CSE 120
Day 3 Notes

Elijah Hantman

Performance

- Latency

  The interval between stimulation and response.

  What is the minimum amount of time to do a single task.

  Lower Latency is better.

  example unit is ms/request.

  > Digital HW operates using a constant clock.
  >
  > Clock period is the duration of a clock cycle.
  >
  > Clock frequency is the number of cycles per second.
  >
  > example calculation
  > $4.0G$Hz$= 4 \times 10^9$Hz
  > Period $= 1/4.0 \times 10^{-9}$s

- Throughput or bandwidth

  The maximum number of tasks completable in a given timeframe.

  Higher throughput is better.

  example unit is requests/sec

- Latency is most affected by clockspeed, Throughput is most affected by parrallelism and clockspeed.

  Throughput is at least as fast as the latency but due to overlapping and parallelism it can be much faster.

- Execution Time = Cycles Per Program / Clock Rate

  Execution can be improved by reducing clock cycles or increasing clock rate.

  Often have to trade off between clock rate and cycle count.

  More complicated instructions often restrict clock rates.

  Execution time is equivalent to latency.

  Execution time is also equal to Cycles * Clock Period

  > Example
  >
  > Computer A: 2GHz clock, 10s CPU time
  >
  > Designing Computer B
  >
  > - Aim for 6s CPU time
  > - Can do faster clock but causes 1.2 times clock cycles
  >
  > $6 = 1.2c/rate$ $10 = c/2$ $c = 20$ $6 = 24/rate$ $rate = 4$
  >
  > Computer B must be run at 4GHz

- Relative Performance Speedup $S = \frac{T_{old}}{T_{new}}$

  Prefer to state speedup in terms of multiples rather than percent.

  2.1 times speedup is more clear than 110% speedup

Performance is the inverse of the execution time.

Bandwidth is measured in Instructions per cycle. Measured as a frequency, inverse of cycle time.

Ex: Chip Fab takes 1 month to complete a chip but can start on a chip every minute.

Latency = 1 month per chip
Throughput = 1 chip per minute
Inverse Throughput = 1 minute per chip

Throughput is only sometimes the inverse of latency. Multiprocessing, overlapping instructions, SIMD, MIMD, etc. can all increase Throughput without decreasing latency.

- Little's Law
  Parallelism = $Throughput \times Latency$
  Throughput = $\frac{Parallelism}{Latency}$

  Sandwich, how to get 240 per hour We need 4 sandwiches per minute Parallelism = $4 * 2$ We then need 8 people to make 240 sandwiches per hour

  Little's law only works with Spatial Parallelism

  If the Workers are not completely independent Little's Law no longer applies, if you have an assembly line then Little's law also no longer works.

  Workers in series is also known as Temporal Parallelism

- Relative Speedup for Multiple Cores

  If you are using n processors, your Speedup is
  $S = \frac{T_1}{T_n}$
  Where T denotes time, and the subscript denotes the number of cores. This definition means the speedup is alwasy greater than or equal to 1.

  Your Speedup Efficiency is
  $E = \frac{S_n}{n}$
  This measures how much you benefit from parallelism.

  The Speedup Efficiency is usually less than 1, this happens because there are tasks which cannot be split between multiple workers, such as initialization, coordination, gathering results back together, and other problem specific tasks.

- Amdahl's Law

  As you add more cores you can approach, but not beat the time taken for the non-parallelizable part of the code.

  $S = \frac{1}{\frac{Fparallel}{n} + F_sequential}$

  Generalized Amdahl's Law, which applies to all resources

  $Speedup = \frac{1}{f_x/S_x + (1-f_x)}$ Where f_x is the fraction of a program speed up by x, and S_x is the factor by which x has sped up the program.

  What is the speedup if half the execution is speed up by 2x?

  The speedup is $\frac{4}{3}$ or 1.3

  If you eliminate half the execution, what is the overal speedup?

  The speedup is 2.

- CSE12 Basics

  Digital Systems are fairly simple

Computers work with 1s or 0s. More complex things are encoded in multiple bits.

1s and 0s are stored via SRAM, DRAM, SSDs, HDDs, etc.

1s and 0s are processed using Logic gates AND, OR, NOT, XOR, Multiplexors, DeMultiplexors, etc.

Each instruction will switch on and off various components to load, store, and change various bits.

High level programs go through compilers to be turned into Assembly. Assembly is then run through an Assembler to convert it into 1s and 0s.

Then each instruction from a binary executable is moved one by one into the CPU and executed.

- Instruction Count and CPI

  IC is the instruction count for a program. This is determined by the program, ISA, and compiler.

  Average cycles per instruction (CPI) is determined by Hardware design, and depends on the program's ratios of different instructions.

  $Cycles = IC * CPI$

- Iron Law

  $Execution = \frac{Instruction}{Program} * \frac{Cycles}{Instruction} * \frac{Seconds}{Cycle}$