BCN1043

COMPUTER ARCHITECTURE & ORGANIZATION

By
Dr. Mritha Ramalingam

Faculty of Computer Systems & Software Engineering
mritha@ump.edu.my

AUTHORS

- **Dr. Mohd Nizam Mohmad Kahar** (mnizam@ump.edu.my)
- **Jamaludin Sallim** (jamal@ump.edu.my)
- **Dr. Syafiq Fauzi Kamarulzaman** (syafiq29@ump.edu.my)
- **Dr. Mritha Ramalingam** (mritha@ump.edu.my)

Faculty of Computer Systems & Software Engineering

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$CPUTime_A = InstructionCount \times CPI_A \times CycleTime_A$$
$$= I \times 2.0 \times 250ps = I \times 500ps$$

A is faster…

$$CPUTime_B = InstructionCount \times CPI_B \times CycleTime_B$$
$$= I \times 1.2 \times 500ps = I \times 600ps$$

$$\frac{CPUTime_B}{CPUTime_A} = \frac{I \times 600ps}{I \times 500ps} = 1.2$$

…by this much

CPI in More Detail

- If different instruction types take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

  - Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

CPI Example

- Alternative compiled code sequences using instructions in type INT, FP, MEM

Type	INT	FP	MEM
CPI for type	1	2	3
IC in Program 1	2	1	2
IC in Program 2	4	1	1

- Program 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Program 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

Performance Summary

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Power Trends

- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

Pitfall: Amdahl's Law

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

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

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

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

  - Can't be done!
- Corollary: make the common case fast

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- CPI varies between programs on a given CPU

Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance

Amdahl’s law

- Amdahl's Law implies that the overall performance improvement is limited by the range of impact when an optimization is applied. Consider the following equations:
- Improvement rate N = (original execution time) / (new execution time)
- New execution time = timeunaffected + timeaffected / (Improvement Factor)
- Time = (instruction count) × CPI × CCT

- CPI = (execution_time x clock_rate)/instructions
- (cpu_ex_time_B/cpu_ex_time_A) – formula to know which is faster

Amdahl's law

- If a program currently takes 100 seconds to execute and loads and stores account for 20% of the execution times, how long will the program take if loads and stores are made 30% faster? For this, you can use Amdahl's law or you can reason it out step by step. Doing it step by step gives (1) Before the improvement loads take 20 seconds (2) If loads and stores are made 30 percent faster they will take 20/1.3 = 15.385 seconds, which corresponds to 4.615 seconds less. (3) Thus, the final program will take 100 - 4.615 = 95.38

Amdahl’s law

- **Amdahl's law**, named after computer architect Gene Amdahl, is used to find the maximum expected improvement to an overall system when only part of the system is improved.
- Amdhal's law can be interpreted more technically, but in simplest terms it means that it is the algorithm that decides the speedup not the number of processors.

Wikipedia

Amdahl's law

- Amdahl's law is a demonstration of the law of diminishing returns: while one could speed up part of a computer a hundred-fold or more, if the improvement only affects 12% of the overall task, the best the speedup could possibly be is

$$\frac{1}{1 - 0.12} = 1.136 \text{ times faster.}$$

Amdahl's law

- More technically, the law is concerned with the speedup achievable from an improvement to a computation that affects a proportion *P* of that computation where the improvement has a speedup of *S*.

Amdahl's law

- For example, if an improvement can speedup 30% of the computation, *P* will be 0.3; if the improvement makes the portion affected twice as fast, *S* will be 2.
- Amdahl's law states that the overall speedup of applying the improvement will be

$$\frac{1}{(1-P)+\frac{P}{S}}$$

Amdahl's law

- To see how this formula was derived, assume that the running time of the old computation was 1, for some unit of time. The running time of the new computation will be the length of time the unimproved fraction takes (which is

  1 – *P*) plus the length of time the improved fraction takes. The length of time for the improved part of the computation is the length of the improved part's former running time divided by the speedup, making the length of time of the improved part *P*/*S*.

Amdahl's law

$$\frac{1}{(1-P)+\frac{P}{S}}$$

- The final speedup is computed by dividing the old running time by the new running time, which is what the above formula does.

Wikipedia

Amdahl's law

Here's another example. We are given a task which is split up into four parts: P1 = .11 or 11%, P2 = .18 or 18%, P3 = .23 or 23%, P4 = .48 or 48%, which add up to 100%. Then we say P1 is not sped up, so S1 = 1 or 100%, P2 is sped up 5x, so S2 = 5 or 500%, P3 is sped up 20x, so S3 = 20 or 2000%, and P4 is sped up 1.6x, so S4 = 1.6 or 160%.

Wikipedia

Amdahl's law

By using the formula $\frac{P1}{S1} + \frac{P2}{S2} + \frac{P3}{S3} + \frac{P4}{S4}$

we find the running time is $\frac{.11}{1} + \frac{.18}{5} + \frac{.23}{20} + \frac{.48}{1.6} = .4575$

or a little less than 1/2 the original running time which we know is 1.

Therefore the overall speed boost is $\frac{1}{.4575} = 2.186$

or a little more than double the original speed

using the formula $\frac{1}{\frac{P1}{S1} + \frac{P2}{S2} + \frac{P3}{S3} + \frac{P4}{S4}}$

- Reference: **William Stallings. 2015. Computer Organization and Architecture – Designing for Performance, 10th Edition, Prentice Hall**

- Chapter ends!