
            x[i][j] = 2 * x[i][j];
```

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

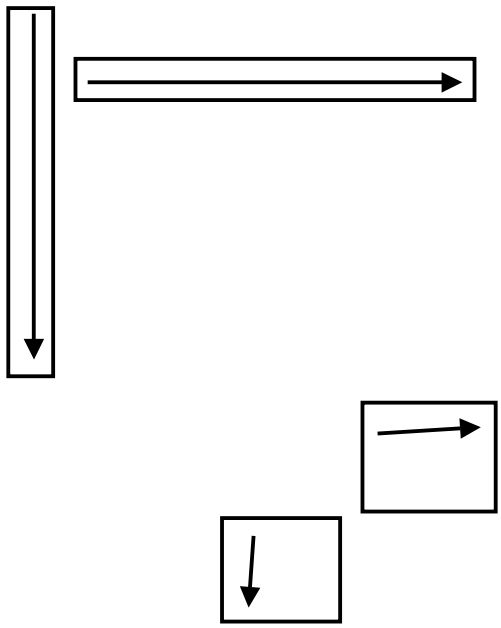
Loop Fusion Example

```
/* 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; improve 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**

Blocking Example

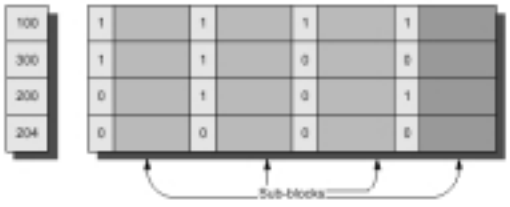
```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
     {r = 0;
      for (k = kk; k < min(kk+B-1,N); k = k+1) {
        r = r + y[i][k]*z[k][j];};
      x[i][j] = x[i][j] + r;
     };
```

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

Reducing Cache Miss Penalty

- **II-1: Giving priority to read misses over writes**
  - cost of writes reduced by a write buffer
- **II-2: Sub-block Placement (**Do not have to load full block on a miss)

  access unit- subblock
  tag unit - a full line
  +reduce tag overhead
  +wrapped fetch



- **II-3: Do not wait for full block to be loaded before restarting CPU**
  - Early restart
  - critical word first - wrapped fetch/ requested word first
  - benefit from large block size
- **II-4: Nonblocking caches (lockup-free caches)**
  - allow cache to continue to supply data during a miss
  - may cause out-of-order accesses

Multi-level Caches

- **Why multi-level caches?**

  Need larger caches to reduce frequency of most costly misses => reduce the cost of misses with different level of caches

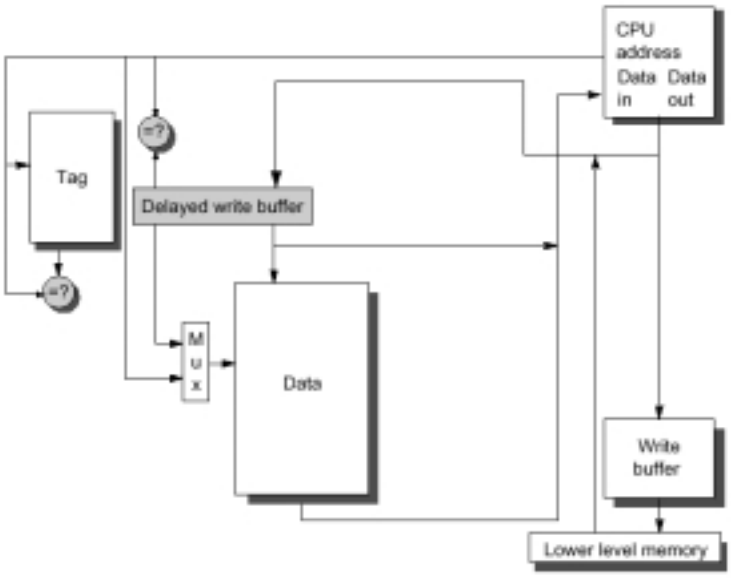
- **II-5: Two-level Caches**
  - motivation
    - reduce average access time by
    - large **cache size** **block size**
    - **cache associativity**
    - hierarchical sharing
    - shield processors from coherence traffic
  - Multi level Inclusion Property
    L2 always contains superset of data in L1

$$A_{i+1} \geq \sum A_i \times \max\left(\frac{B_{i+1}}{B_i}, \frac{S_i}{S_{i+1}}\right)$$

16 x C1 : 16 K, direct map (A = 1) B1 = 16
C2 : 256K, if B2=B1 => A2=16
if B2=4B1 => A2=64

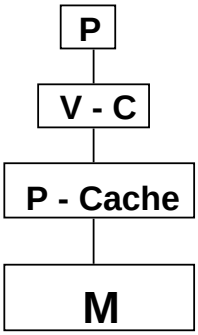
Reducing hit time

- **III-1: Fast Hit times via Small and Simple Caches**
  - Direct Mapped, on chip
- **III-2: Virtual address Cache**
  - avoid address translation during indexing the cache
- **III-3: Pipelined Writes**
  - Pipeline Tag Check and Update Cache as separate stages; current write tag check & previous write cache update
  - Only Write in the pipeline; empty during a miss

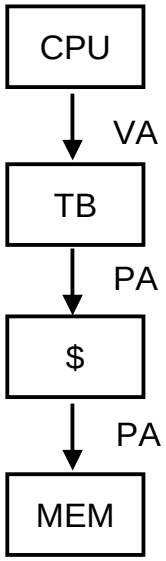


Virtual-Addressed Cache

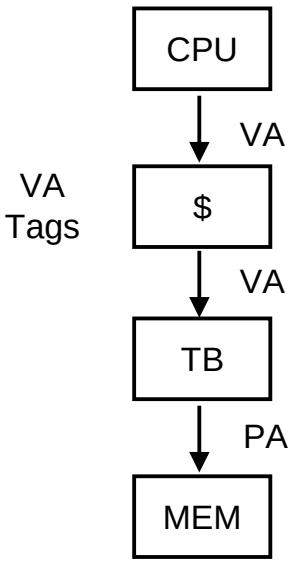
- **Motivation : reduce cache hit time**
- **Approach :**
  - **a**ddress translate partially in ***parallelism***
    => limited by page page
  - virtual caches
- **Problems**
  - address translation require if miss
  - synonym (alias)
    sol :
    - » Inverted Page Table (IBM 801)
    - » reverse translation table
      2 - level cache :
    - » large addr space
      (64 bit)
  - context switching
    - » add PID to tag
    - » flush
  - I/O device
  - cache coherence in MP



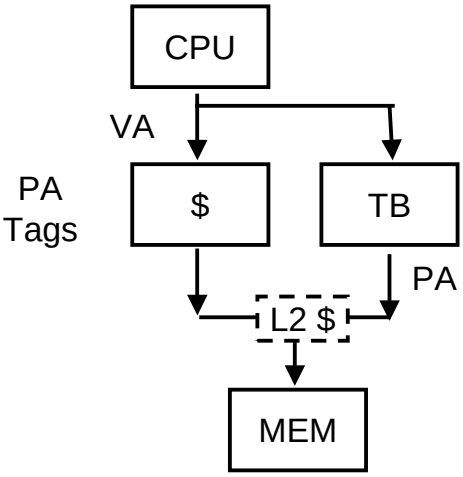
Fast hits by Avoiding Address Translation



Conventional Organization



Virtually Addressed Cache
Translate only on miss
Synonym Problem



Overlap $ access with VA translation: requires $ index to remain invariant across translation

Cache Optimization Summary

Technique	MR	MP	HW	Cmplty
Larger Block Size	+	-		0
Higher Associativity	+	-		1
Victim Caches	+			2
Pseudo-Associative Caches	+			2
HW Prefetching of Instr/Data	+			2
Compiler Controlled Prefetching	+			3
Compiler Reduce Misses	+			0
Priority to Read Misses		+		1
Subblock Placement	-	+		1
Early Restart & Critical Word 1st	+			2
Non-Blocking Caches		+		3
Second Level Caches		+		2
Small & Simple Caches ?	-	+		0
Avoiding Address Translation		+		2
Pipelining Writes		+		1

Using Caches VS Large Register File on chip

Large Register File

Pro :

- All local scalars variables
- Compiler assigns global variables
- Register addressing
- May parallel access
- Small on chip and fast

Against :

- Significant overhead in context switching
- Limit set on number of local variables
- Only small fraction is actively used
- Rely on compiler to max usage

Caches

Pro :

- Recently used local scalar
- Blocks of memory
- Recently used global
- Memory addressing (associative access)

Against :

- Non-uniform access latency
- Slow access time
- More hardware complexity
- Replacement overhead

Main Memory

- Performance of Main Memory:
  - Latency: Cache Miss Penalty
    - Access Time: time between request and word arrives
    - Cycle Time: time between requests
  - Bandwidth: I/O & Large Block Miss Penalty
- **Main Memory is DRAM**

  (Dynamic Random Access Memory)
  - Dynamic since needs to be refreshed periodically (8 ms)
  - Addresses divided into 2 halves (as a 2D matrix)
    - RAS or Row Access Strobe
    - CAS or Column Access Strobe
- **Cache uses SRAM:**

  (Static Random Access Memory)
  - No refresh (6 transistors/bit vs. 1 transistor/bit)
  - Address not divided

Improving main memory bandwidth

(a) one-word-wide

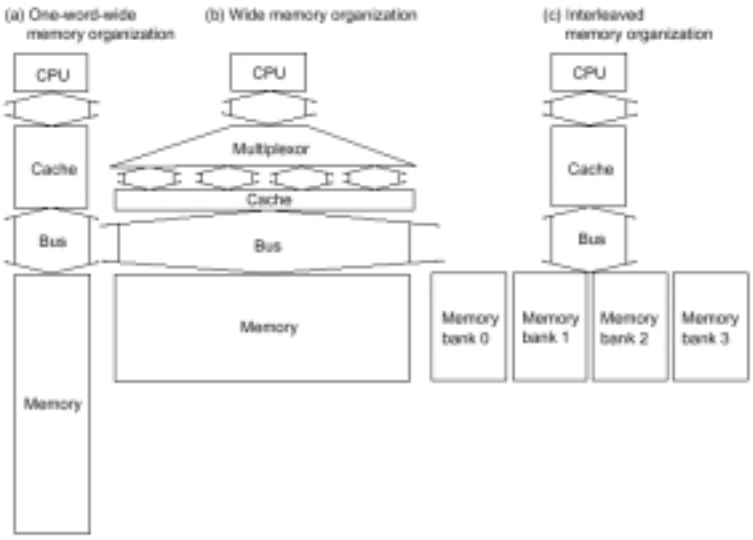
- CPU, Cache, Bus, Memory same width

(b) wide memory

- Mux/ Cache, Bus, Memory N words

(c) interleaved memory

- CPU, Cache, Bus 1 word: Memory N Modules (4 Modules)



Increasing memory bandwidth(more)

- **Independent Memory Banks**
  - allow multiple independent accesses
  - multiple memory controllers allow bank to operate independently
- **Avoiding Bank Conflicts**
  - problem

```
int x[256][512];
    for (j = 0; j < 512; j = j+1)
        for (i = 0; i < 256; i = i+1)
            x[i][j] = 2 * x[i][j];
```

  - Even with 128 banks, since 512 is multiple of 128 conflict
  - solution:

SW: loop interchange or declaring array not power of 2

HW: Prime number of banks

DRAM-specific Interleaving

- **Multiple RAS accesses:**
  - Nibble mode - supply three extra bits
  - page mode - change column address until next access or refresh time
  - static column - do not toggle the column access strobe on each column address change
- **New DRAMs to address gap**
  - Synchronous DRAM:
    Provide a clock signal to DRAM, transfer synchronous to system clock
  - RAMBUS:
    Each Chip a module vs. slice of memory ?Short bus between CPU and chips
    Does own refresh

Variable amount of data returned
1 byte / 2 ns (500 MB/s per chip)