EECS 252 Graduate Computer Architecture

Lec 3 – Performance + Pipeline Review

David Patterson
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~patterson
http://www-inst.eecs.berkeley.edu/~cs252

Review from last lecture

• Tracking and extrapolating technology part of architect’s responsibility
• Expect Bandwidth in disks, DRAM, network, and processors to improve by at least as much as the square of the improvement in Latency
• Quantify Cost (vs. Price)
  – IC \approx f(Area^2) + Learning curve, volume, commodity, margins
• Quantify dynamic and static power
  – Capacitance x Voltage^2 x frequency, Energy vs. power
• Quantify dependability
  – Reliability (MTTF vs. FIT), Availability (MTTF/(MTTF+MTTR)

Outline

• Review
• Quantify and summarize performance
  – Ratios, Geometric Mean, Multiplicative Standard Deviation
• F&P: Benchmarks age, disks fail,1 point fail danger
• 252 Administrivia
• MIPS – An ISA for Pipelining
• 5 stage pipelining
• Structural and Data Hazards
• Forwarding
• Branch Schemes
• Exceptions and Interrupts
• Conclusion

Definition: Performance

• Performance is in units of things per sec
  – bigger is better
• If we are primarily concerned with response time

\[
\text{performance}(x) = \frac{1}{\text{execution}_\text{time}(x)}
\]

"X is n times faster than Y" means

\[
n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(X)}
\]
Performance: What to measure

- Usually rely on benchmarks vs. real workloads
- To increase predictability, collections of benchmark applications-- benchmark suites -- are popular
- SPECCPU: popular desktop benchmark suite
  - CPU only, split between integer and floating point programs
  - SPECint2000 has 12 integer, SPECfp2000 has 14 integer pgms
  - SPECCPU2006 to be announced Spring 2006
  - SPECSFS (NFS file server) and SPECWeb (WebServer) added as server benchmarks
- Transaction Processing Council measures server performance and cost-performance for databases
  - TPC-C Complex query for Online Transaction Processing
  - TPC-H models ad hoc decision support
  - TPC-W a transactional web benchmark
  - TPC-App application server and web services benchmark

How Summarize Suite Performance (1/5)

- Arithmetic average of execution time of all pgms?
  - But they vary by 4X in speed, so some would be more important than others in arithmetic average
- Could add a weights per program, but how pick weight?
  - Different companies want different weights for their products
- SPECRatio: Normalize execution times to reference computer, yielding a ratio proportional to performance =
  \[
  \frac{\text{time on reference computer}}{\text{time on computer being rated}}
  \]

How Summarize Suite Performance (2/5)

- If program SPECRatio on Computer A is 1.25 times bigger than Computer B, then
  \[
  1.25 = \frac{\text{SPECRatio}_A}{\text{SPECRatio}_B} = \frac{\text{ExecutionTime}_A}{\text{ExecutionTime}_B}
  \]
  \[
  = \frac{\text{ExecutionTime}_B}{\text{ExecutionTime}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}
  \]
- Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so choice of reference computer is irrelevant

How Summarize Suite Performance (3/5)

- Since ratios, proper mean is geometric mean (SPECRatio unitless, so arithmetic mean meaningless)

  \[
  \text{GeometricMean} = \sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio}_i}
  \]
- 2 points make geometric mean of ratios attractive to summarize performance:
  1. Geometric mean of the ratios is the same as the ratio of the geometric means
  2. Ratio of geometric means = Geometric mean of performance ratios
     ⇒ choice of reference computer is irrelevant!
How Summarize Suite Performance (4/5)

- Does a single mean well summarize performance of programs in benchmark suite?
- Can decide if mean a good predictor by characterizing variability of distribution using standard deviation
- Like geometric mean, geometric standard deviation is multiplicative rather than arithmetic
- Can simply take the logarithm of SPEC Ratios, compute the standard mean and standard deviation, and then take the exponent to convert back:

\[
\text{GeometricMean} = \exp \left( \frac{1}{n} \sum_{i=1}^{n} \ln(SPECRatio_i) \right)
\]

\[
\text{GeometricStDev} = \exp \left( \text{StDev} \left( \ln(SPECRatio_i) \right) \right)
\]

How Summarize Suite Performance (5/5)

- Standard deviation is more informative if know distribution has a standard form
  - bell-shaped normal distribution, whose data are symmetric around mean
  - lognormal distribution, where logarithms of data—not data itself—are normally distributed (symmetric) on a logarithmic scale
- For a lognormal distribution, we expect that 68% of samples fall in range \( [\text{mean} / \text{gstdev}, \text{mean} \times \text{gstdev}] \)
  95% of samples fall in range \( [\text{mean} / \text{gstdev}^2, \text{mean} \times \text{gstdev}^2] \)
- Note: Excel provides functions EXP(), LN(), and STDEV() that make calculating geometric mean and multiplicative standard deviation easy

Example Standard Deviation (1/2)

- GM and multiplicative StDev of SPECfp2000 for Itanium 2

```
GM = 2712
GStDev = 1.98
```

Example Standard Deviation (2/2)

- GM and multiplicative StDev of SPECfp2000 for AMD Athlon

```
GM = 2086
GStDev = 1.40
```

```
Comments on Itanium 2 and Athlon

- Standard deviation of 1.98 for Itanium 2 is much higher—vs. 1.40—so results will differ more widely from the mean, and therefore are likely less predictable.
- SPEC Ratios falling within one standard deviation:
  - 10 of 14 benchmarks (71%) for Itanium 2
  - 11 of 14 benchmarks (78%) for Athlon
- Thus, results are quite compatible with a lognormal distribution (expect 68% for 1 StDev).

Fallacies and Pitfalls (1/2)

- Fallacies - commonly held misconceptions
  - When discussing a fallacy, we try to give a counterexample.
- Pitfalls - easily made mistakes.
  - Often generalizations of principles true in limited context
  - Show Fallacies and Pitfalls to help you avoid these errors.
- Fallacy: Benchmarks remain valid indefinitely
  - Once a benchmark becomes popular, tremendous pressure to improve performance by targeted optimizations or by aggressive interpretation of the rules for running the benchmark: “benchmarksmanship.”
  - 70 benchmarks from the 5 SPEC releases, 70% were dropped from the next release since no longer useful.
- Pitfall: A single point of failure
  - Rule of thumb for fault tolerant systems: make sure that every component was redundant so that no single component failure could bring down the whole system (e.g., power supply).

Fallacies and Pitfalls (2/2)

- Fallacy - Rated MTTF of disks is 1,200,000 hours or ≈ 140 years, so disks practically never fail.
- But disk lifetime is 5 years ⇒ replace a disk every 5 years; on average, 28 replacements wouldn't fail.
- A better unit: % that fail (1.2M MTTF = 833 FIT).
- Fail over lifetime: if had 1000 disks for 5 years = 1000*(5*365*24)*833 /10^9 = 36,485,000 / 10^6 = 37 = 3.7% (37/1000) fail over 5yr lifetime (1.2M hr MTTF).
- But this is under pristine conditions
  - little vibration, narrow temperature range ⇒ no power failures.
- Real world: 3% to 6% of SCSI drives fail per year
  - 3400 - 6800 FIT or 150,000 - 300,000 hour MTTF [Gray & van Ingen 05]
- 3% to 7% of ATA drives fail per year
  - 3400 - 8000 FIT or 125,000 - 300,000 hour MTTF [Gray & van Ingen 05]

CS252: Administrivia

Instructor: Prof David Patterson
  Office: 635 Soda Hall, patterson@cs
  Office Hours: Tue 11 - noon or by appt.
  (Contact Cecilia Pracher; cpracher@eecs)
T. A: Archana Ganapathi, archanag@eecs
Class: M/W, 11:00 - 12:30pm    203 McLaughlin (and online)
Web page: http://www.cs/~patterson/courses/cs252-S06/
  Lectures available online <9:00 AM day of lecture
Wiki page: ??
Reading assignment: Memory Hierarchy Basics Appendix C (handout) for Mon 1/30
Wed 2/1: Great ISA debate (3 papers) + Prerequisite Quiz
Outline

- Review
- Quantify and summarize performance
  - Ratios, Geometric Mean, Multiplicative Standard Deviation
- F&P: Benchmarks age, disks fail, 1 point fail danger
- 252 Administrivia
- MIPS – An ISA for Pipelining
- 5 stage pipelining
- Structural and Data Hazards
- Forwarding
- Branch Schemes
- Exceptions and Interrupts
- Conclusion

A "Typical" RISC ISA

- 32-bit fixed format instruction (3 formats)
- 32 32-bit GPR (R0 contains zero, DP take pair)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store: base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch

Example: MIPS (- MIPS)

Datapath vs Control

- Datapath: Storage, FU, interconnect sufficient to perform the desired functions
  - Inputs are Control Points
  - Outputs are signals
- Controller: State machine to orchestrate operation on the data path
  - Based on desired function and signals

See: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3
Approaching an ISA

• Instruction Set Architecture
  – Defines set of operations, instruction format, hardware supported data types, named storage, addressing modes, sequencing
• Meaning of each instruction is described by RTL on architected registers and memory
• Given technology constraints assemble adequate datapath
  – Architected storage mapped to actual storage
  – Function units to do all the required operations
  – Possible additional storage (e.g. MAR, MBR, …)
  – Interconnect to move information among regs and FUs
• Map each instruction to sequence of RTLs
• Collate sequences into symbolic controller state transition diagram (STD)
• Lower symbolic STD to control points
• Implement controller

5 Steps of MIPS Datapath

Figure A.2, Page A-8

5 Steps of MIPS Datapath
Figure A.3, Page A-9

Inst. Set Processor Controller
5 Steps of MIPS Datapath

- Instruction Fetch
- Instr. Decode
- Reg. Fetch
- Execute Addr. Calc
- Memory Access
- Write Back

- Data stationary control
  - local decode for each instruction phase / pipeline stage

Visualizing Pipelining

Time (clock cycles)

- Cycle 1
- Cycle 2
- Cycle 3
- Cycle 4
- Cycle 5
- Cycle 6
- Cycle 7

Pipelining is not quite that easy!

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).
One Memory Port/Structural Hazards
(Similar to Figure A.5, Page A-15)

<table>
<thead>
<tr>
<th>Time (clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 1</td>
</tr>
<tr>
<td>Load</td>
</tr>
<tr>
<td>Instr 1</td>
</tr>
<tr>
<td>Instr 2</td>
</tr>
<tr>
<td>Stall</td>
</tr>
<tr>
<td>Instr 3</td>
</tr>
</tbody>
</table>

How do you “bubble” the pipe?

Speed Up Equation for Pipelining

\[
\text{CPI}_{\text{pipelined}} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}
\]

\[
\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}
\]

For simple RISC pipeline, CPI = 1:

\[
\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}
\]

Example: Dual-port vs. Single-port

- Machine A: Dual ported memory (“Harvard Architecture”)
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed
  \[
  \text{SpeedUp}_A = \text{Pipeline Depth}/(1 + 0) \times (\text{clock}_{\text{unpipe}}/\text{clock}_{\text{pipe}})
  \]
  \[
  = \text{Pipeline Depth}
  \]
  \[
  \text{SpeedUp}_B = \text{Pipeline Depth}/(1 + 0.4 \times 1) \times (\text{clock}_{\text{unpipe}}/\text{clock}_{\text{unpipe}} / 1.05)
  \]
  \[
  = (\text{Pipeline Depth}/1.4) \times 1.05
  \]
  \[
  = 0.75 \times \text{Pipeline Depth}
  \]
  \[
  \text{SpeedUp}_A / \text{SpeedUp}_B = \text{Pipeline Depth}/(0.75 \times \text{Pipeline Depth}) = 1.33
  \]
- Machine A is 1.33 times faster

Data Hazard on R1
(Figure A.6, Page A-17)
Three Generic Data Hazards

- **Read After Write (RAW)**
  Instr\(_J\) tries to read operand before Instr\(_I\) writes it.

  \[ \text{I: } \text{add } r1, r2, r3 \]
  \[ \text{J: } \text{sub } r4, r1, r3 \]

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

Three Generic Data Hazards

- **Write After Read (WAR)**
  Instr\(_J\) writes operand before Instr\(_I\) reads it.

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

- 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

Three Generic Data Hazards

- **Write After Write (WAW)**
  Instr\(_J\) writes operand before Instr\(_I\) writes it.

  \[ \text{I: } \text{add } r1, r4, r3 \]
  \[ \text{J: } \text{add } r1, r2, r3 \]
  \[ \text{K: } \text{mul } r6, r1, r7 \]

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

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

  Will see WAR and WAW in more complicated pipes

---

Forwarding to Avoid Data Hazard

**Figure A.7, Page A-19**

**Time (clock cycles)**

\[ \text{add } r1, r2, r3 \]
\[ \text{sub } r4, r1, r3 \]
\[ \text{and } r6, r1, r7 \]
\[ \text{or } r8, r1, r9 \]
\[ \text{xor } r10, r1, r11 \]
What circuit detects and resolves this hazard?

Data Hazard Even with Forwarding
(Similar to Figure A.10, Page A-21)
Software Scheduling to Avoid Load Hazards

Try producing fast code for
\[ a = b + c; \]
\[ d = e - f; \]
assuming \( a, b, c, d, e, \) and \( f \) in memory.

- **Slow code:**
  - LW \( Rb, b \)
  - LW \( Rc, c \)
  - ADD \( Ra, Rb, Rc \)
  - SW \( a, Ra \)
  - LW \( Re, e \)
  - LW \( Rf, f \)
  - SW \( d, Rd \)

- **Fast code:**
  - LW \( Rb, b \)
  - LW \( Rc, c \)
  - ADD \( Ra, Rb, Rc \)
  - SW \( a, Ra \)
  - LW \( Re, e \)
  - LW \( Rf, f \)
  - SW \( d, Rd \)

Compiler optimizes for performance. Hardware checks for safety.

Outline

- Review
- Quantify and summarize performance
  - Ratios, Geometric Mean, Multiplicative Standard Deviation
- F&P: Benchmarks age, disks fail, 1 point fail danger
- 252 Administrivia
- MIPS – An ISA for Pipelining
- 5 stage pipelining
- Structural and Data Hazards
- Forwarding
- Branch Schemes
- Exceptions and Interrupts
- Conclusion

Control Hazard on Branches

Three Stage Stall

10: beq r1, r3, 36
14: and r2, r3, r5
18: or r6, r1, r7
22: add r8, r1, r9
36: xor r10, r1, r11

What do you do with the 3 instructions in between?
How do you do it?
Where is the “commit”?

Branch Stall Impact

- If CPI = 1, 30% branch, Stall 3 cycles => new CPI = 1.9!
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- MIPS branch tests if register = 0 or \( \neq 0 \)
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3
**Pipelined MIPS Datapath**

Figure A.24, page A-38

- Instruction Fetch
- Instr. Decode
- Reg. Fetch
- Execute
- Addr. Calc
- Memory Access
- Write Back

- Next PC
- Address
- MUX
- Memory
- Reg File
- ALU
- WB Data

• Interplay of instruction set design and cycle time.

**Four Branch Hazard Alternatives**

1. **#1: Stall until branch direction is clear**
   - Execute successor instructions in sequence
   - "Squash" instructions in pipeline if branch actually taken
   - Advantage of late pipeline state update
   - 47% MIPS branches not taken on average
   - PC+4 already calculated, so use it to get next instruction

2. **#2: Predict Branch Not Taken**
   - 53% MIPS branches taken on average
   - But haven't calculated branch target address in MIPS
     - MIPS still incurs 1 cycle branch penalty
     - Other machines: branch target known before outcome

3. **#3: Predict Branch Taken**
   - 53% MIPS branches taken on average
   - “But haven't calculated branch target address in MIPS
     - MIPS still incurs 1 cycle branch penalty
     - Other machines: branch target known before outcome

4. **#4: Delayed Branch**
   - Define branch to take place AFTER a following instruction
   - Branch delay of length n
     - 1 slot delay allows proper decision and branch target address in 5 stage pipeline
     - MIPS uses this

**Scheduling Branch Delay Slots (Fig A.14)**

- A. From before branch
- B. From branch target
- C. From fall through

- add $1,$2,$3
- if $2=0 then
- delay slot
- sub $4,$5,$6
- if $1=0 then
- delay slot
- becomes

- add $1,$2,$3
- if $2=0 then
- sub $4,$5,$6
- if $1=0 then
- becomes

- add $1,$2,$3
- if $2=0 then
- add $1,$2,$3
- if $1=0 then
- sub $4,$5,$6
- becomes

- A is the best choice, fills delay slot & reduces instruction count (IC)
- In B, the sub instruction may need to be copied, increasing IC
- In B and C, must be okay to execute sub when branch fails
**Delayed Branch**

- Compiler effectiveness for single branch delay slot:
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled
- Delayed Branch downside: As processor go to deeper pipelines and multiple issue, the branch delay grows and need more than one delay slot
  - Delayed branching has lost popularity compared to more expensive but more flexible dynamic approaches
  - Growth in available transistors has made dynamic approaches relatively cheaper

**Evaluating Branch Alternatives**

<table>
<thead>
<tr>
<th>Scheduling scheme</th>
<th>Branch penalty</th>
<th>CPI speedup v. unpipelined</th>
<th>speedup v. stall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stall pipeline</td>
<td>3</td>
<td>1.60</td>
<td>3.1</td>
</tr>
<tr>
<td>Predict taken</td>
<td>1</td>
<td>1.20</td>
<td>4.2</td>
</tr>
<tr>
<td>Predict not taken</td>
<td>1</td>
<td>1.14</td>
<td>4.4</td>
</tr>
<tr>
<td>Delayed branch</td>
<td>0.5</td>
<td>1.10</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Pipeline speedup = \( \frac{1 + \text{Branch frequency} \times \text{Branch penalty}}{\text{Pipeline depth}} \)

Assume 4% unconditional branch, 6% conditional branch-untaken, 10% conditional branch-taken

**Problems with Pipelining**

- **Exception:** An unusual event happens to an instruction during its execution
  - Examples: divide by zero, undefined opcode
- **Interrupt:** Hardware signal to switch the processor to a new instruction stream
  - Example: a sound card interrupts when it needs more audio output samples (an audio “click” happens if it is left waiting)
- **Problem:** It must appear that the exception or interrupt must appear between 2 instructions (\( I_i \) and \( I_{i+1} \))
  - The effect of all instructions up to and including \( I_i \) is totalling complete
  - No effect of any instruction after \( I_i \) can take place
  - The interrupt (exception) handler either aborts program or restarts at instruction \( I_{i+1} \)

**Precise Exceptions in Static Pipelines**

Key observation: architected state only change in memory and register write stages.
And In Conclusion: Control and Pipelining

- Quantify and summarize performance
  - Ratios, Geometric Mean, Multiplicative Standard Deviation
- F&P: Benchmarks age, disks fail, 1 point fail danger
- Next time: Read Appendix A, record bugs online!
- Control VIA State Machines and Microprogramming
- Just overlap tasks; easy if tasks are independent
- Speed Up ≤ Pipeline Depth; if ideal CPI is 1, then:

\[
\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall} \times \text{CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}
\]

- Hazards limit performance on computers:
  - Structural: need more HW resources
  - Data (RAW, WAR, WAW): need forwarding, compiler scheduling
  - Control: delayed branch, prediction
- Exceptions, Interrupts add complexity
- Next time: Read Appendix C, record bugs online!