Computer Organization & Architecture

Assessment of computer performance

- Key parameters:
  - Performance, cost, size, security, reliability, power consumption
- Application performance depends on
  - Raw speed of the processor
  - Instruction set
  - Choice of implementation language
  - Efficiency of the compiler
  - Skill of the programming done to implement the application

Clock Speed

- Operations performed by processor governed by system clock
  - Typically, all operations begin with the pulse of the clock
  - Speed of processor dictated by pulse frequency produced by clock
  - Measured in cycles per second, or Hertz (Hz)
- Typically, clock signals are generated by a quartz crystal
  - Generates a constant sine wave while power is applied
  - Wave is converted into a digital voltage pulse stream
  - Ex., a 1-GHz processor receives 1 billion pulses per second

Clock Speed

- Rate of pulses is known as the **clock rate**, or **clock speed**
- One increment, or pulse, of clock is referred to **clock cycle**
- Time between pulses is the **cycle time**

Instruction Execution Rate

- Processor is driven by clock with a constant frequency *f*
  - Equivalently, a constant cycle time *t*, where $t = 1/f$
- **Instruction Count**, I_c: number of machine instructions executed
  - Number of instruction executions, not number of instructions in the object code of the program
- Average **cycles per instruction** (CPI):
  - Average number of clock cycles per instruction

Instruction Execution Rate

- **Execution time**, *T*:

  $$T = I_c \times CPI \times t = I_c \times CPI / f$$

  Referred to as the **basic performance equation**

- **Instruction throughput**: number of instructions executed / second

  $$P = f / CPI$$

Instruction Execution Rate

- Average **cycles per instruction** (CPI):
  - If all instructions required same number of clock cycles, then CPI would be a constant value for a processor
  - Number of clock cycles required varies for different types of instructions
- Let CPI_i be the number of cycles required for instruction type *i*, and I_i be the number of executed instructions of type *i* for a given program.
- Overall CPI:

$$\text{CPI} = (\sum CPI_i * I_i) / I_c$$

MIPS

- MIPS: millions of instructions per second
  - Rate at which instructions are executed

$$\text{MIPS rate} = I_c / (T * 10^6) = f / (CPI * 10^6)$$

Example

- Consider the execution of a program that results in the execution of 2 million instructions on a 400-MHz processor. The program consists of four major types of instructions. The instruction mix and the CPI for each instruction type are given below:

Instruction Type	CPI	Instruction Mix (%)
Arithmetic and logic	1	60
Load/store with cache hit	2	18
Branch	4	12
Memory reference with cache miss	8	10

Example

- CPI = 2.24
- MIPS rate ≈ 178

MFLOPS

- Millions of floating-point operations per second (MFLOPS)
- Performance measure with floating-point instructions.
  - Common in many scientific and game applications

MFLOPS rate = (Number of executed floating point operations in a program) / (Execution time * 10^6)

Benchmark Principles

- Measures such as MIPS and MFLOPS
  - Inadequate to evaluating the performance of processors
- Because of differences in instruction sets
  - Instruction execution rate is not a valid means of comparing performance of different architectures
- Performance of a given processor on a given program:
  - May not be useful in determining how that processor will perform on a very different type of application

Benchmark Principles

- Measure the performance of systems using a set of benchmark programs
  - Same set of programs run on different machines and execution times compared
- Benchmarks provide guidance to customers trying to decide which system to buy
  - Useful to vendors and designers in determining how to design systems to meet benchmark goals

Benchmark Principles

- [WEIC90] lists following as desirable characteristics of a benchmark program:
  - Written in a high-level language, making it portable across different machines
  - Representative of a particular kind of programming domain or paradigm, such as systems programming, numerical programming, or commercial programming
  - Can be measured easily
  - Has wide distribution

Weicker, R. “An Overview of Common Benchmarks.” Computer, December 1990

Benchmark Principles

- Generally accepted computer performance measurements
  - Development of standardized benchmark suites
- A benchmark suite is a collection of programs
  - Defined in a high-level language
  - Together attempt to provide a representative test of a computer in a particular application or system programming area

SPEC Benchmarks

- Standard Performance Evaluation Corporation (SPEC):
  - An industry consortium
  - Defines and maintains several benchmark suites aimed at evaluating computer systems
  - SPEC performance measurements are widely used for comparison and research purposes
- Best known of the SPEC benchmark suites is SPEC CPU2006
  - Industry standard suite for processor-intensive applications
  - Appropriate for applications that spend most of their time doing computation rather than I/O

Calculating the Mean

- Use of benchmarks to compare systems involves calculating the mean value of a set of data points related to execution time
  - Multiple alternative algorithms that can be used for calculating a mean value
  - Three common formulas are arithmetic, geometric, and harmonic

Calculating the Mean

- Given a set of n real numbers (x_1, x_2, …, x_n), the three means are defined as follows:

$$AM = (1/n)^* \sum x_i$$

$$GM = (\prod x_i)^{1/n}$$

$$HM = n / \sum(1 / x_i)$$

Amdahl's Law

- Deals with the potential speedup of a program using multiple processors compared to a single processor
- Consider a program running on a single processor such that:
  - A fraction (1 - *f*) of the execution time involves code that is inherently sequential
  - A fraction *f* that involves code that is infinitely parallelizable with no scheduling overhead

Amdahl's Law

- Let *T* be the total execution time of program using a single processor
- Speedup using a parallel processor with *N* processors that fully exploits the parallel portion of the program:

  Speedup = Time to execute program on a single processor / Time to execute program on *N* parallel processors

  $= (Tf + T(1-f)) / (T(1-f) + Tf/N))$

  $= 1 / ((1-f) + f/N)$

Amdahl's Law

Speedup = $1 / ((1-f) + f/N)$

- Two important conclusions can be drawn:
  - When *f* is small, the use of parallel processors has little effect
  - As *N* approaches infinity, speedup is bound by $1/(1 - f)$, so that there are diminishing returns for using more processors

Amdahl's law

- Amdahl's law can be generalized to evaluate any design or technical improvement in a computer system.
- Consider any enhancement to a feature of a system that results in a speedup.
- Speedup = Performance after enhancement / Performance before enhancement

  = Execution time before enhancement / Execution time after enhancement

Amdahl's law

- Ex., Suppose that a task makes extensive use of floating-point operations, with 40% of the time consumed by floating-point operations. With a new hardware design, the floating-point module is sped up by a factor of K.

  Speedup = 1 / (0.6 + 0.4/K)

Thus, independent of K, the maximum speedup is 1.67

END