CSCI 341 Computer Organization
– CPU Performance

Time Discovers Truth.

Seneca

Performance

- **Purchasing perspective**
  - given a collection of machines (or upgrade options)
- **Computer designer perspective**
  - faced with design options
- **Factors to consider**
  - best performance improvement ?
  - least cost ?
  - best performance / cost ?
- **All require basis for comparison and metric for evaluation**
  - solid metrics lead to solid progress!

Understanding Performance

- **Algorithm**
  - Determines number of operations executed
- **Programming language, compiler, architecture**
  - Determine number of machine instructions executed per operation
- **Processor and memory system**
  - Determine how fast instructions are executed
- **I/O system (including OS)**
  - Determines how fast I/O operations are executed

Two Notions of "Performance"

Plane	DC to Paris	Top Speed	Passen-gers	Throughput (pxmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

- Which has higher performance?
  - Time to deliver 1 passenger or 400 passengers?
- Time for 1 job called

  Response Time or Execution Time
- Jobs per day or in unit-time called

  Throughput or Bandwidth

Response Time vs. Throughput

- **Time of Concorde vs. Boeing 747?**
  - **Concord is 6.5 hours / 3 hours = 2.2**
  - **Concord is 2.2 times Boeing (i.e., "120%" slower) in terms of flying time (response time)**
- **Throughput of Boeing vs. Concorde?**
  - **Boeing 747: 286,700 pmph / 178,200 pmph = 1.6 times**
  - **Boeing is 1.6 times Concord (i.e., "60%" better) in terms of throughput**

Definition of Performance

- **If primarily concerned with response time (our current focus)**

  - performance(x) = 1 / execution_time(x)

" F(ast) is *n* times as fast as S(low) " means…

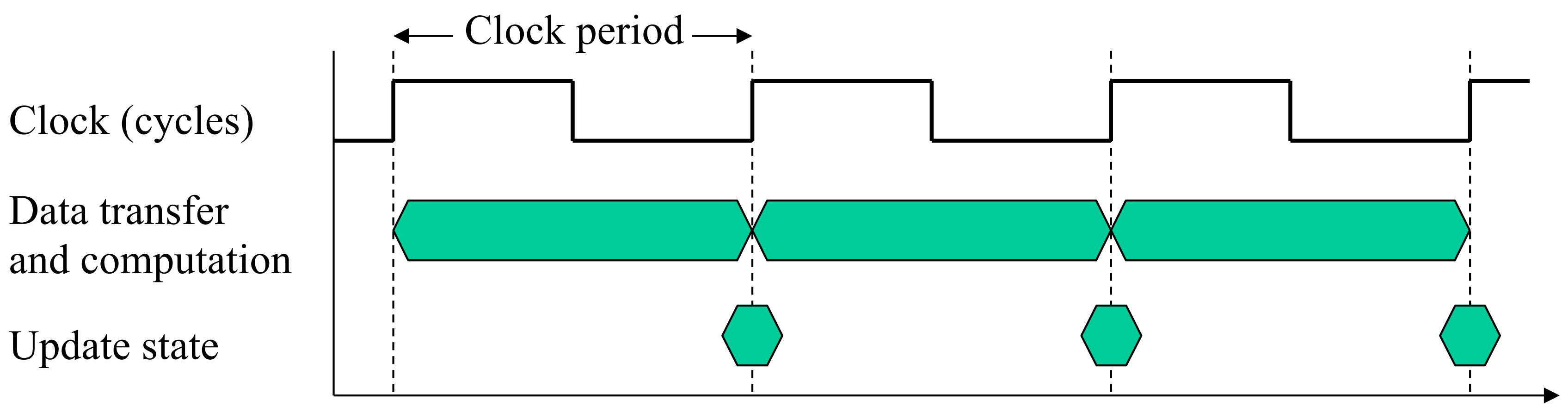
$$n = \frac{\text{performance}(F)}{\text{performance}(S)} = \frac{\text{execution\_time}(S)}{\text{execution\_time}(F)}$$

What is Time?

- **Straightforward definition:**
  - Total time to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, ...
  - “wall-clock time”, “response time” or “elapsed time”
- **Alternative:**
  - just time processor (CPU) is working only on your program (since multiple processes running at same time)
  - “CPU execution time” or “CPU time”
    - =system CPU time (in OS) + user CPU time (in user program)

CPU Clocking

- **Operation of digital hardware governed by a constant-rate clock**



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

How to Measure Time?

- **User Time $\Rightarrow$ seconds**
- **Clock cycles (or informally clocks or cycles)**
  - Discrete time intervals that computers run and generate events in hardware
- **Clock cycle time: length of clock period**
  - e.g., 2 nanoseconds or 2 ns
- **Clock rate**
  - the inverse of the clock period
  - e.g., (1/2ns)=500 megahertz, or 500 MHz

Clock cycle time or clock period	**Clock rate (=1/clock_period)**
1 second	1Hz
1 ms (milli-second) = 10^{-3} s	1 x 10^{3} Hz = 1 KHz
1 µs (micro-second) = 10^{-6} s	1 x 10^{6} Hz = 1 MHz
1 ns (nano-second) = 10^{-9} s	1x 10^{9} Hz = 1 GHz
1 ps (pico-second) = 10^{-12} s	1 x 10^{12} Hz = 1 THz

Measuring Time using Clock Cycles

- CPU execution time for a program

  = Clock Cycles for a program x Clock Cycle Time

- Or

$$= \frac{\text{Clock Cycles for a program}}{\text{Clock Rate}}$$

Defining Clock Cycles

- **Clock Cycles for a program**

= Instructions for a program (i.e., "Instruction Count")

x Average Clock cycles Per Instruction (i.e., "CPI")

- **CPI**
  - Used to compare two machines with the same instruction set, since Instruction Count would be the same

Execution Time Calculation (1/2)

- **CPU execution time for a program**

  = Clock Cycles for a program

  x Clock Cycle Time

- **Substituting for clock cycles:**

  CPU execution time for program

  = (Instruction Count x CPI)

  x Clock Cycle Time

  = Instruction Count x CPI x Clock Cycle Time

Execution Time Calculation (2/2)

$$\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$\text{CPU time} = \frac{\cancel{\text{Instructions}}}{\text{Program}} \times \frac{\cancel{\text{Cycles}}}{\cancel{\text{Instruction}}} \times \frac{\text{Seconds}}{\cancel{\text{Cycle}}}$$

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}}$$

- **Product of all 3 terms**

Terms

Components of performance	Units of measure
CPU Execution time for a program (ET)	Seconds for a program
Instruction Count (IC)	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time (CCT)	Seconds per clock cycle

How to Calculate the 3 Components?

- **Clock Cycle Time:**
  - in specification of computer (Clock Rate in advertisements)
- **Instruction Count:**
  - Use software profiling tools
  - Use a simulator of the architecture
  - Use hardware counters included on many processors
- **CPI:**
  - Calculate: Execution Time / (Clock cycle time x Instruction Count)
  - Use hardware counters or a detailed simulation

Example: Calculating CPI

Operation	$Freq_i$	CPI_i	Product	%Time
ALU	50%	1	0.5	23%
Load	20%	5	1.0	45%
Store	10%	3	0.3	14%
Branch	20%	2	0.4	18%
		Average CPI=	2.2	

Calculating CPI

1. calculate CPI for each individual instruction (`add`, `sub`, `and`, etc.)

- calculate frequency of each individual instruction
- multiply these two for each instruction and add them up to get final CPI (the weighted sum)

Relative Performance

- **Will (try to) stick to “n time as fast”**
  - less confusing than “m % faster”
- **Faster: <u>increased</u> performance and <u>decreased</u> execution time,**
  - to reduce confusion, use “<u>improve performance</u>” or “<u>improve execution time</u>”
- **Speedup =big_execution_time/ small_execution_time**

Relative Performance

- **Define Performance = 1/Execution Time**
- **"X is n time as fast as Y"**

$$\text{Performance}_X / \text{Performance}_Y = \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times as fast as B

Pitfall

- **Improving an aspect of a computer and expecting a proportional improvement in overall performance**

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5 $\times$ overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!

Reality: Amdahl's Law

- **Performance enhancement possible with a given improvement is limited by the amount that the improved feature is used. It is a quantitative version of the law of the diminishing returns**
- **Example: if some aspect of a computer accounts for 50% of program execution time, what is the limit on how many times faster programs can run if engineers focus on improving that aspect?**

Answer: assume initially the program execution time is T, the improvement factor for that aspect is n, then

$T_{improved} = 0.5T/n + 0.5T$

As n goes to infinity, $T_{improved}$ approaches 0.5T, hence the answer is 2 times

Conclusion

- **Response time vs. Throughput**
- **Performance doesn’t depend on any single factor:**
  - Instruction Count, Clocks Per Instruction (CPI) and Clock Rate
- **User Time:**
  - time user needs to wait for program to execute
  - depends heavily on how OS switches between tasks
- **CPU Time:**
  - time spent executing a single program:
  - depends solely on design of processor (datapath, pipelining effectiveness, caches, etc.)

CPU time = Instructions/program x Cycles/instruction x Seconds/cycle

Example1

Suppose we have two implementations of the same instruction set architecture (ISA).

For some program,

Machine A has a clock cycle time of 10 ns. and a CPI of 2.0

Machine B has a clock cycle time of 20 ns. and a CPI of 1.2

1) If two machines have the same ISA which of our quantities (e.g., clock rate, CPI, execution time, # of instructions) will always be identical?

2) What machine is faster for this program, and by how much?

Answer to Example 1

1) Instruction count is identical

2)

For machine A

CPU time = IC × CPI × Clock cycle time

CPU time = IC × 2.0 × 10 ns = 20 IC ns

For machine B

CPU time = IC × 1.2 × 20 ns = 24 IC ns

Speedup = (24 IC) / (20 IC) = 1.2

So machine A is 1.2 times as fast as B

Example2

A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles respectively.

The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C

The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C.

What is the CPI for each sequence?

Which sequence will be faster? How much?

Answer to Example 2

$$\text{CPI for sequence } 1 = \frac{2\times1+1\times2+2\times3}{(2+1+2)} = \frac{10}{5}$$

$$\text{CPI for sequence } 2 = \frac{4\times1+1\times2+1\times3}{(4+1+1)} = \frac{9}{6}$$

$$\text{CPU cycles for sequence } 1 = 10/5\times5 = 10$$

$$\text{CPU cycles for sequence } 2 = 9/6\times6 = 9$$

Sequence 2 is 10/9 times as fast as Sequence 1

Example 3

- **The design team for a simple, single-issue processor is choosing between a pipelined or non-pipelined implementation. Here are some design parameters for the two possibilities:**

Parameter	Pipelined Version	Non-pipelined Version
Clock Rate	500 MHz	350 MHz
CPI for ALU instructions	1	1
CPI for branch instructions	2	1
CPI for memory access instructions	2.7	1

For a program with 20% ALU instructions, 10% branch instructions, and 70% memory access instructions which design is faster? What’s the average CPI for each design?

Answer to Example 3

- Average CPI for pipelined version
  - 0.2*1 + 0.1*2+0.7*2.7 = 2.29
- Average CPI for non-pipelined version
  - 0.2*1 + 0.1*1+0.7*1 = 1.0
- CPU execution time for pipelined version
  - 2.29/500M Hz = 4.5 ns
- CPU execution time for non-pipelined version
  - 1.0/350 MHz = 2.8 ns

The non-pipelined version is faster

Example 4

Computer A has an overall CPI of 1.3 and can be run at a clock rate of 600MHz. Computer B has a CPI of 2.5 and can be run at a clock rate of 750 Mhz. We have a particular program we wish to run. When compiled for computer A, this program has exactly 100,000 instructions. How many instructions would the program need to have when compiled for Computer B, in order for the two computers to have exactly the same execution time for this program?

Answer to Example 4

$(\text{CPUTime})_A = (\text{Instruction count})_A * (\text{CPI})_A * (\text{Clock cycle Time})_A$

$= (100{,}000)*(1.3)/(600*10^6)$ ns

$(\text{CPUTime})_B = (\text{Instruction count})_B * (\text{CPI})_B * (\text{Clock cycle Time})_B$

$= (I)_B*(2.5)/(750*10^6)$ ns

Since $(\text{CPUTime})_A = (\text{CPUTime})_{B,}$

we have to solve for $(I)_B$ and get 65000

Example 5

Assume that multiply instructions take 12 cycles and account for 15% of the instructions in a typical program, and the other 85% of the instruction require an average of 4 cycles for each instruction. What percentage of time does the CPU spend doing multiplication?

Answer to Example 5

The average CPI is 0.15*12 cycles/instruction + 0.85 * 4 cycles/instruction = 5.2 cycles/instructions, of which 0.15*12=1.8 cycles/instructions of that is due to multiplication instructions. This means that multiplications take up 1.8/5.2=34.6% of the CPU time.

Example 6

Assume that multiply instructions take 12 cycles and account for 15% of the instructions in a typical program, and the other 85% of the instruction require an average of 4 cycles for each instruction.

Your hardware engineering team has indicated that it would be possible to reduce the number of cycles required for multiplication to 8, but this will require a 20% increase in the cycle time. Nothing else will be affected by the change. Should they proceed with the modification?

Answer to Example 6

Reducing the CPI of multiplication instructions results in a new average CPI of 0.15*8+0.85*4=4.6. The clock rate will reduce by a factor of 5/6. So the new performance is (5.2/4.6)*(5/6)=26/27.6 times as good as the original. So the modification is detrimental and should not be made.

Example 7

- Consider program P, which runs on a 1GHz machine M in 10 seconds. An optimization is made to P, replacing all instances of multiplying a value by 4 (mult X, X, 4) with two instructions that set x to x+x twice (add X, X, X; add X, X, X). Call this new optimized program P'. The CPI of a multiply instruction is 4, and the CPI of an add is 1. After recompiling, the program now runs in 9 seconds on machine M. How many multiplies were replaced by the new compiler?

Answer to Example 7

- Program P running on machine M takes (10^9 cycles/seconds) * 10 seconds = 10^{10} cycles.
- P ′ takes (10^9 cycles/seconds) * 9 seconds = 9 × 10^9 cycles.
- This leaves 10^9 less cycles after the optimization.
- Every time we replace a mult with two adds, it takes

4 – 2 * 1 = 2 cycles less per replacement.

- Thus, there must have been 10^9 cycles /(2 cycles/replacement) = 5 × 10^8 replacements to make P into P ′ .

Summary

CPU execution time for a program

= Instruction Count x CPI x Clock Cycle Time

Amdahl's Law:

Performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.