The Note of CSE 120

Dongjing Wang

Created date: 03/31/2022

1 Lecture I

1.1 Moore's Law

Moore's Law refers to Gordon Moore's perception that the number of transistors on a microchip doubles every two years, though the cost of computers is halved. Moore's Law states that we can expect the speed and capability of our computers to increase every couple of years, and we will pay less for them. Another tenet of Moore's Law asserts that this growth is exponential.

1.2 Some terms

- **Pipelining**: overlap steps in execution; watch out for dependencies
- **Parallelism**: execute independent tasks in parallel
- **Prediction**: better to ask for forgiveness than permission...
- **Caching**: keep close a copy of frequently used information
- **Indirection**: go through a translation step to allow intervention
- **Amortization**: coarse-grain actions to amortize start/end overheads
- **Redundancy**: extra information or resources to recover from errors
- **Specialization**: trim overheads of general-purpose systems
- **Focus on the common case**: optimize only the critical aspects of the system

2 Lecture II

2.1 Performance

2.1.1 Performance Basics: Latency

- How long it takes to do X?
- Latency: the Base Units

- Clock period: duration of a clock cycle
  E.g., 250ps = 0.25ns = 250×10^{-12}s
  - This is the basic unit of time in all computers

- Clock frequency (rate): cycles per second
  E.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

2.1.2 Performance Basics: Throughput

The rate that a unit of work is completed or started by a mechanism
- How much X work is done in a given time?

2.1.3 Performance: Latency vs Throughput

-Latency improves by speeding up a core?
-Throughput improves by adding more cores?

2.1.4 Execution (or CPU) Time

- Execution time improved by
  -Reducing number of clock cycles
  -Or Increasing clock rate

- These two goals are not always compatible
  -Must often trade off clock rate against cycle count

- Execution Time
  -Execution Time = Cycles *x* Period =“cycles per program” / “clock rate”

2.1.5 Performance Definition

- For a machine A running a program P:
  - Perf(A,P) = 1/(Time(A,P))
- For two computer A and B, on the same program P:
  - IF Perf(A,P) > Perf(B,P) **THEN** Time(A,P) < Time(B,P)

2.1.6 Speedup or "relative performance"

- Speedup = Time(OLD) / Time(NEW)

2.1.7 Relative Performance

- Latency:
  - Time(A,P) = 1 hour, Time(B,P) = 2 hour
  - A is 2 times faster
- Bandwidth/Throughput:
  E.g., A fab can start one new chip per minute and takes 1 month to complete
  - Latency is 1 month
  - Throughput is 1 chip/minute
- More common architecture names:
  - Instructions per cycle
  - Nanoseconds (or picoseconds) cycle time
  - Frequency (inverse of cycle time)

2.2 Different Laws

2.2.1 Bell's Law

- Smaller form factors are initially slower, but more common and cheaper. Eventually they become the dominant form.

2.2.2 Iron Law

- Also known as the "CPU performance equation"
- Execution Time =“Instruction Count” × “Cycles Per Instruction” × “Clock Period”
- [(Implementation) (architecture) (realization)]

2.2.3 Amdahl's Law

- Partial Speedup
  - Performance gain limited by the fraction of improved
  - Speedup = $\frac{\text{CPUTime}_{\text{old}}}{\text{CPUTime}_{\text{new}}} = \frac{\text{CPUTime}_{\text{old}}}{\text{CPUTime}_{\text{old}}\left[(1-f_x)+\frac{f_x}{S_x}\right]} = \frac{1}{(1-f_x)+\frac{f_x}{S_x}}$
- Make Common Case Efficient
  - Lesson's from Amdahl's law
    - Make common cases fast: as $f_x \to 1$, Speedup$\to S_x$
    - But don't over-optimize common case: as $S_x \to \infty$, Speedup$\to$ 1 / (1-f_x)
      - Speedup is limited by the fraction of the code accelerated
      - Uncommon case will eventually become the common one
  - Amdahl's law applies to cost, power consumption, energy...
- Multiple Fractions
  - Do not memorize equation (many uses cases will not have what you need)

2.2.4 little's Law

Parallelism (or occupancy) = Throughput × Latency

- Little's law or flow balance formula
  - The average number of transactions in a stable system is equal to their average arrival rate, multiplied by their average time in the system
- Captures the relation between
  - Latency – how long it takes to do one thing
  - Throughput – the rate which things get done
  - Parallelism – how many things get done at a time

ALU	50%	1
Branches	15%	2
Loads	20%	2
Stores	15%	1

Table 1: An example

- Implications
  - Typically, want to improve one thing while another is constant
  - Example: want to improve throughput but latency is fixed
    - Implies must improve parallelism

2.3 CPI & IPC

2.3.1 Example

In the case of **Table 1**, CPI = 50% × 1 + 15% × 2 + 20% × 2 + 15% × 1 = 1.35, IPC = 1/1.35 ≈ 0.741.

2.3.2 Two Common Usages

- Instruction CPI
  - Each instruction has a fix CPI
  - E.g: load takes 7 cycles, adds 5 cycles
    - CPI_load = 7
    - CPI_add = 5
- Average CPI (most common meaning)
  - Depends on the program or instruction
  - Lower CPI means better performan
    - $\text{CPI} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{IC}} \times \text{CPI}_i$
- IPC (Instructions Per Cycle) is the inverse of CPI