Lecture 2: Performance, MIPS ISA

• Today’s topics:
  ▪ Performance equations
  ▪ MIPS instructions

• Reminder: canvas and class webpage:
  http://www.cs.utah.edu/~rajeev/cs3810/

• Reminder: sign up for the mailing list csece3810

• See info on TA office hours on class webpage

• From your classmate, Jeremy: www.UteSwap.com
Performance Metrics

• Possible measures:
  ▪ response time – time elapsed between start and end of a program
  ▪ throughput – amount of work done in a fixed time

• The two measures are usually linked
  ▪ A faster processor will improve both
  ▪ More processors will likely only improve throughput
  ▪ Some policies will improve throughput and worsen response time

• What influences performance?
Execution Time

Consider a system X executing a fixed workload W

\[ \text{Performance}_X = \frac{1}{\text{Execution time}_X} \]

Execution time = response time = wall clock time
- Note that this includes time to execute the workload as well as time spent by the operating system co-ordinating various events

The UNIX “time” command breaks up the wall clock time as user and system time
Speedup and Improvement

• System X executes a program in 10 seconds, system Y executes the same program in 15 seconds

• System X is 1.5 times faster than system Y

• The speedup of system X over system Y is 1.5 (the ratio)

• The performance improvement of X over Y is 
  \[ \frac{1.5 - 1}{1.5} = 0.5 = 50\% \]

• The execution time reduction for the program, compared to Y is 
  \[ \frac{15 - 10}{15} = 33\% \]
  The execution time increase, compared to X is 
  \[ \frac{15 - 10}{10} = 50\% \]
A Primer on Clocks and Cycles
Performance Equation - I

CPU execution time = CPU clock cycles × Clock cycle time
Clock cycle time = 1 / Clock speed

If a processor has a frequency of 3 GHz, the clock ticks 3 billion times in a second – as we’ll soon see, with each clock tick, one or more/less instructions may complete

If a program runs for 10 seconds on a 3 GHz processor, how many clock cycles did it run for?

If a program runs for 2 billion clock cycles on a 1.5 GHz processor, what is the execution time in seconds?
Performance Equation - II

CPU clock cycles = number of instrs x avg clock cycles per instruction (CPI)

Substituting in previous equation,

Execution time = clock cycle time x number of instrs x avg CPI

If a 2 GHz processor graduates an instruction every third cycle, how many instructions are there in a program that runs for 10 seconds?
Factors Influencing Performance

Execution time = clock cycle time x number of instrs x avg CPI

- Clock cycle time: manufacturing process (how fast is each transistor), how much work gets done in each pipeline stage (more on this later)

- Number of instrs: the quality of the compiler and the instruction set architecture

- CPI: the nature of each instruction and the quality of the architecture implementation
Example

Execution time = clock cycle time x number of instrs x avg CPI

Which of the following two systems is better?

• A program is converted into 4 billion MIPS instructions by a compiler; the MIPS processor is implemented such that each instruction completes in an average of 1.5 cycles and the clock speed is 1 GHz

• The same program is converted into 2 billion x86 instructions; the x86 processor is implemented such that each instruction completes in an average of 6 cycles and the clock speed is 1.5 GHz
Benchmark Suites

- Each vendor announces a SPEC rating for their system
  - a measure of execution time for a fixed collection of programs
  - is a function of a specific CPU, memory system, IO system, operating system, compiler
  - enables easy comparison of different systems

The key is coming up with a collection of relevant programs
SPEC CPU

• SPEC: System Performance Evaluation Corporation, an industry consortium that creates a collection of relevant programs

• The 2006 version includes 12 integer and 17 floating-point applications

• The SPEC rating specifies how much faster a system is, compared to a baseline machine – a system with SPEC rating 600 is 1.5 times faster than a system with SPEC rating 400

• Note that this rating incorporates the behavior of all 29 programs – this may not necessarily predict performance for your favorite program!
Deriving a Single Performance Number

How is the performance of 29 different apps compressed into a single performance number?

• SPEC uses geometric mean (GM) – the execution time of each program is multiplied and the N\textsuperscript{th} root is derived

• Another popular metric is arithmetic mean (AM) – the average of each program’s execution time

• Weighted arithmetic mean – the execution times of some programs are weighted to balance priorities
Amdahl’s Law

• Architecture design is very bottleneck-driven – make the common case fast, do not waste resources on a component that has little impact on overall performance/power

• Amdahl’s Law: performance improvements through an enhancement is limited by the fraction of time the enhancement comes into play

• Example: a web server spends 40% of time in the CPU and 60% of time doing I/O – a new processor that is ten times faster results in a 36% reduction in execution time (speedup of 1.56) – Amdahl’s Law states that maximum execution time reduction is 40% (max speedup of 1.66)
Common Principles

- Amdahl’s Law

- Energy: systems leak energy even when idle

- Energy: performance improvements typically also result in energy improvements

- 90-10 rule: 10% of the program accounts for 90% of execution time

- Principle of locality: the same data/code will be used again (temporal locality), nearby data/code will be touched next (spatial locality)
Recap

• Knowledge of hardware improves software quality: compilers, OS, threaded programs, memory management

• Important trends: growing transistors, move to multi-core, slowing rate of performance improvement, power/thermal constraints, long memory/disk latencies

• Reasoning about performance: clock speeds, CPI, benchmark suites, performance equations

• Next: assembly instructions
Instruction Set

• Understanding the language of the hardware is key to understanding the hardware/software interface

• A program (in say, C) is compiled into an executable that is composed of machine instructions – this executable must also run on future machines – for example, each Intel processor reads in the same x86 instructions, but each processor handles instructions differently

• Java programs are converted into portable bytecode that is converted into machine instructions during execution (just-in-time compilation)

• What are important design principles when defining the instruction set architecture (ISA)?
Instruction Set

• Important design principles when defining the instruction set architecture (ISA):
   keep the hardware simple – the chip must only implement basic primitives and run fast
   keep the instructions regular – simplifies the decoding/scheduling of instructions
A Basic MIPS Instruction

C code:                                  a = b + c;

Assembly code: (human-friendly machine instructions)
                add    a, b, c  # a is the sum of b and c

Machine code: (hardware-friendly machine instructions)
                00000010001100100100000000100000

Translate the following C code into assembly code:
                a = b + c + d + e;
Example

C code    a = b + c + d + e;
translates into the following assembly code:

    add  a, b, c
    add  a, a, d         or        add  f, d, e
    add  a, a, e       add  a, a, f

• Instructions are simple: fixed number of operands (unlike C)
• A single line of C code is converted into multiple lines of assembly code
• Some sequences are better than others… the second sequence needs one more (temporary) variable f
Subtract Example

```
C code    f = (g + h) – (i + j);

Assembly code translation with only add and sub instructions:
```
C code \[ f = (g + h) - (i + j); \]

translates into the following assembly code:

- `add t0, g, h`
- `add f, g, h`
- `add t1, i, j`
- `or sub f, f, i`
- `sub f, t0, t1`
- `sub f, f, j`

- Each version may produce a different result because floating-point operations are not necessarily associative and commutative... more on this later
Title

- Bullet