Notes:

- !!!!CHECK CLASS WEBPAGE!!!!!
  - Webboard up, TAs still trying to log in
  - Notes on Lab issues
- Quiz on Thursday!, New Homework up!
- A note on the on-line lecture system -- research
- Questions?

- Wrapping up chapter 2, Chapter 4, then 3

---

Computer Science Faculty/Student Lunch!

- Where and When
  - Fri January 14th
  - Sequoia room sierra summit
  - 12 noon
- Meet your faculty
- Talk to other CS majors
- NETWORK!
  - Jobs
  - Graduate school or summer research letters
Graduate School

- Master’s or PhD
- GOOD grades
- GRE (like the SAT)
- Letters of recommendation from faculty
- Research Experience
  - UCSD
  - National Programs (NSF REU (research experience for undergraduates), CRAW DMP, ATT, IBM, NASA)
- Timeline:
  - Research in junior year or summer
  - Apply in fall of senior year

Branch and Jump Addressing Modes

- Branch (e.g., beq) uses PC-relative addressing mode (uses few bits if address typically close). That is, target is PC+displacement mode.
  - If opcode is 6 bits, how many bits are available for displacement? How far can you jump?
Memory, alignment, jump addressing

Data value:
1111111000000011111010000100
### Key Points

- **MIPS** is a general-purpose register, load-store, fixed-instruction-length architecture.
- **MIPS** is optimized for fast pipelined performance, not for low instruction count.
- **Four principles** of IS architecture
  - regularity produces simplicity
  - smaller is faster
  - good design demands compromise
  - make the common case fast
Measuring Performance:
Chapter 4!

Or
My computer is faster than your computer...
with thanks to Larry Carter, UCSD

Performance Marches On...

But what is performance?
### Time versus throughput

<table>
<thead>
<tr>
<th>Vehicle</th>
<th>Time to Bay Area</th>
<th>Speed</th>
<th>Passengers</th>
<th>(pm/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferrari</td>
<td>3.1 hours</td>
<td>160 mph</td>
<td>2</td>
<td>320</td>
</tr>
<tr>
<td>Greyhound</td>
<td>7.7 hours</td>
<td>65 mph</td>
<td>60</td>
<td>3900</td>
</tr>
</tbody>
</table>

° Time to do the task from start to finish
  - execution time, response time,

° Tasks per unit time
  - throughput,

### Time versus throughput

- Execution Time is measured in time units/job.
  - For a SINGLE PROGRAM to execute on a system, usually in a dedicated environment

- **Throughput** is measured in jobs/time unit.
  - Total amount of work (multiple jobs) done by a computer for a given amount of time.

- But “time = 1/throughput” may be false.
  - It takes 4 months to grow a tomato.
  - Can you only grow 3 tomatoes a year??
How do you measure Execution Time?

> time foo
... foo’s results ...
90.7u 12.9s 2:39 65%

- user CPU time? (time CPU spends running your code)
- total CPU time (user + kernel)? (includes op. sys. code)
- Wallclock time? (total elapsed time)
  - Includes time spent waiting for I/O, other users, ...
- Answer depends ...
  - On what you are interested in evaluating!

A brief study of time

CPU Time = #CPU cycles executed * Cycle time

**CYCLE TIME:**

- Every conventional processor has a clock with a fixed cycle time often expressed as a clock rate
  --Rate often measured in GHz = millions of cycles/second “I’ve got a 2 GHz machine”
  --Time often measured in ns (nanoseconds)

**CYCLE TIME = \[= \frac{1}{\text{CLOCK RATE}}\]**
**Scientific Prefixes:**

1. $10^{-24}$ (Y) yotta (Greek or Latin octo, “eight”)
2. $10^{-21}$ (Z) zetta (Latin septem, “seven”)
3. $10^{-18}$ (E) exa (Greek hex, “six”)
4. $10^{-15}$ (P) peta (Greek pente, “five”)
5. $10^{-12}$ (T) tera (Greek teras, “monster”)
6. $10^{-9}$ (G) giga (Greek gigas, “giant”)
7. $10^{-6}$ (M) mega (Greek megas, “large”)
8. $10^{-3}$ (k) kilo (Greek chilioi, “thousand”)
9. $10^{-2}$ (h) hecto (Greek hekaton, “hundred”)
10. $10^{-1}$ (da) deka or deca (Greek deka, “ten”)
11. $10^{-1}$ (d) deci (Latin decimus, “tenth”)
12. $10^{-2}$ (c) centi (Latin centum, “hundred”)
13. $10^{-3}$ (m) milli (Latin mille, “thousand”)
14. $10^{-6}$ (mu) micro (Latin micro or Greek mikros, “small”)
15. $10^{-9}$ (n) nano (Latin nanus or Greek nanos, “dwarf”)
16. $10^{-12}$ (p) pico (Spanish pico, “a bit” or Italian piccolo, “small”)
17. $10^{-15}$ (f) femto (Danish-Norwegian femten, “fifteen”)
18. $10^{-18}$ (a) atto (Danish-Norwegian atten, “eighteen”)
19. $10^{-21}$ (z) zepto (Latin septem, “seven”) $10^{-24}$ (y) yocto (Greek or Latin octo, “eight”)

---

**That’s half, the other half…**

CPU Time = \#CPU cycles executed * Cycle time

\#CPU cycles = Instructions executed * CPI

$\text{Average Clock Cycles per Instruction}$

Different codes compile into different numbers of instructions.  

<table>
<thead>
<tr>
<th>COMP50 Iteration for loop</th>
<th>Windows OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>5 billion</td>
</tr>
</tbody>
</table>

Each computer design takes a certain amount of time to execute an “average” instruction
Putting it all together

One of P&H’s “big pictures”

CPU Execution Time = Instruction Count × CPI × Clock Cycle Time

Note: average CPI is somewhat artificial but it’s an intuitive and useful concept.
Note: Use dynamic instruction count (#instructions executed), not static (#instructions in compiled code)

Dynamic Instruction Count versus Static Instruction Count

int x = 10;
for (int j = 0; j<x; j++)
{
    c[j] = a[j]+b[j];
}

Static IC:
Dynamic IC:
What if x is input?

- Static instruction count is determined by the code and the compiler
- Dynamic instruction count is determined by the “choices” made in the execution of the code
  - A video game doesn’t have the same execution time each run...
Practice! \[ ET = IC \times CPI \times CT \]

- A program runs in 100 sec on a 1 GHz machine
  - How many cycles does it take?

- Same program runs in 75 sec on a 600 MHz machine
  - How many cycles does it take?

- How can this possibly be true?

---

Finding “Average” CPI

- Instruction classes
  - Each take different cycle count
    - Integer operations
    - Floating Point Operations
    - Loads/Stores
    - Multimedia Operations?
  - Can say that “on average” X% of insts from a given class

<table>
<thead>
<tr>
<th>type</th>
<th>cycle time</th>
<th>Int</th>
<th>FP</th>
<th>MEM</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI</td>
<td></td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

CPI =
Why “Average” CPI fails

- Consider 2 machines with the same clock rate:
  - BigBlue
    - Int 1; FP 4; Mem 2; MM 5
  - SuperVid
    - Int 2; FP 10; Mem 60; MM 1
- Consider 2 compilers for a particular code:
  - SuperSmart (50$)
    - Int: 10% FP 5% Mem 30% MM 55%
  - GenericSmart (free with machine)
    - Int 50% FP 5% Mem 45% MM 0%
- What is the CPI for each machine with each compiler?
- If you own Big Blue, should you buy the SuperSmart Compiler?
- What if you own SuperVid?

ET = IC * CPI * CT Wrapup

- “Real” CPI exists only:
  - For a particular program with a particular compiler
    - Perhaps a set of common applications (and input sets!)

- You MUST consider all 3 to get accurate ET estimations or machine speed comparisons
  - Instruction Set
  - Compiler
  - Implementation of Instruction Set (386 vs Pentium)
  - Processor Implementation (600 Mhz vs 1 GHz)
## Explaining Execution Time Variation

<table>
<thead>
<tr>
<th>Condition</th>
<th>Instruction Count</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine, different programs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same program, different machines, but same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same program, different ISA’s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}
\]

## Execution Time? Performance?

- We want higher numbers to be “better”

## Relative Performance

- “Computer X is r times faster than Y” or “speedup of X over Y”

\[
\frac{\text{Performance of X}}{\text{Performance of Y}} = r
\]
Quick Practice

- Your program runs in 5 minutes on a 1.8 GHz Pentium Pro and in 3 minutes on a 3.2 GHz Pentium 4. How much faster is it on the new machine?

- You get a new compiler for your Pentium 4 from “SmartGuysRUs” which changes the runtime of a different program from Q seconds to B seconds. How much faster is the new program?

How do we achieve increased performance? (Gene) Amdahl's Law

- The impact of an improvement is limited by the fraction of time affected by the improvement.
  - If you make MMX instructions run 10 times as fast, a program which doesn’t use MMX instructions will not run faster.

\[ ET_{\text{new}} = ET_{\text{old affected/amount of improve}} + ET_{\text{old unaffected}} \]
Amdahl’s Law Practice

• Protein String Matching Code
  - 4 days ET on current machine, spends 20% of time doing integer instructions
  - How much faster must you make the integer unit to make the code run 8 hours faster?

Amdahl’s Law Practice

• Protein String Matching Code
  - 4 days ET on current machine
    • 20% of time doing integer instructions
    • 35% percent of time doing I/O
  - Which is the better economic tradeoff?
    • Compiler optimization that reduces number of integer instructions by 25% (assume each integer inst takes the same amount of time)
    • Hardware optimization that makes I/O run 20% faster?
Think for a minute

- “Insanity is doing the same thing and expecting a different result”
  - Albert Einstein

Amdahl’s Law: Last Words

- Corollary for Processor Design:
  - Make the common case fast!
  - Whatever you think the computer will spend the most time doing, spend the most money and the most time making THAT run fast!

- Really: Parallel Processing
  - Only some parts of program can run in parallel
  - Speedup available by running “in parallel” proportional to amount of parallel work available

\[
\text{Speedup}_{\text{max}} = \frac{1}{\text{Serial} + \frac{(1-\text{Serial})}{\text{#processors}}}
\]
Performance Beyond one Program

<table>
<thead>
<tr>
<th>Program</th>
<th>Computer A</th>
<th>Computer B</th>
<th>Computer C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Program 2</td>
<td>1000</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Total Time</td>
<td>1001</td>
<td>110</td>
<td>40</td>
</tr>
</tbody>
</table>

- Which machine is fastest?

How to summarize performance

- Arithmetic Mean
  \[ \frac{1}{n} \sum_{i=1}^{n} \text{Time}_i \]

- Weighted Arithmetic Mean
  \[ \sum_{i=1}^{n} \text{Time}_i \times \text{Weight}_i \quad (\text{Weights total to 1}) \]

- Harmonic Mean
  \[ \frac{n}{\sum_{i=1}^{n} \frac{1}{\text{Rate}_i}} \]

- Geometric Mean
  \[ \sqrt[n]{\prod_{i=1}^{n} \text{Execution Time Ratio}_i} \]
Performance Beyond one Program

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>W(1)</th>
<th>W(another)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>1</td>
<td>10</td>
<td>20</td>
<td>.5</td>
<td>.999</td>
</tr>
<tr>
<td>Program 2</td>
<td>1000</td>
<td>100</td>
<td>20</td>
<td>.5</td>
<td>.001</td>
</tr>
<tr>
<td>AM: W(1)</td>
<td>500</td>
<td>55</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AM: W(ather)</td>
<td>2</td>
<td>10</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM</td>
<td>31.6</td>
<td>31.6</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summarizing Performance

- Even an “unweighted” arithmetic mean IS weighted
  - The longer the running time, the greater the impact of that code on the mean
- Geometric means of normalized execution times are consistent no matter which machine is faster
  - Ratios of geometric means always give equal weights to all benchmarks - no matter execution times
- Geometric mean does not necessarily prediction execution time for any mix of the programs
Another way of “measuring” performance:

- It’s hard to convince manufacturers to run your program (unless you’re a BIG customer)
- A benchmark is a set of programs that are representative of a class of problems.
  - measure one feature of system
    - e.g. memory accesses or communication speed
  - most compute-intensive part of applications
    - e.g. Linpack and NAS kernel b'marks (for supercomputers)
  - Full application:
    - (int and float) (for Unix workstations)
    - Other suites for databases, web servers, graphics,...

SPEC89 and the compiler

Darker bars show performance with compiler improvements (same machine as light bars)
SPEC on Pentium III and Pentium 4

• What do you notice?

Other SPECs

• HPC (High Performance Computing)
  - Quantum Chemistry, Weather Modeling, Seismic
• JVM (Java)
• JAppletServer
• Web
• Mail
• JBB Java Business Benchmark
• SFS System File Server
  Test many things other than the CPU speed - test entire system performance
Performance Beyond the CPU

- We (and this book) concentrate on the CPU as a lone entity
  - For a while (Chap 7,8,9)
- Memory: A very important part
  - The CPU can only do work if it has data to work on
  - Latency and Bandwidth were our metrics
- Due to modern processor design, improving speed of integer operations by 10% will (likely) NOT speed up ANYTHING!

Key Points

- Be careful how you specify performance
  - Use times faster, practice!
- Execution time = instructions * CPI * cycle time
- Make the common case FAST!
  - Amdahl’s Law
- Use real applications to measure performance
  - Make sure their workload represents the one you care about!
- Use geometric mean to report performance on suites of programs or benchmarks