Performance Issues

Execution Time

- *Elapsed Time*
  - counts everything (*disk and memory accesses, waiting for I/O, running other programs, etc.*) from start to finish
  - a useful number, but often not good for comparison purposes

  elapsed time = CPU time + wait time (I/O, other programs, etc.)

- *CPU time*
  - doesn't count waiting for I/O or time spent running other programs
  - can be divided into *user CPU time* and *system CPU time* (OS calls)

  CPU time = user CPU time + system CPU time

  $\Rightarrow$ elapsed time = user CPU time + system CPU time + wait time

- Our focus: *user CPU time* (*CPU execution time* or, simply, *execution time*)
  - time spent executing the lines of code that are *in our program*

Definition of Performance

- For some program running on machine X:

$$\text{Performance}_X = 1 \,/\, \text{Execution time}_X$$

- *X is n times faster than Y* means:

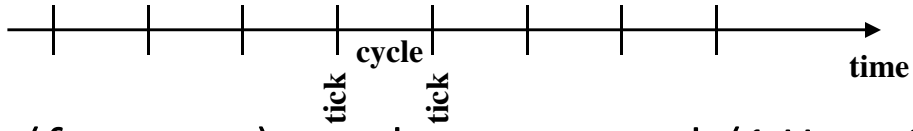
$$\text{Performance}_X \,/\, \text{Performance}_Y = n$$

Clock Cycles

- Instead of reporting execution time in seconds, we often use *cycles*. In modern computers hardware events progress cycle by cycle: in other words, each event, e.g., multiplication, addition, etc., is a sequence of cycles
- *Clock ticks* indicate start and end of cycles:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- *cycle time* = time between ticks = seconds per cycle



- *clock rate* (*frequency*) = cycles per second (1 Hz. = 1 cycle/sec, 1 MHz. = 10^6 cycles/sec)
- *Example*: A 200 Mhz clock has a cycle time

$$\frac{1}{200 \times 10^6} \times 10^9 = 5 \text{ nanoseconds}$$

Performance Equation I

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

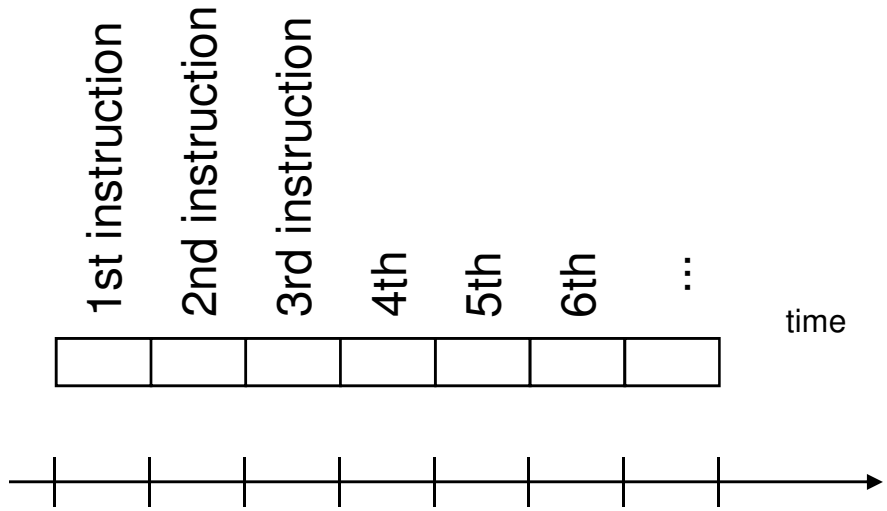
equivalently

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

- So, to improve performance one can either:
  - reduce the number of cycles for a program, or
  - reduce the clock cycle time, or, equivalently, increase the clock rate

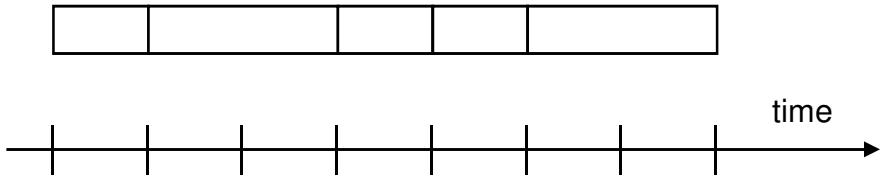
How many cycles are required for a program?

- Could assume that # of cycles = # of instructions



- *This assumption is incorrect!* Because:
  - Different instructions take different amounts of time (cycles)
  - Why…?

How many cycles are required for a program?



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- *Important point*: changing the cycle time often changes the number of cycles required for various instructions because it means changing the hardware design.

Terminology

- A given program will require:
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds
- We have a vocabulary that relates these quantities:
  - *cycle time* (seconds per cycle)
  - *clock rate* (cycles per second)
  - (*average*) *CPI* (cycles per instruction)
    - a floating point intensive application might have a higher average CPI
  - *MIPS* (millions of instructions per second)
    - this would be higher for a program using simple instructions

Performance Measure

- *Performance is determined by execution time*

- Do any of these other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

- *Common pitfall* : thinking one of the variables is indicative of performance when it really isn't

Performance Equation II

CPU execution time for a program = Instruction count for a program × average CPI × Clock cycle time

- *Derive the above equation from Performance Equation I*

CPI Example I

- 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

- *Which machine is faster for this program, and by how much?*
- *If two machines have the same ISA, which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?*

CPI Example II

- 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 1, 2 and 3 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.

- *Which sequence will be faster? How much? What is the CPI for each sequence?*

With Cache Memory

Hit and Miss

- Focus on *any two adjacent* levels – called, *upper* (closer to CPU) and *lower* (farther from CPU) – in the memory hierarchy, because each block copy is always between two adjacent levels
- Terminology:
  - *block*: minimum unit of data to move between levels
  - *hit*: data requested is in upper level
  - *miss*: data requested is not in upper level
  - *hit rate*: fraction of memory accesses that are hits (i.e., found at upper level)
  - *miss rate*: fraction of memory accesses that are not hits
    - miss rate = 1 – hit rate
  - *hit time*: time to determine if the access is indeed a hit + time to access and deliver the data from the upper level to the CPU
  - *miss penalty*: time to determine if the access is a miss + time to replace block at upper level with corresponding block at lower level + time to deliver the data to the CPU

Cache Read Hit/Miss

- *Cache read hit*: no action needed
- *Instruction cache read miss*:
  1. *Send original PC value* (*current PC – I*, as PC has already been incremented in first step of instruction cycle) to memory
  2. Instruct main memory to perform read and wait for memory to complete access – *stall* on read
  3. After read completes *write cache* entry
  4. *Restart* instruction execution at first step to refetch instruction
- *Data cache read miss*:
  - Similar to instruction cache miss
  - To reduce data miss penalty allow processor to execute instructions while waiting for the read to complete *until* the word is required – *stall on use* (why won't this work for instruction misses?)

Cache Write Hit/Miss

- *Write-through* scheme
  - on *write hit*: replace data in cache *and* memory with *every* write hit to avoid *inconsistency*
  - on *write miss*: write the word into cache *and* memory. Bring the corresponding block from memory.
  - Write-through is slow because of always required memory write
    - performance is improved with a *write buffer* where words are stored while waiting to be written to memory – processor can continue execution until write buffer is full
    - when a word in the write buffer completes writing into main that buffer slot is freed and becomes available for future writes
    - DEC 3100 write buffer has 4 words
- *Write-back* scheme
  - on write hit: write the data block *only* into the cache and *write-back* the block to main *only when* it is replaced in cache
  - on write miss: first need to read missed block from memory
  - more efficient than write-through, more complex to implement

Performance

- Simplified model assuming equal read and write miss penalties:
  - CPU time = (execution cycles + memory stall cycles) $\times$ cycle time
  - memory stall cycles = memory accesses $\times$ miss rate $\times$ miss penalty
  - AMAT = Hit time + (Miss rate $\times$ miss penalty)
    - (AMAT = Average Memory Access Time)

- Therefore, two ways to improve performance in cache:
  - decrease miss rate
  - decrease miss penalty
  - *what happens if we increase block size?*

Example Problems

- Assume for a given machine and program:
  - instruction cache miss rate 2%
  - data cache miss rate 4%
  - miss penalty always 40 cycles
  - CPI of 2 without memory stalls
  - frequency of load/stores 36% of instructions

1. *How much faster is a machine with a perfect cache that never misses?*
2. *What happens if we speed up the machine by reducing its CPI to 1 without changing the clock rate?*
3. *What happens if we speed up the machine by doubling its clock rate, but if the absolute time for a miss penalty remains same?*

Solution

1.

- Assume instruction count = I
- Instruction miss cycles = $I \times 2\% \times 40 = 0.8 \times I$
- Data miss cycles = $I \times 36\% \times 4\% \times 40 = 0.576 \times I$
- So, total memory-stall cycles = $0.8 \times I + 0.576 \times I = 1.376 \times I$
  - in other words, 1.376 stall cycles per instruction
- Therefore, CPI with memory stalls = 2 + 1.376 = 3.376
- Assuming instruction count and clock rate remain same for a perfect cache and a cache that misses:

  CPU time with stalls / CPU time with perfect cache

  = 3.376 / 2 = 1.688
- Performance with a perfect cache is better by a factor of 1.688

Solution (cont.)

2.

- CPI without stall = 1
- CPI with stall = 1 + 1.376 = 2.376 (clock has not changed so stall cycles per instruction remains same)
- CPU time with stalls / CPU time with perfect cache

  = CPI with stall / CPI without stall

  = 2.376
- Performance with a perfect cache is better by a factor of 2.376
- Conclusion: with higher CPI cache misses “hurt more” than with lower CPI

Solution (cont.)

3.

- With doubled clock rate, miss penalty = $2 \times 40 = 80$ clock cycles
- Stall cycles per instruction = $(I \times 2\% \times 80) + (I \times 36\% \times 4\% \times 80)$
  $= 2.752 \times I$
- So, faster machine with cache miss has CPI = 2 + 2.752 = 4.752
- CPU time with stalls / CPU time with perfect cache
  = CPI with stall / CPI without stall
  = 4.752 / 2 = 2.376
- Performance with a perfect cache is better by a factor of 2.376
- Conclusion: with higher clock rate cache misses “hurt more” than with lower clock rate

Example Problem

- Assume a 500 MHz machine with
  - base CPI 1.0
  - main memory access time 200 ns.
  - miss rate 5%
- *How much faster will the machine be if we add a second-level cache with 20ns access time that decreases the miss rate to 2%?*

Solution

- Miss penalty to main = 200 ns / (2 ns / clock cycle) = 100 clock cycles
- Effective CPI with one level of cache

= Base CPI + Memory-stall cycles per instruction

= $1.0 + 5\% \times 100 = 6.0$

- With two levels of cache, miss penalty to second-level cache

= 20 ns / (2 ns / clock cycle) = 10 clock cycles

- Effective CPI with two levels of cache

= Base CPI + Primary stalls per instruction

+ Secondary stall per instruction

= $1 + 5\% \times 10 + 2\% \times 100 = 3.5$

- Therefore, machine with secondary cache is faster by a factor of

6.0 / 3.5 = 1.71