Computer Organization
CS 231-01

Improving Performance

Dr. William H. Robinson
November 8, 2004

http://eecs.vanderbilt.edu/courses/cs231/

Topics

“Money's only important when you don't have any.”

– Sting

• Cache
• Scoreboarding

Opportunity for (Easy) Points

• Prepare for Quiz #4

Three Generic Data Hazards

• Read After Write (RAW)
  Instr_j tries to read operand before Instr_i writes it

\[
\begin{align*}
I: & \text{ add } r1, r2, r3 \\
J: & \text{ sub } r4, r1, r3 \\
\end{align*}
\]

• Caused by a “dependence” (in compiler nomenclature). This hazard results from an actual need for communication.

Adapted from John Kubiatowicz’s CS 252 lecture notes. Copyright © 2003 UCB.
Three Generic Data Hazards

• Write After Read (WAR)
  Instr\(_j\) writes operand before Instr\(_i\) reads it

\[\begin{align*}
  I: & \text{ sub } r4, r1, r3 \\
  J: & \text{ add } r1, r2, r3 \\
  K: & \text{ mul } r6, r1, r7 \\
\end{align*}\]

• Called an “anti-dependence” by compiler writers. This results from reuse of the name “r1”.

• Can’t happen in MIPS 5-stage pipeline because:
  – All instructions take 5 stages, and
  – Reads are always in stage 2, and
  – Writes are always in stage 5

The Principle of Locality

• The Principle of Locality:
  – 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 (e.g., loops, reuse)
  – Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)

• Last 15 years, H/W relied on locality for speed

What is a Cache?

• Small, fast storage used to improve average access time to slow memory.

• Exploits spatial and temporal locality

• In computer architecture, almost everything is a cache!
  – Registers a cache on variables
  – First-level cache a cache on second-level cache
  – Second-level cache a cache on memory
  – Memory a cache on disk (virtual memory)

The Principle of Locality

• The Principle of Locality:
  – 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 (e.g., loops, reuse)
  – Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)

• Last 15 years, H/W relied on locality for speed

The Principle of Locality

• The Principle of Locality:
  – 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 (e.g., loops, reuse)
  – Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)

• Last 15 years, H/W relied on locality for speed
**Four Questions for Memory Hierarchy Designers**

- **Q1:** Where can a block be placed in the upper level? *(Block placement)*
  - Fully Associative, Set Associative, Direct Mapped
- **Q2:** How is a block found if it is in the upper level? *(Block identification)*
  - Tag/Block
- **Q3:** Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU
- **Q4:** What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

**Types of Caches**

- **Direct-mapped**
  - Cache line only has 1 possible location
  - Simple to implement
- **Set-associative**
  - Cache line has a “set” of possible locations
  - e.g. 4-way set-associative
- **Fully-associative**
  - Cache line can be placed anywhere
  - Costly in H/W to check entire cache for hit

**Block Identification**

- Generally tags checked in parallel for speed
- Block offset not needed for comparison
- Checking the index is redundant
- Also need valid bit

**Replacement Policy**

- **Random**
  - Spreads allocation uniformly
  - Implemented with pseudorandom number generator
- **Least Recently Used (LRU)**
  - Blocks not accessed for longest time are replaced
  - Complex to implement
- **First In First Out (FIFO)**
  - Determines the oldest block
  - Approximates LRU
Write Policy:
Write-Through vs. Write-Back

- **Write-through**: all writes update cache and underlying memory/cache
  - Can always discard cached data - most up-to-date data is in memory
  - Cache control bit: only a valid bit

- **Write-back**: all writes simply update cache
  - Can’t just discard cached data - may have to write it back to memory
  - Cache control bits: both valid and dirty bits

Other Advantages:
- Write-through:
  - Memory (or other processors) always have latest data
  - Simpler management of cache
- Write-back:
  - Much lower bandwidth, since data often overwritten multiple times
  - Better tolerance to long-latency memory?

Write Policy 2:
Write Allocate vs. No-Write Allocate

What happens on write-miss
- Write allocate: allocate new cache line in cache
  - Usually means that you have to do a "read miss" to fill in rest of the cache-line!

- No-write allocate (or "write-around"):
  - Simply send write data through to underlying memory/cache; don’t allocate new cache line!

Terminology
- **Hit** – memory location found in cache
- **Miss** – memory location not in cache
  - Compulsory
    - Cache is empty at the start of a program
  - Conflict
    - Another valid cache line is currently stored in the location
  - Capacity
    - Cache isn’t large enough to hold the entire working set of a program
### Modern CPU Design

- **Pipelined** — instructions execute in stages
- **Superscalar** — contains multiple functional units

- **In-Order Execution**
  - Issue instructions in program order
  - Retire (complete) instructions in program order
  - May not give optimal performance because of instruction dependencies
    - Created whenever there is a dependence between instructions where pipelining overlap changes the order of access
      - RAW
      - WAW
      - WAR

### 3Cs Absolute Miss Rate (SPEC92)

### 2:1 Cache Rule

\[
\text{miss rate 1-way associative cache size } X = \text{miss rate 2-way associative cache size } X/2
\]
Data Hazards

- **Read After Write (RAW)**
  - True data dependence (most common type)
  - Must preserve program order to ensure correct execution

- **Write After Read (WAR)**
  - Output dependence
  - Only present in pipelines that
    - write in more than one pipe stage
    - allow instructions to proceed after previous instructions stalls

- **Write After Write (WAW)**
  - Anti-dependence
  - Can occur when instructions are reordered

Example: Pipelined, Superscalar CPU

- **2-way superscalar**
  - Can issue up to 2 instructions per cycle

- **For instructions decoded in cycle** $n$
  - Execution starts in cycle $n + 1$
  - ADD/SUB completes in cycle $n + 2$
  - MUL completes in cycle $n + 3$

Rules for Issuing Instructions

- **True data dependence (RAW)**
  - Don’t issue if any operand is being written

- **Anti-dependence (WAR)**
  - Don’t issue if result register is being read

- **Output dependence (WAW)**
  - Don’t issue if result register is being written

Scoreboarding: In-Order Issue/Completion

- A superscalar CPU with in-order issue and in-order completion.
Scoreboarding: In-Order Issue/Completion

- Instruction 4 has a true dependence (RAW) from Instruction 2
  - Stall until R4 is available

Scoreboarding: In-Order Issue/Completion

- Instruction 2 actually completes during cycle 3
  - Must retire instructions in order

Summary

- The average memory access time (AMAT) of a cache can be improved by reducing miss rate

- Scoreboarding is a technique to monitor the dynamic scheduling of instructions

- In-order execution and in-order completion can limit performance