2
Key Concepts

18-548/15-548 Memory System Architecture
Philip Koopman
August 26, 1998

Required Reading: Hennessy & Patterson page 7; Section 1.7
Cragon Chapter 1
Supplemental Reading: Hennessy & Patterson Chapter 1

Assignments

- **By next class read about Physical Memory Architecture:**
  - Hennessy & Patterson (skim): 5.1, 6.1-6.2, pp. 496-497, 7.1, pp. 635-642
  - Cragon 2.0
  - http://www.isdmag.com/Editorial/1996/CoverStory9610.html
  - Supplemental Reading (review): Hennessy & Patterson: 2.0-2.3, 2.7
- **Homework #1 due next Wednesday 9/2 in class**
- **Lab 1 due Friday 9/4 at 3 PM**

Where Are We Now?

- **Where we're going today:**
  - Key concepts: latency, bandwidth, concurrency, balance
- **Where we're going next:**
  - Physical vs. virtual memory
  - Cache operation

Preview

- **Latency**
  - Time delay to accomplish a memory access
- **Bandwidth**
  - Data moved per unit time
- **Concurrency**
  - Can help latency, bandwidth, or both (usually bandwidth is easier to provide)
- **Balance**
  - Amdahl's law -- a bottleneck will dominate speedup limitations

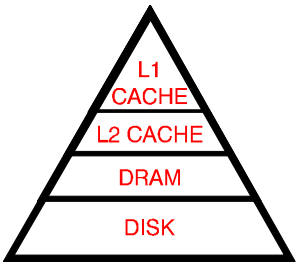
Key Architectural Concepts

- **How do you keep a fast CPU fed?**
  - Use a Memory Hierarchy
- **What concepts determine the effectiveness of a memory hierarchy?**
  - Latency
  - Bandwidth
  - Concurrency
  - Balance

HIERARCHIES

Hierarchies Exploit Locality

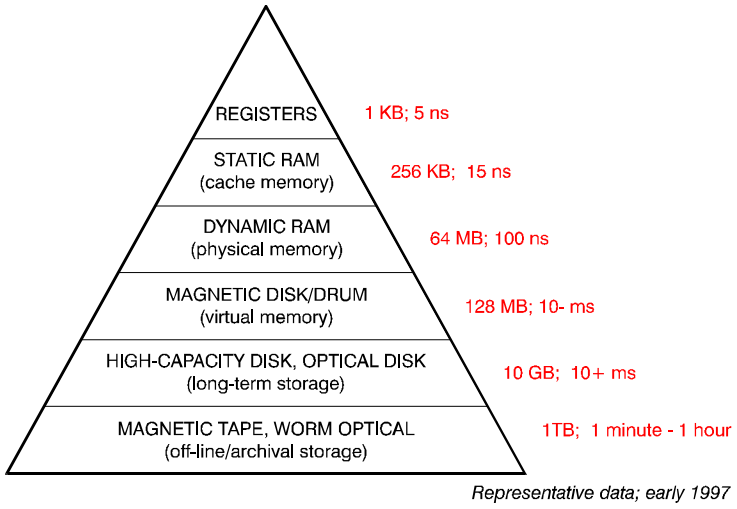
- **Assume that programs have *locality***
  - Loops, subroutines, heavily used data structures
  - "Working set" memory footprint smaller than entire memory space
- **Use memory hierarchy: "layered capabilities"**
  - Small memories designed to be fast & wide, but expensive
  - Large memories designed to be slow & "narrow", but cheap
- **Personal computer example:**
  - 16 KB level 1 cache
  - 512 KB level 2 cache
  - 32 MB DRAM
  - 2 GB hard disk



- **BUT, hierarchies only work when (the right kind of) locality really exists**

Memory Hierarchy as a Solution

- **Use small amounts of fast memory; larger amounts of slow memory**



LATENCY

Latency Is Delay

- **Latency is delay between start and completion of an activity**
  - Memory access time:
    - Read: request -> data ready for use (very important)
    - Write: request -> CPU allowed to proceed with next operation
  - Communication: message creation -> message delivery
- **Latency varies depending on program history**
  - Which level of the memory hierarchy is data stored in?
  - What other activities are interfering with this one?
  - Is resource needed idle? Does it need to be powered up? *etc.*
- **Latency example: read from main memory**
  - 1982 -- 4.77 MHz Original IBM PC
    4 clock cycles (uncached) = 839 ns
  - 1988 -- 16 MHz Titan 1 mini-supercomputer
    ~17 clocks for cache miss = 1063 ns
  - 1998 -- 400 MHz AlphaStation 500/400
    ~48 clocks for cache miss = 120 ns

Average Latency Reduction

- **Memory hierarchy goal is to reduce average observed latency**

Cache memory example:

L1 cache access in 10 ns; main memory access in 100 ns; 90% hit rate

For every 10 memory accesses, 9 will be to cache and 1 to main memory

Time without cache = 10 * 100ns = 1000 ns
Time with cache = 9 * 10ns + 1 * 100ns = 190 ns

$$\text{Speedup} = \frac{\text{Time without Enhancement}}{\text{Time with Enhancement}}$$

$$\text{Speedup} = \frac{1000}{190} \approx 5.3$$

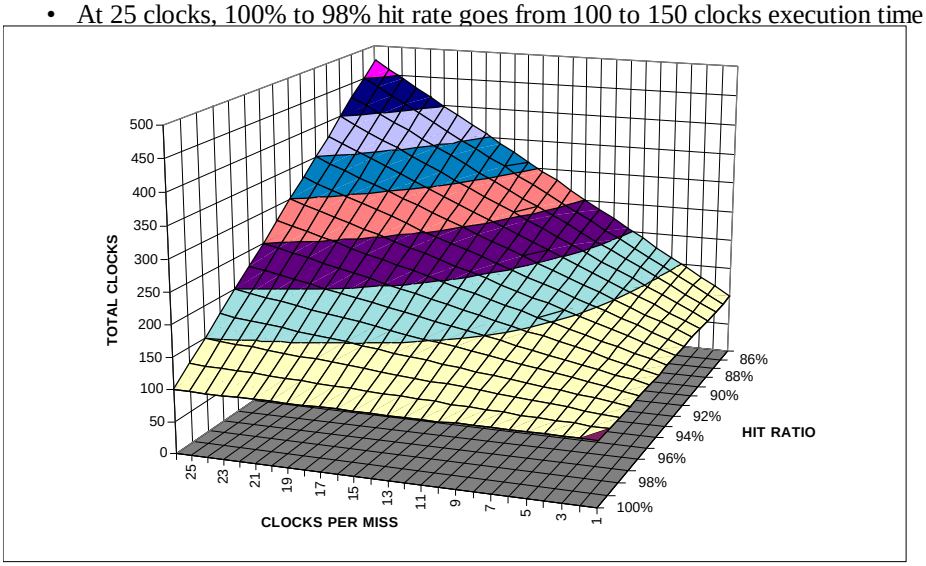
Example Latencies

- **Alpha 21164 Latencies (at ~ 400 MHz):**
  - 1 clock access to on-chip Level 1 cache
  - 6 clock access to on-chip Level 2 cache
  - 48 clock access to DRAM (120 ns)
  - ~4,000,000 clock access to an representative disk drive

- **Pentium Pro Latencies (at ~200 MHz):**
  - 1 clock access to on-chip Level 1 cache
  - 6-1-1-1 clock access to off-chip Level 2 cache
  - 28 clock access to DRAM (140 ns)
  - ~2,000,000 clock access to an example disk drive

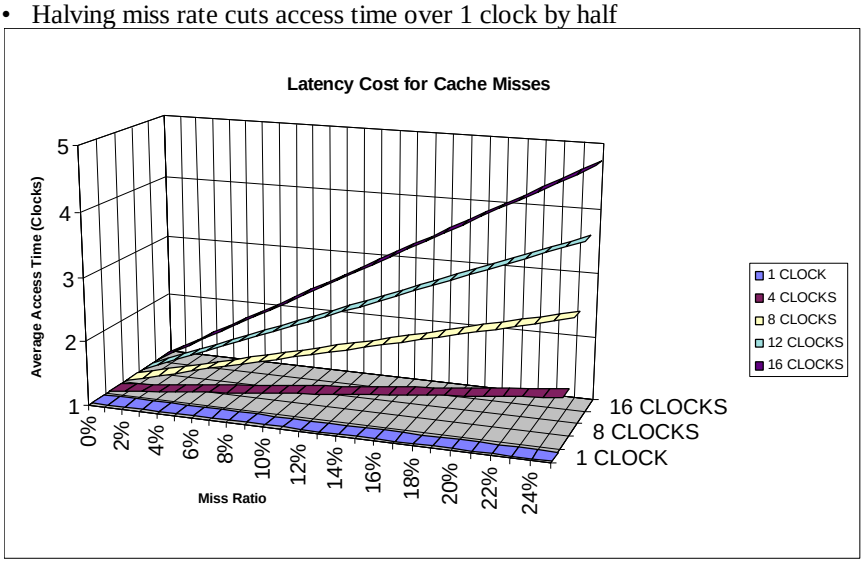
Why Latency Matters

- **A few expensive accesses can overshadow many fast accesses**
  - At 25 clocks, 100% to 98% hit rate goes from 100 to 150 clocks execution time



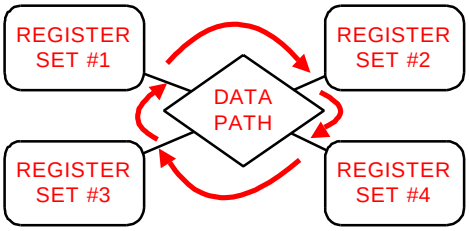
Why Latency Matters -- Part 2

- **Miss Ratio is a more useful concept than Hit Ratio**
  - Halving miss rate cuts access time over 1 clock by half



Latency Tolerance

- **CPU design can tolerate latency**
- **Pipelining**
  - Ordinary CPU pipeline
  - Vector processing
- **Out-of-order execution**
  - Issue all possible operations, subject to data dependencies
  - Speculative execution (guess results of conditional branches, *etc.*)
- **Multi-threading**
  - Perform context switch on cache miss
  - Execute multiple streams in parallel using multiple register sets



BANDWIDTH

Provide High Bandwidth

- **Bandwidth is amount of data moved per unit time**
  Bandwidth = bit rate * number bits

- **Fast bit rates**
  - Short traces
  - Careful design technique
  - High-speed technology
- **Large number of bits**
  - Large number of pins on package
  - Large number of signals on bus
  - Wide memories
  - Multiported memories
- **Faster average memory cycle time**
  - Interleaved memories
  - Burst-mode memories

High Sustained Bandwidth

- **Provide higher average bandwidth by maximizing bandwidth near top of memory hierarchy**

- **Split cache example:**
  - Unified cache:
    16KB, 64-bit single-ported cache at 100 MHz = 800 MB/sec
  - Split cache (separated instruction & data caches)
    2 @ 8KB 64-bit single-ported caches at 100 MHz = 1600 MB/sec

- **Cache block size example:**
  - Assume 64-bit interface from CPU to memory; on-chip L1 cache
  - 64-bit L1 cache block at 100 MHz = 800 MB/sec
  - 256-bit L1 cache block at 100 MHz = 3200 MB/sec
  - Note: bandwidth off-chip is the same in either case; wider block only helps for cache hits -- *average* improvement

Provide Peak Bandwidth

- **PEAK = "guaranteed not to exceed"**
- **Exploit locality to perform block data transfers**
  - Amortize addressing time over multiple word accesses
    - Large cache lines transferred as blocks
    - Page mode on DRAM (provides fast access to a block of addresses)
  - Make cache lines wide to provide more bandwidth on-chip (*e.g.*, superscalar instruction caches are wider than one machine word)
- **Analogue to bandwidth in computations is "peak MFLOPS"**
  - Provide structure capable of multiply-accumulate every clock cycle
  - Provide operand register file faster than system bus

Example Bandwidths

- **Alpha 21164 Bandwidths:**
  - 4.8 GB/sec to L1 cache
  - 1.2 GB/sec to L2 cache
  - 1 GB/sec to DRAM/disk
  - 10 - 100 Mbit/sec to internet via campus network
- **Pentium Pro Bandwidths:**
  - ~3.2 GB/sec to L1 cache
  - 528 MB/sec to L2 cache
  - 132 MB/sec to DRAM
    (528 MB/sec for multiprocessor with interleaved memory)
  - 128 Kbit/sec to internet via ISDN; or worse via modem

CONCURRENCY

Replication of Homogeneous Resources

- **Concurrency can be achieved by using replicated resources**
- **Replication is using multiple instances of a resource**
  - Potentially lower latency if concurrency can be exploited
    - Multiple banks of memory (interleaving) combined with concurrent accesses to several banks rather than waiting for a single bank to complete
  - Potentially higher bandwidth if enough connectivity exists
    - Split instruction and data caches
    - Multiple buses if bus speed is limited
  - Replication in memory system required for efficient parallel processing
    - At least one cache per CPU for multiprocessor
    - Multiple memory banks for multiple simultaneous accesses

Pipelining as a Form of Concurrency

- **Pipelining is an inexpensive way to approximate replication**
  - One set of resources can be shared *N* ways with an *N*-stage pipeline
  - Less effective because logical copies of resource must operate out of phase spread over N clock cycles
  - Example: 6-clock multiplier, if pipelined, can produce 6 times as many results, but each result is still at a 6-clock latency
- **Replicated resources can each be pipelined too**

Thought Question:

- **In everyday life, think of an instance where:**
  - Replication increases bandwidth and decreases latency
  - Replication increases both bandwidth and latency
  - Replication decreases latency without directly affecting bandwidth
  - Replication increases bandwidth without directly affecting latency

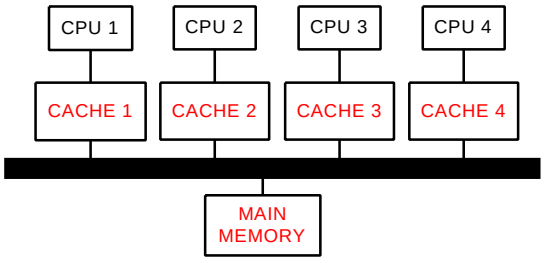
(hint -- think of city transportation)

Computer Concurrency Examples

- **Replication increases bandwidth and decreases latency**
  - Split instruction and data cache -- parallel access to both; single-clock instruction execution possible
- **Concurrency increases both bandwidth and latency**
  - A pipelined processor, especially a vector processor
- **Replication decreases latency without directly affecting bandwidth**
  - The size of a cache memory (number of bytes in cache)
- **Replication increases bandwidth without directly affecting latency**
  - 64-bit bus instead of 32-bit bus (for 32-bit data value computations)

Coherence as a Replication Issue

- **Coherence required to synchronize contents of distributed memory structures**
  - Caches in multiprocessor provide:
    - Increased bandwidth to CPU
    - Decreased latency to CPU
    - Reduced bus traffic
    - ***Potential for same data to be resident in multiple caches***

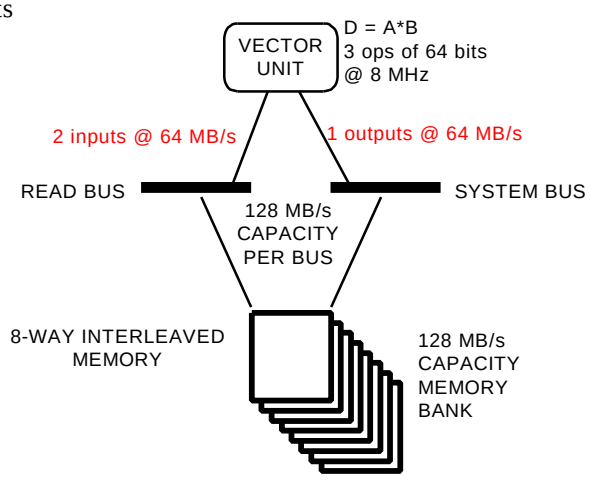


BALANCE

Balance = Lack of Steady-State Bottlenecks

- **A machine is balanced when at each level of the hierarchy the system is sized appropriately for the demands placed upon it**
  - Balance usually refers to an intended steady-state operation point
  - Lack of balance manifests as a ***bottleneck***

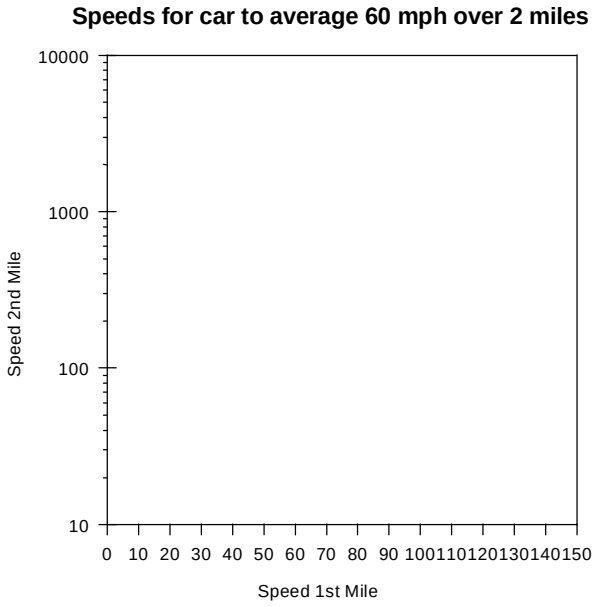
Example:
Titan vector processing



Balance example

- **You have to travel 2 miles and average 60 M.P.H.**
  - You travel the first mile at 30 M.P.H.
  - How fast do you have to go for the second mile to average 60 M.P.H. for the entire 2-mile trip?

Vehicle Speed Balance Example



Amdahl's Law

$$SPEEDUP = \frac{1}{(1 - FRACTION_{ENHANCED}) + \left(\frac{FRACTION_{ENHANCED}}{SPEEDUP_{ENHANCED}}\right)}$$

- **Parallel computations are limited by the scalar portion**
  - Example: 10 CPUs can be used to speed up half a computation; the other half runs on one CPU

$$SPEEDUP = \frac{1}{(1 - 0.5) + \left(\frac{0.5}{10}\right)} = 1.82 \text{ times faster}$$

- **Insight: infinite parallelism doesn't help the scalar portion!**
  - 50% parallel code runs, ***at most***, 2x faster

- **Make the common case fast; but after a while it won't be so common (in terms of time consumed)...**

Amdahl's Law Extended

- **Speedup of part of a calculation is limited by lack of speedup in the rest of the calculation**

Cache memory example revisited:

L1 cache access in 10 ns; main memory access in 100 ns; 90% hit rate

Time without cache = 10 * 100ns = 1000 ns
Time with cache = 9 * 10ns + 1 * 100ns = 190 ns

$$SPEEDUP = \frac{1}{(1 - FRACTION_{ENHANCED}) + \left(\frac{FRACTION_{ENHANCED}}{SPEEDUP_{ENHANCED}}\right)}$$

$$SPEEDUP = \frac{1}{(1 - 0.90) + \left(\frac{0.90}{\left(\frac{100ns}{10ns}\right)}\right)} = \frac{1}{0.10 + \left(\frac{0.90}{10}\right)} = \frac{1}{0.10 + 0.09} = \frac{1}{0.19} \approx 5.3$$

Amdahl / Case Ratio

- **1 MB memory; 1 Mbit/sec I/O; 1 MIPS**
  - Gives rule of thumb for balanced mainframe (from the 1970's)
- **Example: AlphaServer 1000A (21164 CPU) base configuration**
  - ~700 MIPS at 500 MHz
  - 256 MB memory
  - ~100 Mbit/sec disk I/O
  - Lessons:
    - Get a memory upgrade (*e.g.*, to 512 MB) if you're doing serious computing
    - "I/O certainly has been lagging in the last decade" -- Seymour Cray, 1976
    - "Also, I/O needs a lot of work." -- David Kuck, 1988
    - "We're working on I/O" -- Dave Nagle & Greg Ganger, 1998

Other Balance Issues

- **Low latency is harder to provide than high bandwidth**
  - Amdahl's law applied to cache misses
- **Context switching overhead**
  - Amdahl's law applied to register saves & task switches
- **Price/performance points**
  - Often price matters more than performance
  - This course is, however, (mostly) about performance

REVIEW

Review

- **Hierarchy**
  - Is a general approach to exploiting the common case (“locality”)
- **Latency**
  - Minimize time delays to minimize waiting time
- **Bandwidth**
  - Maximize width & speed
- **Concurrency**
  - Bandwidth increase if more connectivity or pipelining
  - Latency decrease for concurrency
- **Balance**
  - Amdahl’s law applies
- **Next lecture: physical memory architecture**