CIS 429/529

Module II & III Review

Prof. Michel A. Kinsy
The course has 4 modules

**Module 1**
- Instruction Set Architecture (ISA)
- Simple Pipelining and Hazards

**Module 2**
- Superscalar Architectures
- Superscalar Processors
- Vector machines
- VLIW
- Multithreading
- GPU

**Module 3**
- Branch Prediction
- Caches
- Memory Models & Synchronization
- Cache Coherence Protocols

**Module 4**
- On-Chip networks
- On-chip Network routing
Reducing Control Flow Penalty

• Software solutions
  ‣ Eliminate branches - loop unrolling increases the run length
  ‣ Reduce resolution time - instruction scheduling compute the branch condition as early as possible (of limited value)

• Hardware solutions
  ‣ Find something else to do - delay slots replaces pipeline bubbles with useful work (requires software cooperation)
  ‣ Speculate - branch prediction speculative execution of instructions beyond the branch
Branch Prediction

• Motivation:
  ‣ Branch penalties limit performance of deeply pipelined processors
  ‣ Modern branch predictors have high accuracy (>95%) and can reduce branch penalties significantly
Branch Prediction

• Required hardware support:
  ‣ Prediction structures: branch history tables, branch target buffers, etc.
  ‣ Mispredict recovery mechanisms:
    • Keep result computation separate from commit
    • Kill instructions following branch in pipeline
    • Restore state to state following branch
Multilevel Memory

• Strategy: Reduce average latency using small, fast memories called caches.

  ▶ Caches are a mechanism to reduce memory latency based on the empirical observation that the patterns of memory references made by a processor are often highly predictable:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>loop</td>
<td>ADD r2, r1, r1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>SUBI r3, r3, #1</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>BNEZ r3, loop</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td></td>
<td>112</td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td>96</td>
</tr>
</tbody>
</table>
Common Predictable Patterns

• Two predictable properties of memory references:

  ‣ Temporal Locality: If a location is referenced it is likely to be referenced again in the near future.

  ‣ Spatial Locality: If a location is referenced it is likely that locations near it will be referenced in the near future.
Management of Memory Hierarchy

• Small/fast storage, e.g., registers
  ▸ Address usually specified in instruction
  ▸ Generally implemented directly as a register file
    • but hardware might do things behind software’s back, e.g., stack management, register renaming

• Large/slower storage, e.g., memory
  ▸ Address usually computed from values in register
  ▸ Generally implemented as a cache hierarchy
    • hardware decides what is kept in fast memory
    • but software may provide “hints”, e.g., don’t cache or prefetch
Inside a Cache

- Processor
- CACHE
- Main Memory

Address Tag

Copy of main memory location 100
Copy of main memory location 101

Data Block

Table:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data Byte</th>
<th>Data Byte</th>
<th>Data Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>Data Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6848</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>416</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Line
Placement Policy

Memory

Block Number

Set Number

Cache

Memory Block Number

Fully Associative anywhere
(2-way) Set Associative anywhere in set 0
Direct Mapped only into block 4

block 12 can be placed

(12 mod 4)

(12 mod 8)
Direct-Mapped Cache

- **Tag**: t
- **Index**: k
- **Block Offset**: b

**Cache Structure**
- V: Valid Bit
- Tag: TOF
- Data Block: Data Word or Byte

**Details**
- \(2^k\) lines
- HIT

Diagram shows the mapping of the cache with tags and valid bits.
2-Way Set-Associative Cache

- Tag
- Index
- Block Offset
- Data Block
- V
- Data
- Tag
- Offset
- Block
- Offset
- Data Word or Byte
- HIT
Fully Associative Cache

Block Offset

Tag

V Tag Data Block

HIT

Data Word or Byte

t = t = t =
Improving Cache Performance

• Average memory access time =

  \[ \text{Hit time} + \text{Miss rate} \times \text{Miss penalty} \]

• To improve performance:
  ‣ reduce the hit time
  ‣ reduce the miss rate (e.g., larger cache)
  ‣ reduce the miss penalty (e.g., L2 cache)

• What is the simplest design strategy?
  ‣ Biggest cache that doesn’t increase hit time past 1-2 cycles
    (approx 8-32KB in modern technology)
Causes for Cache Misses

• Compulsory:
  ‣ first-reference to a block a.k.a. cold start misses
  ‣ misses that would occur even with infinite cache

• Conflict:
  ‣ misses from collisions due to block-placement strategy
  ‣ misses due to the cache being too small to hold all data the program needs
Effect of Cache on Performance

- Larger cache size
  - reduces conflict misses
  - hit time will increase

- Higher associativity
  - reduces conflict misses
  - may increase hit time

- Larger block size
  - reduces compulsory misses
  - exploit burst transfers in memory and on buses
  - increases miss penalty and conflict misses
Replacement Policy

• Which block from a set should be evicted?
  ‣ Random
  ‣ Least Recently Used (LRU)
    • LRU cache state must be updated on every access
    • true implementation only feasible for small sets (2-way)
    • pseudo-LRU binary tree often used for 4-8 way
  ‣ First In, First Out (FIFO) a.k.a. Round-Robin
    • used in highly associative caches
  ‣ Not Least Recently Used (NLRU)
    • FIFO with exception for most recently used block or blocks
Multilevel Caches

- A memory cannot be large and fast
- Increasing sizes of cache at each level

Local miss rate = misses in cache / accesses to cache
Global miss rate = misses in cache / CPU memory accesses
Misses per instruction = misses in cache / number of instructions
Write Policy Choices

• Cache hit:
  ▶ **Write through**: write both cache & memory
    • generally higher traffic but simplifies cache coherence
  ▶ **Write back**: write cache only (memory is written only when the entry is evicted)
    • a dirty bit per block can further reduce the traffic
Write Policy Choices

• Cache miss:
  ‣ **no write allocate**: only write to main memory
  ‣ **write allocate** (aka fetch on write): fetch into cache

• Common combinations:
  ‣ write through and no write allocate
  ‣ write back with write allocate
Names for Memory Locations

- **Machine language address**
  - as specified in machine code

- **Virtual address**
  - ISA specifies translation of machine code address into virtual address of program variable

- **Physical address**
  - operating system specifies mapping of virtual address into name for a physical memory location
Physical or Virtual Address Caches?

- one-step process in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- aliasing problems due to the sharing of pages (-)

Alternative: place the cache before the TLB

(StrongARM)
Address Translation

Virtual Address

TLB Lookup

<table>
<thead>
<tr>
<th>miss</th>
<th>hit</th>
</tr>
</thead>
</table>

Page Table Walk

the page is

not in memory

Page Fault (OS loads page)

in memory

Update TLB

Protection Check

denied

permitted

Protection Fault

Physical Address (to cache)

Where?
Sequential Consistency

• “A system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the order specified by the program” Leslie Lamport

  - Sequential Consistency = arbitrary order-preserving interleaving of memory references of sequential programs
Cache State Transition Diagram

• The MSI protocol

Each cache line has a tag

<table>
<thead>
<tr>
<th>Address tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>state bits</td>
</tr>
</tbody>
</table>

M: Modified
S: Shared
I: Invalid

Write miss
Other processor intends to write
P₁ writes back
Other processor writes or reads
P₁ reads

Read miss
Other processor reads
P₁ writes back

Read by any processor
P₁ intends to write
Other processor intends to write
Cache state in processor P₁
MESI: An Enhanced MSI protocol

Each cache line has a tag

<table>
<thead>
<tr>
<th>state bits</th>
<th>Address tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>M: Modified Exclusive</td>
<td></td>
</tr>
<tr>
<td>E: Exclusive, unmodified</td>
<td></td>
</tr>
<tr>
<td>S: Shared</td>
<td></td>
</tr>
<tr>
<td>I: Invalid</td>
<td></td>
</tr>
</tbody>
</table>

P₁ write or read

Other processor reads
P₁ writes back

Read miss, shared

Read by any processor

P₁ intent to write

Write miss
Other processor intent to write

P₁ read

Read miss, not shared

Cache state in processor P₁
False Sharing

- A cache block contains more than one word
- Cache-coherence is done at the block-level and not word-level
- Suppose $M_1$ writes word $i$ and $M_2$ writes word $k$ and both words have the same block address.

- What can happen?
  - Block may be invalidated many times unnecessarily
• Use snoopy mechanism to keep all processors’ view of memory coherent
Directory Protocols

• Directory keeps the state of every block that is cached
  ‣ Which caches have copies, which do not?

• Directory entries can be distributed, so different directory accesses go to different locations
  ‣ Sharing status of a block always in single known location
  ‣ Will not consider this in this class
Snoopy vs. Directory

• States and transitions on the cache side are similar but actions on transition are different

• Cannot use interconnect as a single point of arbitration in directory scheme

• Interconnect is message-oriented (rather than a transaction-oriented bus) and many messages have explicit responses
On-Chip Networks

- Higher bandwidth with more concurrent communications
- More scalable with better electrical properties
Architecture of Interconnection Networks

- How to connect the nodes up (processors, memories, router line cards, SoC modules)?
  - Topology

- Which path should a message take?
  - Routing and deadlock

- How is the message actually forwarded from source to destination?
  - Flow Control
Architecture of Interconnection Networks

• How to build the routers?
  ‣ Router microarchitecture

• How to build the links?
  ‣ Link Architecture

• How do processing core talk to the network?
  ‣ Network Interface